

PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA
MINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH
Ecole Nationale Polytechnique



المدرسة الوطنية المتعددة التقنيات
Ecole Nationale Polytechnique

Electronic Departement
In partial fulfillment of the requirement for
Engineer's Degree

Integration of a Geographic Information System in a Smart City Solution

BENLEFKI Yasmine
BOUCENNA Manal Maroua

Presented in public on: 21st June 2018

Jury members

President	M. Adel BELOUCHARANI	Prof.	ENP
Supervisor	M. Rabah SADOON	Dr.	ENP
Supervisor	M. Riad HARTANI	Dr.	Padovani Ventures
Examiner	M. Mourad ADNANE	Dr.	ENP

ENP 2018

10, Frères Oudek Avenue, Hassen Badi; Bp, 182, 16200 El Harrach, Algiers, Algeria

PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA
MINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH
Ecole Nationale Polytechnique



المدرسة الوطنية المتعددة التقنيات
Ecole Nationale Polytechnique

Electronic Departement
In partial fulfillment of the requirement for
Engineer's Degree

Integration of a Geographic Information System in a Smart City Solution

BENLEFKI Yasmine
BOUCENNA Manal Maroua

Presented in public on: 21st June 2018

Jury members

President	M. Adel BELOUHRANI	Prof.	ENP
Supervisor	M. Rabah SADOON	Dr.	ENP
Supervisor	M. Riad HARTANI	Dr.	Padovani Ventures
Examiner	M. Mourad ADNANE	Dr.	ENP

ENP 2018

10, Frères Oudek Avenue, Hassen Badi; Bp, 182, 16200 El Harrach, Algiers, Algeria

Dedication

This humble work is dedicated to our dear parents and grand parents,
to our families,
to our brothers and sisters,
to our friends and everyone who trusted and supported us.

Acknowledgements

We thank God Almighty for giving us courage, health, and His support during the most difficult times.

We would like to express our sincere gratitude to our advisor Mr. **Rabah SADOUN** for his patience, motivation and understanding over these past three years. His guidance and relevant questions helped us in all the time of research and writing of this project graduation. We could not have imagined having a better advisor and mentor.

We also thank Mr. **Riad HARTANI** for giving us the chance to work on a project of such a great importance for our country which is the "**Algiers Smart City**" project.

We are also grateful to all the **CDTA**'s team, and more particularly, Mr. **Amine Bouabdellah** for his availability, his valuable advice and encouragement; Mrs. **AKAK Leila** and Mrs. **TITOUCHE Yousra** for their patience and availability.

Our heartfelt thanks go to the members of the jury, Mr. **Adel BELOUHRANI** and Mr. **Mourad ADNANE** for generously offering their time, support, guidance and good will throughout the examination and review of this document.

Thanks to Mr.**BENLEFKI Yazid** and Mrs.**BOUCHAMA Samira** who constantly checked for our errors and omissions and guided us through this work.

Special thanks to our parents for having relentlessly encouraged us to explore any and every subject that interested us.

Last but not the least, we would like to thank our families and friends for constantly supporting us. None of this could have happened without them.

ملخص

أصبح نظام المعلومات الجغرافية أداة لا غنى عنها في إدارة وتحليل وصنع القرار من خلال الجمع السلس بين البيانات المكانية و الوصفية. ستساعد الرؤية الثرية لنظم المعلومات الجغرافية على بناء مدن الغد الذكية المستدامة من خلال زيادة الفعالية وتوفير قرارات أفضل. توضح هذه الوثيقة بالتفصيل التصميم والتنفيذ في بيئة إنتاجية لنظم المعلومات الجغرافية الموجهة إلى إنترنت الأشياء انطلاقاً من مجموعة من البرمجيات مفتوحة المصدر. بعد ذلك تقدم نتائج اختبار الأداء باستعمال ثم تقدم بوابة جغرافية كمثال لإحدى التطبيقات.

الكلمات المفتاحية: نظام المعلومات الجغرافية ، إنترنت الأشياء ، البيانات الكبيرة ، المدينة الذكية.

Résumé

De nos jours, les Systèmes d'Information Géographiques (SIG) sont devenus des outils indispensables de gestion, d'analyse et d'aide à la décision en combinant parfaitement les données spatiales et non-spatiales. Une bonne maîtrise du SIG, et plus particulièrement d'un SIG orienté IoT, peut fortement contribuer au développement des villes intelligentes de demain en leur permettant de prendre des décisions plus éclairées pour la qualité de vie des citoyens, tout en améliorant leur efficacité.

Dans ce présent document, nous allons présenter la conception et l'implémentation, dans un environnement de production, d'un SIG orienté IoT en utilisant des solutions Open Source. Ensuite, nous effectuerons des tests de performances sur notre architecture et enfin nous présenterons l'une des applications cliente possible d'un SIG.

Mots clés: Systèmes d'Information Géographiques, Internet des objets, Ville Intelligente.

Abstract

Geographical Information System (GIS) has become an indispensable tool for managing, analyzing and decision making by seamlessly combining both spatial and non-spatial data. Semantically enriched vision of GIS, particularly IoT-oriented GIS, will help evolve cities into tomorrow's smart cities by providing better decisions, more efficiency and improved collaboration.

This document presents the design and implementation, in a production environment, of an IoT-oriented Web GIS built from a combination of open source software. After that it presents the performance test results and introduces a client Web GIS application.

Key words: Geographical Information System, Internet of Things, Big Data, Smart-City.

Contents

List of Figures

List of Tables

1	General introduction	13
1.1	Motivation and Objectives	13
1.2	Document outline	14
2	IoT-oriented GIS Smart City platform	15
2.1	Introduction	15
2.2	Smart City	15
2.2.1	Definition	15
2.2.2	Smart City applications and benefits	16
2.3	IoT platforms	17
2.3.1	Definitions	17
2.3.2	Types of IoT platforms	18
2.3.3	Components of an IoT platform	18
2.4	Big Data platforms	19
2.4.1	Definitions	19
2.4.2	Types of Big Data platforms	20
2.4.3	Components of a Big Data platform	21
2.5	Geographical Information System platforms	21
2.5.1	Definitions	22
2.5.1.1	Information System	22
2.5.1.2	Geography Information System	22
2.5.2	Components of a GIS	22
2.5.3	GIS for Smart City	23
2.6	Generic architecture for an IoT-oriented GIS Smart City platform:	23
2.7	Conclusion	25
3	Geographic information system (GIS)	26
3.1	Introduction	26
3.2	GIS History	26
3.3	Theoretical part:	27
3.3.1	Geodetic System	27
3.3.1.1	Types of geodesic systems	28
3.3.2	Projections	29
3.3.2.1	EPSG	29

3.3.2.2	Projection used in Algeria	30
3.3.3	Types of maps	30
3.4	Type of GIS applications	31
3.4.1	Standalone GIS	31
3.4.2	Client/Server GIS	31
3.4.2.1	Client Side	31
3.4.2.2	Web Map Server	32
3.5	Data store	33
3.5.1	Vector data	34
3.5.2	Raster data	34
3.5.3	Spatial database	35
3.6	Free and open source GIS software	37
3.7	Conclusion	37
4	Technical choices	38
4.1	Introduction	38
4.2	Cloud platform	38
4.3	IoT platform	39
4.3.1	Definition	39
4.3.2	Architecture	40
4.3.3	Sentilo resources	41
4.3.4	Sentilo's HTTP REST API	42
4.4	Spatial Database Management System	44
4.4.1	Sentilo agents comparison	44
4.4.2	PostgreSQL/PostGis vs. Geomesa	44
4.4.2.1	PostgreSQL/PostGis	44
4.4.2.2	Geomesa	45
4.4.3	Greenplum	46
4.4.3.1	Definition	46
4.4.3.2	Greenplum architecture	46
4.5	Web mapping server	47
4.5.1	GeoServer Vs. MapServer	48
4.5.2	Geoserver	51
4.5.2.1	Architecture	51
4.5.2.2	Geoserver concepts	52
4.5.2.3	GeoServer REST Interface	53
4.5.2.4	Web services	53
4.5.2.5	GeoWebCache	54
4.6	End-to-end architecture	55
4.7	Conclusion	55
5	Implementation	57
5.1	Introduction	57
5.2	Detailed architecture to implement	57
5.3	Sentilo	59
5.3.1	Data	59
5.3.1.1	OpenData	59
5.3.1.2	Simulated Data	62

5.3.2	Sentilo relational agent	63
5.4	Greenplum database	64
5.4.1	Greenplum and Sentilo interconnection	65
5.5	Geoserver	68
5.5.1	Publish the data	68
5.5.1.1	Vector digital map background	68
5.5.1.2	Raster data	72
5.5.1.3	Greenplum sensor's data	74
5.5.2	Geoserver in Production	76
5.5.3	Clustering	76
5.5.3.1	Definitions	76
5.5.3.2	Implementation	77
5.5.3.3	Security	82
5.5.3.4	Watchdog	85
5.5.3.5	GeoWebCache	85
5.6	Conclusion	86
6	Performance testing and web-mapping application	87
6.1	Introduction	87
6.2	Performance testing	87
6.2.1	Apache Jmeter	87
6.2.2	Scenario of stress testing	87
6.2.3	Results	88
6.3	Application	92
6.3.1	Web mapping	92
6.3.1.1	OpenLayers	93
6.3.1.2	ReactJS	93
6.3.1.3	Boundless	93
6.3.2	Web Application's description	94
6.4	Conclusion	95
7	Conclusion and future work	96
7.1	Summary	96
7.2	Future work	96

Bibliography	98
---------------------	-----------

Annexes

Annex A: Software installation	102	
1	Installation of Sentilo in Ubuntu 14.04	102
1.1	Installing dependencies	102
1.2	Download and build code	102
1.3	Redis settings	102
1.4	MongoDB settings	102
1.5	Configure Tomcat7	103
1.6	Start services	103

2	GeoServer installation and configuration	103
2.1	Installing Java	103
2.2	Installation of Tomcat	104
2.2.1	Create Tomcat User	104
2.2.2	Install Tomcat	105
2.2.3	Set up the proper user permissions	105
2.2.4	Create a systemd Service File	105
2.2.5	Adjust the Firewall and Test the Tomcat Server	106
2.2.6	Configure Tomcat Web Management Interface	107
2.2.7	Access the Web Interface	108
2.3	Installing GeoServer	109
2.3.1	Create a geoserver instance	109
2.3.2	Implement Basic Security	110
2.3.3	Configuring multiple instance in a single server	110
2.4	Install Load Balancer	112
3	Installation of PostgreSQL	112
4	Installation of Postgis	114
5	Installation of Greenplum Pivotal 5.7 onCentOS 64-bit 7.x	114
5.1	Setting the Greenplum Recommended OS Parameters (Master Only)	114
5.1.1	Linux System Settings	114
5.1.2	Creating the Greenplum Database Administrative User Account	115
5.1.3	Installing the Greenplum Database Software	115
5.2	Installing and Configuring Greenplum on all Hosts	116
5.3	Installing Greenplum Database Extensions	117
Annex B: GeoServer Administration Interface		118
1	About Status	118
1.1	Data	119
2	Services	120
3	Settings	120
4	Tile Caching	120
5	Security	121
6	Demos	121
Annex C: Scripts		123
1	Watchdog Script	123
2	Python scripts:	125
2.1	Open data sensors	125
2.2	Simulated sensors	125

List of Figures

2.1	Smart City	15
2.2	Smart city action fields	16
2.3	IoT ecosystem	17
2.4	IoT Platform	18
2.5	The 3 V of Big Data	20
2.6	MPP Architecture	20
2.7	Component of a Big Data platform	21
2.8	GIS components	23
2.9	Smart city's generic architecture for GIS case study	25
3.1	Dr. John Snow	26
3.2	Earth's shape	28
3.3	Local and Geocentric coordinate systems	29
3.4	UTM projection	30
3.5	Standalone Vs. Client/Server Architecture	31
3.6	Web Map Service	32
3.7	Web Feature Service	33
3.8	Vector's data	34
3.9	Raster data	35
3.10	Database	35
3.11	Bounding Boxes	36
3.12	The free and open source geographic information software map of 2012.[1]	37
4.1	OpenStack Architecture	39
4.2	Sentilo's user interface	39
4.4	Sentilo Architecture	41
4.6	PostgreSQL and PostGIS Database	45
4.7	Performance comparison[2]	46
4.13	Geoserver Architecture	52
4.14	Relationships between workspaces, stores and layers	53
4.15	GeoWebCache	55
4.16	Logical end-to-end GIS architecture	56
5.1	Detailed architecture to implement	58
5.2	Temperature table's format	59
5.3	A caption of the temperture001.json file	61
5.4	Temperature Sensors	61
5.5	Visualization of temperature data	62
5.6	Visualization of the velib data	63

5.7	Universal Viewer	63
5.8	Greenplum database architecture	65
5.9	Created table	66
5.10	Data transfer from Sentilo to Greenplum	66
5.11	"Sentilo_observations" table's meta data	67
5.12	"Sentilo_observations" geometry column	68
5.13	OpenStreetMap tables	69
5.14	Layer-like OpenStreetMap tables	70
5.15	Creation of a new workspace	70
5.16	Creation of a new store	71
5.17	Creation of aero-poly layer	71
5.18	Creation of OsmMap layer group	72
5.19	Preview of OpenStreetMap data	72
5.20	Adding a new store	73
5.21	Satellite imagery	73
5.22	Preview of Greenplum data	74
5.23	Greenplum table preview	74
5.24	Greenplum Data workflow	75
5.25	Sorted data	76
5.26	High availability	76
5.27	Cluster architecture	77
5.28	Scale up vs scale out	77
5.29	Geoserver cluster architecture	78
5.30	GeoServer cluster architecture with shared data directory	79
5.31	Cluster configuration interface	80
5.32	Clustering configuration settings	80
5.33	Final architecture of GeoServer cluster	81
5.34	Authentication and authorization	82
5.35	Security work flow of Geoserver	82
5.36	Strong Password Encryption	83
5.37	Creating groups	83
5.38	Creating users	84
5.39	Creating roles	84
5.40	Linking Groups with roles	84
5.41	Granting access rights	85
5.42	Watchdog's state machine	85
6.1	Apache Jmeter work flow	88
6.2	Test1 before clustering	89
6.3	Test1 after clustering	89
6.4	Test 2 before clustering	90
6.5	Test 2 after clustering	90
6.6	Test 3 before clustering	91
6.7	Test 3 after clustering	91
6.8	Average response time per of users	92
6.9	Geoportal Web Page	94
6.10	Geoportal's table	94
6.11	Area measurement	94

6.12	New Layer	95
6.13	Geoportal's tools	95
7.4	Apache2 Ubuntu Default Page	113

List of Tables

2.1	GIS applications for smart cities	24
3.1	GIS Development's stages	27
3.2	Ellipsoids in global geodetic systems	28
3.3	local and global Datum	29
3.4	Main types of maps	30
4.1	REST API Methods	42
4.2	Publish Data	43
4.3	PostGIS functions	45
4.4	Characteristics and limits of Greenplum (source: community.pivotal.io)	47
4.5	Comparison of different Geoportals	48
4.6	MapServer Vs. GeoServer	50
5.1	HTTP responses	83
6.1	Stress testing scenario	88
6.2	Boundless SDK functions	93

Chapter 1

General introduction

1.1 Motivation and Objectives

By 2050, 66% of the world's population is expected to live in urban areas [3]. The challenge will be to supply this population with basic resources while also ensuring overall economic, social and environmental sustainability. Without a doubt, this evolution raises significant changes with regard to the deployment and management of all types of infrastructures within cities.

Therefore, city operators must place technology at the core of their development strategy in the upcoming years. This has been translated as the "Smart City" concept.

After Montreal, Cairo, San Diego and Barcelona, the Wilaya of Algiers launched the project "Algiers Smart City" and called for better and efficient collaboration of Startups, Research Centers, Universities, Solution providers, etc. The overarching aim of this project is to improve the quality of living for its citizens by enhancing the efficiency and the performance of its urban services such as energy, transportation and utilities in order to reduce resource consumption, wastage and overall costs.

In the context of our graduation project, we took part in a collaborative work with our colleagues in the electronics/industry department and the computer science school since June 2017, supervised by our tutor Dr. Rabah Sadoun, the CDTA team, and Dr. Riad Hartani.

Each of the projects aimed at contributing in its own way to the implementation of the Algiers Smart City platform. Our work consisted of the design and development of an IoT oriented Geographical Information System (GIS) using Open Source solutions.

GIS is actively used to provide solutions in numerous branches of government services as well as in businesses and industry. Using GIS to visualize and analyze data can enable quicker interpretation of complex geographical phenomena, identify patterns, and aid in planning, resource allocations for policy and decision making in smart cities. However, up to date, The Wilaya of Algiers does not have any IoT oriented GIS. Thus our project will serve as a reference to the Algiers Smart city project.

1.2 Document outline

The present document has seven chapters:

- **Chapter 2** (IoT-oriented GIS Smart City platform) Start gives by giving an overview of the main Smart city key elements. Then it presents and discusses keys elements of our work: IoT, Big Data and GIS platform, and end up with the generic architecture of an IoT-oriented GIS platform we propose.
- **Chapter 3** (Geographic Information System (GIS)) commences with a brief presentation of GIS history, followed with a theoretical part on the geographic concepts. It, then, goes through the different types of GIS architectures and GIS datastores. The chapter concludes with a list of free and Open Source GIS software.
- **Chapter 4** (Technical choices) A comparison of different open source software to choose the most suitable ones for our architecture, it ended up with a functional of an IoT-oriented GIS platform architecture.
- **Chapter 5** (Implementation) discusses the technical details as well as the development guidelines. It illustrates the steps required for the implementation of the IoT-oriented GIS stack, as well as the potential difficulties.
- **Chapter 6** (Performance tests and web mapping application) contains the detailed results of the Performance test conducted on the implemented architecture using Apache Jmeter. It presents then the web-based application of a Geoportal used to illustrate some of the possible features of our architecture.
- **Chapter 7** (Conclusion and future work) summarizes our work and discusses the difficulties we met. It is concluded with a future outlook.
- **Annex** contains the installation and configuration steps for the used software and some scripts.

Chapter 2

IoT-oriented GIS Smart City platform

2.1 Introduction

In this chapter, we will introduce the key concepts for our work: Smart City, IoT, Big Data and Geographical Information System (GIS) platforms. To conclude, we will present a smart city generic architecture in the use case of an IoT oriented GIS.

2.2 Smart City

2.2.1 Definition

In general terms, most agree that a smart city is one that places technology at the core of its development strategy. It is predominantly composed of Information and Communication Technologies that "senses" the environment, using IoT sensors and video cameras, so that the city operators can decide how and when to take action.

Also, according to Dr. Riad Hartani, the supervisor of the "Algiers Smart City" project, a smart city is a collaborative city. A city in which citizens, business firms, knowledge institutions, and municipal agencies collaborate with one another to achieve their goals.



Figure 2.1: Smart City

Source: ookawa-corp.over-blog.com

2.2.2 Smart City applications and benefits

An ideal smart city strategy covers six interrelated action fields, comprising a host of sub-categories and solutions as it is shown in the figure 2.2:



Figure 2.2: Smart city action fields

Source: Think:Act Magazine—Roland Berger

Smart city strategy can bring multiple benefits for municipalities, particularly those that are struggling with specific problems such as inefficient waste management systems, lack of civic participation, or traffic and congestion:

1. Cost Savings:

Increasing use of automation, AI, data-sharing, analytics, and sensors by both businesses and authorities is massively improving the efficiency of public and commercial operations, ultimately leading to huge cost savings.

2. Environmental Impact:

Installing sensors and cameras in order to measure air quality, sound levels, temperature, water levels will act as a tracker for the city, helping city operators to identify ways to save energy, use their resources more effectively and reduce waste production.

3. Economic Prosperity:

Smarter cities attract more businesses, thereby boosting their economies and reducing unemployment.

4. Life Quality:

More efficient transport services, smart parking apps, digitized government services often have a direct positive impact on citizens' lives because they are created specifically with people's needs and experiences in mind.

2.3 IoT platforms

IoT devices and platforms are becoming part of a smart city infrastructure that can combat the strain of city growth, from traffic control to environmental issues.

2.3.1 Definitions

1. Internet of Things (IoT) is an ecosystem of connected physical objects that are accessible through the internet. The 'thing' in IoT is: any object that has an assigned IP address and has the ability to collect and transfer data over a network without manual assistance or intervention. The embedded technology in the objects helps them to interact with internal states or the external environment, which in turn affects the decisions taken[4].

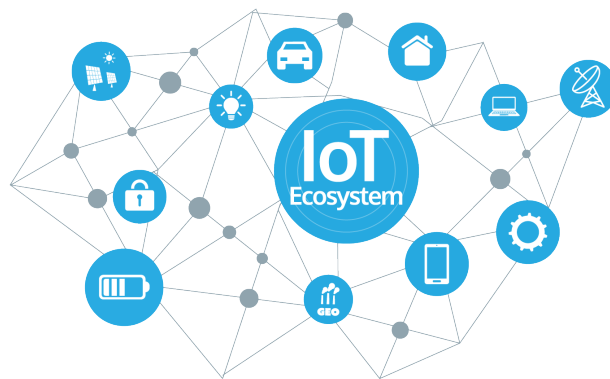


Figure 2.3: IoT ecosystem

Source: www.sensorexpo.com

2. At a high level, an IoT platform is the support software that connects edge hardware, access points, and data networks to other parts of the value chain (which are generally the end-user applications). IoT platforms typically handle ongoing management tasks and data visualization, which allow users to automate their environment[5].

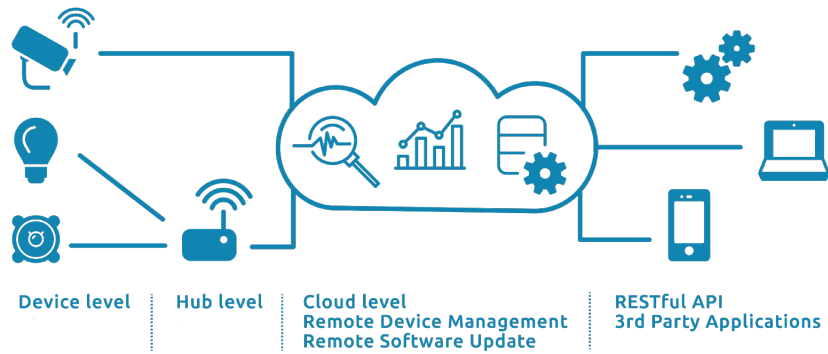


Figure 2.4: IoT Platform

Source: <https://dzone.com>

2.3.2 Types of IoT platforms

There are 4 types of platforms that are often referred to as “IoT Platform”[6] [7]:

1. Connectivity Management Platforms:

These platforms focus mainly on the connectivity of connected IoT devices via telecommunication networks but rarely on the processing and enrichment of different sets of sensor data.

2. IoT Cloud Platforms:

Cloud platforms aim to get rid of the complexity of building complex network stack from scratch and offer the back-end services to monitor and track millions of simultaneous device connections.

3. End-to-end IoT platforms

These platforms provide the hardware, software, connectivity, security, and device management tools to handle millions of concurrent device connections.

4. Data Platform:

These IoT data platforms combine many of the tools you need to route device data and manage/visualize data analytics.

2.3.3 Components of an IoT platform

In its simplest form, an IoT platform is just about enabling connectivity between devices. The architecture may also consist of a software platform, an application development platform or an analytic platform. A modern IoT Platform architecture comprises 6 important components:

1. **Connectivity:** brings different protocols and different data formats into one “software” interface ensuring accurate data streaming and interaction with all devices.
2. **Device management:** ensures the connected devices are working properly.

3. **Database:** scalable storage of device data brings the requirements for cloud-based databases to a new level in terms of data volume, variety, velocity and veracity.
4. **Processing action management:** brings data to life with rule-based event-action-triggers enabling execution of “smart” actions based on specific sensor data.
5. **Analytics:** performs a range of complex analysis from basic data clustering and deep machine learning to predictive analytics extracting the most value out of the IoT data-stream.
6. **Visualization:** enables humans to see patterns and observe trends from visualization dashboards.

2.4 Big Data platforms

Smart cities create voluminous amount of heterogeneous data, which is commonly referred to as big data. Big data can be mined and modelled through analytic techniques to get better insight and to enhance smart cities features.

2.4.1 Definitions

1. **Big Data** is any voluminous amount of Structured, Semi-structured and Unstructured data that can be mined for information. The three main characteristics of big data are:
 - (a) **Volume:**

The size of data should be larger than terabytes and petabytes. It implies enormous volumes of data generated by Sensors, social media, e-commerce, GPS devices etc...
 - (b) **Velocity:**

It represents the rate at which data is pouring in.
 - (c) **Variety:**

It implies to the type of formats and they can be classified into three types:

 - **Structured:** It inserts already tagged and easily sorted data, uses RDBMS (Relational Database Management System) like Oracle, MySQL, etc...
 - **Semi-Structured:** It does not conform to fixed fields but contains tags to separate data elements. Eg: emails, tweets.
 - **Un-Structured:** It is random and difficult to analyze. Eg: photos, videos, audio files.

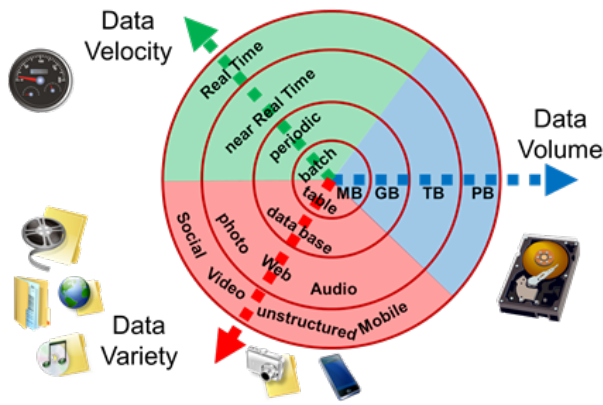


Figure 2.5: The 3 V of Big Data

2. **Big data platform** generally consists of big data storage, servers, database, big data management, business intelligence¹ and other big data management utilities.

2.4.2 Types of Big Data platforms

There are three primary architectures used to handle ‘Big Data’[8]:

1. Massively Parallel Processing (MPP) platform

The Massively Parallel Processing (MPP) platforms manage large volume of structured data by distributing them across many machines, or nodes. Each node contains its own storage and compute capabilities that enables it to execute a portion of the query[9]. MPP systems can grow horizontally to increase performance or capacity by simply adding more processors to the architecture.

MPP Databases are optimized for aggregating and processing large datasets much more quickly and efficiently by using column oriented storage².

MPP Shared Nothing Architecture

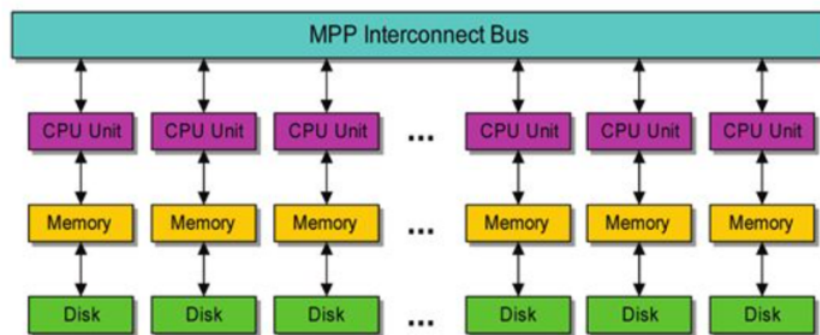


Figure 2.6: MPP Architecture

¹Business intelligence is a technology-driven process for analyzing data and presenting actionable information to help executives, managers and other corporate end users make informed business decisions.

²A column-oriented database stores each column continuously. They are extremely quick at aggregate queries (sum, average, min, max, etc.).

2. NoSQL platforms

NoSQL means “Not Only SQL.” These architectures can provide high performance at a lower cost, with linear scalability, the ability to use commodity hardware and to handle a number of different data types. NoSQL solutions perform better in conditions where there are extremely high data volumes or high content of unstructured data such as documents and media files.

Today the most popular NoSQL platform is Hadoop. Hadoop provides an end-to-end architecture for large volumes of data, including a distributed file system (HDFS), a distributed processing manager (MapReduce) and different databases and various data flow options including Sqoop, Hive, HBase, and Pig.

2.4.3 Components of a Big Data platform

Big data architecture includes mechanisms for ingesting, protecting, processing, and transforming data into filesystems or database structures. Big data architecture has four logical layers:

1. **Big data sources layer:** Data can come from different sources: company servers, sensors, etc.
2. **Data storage layer**
This layer receives data from the sources, converts unstructured data to a format that analytic tools can understand and stores it.
3. **Analysis layer**
The analytics layer interacts with stored data to extract business intelligence.
4. **Consumption layer:**
This layer receives analysis results and presents them to the appropriate output layer.

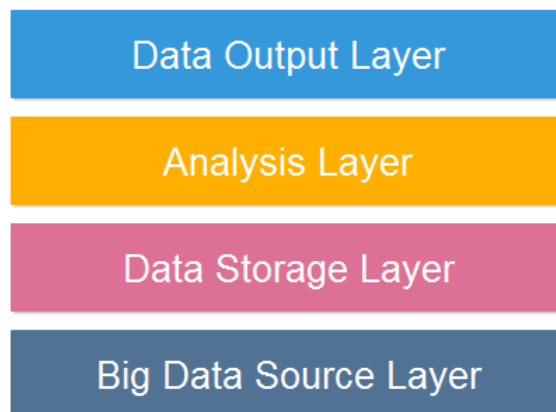


Figure 2.7: Component of a Big Data platform

2.5 Geographical Information System platforms

Designing or managing a smart city demands enormous spatially referenced data. GIS handles spatial data and includes operation which supports spatial analysis.

2.5.1 Definitions

Before explaining what is a GIS, let's first define what is an Information System (IS).

2.5.1.1 Information System

An IS is a combination of hardware, software, infrastructure and trained personnel organized to facilitate planning, control, coordination, and decision making in an organization[10].

2.5.1.2 Geography Information System

GIS is a dedicated IS designed to capture, store, manipulate, analyze, manage, and present all types of geographical data. The key word to this technology is Geography which means that some portion of the data is **spatial**, it referenced to locations on the earth.

Coupled with this data is usually tabular data known as **attribute data** that can be generally defined as additional information about each of the spatial features. It is the partnership of these two data types that enables GIS to be such an effective problem solving tool through spatial analysis.

Example:

An example of this would be schools. The location of the schools is the spatial data. Additional data such as the school name, level of education taught, student capacity would make up the attribute data.

2.5.2 Components of a GIS

An operational GIS has a series of components that combine to make a successful GIS. We can distinguish four components:

1. **Spatial Data:** Geospatial data has a significantly different structure and function of classical data. It includes structured data about objects in the spatial universe – their identity, location, shape and orientation, etc.
2. **Spatial Data Management** Spatial database management deals with the storage, indexing, and querying spatial data, such as location and geometric extent.
3. **GIS applications:** are tools that allow users to manipulate and analyze spatial information.
4. **User interface:** This component addresses the data visualization function.

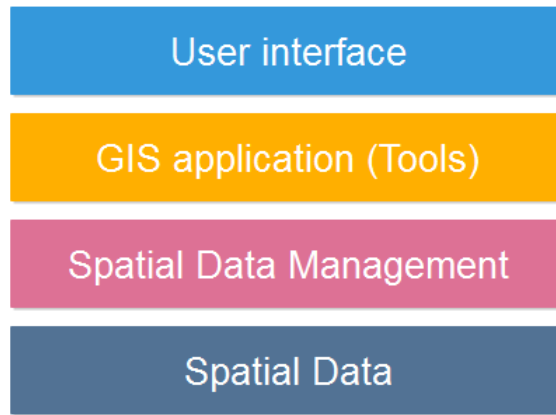


Figure 2.8: GIS components

2.5.3 GIS for Smart City

Designing or managing a smart city demands enormous spatially referenced data.[11] GIS has the capacity to incorporate millions of variables, geocode³ them in order to exploit them for planning, management and development of a city.

The major GIS application for a smart city can be grouped into five categories as detailed in the table 2.1 [12]

2.6 Generic architecture for an IoT-oriented GIS Smart City platform:

Although the literature contains several works about Smart city architectures, we barely found an architecture corresponding to IoT dedicated GIS. The generic architecture that we propose is composed of 3 main layers:

1. IoT devices Layer:

It represents all the hardware devices found in a city that enables the capture and integration of live real-world data . It also includes communication technologies used to connect all the devices.

