POLYTECHNIC UNIVERSITY OF TURIN

DEPARTMENT OF CONTROL AND COMPUTER ENGINEERING

Master of science program Computer engineering

Educational Path - Software

Master's degree thesis

Development of a custom-made Network Infrastructure with End-User Services and a hyper-converged NLP Assistant

Supervisor **Candidate**

Prof. Elio Piccolo Marco Schiapparelli

December 2018

Table of Contents

Chapter 1: Introduction

1.1 Introduction

The main purpose of this project is to setup a complete **Network Infrastructure** providing advanced end-user services, with the addition of an innovative **NLP assistant** for system management, which goes under the name of **David**.

This thesis will serve as a detailed report for the creation of the entire system, divided in three parts: Network (*Chapter 2*), Virtual Farm (*Chapter 3*) and David itself (*Chapter 4*).

This order actually follows the same order of the project development. In fact, the main goal was to provide the basic network access. Later on, many services were introduced to improve the user experience. Finally, the NLP assistant was created to simplify everyday operations of system management. This workflow is presented in the *Figure 1.1* below.

This entire project is a **no-profit** work, created by students to apply the notions learned during university courses to a practical and challenging project.

Being it a free project, the entire creation and implementation was guided by **passion and curiosity**, resulting in a very custom environment were a lot of services has been integrated as desired to create the actual system.

This project goes under the name of **Sala Olimpo**, and it will be briefly described in this chapter, while the next ones will provide further details about each component of the architecture.

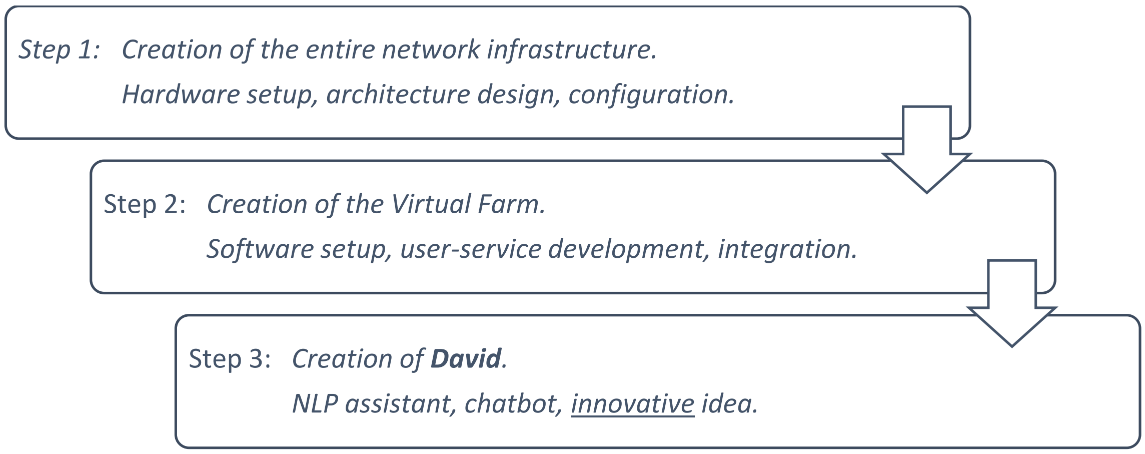

Step 1: Creation of the entire network infrastructure.
Hardware setup, architecture design, configuration.
Step 2: Creation of the Virtual Farm.
Software setup, user-service development, integration.
Step 3: Creation of David.
NLP assistant, chatbot, innovative idea.

Figure 1.1: The general workflow.

1.2 The Network Issue

The first task to be carried out was the creation of the **Network Infrastructure**. In fact, the first aim of the project was the development of a complete Server Room, able to provide internet access to all the users.

The entire network infrastructure was developed in a fully custom fashion since it was created to perfectly fit the building, composed by many floors and hallways. This task turned out to be really challenging, as it required the use of many network devices, especially from the **MikroTik** brand.

Designing the general architecture required a lot of **foresight**, and many improvements were made step by step before reaching the current state. A lot of issues were encountered during this part, especially due to limited founds.

A rack was built in the server room, which became the core component for the network infrastructure, containing all the needed products.

Correctly sizing and integrating this set of devices was a crucial task for this part of the project. In addition, they must be easily managed, and possibly in a centralized way.

Another important part of the system is the **security** aspect. As described later, this issue was covered using an approach heavily based on **VLAN**. Not only this allowed an easy firewall configuration for the system, but also made it well-ordered and easy to manage and check for troubleshooting.

Several devices, like the **NAS** and the **UPS** were later introduced to cover other aspects of the system, like the protection from power outages and data loss. These goals are reached using ad-hoc configurations to better integrate with the system.

As later described, the design pattern adopted for the implementation heavily focus on creating a neat and scalable environment in order to be effectively managed by the admins.

Routes, queue and load balance are crucial for the network to work properly, so everyday adjustments allowed to **fine-tune** the performance of the system, resulting in a very good service which keeps evolving nowadays.

1.3 The Virtual Farm addition

After the accomplishment of the previous task, a set of services was created and implemented to improve the user experience and provide more useful features.

This improvement was performed by using a **centralized server** running a set of virtual machines, in other words, a **Virtual Farm**.

Services are introduced step by step thanks to an environment heavily based on virtualization and segmentation, providing several benefits like the possibility to take snapshots and test services in a sandbox.

The implementation followed a professional and mature workflow, especially with the introduction of many **development environments** to perform testing and tuning of the services. As described later, these development environments are three, named **Test (TEST)**, **Acceptance (ACC)** and **Production (PROD)**.

This practice allowed to reduce the disservice to the minimum, especially since promotion from an environment to another requires a lot of tests, like integration and performance ones.

Moreover, promotion of services follows a precise set of rules, in order to avoid unexpected issues during the deployment. In particular, each step of the configuration or development of a service in a virtual machine has been meticulously recorded into a **Wiki** page, used as a base for promotion to major environments like Acceptance and Production.

This development was really challenging, especially since gaining confidence with the Linux GUI-free environment took a while. From the beginning of the project, every Linux machine was created **with no graphical interface**, and configured by command line using the SSH protocol.

The final number of available services is pretty high, thanks to new ideas that came out every day, one after another, to keep improving.

1.3.1 The developed services

Here is reported a simple list of the workflow adopted, in order to justify all the implemented services. It is presented in chronological order.

- A list of users able to access the computers in the computer room is needed, so an **Active Directory** was created in Windows Server 2016. This is the only non-Linux machine.
- The domain join operation requires a **DNS** server, and having a centralized one would have been very useful. So, a machine with Bind was created.

- User **authentication** to the Wi-Fi avoids improper use of the system, so the Radius machine was built.
- Users may need a **print services**, and accounting is needed to make the user pay for it. So, a print server manager was developed.
- The print server, along with other services, requires a database to work. So, a centralized **MySQL database** was created.
- A free cloud service for the users was an interesting service, especially if linked with a free email account. So, a machine was created to host **NextCloud** and an email server.
- A **reverse proxy server** with SSL certificates is required for all these services to work with only one public IP. So, proxy with SSL offloading was hosted by a dedicated virtual machine.
- An **admin console** could be useful to manage the entire system, along with a **monitoring platform**. So, Rhea Admin Console was created along with the Grafana platform.
- An **UPS** was added, so a VMA was required to proper exploit it. So, RielloBox was created.
- Since a reverse proxy was added, many other **web-based services** could be added. So, VillaWeb was added as a container for those services.
- To solve some **permission issues** with NextCloud and the Windows workstations, a small samba server was introduced.

As can be seen, each virtual machine was created for a well-defined purpose.

This creation required a lot of effort, especially since many of those services are not meant to be integrated with others. This required the development of **orchestrators**, **connectors**, and other workaround like **custom sockets** to make the whole system work properly.

A high-level diagram of the entire architecture is shown in *Figure 1.2*.

No guides were followed to reach this goal, but only the documentation of each software or tools, many of those retrieved using the **man** Linux command or the comment placed by vendors on the configuration files.

The result is an **entirely custom system**, created to fulfil all the functional requirements of this use case. However, the administration of this system by not well trained admins may be difficult to achieve.

To solve this issue, an innovative idea was added to the system as a very special feature, described in the next paragraph.

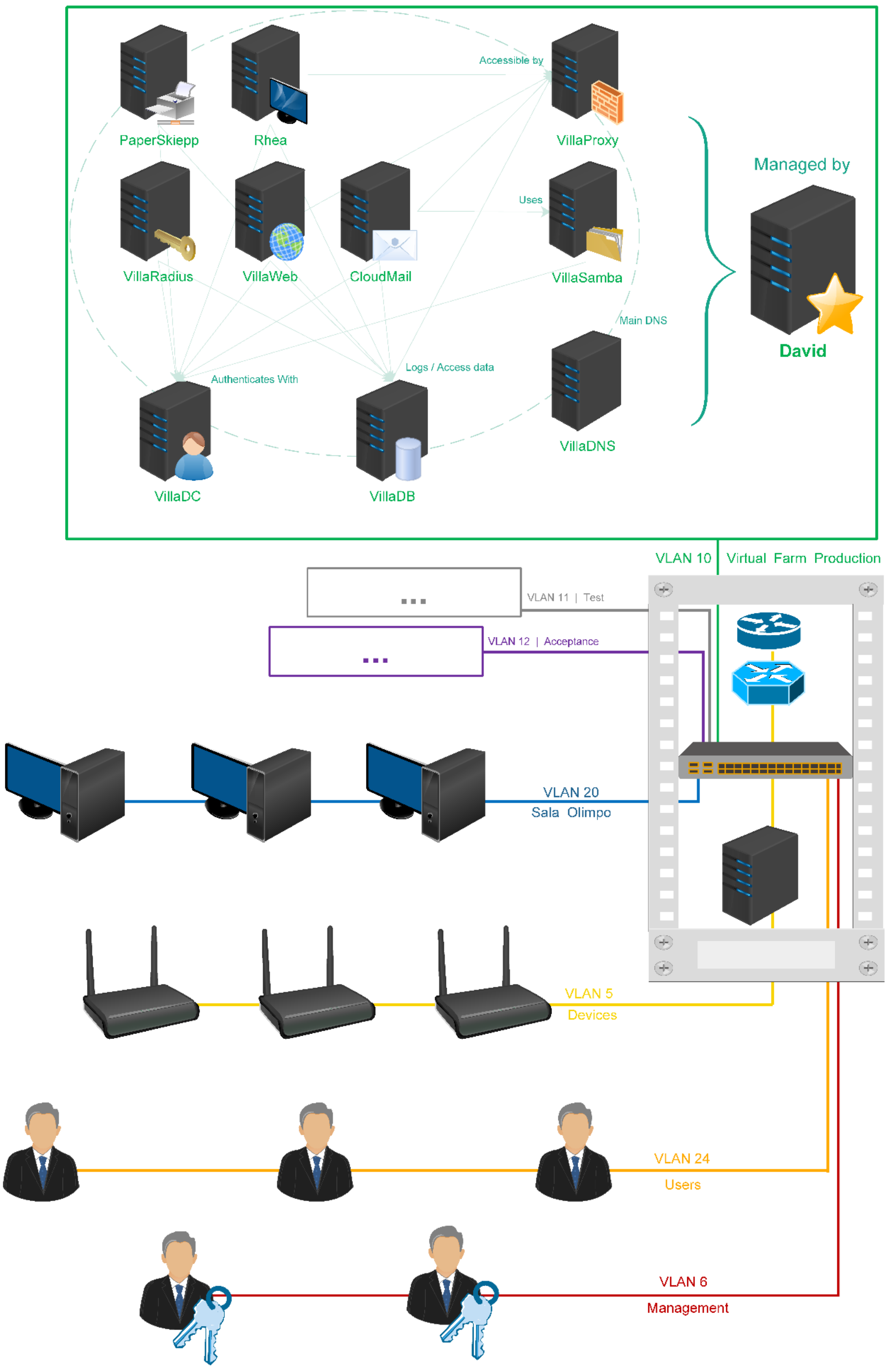


Figure 1.2: A high-level diagram of the architecture.

1.4 The innovative idea: David

After the conclusion of the Production Virtual Farm, the entire project could have been concluded, but the development of a new system feature was started instead.

This peculiar idea was born with the name of **David**: an artificial intelligence-based Virtual Assistant for network infrastructure and production virtual farm management.

The main goal of the project is the possibility to give inputs to this software by means of **Natural Language**(1), allowing natural conversation with the assistant using vocal or textual messages.

Using this approach, admins can interact with David without following a particular scheme, and even without knowing the syntax of the entire set of commands.

David can literally guide the admin to its set of features to help with the resolution of a problem or a fast diagnosis, even using charts and images as output. Moreover, all these functionalities can be exploited without writing a single line of code and without accessing all the components involved in the architecture.

To reach this goal, this virtual assistant interacts with many components, in order to accomplish different task regarding different services, software, hardware and configurations.

In other words, David is essentially a **hyper-converged software** for system management and lifecycle. As a centralized software, it can be exploited using different communication tools:

- The first one introduced was **Amazon Alexa**, a cloud-based voice service available on tens of millions of devices from Amazon and third-party manufacturers.
  By using this platform, the **Speech-To-Text** part is performed automatically, as a dedicated device is in charge of calling David.
  For this project, the Alexa service was developed using one **Echo Dot 2** device, and now it is exploited using also an **Echo Dot 3** device.
- The second one is a **Telegram** bot, as it is one of the most commonly-used instant messaging services.
  This choice made the service extremely portable, as it does not require dedicated hardware or specific client application to call David.
  In addition, this tool also allows to communicate using images as output.

(1) *In linguistics, a **natural language,** or ordinary language, is any language that has evolved naturally in humans through use and repetition without conscious planning or premeditation. Natural languages can take different forms, such as speech or signing. They are distinguished from constructed and formal languages such as those used to program computers or to study logic.*

All of these tools, software and communication channels are perfectly integrated in the final product, which exploits different services and adopts several methods in order to provide an optimal set of features.

Some of this tools and methods are graphically presented in the tag cloud of the *Figure 1.3* below.

Figure 1.3: A Tag Cloud representing some topic related to the development of David.

1.4.1 Alexa: Motivation and issues of the project

This project was created to help the everyday management of the infrastructure, and speed up some commonly performed operations.

Primarily, the goal was to create an automatic tool capable of performing system diagnosis and help troubleshooting. So, the **first prototype** was in charge of checking the status of active services and the network load.

Developed on a Tomcat server, this service was meant to be called using a graphical dashboard reachable via web application. In fact, the idea was more like a set of shortcut for executing a set of frequently used functionalities.

After this first approach, an **important decision** was made, i.e. to c**all this service using a vocal assistant like Alexa**. This drastically changed the requisites for the project, as the interaction with Alexa required an accurate assignment of keywords for all commands.

The main problem in this scenario was the need of correctly spelling all the commands along with their syntax. Moreover, the recognition of custom names like RielloBox was difficult, as Alexa Speech-To-Text does not allow customization.

To overcome this problems, an approach using **Natural Language Processing** was adopted. With this feature, David became able to understand the user commands, which are represented by an **Intents**.

Basically, each intent represents a specific functionality of David, and each sentence can be translated into a particular intent.

This calculation is made automatically by **Amazon Alexa**, by means of machine learning algorithms using neural networks to determine the result. As a machine learning algorithm, some example of phrases must be given to the tool to correctly build the model.

However, by providing only a couple of examples, the engine may not train it not sufficiently, and thus the intent recognition may result in a wrong result. Moreover, Amazon Alexa Service does not provide the option for easily building an entire dialogue structure to follow the user's request.

In order to provide this feature, a secondary intent recognition engine has been used: **Watson Assistant**. It is a service from the IBM Cloud Software and part of the Watson suite.

Basically, the entire system was set up to use Alexa only for the Speech-To-Text part, which is actually really good; while the intent recognition functionality is demanded entirely to Watson.

Not only several benchmarks demonstrated that Watson can perform a better intent and entity recognition, but it also allows to create an entire dialogue system. Intent, entities involved and context variables allow to guide users to a specific function to be performed. Moreover, Watson has an incredible capability of **fuzzy-matching** detection over entities, making the recognition of the customized ones possible. The general workflow is presented in *Figure 1.4*.

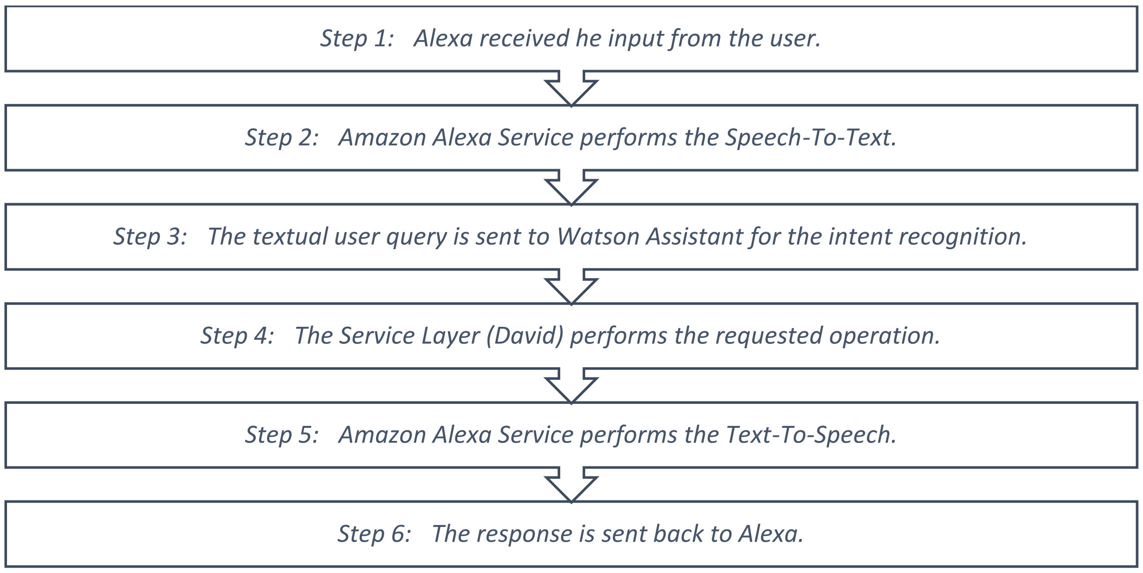

Step 1: Alexa received he input from the user.
Step 2: Amazon Alexa Service performs the Speech-To-Text.
Step 3: The textual user query is sent to Watson Assistant for the intent recognition.
Step 4: The Service Layer (David) performs the requested operation.
Step 5: Amazon Alexa Service performs the Text-To-Speech.
Step 6: The response is sent back to Alexa.

Figure 1.4: The workflow for the Alexa architecture.

1.4.2 Telegram: A further step for the project

One of the biggest lacks of the Alexa system is the impossibility to **communicate using graphical representation of data**, such as graphs or measurements.

In addition, the **portability** is another big issue. The needs for a dedicated hardware, always connected to a network and to a power source, does not allow to create a portable service for the infrastructure.

Moreover, the accountability is highly difficult to obtain, since the same device is used by different admins and it is impossible to keep track of all of them without using some esoteric and complicated configuration.

To overcome this set of problems, the David service was made usable by means of a Telegram bot, with the account name *`@salaolimpo_bot`*.

Same as the Alexa system, this architecture uses Watson Assistant to perform intent and entity recognition. In addition, the entity match for this system performs significantly better than the Alexa one, as messages comes already in textual form.

To help the creation of the Telegram bot, a very special software originally developed by IBM is used. This software is called **Node-RED**. Written in JavaScript, this tool is essentially a flow-based development tool, particularly useful to wire together devices, APIs and online services.

It works by means of flows, each one representing one or more functionalities to provide. For this case, the task is to handle the message set to the bot, in order to correctly perform the intent recognition and the relative action.

Of course, the same Service Layer of the previous architecture is used in this case, as the set of functionalities is pretty much the same, with the exception for visual output and other features like notifications.

The basic workflow is shown in *Figure 1.5*.

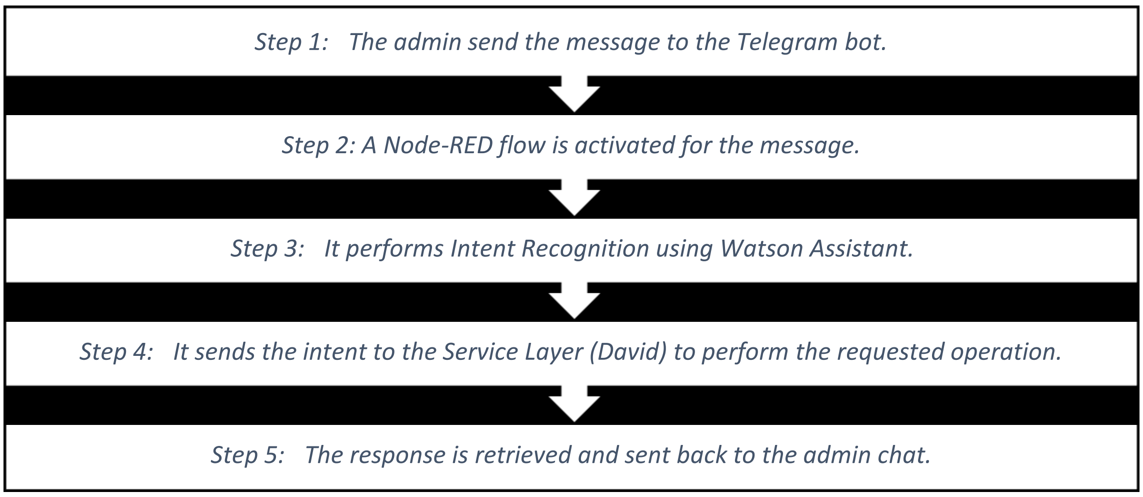

Step 1: The admin send the message to the Telegram bot.
Step 2: A Node-RED flow is activated for the message.
Step 3: It performs Intent Recognition using Watson Assistant.
Step 4: It sends the intent to the Service Layer (David) to perform the requested operation.
Step 5: The response is retrieved and sent back to the admin chat.

Figure 1.5: The workflow for the Telegram bot.

Several improvements were introduced on the chatbot, in particular:

- An **authentication mechanism** is added to the bot. This grants access to the bot only to the system admins, as accessing the bot requires a password. This is actually a custom alternative to the standard authentication method natively implemented by the BotFather of telegram.
- **Vocal messages** can be sent as input to the bot, and in addition it can produce audio responses as output.
  This particular feature is integrated using other services of the IBM Bluemix platform, named Text-To-Speech and Speech-To-Text.
  The first one translates the user vocal notes into text, sent later normally to Watson Assistant. Instead, the second one performs the opposite task, translating an output from David's Service Layer into an audio file.
  These two services perfectly integrate directly with the Node-RED platform; as specific nodes are available to interact with the entire Watson suite.
- A **subscription** can be done to receive notification from the bot. Examples of notifications are resource problems for the Virtual Farm, high temperature for the system, and similar.
  There are four levels for the notification (Critical, Warning, Info, None), which can be set for each subscription.
  This is very useful as immediately informs the admins about a possible problem in the system, allowing a fast diagnosis and prevention of issues.

As later further described, the actual product is adaptable and simple to use.

However, this is actually a POC of what David can become in the future. Lots of features can be added, and lot of operations implemented.

This software actually required a lot of time to be implemented, and effort for the correct integration within the system. Every feature was added separately in order to experiment a functionality or test a new possibility.

As a team project, David will hopefully keep evolving in the future, integrating many new operations and becoming an even more complete system.

For example, new channels for the input could be added along with more different useful functionalities; all of this to highlight even more the hyper-converged nature of this *very special project.*

Chapter 2: Network

2.1 Introduction

This chapter covers all the network set up, in terms of hardware, software and protocols used. The following functional requirements will justify several technical choices in the following chapters:

- The network must serve at least 150 users at the same time.
- Internet access must be provided via Wi-Fi in the entire building.
- Users must be authenticated when using Wi-Fi.
- A computer room is disposed to provide print services and standard computer utilities.
- The access to management infrastructures must be denied to regular users.
- Some services must be accessible from outside the building.
- A DMZ is required to isolate the exposed services.

A top-level design of the system is given in the *Figure 2.1*.

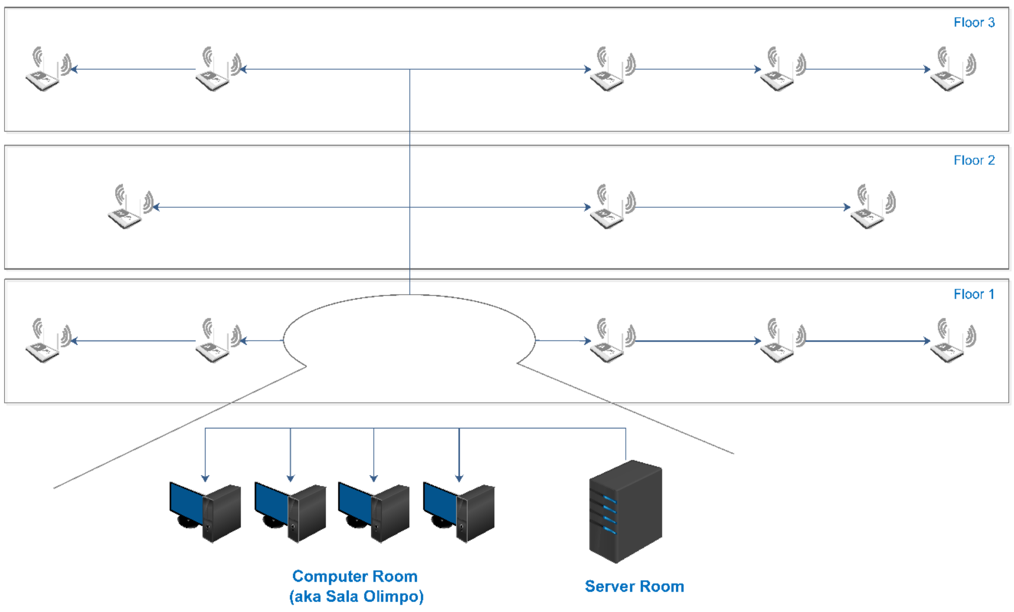


Figure 2.6: Top-level diagram of the Network.

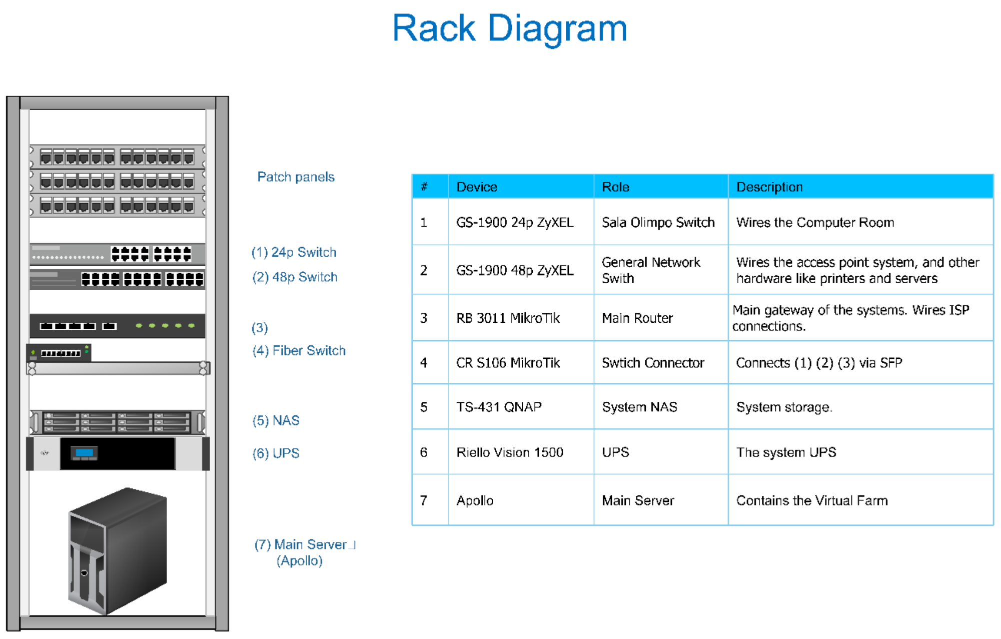
As can be seen, the architecture is composed by a central Server Room, containing a rack with the main network equipment, a computer Room (named **Sala Olimpo**) and many access points providing Wi-Fi access to users.

2.2 Hardware Setup

2.2.1 The rack

As mentioned in the previous paragraph, the server room contains a **rack**, the main core of the network system. As described in *Figure 2.2*, it contains:

- A 24-ports GS-1900 ZyXEL Switch
- A 48-ports GS-1900 ZyXEL Switch
- A RB 3011 MikroTik RouterBoard
- A CR S106 MikroTik RouterBoard
- A TS-431 QNAP NAS
- A Riello Vision 1500 UPS
- The main server containing the Virtual Farm, named **Apollo**



#	Device	Role	Description
1	GS-1900 24p ZyXEL	Sala Olimpo Switch	Wires the Computer Room
2	GS-1900 48p ZyXEL	General Network Swith	Wires the access point system, and other hardware like printers and servers
3	RB 3011 MikroTik	Main Router	Main gateway of the systems. Wires ISP connections.
4	CR S106 MikroTik	Swtich Connector	Connects (1) (2) (3) via SFP
5	TS-431 QNAP	System NAS	System storage.
6	Riello Vision 1500	UPS	The system UPS
7	Apollo	Main Server	Contains the Virtual Farm

Figure 2.7: The main rack.

2.2.2 The main switches and routers

The ZyXEL 24 ports is used entirely to wire the *Sala Olimpo* (our computer room), which contains five computers, named *Demetra*, *Dioniso*, *Efesto*, *Ermes* and *Ares*.

Computer names, along with those of other hardware devices, are inspired by Greek mythology. As an example, the main server is named *Apollo* as it replaces the previous server named *Zeus*, and finally the administration computer used to configure the it is called *Prometheus*.

The remaining ports are used to provide internet access with Ethernet sockets in the computer room. The devices that are physically connected to those sockets form the **VLAN**(1) **20**.

To deal with VLANs, a switch port can be set as **untagged member** or **tagged member** for them. More specifically, it must be either:

- Untagged *for a single* VLAN: packets going through an untagged port are tag-free.
- Tagged *for one or more* VLANs: packets going through a tagged port must contain the VLAN tag in front, otherwise they will be discarded.

The switch is thus configured for accepting packets from VLAN 20, having all ports **untagged** for it. This allows traffic from this VLAN to be accepted without the VLAN tag.

A dedicated 24-ports was chosen in order to have a simple configuration and to apply specific policies for the computer room. An example is the **loop protection**, which is configured only in the 24-ports switch to avoid damage in case two sockets are connected through an Ethernet cable by unaware users.

The switch is then connected to the system by the MikroTik CR S106 routerboard, actually used as a simple **backbone fibre switch**, as shown in *Figure 2.3*. It has 5 **SFP** ports, filled with some Cisco SFP modules and some LC-LC fibre cables.

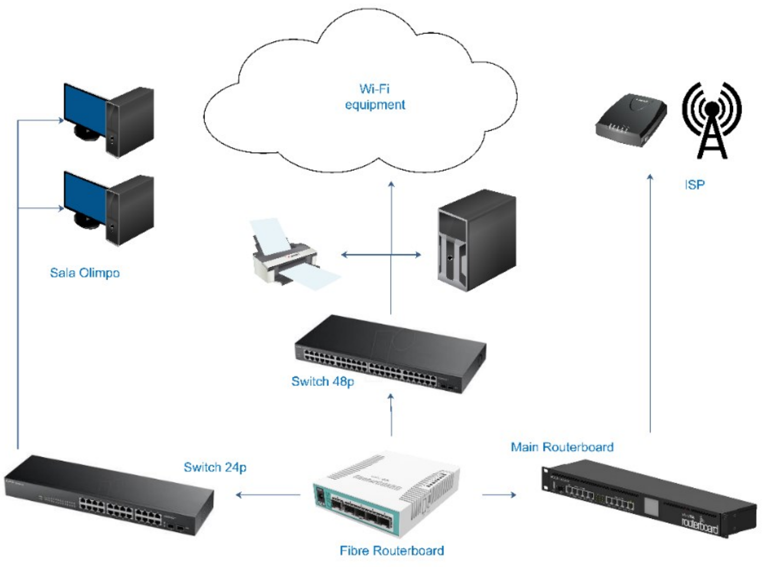


Figure 2.8: The Network backbone using the MikroTik CR S106 fibre switch.

(1) *A **Virtual LAN** is a logical subnetwork that can group together a collection of devices from different physical LANs. Each VLAN has one ID from 1 to 4096 (**VID**), which can be inserted in front of a network packet as a simple header named **tag**.*

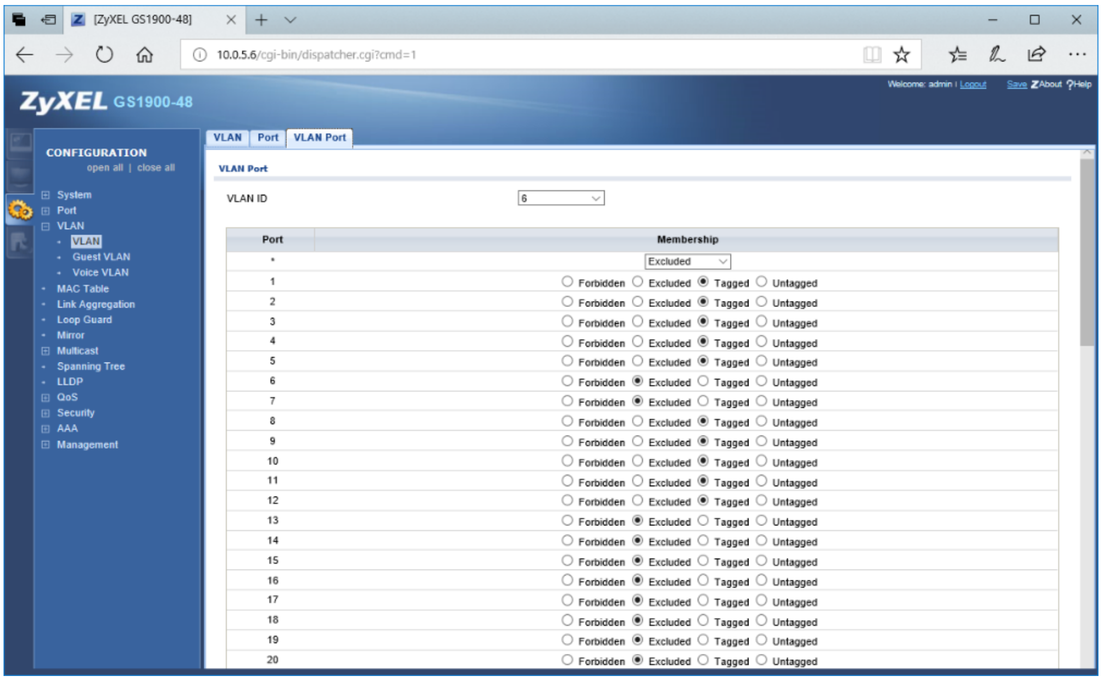
The introduction of this device avoids the employment of Ethernet ports on the other devices, and was a chance to experiment some interesting fibre network system. Moreover, a star point connection is simple to monitor and manage (even if it increases fault risks).

Being a **MikroTik** [1] hardware, it runs a version of **RouterOS**[(1)] [2]. However, we configured this routerboard to be used just as a L2 switch. This can easily be performed with the **bridge** menu in RouterOS, allowing the interconnection of hosts connected to separate LANs as if they were attached to a single L1AN.

This routerboard is also connected to the **ZyXEL** [3] 48 ports, the main switch of our system and all these objects belong to the VLAN 5, used precisely for network hardware.

The 48 ports switch connects many other devices; therefore, each port is configured ad-hoc for its purpose. In example, for the server connection, ports are configured as **tagged** since only packets with the VLAN tag are allowed to pass.

Some **LAG**[(2)] are configured to combine multiple network connections, increasing throughput and providing redundancy in case of failure. ZyXEL switches can be configured using the web interface, as shown in *Figure 2.4* [4].



VLAN Port

VLAN ID: 6

Port	Membership
*	Excluded
1	Forbidden Excluded [✓ Tagged] Untagged
2	Forbidden Excluded [✓ Tagged] Untagged
3	Forbidden Excluded [✓ Tagged] Untagged
4	Forbidden Excluded [✓ Tagged] Untagged
5	Forbidden Excluded [✓ Tagged] Untagged
6	Forbidden [✓ Excluded] Tagged Untagged
7	Forbidden [✓ Excluded] Tagged Untagged
8	Forbidden Excluded [✓ Tagged] Untagged
9	Forbidden Excluded [✓ Tagged] Untagged
10	Forbidden Excluded [✓ Tagged] Untagged
11	Forbidden Excluded [✓ Tagged] Untagged
12	Forbidden Excluded [✓ Tagged] Untagged
13	Forbidden [✓ Excluded] Tagged Untagged
14	Forbidden [✓ Excluded] Tagged Untagged
15	Forbidden [✓ Excluded] Tagged Untagged
16	Forbidden [✓ Excluded] Tagged Untagged
17	Forbidden [✓ Excluded] Tagged Untagged
18	Forbidden [✓ Excluded] Tagged Untagged
19	Forbidden [✓ Excluded] Tagged Untagged
20	Forbidden [✓ Excluded] Tagged Untagged

Figure 2.9: The web interface for ZyXEL switch configuration.

[(1)] ***RouterOS*** *is the operating system of all MikroTik routerboards. It can also be installed on a PC and will turn it into a router with all the necessary features - routing, firewall, bandwidth management, wireless access point, backhaul link, hotspot gateway, VPN server and more.*

[(2)] *A* ***LAG*** *(Link Aggregation Group) combines many physical ports together to make a single high-bandwidth data path.*

Example of connected systems and relative configuration are:

- A Canon C3520 printer, the main printer: port untagged (VLAN 10).
- A LAG for QNAP NAS: LAG untagged (VLAN 10)
- A LAG for Apollo (the main server): LAG tagged (VLAN 10)
- The Wi-Fi equipment, reached via Ethernet cables going to all the floors. Each port is configured as tagged for VLAN 6 (management) and VLAN 21, 22 and 24 (users VLAN).

Last but not least, a crucial device for this section is the RB 3011 MikroTik routerboard.

This is the core router, firewall and gateway of the entire system.

All the employed ISP lines are connected to its Ethernet ports. Those lines belong to different providers like *Telecom*, *Fastweb*, *BBBell* and *Eolo*.

These interfaces can be easily configured with **Winbox**(1) [5], assigning addresses to them and configuring a proper route (*see Chapter 2.3.1*).

In addition, several VLAN were defined to properly manage the network and provide logical segmentation. In particular, VLANs filters enhance network security, perform address summarization, subnetwork isolation, and mitigate network congestion.

Virtual LANs are used to isolate the management network (VLAN 6), from which the virtual farm and the entire network system can be accessed and configured, from the user networks (VLAN 21, 22 and 24). In this way the system can provide different queue speeds, isolation among the different development environments (Test – Acceptance – Production) and other ad-hoc policies. The main VLANs are:

- VLAN 5 – Hardware: it contains devices such as access points and routers.
- VLAN 6 – Management: used by admins to manage the system.
- VLAN 10 – Production Farm: virtual machines providing basic services such as Active Directory, DNS, Radius and many others.
- VLAN 11 – Test Farm: used to deploy and test new machines.
- VLAN 12 – Acceptance Farm: second and last step before production for machines deployment and test.
- VLAN 20 – Sala Olimpo: reserved for computer room.
- VLAN 21 – Free network: similar to users' one but with MAC address whitelist authentication. Used by special machines such as TVs and printers.
- VLAN 22 – Guest: used in special occasions such as parties with guests.
- VLAN 24 – 802.1x: users network with bandwidth limit, accounting and WPA-E authentication with RADIUS.

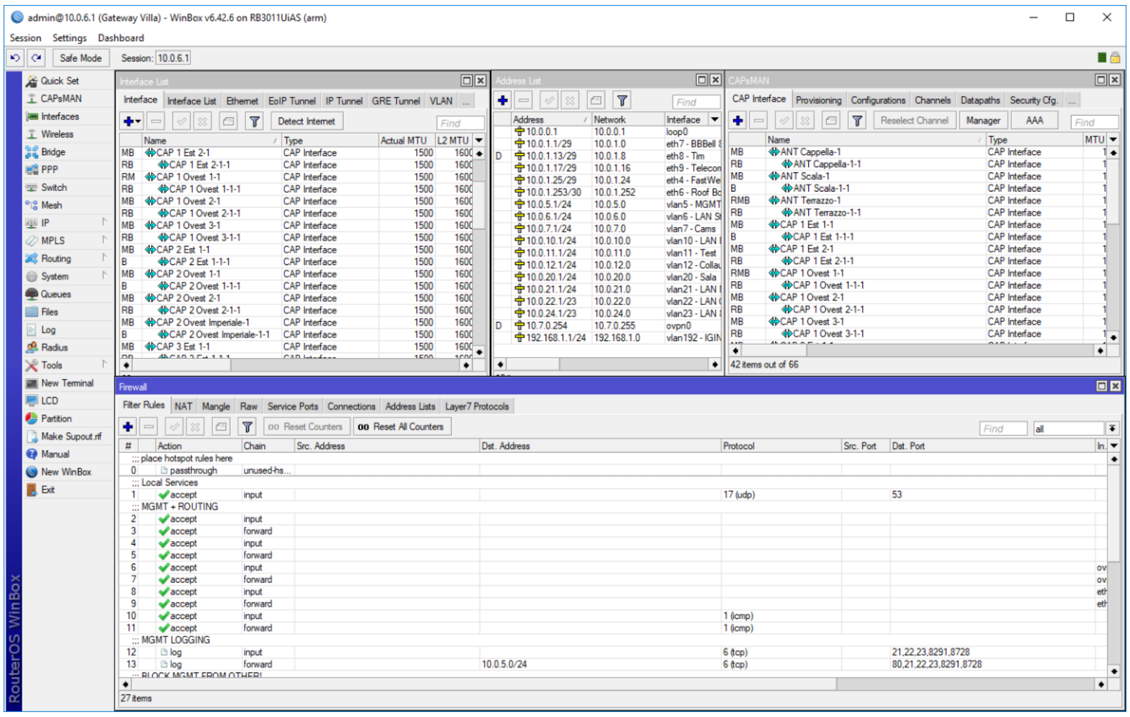
(1) ***Winbox*** *is a small utility that allows administration of MikroTik RouterOS using a GUI.*

For many VLANs a **DHCP pool** is created since the main MikroTik router is also the default gateway and DHCP of the network. For simplicity, address assignment is based on VLAN numbers using the subnets **10.0.V.0/24**, where V is the VID.

Of course, some VLANs (such as VLAN 5 or VLAN 10) don't have a DHCP pool. This is because these networks are composed exclusively by devices with static IPs.

For example, addresses in the VLAN 6 are in the 10.0.6.0/24 network, while addresses in the VLAN 10 are in the 10.0.10.0/24 network. Two exceptions are the VLAN 22 and 24 since they can have more than 253 users. To solve this problem, DHCP pools 10.0.22.0/**23** and 10.0.24.0/**23** were chosen for these networks**.**

An example of Winbox screenshot is given in the *Figure 2.5.*



Interface List

Interface | Interface List | Ethernet | EoIP Tunnel | IP Tunnel | GRE Tunnel | VLAN | ...

Detect Internet

	Name	Type	Actual MTU	L2 MTU
MB	CAP 1 Est 2-1	CAP Interface	1500	1600
RB	CAP 1 Est 2-1-1	CAP Interface	1500	1600
RM	CAP 1 Ovest 1-1	CAP Interface	1500	1600
RB	CAP 1 Ovest 1-1-1	CAP Interface	1500	1600
MB	CAP 1 Ovest 2-1	CAP Interface	1500	1600
RB	CAP 1 Ovest 2-1-1	CAP Interface	1500	1600
MB	CAP 1 Ovest 3-1	CAP Interface	1500	1600
RB	CAP 1 Ovest 3-1-1	CAP Interface	1500	1600
MB	CAP 2 Est 1-1	CAP Interface	1500	1600
B	CAP 2 Est 1-1-1	CAP Interface	1500	1600
MB	CAP 2 Ovest 1-1	CAP Interface	1500	1600
B	CAP 2 Ovest 1-1-1	CAP Interface	1500	1600
MB	CAP 2 Ovest 2-1	CAP Interface	1500	1600
RB	CAP 2 Ovest 2-1-1	CAP Interface	1500	1600
MB	CAP 2 Ovest Imperiale-1	CAP Interface	1500	1600
B	CAP 2 Ovest Imperiale-1-1	CAP Interface	1500	1600
MB	CAP 3 Est 1-1	CAP Interface	1500	1600

Address List

	Address	Network	Interface
	10.0.0.1	10.0.0.1	loop0
	10.0.1.1/29	10.0.1.0	eth7 - BBBell
D	10.0.1.13/29	10.0.1.8	eth8 - Tim
	10.0.1.17/29	10.0.1.16	eth9 - Telecom
	10.0.1.25/29	10.0.1.24	eth4 - FastWe
	10.0.1.253/30	10.0.1.252	eth6 - Roof Bo
	10.0.5.1/24	10.0.5.0	vlan5 - MGMT
	10.0.6.1/24	10.0.6.0	vlan6 - LAN St
	10.0.7.1/24	10.0.7.0	vlan7 - Cams
	10.0.10.1/24	10.0.10.0	vlan10 - LAN
	10.0.11.1/24	10.0.11.0	vlan11 - Test
	10.0.12.1/24	10.0.12.0	vlan12 - Collau
	10.0.20.1/24	10.0.20.0	vlan20 - Sala
	10.0.21.1/24	10.0.21.0	vlan21 - LAN
	10.0.22.1/23	10.0.22.0	vlan22 - LAN
	10.0.24.1/23	10.0.24.0	vlan23 - LAN
D	10.7.0.254	10.7.0.255	ovpn0
	192.168.1.1/24	192.168.1.0	vlan192 - IGIN

CAPsMAN

CAP Interface | Provisioning | Configurations | Channels | Datapaths | Security Cfg. | ...

Reselect Channel | Manager | AAA

	Name	Type	MTU
MB	ANT Cappella-1	CAP Interface	1
RB	ANT Cappella-1-1	CAP Interface	1
MB	ANT Scala-1	CAP Interface	1
B	ANT Scala-1-1	CAP Interface	1
RMB	ANT Terrazzo-1	CAP Interface	1
RB	ANT Terrazzo-1-1	CAP Interface	1
MB	CAP 1 Est 1-1	CAP Interface	1
B	CAP 1 Est 1-1-1	CAP Interface	1
MB	CAP 1 Est 2-1	CAP Interface	1
RB	CAP 1 Est 2-1-1	CAP Interface	1
RMB	CAP 1 Ovest 1-1	CAP Interface	1
RB	CAP 1 Ovest 1-1-1	CAP Interface	1
MB	CAP 1 Ovest 2-1	CAP Interface	1
RB	CAP 1 Ovest 2-1-1	CAP Interface	1
MB	CAP 1 Ovest 3-1	CAP Interface	1
RB	CAP 1 Ovest 3-1-1	CAP Interface	1

42 items out of 66

Firewall

Filter Rules | NAT | Mangle | Raw | Service Ports | Connections | Address Lists | Layer7 Protocols

00 Reset Counters | 00 Reset All Counters | all

#	Action	Chain	Src. Address	Dst. Address	Protocol	Src. Port	Dst. Port	In.
	;;; place hotspot rules here							
0	passthrough	unused-hs...						
	;;; Local Services							
1	accept	input			17 (udp)		53	
	;;; MGMT + ROUTING							
2	accept	input						
3	accept	forward						
4	accept	input						
5	accept	forward						
6	accept	input						ov
7	accept	forward						ov
8	accept	input						eth
9	accept	forward						eth
10	accept	input			1 (icmp)			
11	accept	forward			1 (icmp)			
	;;; MGMT LOGGING							
12	log	input			6 (tcp)		21,22,23,8291,8728	
13	log	forward		10.0.5.0/24	6 (tcp)		80,21,22,23,8291,8728	

27 items

Figure 2.10: The Winbox utility used to configure and manage RouterOS

2.2.3 Access points configuration

Internet access is provided using several MikroTik access points. The chosen model for the basic access point is the hAP Lite RB941, which can easily be configured with the **CAPsMAN**[1] [6] feature of MikroTik products.

This powerful tool basically allows centralization of wireless network management and data processing.

[1] When using **CAPsMAN**, the network consists of many 'Controlled Access Points' (**CAP**) that provide wireless connectivity; and a 'system Manager' (CAPsMAN) that manages the configuration of the APs.

In each hallway the access points are connected as shown in the *Figure 2.6*. This simplifies the wiring from the floors to the server room (with an imperceptible performance drop).

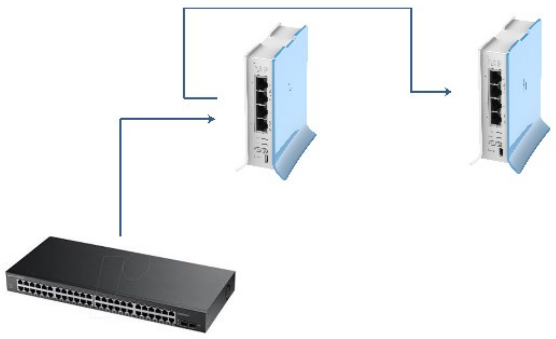


Figure 2.11: The cascaded connection of the CAPs.

Being the access points also MikroTik products, they can be easily accessed using Winbox, creating a user friendly centralized environment.

Finally, several MikroTik SXTG 5HPND outdoor access points were introduced, in order to provide network access *outside* the building. These devices also support CAPsMAN configuration for centralized management. Being outdoor-specific, these devices provide a better and directional range of Wi-Fi access.

In addition, these products support **PoE**[(1)] power supply, which makes them weather-proof and easy to install.

Another ad-hoc configuration is used to provide internet access to a detached building, using a **radio link** with two Ubiquiti M5 300 antennas. However, being this a L2 link it is totally transparent to the entire system in terms of VLAN and configuration. This set up is shown in the *Figure 2.7*.

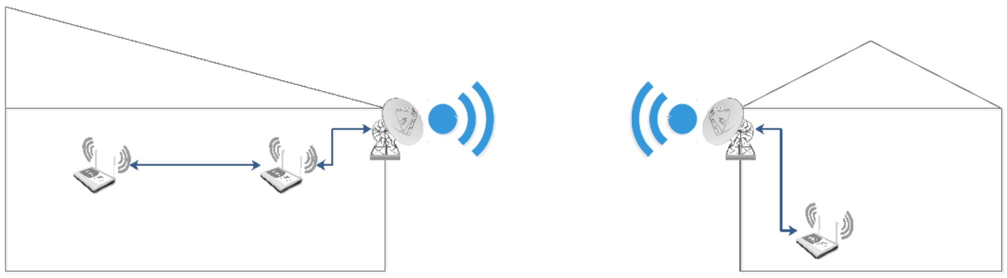


Figure 2.12: The radio link architecture for internet access in separate buildings.

(1) ***Power over Ethernet*** *(PoE) passes electric power along with data on twisted pair Ethernet cabling. This allows a single cable to provide both data connection and electric power to devices such as wireless access points, IP cameras, and VoIP phones.*

2.2.4 UPS and NAS

To prevent power outage damages, a Riello Vision 1500 **UPS**(1) [7] was added.

In this configuration, each rack device is connected to the UPS, which can provide a run-time of at least *twenty minutes* on power outage.

When time is running low, the remaining portion is used by a special virtual machine to gracefully shutdown the server, the virtual machines and eventually other network systems.

This operation is performed by **upsmon**(2) (*see Chapter 3.18*).

Two of the eight sockets of our UPS are called **Enegyshare socket** [8]. They allow to manage the connected devices in a smart way, shutting down the less important ones in order to reserve more battery life for more important devices.

In this system secondary devices are shut down 10 minutes after the power outage.

The last covered topic in this section is the TS-431 QNAP **NAS**(3) [9].

These NAS models are particularly easy to configure, as the simple insertion of the *QNAP Cloud Code* grants access to a cloud-based configuration console named **myQNAPcloud** [10].

This NAS contains four Seagate 2TB disks, configured with a **RAID**(4) **10**.

A RAID 10 is a special combination of two types of RAID:

- RAID 0 consists of **striping**. The throughput of read and write operations is multiplied by the number of disks, as reads and writes are done concurrently in multiple disks.
- RAID 1 consists of data **mirroring**. If a drive fails, the controller uses either the data drive or the mirror drive for data recovery and continues the operation.

(1) *An **Uninterruptible Power Supply** (UPS) is an electrical apparatus that provides emergency power to a load when the mains power fails. The on-battery run-time of most UPS is relatively short, but sufficient to perform a graceful shutdown of the system.*

(2) ***Upsmon** is a client process responsible for the most important part of UPS monitoring: shutting down the system when the power goes out. It can also call out to other helper programs for notification purposes.*

(3) *A **Network-attached storage** (NAS) is a data storage server connected to a network providing data access to a heterogeneous group of clients. A NAS is specialized for serving files either by its hardware, software, or configuration.*

(4) *A **Redundant Array of Independent Disks** (RAID), is a data storage virtualization technology that combines multiple physical disk drive components into one or more logical units for the purposes of data redundancy, performance improvement, or both.*

The RAID 10 combines these benefits as shown in the *Figure 2.8*, providing data redundancy and performance improvement at the same time. This performance is well used by the LAG created for this device, providing a really good overall performance.

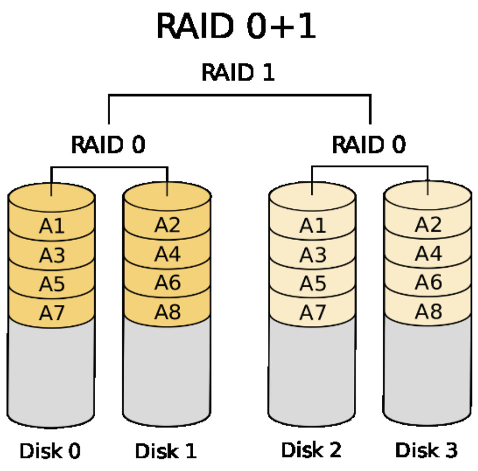


Figure 2.13: The RAID 10 Schema. [11]

With this configuration there are approximately 4TB of space which can be used by the system. **NFS**[1] v2/v3/v4 services were also enabled, as the NAS will be largely used by Linux systems.

This protocol, along with the NAS GUI and others services like **winbind**, simplifies permission management, a crucial issue for network storage security.

As a result, there are many **Shares**[2] configured with specific and carefully assigned permissions for different kinds of users.

Many other aspects of this very important device will be later discussed, especially for services like **NextCloud** and **Active Directory folder redirection**.

2.2.5 The rack colours

To conclude this section, a coloured image of the main rack is posted as *Figure 2.9*. Different colours were used for different purpose cables:

- Yellow: *floor APs*
- Red: *ISP*
- Blue: *computer room*
- Green: *printers / servers*
- Black: *storage / devices*
- Grey: *dedicated lines*
- Teal: *fibre cable*

[1] ***Network File System*** *(NFS) is a distributed file system protocol allowing a computer to access files over a computer network much like local storage is accessed.*

[2] *A* ***shared resource*** *(or network share) is a computer resource made available from one host to other hosts on a computer network.*

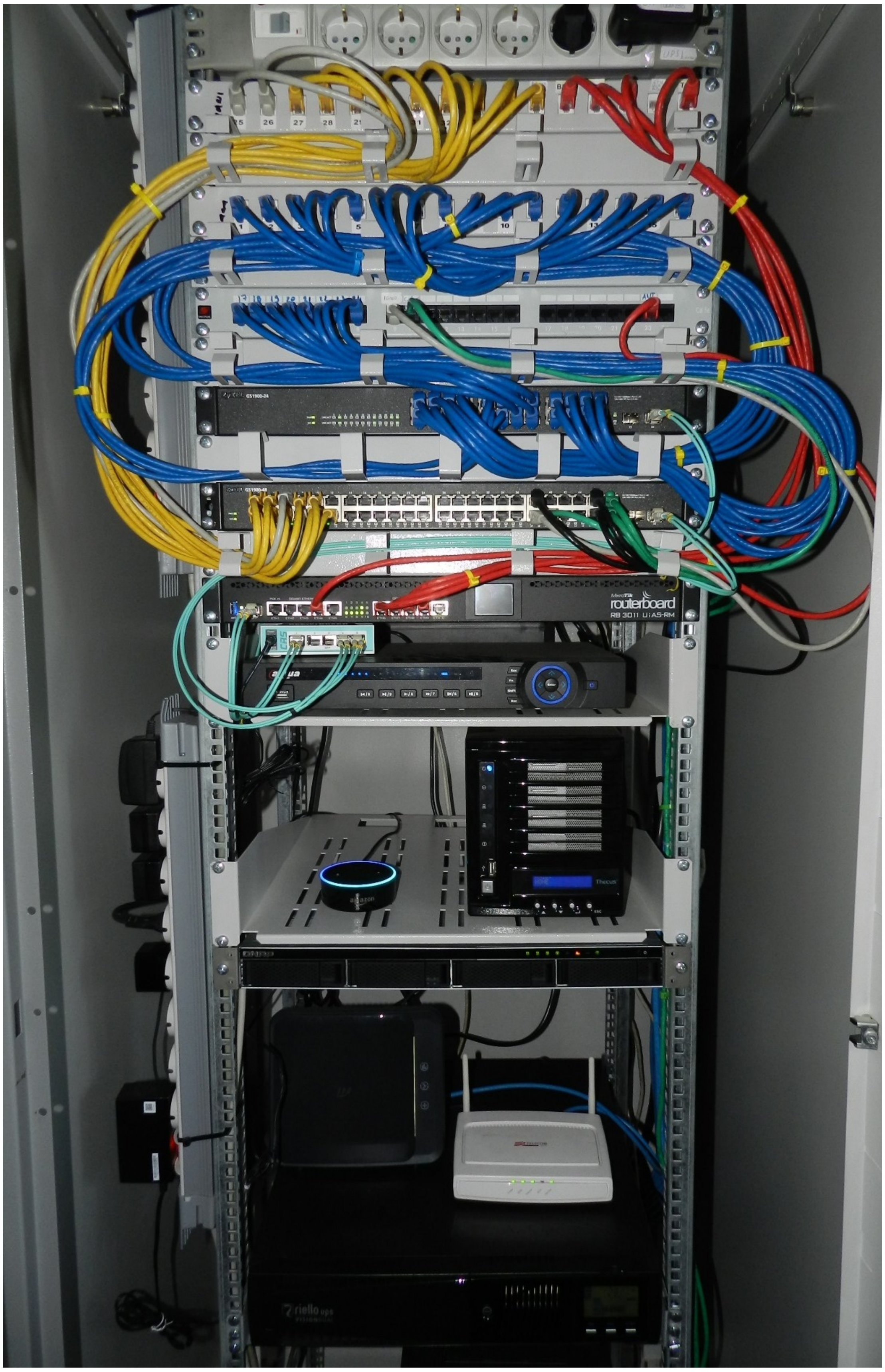


Figure 2.14: The fully coloured rack, core of the network.

Note also the **Amazon Echo Dot 2** device (**Alexa**) whose usage will be largely discussed in the next chapters.

2.3 Network Configuration

2.3.1 Routes and Queues

Having different ISP, many **routes** to those gateways were created.

All of those routes were set to destination 0.0.0.0/0 in the proper routes menu. By doing so, any of these routes can be chosen for navigation to any address. However, a basic load balancer was set up using a special MikroTik feature called **Mangles** [12]. It provides a pre-routing option like *Cisco's policy-map*, allowing a simple address-based network balance.

In particular, the mangle feature process is presented in the *Figure 2.10* and described below:

- The MikroTik firewall assigns to each packet a **routing mark**, depending on the source IP address. The routing mark is assigned as follow:
    - A hash algorithm is applied to the source IP, obtaining a 32-bit value.
    - This value is divided by a configured *denominator*, obtaining a *remainder*.
    - The remainder is used to assign the routing mark. Based on the connection bandwidth, multiple remainders can be assigned to a single routing mark type, allowing a better load balance.
- The routing mark is later used to properly route the packet.

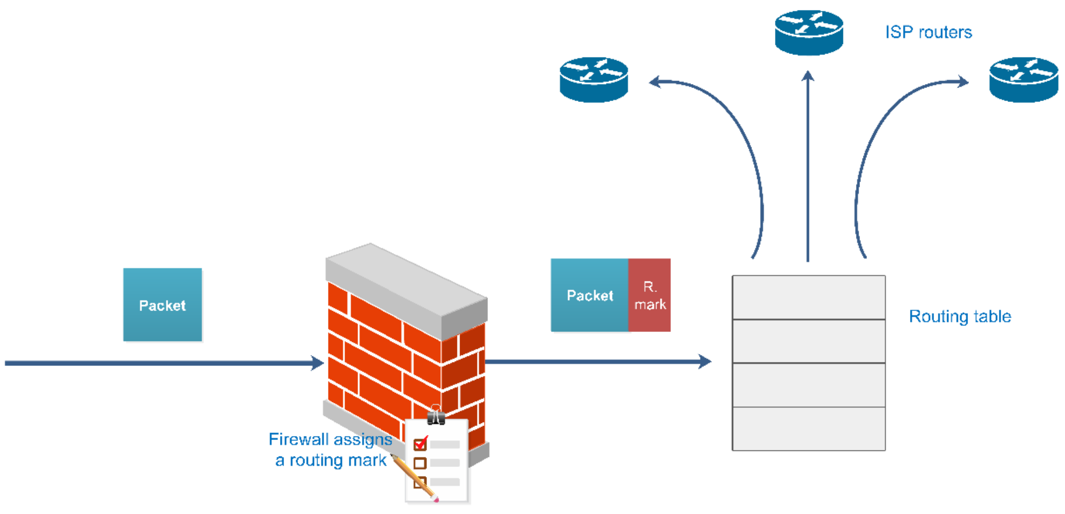


Figure 2.15: The routing mark assignment and utilization.

For some VLANs, routing marks are set regardless of the source address.

For example, the VLAN 10 (Production Farm) has the "*Eolo*" routing mark always set, as many services exposed requires a **static public IP** (*see Chapter 2.3.3*).

In addition, depending on the chosen VLAN policy, a special **PCQ** can be configured.

A **Per Connection Queue** (PCQ) is a queuing discipline that can be used to dynamically shape traffic for multiple users.

In this case, it is configured to set up a *rate limit* over the entire VLAN for a single address. Other parameters such as the *burst limit* can be freely configured.

2.3.2 Firewall

A good firewall configuration is crucial in a network system [13].

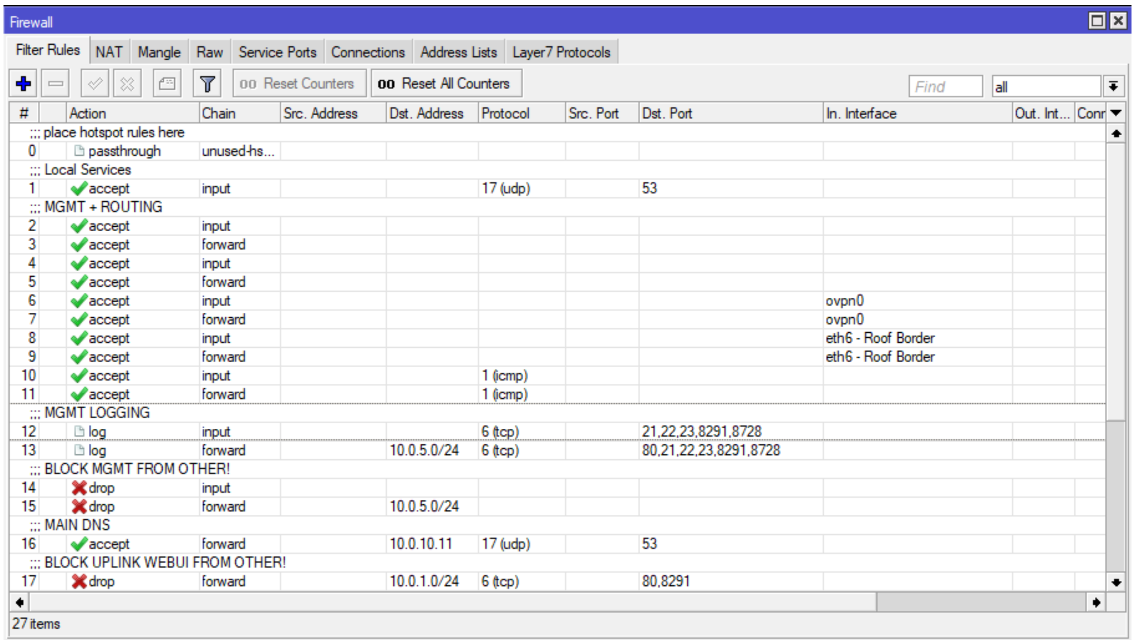
In the MikroTik RouterOS, the firewall operates by means of **firewall rules**. Each rule consists of two parts: the matcher, which matches traffic flow against given conditions; and the action, which defines the operations to be performed.

Firewall filtering rules are grouped together in **chains**. Rules are taken from the chain from top to bottom. If a packet matches the criteria of the rule, then the specified action is performed on it, and no more rules are processed in that chain.

If a packet has not matched any rule within the built-in chain, then it is accepted.

There are three predefined chains:

- **Input**: used to process entering packets with one of the router's addresses destination.
- **Forward**: used to process packets passing through the router.
- **Output**: used to process packets generated by the router.