2. IoT platform layer:

This layer acts like a middleware between the hardware and the big data storage. It provides Visualization, security and device management tools to handle millions of concurrent device connections.

3. Big data platform layer:

In this layer voluminous amount of sensor data is stored, managed and modelled through analytic techniques.

³Geocoding is the process of transforming a description of an address, or a name of a place to a location on the earth's surface.

Table 2.1: GIS applications for smart cities

Area	GIS applications
Facilities Management	Locating underground pipes & cables Planning facility maintenance Energy use tracking & planning
Environment and Natural Resources Management	Comprehensive waste management Environmental impact analysis Disaster management and mitigation
Street Network	Accidents, Anomaly Prediction Smart parking Smart transportation grid
Planning and Engineering	Urban planning Smart energy grid/renewable energy framework Development of public facilities
Land Information System	Cadastre administration Landuse Classification

4. Geographic Information System platform layer:

This layer is a key element for a smart city since it exploits geolocalized information to detect real needs and optimize the resources. It can go from spatial servers and databases to end-client applications like geoportals. It enables to visualize data, share information, apply analytics and predictions on data based on their spatial position using artificial intelligence methods or what is known as Location Intelligence to solve complex problems.

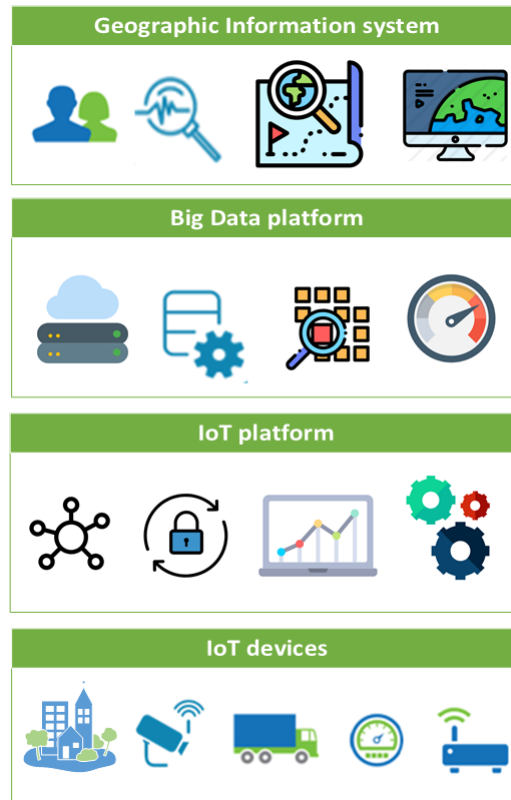


Figure 2.9: Smart city’s generic architecture for GIS case study

2.7 Conclusion

In this chapter, we presented the key concepts of our work: Smart city, IoT platform, Big data and GIS. Then we exposed a generic Smart city’s architecture for a GIS case study. In the next sections, we will introduce GIS most important notions and then explain our technical choices according to the generic Smart city architecture given above.

Chapter 3

Geographic information system (GIS)

3.1 Introduction

Geography, the science of our world, coupled with information system is helping us understand the Earth and apply geographic knowledge to a host of human activities.

We will introduce in this chapter the foundation of geographical information system and spatial data by describing, on a general level, some theoretical concepts, technologies and standards of GIS as described in the literature.

3.2 GIS History

The history of GIS all started in 1854 when Cholera hit the city of London, England. British physician John Snow began mapping outbreak locations, roads, property boundaries and water lines [13].

When he added these features to a map, he noticed that Cholera cases were commonly found **along the water line**.

His work connected geography and public health safety and demonstrated that GIS is a relevant problem-solving tool.

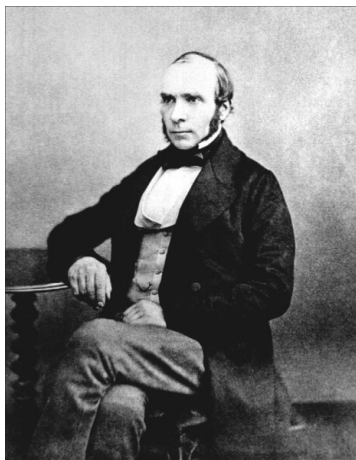


Figure 3.1: Dr. John Snow

Source: en.wikipedia.org

Stages of GIS development:

We can group the history of GIS into several stages of development [14]:

Table 3.1: GIS Development's stages

Stage	Description
Before 1960: The GIS Dark Ages	-All mapping was done on paper.
1960 to 1975: GIS Pioneering	-Roger Tomlinson (GIS's father) works to initiate the development of the Canadian Geographic Information System, resulted in the first computerized GIS in the world in 1963. - At Northwestern University in 1964, Howard Fisher created the first computer mapping software programs known as SYMAP.
1975 to 1990: GIS Software Commercialization	-In the late 1970s, memory size and graphics capabilities were improving. -New computer cartography products included MAPICS, SURFACE, -In the late 1980s, GIS software vendors increased considerably.
1990 to 2000's: User Proliferation	-The importance of spatial analysis for decision-making is recognized. - Software was able to handle both vector and raster data.

3.3 Theoretical part:

This section introduces the theoretical part by highlighting concepts of Earth geometry, coordinate systems and projections.

3.3.1 Geodetic System

Definition According to Geodesy¹, the Earth is a geoid (taking the mean sea level), an irregular sphere, slightly flattened at the poles and bumpy from one continent to another. To model this surface, we use a more regular geometrical figure, the ellipsoid defined by its half major axis A, its half minor axis B and its flattening. [15]

¹Geodesy is the science of accurately measuring and understanding three fundamental properties of the Earth: its geometric shape, its orientation in space, and its gravity field.

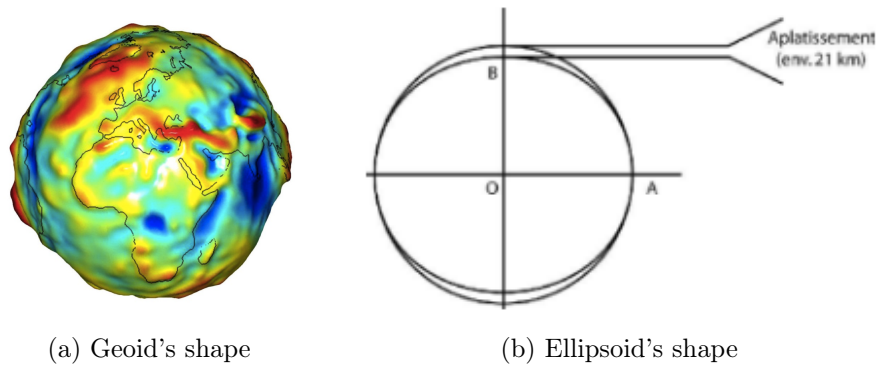


Figure 3.2: Earth's shape

Source: geoconfluences.ens-lyon.fr

The table below provides the parameters of the most commonly used ellipsoids in geodetic systems:

Table 3.2: Ellipsoids in global geodetic systems

Name	A (m)	Flattening $(A-B)/A$
WGS 84	6378137	1/298.257223563
Clarke 1880	6378249.145	1/293.465
South American 1969	6378160	1/298.25
Geodetic Reference System 1980	6378137	1/298.257222101

To define a geographic coordinate system, an ellipsoid is not enough. The definition of a geodetic system requires knowledge of the following parameters:

1. The shape of the ellipsoid (major axis, minor axis),
2. Offset (in m) of the ellipsoid (D_x , D_y , D_z) with respect to the WGS84 geodetic system,
3. Angle of rotation (in degrees) of the X, Y and Z axes of the mark (R_x , R_y , R_z),

parameters.[16]

3.3.1.1 Types of geodesic systems

There are two types of geodesic systems:

1. Global geodesic systems, or geocentric systems:

It is used when we need coordinates at the scale of the whole Earth. The origin of this ellipsoid will be the center of the Earth. Example: WGS84, it has become the reference for geocentric systems. Its accuracy is on average 1 to 2 meters.

2. Local geodetic systems:

A local datum aligns its ellipsoid to closely fit the earth's surface in a particular area. It is much more precise than a geocentric system in the area for which it has been defined.[17]



Figure 3.3: Local and Geocentric coordinate systems

Source: www.esri.com

The table 3.3 gives some examples of local and global datum:

Table 3.3: local and global Datum

Datum	Ellipsoid	Dx	Dy	Dz	Region of use
North Sahara 1959	Clarke 1880	-186	-93	310	Algeria
Merchich	Clarke 1880	31	146	47	Morocco
European 1950	International 1924	-87	-98	-121	EU
WGS 1984	WGS 84	0	0	0	Global Definition

3.3.2 Projections

In order to better visualize a map, we must go from the 3D representation of the earth to a representation in 2D (the map). This involves establishing a mathematical function such as:

$$(X, Y) = F(\textit{latitude}, \textit{longitude})$$

No projection completely preserves the metric properties of the ellipsoid, there is sometimes a loss of the values of the angles or the size of the surfaces

3.3.2.1 EPSG

EPSG stands for European Petroleum Survey Group. EPSG codes are a list of geo-referenced projection coordinate systems used as a GIS standards.

3.3.2.2 Projection used in Algeria

We use in Algeria the UTM projection. UTM stands for Universal Transverse Mercator. The UTM splits the world into a series of 60 vertical slice of meridians spaced 6° and 20 parallel strips of 8° except the most north which is 12° wide. They are designated from north to south by a letter of the alphabet from C to X using not the letters I and O.[18]

Algeria spreads from west to east in four zones: 29, 30, 31 and 32. It is 9° west of the meridian origin and 12° east of the meridian of origin. It's EPSG code is 30731.

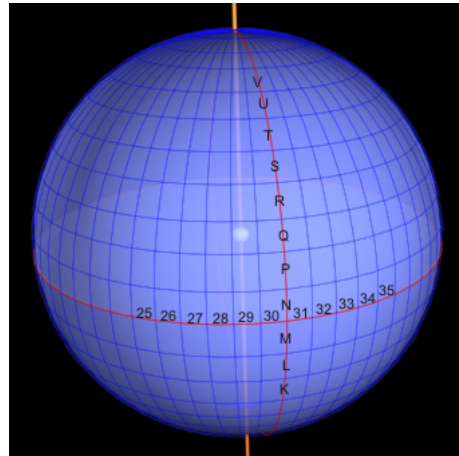


Figure 3.4: UTM projection

Source: gisgeography.com

3.3.3 Types of maps

There are different types of map [19]:

Table 3.4: Main types of maps

Type of map	Definition
Political map	It focuses solely on the state and national boundaries of a place.
Physical map	It generally shows things like mountains, rivers, and lakes.
Topographic map	It uses contour lines instead of colors to show changes in the landscape
Climate map	It shows information about the climate of an area(temperature, amount of snow received)
Road map	It shows major and minor highways and roads
Thematic map	It focuses on a particular theme or special topic like population density

3.4 Type of GIS applications

Nowadays, GIS is more and more globalized, integrated and popular. In general, computerized GIS are classified into two categories however different architectures can be drawn by combining them.

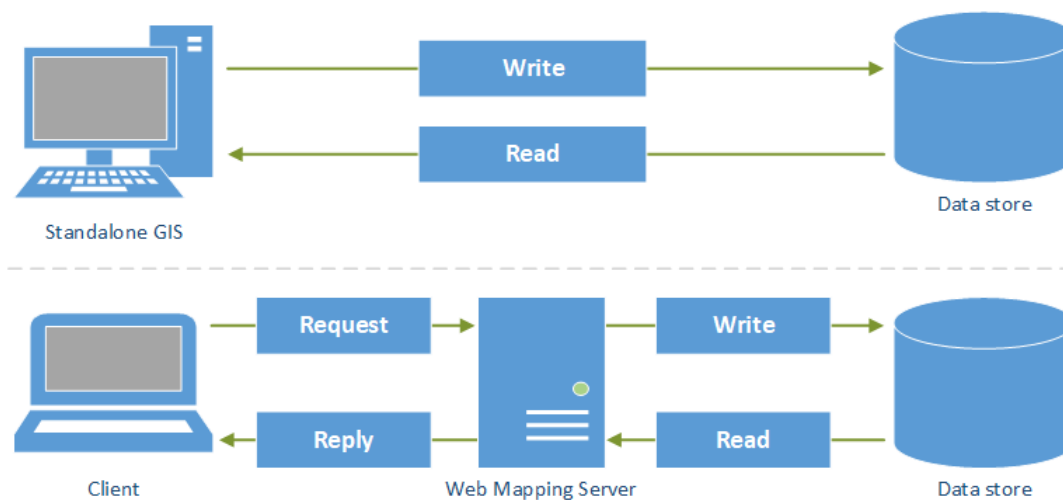


Figure 3.5: Standalone Vs. Client/Server Architecture

3.4.1 Standalone GIS

Standalone GIS is a software that allows users to manipulate spatial data directly from a database, we can list two majors types:

- **Desktop GIS** It includes GIS software that runs locally on a personal computer commonly used by GIS professionals to create and design maps. Eg.: GRASS GIS.
- **Mobile Software** It includes GIS software that runs on a mobile platform and provides special functions that support data acquisition and updating in the field. Eg.: gvSIG Mobile and Quantum GIS for Android.[1]

3.4.2 Client/Server GIS

This category includes software that enables a developer to deliver (via a server software) and view (via a client software) geographic data and maps. It does not communicate directly with the database.

3.4.2.1 Client Side

It contains GIS features and it mainly requests for geographic data from a Web Map server. There are two types of architecture namely thin and thick client depending upon the client processing power:

- **Thin Client**

In this case, the processing power lies in the server and the results are displayed through a browser. This type of system requires frequent communication with the server even

for simple operations like zoom, pan as the client requires the completely processed data for display. Generally, thin GIS clients are Web clients.

- **Thick client**

In thick client architecture, data and logic can reside locally on the client side. This reduces the time delay caused due to frequent communication between client and server. Generally, thick GIS clients are Desktop software such as QGIS.

3.4.2.2 Web Map Server

In essence a web map server is a specialization of a standard web server that can provide Web Mapping Services (WMS), Web Feature Services (WFS), or Web Coverage Services (WCS). Eg.: Geoserver, Mapserver.

1. Web Map Service (WMS)

Web Map Service (WMS) is the standard process of accessing map data from map servers. At each request map tiles will be generated on the server and sent back to the client in a raster format like PNG, JPEG or vector format like SVG².

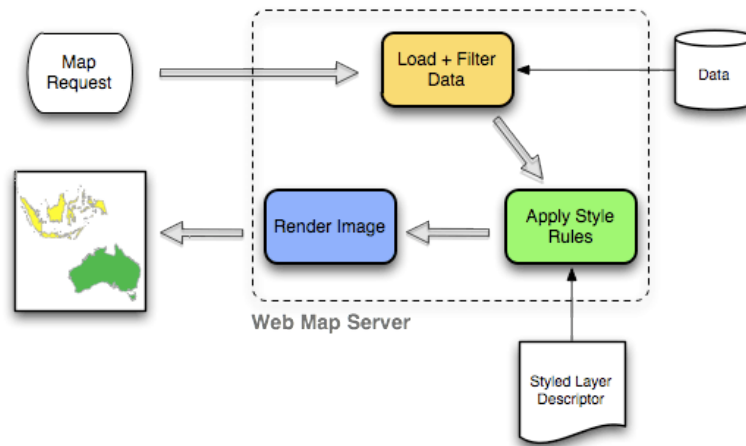


Figure 3.6: Web Map Service

Source: workshops.boundlessgeo.com/

2. Web Map Tile Service (WMTS)

A map service that fulfills requests with pre-created tiles from the cache is called a cached or tiled maps service.

3. Web Feature Service (WFS)

Unlike a WMS, which provides the user only with a map which he cannot edit, a basic WFS allows querying and retrieval of spatial data.

²Scalable Vector Graphics

A transactional Web Feature Service (WFS-T) allows creation, deletion, and updating of spatial data through the Web.

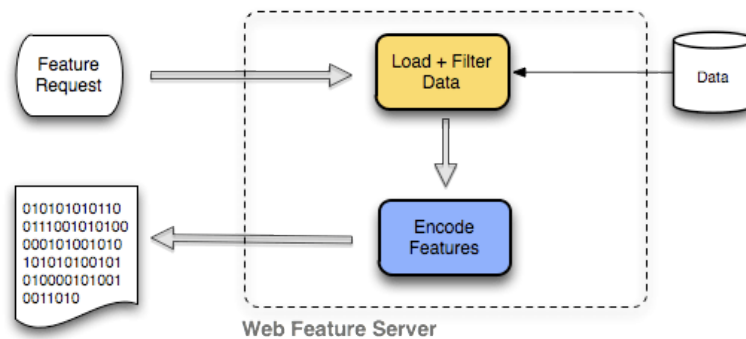


Figure 3.7: Web Feature Service

Source: workshops.boundlessgeo.com/

4. Web Coverage Service (WCS)

The WCS web service is used to transfer “coverages”, ie. objects covering a geographical area. One can think of WCS as the equivalent of WFS, but for raster data instead of vector data. A WCS service returns data in a format that can be used as input for analysis and modeling. This is in contrast with the OGC³ Web Map Service (WMS), which only returns a picture of the data.

5. Web Processing Service (WPS)

A WPS is an OGC standard to invoke serve-side geoprocessing services which are faster than the client-side [20]. It defines an interface that facilitates the publishing of geospatial processes and clients’ discovery of and binding to those processes. WPS can describe any calculation (i.e. process) including all of its inputs and outputs, and trigger its execution as a web service. For example the transformation between 2 systems of coordinates.

3.5 Data store

Each one of the previous architectures needs access to a data store. The data store in a GIS hosts both files and databases. It supports the two primary spatial data types: vector and raster.

³The Open Geospatial Consortium (OGC) is an international voluntary consensus standards organization. In the OGC, more than 500 commercial, governmental, nonprofit and research organizations worldwide collaborate in a consensus process encouraging development and implementation of open standards for geospatial content and services, sensor web and Internet of Things, GIS data processing and data sharing.

3.5.1 Vector data

It is represented as either points, lines, or polygons. Data that has an exact location or hard boundaries are typically shown as vector data. Eg.: country boundaries, the location of roads and railroads using lines, or point data indicating the location of fire hydrants.

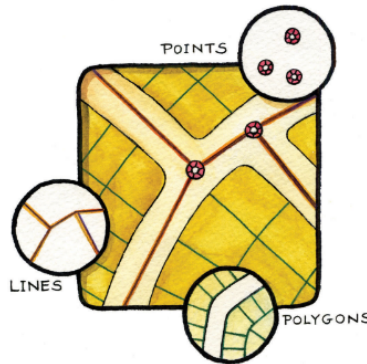


Figure 3.8: Vector's data

Source: saylordotorg.github.io

Some of the most famous vector formats are:

1. Esri⁴'s Shapefile: The shapefile format is made up of at least three files :
 - (a) filename.shp contains the geometric data,
 - (b) filename.dbf contains the attribute data,
 - (c) filename.shx is the link between the file shp and dbf. It also stores the index of geometry thus decreasing the search time space.
2. Keyhole Markup Language (KML): KML was originally used for viewing geographic data in Google Earth but it became a standard and has become more widespread as a GIS data exchange format.
3. GeoJSON: is an open standard format designed for representing simple geographical features, along with their non-spatial attributes. It is based on JSON, the JavaScript Object Notation.

3.5.2 Raster data

Raster data are viewed as a series of grid cells where each cell has a value representing the feature being observed. Data stored in a raster format represents real-world phenomena:

1. Thematic data that represents features such as land-use or soils data.
2. Continuous data that represents phenomena such as temperature, elevation.
3. Satellite images, aerial photographs and pictures that include scanned maps or drawings.

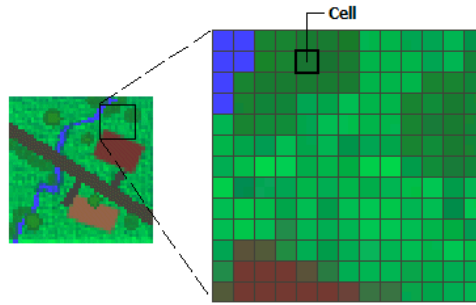


Figure 3.9: Raster data

Source: desktop.arcgis.com

Some of the most famous raster formats are:

1. GeoTIFF: GeoTIFF is a form of TIFF (Tag Image File Format) format for georeferenced raster data.
2. GRIB, GRid In Binary: GRIB is the World Meteorological Organisation's (WMO) standard for grid-based meteorological data.

3.5.3 Spatial database

Definitions:

1. **A database** is a data structure that stores organized information. Most databases contain multiple tables, which may each include several different fields.

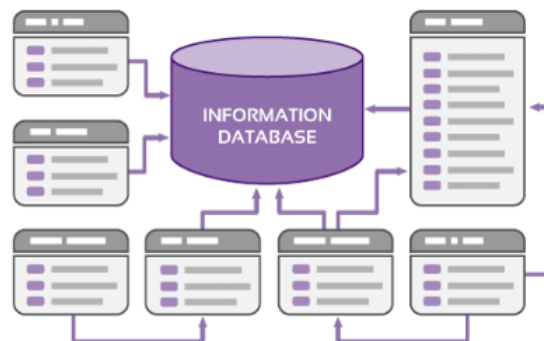


Figure 3.10: Database

Source: www.kbmanage.com

2. Database Management System

A database management system (DBMS) is a collection of programs that enables users

⁴Esri is an international supplier of geographic information system (GIS) software, web GIS and geo-database management applications. The company is headquartered in Redlands, California.

to create and maintain a database. Some DBMS examples include MySQL, PostgreSQL, Microsoft Access, SQL Server, FileMaker, Oracle.

3. **Spatial Database Management System (SDBMS)** provides an alternative to storing geographical data in files. Usually a spatial DBMS is used for large data sets that need high performance. SDBMS could be defined as a DBMS that offers :

- **Spatial data types:**

Spatial data types or geometric data types provide a fundamental abstraction for modeling the geometric structure of objects in space as well as their relationships, properties, and operations.[21]

- **Spatial indexing:**

Indexing an ordinary database is usually done with B-tree indexes⁵, however it is not optimal in the case of spatial data types. Fortunately, spatial databases provide a “spatial index” to optimize spatial queries by using bounding boxes, the smallest rectangle capable of containing a given feature, so rather than giving accurate results, spatial indexes provide approximate results but perform quickly.



Figure 3.11: Bounding Boxes

Source: workshops.boundlessgeo.com

- **Spatial analysis operations:**

In addition to typical SQL queries such as SELECT statements, spatial databases can perform a wide variety of spatial operations [22]

- **Spatial Measurements:** Computes line length, polygon area, the distance between geometries, etc.
- **Spatial Functions:** Modify existing features to create new ones, for example by providing a buffer around them, intersecting features, etc.
- **Spatial Predicates:** Allows true/false queries about spatial relationships between geometries. Examples include "do two polygons overlap".
- **Geometry Constructors:** Creates new geometries, usually by specifying the vertices (points or nodes) which define the shape.

⁵A B-tree partitions puts the data into the hierarchical tree using the natural sort order.

- **Observer Functions:** Queries which return specific information about a feature such as the location of the center of a circle

The most used SDBMS are PostGIS, which provides spatial data types and analysis functions for the free PostgreSQL database, MySQL Spatial project for to the free MySQL database, and the SpatiaLite project for SQLite.

3.6 Free and open source GIS software

Software that is used to create, manage, analyze and visualize geographic data is usually incorporated under the umbrella term ‘GIS software’. Steiniger and Weibel (2009) identified seven major types of GIS software[23]: Desktop GIS, Spatial Data Base Management Systems (SDBMS), Web Map Server, Server GIS, Web GIS clients, Mobile GIS, and Libraries and Extensions. The figure 3.12 [1] list the free and open source geographic information software map for 2012.

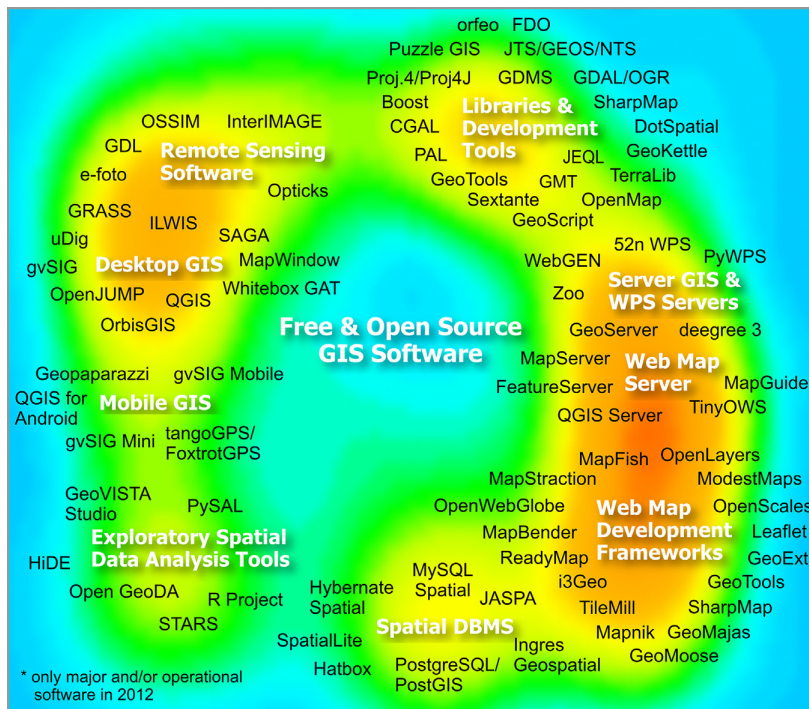


Figure 3.12: The free and open source geographic information software map of 2012.[1]

3.7 Conclusion

In this chapter, we had a brief but complete introduction to spatial data and GIS architecture. We concluded by presenting a state-of-art of Open Source GIS software that will be used, in order to compare and choose the most appropriate solution for our IoT oriented GIS.

Chapter 4

Technical choices

4.1 Introduction

In this chapter, we will first present CDAT¹'s OpenStack cloud where our work was hosted and then compare various open source software to choose the most relevant one for each block of our GIS architecture: IoT platform, spatial database and GIS server.

We will summarize by presenting our end-to-end solution architecture.

4.2 Cloud platform

Our work was hosted on the Cloud Computing infrastructure of CDAT. It is a public cloud that allows easy access, on demand, anytime and anywhere to shared and configurable computing resources.

The physical infrastructure of the CDTA cloud is made up of a set of powerful equipment (servers, switches, routers and firewalls) ,powered by a stable and redundant electrical system, that provides the management of the cloud itself, virtual machine hosting, supercomputing and storage in order to satisfy high availability and the increasing demand of users for resources.

It offers the "IaaS: Infrastructure as a Service"² service model that is deployed using an open-source software layer known as "OpenStack".

OpenStack is a set of Open Source software tools for building and managing cloud computing platforms for public and private clouds. It uses virtualization technology to break up the physical system into several distinct and secure virtual environments. This virtualization makes it possible to determine the processing power, the storage space and the memory to be allocated to the various applicants.

¹Center for Development of Advanced Technologies

²IaaS provides access to computing resources in a virtualized environment “the cloud” on internet. It provides computing infrastructure like virtual server space, network connections, bandwidth, load balancers and IP addresses. The pool of hardware resource is extracted from multiple servers and networks usually distributed across numerous data centers. This provides redundancy and reliability to IaaS.

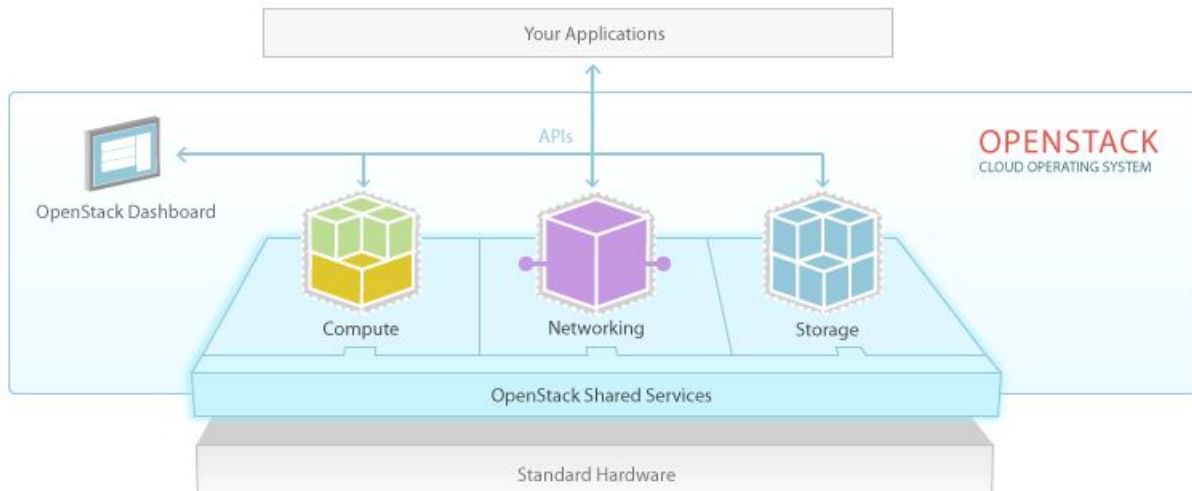


Figure 4.1: OpenStack Architecture

Source: openstack.org

4.3 IoT platform

Sentilo has been chosen by the "Algiers Smart City" project's team as the IoT platform solution in order to exploit the data generated by the layer of sensors deployed in the city.

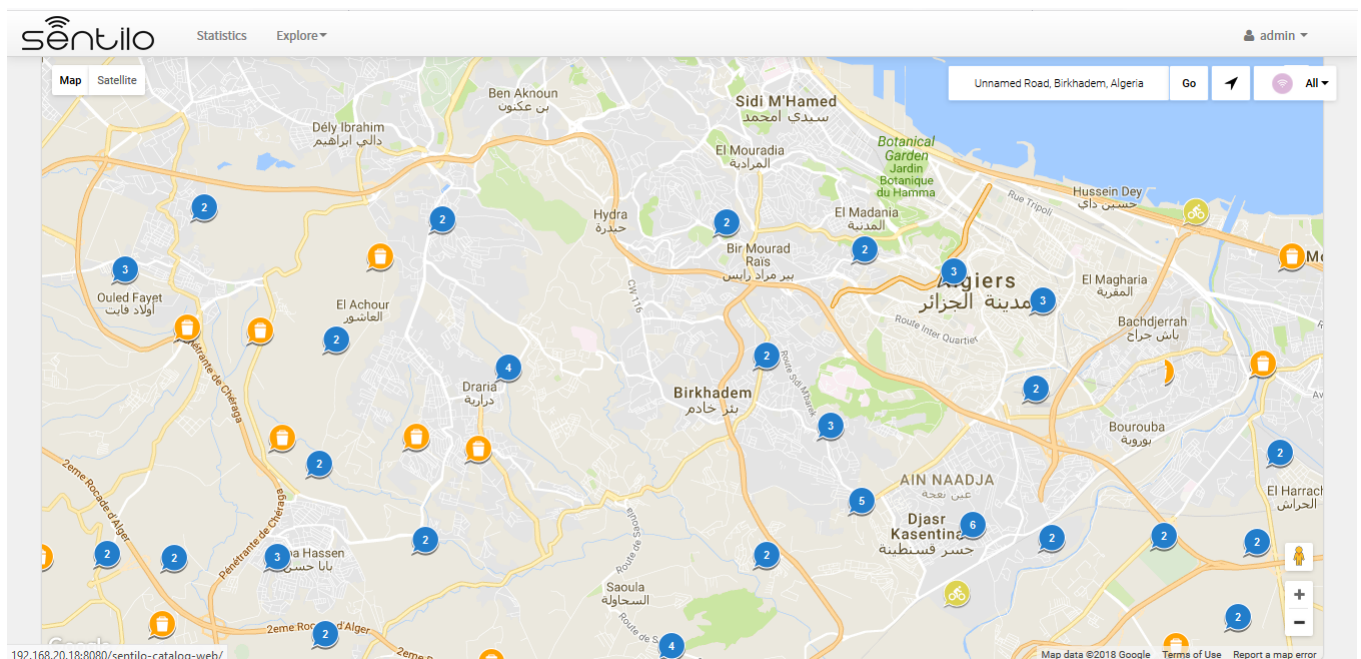


Figure 4.2: Sentilo's user interface

4.3.1 Definition

Sentilo is an open source sensor and actuator platform. It was started in November 2012 by the Barcelona City Council, through the Municipal Institute of Informatics (MII), for its smart city project called City OS.

4.3.2 Architecture

Sentilo is the piece of architecture that isolates applications that use the sensor data from the layer that provides the data.