#	Action	Chain	Src. Address	Dst. Address	Protocol	Src. Port	Dst. Port	In. Interface	Out. Int...	Conn
;;; place hotspot rules here										
0	passthrough	unused-hs...								
;;; Local Services										
1	✓ accept	input			17 (udp)		53			
;;; MGMT + ROUTING										
2	✓ accept	input								
3	✓ accept	forward								
4	✓ accept	input								
5	✓ accept	forward								
6	✓ accept	input						ovpn0		
7	✓ accept	forward						ovpn0		
8	✓ accept	input						eth6 - Roof Border		
9	✓ accept	forward						eth6 - Roof Border		
10	✓ accept	input			1 (icmp)					
11	✓ accept	forward			1 (icmp)					
;;; MGMT LOGGING										
12	log	input			6 (tcp)		21,22,23,8291,8728			
13	log	forward		10.0.5.0/24	6 (tcp)		80,21,22,23,8291,8728			
;;; BLOCK MGMT FROM OTHER!										
14	✗ drop	input								
15	✗ drop	forward		10.0.5.0/24						
;;; MAIN DNS										
16	✓ accept	forward		10.0.10.11	17 (udp)		53			
;;; BLOCK UPLINK WEBUI FROM OTHER!										
17	✗ drop	forward		10.0.1.0/24	6 (tcp)		80,8291			

27 items

Figure 2.16: An example of firewall configuration using Winbox.

The firewall base configuration for this system is presented in the *Figure 2.11*. The basic principle of this firewall is simple, as the rules follow these simple criteria:

a. Input chain packets for local services (such as DNS) are accepted.
b. Input or forward chains packets from management networks (e.g. VLAN 6) are always accepted.
c. Forward chain packet to admin networks (e.g. VLAN 5) are logged.
d. Input chain packets are dropped.
e. Forward chain packets to admin networks (e.g. VLAN 5) are dropped.

This makes the firewall system really easy to read and manage.

2.3.3 Boarder and NAT

Since there are also wireless ISP (like *Eolo*) with public IP addresses, another routerboard was placed on the building roof in order to simplify management, traffic flow and to provide a **NAT** to expose public services such as *Cloud* and *Mail*.

The chosen product for this role is a MikroTik RB 493 AH routerboard (**roof border** hereinafter), connected directly to the main routerboard.

The architecture is shown in *Figure 2.12*.

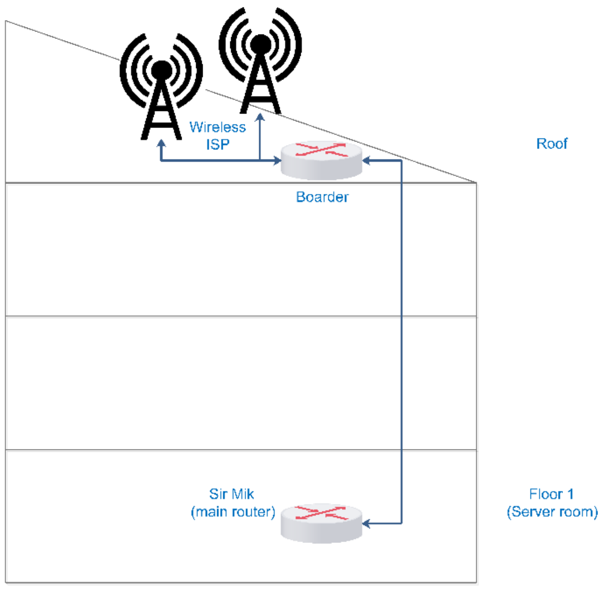


Figure 2.17: The roof border architecture for wireless ISP.

This specific solution was adopted instead of a classic L2 switch to easily deal with the public (not assignable) IP addresses. Another solution to achieve this goal would be to use an L2+ switch and a smart VLAN use.

However, a routing solution was chosen as it was considered to be more suitable for this task, and more easy to be managed.

An interesting fact is that all these ISPs provide network access in exchange for hosting their **BTS**[1] systems, all of them placed in the roof and connected to the roof border. This allows to have internet access for a really cheap price.

Another up-side of having this border is the possibility to use the **NAT**[2] functionality and perform an easy setup of a **VPN**[3] over the public IP address.

To activate this functionality, the **L2TP/IPsec server** was enabled on the border, along with the creation of usernames and secrets to enable connections.

To take full advantage from the NAT feature, two domains were bought (salaolimpo.eu and salaolimpo.cloud) and used to externalize some services, like a cloud platform (www.salaolimpo.cloud), a webmail (www.salaolimpo.eu), an ftp server (nas.salaolimpo.eu), and many others discussed later (*see Chapter 3*).

Many of these services are reachable via an **SSL off-loader** proxy since they all overlay over ports 80 and 443.

Other services using different ports (like ftp for port 21) are directly forwarded by the NAT to the correct machine and potentially to another specified port.

Having two ISP providing public IP address; one of them is employed for Production-Environment services, while the other is used for Acceptance-Environment services. This allowed to correctly perform test before adding a new service or replace an old one, without creating disservice for the users.

Some logos of the services exposed using the NAT functionality are displayed in the *Figure 2.18*.

Figure 2.18: The logos of the exposed user and admin services.

[1] *A **Base Transceiver Station** (BTS) is a piece of equipment that simplifies wireless communication between end user equipment and an ISP network, acting precisely as a transceiver between those actors.*

[2] *The NAT functionality provides external access to many services.*

[3] *The VPN is used by the admins to remotely manage the system, as the IP address assigned by the VPN is part of the management network.*

Chapter 3: Server Farm

3.1 Introduction

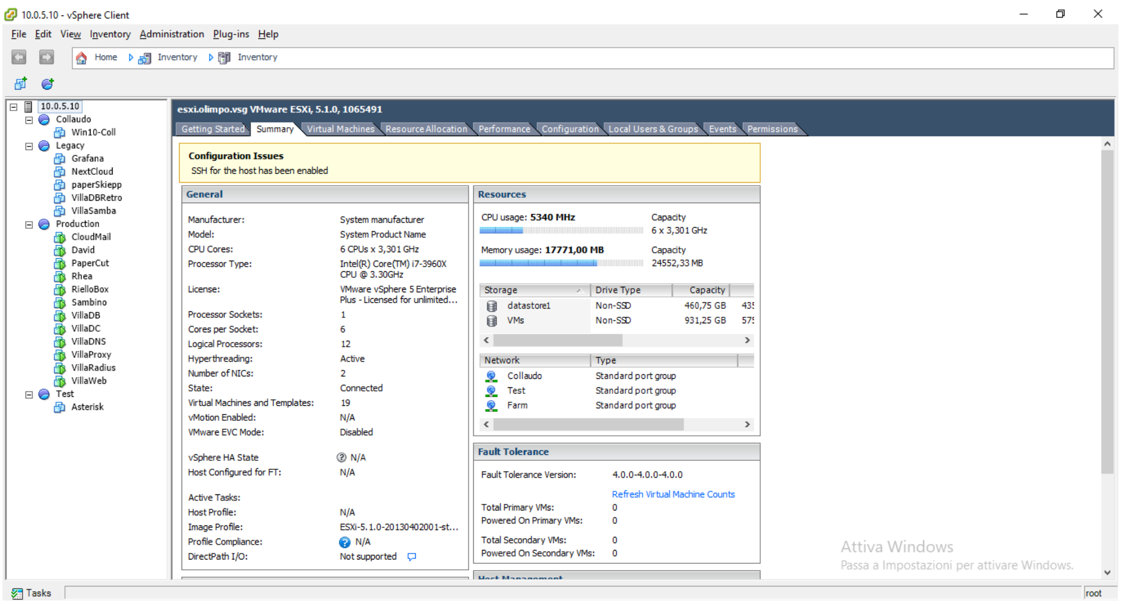
This chapter will cover the **Virtual Server Farm** topic in terms of software, services and protocols used.

A virtual server farm is an environment that employs multiple application and infrastructure servers using a server virtualization program such as VMware.

Firstly, some of the functional requirements for the use case are presented:

- Users must have an account which can be used in many applications.
- Each user has a credit, used for printing purposes.
- The user experience must be as easy as possible.
- Some services like Cloud and Mail must be accessible to users.
- Other services (e.g. a monitoring platform) must be accessible by admins.
- These features must be accessible from outside.
- These services must be properly integrated in the system using either connectors or orchestrators.

For this project, all the virtual farm is hosted in a single physical computer running a version of **VMware ESXi**[(1)] **Server 5.1** [14], as shown in *Figure 3.1*.



esxi.olimpo.vsg VMware ESXi, 5.1.0, 1065491

Configuration Issues
SSH for the host has been enabled

General

Manufacturer:	System manufacturer
Model:	System Product Name
CPU Cores:	6 CPUs x 3,301 GHz
Processor Type:	Intel(R) Core(TM) i7-3960X CPU @ 3.30GHz
License:	VMware vSphere 5 Enterprise Plus - Licensed for unlimited...
Processor Sockets:	1
Cores per Socket:	6
Logical Processors:	12
Hyperthreading:	Active
Number of NICs:	2
State:	Connected
Virtual Machines and Templates:	19
vMotion Enabled:	N/A
VMware EVC Mode:	Disabled
vSphere HA State	N/A
Host Configured for FT:	N/A
Active Tasks:	
Host Profile:	N/A
Image Profile:	ESXi-5.1.0-20130402001-st...
Profile Compliance:	N/A
DirectPath I/O:	Not supported

Resources

CPU usage: **5340 MHz** Capacity 6 x 3,301 GHz

Memory usage: **17771,00 MB** Capacity 24552,33 MB

Storage	Drive Type	Capacity	
datastore1	Non-SSD	460,75 GB	43
VMs	Non-SSD	931,25 GB	57

Network	Type
Collaudo	Standard port group
Test	Standard port group
Farm	Standard port group

Fault Tolerance

Fault Tolerance Version:	4.0.0-4.0.0-4.0.0
Total Primary VMs:	0
Powered On Primary VMs:	0
Total Secondary VMs:	0
Powered On Secondary VMs:	0

Figure 3.19: A screen of the vSphere 5.1 Client, connected to VMware ESXi Server 5.1.

(1) ***VMware ESXi*** *(formerly ESX) is a hypervisor software developed by VMware for deploying and serving virtual machines.*

The design pattern adopted heavily relies on the virtualization of services. Every set of services runs on different virtual machines, providing many benefits like:

- ✓ **Snapshots**[1]: which makes the rollback operation easy and cheap to perform, allowing a huge time savings when a misconfiguration or a change of mind occur.
- ✓ **Migration and Failover**: machines can be moved to another host machine easily, providing uninterrupted service if the prior physical host is, for example, taken down for maintenance. This allows the VM to continue operations from the last-known coherent state, rather than the current state.
- ✓ **Cost saving**: as only a single computer (with high resources) is needed. Good virtualizing software along with proper configurations leads to efficient resource utilization.
- ✓ **Hardware Virtualization**: software executed on these virtual machines is separated from the underlying hardware resources. For example, a computer that is running Microsoft Windows may host a Linux virtual machine.
- ✓ **Sandbox**[2] **and Development Environments**: as shown in *Figure 3.2*, a **DTAP cycle** is used for service deployment, which is easy to be applied in a virtualized environment. Moreover, production deployment can be performed for a specific service without affecting others.
- ✓ **Transparency and independency between services**: by separating all the services between each other, a misconfiguration or more generally an error for one of those VM doesn't affect the other ones. This also allows to separate logs for efficient troubleshooting. As a consequence, resolution attempts can be performed without affecting other systems.

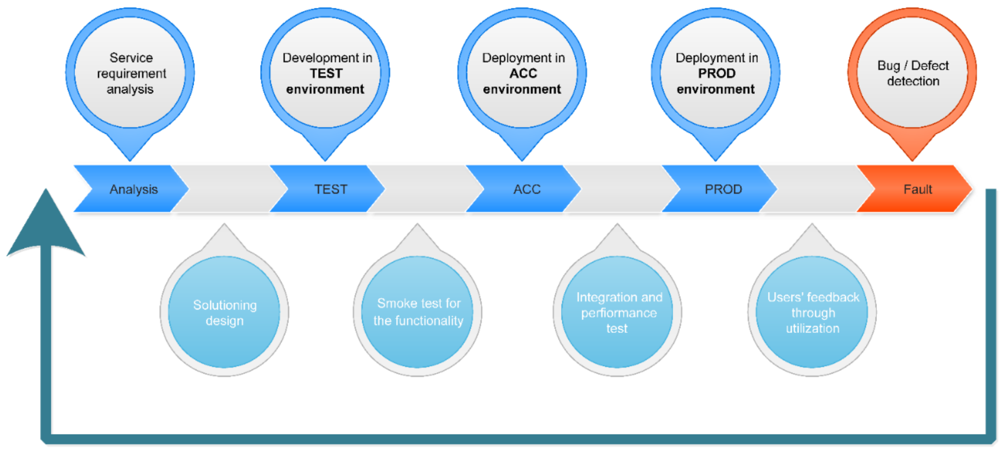

Service requirement analysis
Development in TEST environment
Deployment in ACC environment
Deployment in PROD environment
Bug / Defect detection
Analysis
TEST
ACC
PROD
Fault
Solutioning design
Smoke test for the functionality
Integration and performance test
Users' feedback through utilization

Figure 3.20: The DTAP cycle adopted for services development.

[1] *A **snapshot** is a state of a virtual machine, and generally its storage devices, at an exact point in time. It enables to later restore the virtual machine's state at the time of the snapshot, undoing any changes that occurred afterwards.*

[2] *A **sandbox** is a testing environment that isolates untested code changes from the production environment.*

3.2 Development Environments

For the lifecycle management of all the services, a three-tier architecture was adopted using three different **Development Environment**[1] properly isolated between each other using VLAN and firewall rules.

The stages adopted are:

- **Test (TEST)**, where new virtual machines or new services can be created, configured, and developed without affecting active ones. Services in TEST are **isolated** from others, except when communication between services is required. This isolation is obtained with the VLAN 11 as **TEST VLAN**.
  During this phase, the list of operations and configurations used are stored in a dedicated page on the MediaWiki platform (*see Chapter 3.19*).
  Once the machine or the service is properly working on TEST, it can be promoted to the next development environment.
- **Acceptance (ACC)**, where new virtual machines or services developed in TEST undergo an **integration test**.
  To perform this operation, the dedicated page on MediaWiki is followed by replicating all and only the reported steps. If the deployment succeded, then it is tested within all the system. Otherwise, the deploy will be started over after some edits in TEST environment and in the MediaWiki page.
  The ACC environment is isolated from other environments, but its deployed services are free to communicate with each other, as required for an integration test. This is obtained with the VLAN 12, as **ACC VLAN**.
  If all the integration tests are successful, then the machine is promoted to the next development environment.
- **Production (PROD)**, where only working machines and services are kept. All these services are available for end users, so they must be reliable in order to avoid disservices.
  The **PROD VLAN** (VLAN 10) is free to communicate with the entire system, since successful ACC tests have asserted the integration correctness. Moreover, the installation should always be successful on the first try, as PROD deployment follows the same practices of ACC deployment.

In the next chapters all the virtual machines in PROD environment are discussed more in depth, as they required a lot of effort to configure.

[1] *A **Development Environment** is a computer system where a computer program or software component is deployed and executed. This structured release process allows phased deployment (rollout), testing, and rollback in case of problems.*

3.3 Windows Server

3.3.1 Introduction

This virtual machine is the primary **Active Directory** [15] server and **Domain Controller**[1] of the system.

More specifically, it contains the **LDAP database** [16] for the users, along with an internal PKI acting as the root CA. As an Active Directory server, it manages policies for computers part of the domain (i.e. all the PC in computer room).

Figure 3.21: The Active Directory logo.

This machine was managed and configured using the Remote Desktop Protocol (**RDP**), even though ordinary operations are performed using the Rhea Admin Console (see *Chapter 3.10*) or David (see *Chapter 4*).

The forest contains only a single domain, and its structure is shown in *Figure 3.4*.

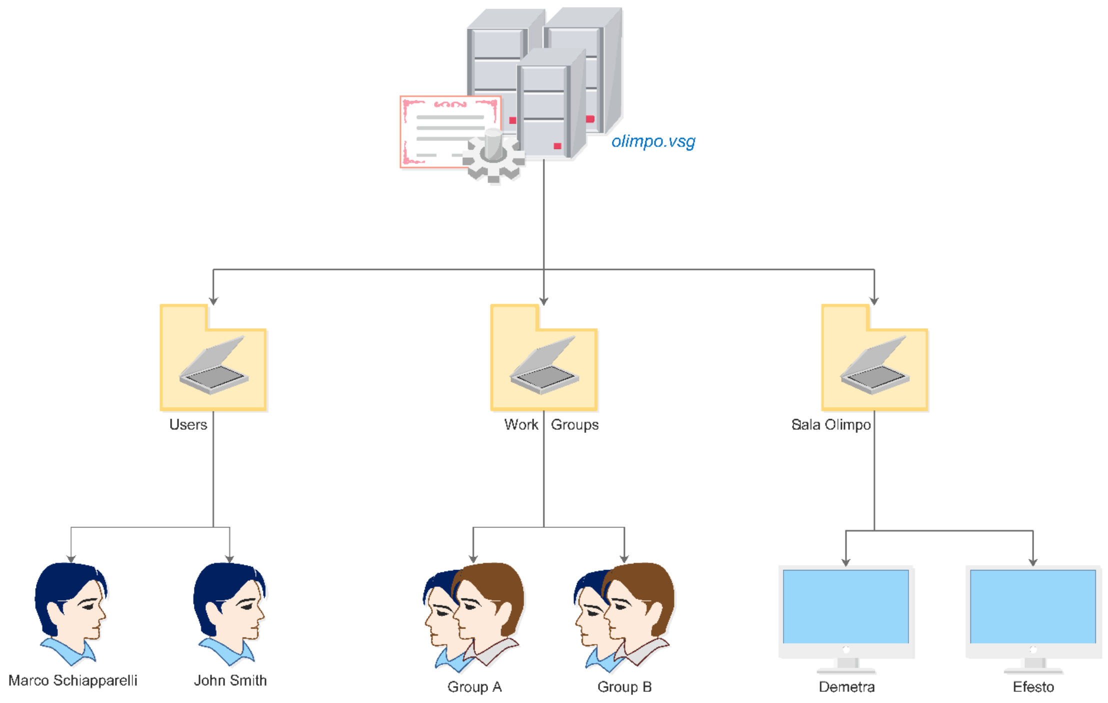


Figure 3.22: The Active Directory Domain structure for the system.

[1] *A **Domain Controller** is a server that responds to security authentication requests within a Windows Server domain.*

3.3.2 Installation and Configuration – Active Directory

Every common operation can be performed using the **Server Manager** [17] application. It is the main component of Windows Server, as it can be used to install all the desired features and properly configure each component.

The first component needed is the Active Directory, which is the core part of the domain. Therefore, the AD feature is firstly added using the Server Manager and then configured to be promoted to a Domain Controller.

This step requires a **forest**(1) creation. In this case, the forest contains only the domain *olimpo.vsg*, and all the computers and virtual machines are part of it.

Promotion to Domain Controller requires a **dynamic DNS update**. The DNS must grant update permission to the DC for some specific zones, as join operation uses **a DNS query of type SRV**, since the supplicant needs to know the address of the primary DC. When a join operation takes place, the DNS is updated one more time to add an A record for the newly joined hostname.

VillaDNS machine (*see Chapter 3.4*) is used as a DNS for this operation. This choice was made as Bind9 is considered more configurable than Window DNS. Moreover, Microsoft itself suggests that Active Directory should be the only running service on a Windows Server machine. This advice was accepted as the "*divide et impera*" practice has been largely adopted in this system.

After the service deployment, the **LDAP database** can be populated by creating user accounts. This feature is crucial in this system, as it provide a backend for **centralized login**(2) used by almost every deployed service.

The *Figure 3.5* gives only some examples of services using this LDAP database.

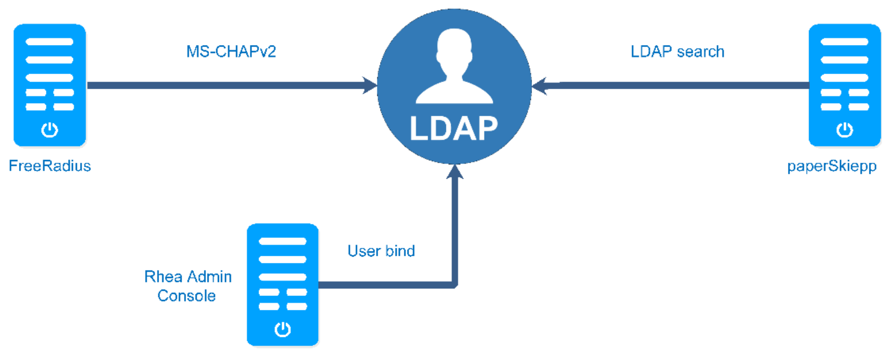


Figure 3.23: A small set of services that uses the LDAP database for authentication purpose.

(1) *A **forest** is a complete instance of Active Directory. Each forest acts as a top-level container in that it houses all domain containers for that particular Active Directory instance.*

(2) ***Authentication** can be exploited using standard authentication methods, like MS-CHAP, or others like LDAP binding. Some other services use synchronization of accounts, performed by an LDAP search on the domain.*

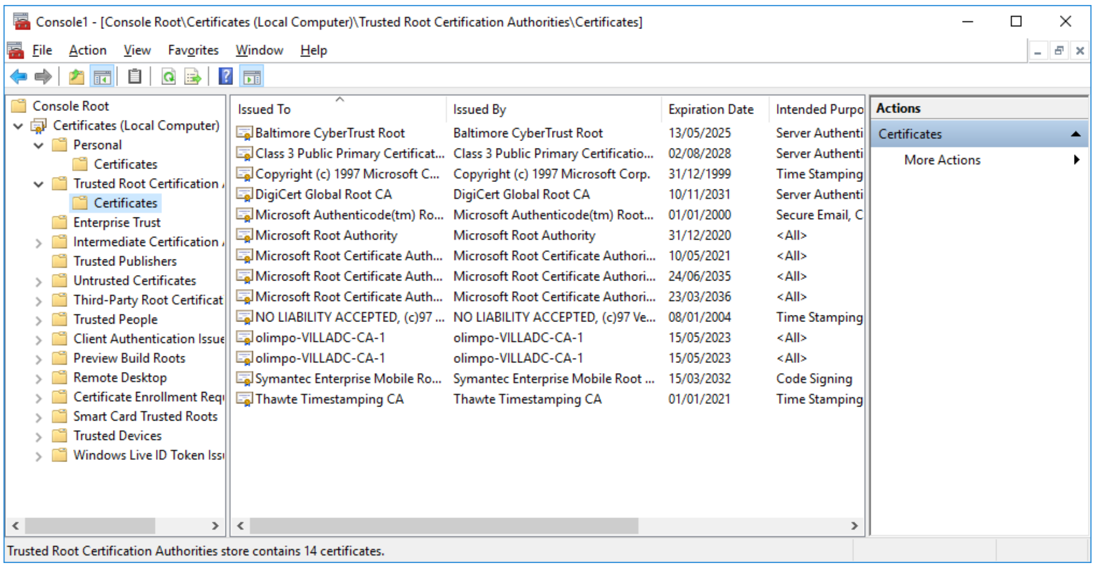
3.3.3 Installation and Configuration – Certificate Server

A **Certificate Authority Server** [18] has been introduced to create a **PKI** for the domain. Basically, it is used to distribute signed certificates over the system.

Certificates are needed to bind in LDAPS (LDAP over SSL), required by some services like FreeRADIUS (*see Chapter 3.5*). In particular, the server is configured as an **Enterprise CA**(1).

Once the Enterprise CA is deployed, a certificate can be issued using the MMC tool, as shown in *Figure 3.6*. During this operations, the certificates can be configured ad-hoc for actual need.

As example, they can be configured to expire within a given date, or to be valid for a specific usage such as Server Authentication or Client Authentication.


Console1 - [Console Root\Certificates (Local Computer)\Trusted Root Certification Authorities\Certificates]

Issued To	Issued By	Expiration Date	Intended Purpo
Baltimore CyberTrust Root	Baltimore CyberTrust Root	13/05/2025	Server Authenti
Class 3 Public Primary Certificat...	Class 3 Public Primary Certificatio...	02/08/2028	Server Authenti
Copyright (c) 1997 Microsoft C...	Copyright (c) 1997 Microsoft Corp.	31/12/1999	Time Stamping
DigiCert Global Root CA	DigiCert Global Root CA	10/11/2031	Server Authenti
Microsoft Authenticode(tm) Ro...	Microsoft Authenticode(tm) Root...	01/01/2000	Secure Email, C
Microsoft Root Authority	Microsoft Root Authority	31/12/2020	<All>
Microsoft Root Certificate Auth...	Microsoft Root Certificate Authori...	10/05/2021	<All>
Microsoft Root Certificate Auth...	Microsoft Root Certificate Authori...	24/06/2035	<All>
Microsoft Root Certificate Auth...	Microsoft Root Certificate Authori...	23/03/2036	<All>
NO LIABILITY ACCEPTED, (c)97 ...	NO LIABILITY ACCEPTED, (c)97 Ve...	08/01/2004	Time Stamping
olimpo-VILLADC-CA-1	olimpo-VILLADC-CA-1	15/05/2023	<All>
olimpo-VILLADC-CA-1	olimpo-VILLADC-CA-1	15/05/2023	<All>
Symantec Enterprise Mobile Ro...	Symantec Enterprise Mobile Root ...	15/03/2032	Code Signing
Thawte Timestamping CA	Thawte Timestamping CA	01/01/2021	Time Stamping

Trusted Root Certification Authorities store contains 14 certificates.

Figure 24: The MMC tool, used to issue and configure certificates.

3.3.4 Installation and Configuration – Folder Redirection

Another important service of Windows Server is the **Folder Redirection**(2) [19].

In this specific case, it is used to redirect user folders in a domain-joined NAS; containing a share with Windows ACL support and read/write permission properly set for Domain Users.

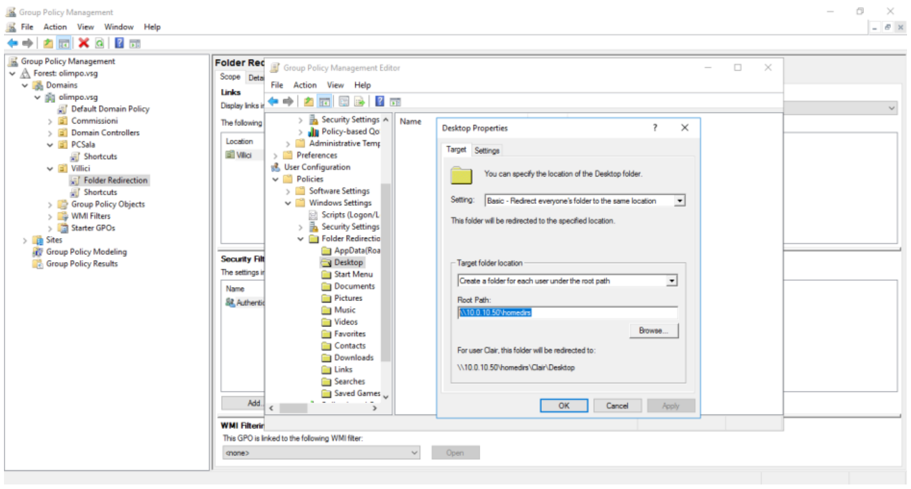
This gives many advantages, for example:

(1) *An* ***Enterprise CA*** *is integrated with Active Directory. The server will use domain services for certificate management, and it integrates with the directory for naming and authentication. This type of CA differs from the Standalone option, that doesn't integrate with AD.*

(2) ***Folder Redirection*** *redirects the path of a known folder to a new location using a Group Policy. The new location can be a folder on the local computer or a directory on a file share.*

- A large storage for user files is not needed as all files are stored in the NAS. In addition, they are protected from data loss with the RAID10 feature.
- User files are not stored directly on the computer, meaning that a user can freely change the PC workstation without losing files. This is particularly useful in this system since users can access each workstation randomly.
- Storing files on shares prevents computer faults to impact these files.
- User files are exposed and can be accessed using the cloud platform.

Using the **Group Policy**[(1)] **Management** tool, a **GPO**[(2)] must be created and configured for the required feature, as shown in *Figure 3.7*. This policy is later linked with the OU containing all the users enabled to use the computer room.



Desktop Properties

You can specify the location of the Desktop folder.

Setting: Basic - Redirect everyone's folder to the same location

This folder will be redirected to the specified location.

Target folder location: Create a folder for each user under the root path

Root Path: \\10.0.10.50\homedirs

For user Clair, this folder will be redirected to: \\10.0.10.50\homedirs\Clair\Desktop

WMI Filtering: This GPO is linked to the following WMI filter: <none>

Figure 3.25: The Group Policy Management tool, used for GPO creation and deployment..

In the workstations, as shown in *Figure 3.8*, a green checkmark over the folder icon points out that it is correctly redirected and synchronized with NAS folder.

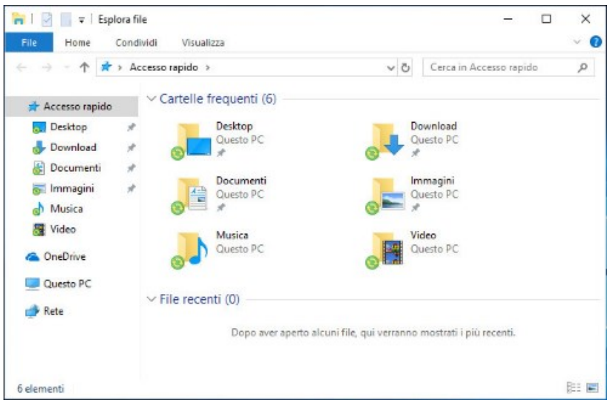


Figure 3.26: The green icon representing the Folder Redirection feature.

(1) ***Group Policy*** *is a feature of the Windows NT family systems that controls the working environment of user and computer accounts. It provides centralized management and configuration of systems, applications, and users' settings in an Active Directory environment.*

(2) *A* ***Group Policy Object*** *(GPO) is a collection of settings that define what a system will look like and how it will behave for a defined group of users.*

3.4 VillaDNS

3.4.1 Introduction

This virtual machine represents a very important service for the network, since it's the main DNS of the system. The software used is **Bind9**[(1)] [20], and its logo is shown in *Figure 3.9*.

BIND

Berkeley Internet Domain Name

Figure 3.27: The Bind9 logo.

Each computer in the VLAN 20, and optionally others, use this machine as DNS.

3.4.2 Installation and configuration

For the configuration part, only some files must be edited:

- `named.conf.options` contains the base options for the service.
  For example, forwarders (like 8.8.8.8 and 8.8.4.4) can be set for non-mastered zones; along with other options like subnet allowed to query, to recur and to update. In addition, the update part can be configured, as the domain controller must be allowed to update the DNS for LDAP SRV records and all the computers joining the domain.
- `named.conf.local` allow to specify options for each zone of our interest.
  In particular, a very important zone is **olimpo.vsg**. For this zone, bind will act as master DNS. For each zone of interest, a specific file must be created containing all the configuration required.
  They contain info such as the SOA (start of authority) record, the version number, the NS record, and other standard ones such as A and CNAME.
- A reverse zone can be also optionally configured. In this case, a zone-like file must be created with a particular type of record named PTR[(2)].
  They are used by some network applications to perform security checks.

An example of configuration file (the Dynamic Zone of the domain *olimpo.vsg*) is posted in *CodeView 3.1*.

[(1)] ***Bind*** *is the most widely used Domain Name System (DNS) software on the Internet. On Unix-like operating systems it is the de facto standard.*

[(2)] ***Pointer records*** *(PTR) are specifically used to map an IP to a host name.*

```
$ORIGIN .
$TTL 7200 ; 2 hours
olimpo.vsg       IN      SOA     VillaDNS.olimpo.vsg. hostmaster.olimpo.vsg. (
                                        2018071203      ; serial (YYYY/MM/DD version)
                                        900             ; refresh (15 minutes)
                                        600             ; retry (10 minutes)
                                        86400           ; expire (1 day)
                                        3600            ; minimum (1 hour)
                                )

                         NS      VillaDNS.olimpo.vsg.

$ORIGIN olimpo.vsg.
$TTL 7200 ; 2 hours

VillaDC                  A       10.0.10.10
VillaDNS                 A       10.0.10.11
VillaRadius              A       10.0.10.12
PaperSkiepp              A       10.0.10.13
VillaDB                  A       10.0.10.14
CloudMail                A       10.0.10.15

VillaProxy               A       10.0.10.16
Rhea                     A       10.0.10.17
RielloBox                A       10.0.10.18
VillaWeb                 A       10.0.10.19
VillaSamba               A       10.0.10.20
David                    A       10.0.10.21
```

CodeView 3.1:The olimpo.vsg Zone configuration file.