The main modules of Sentilo are:

1. Real Time Storage

It is the primary repository where the platform stores all the information received (observations, alarms orders). It uses Redis as a database.

2. PubSub Platform

It allows the platform clients (application/modules or provider/sensors) to publish or retrieve information and to subscribe to system events.

This module is a stand-alone Java process that uses Redis as a publish/subscribe mechanism.

3. Catalog

Catalog is a web application built with Spring on the server side using jQuery and bootstrap as presentation layer and MongoDB as data storage database.

It provides tools to administer, rule and monitor the Sentilo platform resources : providers, applications, components, sensors, alerts and users.

The Catalog allows us also to display some Sentilo statistics like total requests processed, number of sensors registered, current requests per second, max daily average and max average requests per second...

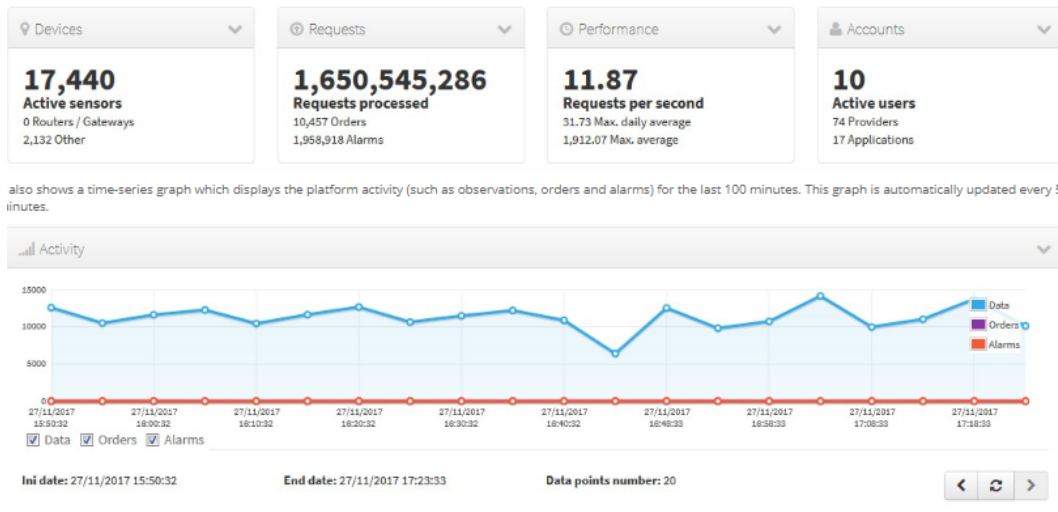


Figure 4.3: Sentilo statistics

4. Restful API

Sentilo gives the possibility to create components and sensors, publish and read data through Restful API requests, that we will detail in the next sections.

5. Agents

Agents are Java processes that expand the core functionality of the platform using the Redis publish and subscribe mechanism.

Sentilo currently provides several agents:

- **Relational database agent:** used to export all information received from Pub-Sub platform into a set of relational databases.
- **Alert agent:** used for processing each data received by the platform and publish a notification when any of the configured integrity rules are not met.
- **Activity Monitor Agent:** subscribes itself to the list of events and uploads them to a search engine Elasticsearch.
- **Historian Agent:** Likewise the Activity Monitor Agent, this module also subscribes itself to the list of events and uploads them to an OpenTSDB database.

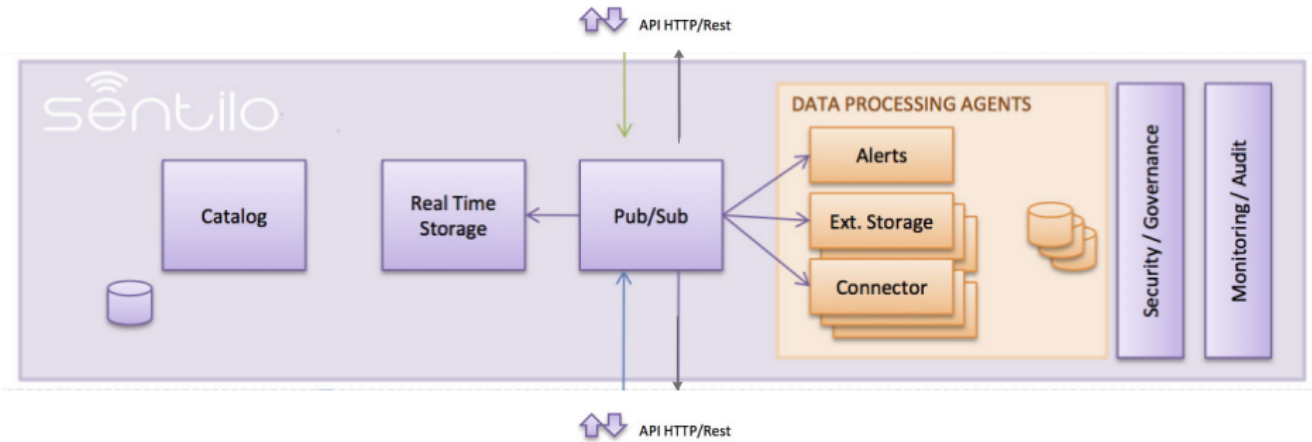


Figure 4.4: Sentilo Architecture

Source: sentilo.io

4.3.3 Sentilo resources

Sentilo components are :

1. **Sensor**

Any software or hardware device that generates data in the Smart City. All sensors are identified by sensor type, data type and the measurement units.

2. **Component**

A component represents one or more sensors and it is identified by its location.

3. **Provider** A provider is an entity that manages devices (sensors). Each provider is defined by a name and an authentication key that allows him to publish and read data through the REST API.

4. **Client Application**

Application are the data clients of the Sentilo platform

4.3.4 Sentilo's HTTP REST API

In this part we will take a closer look at the Sentilo REST API. We will start by defining what is an API and a REST API. Then, we will list the operations that can be performed by Sentilo API and how to use them.

- **Application Programming Interface (API)**

An API represents a set of functions that allows two or more applications to communicate with each other.

APIs make it easier for developers to use certain technologies in building applications by abstracting the underlying implementation and only exposing objects or actions the developer needs.[24]

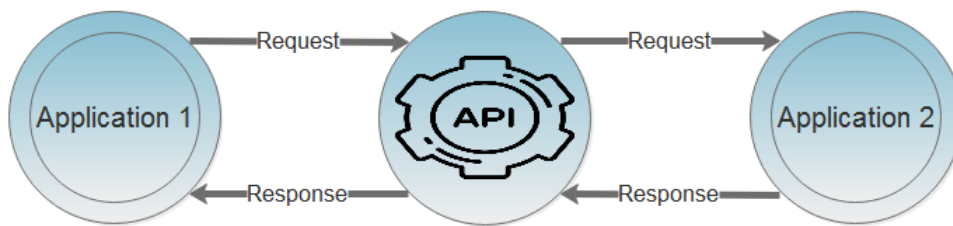


Figure 4.5: API

- **REST API**

A RESTful API is an API that takes advantage of the HTTP methodologies, the table below summarize this:

Table 4.1: REST API Methods

Method	Description
GET	To retrieve information of a resource
POST	To change the state of or update a resource
PUT	To create that resource
DELETE	To remove it

- **Operations of Sentilo REST API**

The services offered by Sentilo through its REST API can be classified into five main groups:

1. **Catalog:** provides operations to insert, update, query and delete catalog resources (sensors, components and alerts).
2. **Data:** provides operations to publish, retrieve, delete the observations of the registered sensors.

3. **Order:** provides operations to publish, retrieve, delete orders to sensors and actuators.
4. **Alarm:** provides operations to publish, retrieve, delete alarms associated with alerts stored in catalog.
5. **Subscribe:** provides operations to subscribe, retrieve and cancel subscriptions to system events(data, order, alarm).

- **Connect to the REST API**

In order to connect to this REST API, we have to specify:

1. **An identifier:** an URL in the form *http(s)://domain:8081/service*
2. **The format of data:** JSON or XML.
3. **Operators:** GET, POST, PUT, DELETE.
4. **Authorization token:** a string of letters and numbers unique to every provider.

- **Example**

The API is in the form of

```
curl -X METHOD -H "IDENTITY-KEY: Authorization-token"
-H "Content-Type: application/json" URL
```

In the table below gives an example of a publish request.

Table 4.2: Publish Data

URL	http://server-ip:8081/data/providerId/sensorId
Method	PUT
Content-Type	application/json
IDENTITY_KEY	Authorization token
Observation	{ "observations":[{ "value":"9.6", "timestamp":"17/02/2016T11:43:45CET", "location": "41.3888 2.15899"}]}

4.4 Spatial Database Management System

Due to limited system resources, Sentilo does not store the sensor data forever. Thus, we have to copy it regularly in a Big Data database and, as seen previously, Sentilo has four agents that expand its core functionality.

In this section, we will discuss each Sentilo agent and the possibilities it gives. Next, we will compare two paradigms for systems that manage Spatial Big Data then choose the best one.

4.4.1 Sentilo agents comparison

1. Activity Monitor Agent:

This agent is a connector to the search engine ElasticSearch, an open-source, RESTful, distributed search and analytic engine. ElasticSearch is suitable only for real time analysis, not as long term repository. Add to that it does not support GIS features and does not allow the connection to a GIS Server.

2. Historian Agent:

This module is a connector to OpenTSDB, a scalable time series database built on top of Hadoop and HBase³. It stores and analyzes large amounts of time-series data generated by endpoints like sensors or servers. However, it does not support GIS functionality and at the same time does not allow the connection to a GIS Server.

3. Alert Agent:

This agent is used as a notifier when any of the configured integrity rules are not met. It is not used for storage or analysis.

4. Relational Agent

The relational agent is used to export all information received from PubSub platform into a set of relational databases such as MySQL, Oracle and PostgreSQL. PostgreSQL has a powerful extension called Postgis, that allows GIS features and connection to a GIS server.

4.4.2 PostgreSQL/PostGis vs. Geomesa

4.4.2.1 PostgreSQL/PostGis

PostgreSQL is an open source relational database management system (DBMS) developed by a worldwide team of volunteers. It provides transaction management, disk storage routines, SQL processing, planning, etc. It supports text, images, sounds, and video, and includes programming interfaces for C/C++ , Java ...

PostGIS is a spatial database which is build on top of PostgreSQL. It provides spatial data types (point, linestring, polygon...), functions (area, distance, union, difference, buffer, touches..) and indexes.

³Apache HBase is an open-source, distributed and non-relational database



Figure 4.6: PostgreSQL and PostGIS Database

Some of the most used PostGIS functions are:

Table 4.3: PostGIS functions

PostGIS Functions	Description
ST_MakePoint(Longitude, Latitude)	Returns a new point.
ST_AsGeoJSON(geometry)	Returns a geometry to a standard GeoJSON format.
ST_Area(geometry)	Returns the area of the geometry in the units of the spatial reference system.
ST_Distance(geometry, geometry)	Returns the distance between two geometries in the units of the spatial reference system.
ST_Crosses(geometry, geometry)	Returns true if a line or polygon boundary crosses another line or polygon boundary, otherwise false.

PostGIS provides most of the Gis functions needed to store and analyze spatial data efficiently. However, the current deluge of big-data, especially in the case of a Smart City, requires spatially-aware big-data platforms which are capable of performing geospatial analytics on distributed computing systems.

The common solutions for a spatial Big Data problem are Geomesa/Accumulo and POSTGIS/Greenplum database. We will compare the two paradigms.

4.4.2.2 Geomesa

GeoMesa enables indexing and querying of spatiotemporal data on key/value databases such as Accumulo⁴.

Similar to the relationship between PostGIS and PostgreSQL, GeoMesa is a library added-on to Accumulo to enable support for spatiotemporal data. GeoMesa uses and extends the OGC SimpleFeature data model and supports spatiotemporal indexing.

A study that compares two solutions Postgis and Geomesa showed that GeoMesa query time is significantly worse than the Vector Cluster⁵ and PostGIS time[2], moreover the doc-

⁴It is not limited to Accumulo, it supports other databases such as HBase, Cassandra, Kafka, etc. But Accumulo is best supported.

⁵Vector Cluster is a geospatial data storage format created by the Geospatial Computing Section at the United States Naval Research Laboratory (NRL)

umentation of Geomesa was complicated to implement.

System	Mean query time (seconds)	standard deviation
GeoMesa	17.50	0.6014
Vector Cluster	1.526	0.0552
PostGIS	0.8024	0.0326

Figure 4.7: Performance comparison[2]

We found a good alternative for Geomesa: "Greenplum", which is a massive parallel processing (MPP) database based on PostgreSQL open-source technology. In addition to providing GIS features for Spatial Big data through PostGis it can connect to Sentilo through the relational agent.

4.4.3 Greenplum

4.4.3.1 Definition

Pivotal Greenplum Database is an advanced, fully featured, open source (since 2015) data platform. It provides powerful and rapid analytics on petabyte scale data volumes. It is essentially several PostgreSQL disk-oriented database instances acting together as one cohesive database management system (DBMS)[25].

4.4.3.2 Greenplum architecture

Greenplum Database is composed of a master and segments. The master coordinates its work with the segments, which store and process the data.

1. Greenplum Master:

The master is the entry point where clients connect and submit SQL statements to the Greenplum Database. It stores the Metadata about the Greenplum Database itself but it does not contain any user data. The master authenticates client connections, processes incoming SQL commands, distributes workloads among segments, coordinates the results returned by each segment, and presents the final results to the client program.

2. Greenplum Segments

Greenplum Database segment instances are independent PostgreSQL databases that each store a portion of the data and perform the majority of query processing.

3. Greenplum Interconnections:

The interconnect is the networking layer of the Greenplum Database architecture. The Greenplum software uses User Datagram Protocol Interconnect with Flow Control (UDP-IFC) to send messages over the network and performs packet verification which couples the reliability of Transmission Control Protocol (TCP), and the performance and scalability of UDP.

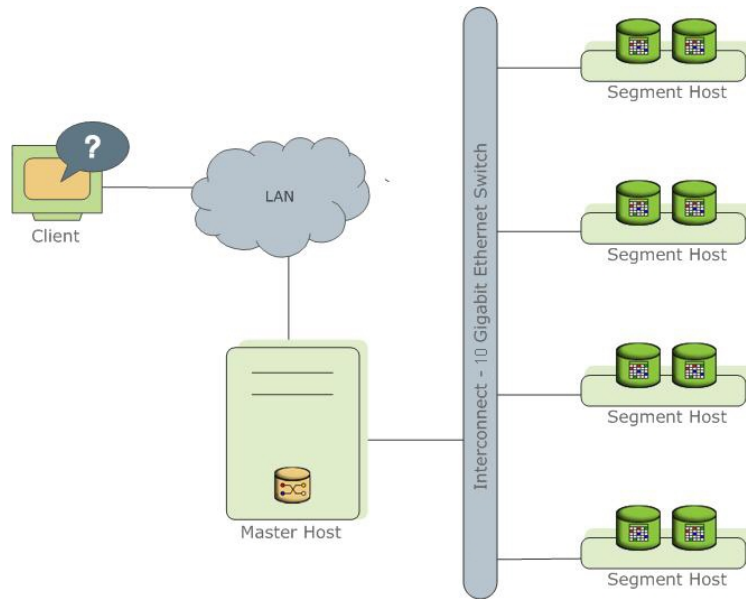


Figure 4.8: High-Level view of the Greenplum Database Architecture

Source: gpdb.docs.pivotal.io

Since Greenplum is a cluster of Postgres database, it gives the possibility to add the spatial extension PostGIS. It also has a powerful open-source library of machine learning's algorithms called: MADlib.

The table 4.4 summarize the main characteristics and limits of Greenplum:

Table 4.4: Characteristics and limits of Greenplum (source: community.pivotal.io)

Dimension	Limit
Maximum size for a database	unlimited
Maximum size of a table	unlimited, 128 TB per partition per segment
Maximum number of rows in a table	2^{48}
Maximum number of columns in a table	1600
Maximum number of tables per database	4200 million
Maximum active concurrent transactions	unlimited

4.5 Web mapping server

The Server Side in the most crucial part in a GIS architecture because it will define its capabilities and features.

In our journey to choose the most suitable GIS server, we analyzed some governmental geoportals, using the Google Chrome DevTools Network, to determine the most used GIS Servers and then compared them based on their capabilities.

In each geoportal we followed the same steps:

1. We sent requests to the GIS server by zooming on the map,
2. The GIS server generated new maps corresponding to the zoom level,
3. At the same time, we tracked the request using Chrome DevTools Network.

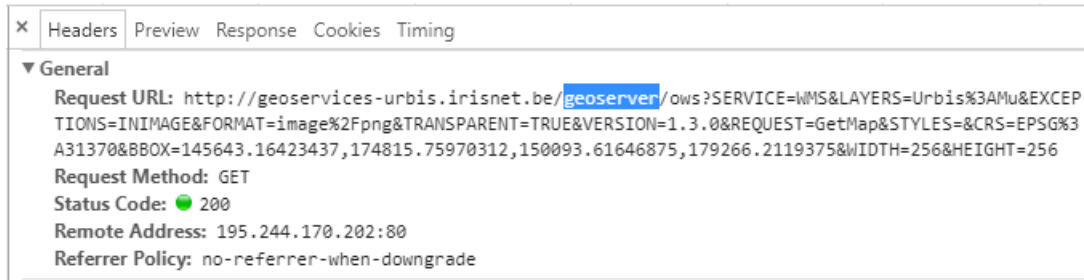


Figure 4.9: Chrome’s Dev Tools

The table 4.5 summarize the results obtained:

Table 4.5: Comparison of different Geoportals

Geoportal	GIS Server used
French geoportal https://www.geoportail.gouv.	GeoServer
Geoportal of Wallonie http://geoportail.wallonie.be	MapServer
Geoportal of Brussels http://geobru.irisnet.be	Geoserver
Commision for the protection of agricultural Land in Quebec Portal http://www.cptaq.gouv.qc.ca/	MapServer
Global Earth Observation System of Systems (GEOSS) Portal http://www.geoportal.org/	GeoServer
Australian flood risk information portal http://www.ga.gov.au/	GeoServer

We can notice that the most used GIS servers are *MapServer* and *GeoServer*. Next we will compare them, based on their capabilities, in order to choose the most suitable solution for our architecture.

4.5.1 GeoServer Vs. MapServer

GeoServer:

GeoServer is an Open Source Java based server that allows map creating and spatial data sharing [26]. It handles both Raster and Vector data, and works under OGC main standards such as: WMS, WFS, WCS, WPS and WFS-Transaction.

GeoServer has a web administration interface for configuring its settings.

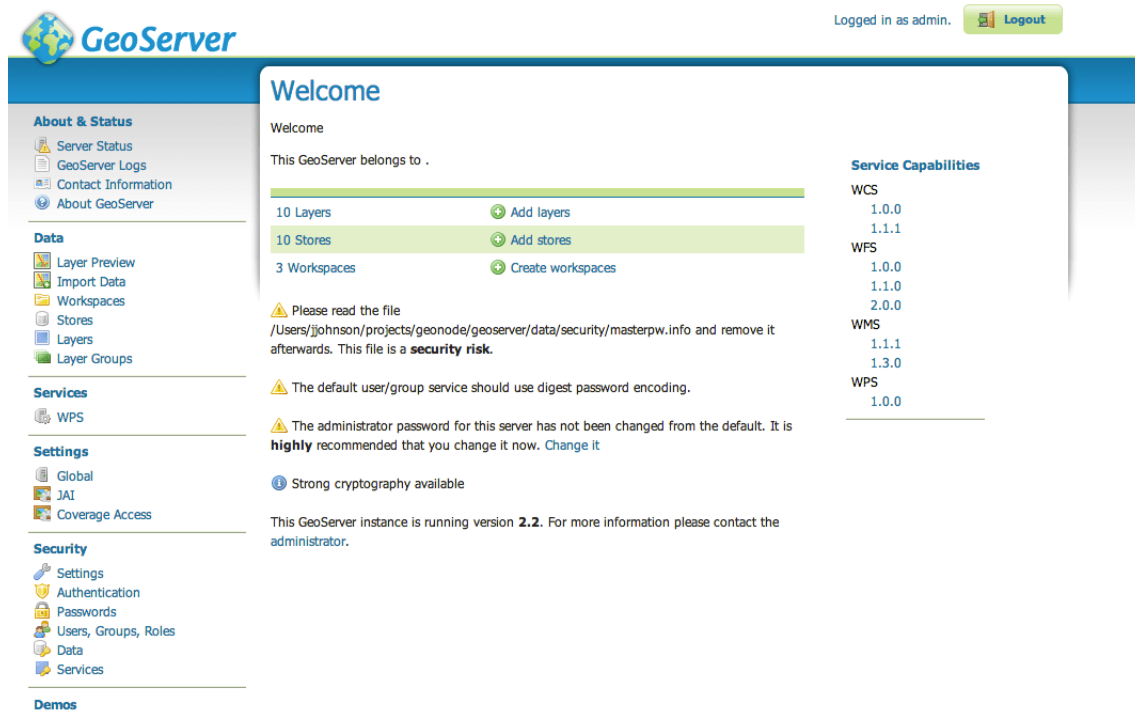


Figure 4.10: GeoServer’s User Interface

MapServer:

To quote from the MapServer home page, “MapServer is an OpenSource development environment for building spatially enabled Internet-web applications.” It also handles both vector and raster data and works under the main OGC standards but unlike GeoServer it does not support Modification Web Services (WFS-T) and server-side geoprocessing services (WPS).

A simple MapServer application consists of [27]:

1. **Map File:** A structured text configuration file (.map) that defines the needed parameters to generate a map like data source, projections, and the path of output image.
2. **MapServer CGI/ MapScript:** The binary or executable file that receives requests and returns images, data, etc. Mapscript provides a scripting interface for MapServer for the construction of Web applications. It is a loadable module that adds MapServer capability to a scripting language like PHP, Perl, Python, etc ..
3. **Web/HTTP Server:** It serves up the HTML pages when hit by the user’s browser. Most of the time Apache HTTP Server is used.

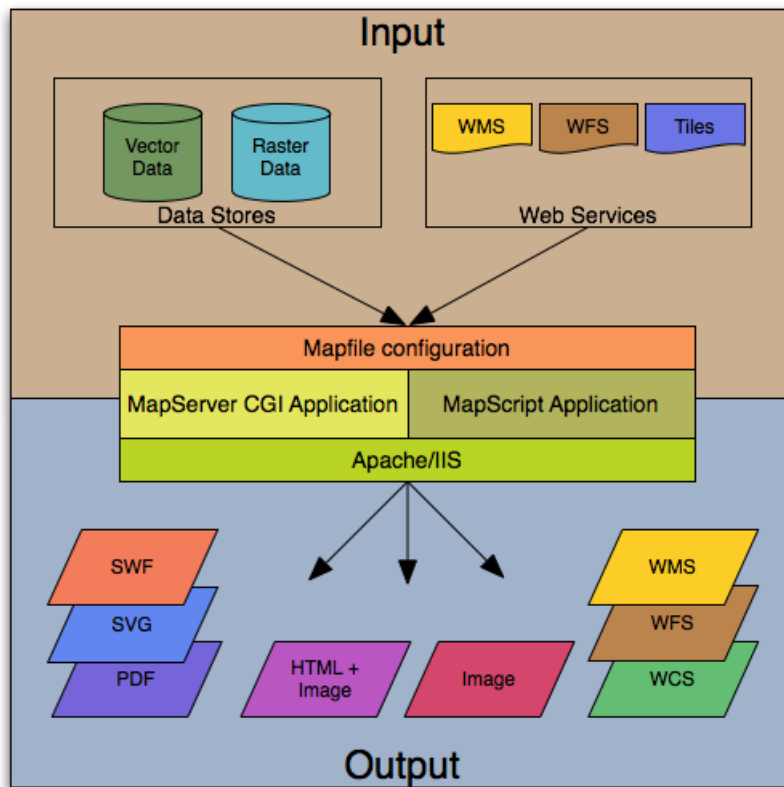


Figure 4.11: MapServer's architecture

Source: www.mapserver.org

The table 4.6 summarize the comparison between the two GIS Servers:

Table 4.6: MapServer Vs. GeoServer

GIS Server	GeoServer	MapServer
Administration	User friendly Tool	Mapfile
Technology	Java based	C/C++
Vector data	YES	YES
Raster data	YES	YES
OGC Services	WMS/WFS/WCS	WMS/WFS/WCS
OGC Modification's Services WFS-T	YES	NO
Server side GeoProcessing WPS	YES	NO

After comparing the two open solutions, we opted for *Geoserver*. Unlike MapServer, GeoServer uses WPS and WFS-T OGC standards which allow a better flexibility in spatial

data management and geoprocessing. [28].

4.5.2 Geoserver

Since GIS server is a crucial element in our architecture, we will take a closer look to GeoServer's architecture and main components in order to better exploit them.

4.5.2.1 Architecture

Geoserver is built on top of Open Source Java libraries: *GeoTools*, *Java Topology Suite*, *Spring Framework* and *Freemaker*. To better understand how it works we will define each one of them:

1. **GeoTools:**

GeoTools is an open source Java library[29]. It is the equivalent of PROJ⁶ and GDAL/OGR⁷ all assembled into one. It includes lots of plug-ins to support different data formats (vector and raster) like ESRI Shapefile, Geographic Markup Language (GML) and Geotiff. It supports also spatial databases such as PostGIS, Oracle Spatial and MySQL.

2. **Java Topology Suite (JTS):**

JTS is a Java library for creating, manipulating and representing geometric objects and the relationships among them. It provides robust implementations of the fundamental geometric functions.

3. **Spring Framework:**

Spring is an open source framework that allows Java developers to easily build applications on the Java EE (Enterprise Edition) platform. It has many positive features such as being lightweight and modular.

Using Spring Framework in 2006 made Geoserver more component-oriented architecture which allows developers to extend it via simple and isolated plug-ins without having to learn the inner works of the entire system.

Spring has a complete security subsystem that supports a wide variety of authentication schemes which is essential for GeoServer.

4. **Freemaker:**

Freemaker is an open source 'template engine' which takes data, runs it through a series of rules and logic, and produces a presentation of that data based on those rules.

⁶Program proj is a software library that converts geographic longitude and latitude coordinates into Cartesian coordinates

⁷It is a software library for writing and reading spatial data in different data format.

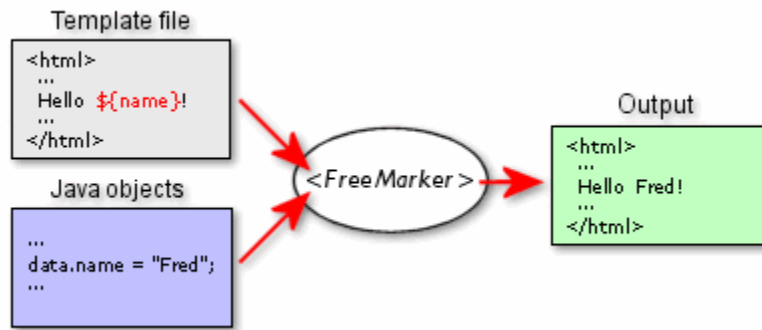


Figure 4.12: Freemarker Example

Source: freemarker.apache.org

The figure 4.13 shows the architecture of Geoserver:

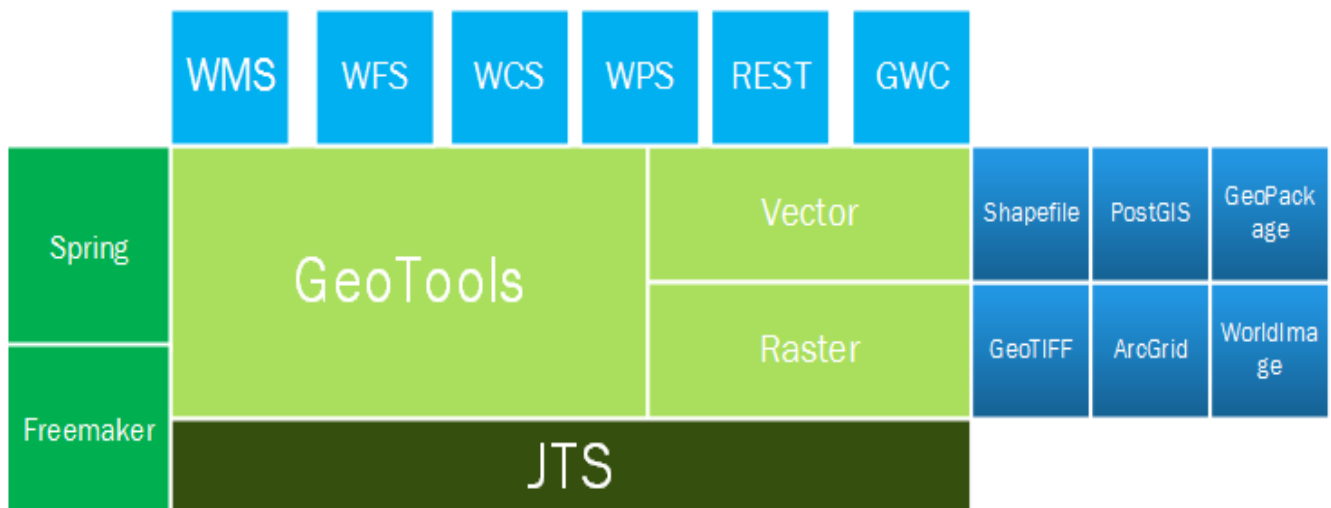


Figure 4.13: Geoserver Architecture

4.5.2.2 Geoserver concepts

In this section we will introduce some of the commonly-used terms in Geoserver:

1. Workspace

A workspace is a virtual container related to a certain project, it organizes the layers by grouping similar data together. Different workspace allows us to create layers and stores with the same name.

2. Store

A store is connected to a data source that contains raster or vector data. It can be a file or group of files, a database or a raster file [28]. A store must contain at least one layer. Each store must be associated with a unique workspace [30].

3. Layer

A layer is a collection of geospatial features or a coverage and their associated data such as projection information, bounding box, and styles. Typically a layer contains one type of data (points, lines, polygons, raster). A layer corresponds to a table or view⁸ from a database, or an individual file. Each layer must be associated with a unique workspace.

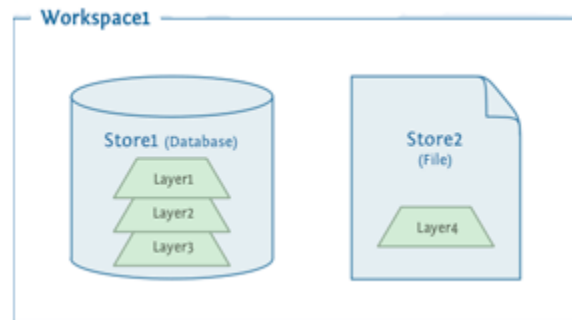


Figure 4.14: Relationships between workspaces, stores and layers

4. Layer Group

A layer group, as its name suggests, is a collection of layers that allows requesting multiple layers with a single WMS request[30].

5. Style

A style in GeoServer contains rules for color, shape, and size, along with logic for styling certain features or points in certain ways based on attributes or scale level [30].

4.5.2.3 GeoServer REST Interface

The main reason behind the use of Geoserver REST API is to automate the repetitive administrative tasks such as the creation of workspaces, data stores, styles and Layers by using GET, POST, PUT, and DELETE operations.

For example to get the workspaces defined in our Geoserver we can use cURL⁹:

```
curl -u admin:geoserver -v -XGET -H 'Accept: application/json'
http://address:8080/geoserver/rest/workspaces
```

4.5.2.4 Web services

As seen previously Geoserver supports the OGC main standards such as WMS, WFS, WCS, WPS.

- WMS

An example of GetMap request that retrieves a map image for a specified area and content.

⁸In a database, a view is the result set of a stored query on the data.

⁹cURL is a computer software project providing a library and command-line tool for transferring data using various protocols[31] from the command line

```
http://localhost:8080/geoserver/topp/wms?service=WMS\
&version=1.1.0&request=GetMap&layers=top:tasmania&styles=&
bbox=145,-43,148,-40&width=512&height=427&srs=EPSG:4326&
format=image/png
```

Where "srs" is the spatial Reference System for map output and format is the format of the map output.

- **WFS**

An example of a GetFeature request that returns a selection of features from a data source including geometry and attribute values.

```
http://localhost:8080/geoserver/wfs?service=wfs&version=1.1.0&
request=GetFeature&typeName=namespace:featuretype
```

- **WPS**

An example of a " DescribeProcess" request that returns an XML document that provides a detailed description of the process JTS:buffer is:

```
http://localhost:8080/geoserver/ows?
service=WPS&
request=DescribeProcess&
identifier=JTS:buffer
```

- **WCS**

An example of "GetCapabilities" request that retrieves a list of the server's data, as well as valid WCS operations and parameters:

```
http://localhost:8080/geoserver/wcs?service=wcs&
AcceptVersions=1.1.0&
request=GetCapabilities
```

4.5.2.5 GeoWebCache

For each GetMap request Geoserver load the data, apply styles, deliver the result in a raster format, and send it to the client. This may be time and resource consuming especially when data are not changing and tiles generated by WMS are always the same. That is where GeoWebCache comes handy.

GeoWebCache is a tiling server that runs as a proxy between a map client and map server, caching (storing) tiles as they are requested, eliminating redundant request processing and thus saving large amounts of processing time.

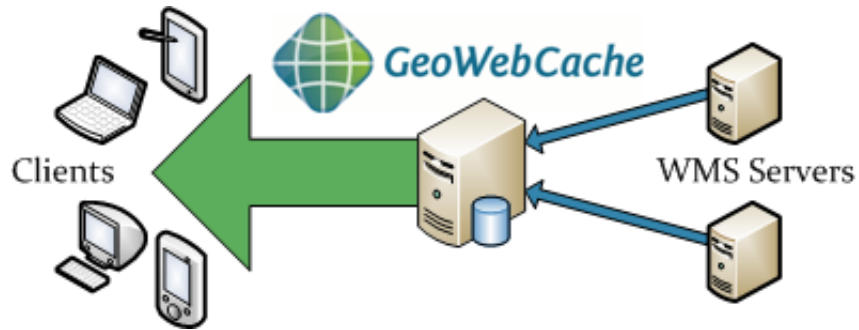


Figure 4.15: GeoWebCache

Source: geowebcache.org

4.6 End-to-end architecture

We will summarize the previous technical choices in the logical architecture 4.16.

4.7 Conclusion

In this chapter, we first presented the CDTA's OpenStack cloud that hosted our work then compared various open source software and picked the most suitable one for each part of our IoT-oriented GIS architecture:

- **IoT platform:** Sentilo.
- **Spatial Database:** PostGIS/PostgreSQL and Greenplum Pivotal.
- **Web Mapping Server:** Geoserver.

In the next chapter, we will present our physical architecture and its implementation, in a production environment.

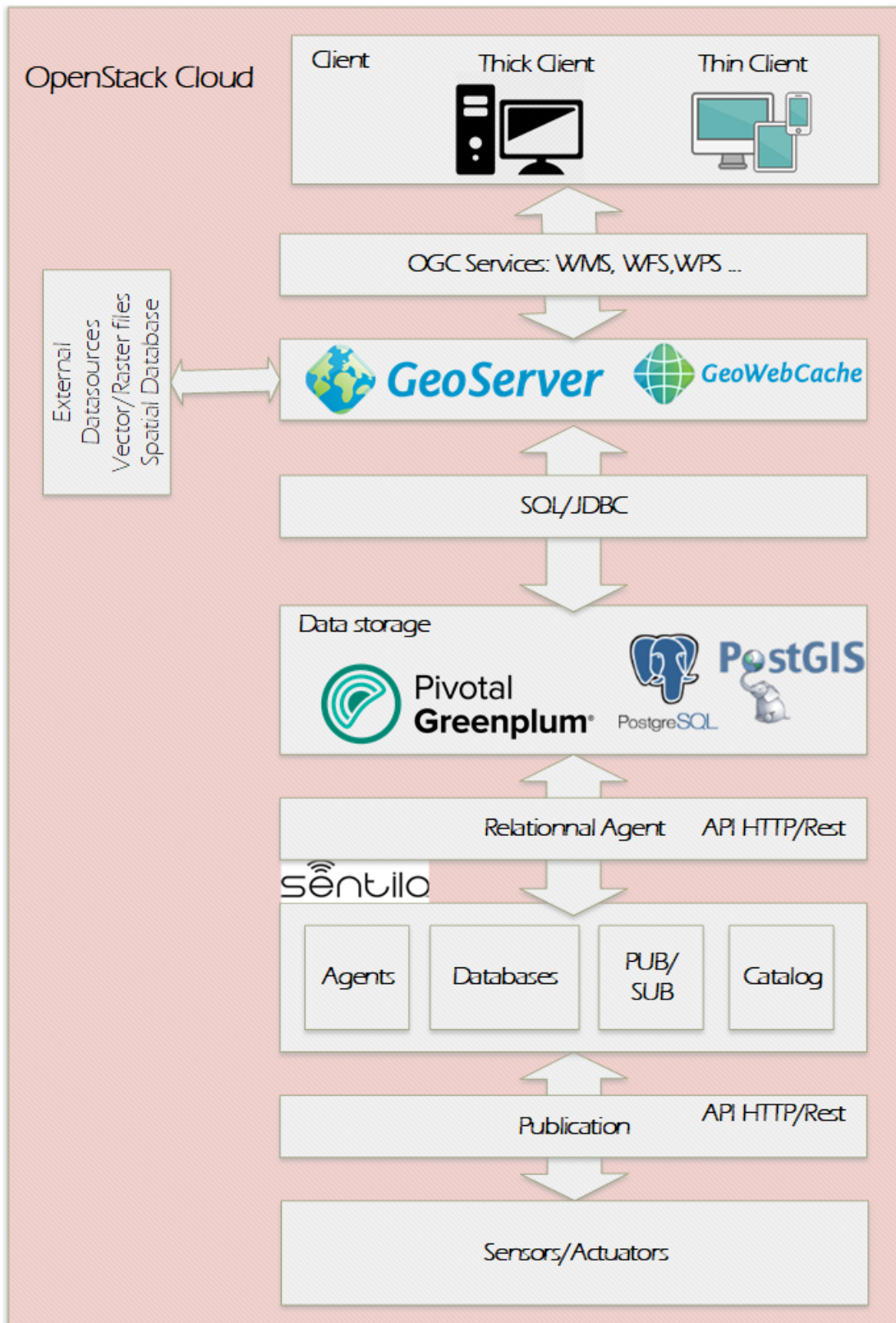


Figure 4.16: Logical end-to-end GIS architecture

Chapter 5

Implementation

5.1 Introduction

In this chapter we will document the implementation in a production environment of our web based IoT oriented GIS according to the technical choices fixed previously.

5.2 Detailed architecture to implement

In order to exploit the sensor data for GIS end user applications, we followed this procedure:

1. We injected Smart city's sensor data in the IoT platform Sentilo using its API REST,
2. Since Sentilo does not store data forever because of limited system resources, we had to copy them in another database using Sentilo's relational agent,
3. We stored the data in the MPP database Greenplum Pivotal where we performed spatial analysis operations to exploit the data,
4. We shared these sensors' data, as well as other spatial data , using Geoserver's web services.

Each of the chosen platform has been installed for a production environment that requires high availability and scalability

1. Sentilo : one instance for test before migrating to the official Sentilo's platform implemented by the "Algiers Smart City" project's team.
2. Greenplum Pivotal: one master and three segments,
3. Geoserver: four instances (distributed on two physical machines, two in each machine) supervised by a load balancer,

The figure 5.1 summarize our physical architecture.

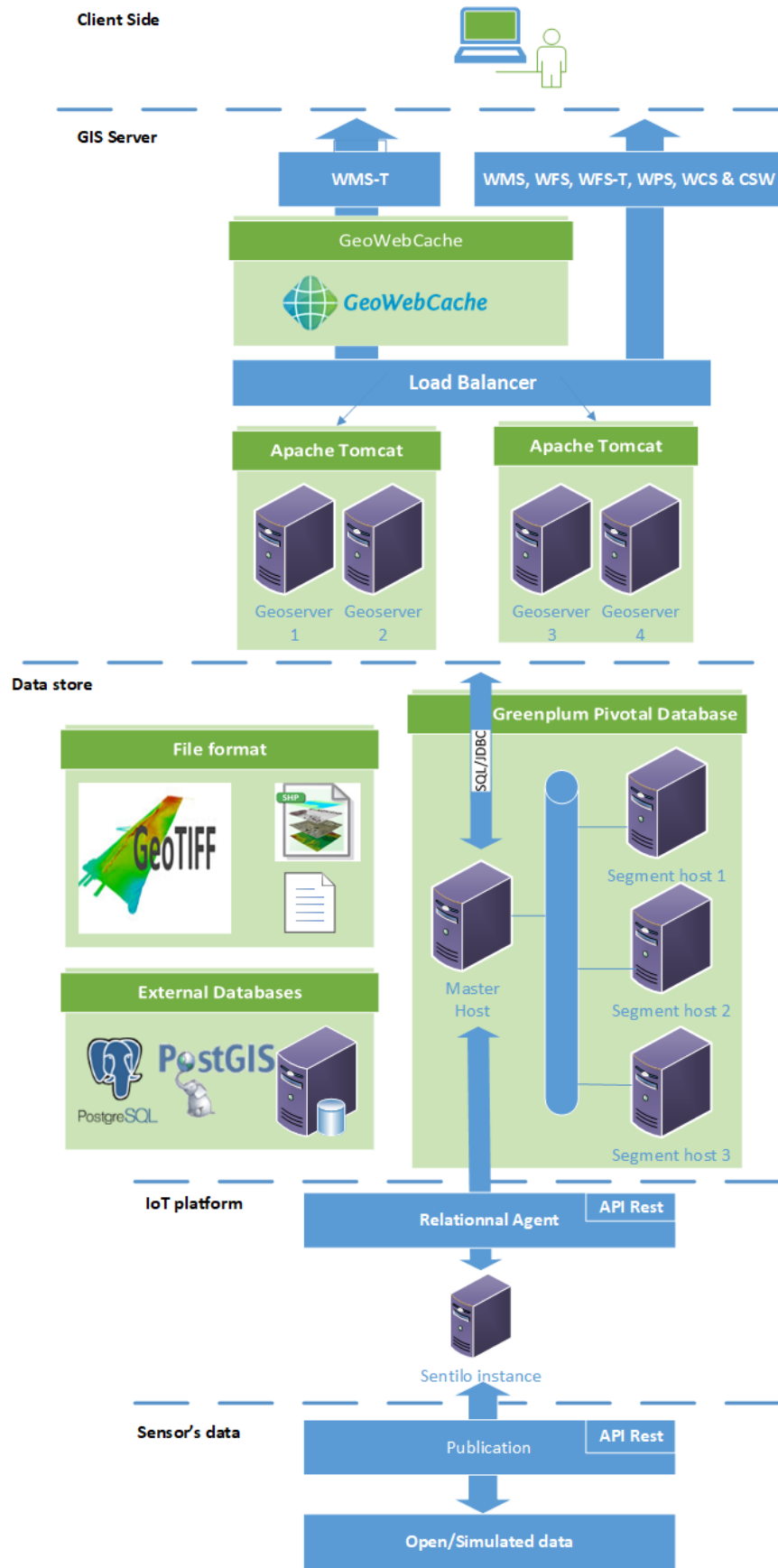


Figure 5.1: Detailed architecture to implement

5.3 Sentilo

5.3.1 Data

Since sensors and actuators are not deployed yet in the Wilaya of Algiers, which means there is no available data, we used Open and simulated Data in order to test our architecture.

5.3.1.1 OpenData

Open data is a data that can be freely used, shared and built-on by anyone, anywhere, for any purpose [32]. There is open information in transport, science, products, education, sustainability, maps, etc.

We used sensors data published by PARISDATA for temperature and pressure. But considering that Sentilo's API Rest requires a specific format of data, we had to modify it.

- **Preparation of the data**

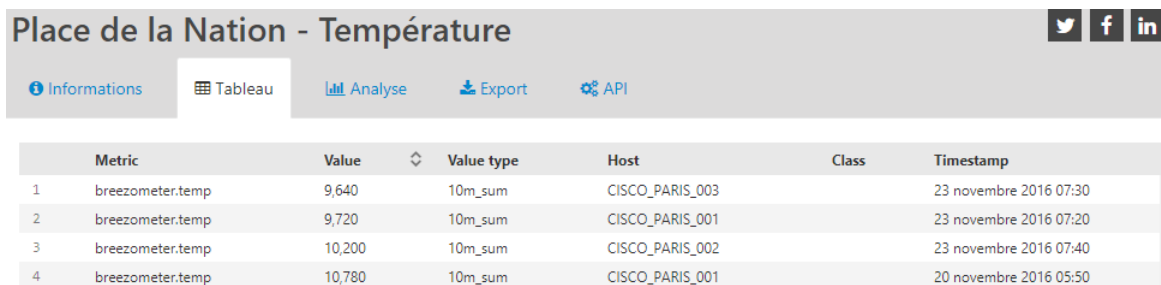
First, we prepared the data by:

- Putting it in the format below:

```
{ "observations": [{
  "value": "9.6", "timestamp": "17/02/2016T11:43:45C ET",
  "location": "41.3888 2.15899"}
]}
```

- Linking the sensors to positions in Algiers.

The figure 5.2 shows the original data's format:



	Metric	Value	Value type	Host	Class	Timestamp
1	breezometer.temp	9,640	10m_sum	CISCO_PARIS_003		23 novembre 2016 07:30
2	breezometer.temp	9,720	10m_sum	CISCO_PARIS_001		23 novembre 2016 07:20
3	breezometer.temp	10,200	10m_sum	CISCO_PARIS_002		23 novembre 2016 07:40
4	breezometer.temp	10,780	10m_sum	CISCO_PARIS_001		20 novembre 2016 05:50

Figure 5.2: Temperature table's format

Followed steps:

We will present the followed steps for temperature data but the same ones are used for pressure data. We used the data of eight sensors in Paris.

1. Download the csv file and delete the empty column "Class":

```
wget https://opendata.paris.fr/explore/dataset/place-de-la-nation-temperature/download/?format=csv
vi temperature.csv
:g/;;/s//;/g
```

2. Import csv file into PostgreSQL table to prepare it.

- (a) Create a new user and a database:

```
createuser opendata_user
createdb -e opendata -O opendata_user
```

- (b) Create a new table "temperature" with the open data's format.

```
psql -h localhost -U opendata_user opendata
```

```
CREATE TABLE temperature (
metric character varying(50),
value float,
value_type character varying(50),
host character varying(50),
timestamp date);
```

- (c) Copy the data into the table created

```
COPY temperature FROM '/home/gpadmin/temperature.csv'
DELIMITER ';' CSV HEADER;
```

3. Create a table named "position" with two columns "host" and "location" and fill it with the name of the eight sensors and positions in Algiers:

```
CREATE TABLE position (host character varying(50),
location text);
INSERT INTO position(host, location)
VALUES
('CISCO_CT_PARIS_257', '36.60507 3.11'),
('CISCO_CT_PARIS_258', '36.66138 2.88254'),
('CISCO_PARIS_005', '36.8678 2.77954'),
('CISCO_PARIS_004', '36.64078 2.93541'),
('CISCO_CT_PARIS_260', '36.69914 2.9512'),
('CISCO_PARIS_003', '36.58791 2.95052'),
('CISCO_PARIS_002', '36.714170 3.021964'),
('CISCO_PARIS_001', '36.788722 3.015700'));
```

4. Join the "temperature" table with "position" table, modify its format so it matched that of Sentilo and copy the table into a json file:

```
CREATE TABLE temperature001 AS (
SELECT CAST(j.value AS varchar(10)) AS value,
TO_CHAR(j.timestamp, 'dd/mm/yyyyTfmhh:mi:ss') AS timestamp,
l.location AS location FROM temperature AS j,
position AS l WHERE j.host=l.host AND
j.host='CISCO_PARIS_001' );
COPY (SELECT row_to_json(t) || ',' FROM temperature001 as t ) to
'/home/ubuntu/temperature001.json';
```

Script explanation

- Convert value to a *char* using CAST function,
- Change the format of data to dd/mm/yyyyTfmhh:mi:ss to match Sentilo's one,

- Get the location by joining the two tables using the relationship established by "host",
 - Repeat for all the sensors.
5. Add a '{"observations":[' at the beginning of the json file and '']}' at the end. We get files in the format of the figure 5.3, which can be now used with Sentilo's API Rest:

```

{"observations":[
{"value":"11.4799995","timestamp":"21/11/2016T12:00:00","location":"36.788722 3.015700"},
{"value":"10.1999998","timestamp":"22/11/2016T12:00:00","location":"36.788722 3.015700"},
{"value":"10.7700004","timestamp":"22/11/2016T12:00:00","location":"36.788722 3.015700"},
{"value":"8.52999973","timestamp":"14/11/2016T12:00:00","location":"36.788722 3.015700"}
]

```

Figure 5.3: A caption of the temperture001.json file

• Publish the data

Now that the data are ready, we published it as follow:

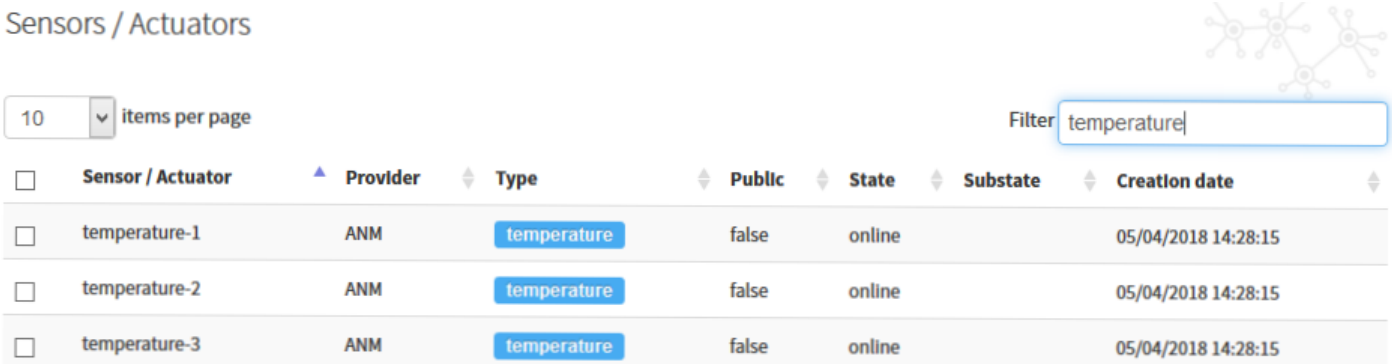
1. Create a provider as described in Sentilo's website,
2. Create the sensors by running the python program in the annex C.

Script explanation:

The code takes advantage of Sentilo REST API by defining, using a loop, a JSON file that holds the sensors' configurations, and then send a POST request to create them.

3. The creation of the sensors can be verified in Sentilo website as shown in the figure 5.4 :

Sensors / Actuators



<input type="checkbox"/>	Sensor / Actuator	Provider	Type	Public	State	Substate	Creation date
<input type="checkbox"/>	temperature-1	ANM	temperature	false	online		05/04/2018 14:28:15
<input type="checkbox"/>	temperature-2	ANM	temperature	false	online		05/04/2018 14:28:15
<input type="checkbox"/>	temperature-3	ANM	temperature	false	online		05/04/2018 14:28:15

Figure 5.4: Temperature Sensors

4. Publish the data by running this command from the bash:

```

curl -X PUT -H "IDENTITY_KEY:Identity_key"
-H "Content-Type: application/json"
-d @temperature001.json
http://ip-address:8081/data/ANM/temperature-1

```

The figure 5.5 shows that the data have been correctly published:

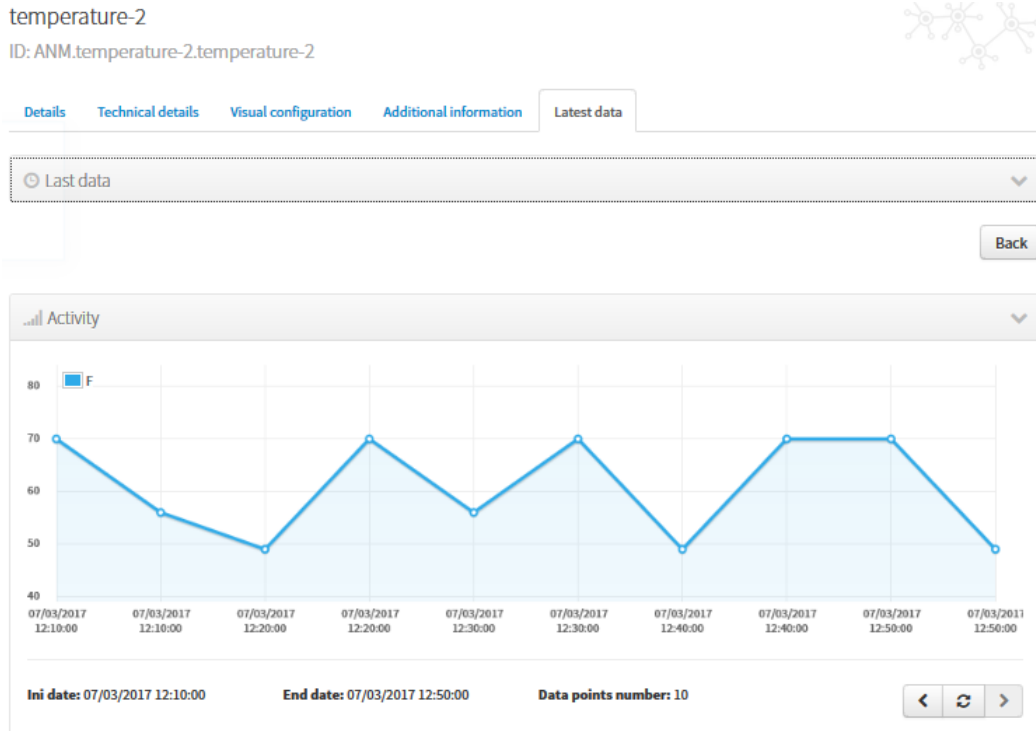


Figure 5.5: Visualization of temperature data

5.3.1.2 Simulated Data

Some Open data were not available, so we decided to simulate our own data of trash cans and public bicycle sharing stations (*vélib*s) to emulate a realistic smart city scenario.

To do so we wrote a Python program that:

1. Generates random time, location and sensor's data using Python's function "`np.random.uniform`":

```
np.random.=np.random.uniform(low=[lower_limit],
high=[upper_limit],size=[size_of_vectordata])
```

2. Creates, using a loop, the sensors in Sentilo and publish the data by taking advantage of Sentilo's API REST syntax as done previously.

The complete code is attached in the annex C. By running the program we can see in the figure 5.6 that the data have been received

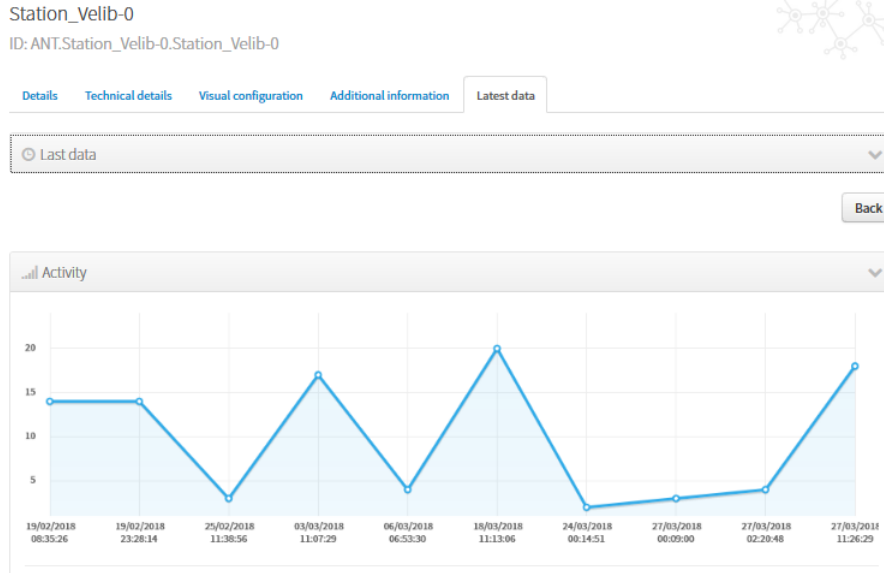


Figure 5.6: Visualization of the velib data

The figure 5.7 shows different sensors deployed in the Wilaya of Algiers.

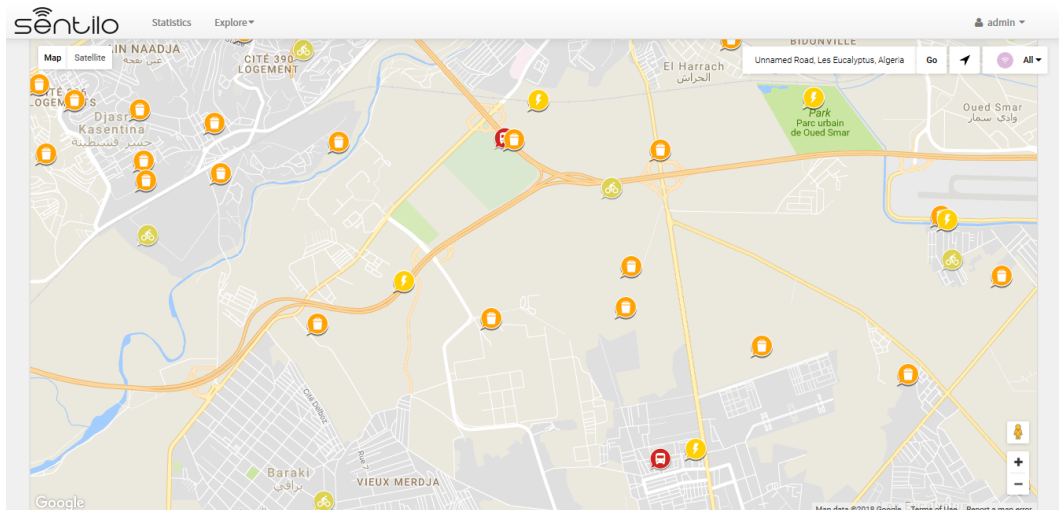


Figure 5.7: Universal Viewer

5.3.2 Sentilo relational agent

Sentilo does not store data forever because of limited system resources so we have to transfer it regularly in another database. As defended in the previous chapter, we opted for Greenplum database and connected it with Sentilo using the relational agent.

However, by default, this agent is configured to send data to MySQL database. In the following, we will explain the configurations performed in order to connect Sentilo with Greenplum.

1. The relational agent's POM¹ contains dependency for MySQL connector, so we replaced

¹POM is an acronym for Project Object Model. file, The pom.xml file contains information of project and configuration information for the maven to build the project such as dependencies, build directory, source directory, test source directory, plugin, goals etc.[33]

it by PostgreSQL's connector:

```
cd sentilo/sentilo-agent-relational
vi pom.xml
```

Commented the dependency of MySQL and added the code below:

```
<dependency>
  <groupId>org.postgresql</groupId>
  <artifactId>postgresql</artifactId>
  <version>42.2.1</version>
</dependency>
```

2. Next, we specified the properties of our database: ip-address, port, user, password, database's name.

```
vi src/main/resources/properties/relational-client-config.properties

sentiloDs.jdbc.driverClassName=org.postgresql.Driver
sentiloDs.url=jdbc:postgresql://ip-address:5432/sentilo
sentiloDs.username=sentilo_user
sentiloDs.password=sentilo_pwd
sentiloDs.validationQuery= Select 1
```

3. We generated an "appassembler" directory inside ./sentilo-agent-relational/target/ :

```
mvn clean install
mvn package appassembler:assemble -P dev
```

4. The "appassembler" folder contains two sub directories "repo" and "bin" [?]:

- **repo**: directory contains all libraries needed to run the process.
- **bin**: directory contains the script (sentilo-agent-relational) needed to initialize the process (there are two scripts, one for Linux systems and one for Windows).

5. We moved this two files in the root directory, and changed the permission of the script:

```
mv target/appassembler/* /opt/sentilo-agent-relational/
chmod 555 /opt/sentilo-agent-relational/bin/*
```

6. we run the script in order to send the data regularly to the corresponding database.

```
/opt/sentilo-agent-relational/bin/sentilo-agent-relational-server
```

5.4 Greenplum database

We managed to install Greenplum database on one master and three segment hosts as in the figure 5.8 (The complete installation process is found in the annex A).

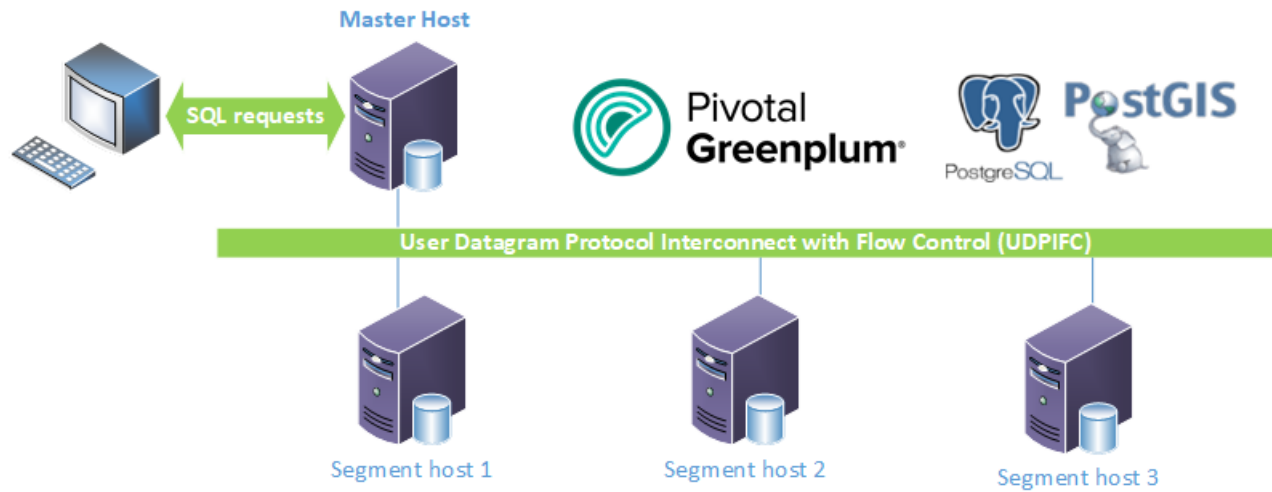


Figure 5.8: Greenplum database architecture

Once we had Greenplum installed, the next step was to create the database that holds Sentilo's data and connect it with Sentilo via the relational agent then adapt the table to GeoServer.

5.4.1 Greenplum and Sentilo interconnection

1. Create a new user and database

```
sudo -u postgres createuser -P sentilo_user
sudo -u postgres createdb -O sentilo_user sentilo
psql -h localhost -U sentilo_user sentilo
```

2. Create the table:

```
CREATE SEQUENCE sentilo_observations_seq;
CREATE TABLE sentilo_observations (
  id INT NOT NULL DEFAULT NEXTVAL ('sentilo_observations_seq'),
  provider VARCHAR(128) NOT NULL,
  sensor VARCHAR(128),
  value VARCHAR(512) NOT NULL,
  timestamp varchar(20) NOT NULL,
  event_timestamp timestamp(0) NOT NULL,
  published_at timestamp(0) NOT NULL,
  publisher VARCHAR(128),
  location varchar(50),
  PRIMARY KEY(id));
```

```
mysql> describe sentilo_observations;
```

Field	Type	Null	Key	Default	Extra
id	int(11)	NO	PRI	NULL	auto_increment
provider	varchar(128)	NO		NULL	
sensor	varchar(128)	YES		NULL	
value	varchar(512)	NO		NULL	
timestamp	varchar(20)	NO		NULL	
event_timestamp	timestamp	NO		CURRENT_TIMESTAMP	on update CURRENT_TIMESTAMP
published_at	timestamp	NO		0000-00-00 00:00:00	
publisher	varchar(128)	YES		NULL	
location	varchar(50)	YES		NULL	

9 rows in set (0.00 sec)

Figure 5.9: Created table

3. Start the sentilo's relational agent

Now, that the table is created, we run the sentilo's relational agent to start the connection. As we can see in the figure 5.10 the observations have been sent:

```
sentilo=# select * from sentilo_observations where provider='ANT';
```

id	provider	sensor	value	timestamp	event_timestamp	published_at	publisher	location
35	ANT	Station_velib-0	20	18/03/2018T11:13:06	2018-03-18 11:13:06	2018-04-12 11:32:19	ANT	36.6710045958 3.02001767242
49	ANT	Station_velib-1	5	24/03/2018T00:14:51	2018-03-24 00:14:51	2018-04-12 11:32:19	ANT	36.6504563383 3.15721488166
52	ANT	Station_velib-1	3	18/03/2018T11:13:06	2018-03-18 11:13:06	2018-04-12 11:32:19	ANT	36.6504563383 3.15721488166
63	ANT	Station_velib-2	8	06/03/2018T06:53:30	2018-03-06 06:53:30	2018-04-12 11:32:19	ANT	36.7343528809 2.77492933289
68	ANT	Station_velib-2	2	06/01/2018T00:36:03	2018-01-06 00:36:03	2018-04-12 11:32:19	ANT	36.7343528809 2.77492933289
71	ANT	Station_velib-2	17	08/02/2018T08:31:21	2018-02-08 08:31:21	2018-04-12 11:32:19	ANT	36.7343528809 2.77492933289
85	ANT	Station_velib-3	9	27/03/2018T00:09:00	2018-03-27 00:09:00	2018-04-12 11:32:19	ANT	36.6457068287 2.85933007962
104	ANT	Station_velib-5	1	24/03/2018T00:14:51	2018-03-24 00:14:51	2018-04-12 11:32:19	ANT	36.6272911255 2.76517614962
118	ANT	Station_velib-5	19	05/01/2018T13:18:18	2018-01-05 13:18:18	2018-04-12 11:32:19	ANT	36.6272911255 2.76517614962
121	ANT	Station_velib-6	0	27/03/2018T02:20:48	2018-03-27 02:20:48	2018-04-12 11:32:19	ANT	36.6313453531 3.05223092874

Figure 5.10: Data transfer from Sentilo to Greenplum

However, at this point our data does not have support for spatial operations, thus it cannot be exploited by a GIS server, because the table sent by Sentilo had no "geometry"² column. As a solution to this issue, we managed to:

1. Create a geometry column named "geom" and then create a spatial index using PostGIS spatial function:

- (a) AddGeometryColumn: Adds a geometry column to an existing table of attributes: text AddGeometryColumn(varchar table_name, varchar column_name, integer srid³, varchar type, integer dimension);

```
SELECT AddGeometryColumn(sentilo_observations, geom,
4326, POINT, 2);
```

- (b) CREATE INDEX: build a spatial index on a table with a geometry column.


```
CREATE INDEX [indexname] ON [tablename] USING GIST ( [geometrycolumn]
);
```

²The geometry column is in the standard format: Well-Known Binary (WKB). It includes information about the type of the object and the coordinates which form the object.