3.4.3 Final notes

This machine was actually the first one promoted to production, as having a DNS allows easier configurations on other machines. Having a centralized DNS allows to simply change a DNS records when a machine (or a service) changes the address.

Troubleshooting was simplified by the **dig**[1] command, found inside `dnsutils`.

3.5 VillaRadius

3.5.1 Introduction

This virtual machine allows users to authenticate against LDAP to obtain internet access. The software used is **FreeRADIUS**[2] as shown in *Figure 3.10*.

This protocol has a better user experience than mechanism such as **Captive Portals**, and provides accounting features unlike the simple WPA/WEP protocols.

[1] ***Domain Information Groper*** *(dig) is a network administration command-line tool for querying DNS servers. It is a useful troubleshooting tool, as it provides many information regarding the DNS query performed.*

[2] ***FreeRADIUS*** *is a modular, high performance free RADIUS suite developed and distributed under the GNU General Public License.*

Figure 3.28: The FreeRADIUS logo.

Each device on the VLAN 24 authenticates itself with the protocol WPA2-E (or WPA-802.1X). It follows the steps described and shown in *Figure 3.11*:

- An **EAPoL**[1] is started with a hAP. In this step, the authenticator requests the identity to the supplicant.
- A radius access request is sent to the authenticator server.
  At this point, an **SSL/TLS inner tunnel** is created between the supplicant and the authentication server. The hAP in this configuration acts like a pass-through for the tunnel.
- Using this tunnel, a challenge-based authentication is performed for effectively authenticate the user.
- An access or reject response is sent back to the client, and a standard WPA/WPA2 communication continues from now on.

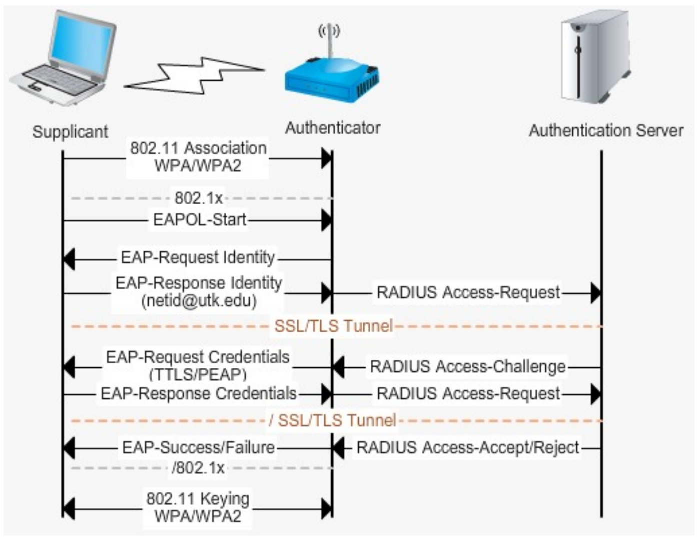


Figure 3.29: The FreeRADIUS authentication and access protocol. [21]

3.5.2 Installation and configuration

This configuration part requires to properly configure some features named **virtual servers** (i.e. virtual RADIUS servers) [22]. They are very important since they have the ability to **separate policies**.

[1] **EAPoL** *is a network port authentication protocol used to give a generic network sign-on to access network resources.*

Each virtual server has his own file, configured to contain many **sections** specifying policies. Examples of sections are:

- authorize, used to authorize the user.
- authenticate, used to authenticate the user.
- accounting, used to keep track of the user utilization.

In particular, the two virtual servers configured are:

- The `default`, configured to listen on default ports (1812 for authentication, 1813 for accounting) for all IP addresses. For authorize and authenticate, the **eap** module is enabled as it will be used for the tunnel.
- The `inner-tunnel`, which is configured to listen on a specific port (18120) for all IP address. The protocol MS-CHAP is listed in the authenticate section as LDAP will be used for the authentication. Therefore, the `ldap` and `mschap` modules are enabled in this same file.

In addition to virtual servers, all the needed modules of FreeRADIUS must be enabled and properly configured. As example:

- `ldap`, configured with info about the domain controller.
- `mschap`, configured to use `ntlm_auth`, being it Winbind's NTLM authentication function.
- `eap`, providing many information such as:
    - The eap type (**peap**).
    - The virtual server encapsulated (`inner-tunnel` in this case).
    - The **tls properties**, where the certificates must be specified.

Lastly, a simple Samba service must be configured for joining the domain. Since Samba will be better explained in *Chapter 3.13*, other details are not given here.

An example of the inner-tunnel file is presented in *CodeView 3.2*.

3.5.3 Final notes

This machine is really important since it provides authentication for internet access. On the MikroTik side, the utilization of this machine is performed by simply enable the Radius feature for Wireless access.

By using an Active Directory LDAP system as backend, user management is simple and centralized as users can be disabled from LDAP, and a filter over groups and OU can be set to proper select the users allowed to use the network.

The integration tests were easily performed using FreeRADIUS debug mode, really useful as it provides a complete trace of the messages exchanged.

```
server inner-tunnel {
  listen {
           ipaddr = 127.0.0.1
           port = 18120
           type = auth
  }
  authorize {
           mschap
           eap { ok = return }
           ldap
           expiration
           logintime
           ntlm_auth
  }
  authenticate {
           Auth-Type MS-CHAP   { mschap }
           Auth-Type ntlm_auth { ntlm_auth }
           eap
  }
  session {  }
  post-auth {
           ldap
           Post-Auth-Type REJECT {
                    attr_filter.access_reject
                    update outer.session-state {
                             &Module-Failure-Message := &request:Module-Failure-Message
                    }
           }
  }
  pre-proxy {  }
  post-proxy {
           eap
           update {
                    &outer.session-state: += &reply:
           }
  }
}
```

CodeView 3.2 The inner-tunnel RADIUS configuration file.

3.6 PaperSkiepp

3.6.1 Introduction

This virtual machine is used to manage print accounts, since users are allowed to print using a credit system which must be automatically detracted or charged.

These features can be obtained with software like **PaperCut**. However, since it is not a free software; a reverse engineering of the program has been created under the name of **PaperSkiepp** using open source software from **PyKota** website.

This software is created by the combination of three main components: **CUPS**, **Tea4CUPS** and **pkpgcounter** (as shown in *Figure 3.12*), with the addition of a MySQL database (*see Chapter 3.7*) and Rhea Admin Console (*see Chapter 3.10*).

UNIX PRINTING SYSTEM

PYKOTA OFFICIAL

Figure 3.30: CUPS and PyKota logos

When a user starts a print job, the interaction of these components goes as follow:

1. The print job is received by **CUPS** [23], a modular printing system for Unix-like operating systems which allows a computer to act as a print server. Basically, it is a print spooler and scheduler.
2. CUPS forwards any print job to the **Tea4CUPS** backend [24]. It is a wrapper which can capture print data before it is sent to a printer in order to be processed. It executes a **bash script** performing the remaining actions.
3. The bash script executes **pkpgcounter** [25] to calculate the number of pages, the page size, the colour options and the username which is printing. The fee for the job is calculated and detracted from the user's credit in the database. Of course, if the credit of the user is not enough, the transaction is not performed and the print job is discarded.
4. If the user has enough credit, then the print job leaves Tea4CUPS and reaches the printer.

This flow is graphically presented in *Figure 3.13*.

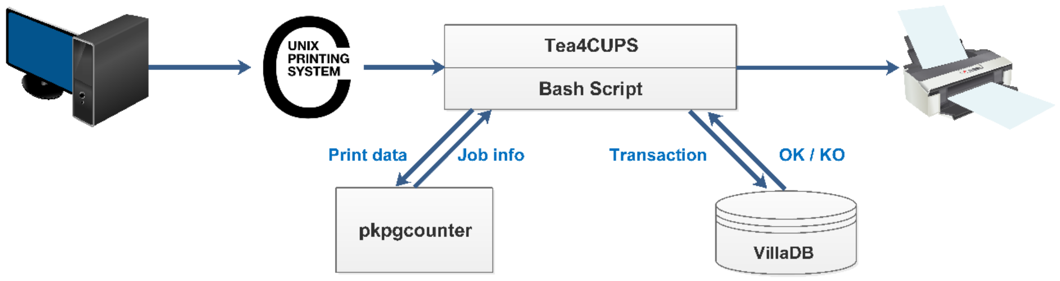


Figure 3.31: The interaction of all PaperSkiepp components.

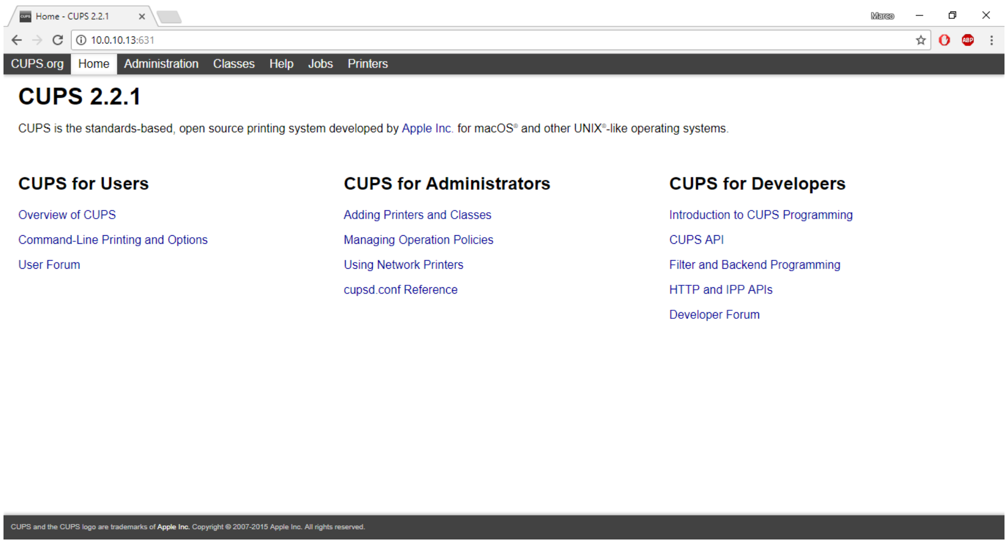
3.6.2 Installation and configuration

For the installation of pkpgcounter and Tea4CUPS, they can be downloaded directly from the distributor website: *www.pykota.com*.

As for the configuration, CUPS can be managed by accessing it through http on port 631. The only operation needed is adding the printer with a general **Linux PCL drivers**, as pkpgcounter works well with the **PCL**(1) format. The printer must be configured as **shared**, as it will be used by many computers in our network.

After the printer configuration, the **DeviceURI** of the printer must be set to **tea4cups:**//socket://<IP>. This is a special **Virtual Device URI** used as wrapper to the communication with the printer, as Tea4CUPS backend will performs operation with the print job before sending it to the printer.

(1) *The **Printer Command Language** (PCL), is a page description language (PDL) developed by Hewlett-Packard as a printer protocol and has become a de facto industry standard.*


Home - CUPS 2.2.1

10.0.10.13:631

CUPS.org Home Administration Classes Help Jobs Printers

CUPS 2.2.1

CUPS is the standards-based, open source printing system developed by Apple Inc. for macOS® and other UNIX®-like operating systems.

CUPS for Users	CUPS for Administrators	CUPS for Developers
Overview of CUPS	Adding Printers and Classes	Introduction to CUPS Programming
Command-Line Printing and Options	Managing Operation Policies	CUPS API
User Forum	Using Network Printers	Filter and Backend Programming
	cupsd.conf Reference	HTTP and IPP APIs
		Developer Forum

CUPS and the CUPS logo are trademarks of Apple Inc. Copyright © 2007-2015 Apple Inc. All rights reserved.

Figure 3.32: A screen of the CUPS web configuration panel.

Once these operations are concluded, the Tea4CUPS configuration files needs to be edited to use the proper objects. Three types of objects can be created:

- **filters** (one per queue), which modifies the input job's data before it is sent to printer.
- **prehooks** (n per queue), guaranteed to be launched before the print job is sent to the printer. If a prehook returns -1, it cancels the print job.
- **posthooks** (n per queue), guaranteed to be launched after the print job has been sent to the printer, unless the job was previously cancelled.

The order in which the operations are performed by Tea4CUPS backend is presented in the *Figure 3.15.*

In this case, a single prehook was created. In particular, the script uses some **environment variables** to retrieve the username, the printer, the title and the data to be print. It performs also output redirection to proper log.

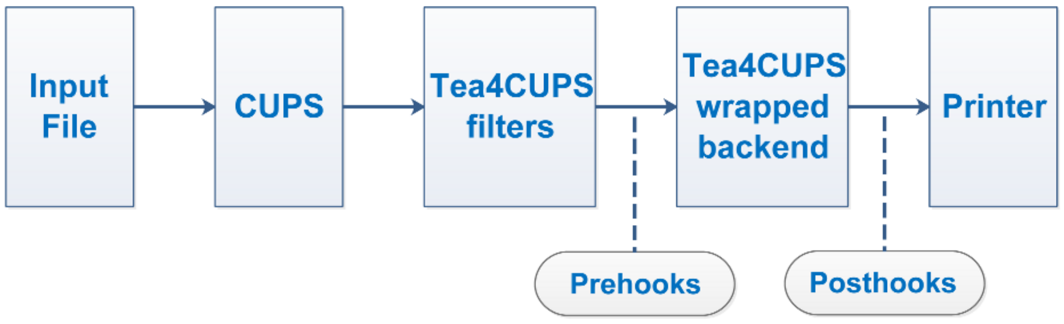


Figure 3.33: The operation performed using by objects of the Tea4CUPS beckend.

The executed script calculates the fee using the pkpgcounter command line utility. It takes as input the PCL file to be parsed, and displays in output the options for that print job.

The bash script executed is presented integrally in *CodeView 3.3.*

```
#!/bin/bash
PKRESULT=$(pkpgcounter -d $2 2>&1)

#---Check if the print if A3 / A4---
if echo $PKRESULT | grep -q "A3" ; then
        FORMAT=A3
else
        FORMAT=A4
fi
#---Check for the Duplex Option---
if echo $PKRESULT | grep -q "Duplex" ; then
        SIXDUP=DUX
else
        SIXDUP=SIX
fi
#---Check for the Colors Option---
if echo $PKRESULT | grep -q "Color" ; then
        COLORS=COL
else
        COLORS=BW
fi

#============================================================================
#---Get the number of pages---
pages=$(pkpgcounter $2)

#---Retrieving prices and ID---
printerdata=$(mysql -u $USER -p$PASSW -h $IPVDB -D$DATAB -e "SELECT $FORMAT$SIXDUP$COLORS, ID_P FROM printers WHERE
PSalias='$1'" -N -s -B)
IDprinter=$(echo $printerdata | cut -f2 -d' ')

#---Calculating the fee for the print job---
credit=$(echo $printerdata | cut -f1 -d' ')
credit=$(bc <<< "$credit * $pages")

#---Getting the actual credit---
userdata=$(mysql -u $USER -p$PASSW -h $IPVDB -D$DATAB -e "SELECT unlimited, can_print, credit, pages, ID_U FROM users
WHERE username LIKE('$3')" -N -s -B)

#============================================================================
#---Retrieving all the properties of the user---
userli=$(echo $userdata | cut -f1 -d' ')
useren=$(echo $userdata | cut -f2 -d' ')
usercr=$(echo $userdata | cut -f3 -d' ')
userpg=$(echo $userdata | cut -f4 -d' ')
userna=$(echo $userdata | cut -f5 -d' ')

#---Is the user enabled? If NO: Stop---
if [[ "$useren" -eq "0" ]] ; then
        return -1
fi
#---Is the user unlocked?---
if [[ "$userli" -eq "0" ]] ; then

        #---If NO: has the use enough credit?---
        if (( $(echo "$usercr < $credit" | bc) )) ; then
                return -1
        fi
fi

#============================================================================
#--- printJob accepted ---

#---Inserting job into print log---
mysql -u $USERN -p$PASSW -h $IPVDB -D$DATAB -e "INSERT INTO printlog (ID_U, ID_P, pdate, pages, title, payed) VALUES
($userna, $idp, NOW(), $pages, '$4', $credit);"

#---Calculating new credit & pages---
credit=$(bc <<< "$usercr - $credit")
let "userpg = userpg + pages"

#---Update user's credit & printed pages---
mysql  -u  $USERN  -p$PASSW  -h  $IPVDB  -D$DATAB  -e  "UPDATE  users  SET  credit  =  $credit,  pages  =  $userpg  WHERE
username='$3';"

return 0
```

CodeView 3.3: The PaperSkiepp bash script.

3.6.3 Final notes

This service required a lot of effort to be developed, as it is created from scratch in order to avoid purchasing software like **PaperCut**. Since PaperCut is the de-facto only available software capable of reaching this goal, this solution has been created using only open source software. The effort required was higher due to the almost totally absence of documentation and community on web for this kind of projects.

3.7 VillaDB

3.7.1 Introduction

This virtual machine is used as a centralized database with two types of sources: **MySQL** [26] (a Relational DBMS developed by Oracle) and **InfluxDB**(1) [27] (a time-series DBMS developed by InfluxData).

Figure 3.34: MySQL and InfluxDB logos.

This centralization was chosen to make data management easier, especially with software like **phpMyAdmin**(2) [28]. Instead, as InfluxDB doesn't require daily management operations, no graphic tool is installed for that DBMS.

Currently, MySQL contains specific databases for many services in the Virtual Farm, while InfluxDB contains a set of measures for each resource to monitor, and is used specifically by David and Grafana as data source.

3.7.2 Installation and configuration

By using the `mysql` command line utility or the phpMyAdmin GUI, databases and/or tables can be easily created or imported. It's important to properly assign privileges to users using the **GRANT** operation of MySQL. Users are specified in form of **user@host**.

It's a good practice to follow the **Principle of Least Privilege**(3) during this operation, as users should access only one database and sometimes only a set of tables. For this purpose, several **application users**(4) were created for many services, each one with the proper rights over the database resources.

---

(1) ***InfluxDB*** *is an open-source time series database developed by InfluxData optimized for fast, high-availability storage and retrieval of time series data.*

(2) ***phpMyAdmin*** *is a PHP graphic tool used to perform operations on MySQL database.*

(3) *The* ***Principle of Least Privilege*** *(****PoLP****) requires that every module (such as a process, a user, or a program) must be able to access only the information and resources that are necessary for its legitimate purpose.*

(4) *An* ***application user*** *is not an actual user who logs into an application; it is a special user defined to handle access to a service.*

InfluxDB can be configured by its command line utility, accessible with the `influx` command. The set of commands is similar to the MySQL one [29]: for example, the database can be created using the function `CREATE DATABASE`. Instead, the concept of table is replaced by the concept of **measurement**.

A measurement is a series of records, each one composed by a **timestamp**(1) and one or more **values**, optionally enriched by **tags**. For simplicity, it was used one measurement for each resource to monitor; and a single value for each record.

Influx is not considered as a CRUD database, but only as a CRd.

This is because create and read operations are very similar to MySQL and can be easily performed, using the so-called **InfluxQL**. An example of query is given in the *CodeView 3.4*.

The delete operation is only partially supported, while the update operation doesn't exist, as a time-series database is not meant to perform updates.

Another important feature of the InfluxDB is the possibility to create Retention Policies. They are data structures that describes for how long InfluxDB keeps data (**duration**), and how many copies of those data are stored in the cluster (**replication factor**).

In this case, we set up a retention policy for a 7-days permanence and no replication, as presented in *CodeView 3.4*.

```
SELECT difference(<value>) FROM <measure> WHERE time >= NOW() - 6h;

CREATE RETENTION POLICY <policy> ON <database> DURATION 7d REPLICATION 1;
```

CodeView 3.4: Examples of InfluxDB commands.

3.8 CloudMail

3.8.1 Introduction

This virtual machine contains two important services:

- **NextCloud** [30], an open source and self-hosted file share and communication platform. It is a fork of the original ownCloud software.
- A custom **Mail Server**, composed by **Postfix** [31], **Dovecot** [32] and **RoundCube** [33].

These are some crucial services for the system, so a lot of effort was necessary to create this machine.

(1) *In **UNIX time format**, defined as an approximation of the number of seconds that have elapsed since 00:00:00 Coordinated Universal Time (UTC), Thursday, 1 January 1970.*

These services are available internet, making this platform accessible from everywhere. Third-part applications like mail clients and NextCloud Android app can also access and be synchronized with these services. Their logos are shown in *Figure 3.17*.

Figure 3.35: NextCloud and RoundCube logos.

3.8.2 Installation and configuration – NextCloud

Firstly, a share for NextCloud is created in the NAS and mounted with the `/etc/fstab` file. This practice was chosen to keep all the user files safe in the NAS storage, and take full advantage of the RAID 10 in terms of size and data protection. In other words, the user files **will be not affected** by accidental machine faults, and neither by software upgrade or machine substitution.

The configuration can be performed using the web GUI [34] by setting the database and folder options. Once configured, many other applications can be installed directly from the NextCloud interface, as shown in *Figure 3.18*.

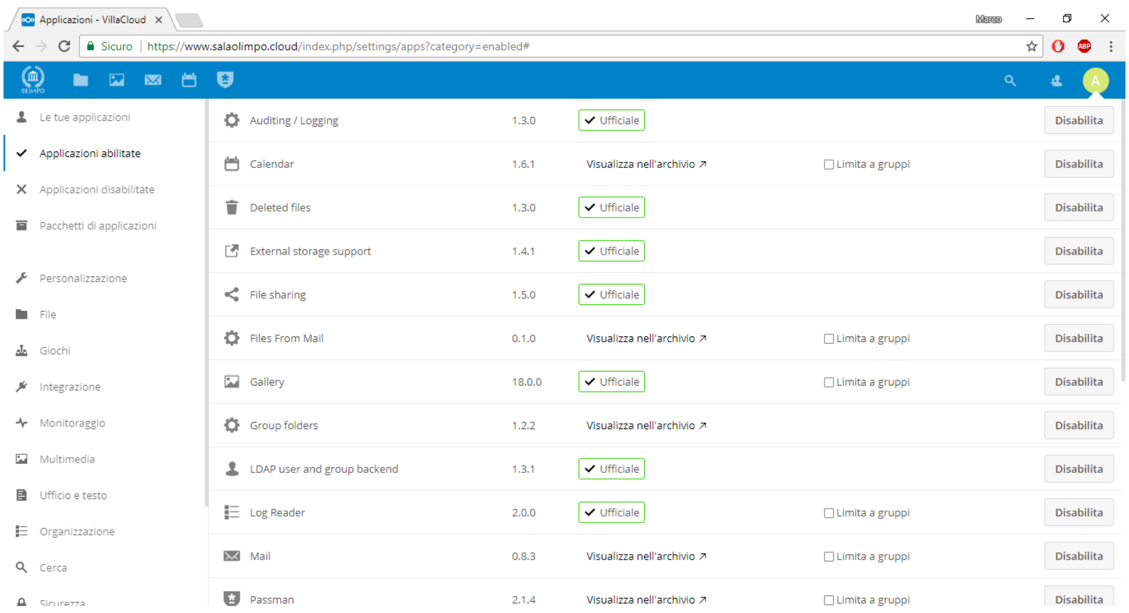
In fact, **NextClous is way more than a simple file cloud platform**.

As example, some applications installed [35] are:

- **Calendar**, which can register, sync and share events and engagements.
- **External storage support**, used to mount folders directly from the NAS. This is particularly useful for public folders shared among all the users.
- **Gallery**, **Video Player** and **PDF Viewer**, to browse multimedia files.
- **Group Folders**, to create group-specific folder for member users.
- **Mail**, which provides a webmail interface for users (*see Chapter 3.15*.3).
- **Passman**, which acts as a safe password storage.

The most important extension is the **LDAP user and group backend**, which allows users from the domain to access NextCloud via LDAP authentication. By accessing the settings of this application we can set the LDAP connection information, such as host, post, and the administrator credentials for the domain, and it can be set to import all users or only from a selected group or OU.

Other settings can be manually set using the `config.php` file. As example, the default mail account can be set.


Applicazioni - VillaCloud
https://www.salaolimpo.cloud/index.php/settings/apps?category=enabled#
Le tue applicazioni
Applicazioni abilitate
Applicazioni disabilitate
Pacchetti di applicazioni
Personalizzazione
File
Giochi
Integrazione
Monitoraggio
Multimedia
Ufficio e testo
Organizzazione
Cerca
Sicurezza

Auditing / Logging	1.3.0	✓ Ufficiale		Disabilita
Calendar	1.6.1	Visualizza nell'archivio ↗	☐ Limita a gruppi	Disabilita
Deleted files	1.3.0	✓ Ufficiale		Disabilita
External storage support	1.4.1	✓ Ufficiale		Disabilita
File sharing	1.5.0	✓ Ufficiale		Disabilita
Files From Mail	0.1.0	Visualizza nell'archivio ↗	☐ Limita a gruppi	Disabilita
Gallery	18.0.0	✓ Ufficiale	☐ Limita a gruppi	Disabilita
Group folders	1.2.2	Visualizza nell'archivio ↗		Disabilita
LDAP user and group backend	1.3.1	✓ Ufficiale		Disabilita
Log Reader	2.0.0	✓ Ufficiale	☐ Limita a gruppi	Disabilita
Mail	0.8.3	Visualizza nell'archivio ↗	☐ Limita a gruppi	Disabilita
Passman	2.1.4	Visualizza nell'archivio ↗	☐ Limita a gruppi	Disabilita

Figure 3.36: The NextCloud interface for installing applications.

To properly load the user files added, deleted or moved using the link on the Sala Olimpo computers (*see Chapter 3.13*), some operations in www-data **cron**[1] are added, along with the cron.php script.

By properly setting these jobs, all files are automatically synchronized, and made accessible from NextCloud. Examples are shown in *CodeView 3.5*.

```
*/5  * * * * php /var/www/html/nextcloud/occ files:cleanup
*/10 * * * * php /var/www/html/nextcloud/occ files:scan --all
*/15 * * * * php -f /var/www/html/nextcloud/cron.php
*/60 * * * * php /var/www/html/nextcloud/occ preview:pre-generate
```

CodeView 3.5: NextCloud cron operations for file synchronization.

This is required as NextCloud generates caches and indexes for its files, providing fast access to the resources. However, file added manually are not automatically cached or indexed, so the `files:scan` procedure needs to be called manually.

This is not the best solution in terms of performance, as it is a sort of polling for new files, performed for all users of the system. Some better solutions could be:

- **Inserting users-specific synchronization procedures when a user logs in into NextCloud**. This solution was discarded as requires software patching, and it would be lost after one of the frequent NextCloud updates.
- **Adding a synchronization procedure each time a user logs in and logs out from a Sala Olimpo PC.** This solution was discarded as considered fragile for PCs, especially since it would need a centralized management for these scripts. In additions, eventual missed synchronization may lead to dangerous data inconsistency.

[1] **Cron** *is a time-based job scheduler in Unix-like computer operating systems.*

The chosen solution is indeed the best trade-off between easy to implement, easy to maintain, and sufficient effectiveness.

A screen of the NextCloud platform is presented in *Figure 3.19*.

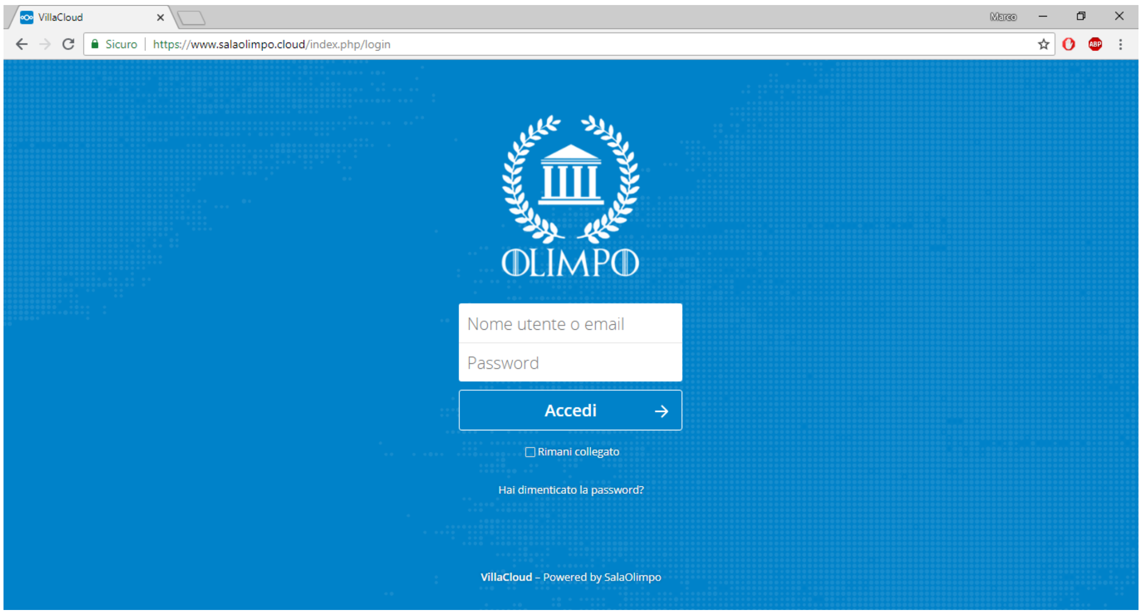

VillaCloud
Sicuro
https://www.salaolimpo.cloud/index.php/login
OLIMPO
Nome utente o email
Password
Accedi
Rimani collegato
Hai dimenticato la password?
VillaCloud – Powered by SalaOlimpo

Figure 3.37: The NextCloud login screen

3.8.3 Installation and configuration – Mail Server

The mail server is composed by many components. However, since this is not a configuration guide, each one will be only briefly discussed here:

- **Postfix** is a free and open-source mail transfer agent (MTA) that routes and delivers electronic mail.
  All of its configuration files are stored under `/etc/postfix` [36], in particular most of the settings are contained in the `main.cf` file.
  After the LDAP configuration, the **postmap** command must be executed to create one or more Postfix lookup tables, or update an existing one.
- **SASL** (Simple Authentication and Security Layer), a framework for authentication and data security, used to perform SMTP authentication.
  After the configuration with the LDAP parameters [37], the functionality can be checked using the telnet command. A trace of the transaction is given in the *CodeView 3.6* (the asterisk-marked parts are user input).
- **Dovecot**, an open-source IMAP and POP3 server for Linux systems.
  A NAS folder is used to store emails, for the same reason already explained in previous chapters. To access this folder, an application user which own all the mails (vmail user) is created, since it's easier than use specific accounts which can lead to permission problems. So, dovecot is simply configured to write mails using that user [38].

```
$  telnet localhost 25
   Trying ::1...
   Trying 127.0.0.1...
   Connected to localhost.
   Escape character is '^]'.
   220 <hostname> ESMTP Postfix
*  EHLO <token>
   250-<hostname>
   250-PIPELINING
   250-SIZE 10240000
   250-ETRN
   250-AUTH PLAIN LOGIN
   250-AUTH=PLAIN LOGIN
   250-ENHANCEDSTATUSCODES
   250-8BITMIME
   250 DSN
*  AUTH LOGIN
   334 VXNlcm5hbWU6
*  <base_64_username>
   334 UGFzc3dvcmQ6
*  <base_64_password>
   235 2.7.0 Authentication successful
*  QUIT
   221 2.0.0 Bye
```

CodeView 6: The telnet trace for SASL authentication for SMTP service.

- **Roundcube** is a web-based IMAP email client. It's most prominent feature is the pervasive use of AJAX technology.
  It doesn't require special configuration, except the database set up by means of RoundCube's configuration file and the relative schema import [39].
  A screen of RoundCube is presented in *Figure 3.20*.

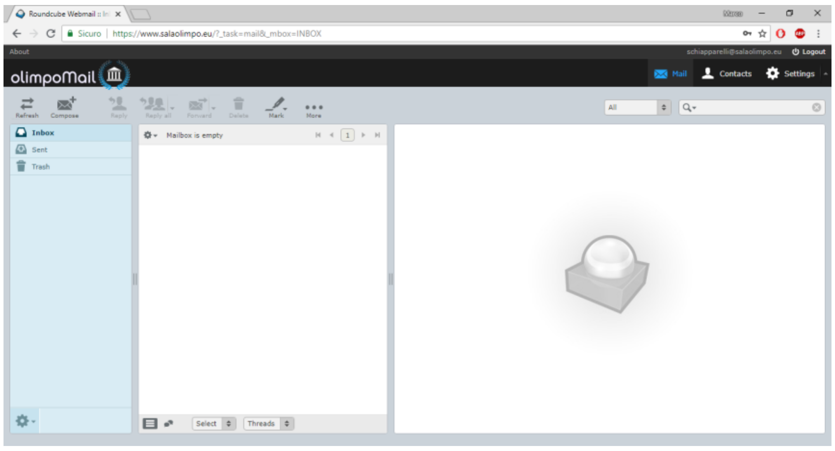


Figure 38: a screen of the RoundCube webmail.

3.8.4 Final notes

This machine was sure one of the best challenging ones to create, especially because it was customized a lot for this case. However, the result is great since all of the files and mails are hosted locally on the NAS, and can be accessed with high speed through the internal network and with normal speed through the external network.

The mail server is also useful for our internal activities, since user mails shares the LDAP account, and are really fast also with attachment if sent locally.

3.9 VillaProxy

3.9.1 Introduction

This virtual machine act as a reverse proxy with SSL offloading for all exposed web services. The software used are **NGINX**(1) [40] and **Certbot**(2) [41] for deploying **Let's Encrypt** [42] certificates, as shown in *Figure 3.21*.

Figure 3.39: NGINX and Let's Encrypt logos.

This solution was adopted for these main reasons:

- Using a reverse proxy with many hostnames provides a better user experience, since remembering a hostname is easier than a port.
- A proxy pass is easier to configure than a set of port forwarding policies, as it only requires to edit few files.
- SSL certificates are installed directly and only in the proxy machine.
- Application servers have less work to do, as SSL handling is not necessary.
- The proxy can be configured to work as a **bastion host** for the network.

3.9.2 Installation and configuration – NGINX part

For the configuration of the NGINX server, one or more files are created in `/etc/nginx/sites-available/`, and then linked under `/etc/nginx/sites-enabled/` [43]. Each file contains one or more virtual servers, each one representing a different service endpoint. The basic operation performed is shown in *Figure 3.22*.

In order to choose the correct proxy-pass policy, a match with the hostname of the endpoint is executed (set using the **server_name** option). The proxy-pass allows to forward packets from a given port to another hosts, and optionally another ports.

(1) ***NGINX** is a free, open-source, high-performance HTTP server and reverse proxy.*

(2) ***Certbot** is a free client for **Let's Encrypt** service, which is a free, automated, and open Certificate Authority.*

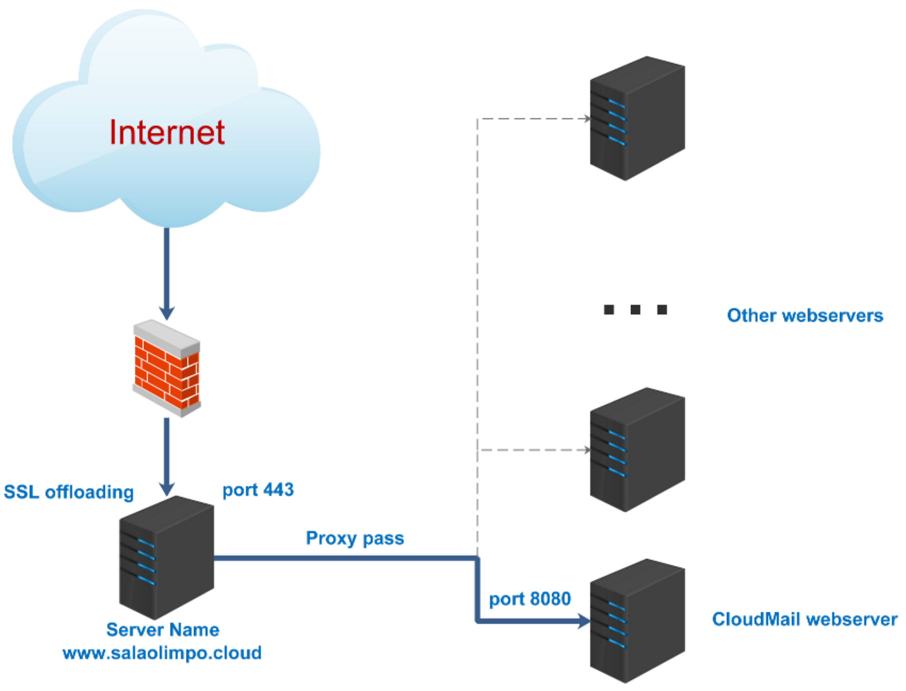


Figure 3.40: The NGIX reverse proxy operation with SSL offloading.

In this case, two files where created: one for http services and the other for https services. In each file it must be specified:

- The port to listen to (typically 80 or 443).
- The server name, i.e. the hostname of the called endpoint.
- The proxy pass target, by means of IP address and port.
- Eventually, some options for header management.

An example of configuration is given in *CodeView 3.7*. It forces https for the host *alexa.salaolimpo.cloud*, and performs proxy pass to 10.0.10.21 on port 8080.

```
server {
   listen 80;
   server_name alexa.salaolimpo.cloud;

   return 301 https://$host$request_uri;
}
server {
   listen 443 ssl;
   server_name alexa.salaolimpo.cloud;

   location / {
      proxy_pass http://10.0.10.21:8080;
      proxy_set_header Host $host;
      proxy_set_header X-Forwarded-Proto $scheme;
      proxy_set_header X-Real-IP $remote_addr;
      proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

      add_header Front-End-Https on;
   }
   ssl_certificate /etc/letsencrypt/live/cert_dir/fullchain.pem; #managed by Certbot
   ssl_certificate_key /etc/letsencrypt/live/cert_dir/privkey.pem; #managed by Certbot
   include /etc/letsencrypt/options-ssl-nginx.conf; #managed by Certbot
   ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; #managed by Certbot
}
```

CodeView 3.7: An example of configuration for the proxy-pass of alexa.salaolimpo.cloud using NGINX.

3.9.3 Installation and configuration – Certbot part

As Certbot has built-in compatibility with NGINX, it can be simply run using the command line tools, specifying all the required domain to issue a certificate for [44].

Being its working algorithm quite interesting [45], an overview of the process will be presented above.

The first step is the **domain validation**, where the agent proves that the web server controls a given domain. This step is performed in this way:

1. The agent generates a key pair, and sends a start request to Let's Encrypt.
2. Let's Encrypt responds with one or more **challenges** and a nonce.
3. The agent exposes the challenges data by means of a **public URI**, under the supplicant domain.
4. The agent sends the signed nonce from step 2.
5. By verifying the signature of the nonce and the challenge exposed by the URI, the domain validation is completed.

The second step is the certificate issuance. The key pair generated in step 1 is now called **authorized key pair**. By using this key pair, the issuance is simple:

6. The agent constructs a **PKCS#10 Certificate Signing Request**, asking a certificate for the domain with a specified public key.
7. The CSR is signed by the private key corresponding to the public key of the CSR, and the whole packet is signed with an authorized key, so that the CA knows that it is authorized.
8. The CA verifies both signatures, and if everything is ok, the certificate is issued with the CSR public key.

A graphical representation is given in *Figure 3.23*:

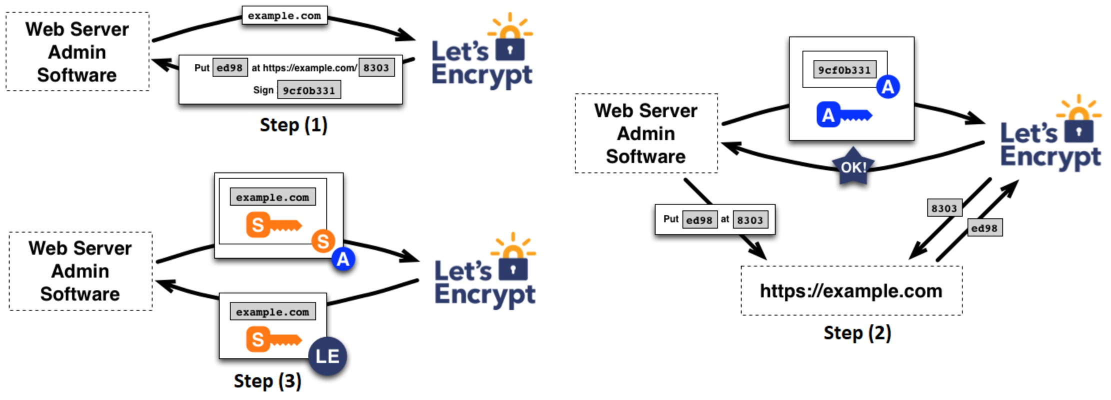


Figure 3.41: The Let's Encrypt algorithm for certificate issue.

3.10 Rhea

3.10.1 Introduction

This virtual machine is used to host two consoles:

- **Rhea Admin Console**, a simple web application developed by the team to perform admin operations (like charge money to users, enable and disable users, view operations log).
- **Grafana** [46], an open-source dashboard and graph composer, which runs as a web application. It supports InfluxDB (*see Chapter 3.7*) and many other data sources.

Both of these services run over a **LAMP**(1) server, as seen in *Figure 3.24*.

Figure 3.42: LAMP and Graphana logos.

Rhea Admin Console was created for these specific reasons:

- Ordinary administration needs to be easy and fast to be performed.
- Some applications require connectors to properly work. In other words, **Rhea Admin Console acts as an orchestrator for many services.**
- Logs must be easy to be read and monitored.
- PaperSkiepp needs a simple way to charge and detract credit from users.
- The user interface is really simple as it provides only the needed services.

Instead, Grafana was added to the system to have a centralized monitor system, allowing fast troubleshooting for hardware faults and network congestions.

3.10.2 Installation and configuration – Rhea Admin Console

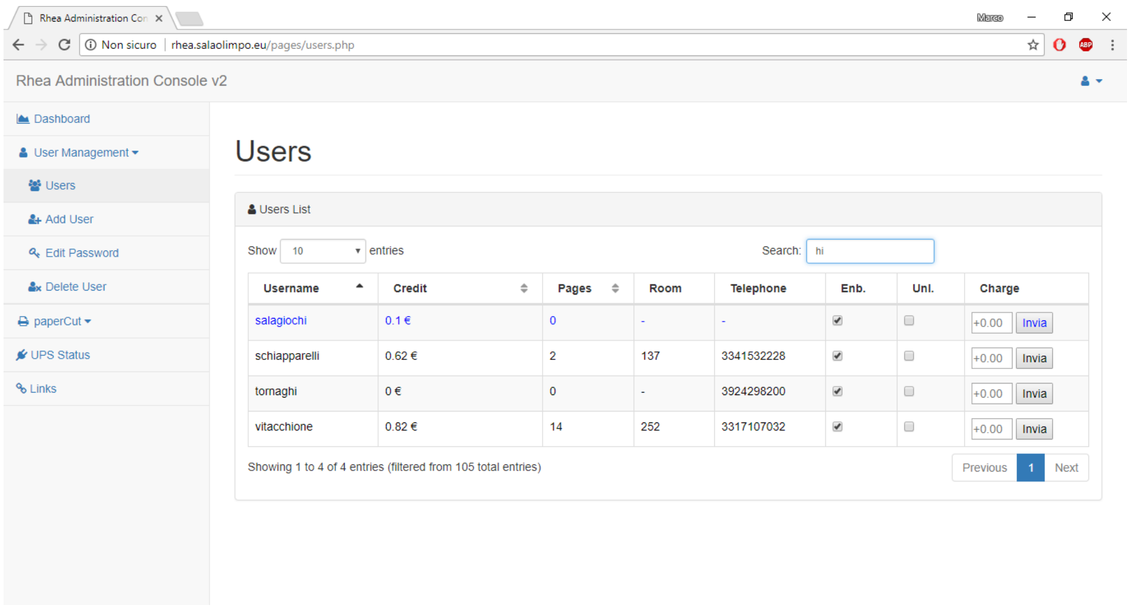
The layout is developed using the Bootstrap Template **SB Admin 2** [47], as the use of this framework made the responsive part already embedded in the page code. It will be discussed briefly as it is not the central topic of this thesis. In particular, only its functionalities will be presented with a very short description.

(1) ***LAMP*** *is an archetypal model of web service stacks, named as an acronym of Linux, Apache, MySQL, and PHP.*

It is composed by:

- A Dashboard containing the last operations performed (e.g.: created user xxx, charged xxx euro to user yyy, etc.). This operation list is simply retrieved from the MySQL database.
  Daily operations are meant to be performed using Rhea or **David** services by means of Alexa or the Telegram bot (see *Chapter 4*).
- Some graphs containing the printer usage, the euro charged, and some other counters. Since the entire printer service is custom, the charge operations are very easy to log. Also, **automatic charging** operations can be set.
- A user management section (*Figure 3.25*) used to enable and disable users, charge money, unlock the print account, etc. This set of functionalities are performed using the php **ldap extension**, which operates directly on LDAP.
- A user section to perform creation of new users or groups. These functionalities are achieved using Rhea Admin Console as an **orchestrator**, as during user creation some other operations are performed, like:
  - The user is created and added to LDAP.
  - The user is loaded into PaperSkiepp database.
  - The user folder is predisposed for Nextcloud use (*see Chapter 3.8*).

  The adoption of a central orchestrator allows an efficient and centralized connector system, which makes this custom environment manageable.



Users

Users List

Show 10 entries Search: hi

Username	Credit	Pages	Room	Telephone	Enb.	Unl.	Charge
salagiochi	0.1 €	0	-	-	✓	☐	+0.00 Invia
schiapparelli	0.62 €	2	137	3341532228	✓	☐	+0.00 Invia
tornaghi	0 €	0	-	3924298200	✓	☐	+0.00 Invia
vitacchione	0.82 €	14	252	3317107032	✓	☐	+0.00 Invia

Showing 1 to 4 of 4 entries (filtered from 105 total entries)

Previous 1 Next

Figure 3.43: The user management section of Rhea Admin Console.

A drawback of using a centralized orchestrator is the prohibition of performing manual operations, as they can lead to inconsistent data, incomplete operations performed, and logs absence (i.e. they can be hardly retrieved since they are spread among different machines).

3.10.3 Installation and configuration – Grafana

Grafana can retrieve data from different **Data Sources**. In this case, only the InfluxDB data source (*see Chapter 3.7*) is configured by setting the database connections options on the application, running on the machine on port 3000 [48].

Next, plugins can be downloaded and installed allowing the use of different data sources and graphics elements [49].

Grafana organizes different data presentations by means of **Dashboards**. They are composed of individual **Panels** arranged on a grid, and each Panel can interact with data from any configured Data Source.

As example, three Dashboard were created:

- System Health.
- Network Load.
- VM Information.

These dashboards are filled with many different panels to form an efficient (and possibly attractive) layout. An example is given in *Figure 3.26*.

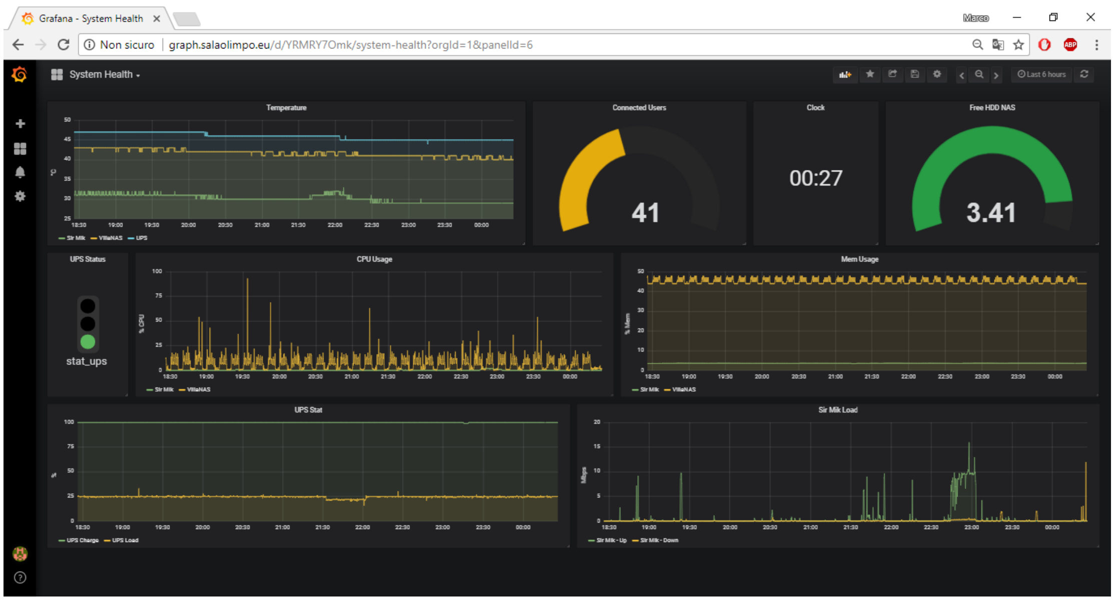


Figure 3.44: The Graphana System Health dashboard.

For each Panel one or more series can be chosen by means of an **InfluxQL** query, selecting also time ranges and measurements. Many other options can be configured, such as panel and series names, legends, axes, and others.

The selection of different ranges of time is performed natively by the system, along with the auto-refresh option. These features are great to obtain either an aggregated value over a time span, or display a behaviour in a specific range of time.

It is important to notice that Grafana displays in graphic form the data collected from different system components, by means of solutions like SNMP or **Telegraf**(1).

(1) ***Telegraf*** *is an agent written in Go for collecting, processing, aggregating, and writing metrics.*

Telegraf was installed on each production VM, so that every fifteen seconds a metric for a particular resource is stored in the InfluxDB along with the timestamp.

The resources to be monitored can be listed in the Telegraf configuration file [50]. An example (for this VM in particular) is given in *CodeView 3.8*.

```
[agent]
  interval = "15s"

[[outputs.influxdb]]
  url = "http://10.0.10.14:8086"
  database = "grafana"
  precision = "s"

[[inputs.cpu]]
  name_override = "cpu_17"
  percpu = false
  report_active = true

[[inputs.disk]]
  mount_points = ["/"]
  name_override = "disk_17"

[[inputs.mem]]
  name_override = "mem_17"
```

CodeView 3.8: An example of Telegraf configuration for Rhea machine.

3.10.4 Final notes

These services are really important for the system, since they are basically portals daily-accessed by admins. Even if they are not vital for the environment, the adoption of these services allowed lot of time saving for operations and troubleshoot.

3.11 RielloBox

3.11.1 Introduction

This virtual machine is a **vMA**[1] used to proper handle power outages performing a graceful shutdown of all the virtual machines and the physical server itself.

Figure 45: The Riello UPS logo.

[1] *A **vSphere Management Assistant** (vMA) is a pre-packaged Linux virtual machine in which third-party agents used to manage ESX and ESXi systems can be deployed.*

In particular, the vMA is distributed along with the **Riello Vision 1500 UPS** as an **ova** file, and it was imported and configured in the server for this UPS model.

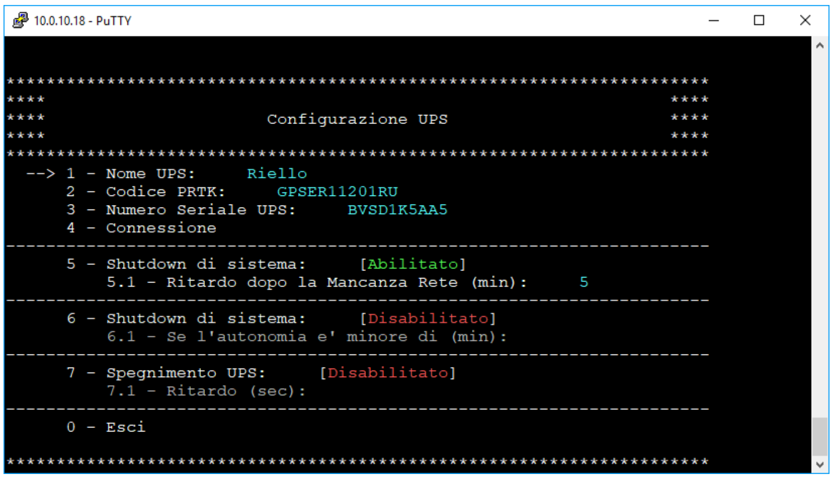
This ova contains a SUSE Linux, later configured to include these two software:

- **upsmon** (pre-installed) to perform the shutdown operations [51].
- **NUT** (Network Ups Tools), used to expose information about the UPS to other applications, such as Rhea Admin Console or David [52].

3.11.2 Installation and configuration

This VMA will interact with the VMware host to execute its set of functions.

One of these function is the **upsmon** service, configured by using its built-in scripts and specifying information such as the **PRTK code** (identifying the family of the product) and the connection type (USB) and port, as shown in *Figure 3.28*.



```
*************************************************************************
****                                                                 ****
****                        Configurazione UPS                       ****
****                                                                 ****
*************************************************************************
  --> 1 - Nome UPS:     Riello
      2 - Codice PRTK:     GPSER11201RU
      3 - Numero Seriale UPS:     BVSD1K5AA5
      4 - Connessione
-------------------------------------------------------------------------
      5 - Shutdown di sistema:     [Abilitato]
          5.1 - Ritardo dopo la Mancanza Rete (min):     5
-------------------------------------------------------------------------
      6 - Shutdown di sistema:     [Disabilitato]
          6.1 - Se l'autonomia e' minore di (min):
-------------------------------------------------------------------------
      7 - Spegnimento UPS:     [Disabilitato]
          7.1 - Ritardo (sec):
-------------------------------------------------------------------------
      0 - Esci

*************************************************************************
```

Figure 3.46: The upsmon configuration using the built-in scripts.

For the NUT part, once downloaded and installed, it can be configured only by editing some files. In particular, among these files the UPS driver needed must be specified (in our case, *riello_usb*), and some other options like users and password.

The solution adopted is to make NUT data accessible to other services using an Apache2 server exposing a **REST API** which provides the UPS information.

This comes in form of a PHP script that simply calls the NUT scripts and formats the response as a JSON object.

A portion of a possible response is reported in *CodeView 3.9*.

```
{
   "battery.capacity":"9",          "battery.charge":"100",          "battery.runtime":"1380",
   "battery.voltage":"41.0",        "battery.voltage.nominal":"36",  "driver.name":"riello_usb",
   "input .frequency":"50.00",      "input .voltage":"220",          "ups.power.nominal":"1500",
   "output.frequency":"50.00",      "output.frequency.nominal":"50.0",  "ups.temperature":"42",
   ...
}
```

CodeView 3.9: An example of output from the Apache2 API, reporting Riello UPS info from NUT.

3.12 VillaWeb

3.12.1 Introduction

This virtual machine will be used to contain many exposed services. These are addressed using different ports and reached with different hostnames using the proxy-pass rules (see *Chapter 3.9*).

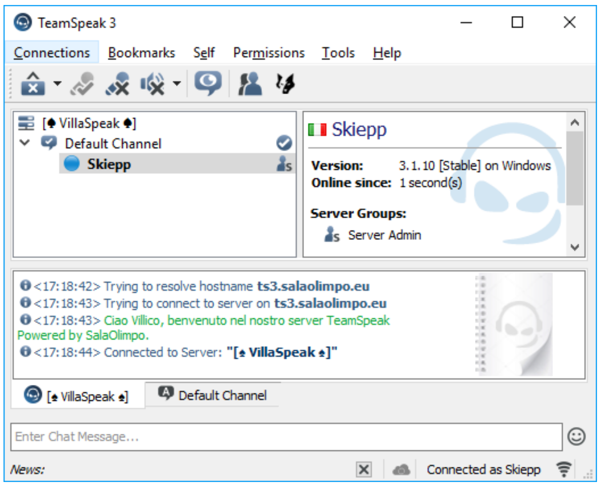
As shown in *Figure 3.29*, the exposes services are:

- **TeamSpeak3** [53], a proprietary **VoIP** software for audio communication. It is used by the team to make meetings and brainstorming.
- **WebCollab** [54], an easy web application for **project management**. It is used as a shared to-do list and **ticketing system**.
- **MediaWiki** [55], a free and open-source **wiki software**, used by many websites including Wikipedia, Wiktionary and Wikimedia Commons. It is actually used as a guide for future generations and as a task list to the deployment on ACC and PROD.

Figure 3.47: The logo of the exposed services: TeamSpeak, WebCollab, MediaWiki.

3.12.2 Installation and configuration – TeamSpeak3

TeamSpeak can be installed by downloading it through the TeamSpeak official site. After the file extraction, it can be run or stopped using the provided scripts. The TeamSpeak client screen is shown in *Figure 3.30*.



Skiepp
Version: 3.1.10 [Stable] on Windows
Online since: 1 second(s)
Server Groups:
Server Admin

```
<17:18:42> Trying to resolve hostname ts3.salaolimpo.eu
<17:18:43> Trying to connect to server on ts3.salaolimpo.eu
<17:18:43> Ciao Villico, benvenuto nel nostro server TeamSpeak
Powered by SalaOlimpo.
<17:18:44> Connected to Server: "[♠ VillaSpeak ♠]"
```

Figure 3.48: The TeamSpeak3 Client.

The ports used by this programs need to be open on the system, and it can be done using the **iptables** utility over Debian distribution. These ports are:

- UDP 9987, used for the voice.
- TCP 10011, used for **ServerQuery**, a command-line interface built into the server with scripting and automation tools.
- TCP 30033, used for the **file transfer** service.

This service extremely versatile, being it a light communication software. TeamSpeak was a perfect choice since it is freeware, easy to install and configure, and creates few network traffic. Another important aspect is that it doesn't require account for users but just a password, making it easy for users to connect to.

3.12.3 Installation and configuration – WebCollab

As a PHP-based web application, WebCollab can be downloaded from the vendor site. Next, a **virtual host** in Apache2 must be created and configured to listen on a specific port. An example of configuration is shown in *CodeView 3.10*.

```
<VirtualHost *:8080>
        DocumentRoot "/var/www/html/webcollab"

        TransferLog /var/log/apache2/webcollab_access.log
        ErrorLog /var/log/apache2/webcollab_error.log
</VirtualHost>
```

CodeView 3.10: An example configuration file for WebCollab over Apache2.

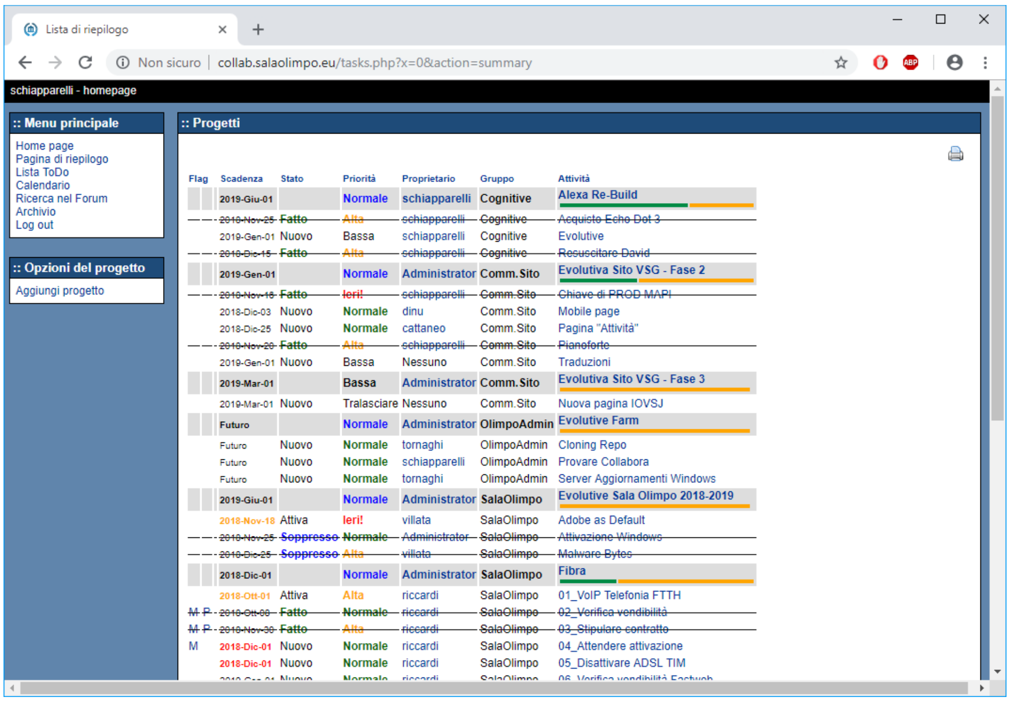
It can be configured using its PHP-based GUI, requiring only to input the MySQL database property (*see Chapter 3.7*).

A little patch was applied to this software as it doesn't completely support LDAP authentication. The login script was thus edited.

Since the software stores username in the database, the script is edited as follow:

1. When the user hits the login button it performs a bind on the LDAP server (*see Chapter 3.3*).
2. Using the same LDAP connection, a search is performed using *sAMAccountName* and *memberOf(Domain Admins)* as filters.
3. If the search found a user, then the username is searched in the database:
   - If not present, then it is inserted. Note that it is inserted password-less, since authentication is still demanded to the LDAP server.
4. Call the standard login procedure.

WebCollab was a good choice to organize teamwork since it is freeware and easy to install, configure and patch, being PHP-based. In addition, projects and activities can be created by a user and assigned to another, making the teamwork organization efficient. A screen is proposed in *Figure 3.31*.



Flag	Scadenza	Stato	Priorità	Proprietario	Gruppo	Attività
	2019-Giu-01		**Normale**	**schiapparelli**	**Cognitive**	**Alexa Re-Build**
	~~2018-Nov-25~~	~~Fatto~~	~~Alta~~	~~schiapparelli~~	~~Cognitive~~	~~Acquisto Echo Dot 3~~
	2019-Gen-01	Nuovo	Bassa	schiapparelli	Cognitive	Evolutive
	~~2018-Dic-15~~	~~Fatto~~	~~Alta~~	~~schiapparelli~~	~~Cognitive~~	~~Resuscitare David~~
	2019-Gen-01		**Normale**	**Administrator**	**Comm.Sito**	**Evolutiva Sito VSG - Fase 2**
	~~2018-Nov-16~~	~~Fatto~~	~~Ieri!~~	~~schiapparelli~~	~~Comm.Sito~~	~~Chiave di PROD MAPI~~
	2018-Dic-03	Nuovo	Normale	dinu	Comm.Sito	Mobile page
	2018-Dic-25	Nuovo	Normale	cattaneo	Comm.Sito	Pagina "Attività"
	~~2018-Nov-20~~	~~Fatto~~	~~Alta~~	~~schiapparelli~~	~~Comm.Sito~~	~~Pianoforte~~
	2019-Gen-01	Nuovo	Bassa	Nessuno	Comm.Sito	Traduzioni
	2019-Mar-01		**Bassa**	**Administrator**	**Comm.Sito**	**Evolutiva Sito VSG - Fase 3**
	2019-Mar-01	Nuovo	Tralasciare	Nessuno	Comm.Sito	Nuova pagina IOVSJ
	Futuro		**Normale**	**Administrator**	**OlimpoAdmin**	**Evolutive Farm**
	Futuro	Nuovo	Normale	tornaghi	OlimpoAdmin	Cloning Repo
	Futuro	Nuovo	Normale	schiapparelli	OlimpoAdmin	Provare Collabora
	Futuro	Nuovo	Normale	tornaghi	OlimpoAdmin	Server Aggiornamenti Windows
	2019-Giu-01		**Normale**	**Administrator**	**SalaOlimpo**	**Evolutive Sala Olimpo 2018-2019**
	2018-Nov-18	Attiva	Ieri!	villata	SalaOlimpo	Adobe as Default
	~~2018-Nov-25~~	~~Soppresso~~	~~Normale~~	~~Administrator~~	~~SalaOlimpo~~	~~Attivazione Windows~~
	~~2018-Dic-25~~	~~Soppresso~~	~~Alta~~	~~villata~~	~~SalaOlimpo~~	~~Malware Bytes~~
	2018-Dic-01		**Normale**	**Administrator**	**SalaOlimpo**	**Fibra**
	2018-Ott-01	Attiva	Alta	riccardi	SalaOlimpo	01_VoIP Telefonia FTTH
~~M P~~	~~2018-Ott-08~~	~~Fatto~~	~~Normale~~	~~riccardi~~	~~SalaOlimpo~~	~~02_Verifica vendibilità~~
~~M P~~	~~2018-Nov-30~~	~~Fatto~~	~~Alta~~	~~riccardi~~	~~SalaOlimpo~~	~~03_Stipulare contratto~~
M	2018-Dic-01	Nuovo	Normale	riccardi	SalaOlimpo	04_Attendere attivazione
	2018-Dic-01	Nuovo	Normale	riccardi	SalaOlimpo	05_Disattivare ADSL TIM
	2019-Gen-01	Nuovo	Normale	riccardi	SalaOlimpo	06_Verifica vendibilità Fastweb

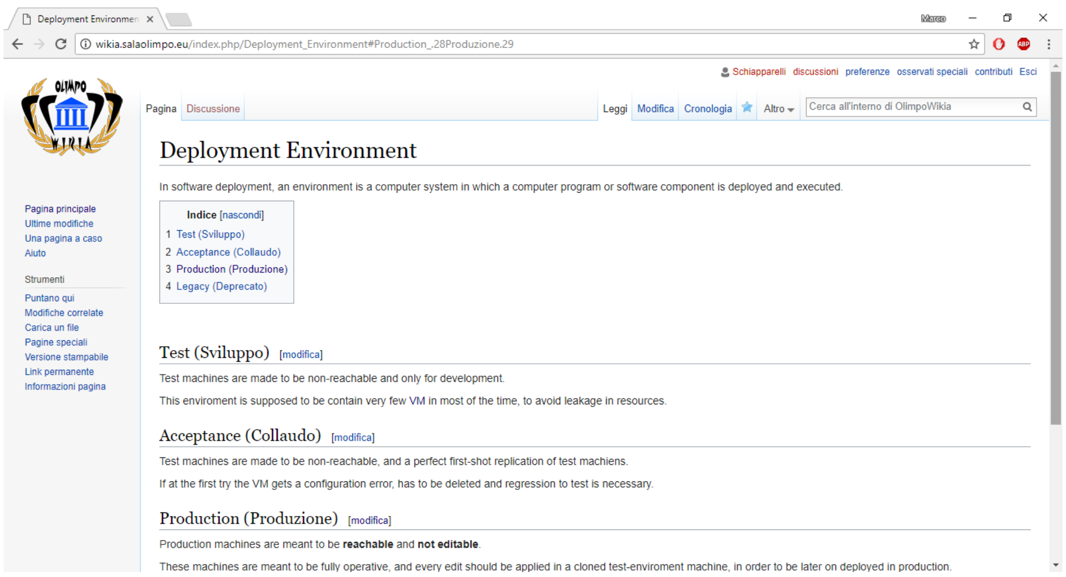
Figure 3.49: A screen of the WebCollab page with some opened tickets.