³A spatial reference identifier (SRID) is a unique identifier associated with a specific coordinate system, tolerance, and resolution. There are various recognized standard SRIDs, such as those defined by EPSG.


```
CREATE INDEX index_gis ON sentilo_observations USING
GIST (geom);
```

2. Create a rule that updates the table at each new insert event (new data), transforms the char type of the "location" column into a geometry type and puts the result in the "geom" column. We used Greenplum and PostGIS functions:

(a) CREATE RULE: defines a new rule applying to a specified table or view: CREATE RULE [rule_name] AS ON [event] TO [table_name] DO [ALSO | INSTEAD] command

(b) UPDATE: updates rows of a table:
UPDATE [table_name] SET [column_name]=command

(c) ST_PointFromText — Makes a point Geometry from a text:
geometry ST_PointFromText(text TEXT);

(d) split_part: Split string on delimiter and return the given field (counting from one)
split_part(string text, delimiter text, field int)

```
CREATE RULE test as on INSERT to sentilo_observations do also
UPDATE sentilo_observations
SET geom= (CASE WHEN (location IS NOT NULL)
THEN ST_PointFromText('POINT(' || split_part(location, ' ',
2) || ' ' ||
split_part(location, ' ', 1) || ')', 4326)
ELSE geom end);
```

The figures 5.11 and 5.12 show that the SQL script did the job as we were expecting by creating geometric objects in the geometry column "geom".

Column	Type	
id	integer	not null
provider	character varying(128)	not null
sensor	character varying(128)	
value	character varying(512)	not null
timestamp	character varying(20)	not null
event_timestamp	timestamp(0) without time zone	not null
published_at	timestamp(0) without time zone	not null
publisher	character varying(128)	
location	character varying(50)	
geom	geometry	

Indexes:

- "sentilo_observations_pkey" PRIMARY KEY, btree (id)
- "index_gis" gist (geom)

Figure 5.11: "Sentilo_observations" table's meta data

location		geom
36.6710045958	3.02001767242	0101000020E610000068168306FF2808409E
36.6504563383	3.15721488166	0101000020E6100000C86439E0F9410940BA
36.6504563383	3.15721488166	0101000020E6100000C86439E0F9410940BA
36.7343528809	2.77492933289	0101000020E610000008CA6B260E3306402D
36.7343528809	2.77492933289	0101000020E610000008CA6B260E3306402D
36.7343528809	2.77492933289	0101000020E610000008CA6B260E3306402D
36.6457068287	2.85933007962	0101000020E6100000ED7EE372E8DF064002
36.6272911255	2.76517614962	0101000020E6100000776052AC141F0640F9

Figure 5.12: "Sentilo_observations" geometry column

Now the data are ready to be exploitable for GIS purposes such as:

1. Spatial data analysis and machine learning using MADlib,
2. OGC Services through a GIS Server such as GeoServer.

5.5 Geoserver

In this section, we will explain the steps we followed to implement Geoserver in a production environment. We will present

1. The publication of our sensor's data coupled with vector and raster backgrounds,
2. The development of the architecture in a production environment to provide Scalability, Security and High availability.
3. The integration of GeoWebCache in a cluster architecture.

5.5.1 Publish the data

An IoT-oriented GIS solution needs to deliver efficiently both vector and raster data in addition to real time sensor's data.

Our database will be built around the following repositories:

- A vector digital map background,
- A raster background: satellite images,
- Greenplum sensor's data.

5.5.1.1 Vector digital map background

Accessing government data is expensive and complicated, so we used the OpenStreetMap's (OSM) data for testing our architecture.

OpenStreetMap is a free, editable map of the whole world that is being built by volunteers and released with an open-content [34]. Often considered as the geographic analog of Wikipedia. The real power of OSM is that it gives access to the data rather than just rendering the map as in the case of Google-Map.

In this part we will explain the followed steps to publish OSM data in Geoserver. OSM is made up of polygons, lines and points so we used PostgreSQL database with POSTGIS extension to publish it.

1. First we got the OSM data of Algeria in the osm.pbf⁴

```
wget http://download.openstreetmap.fr/extracts/africa/algeria
-latest.osm.pbf
```

2. Next, we created a new database and added the postgis extension.

```
sudo -u postgres createdb -O postgres osm
sudo -u postgres psql -c "CREATE EXTENSION postgis;
CREATE EXTENSION postgis_topology;" osm
```

3. We loaded the file into the osm database using osm2pgsql command where default.style is the default osm2pgsql .style.

```
osm2pgsql -c -d osm -U postgres -W -H
localhost -S default.style algeria-latest.osm.pbf
```

This resulted in four tables:

public	planet_osm_line	table	postgres
public	planet_osm_point	table	postgres
public	planet_osm_polygon	table	postgres
public	planet_osm_roads	table	postgres

Figure 5.13: OpenStreetMap tables

4. In order to have a comprehensible map, we created “layer-like” tables using the "createDBobjects" code published by Boundless.

```
wget https://github.com/boundlessgeo/OSM/blob/master/create
DBobjects.sql
psql -h localhost -U postgres -d osm -a -f createdBObjects.sql
```

⁴PBF Format ("Protocolbuffer Binary Format") is a highly compressed, optimized binary alternative to the XML format [35].

⁴osm2pgsql is a command-line based program that converts OpenStreetMap data to postGIS-enabled PostgreSQL databases[36]

List of relations			
Schema	Name	Type	Owner
public	aero-poly	table	postgres
public	agriculture	table	postgres
public	amenity-areas	table	postgres
public	beach	table	postgres
public	building	table	postgres
public	forest	table	postgres
public	geography_columns	view	postgres
public	geometry_columns	view	postgres
public	grass	table	postgres
public	highway-label	table	postgres
public	park	table	postgres
public	parking-area	table	postgres
public	placenames-medium	table	postgres
public	planet_osm_line	table	postgres
public	planet_osm_point	table	postgres
public	planet_osm_polygon	table	postgres
public	planet_osm_roads	table	postgres
public	raster_columns	view	postgres
public	raster_overviews	view	postgres
public	route-bridge-0	table	postgres
public	route-bridge-1	table	postgres
public	route-bridge-2	table	postgres
public	route-bridge-3	table	postgres
public	route-bridge-4	table	postgres
public	route-bridge-5	table	postgres
public	route-fill	table	postgres
public	route-line	table	postgres
public	route-tunnels	table	postgres
public	route-turning-circles	table	postgres
public	spatial_ref_sys	table	postgres
public	water	table	postgres
public	water-outline	table	postgres
public	wetland	table	postgres
topology	layer	table	postgres
topology	topology	table	postgres
topology	topology_id_seq	sequence	postgres

(36 rows)

Figure 5.14: Layer-like OpenStreetMap tables

5. Next, we configured Geoserver to connect and exploit the database:

(a) Create a new workspace as described in figure 5.15:

New Workspace
Configure a new workspace

Name:

Namespace URI:
The namespace uri associated with this workspace

Default Workspace

Isolated Workspace

Figure 5.15: Creation of a new workspace

(b) Create a new store as described in figure 5.16:

New Vector Data Source

Add a new vector data source

PostGIS
PostGIS Database

Basic Store Info

Workspace *

OSM

Data Source Name *

osm_store

Description

Contains OSM data

Enabled

Connection Parameters

host *

postgres IP

port *

5432

database

osm

schema

public

user *

postgres

passwd

.....

Figure 5.16: Creation of a new store

6. We created styles using the CSS extension⁵. The code below shows an example of the Grass layer's style:

```
* {  
  fill: #bbf9ca;  
  fill-opacity: 50%;  
  stroke: #7cabf9;  
  stroke-width: 0.5;  
  stroke-opacity: 50%;  
}
```

7. We published the data and linked each table with its corresponding style.

OSM:aero-poly

Configure the resource and publishing information for the current layer

Data Publishing Dimensions Tile Caching

Edit Layer

Basic Resource Info

Name

aero-poly

Enabled

Advertised

Title

aero-poly

Figure 5.17: Creation of aero-poly layer

⁵The installation of the extension is explained in the annex A

8. To get the map, we created a layer group that contains all the layers:

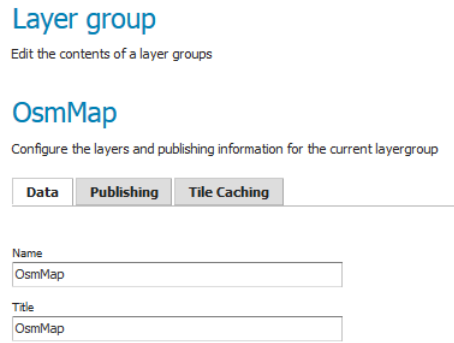


Figure 5.18: Creation of OsmMap layer group

9. We can visualize the map from layer preview section:

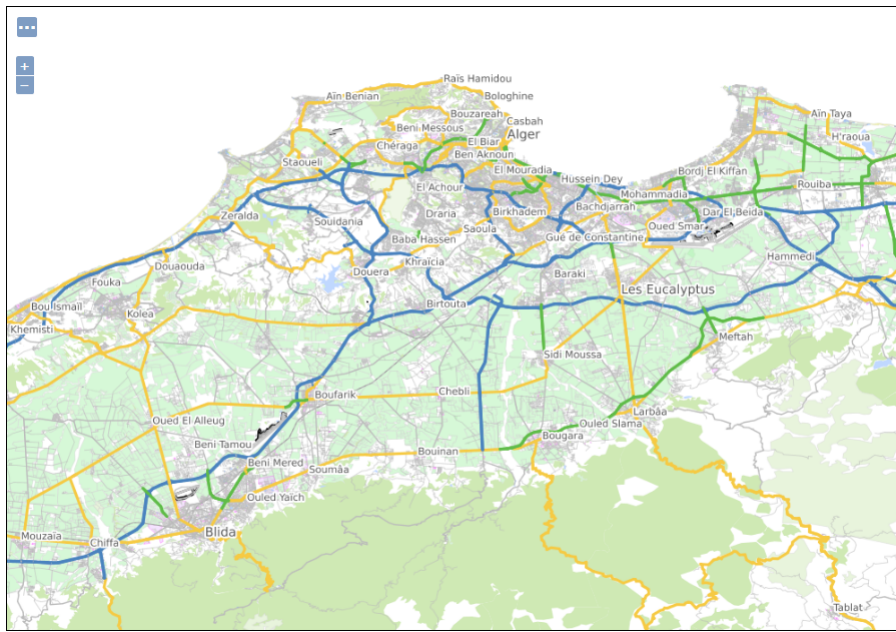


Figure 5.19: Preview of OpenStreetMap data

5.5.1.2 Raster data

Raster data of Algiers are not available for free, however in our researches we came across Global Mapper software, defined as "An affordable and easy-to-use GIS application that offers access to an unparalleled variety of spatial data sets and provides just the right level of functionality to satisfy both experienced GIS professionals and beginning users."

By using the free trial we were able to download some raster images for testing with the resolution of 1:500 000.

1. Prepare the data

Structuring and organizing the data can make a huge difference on the response times of a mapping server. So in this part, we will explain the steps we went through to prepare satellite imagery using GDAL library ⁶[37].

⁶GDAL is an open source library designed to process and transform raster data.

- We added overviews⁷ to our GeoTIFF file.

```
gdaladdo -r nearest -b 1 algiers.tiff 2 4 8 16
```

- We wrote the image data in uncompressed tiled structure so that it can be extracted easily.

```
gdal_translate -of GTiff -a_srs EPSG:4326 -co "COMPRESS=NONE"
"TILED=YES" algiers.tif algiers_tiled.tif
```

2. Publish the data

Publishing raster data in Geoserver is a straight forward task, we just created a new store and added the corresponding file.

Figure 5.20: Adding a new store

The data can be visualized from the layer preview:



Figure 5.21: Satellite imagery

⁷Overviews are different zoom's levels of the complete image.

⁷Tiles arrange the image data in blocks.

5.5.1.3 Greenplum sensor's data

We needed also to connect GeoServer with the Greenplum database to share sensor's data. As we have seen before, the master is the entry point to the Greenplum Database system, so GeoServer needs to connect to the Master host in the port 5432.

Connect to the database

To do that we had to:

1. Create a new "Store" in GeoServer and enter the required parameters about the Greenplum database (Master's host IP address, Database,...) as described previously in the figure 5.16:
2. Create a new layer with the following parameters:
 - (a) Name title: greenplum_sentilo_observations
 - (b) Native SRS: EPSG:4326 which correspond to WGS84 datum.
 - (c) Native Bounding Box: Compute from data.
 - (d) Lat/Lon Bounding Box: Compute from native bounds.

The data can be visualized from the layer preview as shown in the figure 5.22

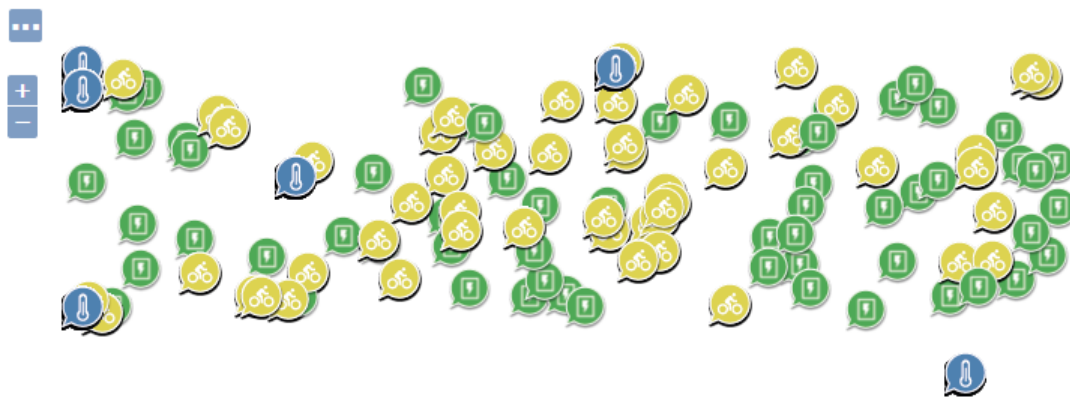


Figure 5.22: Preview of Greenplum data

By clicking on a sensor, we can see a table of all its data as shown in the figure 5.23:

greenplum_sentilo_observations									
fid	provider	sensor	value	timestamp	event_timestamp	published_at	publisher	lo	
greenplum_sentilo_observations.41328	ANM	temperature-1	43	23/12/2016T10:40:00	Dec 23, 2016 10:40:00 AM	Apr 12, 2018 11:38:53 AM	ANM	36	
greenplum_sentilo_observations.41331	ANM	temperature-1	55	10/01/2017T06:40:00	Jan 10, 2017 6:40:00 AM	Apr 12, 2018 11:38:53 AM	ANM	36	
greenplum_sentilo_observations.41332	ANM	temperature-1	79	10/01/2017T06:30:00	Jan 10, 2017 6:30:00 AM	Apr 12, 2018 11:38:53 AM	ANM	36	
greenplum_sentilo_observations.41339	ANM	temperature-1	53	11/01/2017T12:20:00	Jan 11, 2017 12:20:00 PM	Apr 12, 2018 11:38:53 AM	ANM	36	
greenplum sentilo observations.41342	ANM	temperature-1	70	11/01/2017T01:40:00	Jan 11, 2017 1:40:00 AM	Apr 12, 2018 11:38:53 AM	ANM	36	▼

Figure 5.23: Greenplum table preview

So the data feedback worked perfectly. However all the data is mixed up, also each time a request is made all Greenplum database's content is loaded, which over time can represent a huge amount of data, and slows down the time response of the server. As a solution we used GeoServer's option "SQL Views"⁸to :

1. Separate each sensor,
2. Upload only the last ten observations for each sensor. Ten was a personal choice, it may vary as needed.
3. Create a layer for each family of sensor,

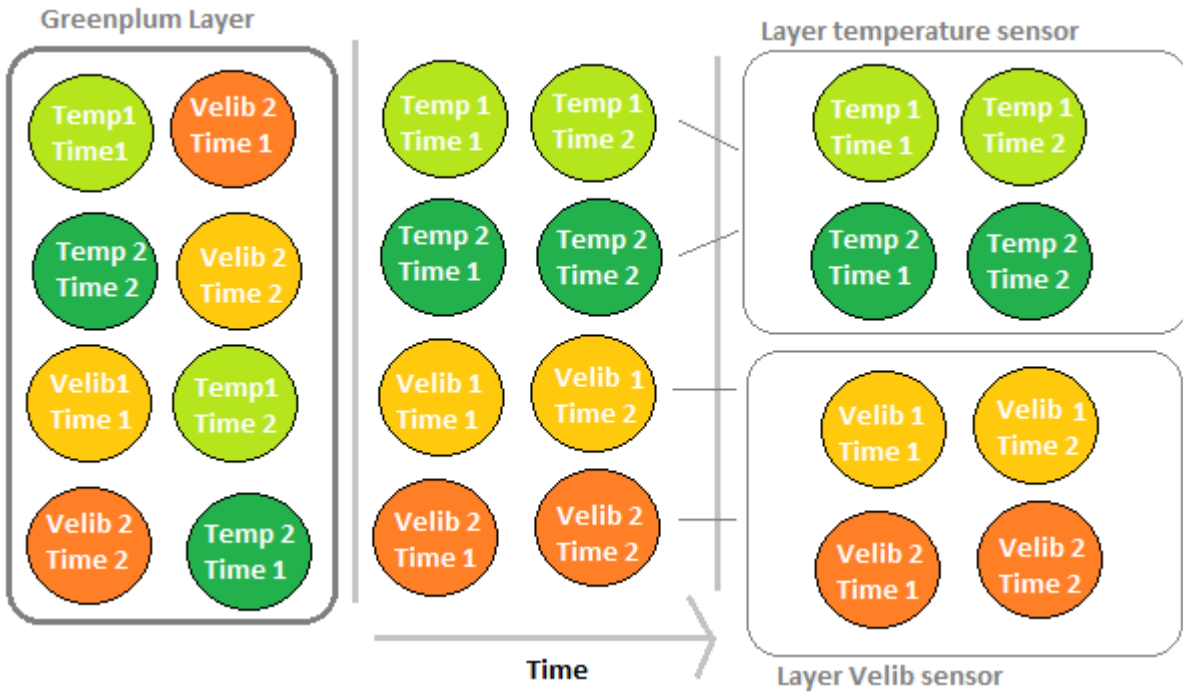


Figure 5.24: Greenplum Data workflow

SQL Script:

```
SELECT *
FROM (
  SELECT
    ROW_NUMBER() OVER (PARTITION BY sensor ORDER BY timestamp DESC)
    AS r,
    t.*
  FROM
    sentilo_observations t
  where t.sensor LIKE '%temperature-%') x
WHERE
  x.r <= 10
```

⁸SQL Views allow executing a stored SQL query on each request to the layer.

At the end, we will have sorted and recent data.

Couches

Gérer les couches publiées via GeoServer

-  Ajouter une nouvelle ressource
-  Retirer les ressources sélectionnées

<< < 1 > >> Résultats 1 à 10 (sur les 10 correspondances des 26 articles)

<input type="checkbox"/>	Type	Title	Nom de la couche	Entrepôt	Activée ?	SRC natif
<input type="checkbox"/>	•	Capteurs de pression	algeria:Capteurs de pression	Greenplum_sentilo	✓	EPSG:4326
<input type="checkbox"/>	•	Capteurs de température	algeria:Capteurs de température	Greenplum_sentilo	✓	EPSG:4326
<input type="checkbox"/>	•	Stations vélib	algeria:Stations vélib	Greenplum_sentilo	✓	EPSG:4326
<input type="checkbox"/>	•	Travaux actuels	algeria:Travaux actuels	Greenplum_sentilo	✓	EPSG:4326
<input type="checkbox"/>	•	Velib_data	algeria:Velib_data	Greenplum_sentilo	✓	EPSG:4326

Figure 5.25: Sorted data

5.5.2 Geoserver in Production

Now that our data is available and ready to be used, we need to provide a GIS Server that couples high-availability and reliability. In other word the system needs to perform with minimum down-time. For example if one instance fails, a fail over instance should be provided while fixing the problem using a watchdog.

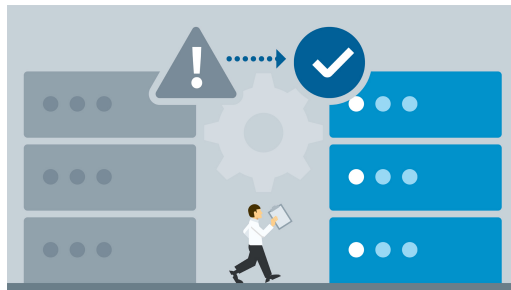


Figure 5.26: High availability

Source: kb.vmware.com

Strategies for achieving high availability include [30]:

- Balancing processing load across multiple servers: clustering and load-balancing,
- Monitoring the server and removing single points of failure: watchdog.

5.5.3 Clustering

5.5.3.1 Definitions

A cluster, in the context of servers, is a group of computers that are connected with each other and operate closely to act as a single computer.[38]

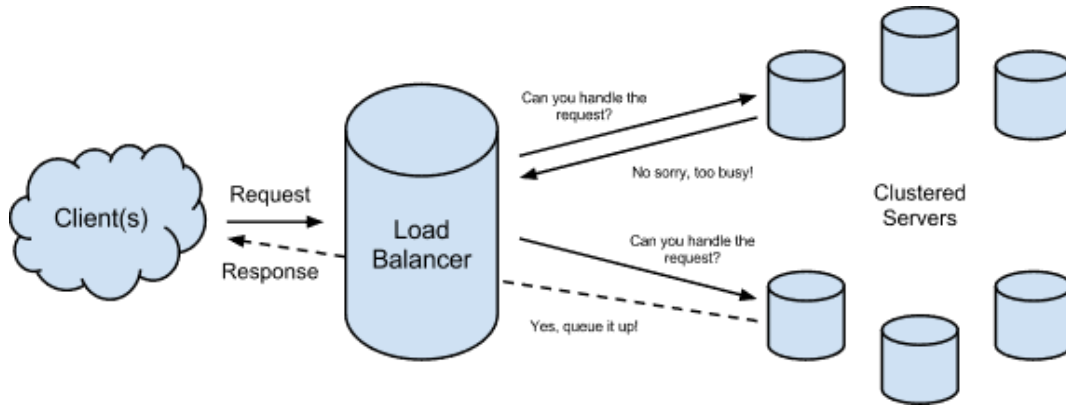


Figure 5.27: Cluster architecture

Source: boundlessgeo.com

To gain in performance we need to scale Geoserver both vertically (Scale Up) and horizontally (Scale Out).

1. **Scaling horizontally** is adding an additional node to the environment to increase its capacity.
2. **Scaling vertically** is running more than one instance of Geoserver on the same machine to maximize the use of the resources on each node.

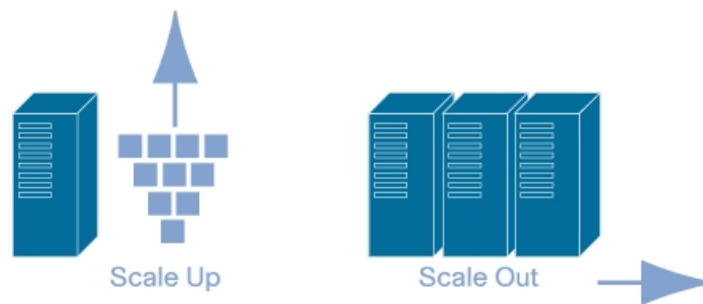


Figure 5.28: Scale up vs scale out

Source: <http://www.pc-freak.net>

5.5.3.2 Implementation

Before implementing a cluster of Geoserver we need to take into consideration some facts about the internal work of Geoserver.

1. By default, the internal configuration of Geoserver (List of workspaces, layers, ...) is contained in XML files inside the data directory.
2. A cluster of Geoserver needs to share the same data directory.
3. When we add, change or delete a workspace, a Store, a layer, .. through the Web GUI or the REST API, it is written in the configuration files.
4. The configuration are loaded into memory at start up.

5. Changes in this files are not token into consideration by other GeoServer's instances unless the reload catalog operation is called or the instance is restarted.

Reloading the catalog in all the instances manually can be overwhelming, fortunately Geoserver provides a **clustering extension**, based upon a Message Oriented Middleware⁹ (MOM) to automate the reload.

The extension has three components:

1. **MOM:** Makes the masters and slaves exchange the configuration changes through messages.
2. **The master:** A master instance has the permissions to make changes in the configuration files. It publishes the changes to its own MOM which in turn is configured to automatically discover other MOM in the same network via Multicast and spread the changes over to the Slaves.
3. **The Slaves:** A slave instance doesn't have the permissions to make the changes, it receives the new configuration changes from the master through the MOM.

To implement our architecture, we followed these steps:

1. Create multiple instances of Geoserver,
2. Create a shared data directory for those machines,
3. Install Geoserver's cluster extension to keep all the nodes in the cluster in synch.
4. Create a load balancer to distribute requests between the multiple instances of GeoServer.

Explanation:

1. First, we installed 4 instances of Geoserver in two physical machines; 2 instances in each one; as in the figure below (The complete installation process is found in the annex A).

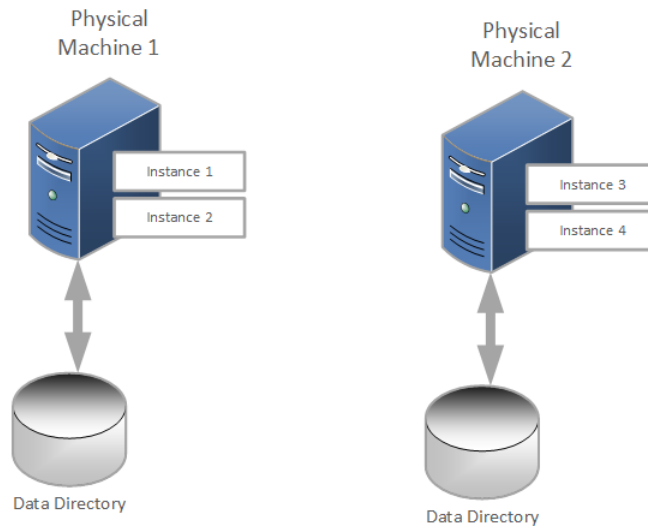


Figure 5.29: Geoserver cluster architecture

⁹Message oriented middleware (MOM), also called broker, is a type of software product that enables message distribution over complex IT systems. In general, middleware serves as a connector for two different applications or platforms.[39]

2. However, the two physical machines didn't share the same data directory and this could be a problem when upgrading or synchronizing all the machines. In order to do that we used sshfs¹⁰.

```
#INSTALL SSHFS
sudo apt-get install sshfs
#CREATE A NEW DATA DIRECTORY
mkdir /mnt/share/geoserverdata
#Run the sshfs command:
sudo sshfs -o allow_other,uid=1002,gid=1002
ubuntu@192.168.20.40:/mnt/share/shared_geoserver_data/
/mnt/share/geoserverdata/
-o IdentityFile=/home/ubuntu/enp-binome2.pem
```

3. Next, we added the MOM extension and configured the instances as Master and Slave at the same time allowing a Peer-to-Peer set-up¹¹.

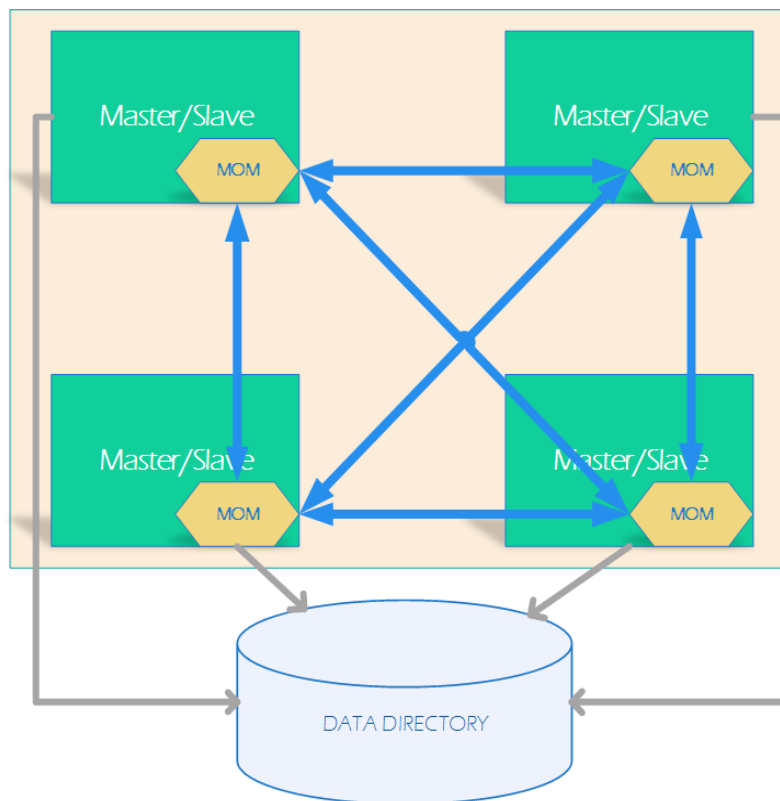


Figure 5.30: GeoServer cluster architecture with shared data directory

- Download the extension:

```
wget https://build.geoserver.org/geoserver/2.13.x/
community-latest/geoserver-2.13-SNAPSHOT-jms-cluster-
plugin.zip
```

¹⁰SSHFS is a filesystem client to mount and interact with directories and files located on a remote server over a normal ssh connection.

¹¹In a Peer-to-peer network, the "peers" are instances which are connected to each other via the Internet. Files can be shared directly between systems on the network without the need of a central server.

- Unzip it inside Geoserver's lib directory

```
unzip geoserver-2.13-SNAPSHOT-jms-cluster-plugin.zip -r
/opt/tomcat/webapps/geoserver/WEBINF/lib
```

- Repeat for all the instances and restart them.

```
sudo systemctl restart tomcat
```

- Verify that the extension was installed properly by checking a new section in the GeoServer user interface called Cluster configuration as shown below.

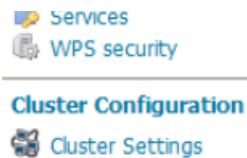


Figure 5.31: Cluster configuration interface

- Configure the cluster extension in order to get the architecture in the figure 5.30:
 - Enable the embedded broker (MOM).
 - Enable master
 - Enable slave.
 - Enable slave connection .
 - Disable the "read only" toggle.