3.12.4 Installation and configuration – MediaWiki

Same as WebCollab, MediaWiki can be installed by downloading it from the official site, and enabled using an Apache2 Virtual Host. It will later be configured using the web-based GUI. An example page is presented in *Figure 3.32*.

For the LDAP authentication, the **LdapAuthentication** plugin [56] must be downloaded, installed and enabled by editing the configuration file of MediaWiki, providing information regarding the LDAP service.

MediaWiki is one of the most important services of the entire system.



Deployment Environment

In software deployment, an environment is a computer system in which a computer program or software component is deployed and executed.

Indice [nascondi]
1 Test (Sviluppo)
2 Acceptance (Collaudo)
3 Production (Produzione)
4 Legacy (Deprecato)

Test (Sviluppo) [modifica]

Test machines are made to be non-reachable and only for development.

This enviroment is supposed to be contain very few VM in most of the time, to avoid leakage in resources.

Acceptance (Collaudo) [modifica]

Test machines are made to be non-reachable, and a perfect first-shot replication of test machiens.

If at the first try the VM gets a configuration error, has to be deleted and regression to test is necessary.

Production (Produzione) [modifica]

Production machines are meant to be **reachable** and **not editable**.

These machines are meant to be fully operative, and every edit should be applied in a cloned test-enviroment machine, in order to be later on deployed in production.

Figure 3.50: The MediaWiki Development Environment page, used as example.

To assert a clean installation of other services, the configuration steps of a TEST machine are reported. Later, they are followed step by step to create the ACC machine (*see Chapter 3.2*). If the machine and its services are successfully created using all and only those steps, then it is deployed in PROD.

Else, the development process restarts from TEST by updating the wiki page.

MediaWiki was the best choice to reach this goal since it is freeware, easy to install and configure, and uses few resources. In addition, MediaWiki is wide-known, and so it has a wide community, making search for support and fixes really easy.

Finally, this service has a great sentimental value, since it reports all of the system configurations, *created with blood, sweat, and tears; not for money, but for passion.*

3.13 VillaSamba

3.13.1 Introduction

This virtual machine is used as a workaround for the problem here descripted:

- Users have an account on the NextCloud platform (*see Chapter 3.8*).
- All the PCs must have a link on the desktop pointing NextCloud data folder.
- When a user writes some files in the folder using that shortcut, NextCloud must "accept" those files (and vice-versa).
- However, **there is a permission problem** since files written through NextCloud are **owned by the www-data local user**, while the files written through desktop link are **owned by the relative domain user.**
- Therefore, this machine is created to **force file permission** to www-data.

To reach this goal, a domain- joined installation of **Samba4**(1) [57] is used. In particular, it will be used to expose a network share with permission forcing.

Figure 51: The Samba4 logo.

The Samba4 logo is shown in *Figure 3.33*.

(1) ***Samba*** *is a free re-implementation of the SMB/CIFS networking protocol, and provides file services for various Windows clients. It can also integrate with a Windows Server domain, either as a Domain Controller or as a Domain Member.*

3.13.2 Installation and configuration

Firstly, the NAS share of NextCloud must be mounted editing the `/etc/fstab` file. By doing so, the NextCloud folder on the NAS will be mounted as NFS locally on the machine, making access to the folder easier.

The entire configuration of samba can be done by just editing `smb.conf` file [58], as this is a small installation used only to force the permissions.

The working principle is shown in *Figure 3.34*.

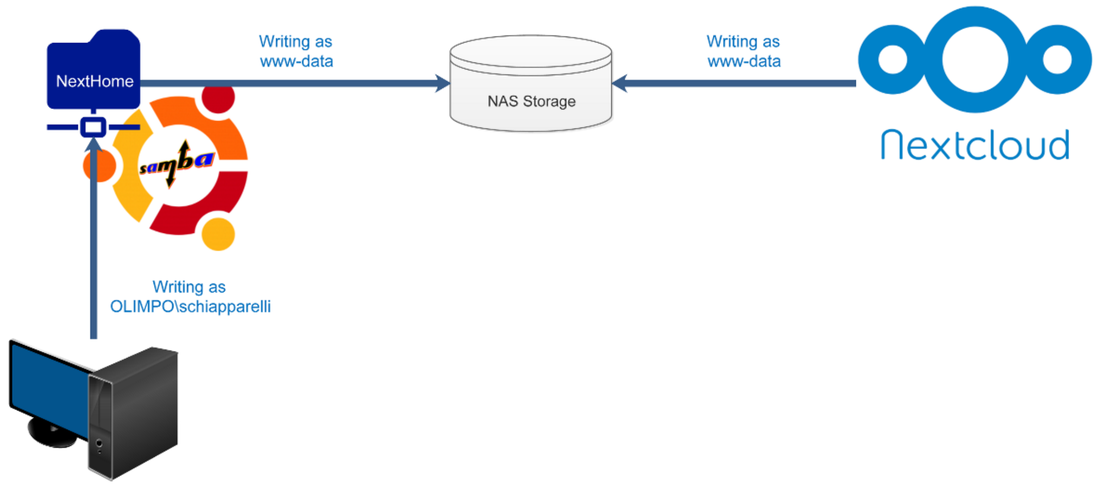


Figure 3.52: The working principle of the Samba4 used to enforce permissions.

Each section in the configuration file describes a **shared resou**rce, or simply "**share**". A share consists of a target directory plus a description of the access rights which are granted to the user of the service.

An exception is the [global] section, which applies to the server as a whole and is used as basic samba parameter section. Only the information of the domain needed to join it later was entered.

In this case, the only share configured is the *NextHome* one.

In this share, the access rights of the folder are **masked** with the access rights of the specified UNIX user, as shown in *CodeView 3.35*.

```
[NextHome]
read only          = No
create mask        = 0660
directory mask     = 0770
browseable         = yes
path               = /mnt/nextcloud/data/%U/files
force user         = www-data
force group        = www-data
```

Figure 3.53: The configuration for the NextHome share.

As shown, the rights of this folder are masked as rw, and writing is performed with the user *www-data* of group *www-data*.

3.13.3 Final notes

Using this workaround (along with the NextCloud cron job, *see Chapter 3.8*), NextCloud can correctly access these files.

Note the environment variable `%U` in the path parameter of the share: it allows to automatically access the proper NextCloud user folder on the NAS.

Once all the needed files are configured, the machine can be joined to the domain using the `net ads join` command line tools. As the setup part is complete, on each client the creation of the symbolic link in the desktop can be performed pointing to `\\VillaSamba\NextHome`.

This machine was built to overcome this really annoying problem. Even if this solution is a cheap one, it's not the best workaround if we consider the **security aspect**. As the access to this folder is masked with a single application user, if someone could login with this user on the server, then it will be able to see and modify every user folder or file.

Chapter 4: David

4.1 Introduction

This chapter is going to cover all the aspects of the David architecture.

David is a set of modules and software used to perform simple management, monitoring and troubleshooting operations.

Basically, it is composed by Java and JavaScript applications that perform the following operations:

1. It receives an input from Alexa or the Telegram bot.
2. It detects the user intent using NLP operations.
3. It performs the requested operation.
4. It creates and sends the response to Alexa or Telegram.

As said, to create a nice user experience the entire system is based on many **NLP**(1) services, instead of being command-based. For these NLP operations, it will be used some **Watson** [59] services from the **IBM Cloud** [60].

IBM Watson is a set of **cloud-computing services** for business offered by IBM. It includes software as a service (SaaS) and platform as a service (PaaS) offered through public, private and hybrid cloud delivery models. The IBM Watson logo is presents in *Figure 4.1.*

At the current time, David is composed by a limited set of functions, but the conditions are right for the development of new ones. This chapter is composed by three parts:

- The first part describes the architecture used for **Alexa** integration.
- The second part describes the integration of a **Telegram** bot, along with the connectors deployed using **Node-RED**.
- The third part describes more in details **David Service Layer**.

Figure 4.54: The IBM Watson logo.

(1) ***Natural language processing*** *(NLP) is a topic of artificial intelligence concerned with the interactions between computers and human natural languages.*

4.2 Alexa and David

4.2.1 Alexa Skills

In this section the interaction of Alexa and David is explained in details.

Amazon Alexa is a virtual assistant developed by Amazon, first used in the Amazon Echo and the Amazon Echo Dot smart speakers.

Figure 4.55: An Amazon Echo Dot 2 device.

For the **POC**[1] of the system, an Amazon Echo Dot 2 (shown in *Figure 4.2*) is used as a device for the Alexa service. Nowadays, an Amazon Echo Dot 3 (shown in *Figure 4.3*) device is used as it supports more languages like Italian, mainly used by admins [61].

Figure 4.3: An Amazon Echo Dot 3 device.

In fact, Alexa is a cloud-based service hosted with the name of **Amazon Alexa Services** (AAS), and can be exploited using these kind of devices. Functionalities are deployed on an Alexa device by means of **Skills**.

Skills are capabilities that enable customers to create a more personalized experience. There are now tens of thousands of cloud-based skills from several brands as well as individual designers and developers.

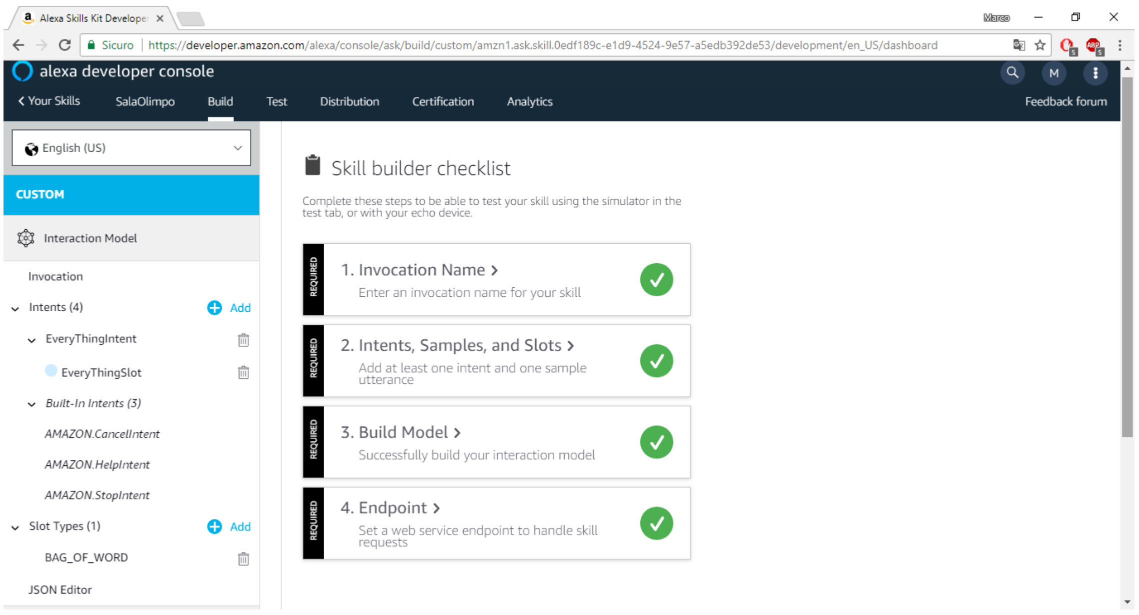
Basically, Skills follow the same principles of the app for Android and iOS devices, and thus they can be installed by the Amazon Skill store [62].

[1] A ***Proof of concept*** *(PoC) is a realization of a certain method or idea in order to demonstrate its feasibility, or verifying that some concept or theory has practical potential. A proof of concept is usually small and may or may not be complete.*

They can be installed directly using the store, or even with the Amazon app or developed and deployed automatically on the device. In particular, the Amazon website provides a service named **Alexa Skills Kit** (ASK) [63].

It is a **collection of self-service APIs**, tools, documentation, and code samples that makes fast and easy to add skills to Alexa. ASK enables designers, developers, and brands to build engaging skills and reach customers through tens of millions of Alexa-enabled devices.

In this case, a new skill named "*SalaOlimpo*" is created, using the Alexa Developer Console web application, properly configuring it as shown in *Figure 4.4*.



Skill builder checklist

Complete these steps to be able to test your skill using the simulator in the test tab, or with your echo device.

1. Invocation Name
Enter an invocation name for your skill

2. Intents, Samples, and Slots
Add at least one intent and one sample utterance

3. Build Model
Successfully build your interaction model

4. Endpoint
Set a web service endpoint to handle skill requests

Figure 4.4: The Alexa Developer Console.

Basically, there are two options for hosting a custom skill:

- Hosting as a **Web Service** [64], which requires the setup of a web server that accepts requests and sends responses to the Alexa service in the cloud. The service must be internet-accessible using HTTP over SSL/TLS on port 443, and must present a certificate with a subject alternate name that matches the domain name of the endpoint (issued in this case using Certbot, *see Chapter 3.9*).
- Hosting as an **AWS Lambda Function** [65], an Amazon Web Services offering that runs code only when it's needed and scales automatically. Supported programming languages are Node.js, Java, Go, C# and Python. In this scenario, the code can be simply uploaded and AAS executes it in response to Alexa voice interactions.

For this system the choice made was to host as a Web Service, since operations need to be performed locally by the main server Apollo. In addition, running the entire service locally allows to better manage and maintain it.

The endpoint set for this skill is an URI exposed with a public domain (*alexa.salaolimpo.cloud*) and forwarded by the NGINX proxy (*see Chapter 3.9*).

By doing so, after the Speech-To-Text performed by Alexa's Cloud services, the user request is sent as a JSON to the specified URI via https POST.

Finally, each skill has an **invocation name** to begin an interaction with it. As example, the invocation name for this skill is set to "david", same as the architecture name. By doing so, the skill can be activated by saying *"Alexa, open David"*, or *"Alexa, ask David to…"*.

4.2.3 NLP basics - Intent Recognition

Before proceeding, some basic aspects of NLP for this use case are explained.

The first crucial part of NLP is the **Intent and Entity Recognition** [66]. With this step, an input utterance is analysed to extract two components, later explained.

These sentences are given as example for this chapter:

- *"Is the VillaDNS machine ok?"*
- *"How is the DNS?"*
- *"Check all the virtual machines"*:

The two components extracted are explained below based on the previous examples:

- ✓ The intent, representing what the user is trying to do. In other words, **Intents are a discrete set of the end-user intentions**.

  Every utterance is thus translated into an intent to detect what the user is trying to request. In the examples given before, the relative intent for the first two phrases is "#*CheckMachine*", as the user is requesting to check the state of a virtual machine. For the third example, the intent is *"#CheckAllMachines"*, because the user is requesting to check all the VM and not a specific one.

  Different phrases can lead to the same intent, making the user input extremely elastic. Moreover, a good intent recognition process can overcome errors in the input utterance, generated either by the Speech-To-Text engine or by a user grammatical error.

- ✓ The entity, representing one or more target for an intent. **Entities are domain-specific word representing actors involved in an intent**.

  An utterance can contain some entities for specify additional information for a request. In addition, they can be used by the recognition engine for correctly assign intents.

In the examples given, the entity *"@VirtualMachine"* with value *"VillaDNS"* is recognized in the first two phrases, while the third one has no entities. This absence of any entity hints the recognition engine that the intent is *"#CheckAllMachines"*, and not *"#CheckMachine"*. Moreover, it can specify to the backend the machine that the user is requesting to check.

Different words can lead to the same entity, making the introduction of a **lemma**(1) possible, useful to redirect the backend to the correct target.

The extracted intent and entities for the examples are summarized in *Table 4.1*:

Is the VillaDNS machine ok?	*#CheckMachine*	*@VirtualMachine: VillaDNS*
How is the DNS?	*#CheckMachine*	*@VirtualMachine: VillaDNS*
Check all the virtual machines	*#CheckAllMachines*	-

Table 4.1: The Intents and Entities of the given examples.

For these functionalities, **Watson Assistant** [67] is used as a cloud service. It is a service of the IBM Cloud platform which, in addition to intent and entity recognition, can manage a complete user interaction.

The interaction is shaped using a **Dialog Graph**, which represents a full-dialogue with the end user. When a user asks a question, it is visited like a standard tree graph, and each step (i.e. each sentence) corresponds to visiting a child of the current node.

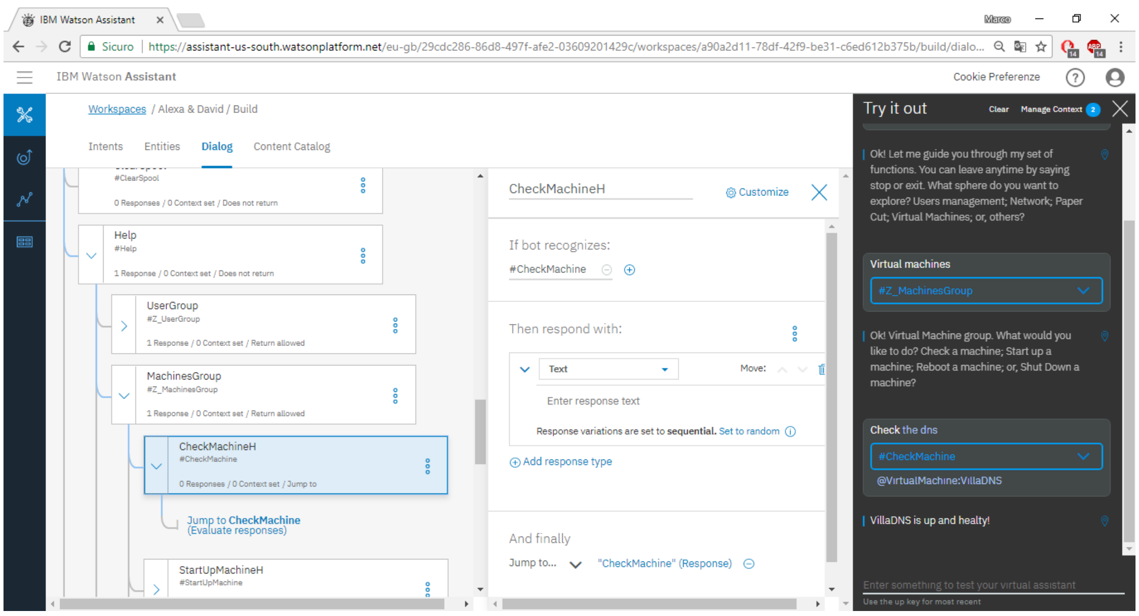
Basically, **Alexa performs only the Speech-To-Text** part, while all the dialogue handling is demanded to Watson Assistant, called by David Service Layer (*see Chapter 4.2.5*) or the Node-RED flow (*see Chapter 4.3*).

Even if Amazon Skill Kit itself supports intent and entity recognition, Watson Assistant was chosen as easier to manage and configure. Moreover, tests evidenced the Watson intent recognition gives **better results** for this kind of sentences, even if calling a third-part service introduces some delay.

Finally, Watson can be organized using different workspaces, allowing to store different languages and sandboxes, in order to test new intents and entities without creating disservices.

An example of dialog graph from Watson Assistant is shown in *Figure 4.5*. It is the actual PROD dialog graph of David, in English language.

(1) *In morphology, a **lemma** is the canonical form or dictionary form of a set of words.*



https://assistant-us-south.watsonplatform.net/eu-gb/29cdc286-86d8-497f-afe2-03609201429c/workspaces/a90a2d11-78df-42f9-be31-c6ed612b375b/build/dialo...

#ClearSpool
0 Responses / 0 Context set / Does not return

Help
#Help
1 Response / 0 Context set / Does not return

UserGroup
#Z_UserGroup
1 Response / 0 Context set / Return allowed

MachinesGroup
#Z_MachinesGroup
1 Response / 0 Context set / Return allowed

CheckMachineH
#CheckMachine
0 Responses / 0 Context set / Jump to

Jump to **CheckMachine** (Evaluate responses)

StartUpMachineH
#StartUpMachine

CheckMachineH

If bot recognizes:
#CheckMachine

Then respond with:
Text
Move:
Enter response text
Response variations are set to **sequential.** Set to random
Add response type

And finally
Jump to... "CheckMachine" (Response)

Ok! Let me guide you through my set of functions. You can leave anytime by saying stop or exit. What sphere do you want to explore? Users management; Network; Paper Cut; Virtual Machines; or, others?

Virtual machines
#Z_MachinesGroup

Ok! Virtual Machine group. What would you like to do? Check a machine; Start up a machine; Reboot a machine; or, Shut Down a machine?

Check the dns
#CheckMachine
@VirtualMachine:VillaDNS

VillaDNS is up and healty!

Enter something to test your virtual assistant
Use the up key for most recent

Figure 4.5: Watson Assistant dialog graph.

4.2.4 Watson Assistant model creation

Watson Assistant provides a web interface to build the entire dialog system. The first step to be performed is the **intents creation** [68]. Every intent created in Watson Assistant is **mapped with a specific function** in David service layer.

To create an intent, a name must be provided along with some example of utterances. For example, for the "*#CheckMachine*" intent, some examples are:

- *Can you check @VirtualMachine?*
- *Check @VirtualMachine?*
- *How is @VirtualMachine?*
- *Is @VirtualMachine online?*

Examples should be representative and typical of data that end-users will provide. Note the notation *@VirtualMachine*, which represent an entity of that type, which will be later discussed.

After the intent creation, Watson starts to train the model on the examples given. By using the "Try it out" button on the top-right side of the web page, the intent recognition can be tested and corrected.

Reporting a correction manually is extremely useful as Watson uses **Machine Learning**[(1)] algorithms to train the model to correctly detect intents. This is an **iterative** paradigm, as ML train requires **fine tuning**[(2)] to be performed repetitively to achieve a good level of correctness and confidence.

[(1)] ***Machine Learning** is a field of computer science that uses statistical techniques to give computer systems the ability to learn, using data and without being explicitly programmed.*

[(2)] *A **fine tuning** means making small adjustments to in order to improve a service.*

The second step to be performed is the entities creation. Every entity created in Watson Assistant **is mapped with a specific resource** in David service layer.

To create an entity, a name must be provided along with some example of values for that entity. For example, for the *“@VirtualMachine”* entity, some values are:

- VillaDNS
- VillaRADIUS
- PaperSkiepp
- VillaDB

For each of these “main values”, many examples should be given. This is very important since the use of a main lemma simplifies a lot the backend operations.

An interesting fact is that being these entities domain-specific, the Alexa’s Speech-To-Text engine can misunderstand those words. Examples are the words *“Villa”*, which is an Italian word, and *“RielloBox”* which is a made-up name.

In other words, these words may contain **noises** generated by Alexa, making the detection of the lemma difficult to perform. To overcome this problem, some of those misunderstood words are added to Watson as **Synonyms** of the main lemma.

In addition, the “**Fuzzy Matching**” option can be enabled, helping the entity recognition engine to work in presence of noises.

As example, some synonyms of entity values are reported in *Table 4.2*:

VillaDNS	*DNS, Bind, Bind 9, dot 11*	*the ns, vila dns, vila the ns*
VillaDB	*Database, SQL, My SQL, dot 14*	*vila the bee, data basil, deck of basic*
RielloBox	*Riello, UPS, battery, dot 18*	*real a box, real low, rio boxer*

Table 4.2: Some example of noisy synonyms for some lemmas.

Note the presence of very noisy terms like “*real a box*” and “*data basil*” to cover misunderstanding of the words “*RielloBox*” and “*database*”.

This workaround actually gives pretty good results, and is one of the few options available to handle the impossibility of modelling the Alexa Speech-To-Text engine with domain-dependant words.

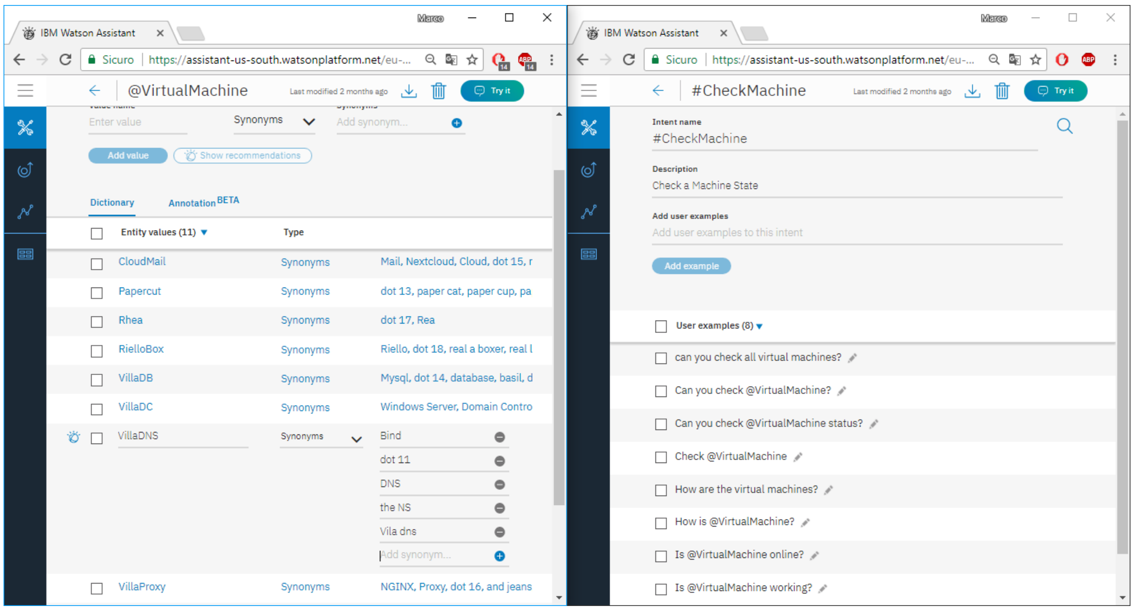
Another option for entity creation is the possibility to define patterns, in form of **regex expression**. Examples of entities defined with patters are:

- *@IP*, with regex `(\d{1-3}[.]){3}\d{1-3}`
- *@Room*, with regex `[123][0-9]{2}`
- *@MAC*, with regex `([0-9A-Fa-f]{2}[:]){5}[0-9A-Fa-f]{2}`

As entities are required for some operations to be performed, Watson Assistant provides a special feature called **Slots**. More specifically, slots are used to gather multiple pieces of information from a user within that node, and the service asks only for the missing details required.

Of course, specific questions capable of asking the missing information can be set for each slot.

To end this chapter, a screen of intent and entity creation using the Watson Assistant web interface is shown in *Figure 4.6*.



@VirtualMachine

Enter value | Synonyms | Add synonym...

Entity values (11)	Type	
CloudMail	Synonyms	Mail, Nextcloud, Cloud, dot 15, r
Papercut	Synonyms	dot 13, paper cat, paper cup, pa
Rhea	Synonyms	dot 17, Rea
RielloBox	Synonyms	Riello, dot 18, real a boxer, real l
VillaDB	Synonyms	Mysql, dot 14, database, basil, d
VillaDC	Synonyms	Windows Server, Domain Contro
VillaDNS	Synonyms	Bind dot 11 DNS the NS Vila dns Add synonym...
VillaProxy	Synonyms	NGINX, Proxy, dot 16, and jeans

#CheckMachine

Intent name: #CheckMachine

Description: Check a Machine State

Add user examples: Add user examples to this intent

User examples (8)

- can you check all virtual machines?
- Can you check @VirtualMachine?
- Can you check @VirtualMachine status?
- Check @VirtualMachine
- How are the virtual machines?
- How is @VirtualMachine?
- Is @VirtualMachine online?
- Is @VirtualMachine working?

Figure 4.6: The intent and entity creation with Watson Assistant.

4.2.5 Service Layer backed – Tomcat and Spring Boot

The Watson Assistant service is called using the REST API service [69] by a Java application running on an **Apache Tomcat**(1) server [70], which represents the core part of the David software.

This application provides REST services which can be called either by Alexa by means of the Amazon Alexa Services; or the Telegram bot by means of the Node-RED flow (*see Chapter 4.3*).

In particular, the service layer can be called with a POST request (made by AAS), which after the conclusion of the operations sends a JSON-formatted response. It is written using the **Spring Boot** framework [71].

(1) ***Apache Tomcat*** *is an open-source Java Servlet Container developed by the Apache Software Foundation. It implements several Java EE specifications and provides a Java HTTP web server environment in which Java code can run.*

Spring Boot is Spring[2] Framework's convention-over-configuration solution for creating stand-alone and production-grade Spring-based Applications. It provides a preconfigured framework for creating standalone Spring application without need for XML configuration or code generation.

The basic architecture of the system, later described, is displayed in the *Figure 4.7*.

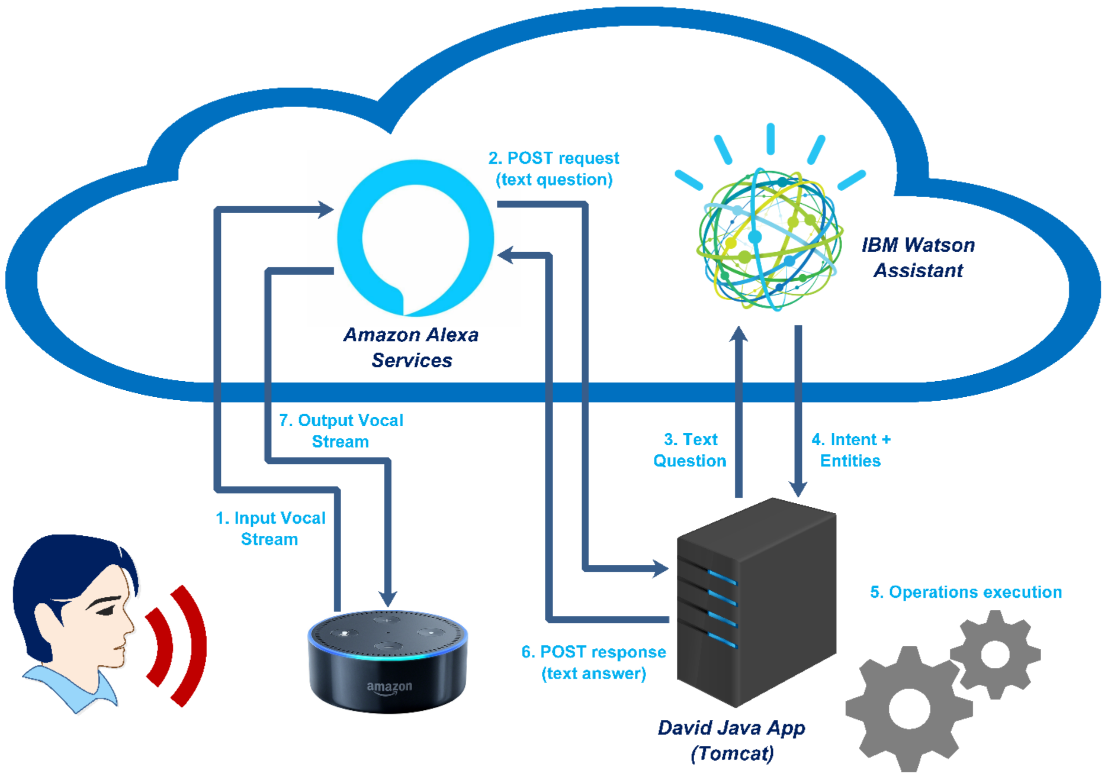


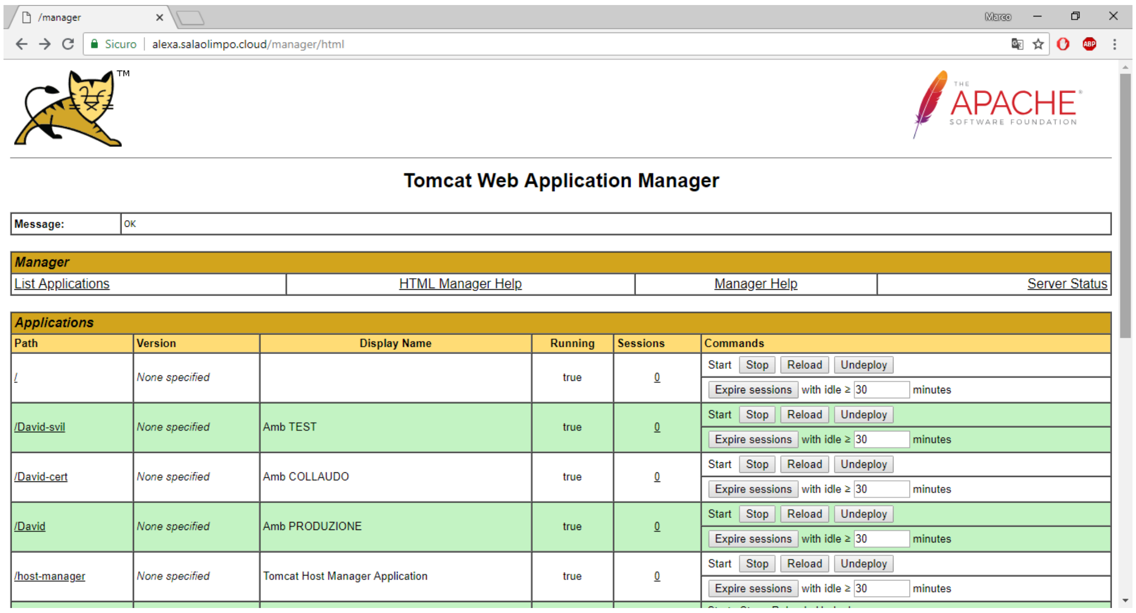
Figure 4.7: The architecture of the Alexa and David solution.