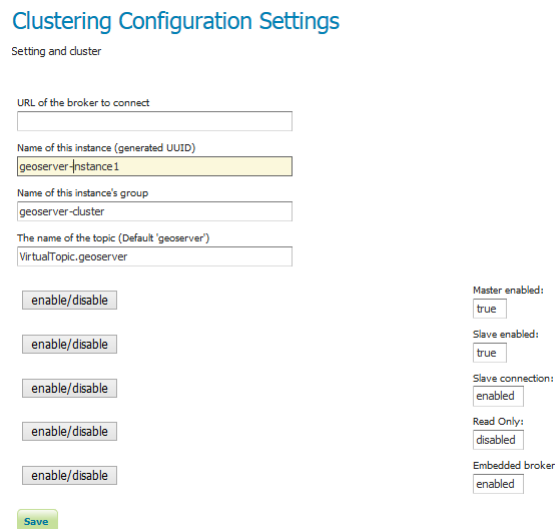


Figure 5.32: Clustering configuration settings

4. Finally we installed and configured the load balancer Apache2 as seen in the figure 5.33(Detailed installation in the annex A). We configured it by:

- Modifying the server.xml files of our tomcat instances inside /opt/tomcat/conf/server.xml

```
#Add the following portion of code
<Engine name="Catalina"
defaultHost="localhost"
jvmRoute="route1">
```

- Modifying the file /etc/apache2/mods-enabled/ proxy_balancer.conf:

```
#Add the following portion of code
<Proxy "balancer://mycluster">
BalancerMember "ajp://192.168.20.40:8009" route=route1
BalancerMember "ajp://192.168.20.40:8010" route=route2
BalancerMember "ajp://192.168.20.27:8009" route=route3
BalancerMember "ajp://192.168.20.27:8010" route=route4
ProxySet lbmethod=bybusyness
Order deny,allow
Allow from all
</Proxy>
ProxyPass "/geoserver" "balancer://mycluster/geoserver"
stickysession=JSESSIONID
```

To make it simple:

- The user accesses the cluster via: `http://[IP_address]/geoserver`
- The load balancer points to the geoserver instances using the `ajp` protocol¹².
- It distribute the requests over the instances based on the busyness of each one.

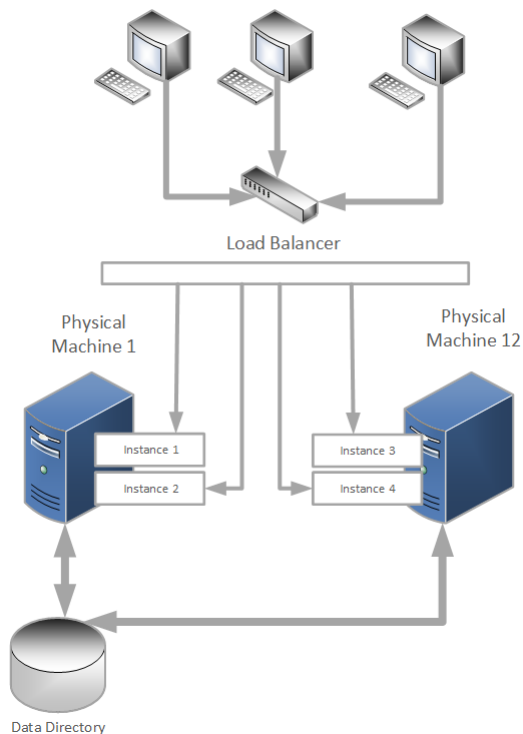


Figure 5.33: Final architecture of GeoServer cluster

¹²Apache JServ Protocol (AJP) is a Tomcat connector that allows Tomcat to communicate with the Apache load balancer. It is an optimized binary version of HTTP.

5.5.3.3 Security

As with any other production system, we must consider securing our instances of Geoserver against malicious attacks and preventing unauthorized access to certain layers.[37]
A security model must respond to two questions before allowing access to the system:

1. **Who?** who is making the request defined as user authentication.
2. **What?** What can this person do in other words user authorization.



Figure 5.34: Authentication and authorization

Source: www.soapui.org

The following diagram shows the security work flow of Geoserver:

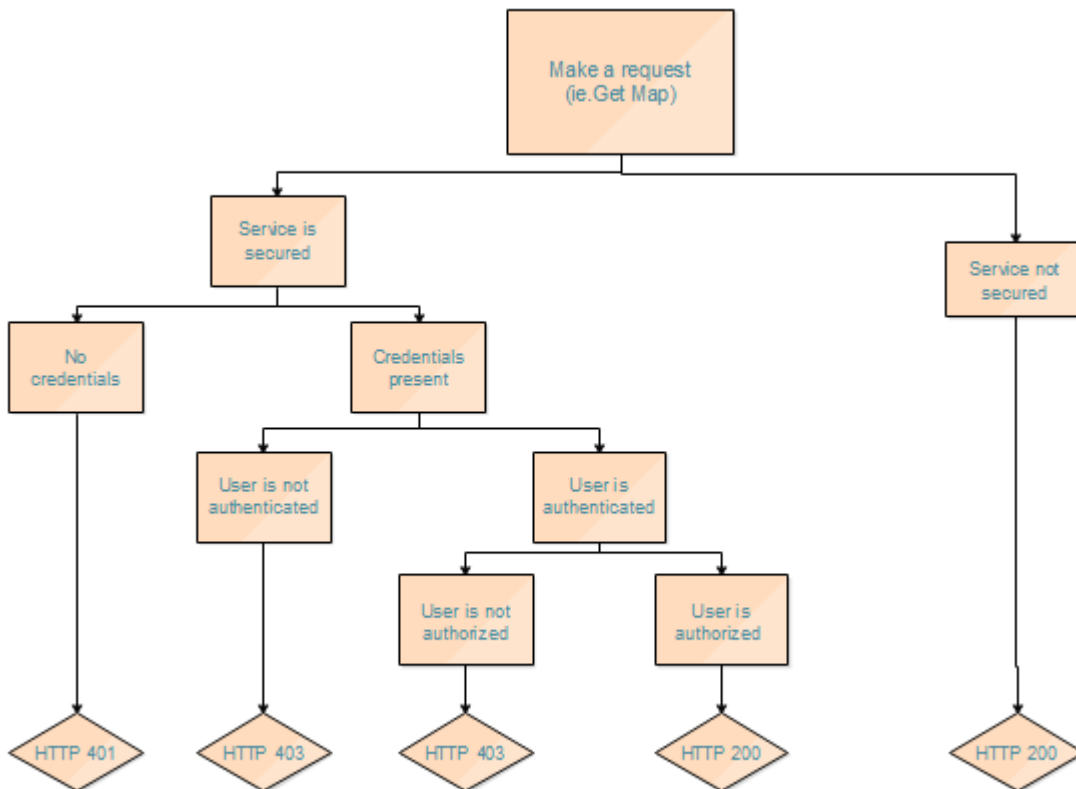


Figure 5.35: Security work flow of Geoserver

The meaning of each response is summarized in the table 5.1:

Table 5.1: HTTP responses

Code	Definition
HTTP 401	The request lacks valid authentication credentials for the target resource.
HTTP 403	The request was valid, but the server is refusing action
HTTP 200	Standard response for successful HTTP requests

In order to secure our GeoServer, we needed to:

1. Change the default password,
2. Activate the encryption¹³,

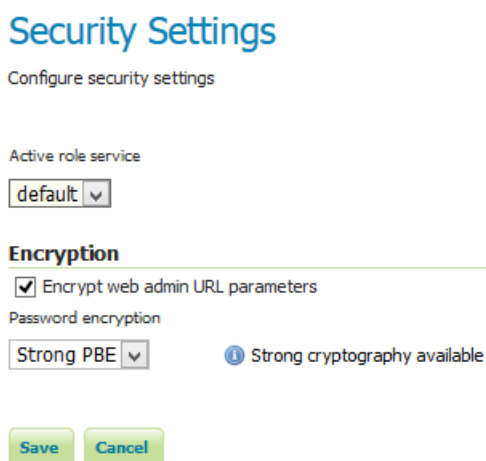


Figure 5.36: Strong Password Encryption

3. Secure our data and who is accessing it:

Security in Geoserver is based on a role system where each role defines a specific set of functions (read, write..). Roles can be assigned to users/groups.

The configuration of security (list of the users, groups, and passwords) can be stored in a XML files (by default), a JDBC database or LDAP¹⁴ directory. Due to the lack of the time, we used the default service (XML based) to create groups and users and associate them with roles.

- (a) **Create groups:** Admins, Editors, Viewers.

Groupname	Enabled
GIS_Admins	✓
GIS_Editors	✓
GIS_Publishers	✓

Figure 5.37: Creating groups

¹³The GeoServer user interface (UI) can sometimes expose parameters in plain text inside the URLs. As a result, it may be desirable to encrypt the URL parameters.[28]

¹⁴The Lightweight Directory Access Protocol (LDAP) is a directory service protocol that runs on a layer above the TCP/IP stack. It provides a mechanism used to connect to, search, and modify Internet directories.

(b) **Create Users** create some users and link them with groups.

Username	Enabled
9edour	✓
Bou3lam	✓
Manel	✓
Yasmine	✓
admin	✓

Figure 5.38: Creating users

(c) **Create roles**

Role	Parent
ADMIN	
GIS_ADMIN	GIS_EDITOR
GIS_EDITOR	GIS_VIEWER
GIS_VIEWER	

Figure 5.39: Creating roles

A child role inherits all the grants from the Parent role.

(d) **Associate a role to users or groups by editing the groups.**

The screenshot shows the 'Edit group' interface. At the top, it says 'Edit group' and 'You can enable/disable the group or change group roles'. Below this, there is a 'Group name' field containing 'GIS_Admins' and an 'Enabled' checkbox which is checked. Underneath, there is a section titled 'Roles taken from active role service: default'. This section contains two columns: 'Available Roles' and 'Selected Roles'. The 'Available Roles' column lists: ADMIN, GIS_EDITOR, GIS_VIEWER, GROUP_ADMIN, and boundless. The 'Selected Roles' column lists: GIS_ADMIN. Arrows between the columns indicate the ability to move roles between them.

Figure 5.40: Linking Groups with roles

(e) **Secure data and service** Now, that groups are linked to roles, we can secure data and service by specifying which role can modify or read specific layer..

New data access rule

Configure a new data access rule

Global layer group rule

Workspace
algeria

Layer and groups
*

Access mode
Read

Roles

Grant access to any role

Available Roles	Selected Roles
Admin GIS_ADMIN GIS_EDITOR GROUP_ADMIN ROLE_ANONYMOUS ROLE_AUTHENTICATED	GIS_VIEWER

Figure 5.41: Granting access rights

5.5.3.4 Watchdog

When running GeoServer in a production environment, we need to check at schedule that Geoserver is still up and responding to requests using a watchdog script.

The watchdog script performs the checks described in the state machine of figure 5.42

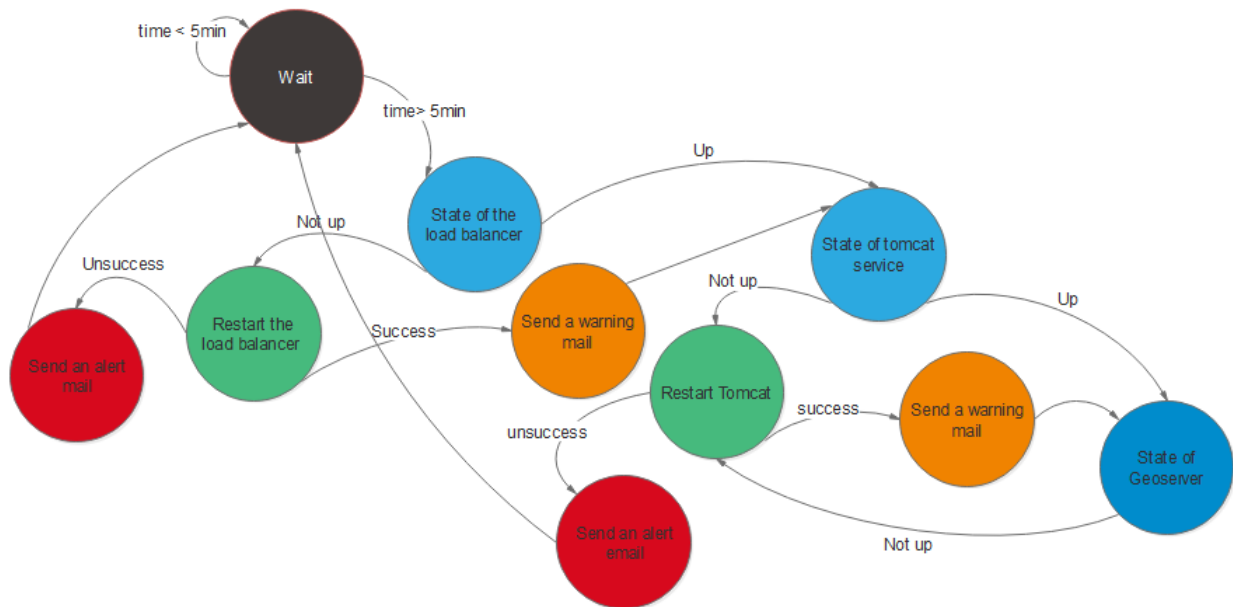


Figure 5.42: Watchdog's state machine

Our watchdog code (can be found in the Annex C) has been inspired by the code published in the book: Mastering GeoServer [37].

5.5.3.5 GeoWebCache

GeoWebCache is integrated with GeoServer, however, in a cluster architecture it is better to use it as a standalone product to avoid duplication and waste of memory. In this part we will go through the steps followed to implement Geowebcache as standalone.

1. Download geowebcache war and unzip it inside tomcat folder

```
wget https://sourceforge.net/projects/geowebcache/files/geowebcache/1.12.2/geowebcache-1.12.2-war.zip
unzip geowebcache-1.12.2-war.zip -r /opt/tomcat/webapps
```

2. Restart tomcat,
3. Configure the layers from geowebcache.xml file. In our case it is located by default inside /opt/tomcat/temp/geowebcache:

```
vi geowebcache.xml
```

4. Add the static layer, in our case satellite imagery and OSM data.

```
<wmsLayer>
  <name>raster layer</name>
  <mimeFormats>
    <string>image/gif</string>
    <string>image/jpeg</string>
    <string>image/png</string>
    <string>image/png8</string>
  </mimeFormats>
  <wmsUrl>
    <string>http://ip-address/geoserver/wms</string>
  </wmsUrl>
  <wmsLayers>algeria:algiers_image1</wmsLayers>
<legends defaultWidth="20" defaultHeight="20"
defaultFormat="image/png">
  <legend style=""/>
</legends>
</wmsLayer>
```

5. In the demo page, reload the configuration.
6. GWC creates a folder for each layer and every time a tile is created it is stored to be used.

5.6 Conclusion

In this chapter, we documented the development and implementation in a production environment of our Iot-oriented GIS built from the combination of Open Source solutions: Sentilo, Greenplum, Geoserver.

Chapter 6

Performance testing and web-mapping application

6.1 Introduction

In this last chapter, we will start by testing the performance of our GeoServer architecture and, at the same time, highlighting the effects of the cluster on the response time.

We will conclude by presenting our GIS end-user application: a Geoportal. Which underlines the main features of our architecture. A Geoportal is one of the many applications of GIS in Smart City.

6.2 Performance testing

In the following, we will present the results obtained from the performance testing using Apache Jmeter. These results show the impact of the clustering on the response time of the server.

6.2.1 Apache Jmeter

Apache JMeter is an open source Java desktop application, built to verify functional behavior, perform load tests, and measure performance. JMeter send requests to a target server by simulating a group of users as shown in the figure 6.1 then collect data to calculate statistics and display performance metrics through various formats.

6.2.2 Scenario of stress testing

The performance test aims to stress GeoServer and evaluates the response time and throughput with an increasing number of simulated users sending concurrent requests for vector data to the server.

The test's configurations are:

1. Test duration: 600 s for scenario 1 and 2, 1200s for scenario 3.
2. Jmeter is sending the requests from a machine in the same network as GeoServer.
3. The data layer used for the tests is the sensors data layer : `algeria:greenplum_sentilo_observations`.

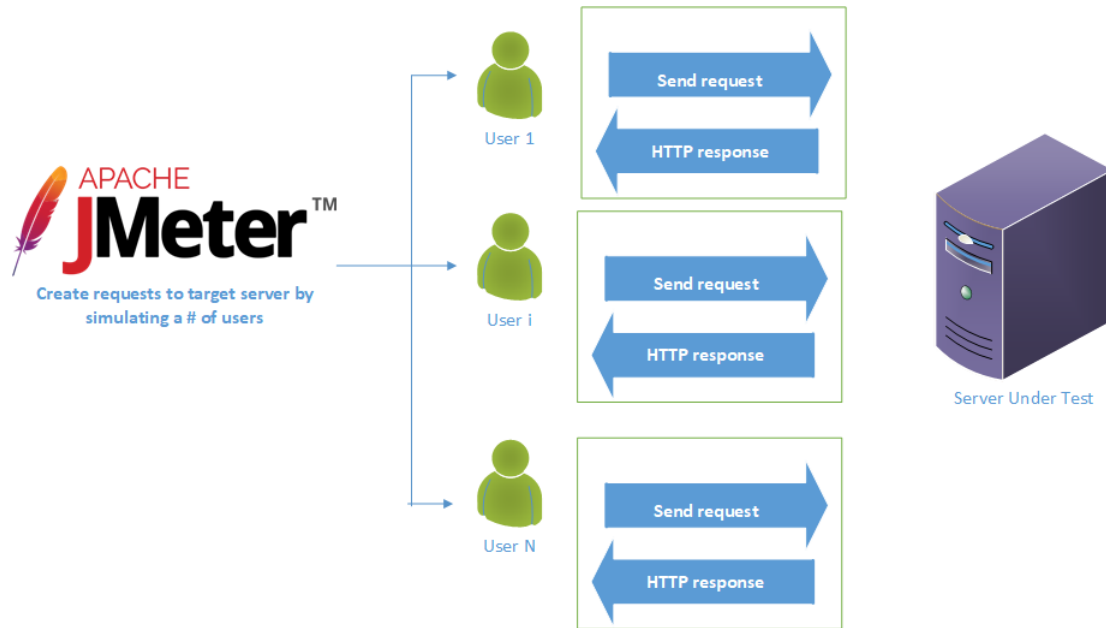


Figure 6.1: Apache Jmeter work flow

4. The Geo Web cache is not activated

The number of users has been gradually increased to simulate low and high demand scenarios and test the availability of the server. In each test, we tried different scenarios, that are summarized in the table 6.1:

Table 6.1: Stress testing scenario

Scenario	# of users
Test 1	4
Test 2	30
Test 3	60

Those tests have been done twice: before and after clustering in order to compare the impact of clustering on the performance of Geoserver.

6.2.3 Results

The performance test aims to stress the server and to evaluate the response time¹ with an increasing number of simulated users sending concurrent requests to the server.

We will evaluate and compare the response time for each scenario before and after clustering.

1. Results of scenario 1:

Before clustering:

¹The total amount of time it takes to respond to a request for service

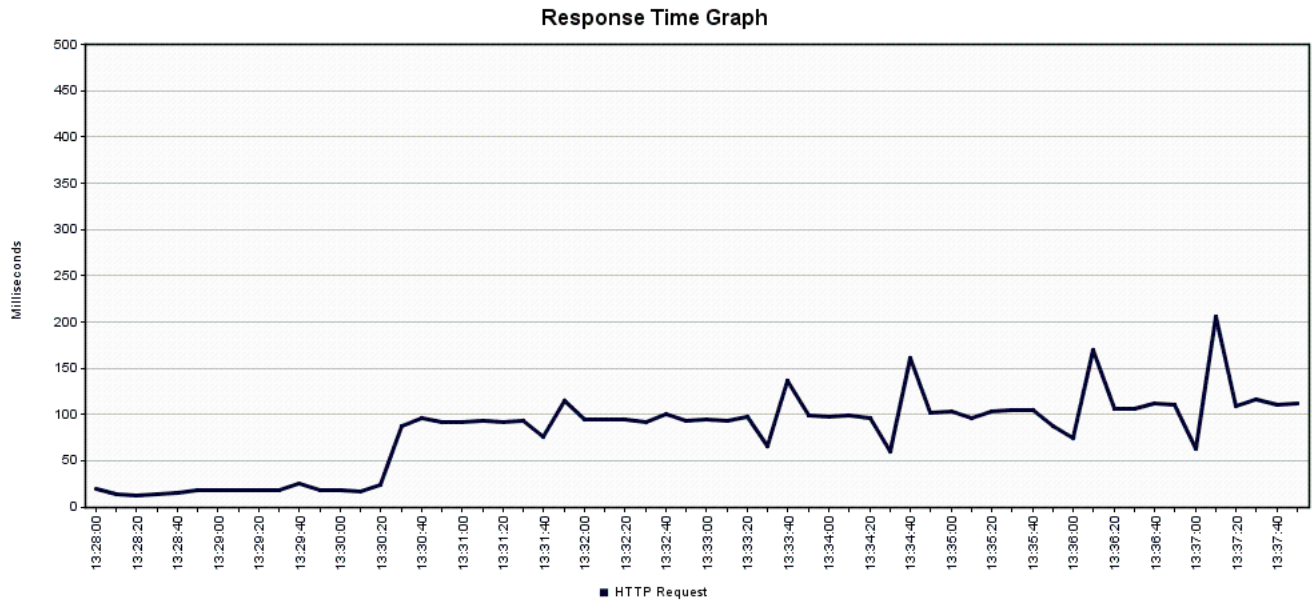


Figure 6.2: Test1 before clustering

After clustering:

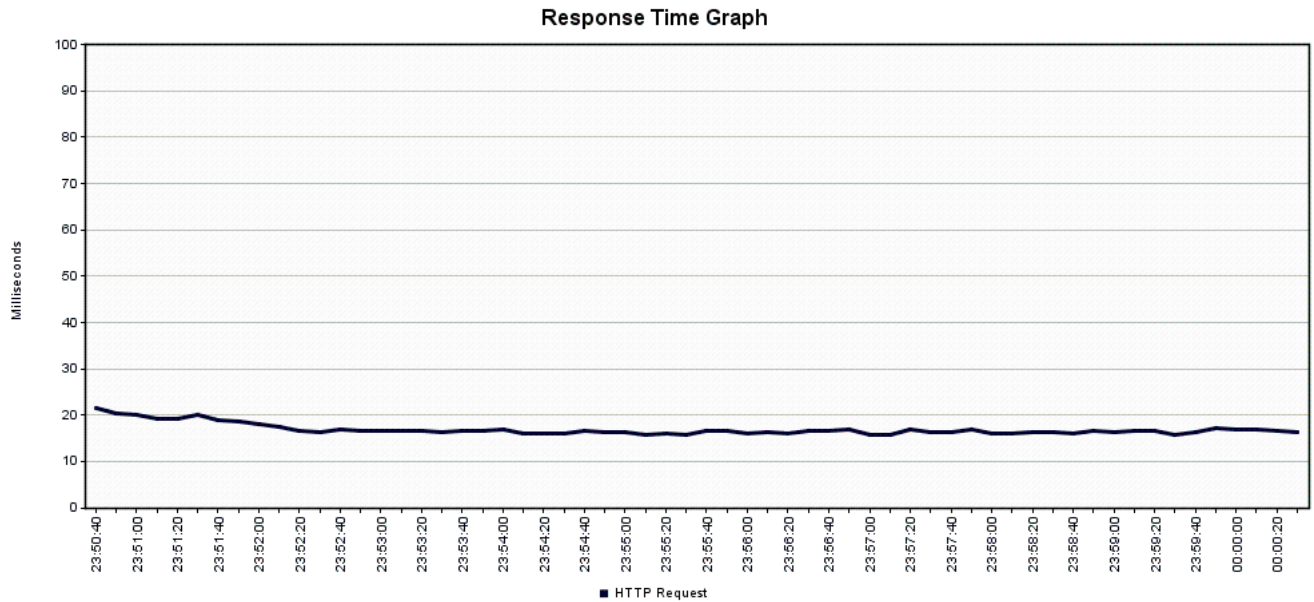


Figure 6.3: Test1 after clustering

2. Results of scenario 2:

Before clustering:

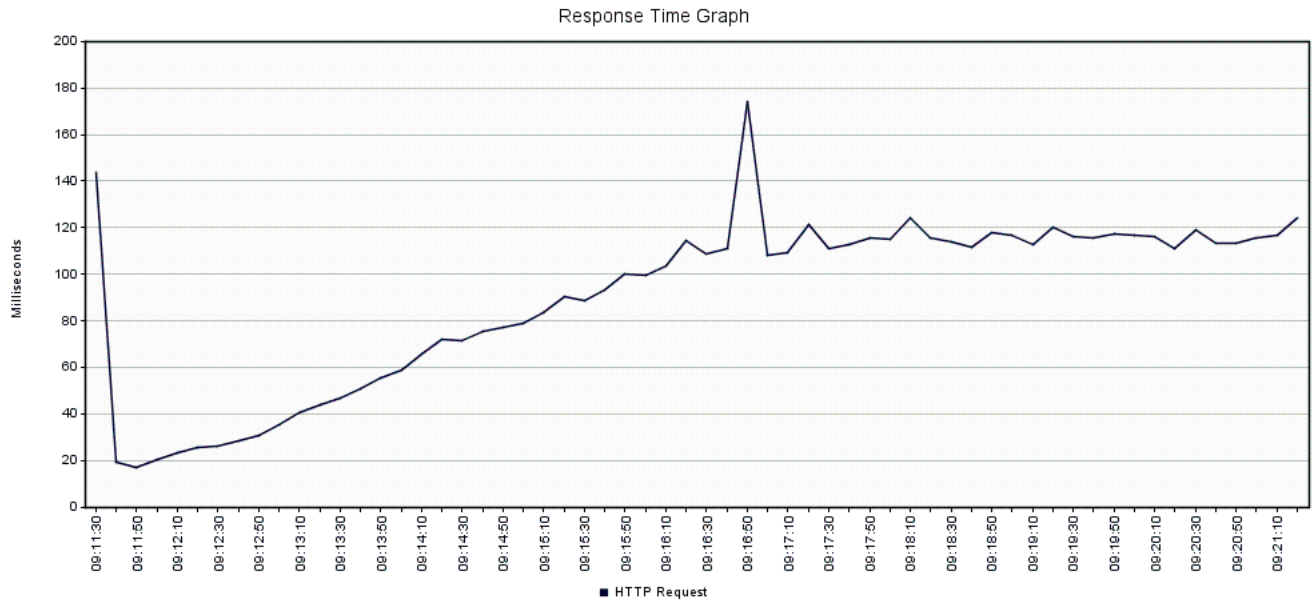


Figure 6.4: Test 2 before clustering

After clustering:

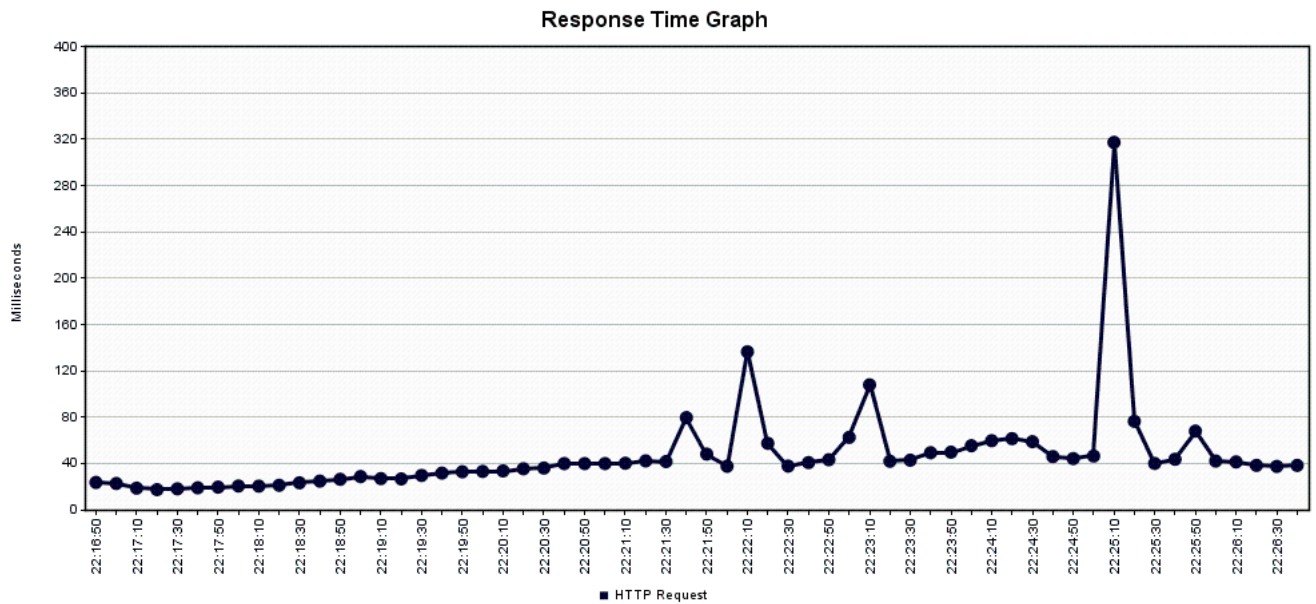


Figure 6.5: Test 2 after clustering

3. Results of scenario 3:

Before clustering:

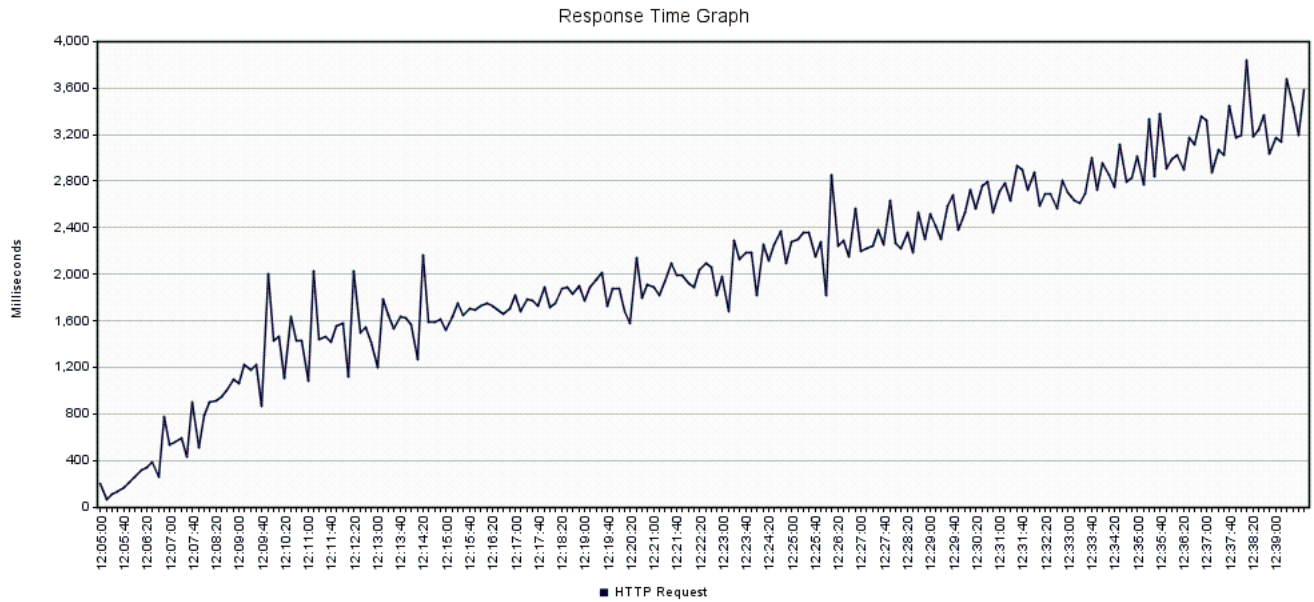


Figure 6.6: Test 3 before clustering

After clustering:

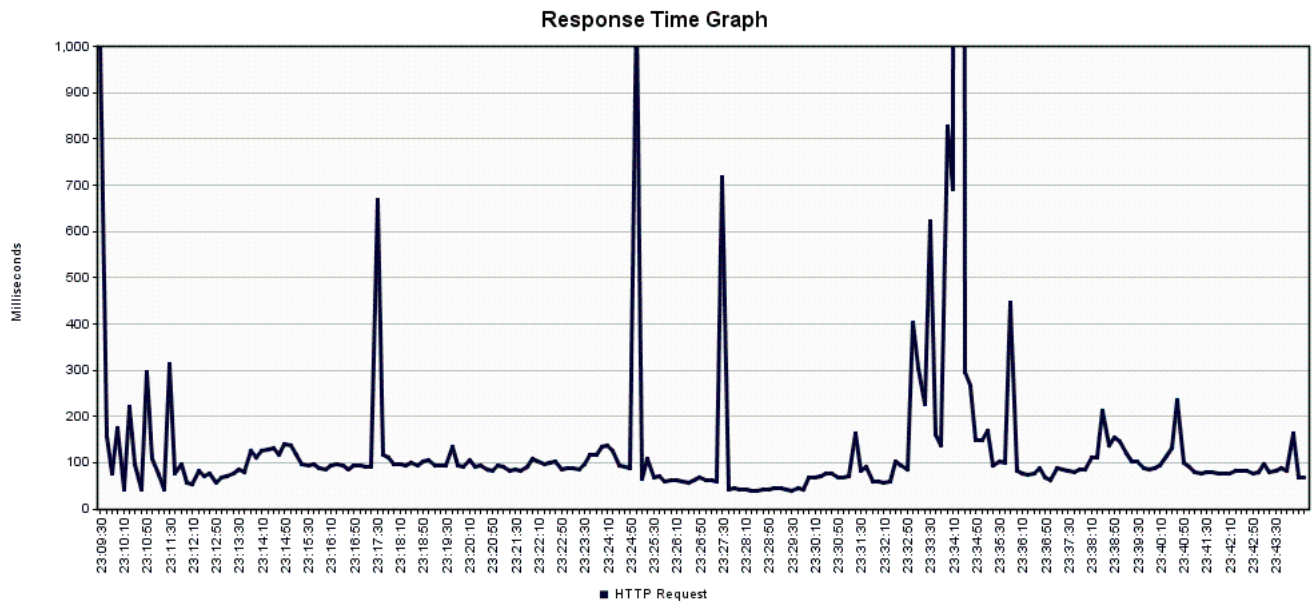


Figure 6.7: Test 3 after clustering

Observations:

1. In non-clustered architecture the response time proportionately increases with the load charge while in the clustered architecture the response time is relatively constant.
2. The average response time is improved with cluster architecture because since more GeoServer instances are available, the requests are processed faster.