This servlet is hosted by a new virtual machine in the Virtual Farm, just like the other ones already discussed (*see Chapter 3*). However, this machine contains only the Tomcat server (which doesn't require particular configurations) and Node-RED along with Node.js, described in the next chapters (*see Chapter 4.3*).

Deployment of the services can be made remotely using a **Maven** run configuration. Apache Maven [72] is a software project management and comprehension tool. It can be used to manage a project's build, reporting and documentation.

Same as the virtual farm development, this software is build using the three development environment already discussed (*see Chapter 3.2*). Therefore, the service layer is deployed in TEST, ACC and PROD environments, as shown in the **Tomcat Web Application Manager** displayed in *Figure 4.8*.

[2] *The **Spring Framework** is an application framework and inversion of control container for the Java platform.*


/manager

Sicuro | alexa.salaolimpo.cloud/manager/html

Tomcat Web Application Manager

Message:	OK

Manager

List Applications	HTML Manager Help	Manager Help	Server Status

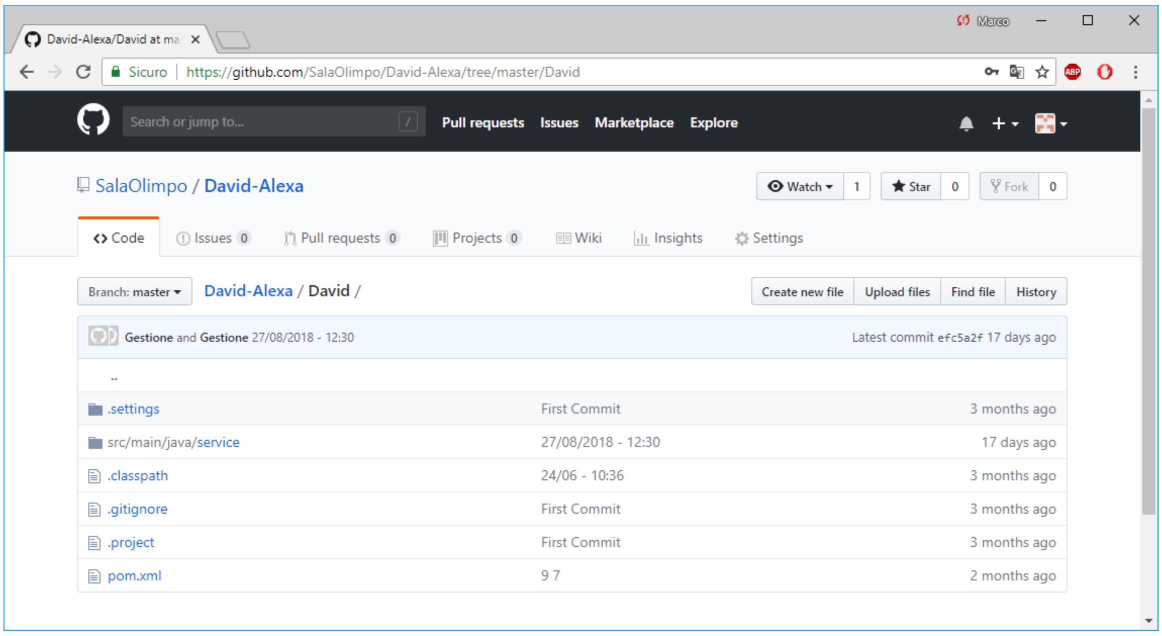
Applications

Path	Version	Display Name	Running	Sessions	Commands
/	None specified		true	0	Start Stop Reload Undeploy / Expire sessions with idle ≥ 30 minutes
/David-svil	None specified	Amb TEST	true	0	Start Stop Reload Undeploy / Expire sessions with idle ≥ 30 minutes
/David-cert	None specified	Amb COLLAUDO	true	0	Start Stop Reload Undeploy / Expire sessions with idle ≥ 30 minutes
/David	None specified	Amb PRODUZIONE	true	0	Start Stop Reload Undeploy / Expire sessions with idle ≥ 30 minutes
/host-manager	None specified	Tomcat Host Manager Application	true	0	Start Stop Reload Undeploy / Expire sessions with idle ≥ 30 minutes

Figure 4.8: The Apache Tomcat Manager screen.

Using the M2E plugin in Eclipse, a run configuration was created to automatically deploy the software on the Tomcat Server, and each instance of the application can be started or stopped using the Web Application Manager [73].

In addition, a **GitHub**[1] **repository** is created to host the code of the service layer, using the Git-Plugin for Eclipse to perform all the **Push** and **Pull** operations. As a public repository, the code can be found and cloned from its GitHub page, located in *https://github.com/SalaOlimpo/David-Alexa* [74] as shown in *Figure 4.9*.


David-Alexa/David at ma

Sicuro | https://github.com/SalaOlimpo/David-Alexa/tree/master/David

SalaOlimpo / David-Alexa

Branch: master ▾ David-Alexa / David /

Gestione and Gestione 27/08/2018 - 12:30 — Latest commit efc5a2f 17 days ago

..		
.settings	First Commit	3 months ago
src/main/java/service	27/08/2018 - 12:30	17 days ago
.classpath	24/06 - 10:36	3 months ago
.gitignore	First Commit	3 months ago
.project	First Commit	3 months ago
pom.xml	9 7	2 months ago

Figure 4.9: The GitHub page for the project.

[1] ***GitHub Inc.*** *is a web-based hosting service for version control using Git. It offers distributed version control and source code management functionalities of Git as well as other features, like access control, bug tracking, task management and wikis.*

The remote deployment, along with the GitHub repository, encouraged teamwork and made the development easy and the code portable for this application.

4.2.6 An example of user request handling with Alexa

As shown in a previous image (*Figure 4.7*), the entire system works as follow:

1. After receiving the **Wake Word** (*"Alexa"*) and the user command (e.g. *ask David to disable user 137*), the input is sent as an audio stream to Amazon Alexa Services to be analysed. Here, the Speech-To-Text is performed, rewriting the user input in textual form.
2. Since the request contains the **Invocation Name** of the SalaOlimpo skill (formerly *David*), Amazon performs a *dummy* intent recognition on the command for that specific skill.
   In fact, the only one configured is "*EverithingIntent*", as intent recognition is demanded to Watson. This solution was chosen as Watson is more customizable and can be **shared** with the Telegram bot architecture.
   In other words, AAS sends to David Service Layer an input message with intent *EverithingIntent* and the actual user input as *Entity*.
3. A POST request is sent to the configured endpoint, which is in fact the David Service Layer, **stating its execution**.
   The information sent from AAS are now parameters used by David.
4. First of all, using the local MySQL database, it retrieves Watson Assistant's **Context Object**, used as a state variable to continue a previous conversation. It is prepared to be sent along with the new text to Watson Assistant, using a basic-authenticated POST request.
   This solution is provided natively by Watson Assistant in response to the stateless nature of a REST API service.
5. David SL sends the request to Watson, and retrieves an updated Context Object, along with one intent and zero or more entities.
6. The Context Object is updated in the database, and the query is logged.
7. A simple **switch** operation is performed on the intent returned, along with the extraction of the entities contained in Watson response.
8. The selected action is performed, and on output text based on the result is generated and formatted for AAS.
9. The response is sent back to AAS, which performs the Text-To-Speech.
10. The audio is sent to the Echo Dot device, that reproduces it concluding the entire flow.

Other details about the application are be given in next sections (*see Chapter 4.4*).

Table 4.3 shows a list of implemented functions with examples for sentences. Each one can be called using *“Alexa, ask David to…”* or after *“Alexa, open David”*.

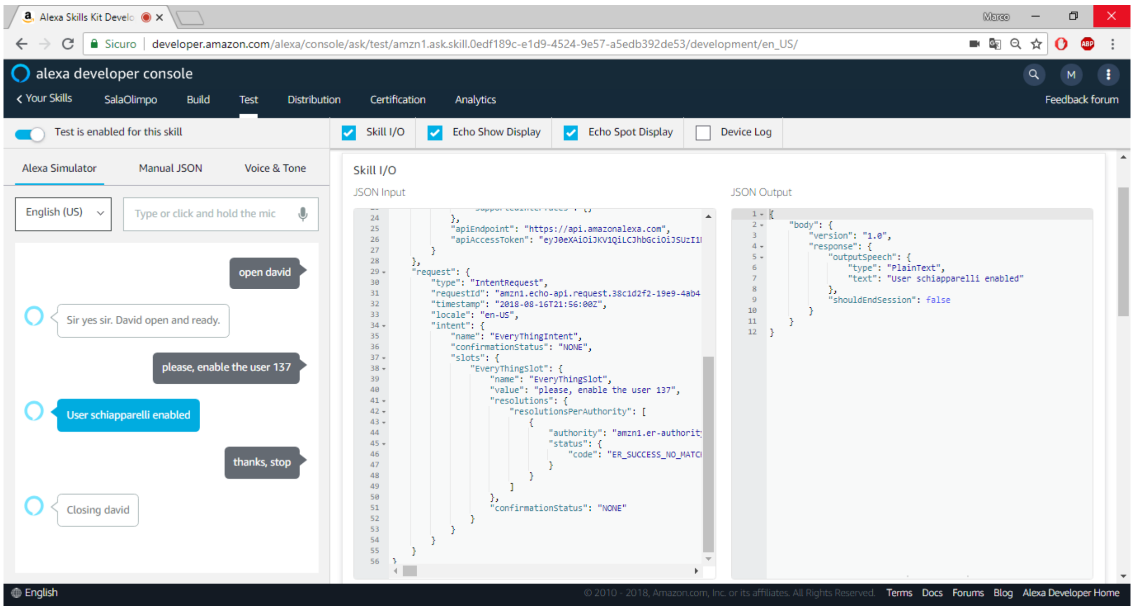
Charge or detract money from a user.	*“Add 3 euros to 137”*
Check one or more machine status.	*“Check the dns”*
Clear Alexa command history.	*“Clear the history”*
Clear CUPS’s printing queue.	*“Clear the printer spool”*
Promote or demote a user to admin.	*“Promote 137 to admin”*
Enable or disable a user.	*“Enable the user 137”*
Drop the network leases.	*“Drop the leases”*
Check the network load.	*“Check the network load”*
Count the connected people.	*“How many users are online?”*
Reboot the MikroTik.	*“Reboot the router”*
Renew Certbot’s certificates.	*“Renew the cerficates”*
Reset all the Access Points (CAPs).	*“Reset all the access points”*
Shutdown, startup, or restart a VM.	*“Shutdown the radius”*
Re-configure the CAPs channels.	*“Setup the channels”*
Cancel and stop a print job.	*“Stop the printer!”*
Increment or decrement the user speed.	*“Reduce the user limit”*
Insert a MAC in VLAN 21 whitelist.	*“Unlock the MAC 11:11:11:11:11:11”*
Check if the UPS is online / on battery.	*“Check the UPS”*

Table 4.3: Some implemented functionalities with examples.

In addition, by saying “Help”, “Guide me”, or similar sentences; Alexa prompts a vocal menu with all the implemented functions. It is useful if the user can’t remember a command syntax (even if Watson Assistant ML functionality gives some room for the correct understanding of user intentions).

To conclude this chapter, the screen in *Figure 4.10* shows a test made using the **Alexa Developer Console**, accessible by web interface. It is useful for testing, as the entire body sent by the POST request is reported, allowing an easy diagnostic and troubleshooting process pre and post Service Layer.

In addition, the Developer Console allows also to test vocal input using the PC microphone to test the AAS Spech-To-Text for the skill.



open david

Sir yes sir. David open and ready.

please, enable the user 137

User schiapparelli enabled

thanks, stop

Closing david

```
            },
            "apiEndpoint": "https://api.amazonalexa.com",
            "apiAccessToken": "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1
        }
    },
    "request": {
        "type": "IntentRequest",
        "requestId": "amzn1.echo-api.request.38c1d2f2-19e9-4ab4
        "timestamp": "2018-08-16T21:56:00Z",
        "locale": "en-US",
        "intent": {
            "name": "EveryThingIntent",
            "confirmationStatus": "NONE",
            "slots": {
                "EveryThingSlot": {
                    "name": "EveryThingSlot",
                    "value": "please, enable the user 137",
                    "resolutions": {
                        "resolutionsPerAuthority": [
                            {
                                "authority": "amzn1.er-authorit
                                "status": {
                                    "code": "ER_SUCCESS_NO_MATC
                                }
                            }
                        ]
                    },
                    "confirmationStatus": "NONE"
                }
            }
        }
    }
}
```

```
{
    "body": {
        "version": "1.0",
        "response": {
            "outputSpeech": {
                "type": "PlainText",
                "text": "User schiapparelli enabled"
            },
            "shouldEndSession": false
        }
    }
}
```

Figure 4.10: The Alexa Developer Console testing the SalaOlimpo skill.

4.3 Telegram and Node-RED

4.3.1 Creating a bot

In addition to the Alexa Skill, David serves user requests coming from a **Telegram** bot. So, in this section the interaction of Telegram and David is explained in detail.

Telegram is a cloud-based instant messaging and VoIP service. Alongside the standard messaging utility, Telegram can be used to call functions and services by means of **bots** [75]. **Bots are Telegram accounts operated by programs**. They can respond to messages, can be added to groups and can be integrated with other programs. They can be messaged exactly like common users.

In this case, a new bot named *"David"* with ID *"@salaolimpo_bot"* is created, using **The BotFather** [76]. Basically it is itself a bot used to generate new bots and retrieve the **HTTP token** for them, using the `/newbot` function.

After retrieving the token, the bot can be handled using **Node-RED** [77]. The BotFather logo is shown in *Figure 4.11.*

Figure 4.561: The BotFather logo.

Node-RED is a flow-based development tool developed originally by IBM for wiring together devices, APIs and online services as part of the **IoT**[1]. It provides a browser-based flow editor, which can be used to create JavaScript functions.

Node-RED is installed in the same virtual machine hosting the David Service Layer (*see Chapter 4.2*) alongside its run-time environment, which is **Node.js**[2].

---

[1] *The **Internet of Things** (IoT) is the network of physical devices, home appliances, and other items embedded with electronics, software and sensors which enables these things to connect and exchange data. It encourages integration of the physical world into computer-based systems, resulting in efficiency improvements, economic benefits, and reduced human exertions.*

[2] ***Node.js** is an open-source and cross-platform JavaScript run-time environment that executes JavaScript code outside of a browser.*

Once installed, it can be used as a web application, and the first operation to be performed is the **palette** installation [78]. They are essentially a set of **nodes** that can be added to the standard ones, and can be used to performs specific tasks.

A Node-RED node represents a function itself. They are linked with connectors, which are useful to draw a graphical flow for each packet going through the system. Node-RED comes with a core set of pre-installed nodes, but there are a growing number of additional ones available for install into Node-RED projects.

More specifically, the most important palettes installed are:

- `contrib-telegrambot`, used to manage the Telegram bot.
- `node-watson`, used to call Watson cloud services.
- `contrib-influxdb`, used to save data in InfluxDB (*see Chapter 3.14*).
- `node-snmp`, used to retrieve data using the SNMP protocol.
- `contrib-key-value-store`, `contrib-splitter` and minors.

Nodes can be deployed in many **flows**, each one ideally representing a functionality. Each flow is crossed from left to right by means of a JSON object named **msg**, which contains a field named **payload** mostly used to carry information, some ID fields representing the message, and other custom fields.

Flows contain also some debug nodes (or **Torch**) which can be used to check the content of the msg object and/or its msg.payload object.

An example of flow (the David Main flow) is presented in *Figure 4.12*.

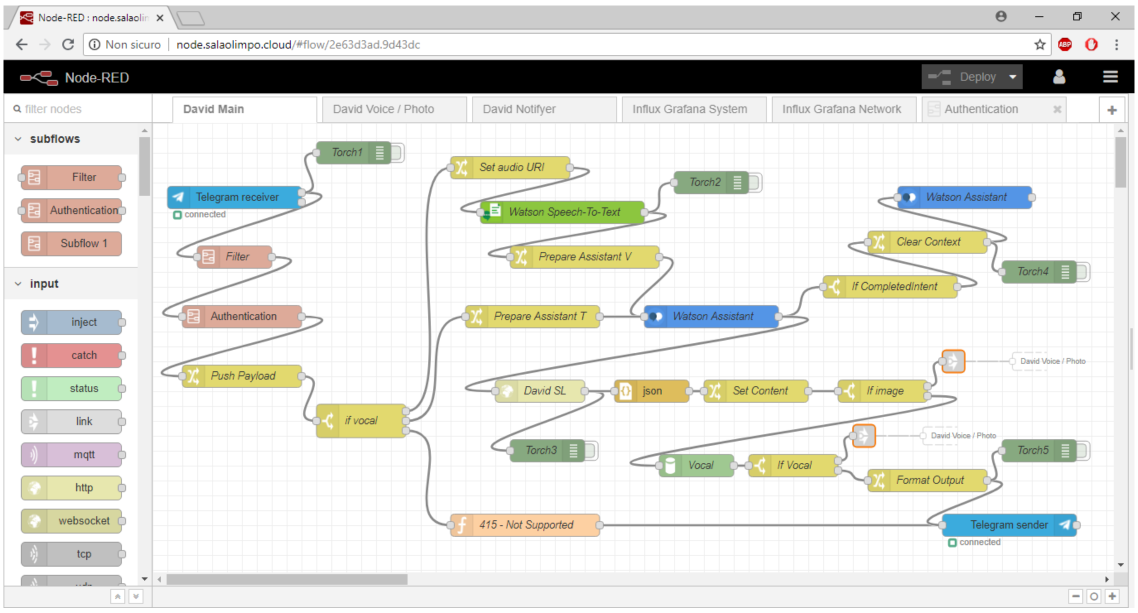


Figure 4.57: The David Main flow.

4.3.2. David Main Flow description

The base flow for this chapter is called *"David Main"*. It is in charge to perform the basic action of the David **chatbot**[1], i.e. to handle a user request.

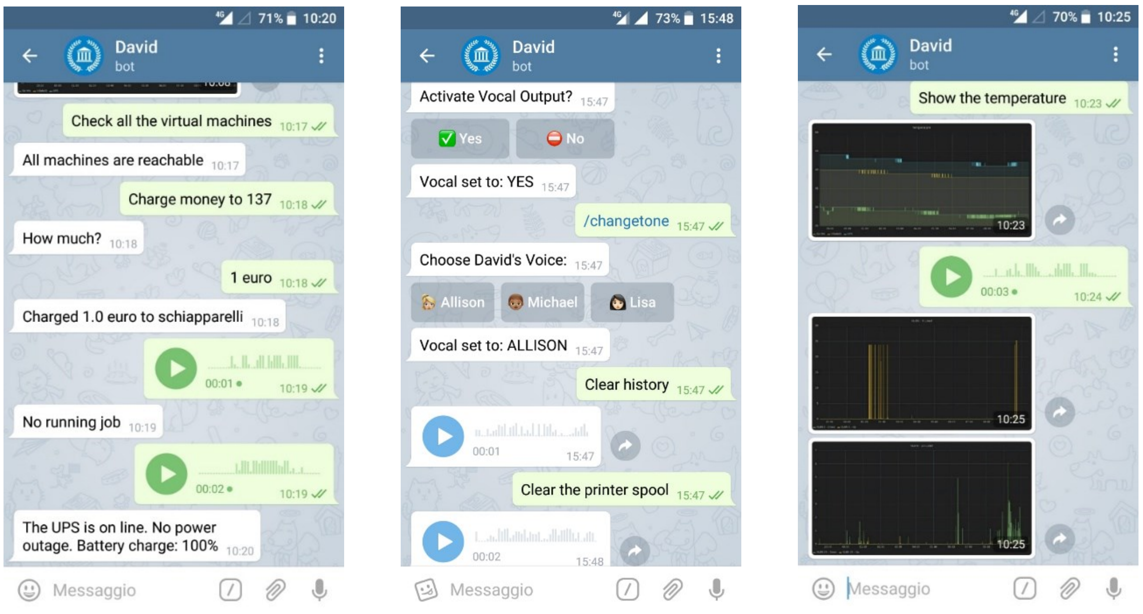
The main steps performed by this flow are shown in *Figure 4.12* and here reported:

1. The **Telegram Receiver** node is activated, receiving the message [79].
   It can be simply configured by inserting the HTTP Token of the bot, along with its name. This node creates the first msg packet, inserting in the payload the **chatId** (which represents the chat identifier, unique for a single user or group), the **type** (message, voice, etc.) and the **content**, representing the textual content of the message or an URI pointing the multimedia resource (like an image or an audio in *oga* format).
2. A **sub-flow** name **filter** is applied to exclude the messages representing commands. To do so, each message matching the regular expression *"\/.*"* are excluded.
   This operation is performed as command are managed by different flows.
3. A sub-flow named "*Authentication*" asserts that the user is authenticated using a simple password, which must be inserted only the first time a user starts a chat with the bot.
   This sub-flow will be described in the next chapter.
4. **A Switch** node checks the property *type* inside the payload, to determine if the message is textual, vocal or something else.
   a. If the type is **vocal**, [go-to 5].
   b. If the type is **message** (i.e. textual), [go-to 6].
   c. If the type is neither vocal or message, then a 415 error response is generated to inform the user that the content is not supported. Examples of unsupported content are images or videos.
      Then [go-to 11].
5. The audio URI, retrieved by means of the **weblink** attribute, is sent to **Watson Speech-To-Text** [80], using the proper node.
   This cloud service converts audio and voice into written text. The response is then prepared to be sent to Watson Assistant using the **Change** node to edit the content of msg.
   This step allows to interact with the bot by means of vocal notes, really useful in some situations. Finally, [go-to 7].
6. Using a change node, the textual message retrieved from the payload is prepared to be sent to Watson Assistant, so [go-to 7].

[1] *A **chatbot** (also known as a talkbot or chatterbot) is a computer program or an artificial intelligence which conducts a conversation via auditory or textual methods. Such programs are often designed to convincingly simulate how a human would behave as a conversational partner.*

7. The prepared text is sent to **Watson Assistant**, the same IBM service used by Alexa for intent and entity recognition (*see Chapter 4.2*).
   The response is sent in parallel to two steps, [go-to 8 & 9].
8. If an intent requiring *slots and entities* is concluded, the context is cleared using again the Assistant node to override the locally-saved context.
   Then [exit flow].
9. The result is sent as a standard POST request to **David's Service Layer** to perform the required action. This service layer has been already described in previous chapter (*see Chapter 4.1*), and will be discussed more in detail in the next chapters (*see Chapter 4.3*).
   The generated response is formatted as JSON with the proper node and another change node.
10. Two switch node are used to check if the output is a text, voice or photo.
    a. If the output is a photo (detected by the presence of one or more links in the David SL response) or it must be translated to voice (detected by a flag saved for each user), the packet is sent to the **David Voice/Photo flow** described later.
    b. Else it is formatted and sent to the final node [go-to 11].
11. The **Telegram Sender** node receives the msg packet, and sends the text contained in the payload to the user chat through the bot.
    As for the receiver, it is configured by inserting the HTTP Token of the bot.

The result is actually pretty good especially for textual messages, since Watson Assistant performs a better intent recognition in absence of noises introduced by the Speech-To-Text operation, needed in the Alexa system and for Telegram vocal messages. Example of chats using different requests and outputs are presented in the *Figure 4.13* below.


71% 10:20
David
bot
Check all the virtual machines 10:17
All machines are reachable 10:17
Charge money to 137 10:18
How much? 10:18
1 euro 10:18
Charged 1.0 euro to schiapparelli 10:18
00:01 10:19
No running job 10:19
00:02 10:19
The UPS is on line. No power outage. Battery charge: 100% 10:20
Messaggio

73% 15:48
David
bot
Activate Vocal Output? 15:47
Yes
No
Vocal set to: YES 15:47
/changetone 15:47
Choose David's Voice: 15:47
Allison
Michael
Lisa
Vocal set to: ALLISON 15:47
Clear history 15:47
00:01 15:47
Clear the printer spool 15:47
00:02 15:48
Messaggio

70% 10:25
David
bot
Show the temperature 10:23
10:23
00:03 10:24
10:25
10:25
Messaggio

Figure 4.58: Examples of chat using textual messages, voice notes and photos..

Despite the noise problem for audio notes, Watson demonstrated good accuracy for the Speech-To-Text service, and the possibility to send vocal messages to the bot is useful and often faster than typing the message.

In addition, the Telegram bot allows to communicate using **images**, used in this case to represent graphs extracted using the Grafana API to render the images. This is extremely useful and faster than browsing manually to the Grafana web interface, especially if only one or few graphs are required to check.

4.3.3 Authentication Sub-Flow description

The authentication sub-flow is actually pretty simple. Although Telegram provides its own authorization policies by means of tokens, it has been chosen a custom solution in order to simplify the authentication with a password to be introduced.

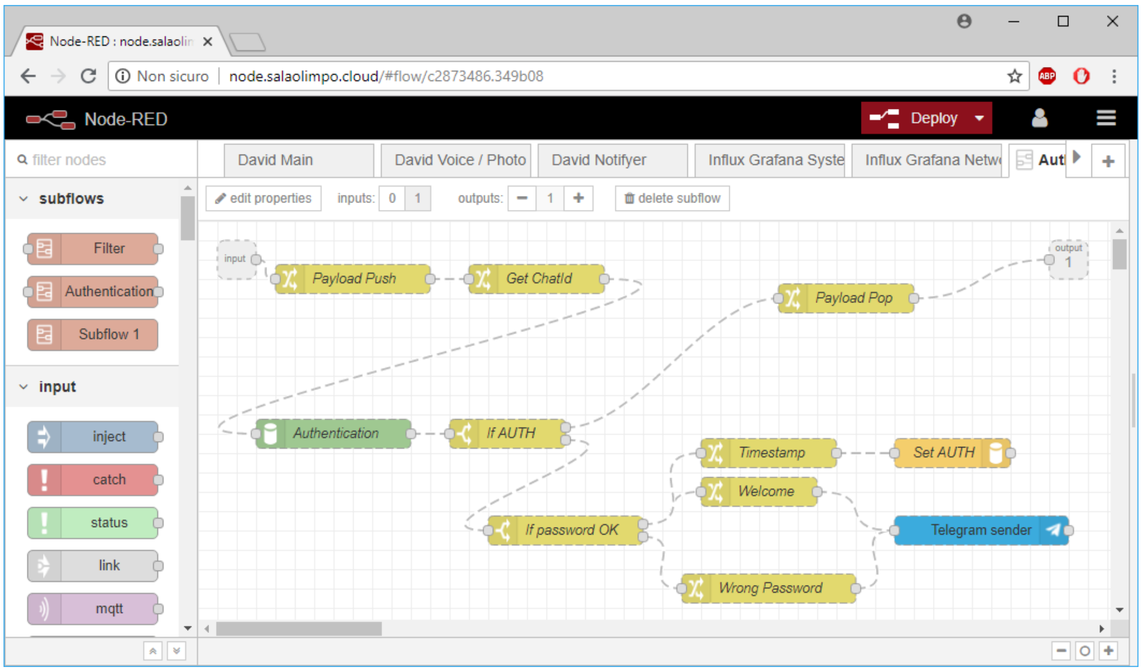


Figure 4.594: The Authentication flow.

The flow is reported in the *Figure 4.14*, and the steps are here reported:

1. The content of the payload is **saved** to another parameter, in order to be recovered at the end of authentication process and leave the payload free for the other operations of this sub-flow.
2. The chatId (as unique identifier for the chat) is used as a key in a **key-value store** to check if a user has been previously authenticated.
3. If yes, then the payload is recovered and sent to the main David Flow.
4. Else, the content of the message is checked for the password.
    a. If it is correct, then the key-value store is **updated** to set the user as authenticated. Then a response is built to inform the user.

    b. Else, a response is built to request the correct password to the user.
5. The response is sent using the Telegram sender node, and the payload is recovered to be correctly returned to David Main flow.

Note that if a user is already authenticated, then only the first two steps are performed, introducing only a small delay to the entire flow.

4.3.4 David Voice/Photo Flow description

David chatbot has also the possibility to respond using vocal notes. This option can be enabled with **commands**[1].

Commands currently implemented by this bot are:

- `/setvocal`, which enables the vocal output for requests.
- `/changetone`, which changes David voice tone.
- `/notifyme`, which sets a notification level (see *Chapter 4.3.5*).

To provide this functionality, the Node-RED palette for Telegram provides a specific node called **Command node** which can be set to handle a specific one.

When a command is sent, the flow proceeds as follow. It can be seen graphically on the top part of *Figure 4.15*.

1. The specific **Command node** activates and creates the first packet.
   This packet of course is filtered with the Authentication Sub-Flow, same as standard messages (*see Chapter 4.3.3*)
2. A confirm message is created using the **Function node** (performing a general JavaScript function).
   This confirm message is disposed to use the **Inline Keyboard**, which basically displays some options like "Yes" and "No".
3. The Telegram sender node sends the message.
4. After the user chooses the option, the **Callback Query node** is activated to handle that message, as pressing an option in the inline keyboard is not considered as a standard message.
5. A Switch node detects the correct command received (`/setvocal`, `/changetone`), and choses the correct path.
6. An update on the key-value store is performed to correctly set the user preference for the command.
7. Finally, a confirm message is sent with the Telegram sender, to inform the user about the correctness of the operation.

[1] ***Commands*** *are Telegram messages preceded by a slash, and each one represents a specific operation for the chatbot.*

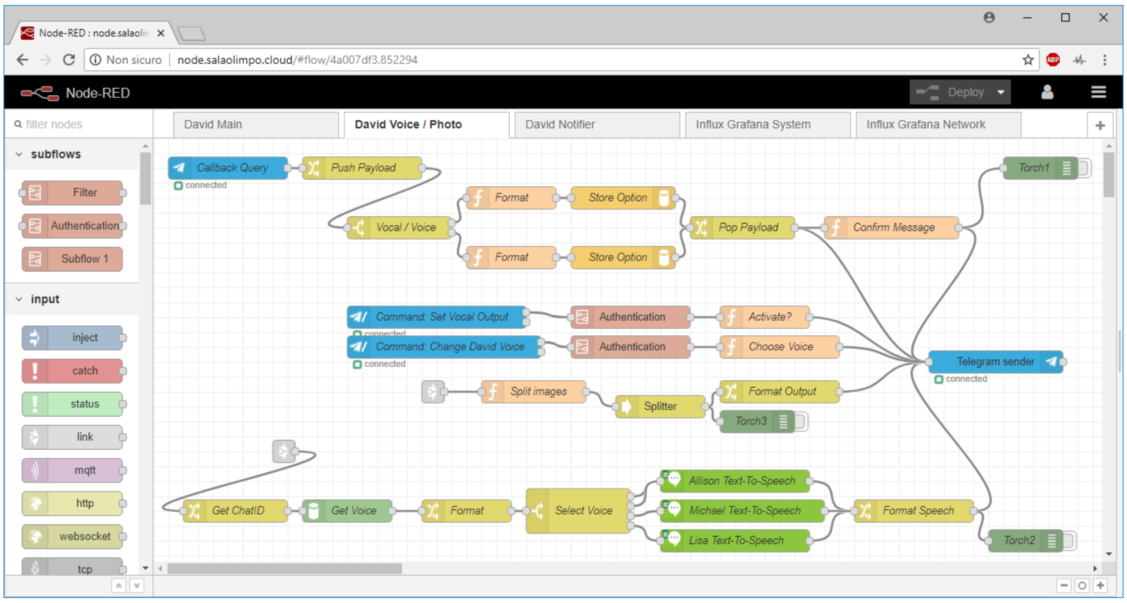


Figure 4.60: The David Voice/Photo flow.