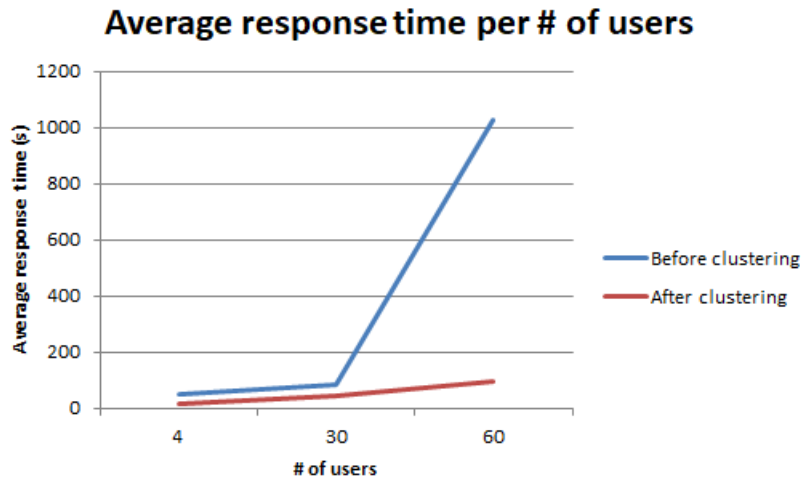


Figure 6.8: Average response time per # of users

3. We can also notice peaks with the increase in the number of users that we could not interpret.

From these results, we can come with the conclusion that the cluster architecture had considerably improved the performance of GeoServer by decreasing the response time.

6.3 Application

Finally, to illustrate our work with a web mapping application, we opted for a geoportal. Geoportals are crucial for smart cities, they play an increasingly important role in sharing geographic information via Internet and can avoid duplicated efforts, inconsistencies, delays, confusion and waste of resources.[40]. They act as the bridge towards e-government services by providing information about the city and its available services such as:

1. Parking facilities
2. Light sensors distributed throughout the city
3. City temperature and humidity levels
4. Garbage collection

6.3.1 Web mapping

As we said in the chapter 3 (client side), a web-based map application is a thin GIS client. In this section, we will present the solutions used for the development of our web mapping application. In our journey to find an easy and rapid way to develop our web map application, we get across Boundless SDK². The strength of Boundless SDK is that it is based on OpenLayers and ReactJS.

²An SDK or Software Development Kit provides a set of tools, libraries, relevant documentation, code samples, processes, and or guides that allow developers to create software applications on a specific platform.

6.3.1.1 OpenLayers

OpenLayers is a free Open Source JavaScript library for displaying map data in web browsers. Its main role is asking the map server for maps through OGC services.

6.3.1.2 ReactJS

ReactJS is a JavaScript library for building user interfaces by providing modular components, which can be used to create complete web applications quickly and easily.

6.3.1.3 Boundless

Boundless provides out of the box components for making it easy to build modern web mapping applications. It provides modular components, in other words, building an application become as simple as playing legos. In the table 6.2, we will define the main Boundless's functions:

Table 6.2: Boundless SDK functions

Function	Definition
AddLayerModal	Modal window to add layers from a WMS, WFS,WMTS or ArcGIS REST service.
BaseMapModal	A modal of basemap thumbnails for selecting a basemap to be used in the map.
DrawFeature	Allows users to draw new features.
Geocoding	Input field to search for placenames using a geocoding service (OSM nominatim).
Geolocation	Enable geolocation which uses the current position of the user in the map
ImageExport	Export the map as a PNG file.
InfoPopup	Popup to show feature info. This can be through WMS/WMTS GetFeatureInfo or local vector data.
MapConfig	Export the map configuration and ability to reload it from local storage.
Measure	Adds area and length measure tools as a menu button
Playback	Adds a slider to the map that can be used to select a given date, and modifies the visibility of layers and features depending on their timestamp and the current time.
ZoomToLatLon	Component that allows zooming the map to a lat/lon position.

6.3.2 Web Application's description

In our application we wanted to give the main features of a Geoportal in order to provide professionals and general public with access to geographic information. The developed application has the following features (The complete code can be found on Github link: <https://github.com/ManalBoucennaMaroua/IoT-oriented-GIS-Geoportal.git>):

1. Provide different map backgrounds,
2. Connect to GeoServer and display its layers using WMS service, for instance the sensor layers: "pressure", "temperature"..

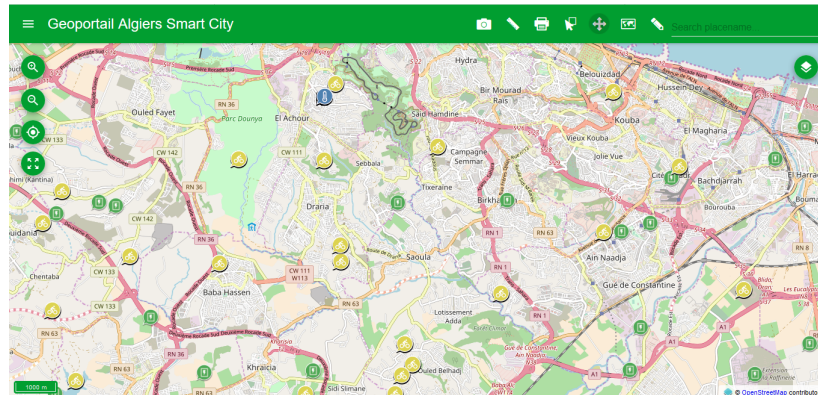


Figure 6.9: Geoportal Web Page

3. Connect to GeoServer and display its data in a table and chart using WFS service,

Layer		Filter	
Capteurs de température		Type filter	
		r	id
<input type="checkbox"/>		9	97971
<input type="checkbox"/>		7	80236
<input type="checkbox"/>		6	24426
<input type="checkbox"/>		5	39190
<input type="checkbox"/>		1	110697

Figure 6.10: Geoportal's table

4. Measure distance and area,

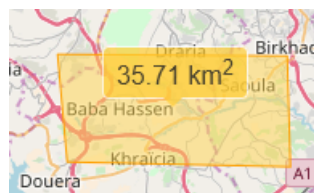


Figure 6.11: Area measurement

5. Allow the user to upload its own spatial files, modify them for personal uses and then download them.

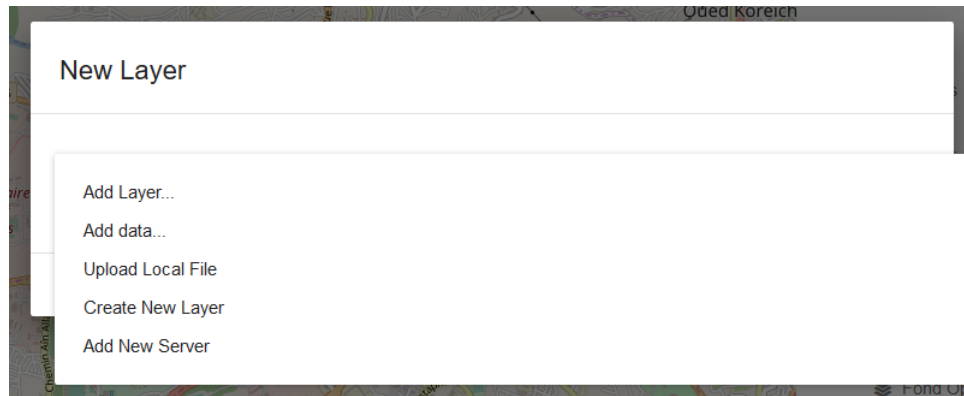


Figure 6.12: New Layer

6. Search by place names using OpenStreetMap address's catalog.
7. Query features to filter the data,
8. Print or export map tiles,
9. Draw features (point, line and polygons),



Figure 6.13: Geoportal's tools

10. Geolocation.

6.4 Conclusion

In this chapter, we presented the performance test results conducted on the implemented architecture using Apache Jmeter. We noticed that the clustering of Geoserver enhanced the response time. Next, we introduced the web application used to illustrate some of the possible features of our architecture: a Geoportal. We believe that this web application can be improved in the future and that many other projects can be drawn from it.

Chapter 7

Conclusion and future work

7.1 Summary

As discussed in the introduction chapter, making cities smarter using geospatial technologies is now crucial. This work contributes towards smart cities technology by providing an end-to-end solution for an IoT-oriented GIS architecture using Open Source that collect the sensor's data and store it in a big data platform in order to manipulate them and share the results through the web using OGC services.

In the first chapters of this work we focused on the theoretical concepts of the Smart city's key elements: IoT, Big Data and GIS.

Then, we defended our technical choices for the design and implementation of this project. Different Open Source technologies were used, ranging from GIS technologies to IoT and Big Data technologies:

1. IoT platform: Sentilo,
2. Big data platform: Greenplum Pivotal MPP database,
3. GIS Server: GeoServer.

Next, we presented the steps followed for the implementation of each element of our architecture and validated that it works properly with performance tests.

Finally, we highlighted the possible features of our architecture through a web-based application: Geoportal.

Despite the difficulties we have faced during our graduation project such as logistic problems with the industrial partners (we couldn't exploit samples of their data or get deeply in touch with their GIS team), we wanted to present a work that approaches reality using Open data.

7.2 Future work

As part of the continuation of this project, recommendations and future work could focus on:

- Improve the security and authentication configurations,
- Apply more in-depth performance tests on vector and raster data,

- Employ data mining methods to derive more information from the spatial data acquired from sensors by using MADlib library,
- Migrate to the official Sentilo's platform implemented by the "Algiers Smart City" project's team.

To illustrate the capabilities of our architecture we opted for a Geoportal. However, this is just the tip of the iceberg, hundreds of applications can be drawn from this work such as:

- **Waste management:**

A web map application that allows the visualization in real time of the sensor's data installed in trash cans to provide the garbage trucks with information on the state of the trash cans (full or empty).

- **GPS tracking:**

an application that estimates the time of arrival of the bus to its stop station based on the GPS sensors data installed in buses and the state of the traffic and share this information through a web mapping with the clients.

- **Smart parking:**

An application that allows the visualization of the nearest free parking spots in a map and to pay directly through the map.

Bibliography

Bibliography

- [1] A. J. H. Stefan Steiniger, “The 2012 free and open source gis software map – a guide to facilitate research, development, and adoption.”
- [2] M. A. Toups, “A study of three paradigms for storing geospatial data:distributed-cloud model, relational database, and indexed flat file.” <https://scholarworks.uno.edu/cgi/viewcontent.cgi?article=3292&context=td>. [Online; accessed April-2018].
- [3] T. D. of Economic and S. A. of the United Nations Secretariat, “World urbanization prospects.” <https://esa.un.org/unpd/wup/publications/files/wup2014-highlights.pdf>. [Online; accessed June-2018].
- [4] “What is iot?.” <https://www.happiestminds.com/Insights/internet-of-things>.
- [5] “Iot platforms: What they are how to select one.” <https://www.link-labs.com/blog/what-is-an-iot-platform>, 2016.
- [6] “5 things to know about the iot platform ecosystem.” <https://iot-analytics.com/5-things-know-about-iot-platform/>, 2016.
- [7] “How to choose the right iot platform: The ultimate checklist.” <https://hackernoon.com/how-to-choose-the-right-iot-platform-the-ultimate-checklist-47b5575d4e20>, 2018.
- [8] “Big data platform options and technologies.” <http://www.bodhtree.com/blog/2012/09/14/big-data-platform-options-and-technologies/>, 2012.
- [9] “Analytical mpp databases.” <https://databases.looker.com/analytical>. [Online; accessed March-2018].
- [10] “Information system defintion.” <http://www.businessdictionary.com/definition/information-system.html>.
- [11] S. Shukla, “Gis brings out the best in smart cities.” <https://www.geospatialworld.net/blogs/gis-smart-cities-go-together>. [Online; accessed March-2018].
- [12] S. DEOGAWANKA, “How gis supports the planning and development of smart cities.” <https://www.gislounge.com/how-gis-supports-the-planning-and-development-of-smart-cities>. [Online; accessed March-2018].
- [13] C. DEMPSEY, “John snow’s cholera map using gis data.” <https://www.gislounge.com/john-snows-cholera-map-gis-data>. [Online; accessed June-2018].

- [14] E. Hamilton, “Client-side versus server-side geoprocessing: Benchmarking the performance of web browsers processing geospatial data using common gis operations.” http://cugos.org/image/slides/CUGOS_20140915.pdf. [Online; accessed May-2018].
- [15] S. Depraz, “Ressources de géographie pour les enseignants.” <http://geoconfluences.ens-lyon.fr/glossaire/geoide-ellipsoide>. [Online; accessed March-2018].
- [16] “Système géodésique (datum).” https://georezo.net/wiki/main/dico/systeme_geodesique. [Online; accessed March-2018].
- [17] “Local vs. earth centered datums.” http://gsp.humboldt.edu/olm_2015/Lessons/GIS/02%20Datums/Local_vs_Earth_Centered_Datums.html. [Online; accessed March-2018].
- [18] F. DANIEL, “Mercator projection.” <https://www.universalis.fr/encyclopedie/projection-cartographique-de-mercator>. [Online; accessed March-2018].
- [19] A. Briney, “Types of maps: Topographic, political, climate, and more.” <https://www.thoughtco.com/types-of-maps-1435689>. [Online; accessed March-2018].
- [20] “The remarkable history of gis.” <https://gisgeography.com/history-of-gis>, 2018. [Online; accessed May-2018].
- [21] M. Schneider, “Spatial data types.” <https://www.cise.ufl.edu/~mschneid/Research/papers/Sch09BoChb.pdf>. [Online; accessed March-2018].
- [22] “Spatial database.” https://en.wikipedia.org/wiki/Spatial_database. [Online; accessed March-2018].
- [23] S. Steiniger and R. Weibel, “Gis software - a description in 1000 words,” 2009.
- [24] “Application programming interface.” https://en.wikipedia.org/wiki/Application_programming_interface. [Online; accessed March-2018].
- [25] “About the greenplum architecture.” <https://gpdb.docs.pivotal.io>, 2018.
- [26] K. Kommana, “Implementation of a geoserver application for gis data distribution and manipulation,” 2013.
- [27] “Mapserver web site.” <http://mapserver.org/fr>. [Online; accessed January-2018].
- [28] “Geoserver web site.” <http://docs.geoserver.org>. [Online; accessed January-2018].
- [29] “Geotools web site.” www.geotools.org. [Online; accessed May-2018].
- [30] “Introduction to geoserver.” <http://workshops.boundlessgeo.com/geoserver-intro/overview/concepts.html>.
- [31] “Client url definition.” <https://fr.wikipedia.org/wiki/CURL>. [Online; accessed April-2018].
- [32] L. James, “Defining open data.” <https://blog.okfn.org/2013/10/03/defining-open-data>. [Online; accessed March-2018].

- [33] "Maven pom.xml file." <https://www.javatpoint.com/maven-pom-xml>.
- [34] "Openstreetmap web site." https://wiki.openstreetmap.org/wiki/About_OpenStreetMap.
- [35] "Openstreetmap wiki site." <https://wiki.openstreetmap.org>. [Online; accessed March-2018].
- [36] "Openstreetmap data to postgis-enabled postgresql databases." <https://wiki.openstreetmap.org/wiki/Osm2pgsql>.
- [37] C. Henderson, "Mastering geoserver."
- [38] "Cluster definitions." <https://www.techopedia.com/definition/997/cluster-servers>. [Online; accessed March-2018].
- [39] "Message oriented middleware, howpublished = <https://www.techopedia.com/definition/27589/message-oriented-middleware-mom>, note =."
- [40] "How are digital technologies influencing geospatial technology trends?." <https://www.geospatialworld.net/blogs/geospatial-technologies-in-digital-platforms>. [Online; accessed June-2018].
- [41] "Gnu lesser general public license." https://en.wikipedia.org/wiki/GNU_Lesser_General_Public_License. [Online; accessed March-2018].
- [42] "Introduction to postgis." <http://workshops.boundlessgeo.com/postgis-intro/introduction.html>. [Online; accessed March-2018].
- [43] "Geowebcache." <http://docs.geoserver.org/stable/en/user/geowebcache/index.html>.
- [44] "Rest api definition." <https://searchmicroservices.techtarget.com/definition/RESTful-API>. [Online; accessed March-2018].
- [45] R. D. G. Sonam Agrawal, "Development and comparison of open source based web gis frameworks on wamp and apache tomcat web servers," 2014.
- [46] J. R. P. Miguel R. Luaces, Nieves R. Brisaboa and J. R. Viqueira, "A generic architecture for geographic information systems."
- [47] B. Y. Stephano Lacovella, "Geoserver beginner's guide."
- [48] S. Lacovella, "Geoserver beginner's guide second edition."
- [49] "Why should a municipality become a smart city? the six key benefits of transforming the place we call home.." <https://hub.beesmart.city/strategy/6-key-benefits-of-becoming-a-smart-city>, 2018.
- [50] M. F. Goodchild, "Citizens as sensors: the world of volunteered geography," 2007.

Annex A: Software installation

1 Installation of Sentilo in Ubuntu 14.04

Installing Sentilo requires:

1. Configuring Redis,
2. Configuring MongoDB,
3. Installing dependencies: git, maven2, redis, tomcat.

1.1 Installing dependencies

We need first to install all the dependencies needed for the server:

```
sudo apt-get install git maven2 redis-server mongodb tomcat
```

1.2 Download and build code

The source code of Sentio project can be obtained from git, cloning the remote project in a local directory named *Sentilo*:

```
git clone https://github.com/sentilo/sentilo.git.sentilo
```

Then, we need to build the project with maven in order to create the executable:

```
cd sentilo
mvn clean install
mvn eclipse:clean eclipse:eclipse
```

1.3 Redis settings

The default configuration of redis listens to the port 6379, host 127.0.0.1, and with the parameter requirepass enabled and with value sentilo.

1.4 MongoDB settings

Sentilo default settings consider MongoDB will be listening on 127.0.0.1:27017, and requires an existing database named sentilo, created before starting the platform, with authentication enabled and with login credentials preconfigured as sentilo/sentilo (username:sentilo, password:sentilo).

1.5 Configure Tomcat7

To deploy the web application in tomcat7 we need to move the .war file to the tomcat server webapps folder and restart the server.

```
sudo cp ~/sentilo/sentilo-catalog-web/target/sentilo-catalog-web.war
/var/lib/tomcat7/webapps
sudo service tomcat7 restart
```

1.6 Start services

There are 2 binaries we will need to launch in order to start the background processes of sentilo. First, create the folders:

```
mkdir /opt/sentilo-server
mkdir /opt/sentilo-agent-relational
```

Then copy all the files into the folders.

```
mv ~/sentilo/sentilo-platform/sentilo-platform-server/target/
appassembler/* /opt/sentilo-server
mv ~/sentilo/sentilo-agent-relational/target/appassembler/*
/opt/sentilo-agent-relational
```

At last, to launch Sentilo we run:

```
sudo chmod 500 -Rf /opt/sentilo-server/bin/
sudo /opt/sentilo-server/bin/sentilo-server
sudo /opt/sentilo-agent-relational-server
```

2 GeoServer installation and configuration

Installing GeoServer requires:

1. Installing Java
2. Installing Tomcat

2.1 Installing Java

From Geoserver website we quote: "GeoServer's speed depends a lot on the chosen Java Runtime Environment (JRE) ... For best performance we recommend the use Oracle JRE 8. As of GeoServer 2.0, a Java Runtime Environment (JRE) is sufficient to run GeoServer. GeoServer no longer requires a Java Development Kit (JDK)."

To install it, we follow thoses steps:

1. Copy and save this code as get_java.py :

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

try:
```

```

    from urllib.request import request, urlopen
except ImportError: # python 2
    from urllib2 import request, urlopen
import re
import os

valid_packages = ['jre', 'server-jre', 'jdk']
valid_java_versions = xrange(7, 9, 1)

def regex_websearch(url, pattern):
    req = request(url)
    resp = urlopen(req)
    content = resp.read().decode('utf-8')
    resp.close()
    match = re.search(pattern, content)
    return match

```

2. Run the code:

```
python get_java.py
```

3. Download the JRE-8 version, unpack it and move it to /usr/java/ :

```

wget http://download.oracle.com/otn-pub/java/jdk/8u161-b12/
2f38c3b165be4555a1fa6e98c45e0808/jre-8u161-linux-x64.tar.gz to
jre-8u161-linux-x64.tar.gz
tar zxvf jre-8u161-linux-x64.tar.gz
mvn /usr/java/

```

4. Configure the JRE as the primary Java alternative in your system:

```
sudo update-alternatives --install /usr/bin/java java
```

```

\item and choose:
\begin{lstlisting}[language=bash]
/usr/java/jre1.8.0_161/bin/java

```

2.2 Installation of Tomcat

2.2.1 Create Tomcat User

For security purposes, Tomcat should be run as an unprivileged user (i.e. not root). We will create a new user and group that will run the Tomcat service.

1. Create a new tomcat group:

```
sudo groupadd tomcat
```

2. Create a new Tomcat user:

```
sudo useradd -s /bin/false -g tomcat -d /opt/tomcat tomcat
```

2.2.2 Install Tomcat

1. Download the latest binary release:

```
curl -O http://www-us.apache.org/dist/tomcat/tomcat-8/v8.5.29/
bin/apache-tomcat-8.5.29.tar.gz
```

2. Install Tomcat to the `/opt/tomcat` directory:

```
sudo mkdir /opt/tomcat
sudo tar xzvf apache-tomcat-8*tar.gz -C /opt/tomcat
--strip-components=1
```

2.2.3 Set up the proper user permissions

The tomcat user that we set up needs to have access to the Tomcat installation.

1. Go to the directory where you installed tomcat

```
cd /opt/tomcat
```

2. Give the tomcat group ownership over the entire installation directory:

```
sudo chgrp -R tomcat /opt/tomcat
```

3. Next, give the tomcat group read access to the conf directory and all of its contents, and execute access to the directory itself:

```
sudo chmod -R g+r conf
sudo chmod g+x conf
```

4. Make the tomcat user the owner of the webapps, work, temp, and logs directories:

```
sudo chown -R tomcat webapps/ work/ temp/ logs/
```

2.2.4 Create a systemd Service File

Create a systemd service file¹ to manage the Tomcat process:

1. Open a file called tomcat.service in the `/etc/systemd/system` directory

```
sudo vi /etc/systemd/system/tomcat.service
```

2. Paste the following contents into your service file and save it:

```
[Unit]
Description=Apache Tomcat Web Application Container
After=network.target

[Service]
```

¹Systemd service file provides an easy way to manage and control services and a simple method of creating your own services.

```
Type=forking
```

```
Environment=JAVA_HOME=/usr/java/jre1.8.0_161/  
Environment=CATALINA_PID=/opt/tomcat/apache-tomcat-8.5.5/temp/  
tomcat.pid  
Environment=CATALINA_HOME=/opt/tomcat/apache-tomcat-8.5.5/  
Environment=CATALINA_BASE=/opt/tomcat/apache-tomcat-8.5.5/  
Environment='CATALINA_OPTS=-Xms512M -Xmx1024M -server  
-XX:+UseParallelGC'  
Environment='JAVA_OPTS=-Djava.awt.headless=true  
-Djava.security.egd=file:/dev/./urandom'  
Environment='GEOSERVER_DATA_DIR=/mnt/share/geoserver-data'  
Environment='GEOSERVER_LOG_LOCATION=\$GEOSERVER_DATA_DIR/logs/  
geoserver-1.log'  
ExecStart=/opt/tomcat/apache-tomcat-8.5.5/bin/startup.sh  
ExecStop=/opt/tomcat/apache-tomcat-8.5.5/bin/shutdown.sh
```

```
User=tomcat  
Group=tomcat  
UMask=0007  
RestartSec=10  
Restart=always
```

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

3. Reload the systemd daemon so that it knows about our service file:

```
sudo systemctl daemon-reload
```

4. Start the Tomcat service and check the status

```
sudo systemctl start tomcat  
sudo systemctl status tomcat
```

2.2.5 Adjust the Firewall and Test the Tomcat Server

1. Tomcat uses port 8080 to accept conventional requests. Allow traffic to that port and Open in Web browser

```
sudo ufw allow 8080  
http://server_domain_or_IP:8080
```

You will see the default Tomcat splash page, in addition to other information.

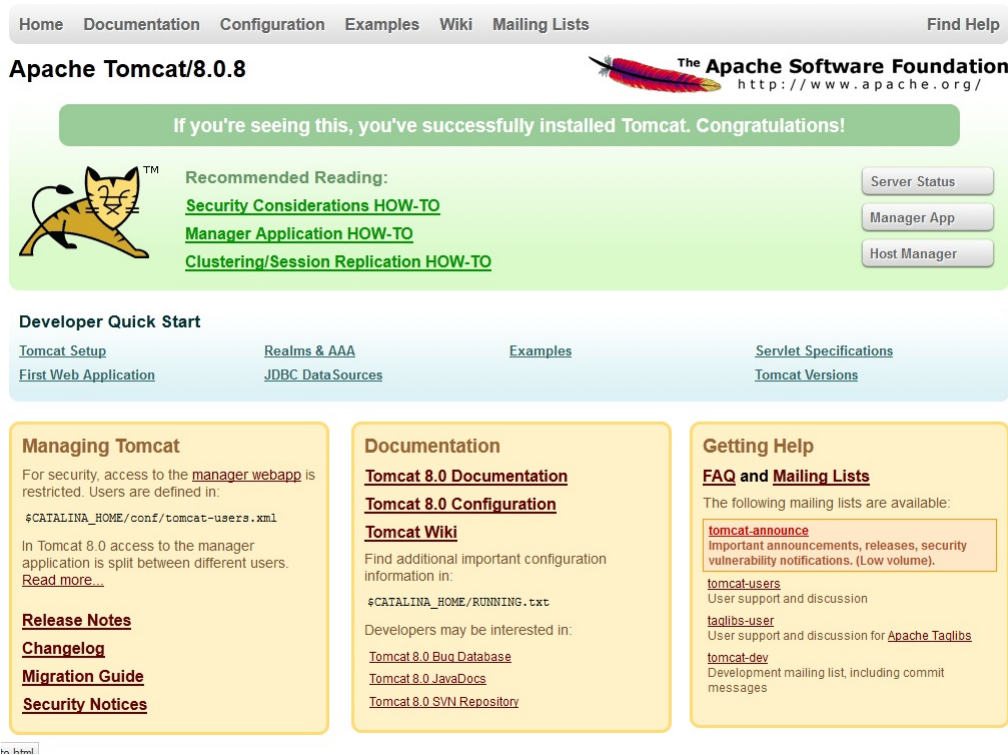


Figure 7.1: Tomcat home page

2. Enable the service file so that Tomcat automatically starts at boot:

```
sudo systemctl enable tomcat
```

2.2.6 Configure Tomcat Web Management Interface

In order to use the manager web app that comes with Tomcat, we must add a login to our Tomcat server.

1. We edit the tomcat-users.xml file to add a user who can access to manager-gui and admin-gui(web apps that come with Tomcat)

```
sudo vi /opt/tomcat/apache-tomcat-8.5.5/conf/tomcat-users.xml
```

```
<tomcat-users . . .>
  <user username="admin"
    password="password"
    roles="manager-gui,admin-gui"/>
```

2. By default, newer versions of Tomcat restrict access to the Manager and Host Manager apps to connections coming from the server itself. To change the IP address restrictions on these, open the appropriate context.xml files.

- (a) For the Manager app

```
sudo vi /opt/tomcat/apache-tomcat-8.5.5/webapps/manager/
META-INF/context.xml
```

(b) For the Host Manager app For the Host Manager app

```
sudo vi /opt/tomcat/apache-tomcat-8.5.5/webapps/host-manager/META-INF/context.xml
```

Inside, comment out the IP address restriction to allow connections from anywhere.

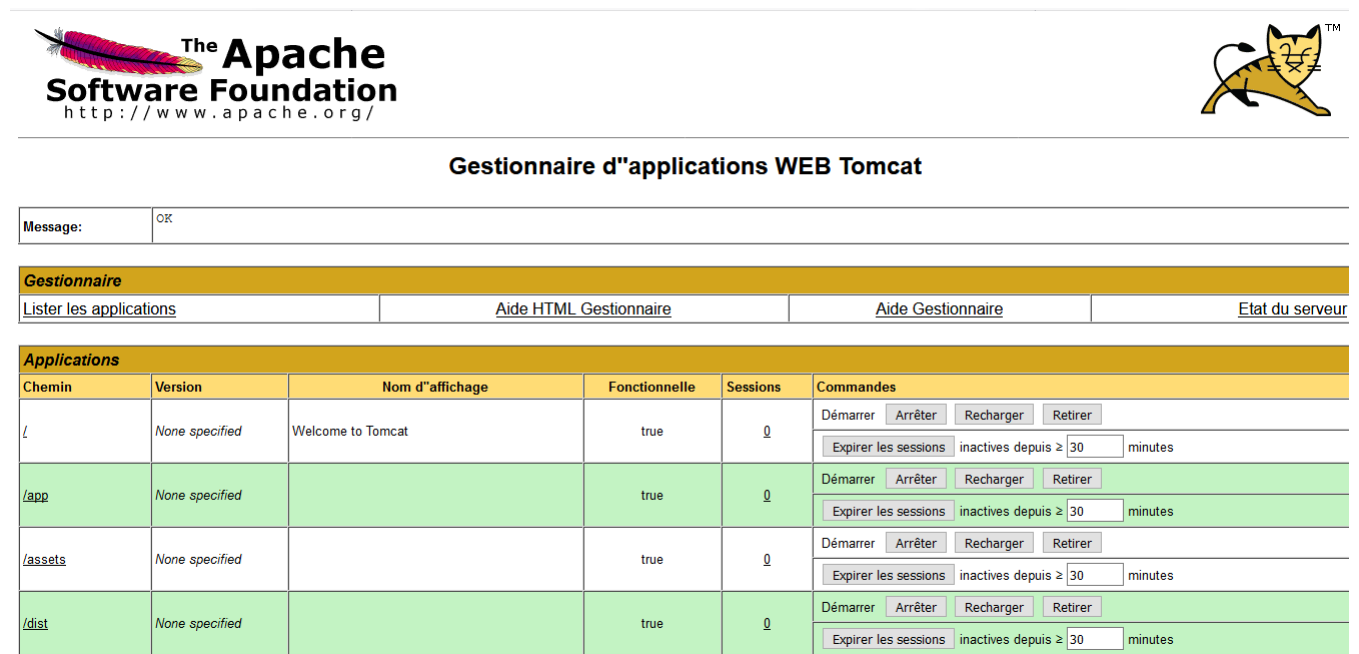
```
<Context antiResourceLocking="false" privileged="true" >
  <Valve className="org.apache.catalina.valves.RemoteAddrValve"
    allow="127\.\d+\.\d+\.\d+|::1|0:0:0:0:0:0:0:1" />
</Context>
```


3. Restart the Tomcat service

```
sudo systemctl restart tomcat
```

2.2.7 Access the Web Interface

1. Open in web browser `http://server_domain_or_IP:8080` The page you see should be the same one you were given when you tested earlier.
2. Enter the account credentials that you added to the `tomcat-users.xml` file.



The Apache Software Foundation 

Gestionnaire d'applications WEB Tomcat

Message: OK

Gestionnaire

[Lister les applications](#) [Aide HTML Gestionnaire](#) [Aide Gestionnaire](#) [Etat du serveur](#)

Applications

Chemin	Version	Nom d'affichage	Fonctionnelle	Sessions	Commandes
/	None specified	Welcome to Tomcat	true	0	Démarrer Arrêter Recharger Retirer Expirer les sessions inactives depuis ≥ 30 minutes
/app	None specified		true	0	Démarrer Arrêter Recharger Retirer Expirer les sessions inactives depuis ≥ 30 minutes
/assets	None specified		true	0	Démarrer Arrêter Recharger Retirer Expirer les sessions inactives depuis ≥ 30 minutes
/dist	None specified		true	0	Démarrer Arrêter Recharger Retirer Expirer les sessions inactives depuis ≥ 30 minutes

Figure 7.2: Web Application Manager

The Web Application Manager is used to manage your Java applications. You can Start, Stop, Reload, Deploy, and Undeploy here. You can also run some diagnostics on your apps.

3. Currently, Tomcat installation is functional, but entirely unencrypted. This means that all data, including sensitive items like passwords, are sent in plain text that can be intercepted and read by other parties on the internet. In order to prevent this from happening, it is strongly recommended that you encrypt your connections with SSL.

2.3 Installing GeoServer

2.3.1 Create a geoserver instance

1. Download the OS-independent version from Geoserver's download page and unzip it:

```
wget http://sourceforge.net/projects/geoserver/files/GeoServer/2.12.2/geoserver-2.12.2-war.zip
unzip geoserver-2.12.2-war.zip
```

2. The war file for GeoServer is quite big (>52 MByte). In Tomcat 8 Manager there is a default limit for deployable application that is at 50 MByte that we will need to change.

```
vi /opt/tomcat/apache-tomcat-8.5.5/webapps/manager/WEB-INF/web.xml
```

Look for this section:

```
<multipart-config>
<!-- 50MB max -->
<max-file-size>52428800</max-file-size>
<max-request-size>52428800</max-request-size>
<file-size-threshold>0</file-size-threshold>
</multipart-config>
```

Set the max-file-size to 62914560 value both in max-file-size and max-request-size parameters. Save the file and restart Tomcat.

3. Copy and Paste the war file into /opt/tomcat/webapps/

```
mv geoserver-2.12.2-war /opt/tomcat/apache-tomcat-8.5.5/webapps/
```

This will create a geoserver file.

4. Open in a Web server: http://IP-Address:8080/geoserver

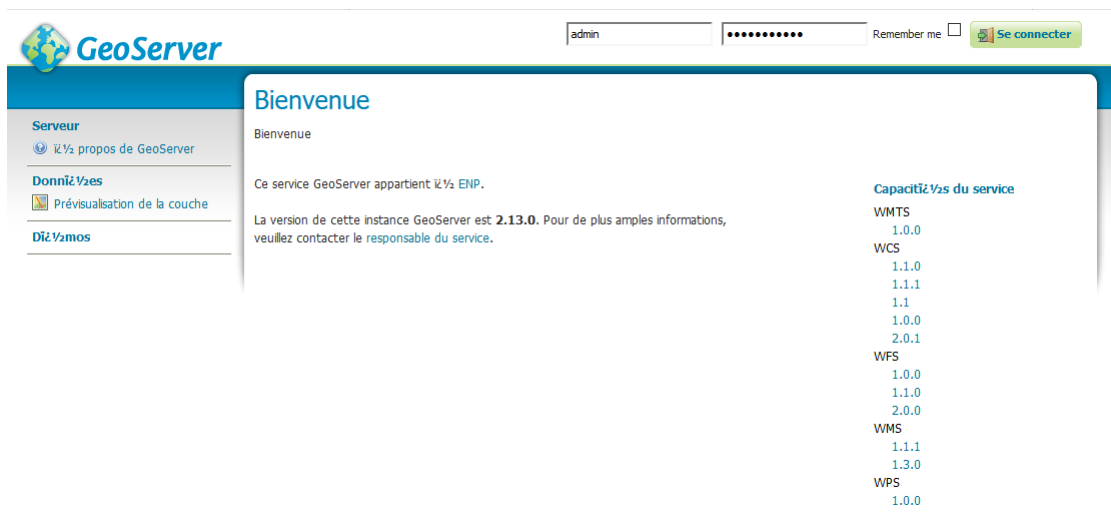


Figure 7.3: GeoServer home page

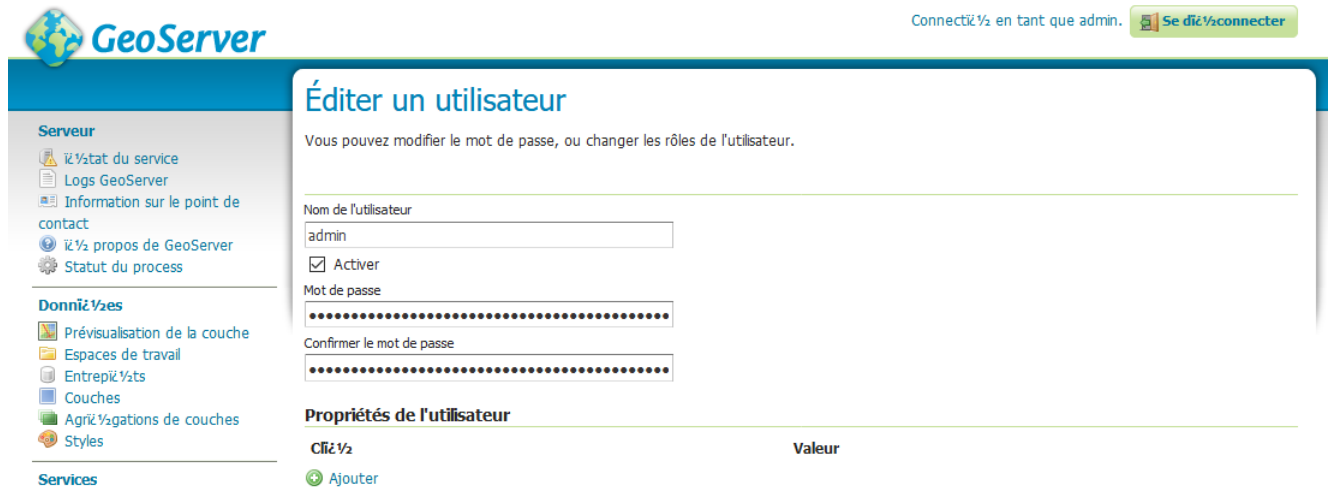
Connect using the default username and password (admin, geoserver)

2.3.2 Implement Basic Security

The new interface will show you some warning about security issues:

- ⚠ Please read the file `/opt/apache-tomcat-7.0.27/webapps/geoserver/data/security/masterpw.info` and remove it afterwards. This file is a **security risk**.
- ⚠ Please remove the file `/opt/apache-tomcat-7.0.27/webapps/geoserver/data/security/users.properties.old` because it contains user passwords in plain text. This file is a **security risk**.
- ⚠ The default user/group service should use digest password encoding.
- ⚠ The administrator password for this server has not been changed from the default. It is **highly** recommended that you change it now. [Change it](#)
- ⚠ No strong cryptography available, installation of the unrestricted policy jar files is recommended

- Change the default password for the administrator. Click on the Change it link on the left-hand side of the warning.



- The `users.properties.old` file is a security risk because it contains user passwords in plain text. GeoServer does not need it so it's safe to delete it:

```
sudo rm /opt/tomcat/apache-tomcat-8.5.5/webapps/geoserver/data/security/users.properties.old
```

- Now open the `masterpw.info` file. It contains the password generated by GeoServer for the root user. Store it in a secure place and delete the file.

```
sudo rm /opt/tomcat/apache-tomcat-8.5.5/webapps/geoserver/data/security/masterpw.info
```

2.3.3 Configuring multiple instance in a single server

1. Stop the Tomcat service:

```
sudo systemctl stop tomcat
```

2. Move the configuration folder to an external location:

```
sudo mv /opt/tomcat/apache-tomcat-8.5.5/webapps/geoserver/data
/mnt/share/geoserver_data
```

3. Edit the tomcat.service file to make GeoServer aware of the new configuration folder:

```
sudo vi /etc/systemctl/system/tomcat.service
Environment='GEOSERVER_DATA_DIR=/mnt/share/geoserver-data'
Environment='GEOSERVER_LOG_LOCATION=$GEOSERVER_DATA_DIR/logs/
geoserver-1.log'
```

4. Save the file and then restart the Tomcat service:

```
sudo systemctl restart tomcat
```

5. Check that the configuration was properly read.

6. Again, stop the Tomcat service, and copy it to a new location:

```
sudo cp -r /opt/tomcat /opt/tomcat2
```

7. Open the server.xml file of the new Tomcat and modify the server ports:

```
sudo vi /opt/tomcat2/apache-tomcat-8.5.5/conf/server.xml
<Server port="8105" shutdown="SHUTDOWN">
<Connector port="8180" protocol="HTTP/1.1"
connectionTimeout="20000" redirectPort="8443" />
<Connector port="8109" protocol="AJP/1.3"
redirectPort="8443" />
```

8. Create another a systemd Service File to manage the new tomcat:

```
vi /etc/systemd/system/tomcat2.service
[Unit]
Description=Apache Tomcat Web Application Container
After=network.target

[Service]
Type=forking

Environment=JAVA_HOME=/usr/java/jre1.8.0_161/
Environment=CATALINA_PID=/opt/tomcat2/apache-tomcat-8.5.5/
temp/tomcat.pid
Environment=CATALINA_HOME=/opt/tomcat2/apache-tomcat-8.5.5/
Environment=CATALINA_BASE=/opt/tomcat2/apache-tomcat-8.5.5/
Environment='CATALINA_OPTS=-Xms512M -Xmx1024M -server
-XX:+UseParallelGC'
Environment='JAVA_OPTS=-Djava.awt.headless=true
-Djava.security.egd=file:/dev/./urandom'
Environment='GEOSERVER_DATA_DIR=/mnt/share/geoserver-data'
```

```
Environment='GEOSERVER_LOG_LOCATION=\$GEOSERVER_DATA_DIR/
logs/geoserver-2.log'
ExecStart=/opt/tomcat2/apache-tomcat-8.5.5/bin/startup.sh
ExecStop=/opt/tomcat2/apache-tomcat-8.5.5/bin/shutdown.sh
```

```
User=tomcat
Group=tomcat
UMask=0007
RestartSec=10
Restart=always
```

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

We have two instances of GeoServer running on our server and sharing a common data directory. You can test it by opening a web browser and visiting the following:
GeoServer instance 1: [http://\[your server address\]:8080/geoserver](http://[your server address]:8080/geoserver)
GeoServer instance 2: [http://\[your server address\]:8081/geoserver](http://[your server address]:8081/geoserver)

2.4 Install Load Balancer

1. Install apache2

```
sudo apt-get install apache2
sudo service apache2 start
```

2. In the web browser enter you ip adress to verify that apache2 is running.

3 Installation of PostgreSQL

- Add the official PostgreSQL Apt Repository

```
sudo add-apt-repository "deb http://apt.postgresql.org/
pub/repos/apt/ xenial-pgdg main"
```

- Import the relevant signing key

```
wget --quiet -O - https://www.postgresql.org/media/
keys/ACCC4CF8.asc | sudo apt-key add -
```

- Update your packages


```
sudo apt update
```

- Install PostgreSQL and the “contrib” package

```
sudo apt install postgresql postgresql-contrib
```

- Check your PostgreSQL Version

```
psql --version
```



Apache2 Ubuntu Default Page

ubuntu

It works!

This is the default welcome page used to test the correct operation of the Apache2 server after installation on Ubuntu systems. It is based on the equivalent page on Debian, from which the Ubuntu Apache packaging is derived. If you can read this page, it means that the Apache HTTP server installed at this site is working properly. You should **replace this file** (located at `/var/www/html/index.html`) before continuing to operate your HTTP server.

If you are a normal user of this web site and don't know what this page is about, this probably means that the site is currently unavailable due to maintenance. If the problem persists, please contact the site's administrator.

Configuration Overview

Ubuntu's Apache2 default configuration is different from the upstream default configuration, and split into several files optimized for interaction with Ubuntu tools. The configuration system is **fully documented in `/usr/share/doc/apache2/README.Debian.gz`**. Refer to this for the full documentation. Documentation for the web server itself can be found by accessing the **manual** if the `apache2-doc` package was installed on this server.

The configuration layout for an Apache2 web server installation on Ubuntu systems is as follows:

```
/etc/apache2/
|-- apache2.conf
|   |-- ports.conf
|-- mods-enabled
|   |-- *.load
|   |-- *.conf
|-- conf-enabled
|   |-- *.conf
|-- sites-enabled
|   |-- *.conf
```

- `apache2.conf` is the main configuration file. It puts the pieces together by including all remaining configuration files when starting up the web server.

Figure 7.4: Apache2 Ubuntu Default Page

4 Installation of Postgis

- Add UbuntuGIS-stable repository and update packages.

```
sudo add-apt-repository ppa:ubuntugis/ubuntugis-stable
sudo apt update
```

- Install Postgis

```
sudo apt install postgis postgresql-10-postgis-2.4
```

5 Installation of Greenplum Pivotal 5.7 onCentOS 64-bit 7.x

Greenplum requires some specifications for servers intended to support Greenplum Database on Linux systems in a production environment. All servers in your Greenplum Database system must have the same hardware and software configuration:

1. Operation system: Cenots 64 bits 6.x or 7.x
2. File Systems: xfs
3. Minimum Memory:16 GB RAM per server
4. Disk Requirements:
 - (a) 150MB per host for Greenplum installation
 - (b) 300MB per segment instance for meta data
 - (c) Appropriate free space for data with disks at no more than 70% capacity
5. Network Requirements: 10 Gigabit Ethernet within the array
6. Software and Utilities: zlib, bash shell, GNU tars, GNU zip, GNU sed, perl, secure shell.

5.1 Setting the Greenplum Recommended OS Parameters (Master Only)

Greenplum requires the certain Linux operating system (OS) parameters be set on all hosts in your Greenplum Database system (masters and segments).

5.1.1 Linux System Settings

1. Edit the `/etc/hosts` file and include the host names and all interface address names for every machine participating in your Greenplum Database system.
2. Set the following parameters in the `/etc/sysctl.conf` file and reboot:


```
kernel.shmmax = 500000000
kernel.shmmni = 4096
kernel.shmall = 4000000000
kernel.sem = 250 512000 100 2048
kernel.sysrq = 1
kernel.core_uses_pid = 1
kernel.msgmnb = 65536
kernel.msgmax = 65536
kernel.msgmni = 2048
net.ipv4.tcp_syncookies = 1
net.ipv4.conf.default.accept_source_route = 0
net.ipv4.tcp_tw_recycle = 1
net.ipv4.tcp_max_syn_backlog = 4096
net.ipv4.conf.all.arp_filter = 1
net.ipv4.ip_local_port_range = 10000 65535
net.core.netdev_max_backlog = 10000
net.core.rmem_max = 2097152
net.core.wmem_max = 2097152
vm.overcommit_memory = 2
```

3. Set the following parameters in the `/etc/security/limits.conf` file:

```
* soft nofile 65536
* hard nofile 65536
* soft nproc 131072
* hard nproc 131072
```

5.1.2 Creating the Greenplum Database Administrative User Account

You must create a dedicated operating system user account on the master node to run Greenplum Database. This user account is named, by convention, `gadmin` and it must have permission to access the services and directories required to install and run Greenplum Database.

```
# groupadd gadmin
# useradd gadmin -g gadmin
# passwd gadmin
```

And introduce the new password.

5.1.3 Installing the Greenplum Database Software

1. Log in as root on the master machine.
2. Download the RPM distribution file to the master host machine from Greenplum network webpage and unzip it.
3. Launch the installer using bash:

```
rpm -Uvh ./greenplum-db-5.7.0-rhel7-x86_64.rpm
```

The RPM installation copies the Greenplum Database software into a version-specific directory, `/usr/local/greenplum-db-5.7`.

4. Change the ownership and group of the installed files to `gadmin`:

```
# chown -R gadmin /usr/local/greenplum*
# chgrp -R gadmin /usr/local/greenplum*
```

5.2 Installing and Configuring Greenplum on all Hosts

1. Log in to the master host as root:

```
\$ su -
```

2. Source the path file from your master host's Greenplum Database installation directory:

```
# source /usr/local/greenplum-db/greenplum_path.sh
111
```

3. Create a file called `hostfile_exkeys` that has the machine configured host names for each host in your Greenplum system (master and segments) by checking the correct host name in `/etc/hosts`. The names below are personal choices:

```
greenplum-master-1
greenplum-slave-1
greenplum-slave-2
greenplum-slave-3
```

4. Run as root, `gpsegininstall` that copies the Greenplum Database installation from the current host and installs it on a list of specified hosts, creates the Greenplum operating system user account (`gadmin`), sets the account password (default: `changeme`), sets the ownership of the Greenplum Database installation directory, and exchanges ssh keys between all specified host address names (both as root and as the specified user account).

```
# gpsegininstall -f hostfile_exkeys
```

To confirm the Greenplum software was installed and configured correctly, run the following confirmation steps from your Greenplum master host:

1. Log in to the master host as `gadmin`:

```
\$ su - gadmin
```

2. Source the path file from Greenplum Database installation directory:

```
# source /usr/local/greenplum-db/greenplum_path.sh
```

3. Use the `gpssh` utility to see if you can login to all hosts without a password prompt, and to confirm that the Greenplum software was installed on all hosts using the `hostfile_exkeys` file you used for installation.

```
\$ gpssh -f hostfile_exkeys -e ls -l \${GPHOME}
```

If the installation was successful, you should be able to log in to all hosts without a password prompt.

5.3 Installing Greenplum Database Extensions

We managed to install the PostGIS extension in Greenplum:

```
source /usr/local/greenplum-db-5.7.0/greenplum-path.sh
sudo yum install json-c-devel -y
sudo yum install geos-devel -y
sudo yum install -y proj-devel
sudo yum install git
git clone https://github.com/greenplum-db/geospatial.git
sudo yum install gcc.x86_64
sudo yum install libpqxx.x86_64 libpqxx-devel.x86_64
sudo yum install libxml2-devel.x86_64
./configure --with-pgconfig=/usr/local/greenplum-db-5.7.0/bin/
pg_config --without-raster --without-topology --prefix=\$GPHOME
make USE_PGXS=1 clean all install
sudo yum install byacc --enablerepo=base
sudo yum groupinstall "Development tool"
```

Annex B: GeoServer Administration Interface

There are three main areas in the GeoServer web interface:

- **The central area** is where information is shown; elements inside it change according to the operation you are performing. Just after you log on, it shows you a briefing of configured data, and warning or errors that you should correct.
- **On the right-hand side**, there is a list showing GeoServer capabilities. The listed acronyms refer to standard OGC protocols.
- **On the left-hand side**, there is a table of contents listing the configuration areas. Each area contains links to administrative operations. When you click on one of them, the central area shows you contextual options.

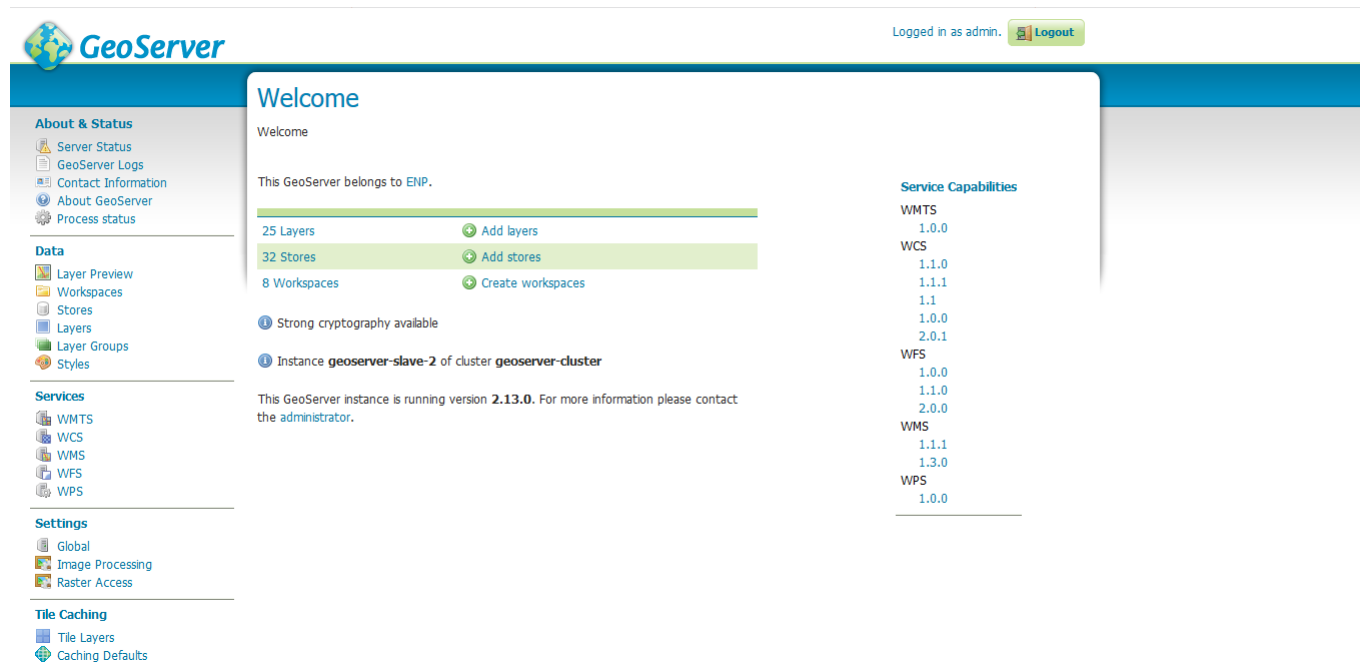


Figure 7.5: GeoServer’s User Interface

1 About Status

This area contains information about runtime variables and how GeoServer is described to clients that connect to it.

- **Server Status:** Information about the data directory, memory usage, Java version,...
- **Geoserver Logs:** Explore the error and warnings contained in the log file.
- **Contact Information:** Information on the organization and managers of Geoserver to allow users contact them
- **About Geoserver:** General information about the version of geoserver and links to documentation
- **Process Status:** List of all the running and recently completed processes.

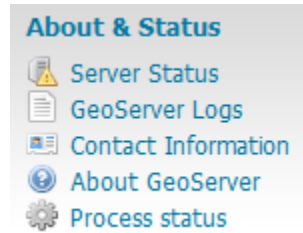


Figure 7.6: About and Status

1.1 Data

This area contains links to the data configuration engine

- **the Layer Preview:** See the published layers in various formats for each. (openlayers, KML, Geojson, GeoTIFF ..)
- **Workspaces:** to organize the layers in groups
- **Stores:**Configure the stores that provide data to Geoserver.
- **Layers:** Set necessary configuration to publish the data.
- **Layers group:** Define and manage layer groupings
- **Styles** describes how a feature type has to be drawn.

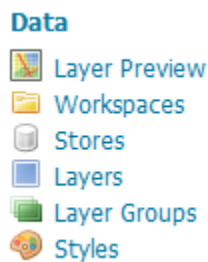


Figure 7.7: Data

2 Services

Geoserver exposes the data through standard services. This section contains the general configuration for each service (WMS, WFS, WPS..).



Figure 7.8: Services

3 Settings

This area contains general configuration parameters of Geoserver.

- **Global:** Settings that apply to all OGC services and control the internal behavior of GeoServer.
- **Image Processing:** Administer settings for Java Advanced image processing and raster encoding.
- **Raster Access:** Administer settings related to accessing raster data.

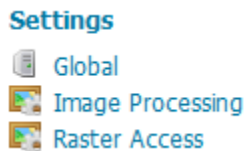


Figure 7.9: Settings

4 Tile Caching

This area contains some configuration parameters of the integrated GeoWebCache, that improves the performance of the server

- **Tile Layers:** Manage and list the cached layers.
- **Caching Defaults:** Configure the global settings.
- **Gridsets:** Manage the available tiling schemas or create a new one
- **Disk Quota and BlobStores:** Configure the disk limits for layers.

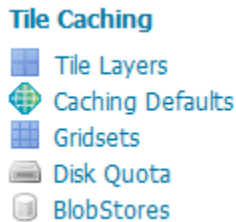


Figure 7.10: Tile Caching

5 Security

This area contains the configuration parameters of Geoserver security subsystem, it allows the creation of users and roles and binding them with the published data and services.

- **Settings:** Configure security settings
- **Authentication:** Authentication providers and settings
- **Passwords:** Password settings
- **Users, Groups, Roles:** Manage and list users, groups and role services
- **Data:** Manage data security.
- **Services:** Manage service security
- **WPS security:** Manage the processing service security configuration

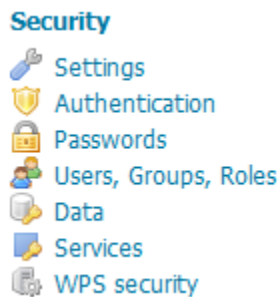


Figure 7.11: Security

6 Demos

A few demo applications are included with GeoServer.

- WPS request builder
- Demo requests
- SRS List

- [Reprojection console](#)
- [WCS request builder](#)

GeoServer Demos

Collection of GeoServer demo applications

- [WPS request builder](#) Step by step WPS request builder
- [Demo requests](#) Example requests for GeoServer (using the TestServlet).
- [SRS List](#) List of all SRS known to GeoServer
- [Reprojection console](#) Simple coordinate reprojection tool
- [WCS request builder](#) Step by step WCS GetCoverage request builder

Figure 7.12: Demos

Annex C: Scripts

1 Watchdog Script

```
# Declare instance related variables
INSTANCE=1
HTTP_PORT=8080
# Check the command-line arguments
if [ ! $# -eq 4 ]; then
# Insufficient arguments passed to the script
echo >&2 "usage: $0 -i [instance_number] -p [http_port]"
    exit 1
else
# Correct number of arguments
    while [ $# -gt 0 ]; do
        case "$1" in
            -i) INSTANCE=$2; shift;;
            -p) HTTP_PORT=$2; shift;;
            -*) echo >&2 \
                "usage: $0 -i [instance_number] -p [http_port]"
                exit 1;;
            *) break;;
        esac
        shift
    done
fi
# Declare other variables
PID= cat /opt/tomcat${INSTANCE}/apache-tomcat-8.5.5/temp/tomcat.pid

HTTP_URL=http://localhost:${HTTP_PORT}/geoserver/openlayers/img/
west-mini.png

TOMCAT_SERVICE=tomcat${INSTANCE}
# Set the logfile to stdout if the file does not exist

LOG_FILE=/mnt/share/geoserverdata/logs/logs-1/watchdog-${INSTANCE}.log
if [ ! -e "${LOG_FILE}" ]; then
    LOG_FILE="/dev/stdout"
fi
if [ -d /proc/$PID ]; then
```

```

# Tomcat is running so we need to check it is responding to requests
echo "'date' WatchDog Status: Tomcat service ${TOMCAT_SERVICE}
is running" >> "${LOG_FILE}"
wget $HTTP_URL -T 1 --timeout 20 -O /dev/null &> /dev/null
if [ $? -ne "0" ]; then
# HTTP not responding, restart service

echo "'date' WatchDog Status: Tomcat service ${TOMCAT_SERVICE} is
not responding to HTTP requests"

echo "'date' WatchDog Action: Restarting Tomcat
service ${TOMCAT_SERVICE}" >> "${LOG_FILE}"
service $TOMCAT_SERVICE restart >> "${LOG_FILE}"
wget $HTTP_URL -T 1 --timeout 20 -O /dev/null &> /dev/null
    if [ $? -ne "0" ]; then
#Send an alert mail
echo "Geoserver is not responding, verify the log files for more
informations" | mail -s "Alert" yasmine.benlefki@g.enp.edu.dz
else
#Send a warning mail
echo "Geoserver was not responding so tomcat has been restarted,
verify the log file for more
informations verify the log file
for more informations" | mail -s "warning" myemail@watchdog.com

fi
else
# HTTP is responding
echo "'date' WatchDog Status: Response OK - Tomcat service
${TOMCAT_SERVICE} is responding to HTTP requests" >> "${LOG_FILE}"
    fi
else
# Tomcat is not running, restart the service
echo "'date' WatchDog Status: Tomcat service
${TOMCAT_SERVICE} process appears to be dead" >> "${LOG_FILE}"
echo "'date' WatchDog Action: Restarting Tomcat service ${TOMCAT_SERVICE}"
service
${TOMCAT_SERVICE} restart >> "${LOG_FILE}"
if [ -d /proc/$PID ]; then
#Send an alert mail

echo "Tomcat couldn't be restarted, verify the log file for more
informations" | mail -s "Alert" email@watchdog.com
else
#Send a warning mail
echo "Tomcat has been restarted, verify the log file for more
informations | mail -s "Warning" email@watchdog.com
fi

```

```
fi
```

2 Python scripts:

2.1 Open data sensors

```
import os
import simplejson as json

i=8 #number of sensors
#define arrays of longitude and latitude
longitude=[36.60507,36.66138,36.8678,36.64078,36.69914,36.58791,
36.714170,36.788722]
latitude=[3.11,2.88254,2.77954,2.93541,2.9512,2.95052,3.021964,
3.015700]
j=0
#Create ...
json2= {"sensors":[{"sensor": "temperature-"+str(j+1),
    "description": "Temperature ambiente",
    "type": "temperature",
    "dataType": "number",
    "unit":"C",
    "component":"temperature-"+str(j+1),
    "componentType": "temperature",
    "location": str(longitude[j])+" "+str(latitude[j]),
    "timeZone": "CET"}
        for j in range (i)]}
j=0
json2=json.dumps(json2)
#create the sensors using curl request
C="curl -X POST -H \'Content-Type:application/json\'
-H \'IDENTITY_KEY:identity_key\'
-d \' " +json2+ " \' http://192.168.20.18:8081/catalog/ANM"
os.system(C)
```

2.2 Simulated sensors

```
import os
from os.path import exists
import simplejson as json
import random
import pprint as pprint
import numpy as np
from datetime import datetime
from random import randint
#ofdata
k=15
```

```

year="2018"
month=np.random.randint(low=1, high=4, size=(k,))
day=np.random.randint(low=1, high=28, size=(k,))
hour=np.random.randint(low=0, high=24, size=(k,))
minute=np.random.randint(low=0, high=60, size=(k,))
second=np.random.randint(low=0, high=60, size=(k,))
i=0
timestamp=[]
[timestamp.append(str(day[i])+"/"+str(month[i])+"/"+year+"T"+
str(hour[i])+":"+str(minute[i])+":"+str(second[i])+"CET") for i
in range(k)]

i=50 #number of sensors

longitude=np.random.uniform(low=36.6266, high=36.742, size=(i,))
latitude=np.random.uniform(low=2.752, high=3.2032, size=(i,))
j=0
sensors= {"sensors":[{"sensor": "Station_Velib-"+str(j),
"description": "Velibs disponibles",
"type": "velib",
"dataType": "number",
"component": "Station_Velib-"+str(j),
"componentType": "velib",
"location": str(longitude[j])+" "+str(latitude[j]),
"timeZone": "CET"}
for j in range (i)]}

j=0
a=0
messages={"sensors":[{"
"sensor": "Station_Velib-"+ str(j),
"observations":[
{"value": randint(0,20),
"timestamp": timestamp[a],
"location": str(longitude[j])+" "+str(latitude[j])}]
for a in range(k) ]}
for j in range (i)]}

sensors=json.dumps(sensors)
messages=json.dumps(messages)
C="curl -X POST -H \'Content-Type: application/json\'
-H \'IDENTITY_KEY:identity_key\' -d \' " +sensors+
" \' http://ip-address:8081/catalog/ANT"
os.system(C)
M="curl -X PUT -H \'Content-Type: application/json\'
-H \'IDENTITY_KEY:identity_key\' -d \' " +messages+
" \' http://ip-address:8081/data/ANT"
os.system(M)

```