This flow contains also the operation needed to output an image or a photo, based on the user preference already discussed or the necessity to display a graph generated with the Grafana API [81].

Referencing the bottom part of *Figure 4.15*, the steps performed are:

- For the voice type, the user preference set with the `/changetone` command is retrieved, to select the desired tone for vocal output.
  Later, the msg coming from the main flow is prepared using a change node to be passed to **Watson Text-To-Speech** [82], after selecting the correct parameters for the Watson call using the Switch node.
  This cloud service **converts written text into audio and voice**, and it is used to create vocal notes used by the bot.
  The response comes by means of an Ogg file (.oga) saved in the msg payload in binary form (resulting in a buffer type for JavaScript).
  The response is finally formatted for the output by setting the type *voice* and it is sent to the Telegram sender node.
- For the photo type, the msg comes from the main flow in form of a set of URL, each one representing a locally-saved photo retrieved using the Grafana API.
  The packet is thus split into a set of messages, each one sent to the Telegram sender node after a proper formatting, performed by setting the type of the message to *photo*.

The implementation of voice notes and images as a response is a very peculiar feature of this chatbot, and makes its utilization customizable, flexible and simple.

In addition, it is nice to use different voice tone based on a user preference.

4.3.5 David Notifier Flow description

David has also the possibility to send notification for different kind of events, like power outage and unwanted access to a part of the system.

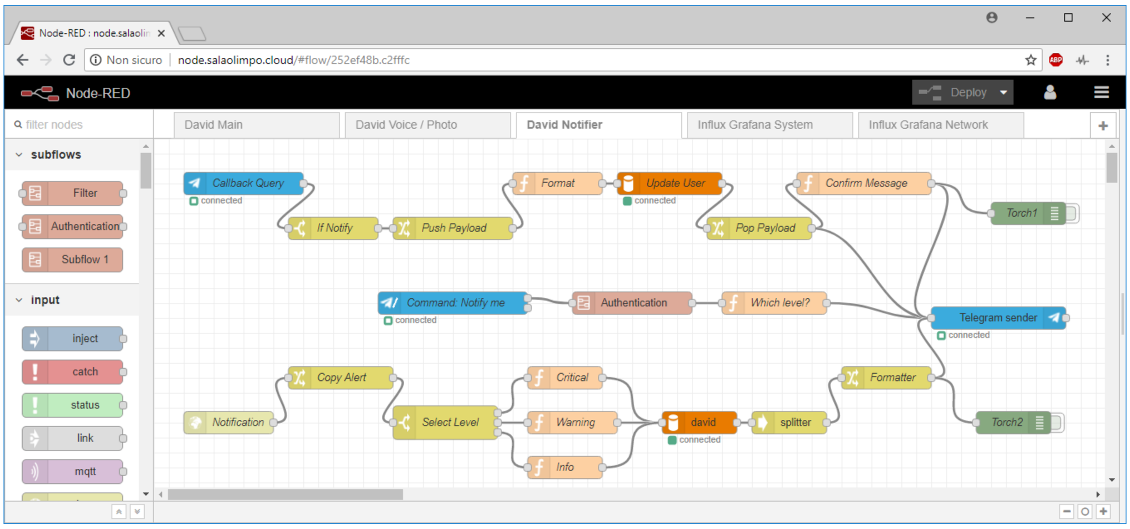


Figure 61: The David Notifier flow.

This option can be enabled with the command `/notifyme`, acting as described below and presented in *Figure 4.16*.

- For the command part, same as the voice commands, the packet is received, authenticated, and formatted to display an inline keyboard. Finally, it is sent to the Telegram sender node.
  Same as before (*see Chapter 4.3.4*), the Callback Query node is activated in response to the user choice. Finally, an update on the key-value store is performed and the confirmation message is sent back.
- For the actual notification part, an http POST method is exposed under a specific URL.
  An Alert is created using the Grafana specific function, which allows to specify a channel to send a notification when an **Alert Rule** activates. Some rules are presented in *Figure 4.17*.
  In particular, the channel configured is a **webhook** [83], which actually sends the notification via POST.
  Later on, a switch along with a key-value-store node selects all the users which have requested notification. Finally, the message is split for each user using a splitter node and sent to the Telegram sender node.

This notification service is really useful, and along with the possibility to retrieve graphical representation of data using the already described flow (*see Chapter 4.3.4*) it makes diagnose and troubleshooting easy and effective, especially since it needs only a smartphone to be brought along.

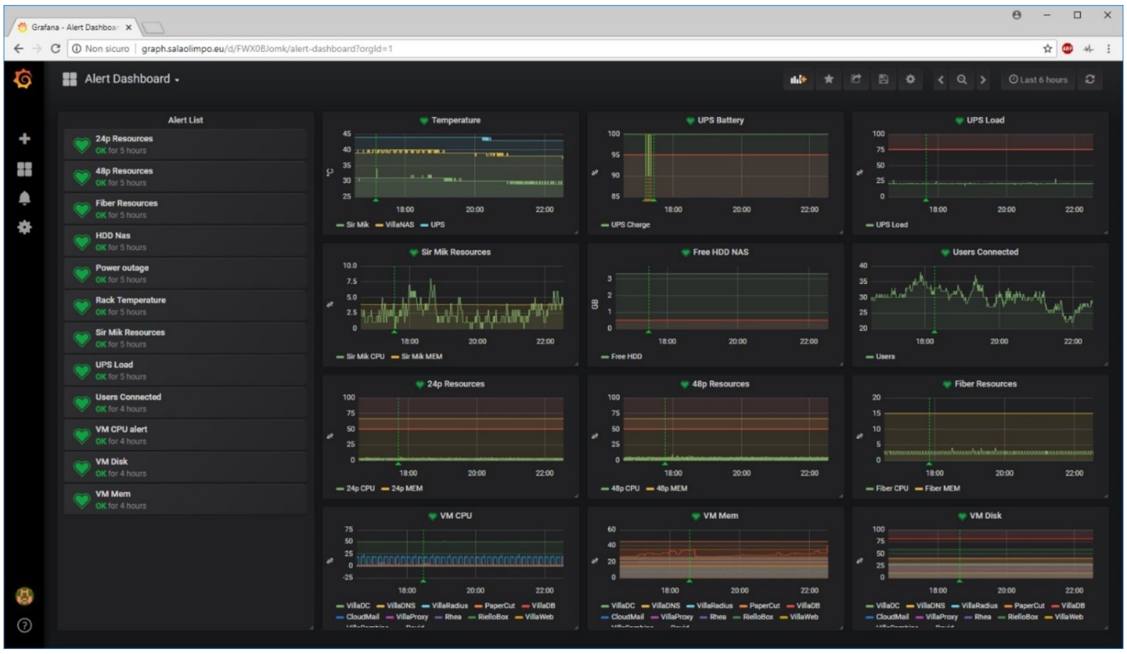


Figure 4.62: The Grafana dashboard for alert rules.

4.4 David Service Layer

4.4.1 Frameworks used

This section describes David Service Layer more in depth.

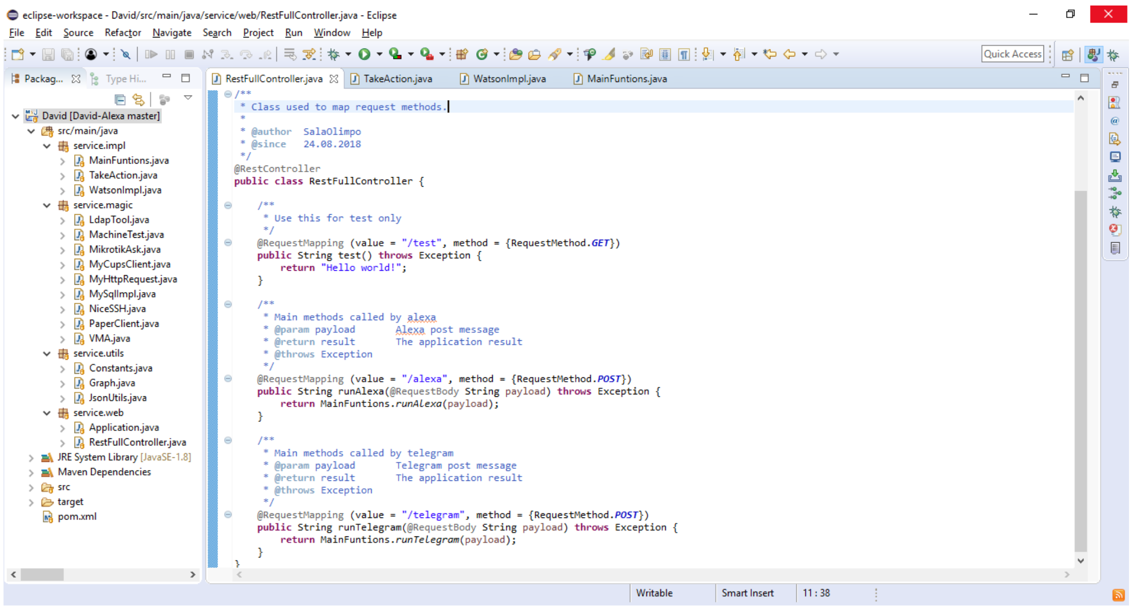
It is called by both Alexa and Telegram to perform the requested operation and, in the Alexa case, the intent and entity recognition. In the Telegram architecture, it is not needed since this step is performed by Node-RED.

As mentioned in the previous chapters, this service is developed using the **SpringBoot framework** and the **Maven build tool**.

All the endpoints are mapped in the **RestFulController** class of the `service.web` package, which simply calls the main static methods contained in the **MainFunctions** class of the `service.impl` package. It is shown in *Figure 4.18*.

While the `service.web` package contains all the classes strictly needed by SpringBoot, along with some initializations, the `service.impl` package contains all the classes implementing the general functions, like:

- ✓ **MainFunctions**, containing the main method called by Alexa and Telegram functions of the RestFulController class.
- ✓ **TakeAction**, which invokes the correct operation based on the intents and entities returned by Watson Assistant.
- ✓ **WatsonImpl**, containing the logic needed to fire the call to Assistant and generate the proper URL needed.



```java
/**
 * Class used to map request methods.
 *
 * @author  SalaOlimpo
 * @since   24.08.2018
 */
@RestController
public class RestFullController {

    /**
     * Use this for test only
     */
    @RequestMapping (value = "/test", method = {RequestMethod.GET})
    public String test() throws Exception {
        return "Hello world!";
    }

    /**
     * Main methods called by alexa
     * @param payload       Alexa post message
     * @return result       The application result
     * @throws Exception
     */
    @RequestMapping (value = "/alexa", method = {RequestMethod.POST})
    public String runAlexa(@RequestBody String payload) throws Exception {
        return MainFuntions.runAlexa(payload);
    }

    /**
     * Main methods called by telegram
     * @param payload       Telegram post message
     * @return result       The application result
     * @throws Exception
     */
    @RequestMapping (value = "/telegram", method = {RequestMethod.POST})
    public String runTelegram(@RequestBody String payload) throws Exception {
        return MainFuntions.runTelegram(payload);
    }
}
```

Figure 4.18: The RestFulController class and the project explorer screen.

Other packages are:

- ✓ The `service.magic`, containing classes for calling specific services. Examples of classes included in this package are reported in *Table 4.4*:

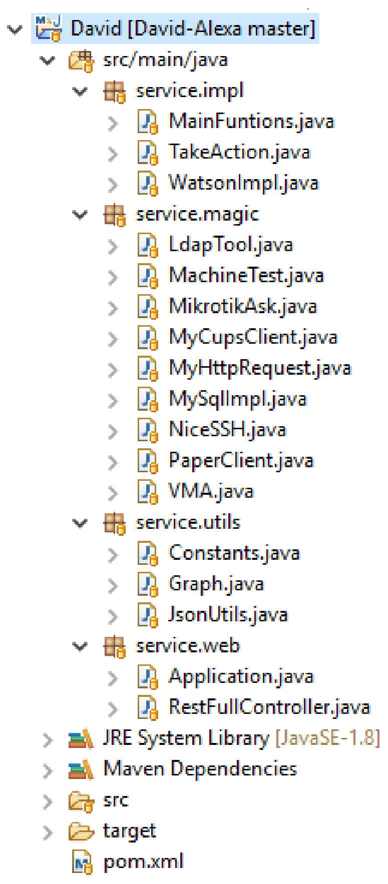
LdapTool, for LDAP operations.	*MyHttpRequest*, for internal APIs.
MachineTest, to check the VMs.	*MySqlImpl*, for MySQL operations.
MikrotikAsk, for MikroTik functions.	*NiceSSH*, for SSH connections.
MyCupsClient, for CUPS operations.	*VMA*, to use VMware API.

Table 4.4: Some classes of the service.magic package.

  This set of functions is not described in detail, being those classes standard code to execute standard operations.

- ✓ The `service.utils`, containing standard utilities such as:
  - **Constants**, containing all the constants of the Service Layer. This class is very useful since all the edit of API Tokens, username, password and such items can be performed just by editing this file.
  - **Graph**, representing the relationship between a graphs and IDs. Each instance represents a set of graph IDs related to a particular set of graphs, identified by an entity returned by Watson. As example, the "*ISP graph*" set (the entity returned by Watson) has two IDs, one for download and one for upload.
  - **JsonUtils**, used to create json-formatted objects for many services. Examples are the responses prepared for Alexa, or the creation of the body to be sent to Watson Assistant.

A screen of the packages used by the Service Layer, along with the content of those packages, is shown in *Figure 4.19*.



```
David [David-Alexa master]
  src/main/java
    service.impl
      MainFuntions.java
      TakeAction.java
      WatsonImpl.java
    service.magic
      LdapTool.java
      MachineTest.java
      MikrotikAsk.java
      MyCupsClient.java
      MyHttpRequest.java
      MySqlImpl.java
      NiceSSH.java
      PaperClient.java
      VMA.java
    service.utils
      Constants.java
      Graph.java
      JsonUtils.java
    service.web
      Application.java
      RestFullController.java
  JRE System Library [JavaSE-1.8]
  Maven Dependencies
  src
  target
  pom.xml
```

Figure 4.63: The project explorer screen.

The deploy on the Tomcat Server (*see Chapter 4.1*) is performed using the `tomcat7-maven-plugin`, which can be configured on the `pom.xml` file [84].

To start the deploy, a run configuration can be called using `tomcat7:deploy` (for the first release) or `tomcat7:redeploy` (for updates) as goal.

Any change of the code is committed, pushed and pulled from the **GitHub repository**, found as said in *https://github.com/SalaOlimpo/David-Alexa*.

4.4.2 Operations performed

In this section an example of execution of the Alexa service is proposed.

1. A POST request is sent to the Service Layer by Alexa, and it immediately calls the proper function in the MainFunctions class.
2. The session ID is retrieved from the packet, and used to retrieve the **Context** object from the previous conversation. This object is used by Watson as a state variable for visiting the dialogue graph.

3. If the request type is a **LauchRequest** (i.e. the first request opening David [85]), an empty message is generated for Watson using the JsonUtil class.
4. If the request type is an **IntentRequest** with **AMAZON.StopIntent**, then the context saved on the database is cleared and the response is built and returned.
   This basically means that the user has ended the dialogue with Alexa.
5. If the request type is an IntentRequest with **AMAZON.HelpIntent**, then the input text is set to "Help". Else, the text is extracted from the **EveryThingSlot**.
6. The **Watson Assistant call** is performed with a method of WatsonImpl class. The text is later extracted to be prepared for the output.
7. The context retrieved from Watson Assistant is saved on the database, and the response is built with a method of JsonUtil class.
8. If the response contains an **Intent**, it is checked using a series of conditional instruction in a method of the **TakeAction** class. Then the proper operation is performed using the correct class from the `service.magic` package.
9. If the response contains a **CompletedIntent** (i.e. an intent containing all the requires slots), it is checked using another method of the TakeAction class. Then, the context is cleared to delete the slots filled by Watson.
10. The response from TakeAction is used to build a response packet for Alexa. If an exception occurs, then the error message is returned in the same way.

The execution of the Telegram service is similar but simpler, since many functions are demanded to Node-RED, and Amazon standard Intents (i.e. LauchRequest, HelpIntent and StopIntent) are not present.

In particular:

1. A POST request is sent to the SL by Alexa, and it immediately calls the proper function in the MainFunctions class.
2. If it contains an **Intent**, it is checked for Intents and CompletedIntent using the **TakeAction** class, exactly like before.
3. The response from TakeAction is used to build a response packet for Telegram.

One important difference the possibility to send photo. In particular, if a graph is required, the result of the Grafana render API is saved locally to a file, using a standard **HttpUrlConnection** operation with **Basic Authentication**.

The URLs of those files are finally returned by the Service Layer to Node-RED, in order to be formatted for the output (*see Chapter 4.3.4*).

To conclude this chapter, an example of method called (the main method of the Alexa flow) is reported in *CodeView 4.1.*

```java
/**
 * Method called for the Alexa service
 *
 */
public static String runAlexa(String payload) throws Exception {
          JSONObject          args                = new JSONObject(payload);
          JSONObject          watsonBody          = null;
          String              requestText;

          //=== Retrieving session ID ===
          String sessID       = args.getJSONObject("session").getString("sessionId");

          //=== If LaunchRequest, it doesn't contain intents ===
          if (args.getJSONObject("request").getString("type").equals("LaunchRequest")) {
                    watsonBody = buildWatsonData("");

                    //--- Fire the Watson call for this specific case ---
                    JSONObject watsonResponse = getWatsonResponse(watsonBody);
                    String      textResponse  = watsonResponse.getJSONObject("output")
                                                        .getJSONArray("text").getString(0);

                    LINK.setContext(sessID, watsonResponse.getJSONObject("context").toString(), General.ALEXA);
                    return buildAlexaResponse(textResponse, false) .toString();
          }

          //=== If StopIntent, close david ===
          if (args.getJSONObject("request").getJSONObject("intent").getString("name").equals("AMAZON.StopIntent")) {
                    LINK.delAlexaContext();
                    return buildAlexaResponse("Closing david", true).toString();
          }

          //=== If HelpIntent, handle with Assistant ===
          if (args.getJSONObject("request").getJSONObject("intent").getString("name").equals("AMAZON.HelpIntent")) {
                    requestText = "Help";
          } else {
                    requestText = args.getJSONObject("request").getJSONObject("intent")

          .getJSONObject("slots").getJSONObject("EveryThingSlot").getString("value");
          }

//-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-

          //=== Retrieving the contect and logging the query ===
          String watsonID     = LINK.getContext(sessID, General.ALEXA);
          LINK.logQuery(requestText, General.ALEXA);

          //=== Building the body for Watson call ===
          if(watsonID == null)
                    watsonBody = buildWatsonData(requestText);
          else
                    watsonBody = buildWatsonData(requestText, watsonID);

          //=== Fire the Watson call ===
          JSONObject          watsResponse = getWatsonResponse(watsonBody);
          String              textResponse;

          if(watsonResponse.getJSONObject("output").getJSONArray("text").length() > 0)
                    textResponse = watsonResponse.getJSONObject("output").getJSONArray("text").getString(0);
          else
                    textResponse = "No response for this action";
          LINK.setContext(sessID, watsonResponse.getJSONObject("context").toString(), General.ALEXA);

          //=== Execute the relative action ===
          try {
                    String actionResponse = null;
                    if(watsResponse.getJSONObject("output").has("completedIntent")) {
                              actionResponse = TakeAction.pickCompleted(
                                        watsResponse.getJSONObject("output").getString("completedIntent"),
                                        watsResponse.getJSONObject("context")
                              );
                              if(actionResponse != null) textResponse = actionResponse;
                    } else if(watsResponse.getJSONArray("intents").length() > 0) {
                              actionResponse = TakeAction.pickUp (
                                        watsResponse.getJSONArray("intents").getJSONObject(0).getString("intent"),
                                        watsResponse.getJSONObject("context")
                              );
                              if(actionResponse != null) textResponse = actionResponse;

          } catch (Exception ex) {
                    return buildAlexaResponse(ex.getMessage(), true) .toString();
          }

          return buildAlexaResponse(textResponse, false) .toString();
}
```

CodeView 4.11: The runAlexa method of the MainFunctions class.

Chapter 5: Closing notes

5.1 Conclusions

During this project realization, a lot of issues were encountered. In particular, the absence of any guide or reference manual made the implementation of this solution extremely complex.

However, the result is outstanding as the system is **completely customized** to perfectly fit the use case and the functional requirements of the project.

In addition, all the faced issues led to an incredible **personal growth** for all the components of the development team, especially since this project covers the majority of the topic discussed during university courses. This made the realization of this architecture a perfect chance to practice all the received teachings and master the concepts through practical application.

Besides working, the solution also grants an excellent user experience, since the system is complete and fast. Many positive feedback comments from the end users demonstrate that this implementation was a success.

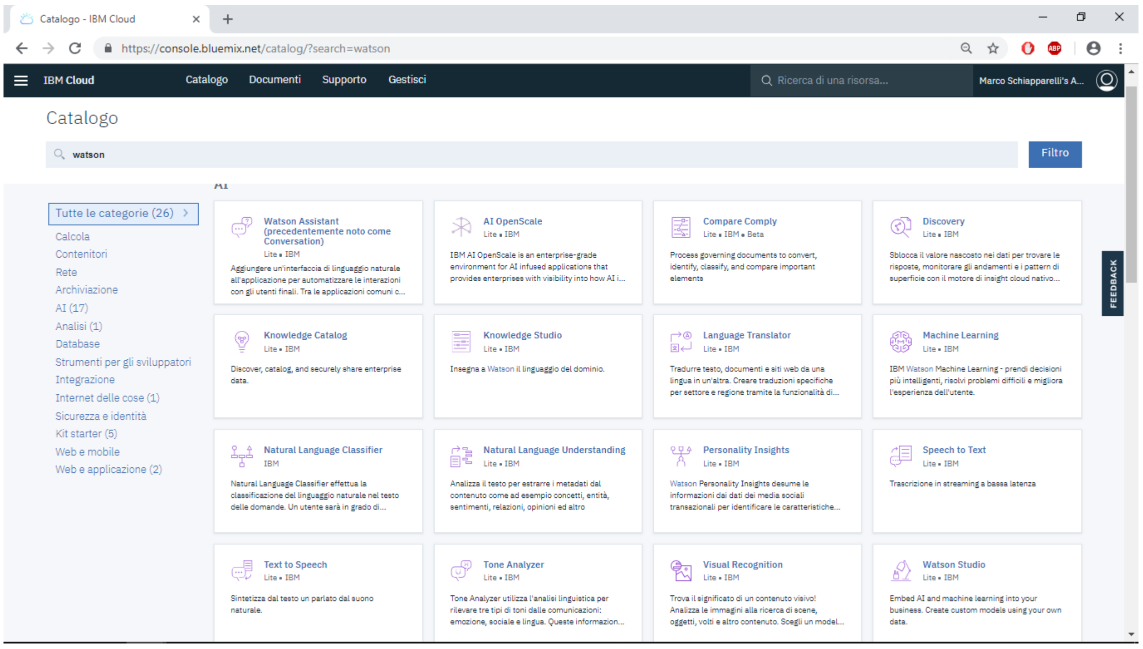
The introduction of David was also a further step, which wasn't necessary for the system to work, but it is actually a useful service created in order to test several **new technologies** like **IBM Watson**, **Amazon Alexa** and the **Telegram bot** engine, in a really custom fashion.

The result far exceeded the expectations, as it perfectly integrates with the entire architecture to provide all of its features. As already mentioned, despite being fully functional, this can be considered only a prototype, since there is much room to keep evolving this platform by adding communication channels, possible operations and lots of functionalities to increase the coverage of the product.

In fact, the entire project keeps evolving every day, for example:

- The network infrastructure, along with the configuration, is evolving to provide even better service, by upgrading the hardware and updating the configurations. As example, in the nearly future the entire system will be fibre-based and using only 5.0 GHz access point.
- The server contains a lot of newly-created machine in TEST and ACC environments, which are still under development to provide more services to the system and/or to the users. Some examples are the Windows Update Server, the local repository for Linux updates, an Asterisk machine for creating a VoIP infrastructure, a document server and many other. Even more servers are currently on analysis to be later developed.

➢ David keeps receiving daily updates, in order to provide better access to the platform and add more unique features.
An example is the addition of the **Locality** feature of the Amazon Skill Kit, which allows to set different languages for the service. In fact, David is currently under development in order to add the Italian language.
In addition, Watson Assistant can always be improved by adding new examples, removing conflicting ones, and in general better refine old or new intents to keep improving the overall service.
Maybe other services from IBM Cloud will be added in the future, to improve the **cognitive aspect** of this project. The Watson Suite is evolving fast, and new services **are created and made accessible every day**. An example of the services from the Watson Suite, retrieved from the IBM Content Catalog, is shown in *Figure 5.1*.



Watson Assistant (precedentemente noto come Conversation)
Lite • IBM
Aggiungere un'interfaccia di linguaggio naturale all'applicazione per automatizzare le interazioni con gli utenti finali. Tra le applicazioni comuni c...

AI OpenScale
Lite • IBM
IBM AI OpenScale is an enterprise-grade environment for AI infused applications that provides enterprises with visibility into how AI i...

Compare Comply
Lite • IBM • Beta
Process governing documents to convert, identify, classify, and compare important elements

Discovery
Lite • IBM
Sblocca il valore nascosto nei dati per trovare le risposte, monitorare gli andamenti e i pattern di superficie con il motore di insight cloud nativo...

Knowledge Catalog
Lite • IBM
Discover, catalog, and securely share enterprise data.

Knowledge Studio
Lite • IBM
Insegna a Watson il linguaggio del dominio.

Language Translator
Lite • IBM
Tradurre testo, documenti e siti web da una lingua in un'altra. Creare traduzioni specifiche per settore e regione tramite la funzionalità di...

Machine Learning
Lite • IBM
IBM Watson Machine Learning - prendi decisioni più intelligenti, risolvi problemi difficili e migliora l'esperienza dell'utente.

Natural Language Classifier
IBM
Natural Language Classifier effettua la classificazione del linguaggio naturale nel testo delle domande. Un utente sarà in grado di...

Natural Language Understanding
Lite • IBM
Analizza il testo per estrarre i metadati dal contenuto come ad esempio concetti, entità, sentimenti, relazioni, opinioni ed altro

Personality Insights
Lite • IBM
Watson Personality Insights desume le informazioni dai dati dei media sociali transazionali per identificare le caratteristiche...

Speech to Text
Lite • IBM
Trascrizione in streaming a bassa latenza

Text to Speech
Lite • IBM
Sintetizza dal testo un parlato dal suono naturale.

Tone Analyzer
Lite • IBM
Tone Analyzer utilizza l'analisi linguistica per rilevare tre tipi di toni dalle comunicazioni: emozione, sociale e lingua. Queste informazion...

Visual Recognition
Lite • IBM
Trova il significato di un contenuto visivo! Analizza le immagini alla ricerca di scene, oggetti, volti e altro contenuto. Scegli un model...

Watson Studio
Lite • IBM
Embed AI and machine learning into your business. Create custom models using your own data.

Figure 5.1: The IBM Content Catalog, showing some services from the Watson Suite.

It surely was a challenging task, but the entire team is really satisfied with the outcome and glad for this unique opportunity to freely experiment a large set of software, products and devices while creating this beloved infrastructure.

5.2 Acknowledgements

Firstly, I would like to say thanks to my dearly beloved family, who gave me the opportunity to pursue this university course, and sustained me every single day. This entire thesis, along with the possibility to attend the Computer Engineering course while living in Villa San Giuseppe, would not have been possible without their sacrifices and their support.

Secondly, I would like to thanks all the people involved in the project, especially Marco, dear friend and irreplaceable "programming partner" who invested a lot of effort for this project. Others special thanks are for Alessandro, who helped the design and the configuration of the entire system with his keen eye and his outstanding capabilities; and for Federico, precise and meticulous manager, especially for the economic part. Of course, there are many other people who invested time in this project, and I would like to thank them all.

Other thanks are for my colleagues of Blue Reply, who gave me the opportunity to test and learn about many special services, in particular from IBM Watson.

Last but not least, the final thanks are for all my friend in Villa San Giuseppe, who shared with me the major part of these years. As irreplaceable and trustworthy friends, they contributed a lot to my personal growth and gave me really special and important memories of these years.

A particular, final thanks is for my closest friends from the "Caffè Schiapparelli". We may take different paths, but we never forget each other's. Love you guys.

5.3 Bibliography

5.3.1 Network Part

[1] MikroTik, “MikroTik Home Page” [Online]. Available: https://mikrotik.com/.

[2] MikroTik, “MikroTik and RouterOS Documentation” [Online]. Available: https://www.mikrotik.com/documentation.

[3] ZyXEL, “ZyXEL Home Page” [Online]. Available: https://www.zyxel.com.

[4] ZyXEL, “ZyXEL 1900 Documentation” [Online]. Available: https://prodotti.zyxel.it/USERSGUIDE/ZYXGS-1900-SERIE.pdf.

[5] MikroTik, “MikroTik Winbox Documentation” [Online]. Available: https://wiki.mikrotik.com/wiki/Manual:Winbox.

[6] MikroTik, “Mikrotik CAPsMAN Documentation” [Online]. Available: https://wiki.mikrotik.com/wiki/Manual:CAPsMAN.

[7] Riello_UPS, “Riello UPS Home Page” [Online]. Available: https://www.riello-ups.it/.

[8] Riello_UPS, “Riello UPS Vision - Energyshare” [Online]. Available: https://www.riello-ups.it/products/1-gruppi-di-continuita/64-vision-dual.

[9] QNAP, “QNAP Home Page” [Online]. Available: https://www.qnap.com/it-it/.

[10] QNAP, “QNAP myQNAPCloud Documentation” [Online]. Available: http://docs.qnap.com/nas/4.1/Home/en/index.html?myqnapcloud_service.htm.

[11] Wikipedia, “Wikipedia image - Raid 1+0” [Online]. Available: https://upload.wikimedia.org/wikipedia/commons/a/ad/RAID_01.svg.

[12] MikroTik, “MikroTik Mangle Manual” [Online]. Available: https://wiki.mikrotik.com/wiki/Manual:IP/Firewall/Mangle.

[13] MikroTik, “MikroTik Firewall Filter” [Online]. Available: https://wiki.mikrotik.com/wiki/Manual:IP/Firewall/Filter.

5.3.2 Server Farm Part

[14] VMware, “VMware vSphere 5.1 Documentation” [Online]. Available: https://pubs.vmware.com/vsphere-51/index.jsp.

[15] Microsoft, “Microsoft Docs - Active Directory Domain Services” [Online]. Available: https://docs.microsoft.com/en-us/windows-server/identity/ad-ds/active-directory-domain-services.

[16] Microsoft, “Microsoft Docs - LDAP AD Schema” [Online]. Available: https://docs.microsoft.com/en-us/windows/desktop/adschema/active-directory-schema.

[17] Microsoft, “Microsoft Docs - Server Manager Documentation” [Online]. Available: https://docs.microsoft.com/en-us/windows-server/administration/server-manager/server-manager.

[18] Microsoft, “Microsoft Docs - Certificate Services Documentation” [Online]. Available: https://docs.microsoft.com/en-us/windows/desktop/seccrypto/certificate-services.

[19] Microsoft, “Microsoft Docs - Folder Redirection” [Online]. Available: https://docs.microsoft.com/it-it/windows-server/storage/folder-redirection/deploy-folder-redirection.

[20] ISC, “ISC - Bind9 Reference Manual” [Online]. Available: http://ftp.isc.org/isc/bind9/9.10.8-P1/doc/arm/Bv9ARM.pdf.

[21] Eduroam, “Eduroam WPA2-E and Image” [Online]. Available: https://www.eduroam.us/node/10.

[22] FreeRADIUS, “FreeRADIUS Docuementation” [Online]. Available: https://freeradius.org/documentation/.

[23] A. Inc., “Apple Inc. - CUPS Page” [Online]. Available: https://www.cups.org/.

[24] PyKota, “PyKota Tea4CUPS Home Page” [Online]. Available: http://www.pykota.com/software/tea4cups.

[25] PyKota, “PyKota pkpgcounter Home Page” [Online]. Available: http://www.pykota.com/software/pkpgcounter.

[26] Oracle, “Oracle MySQL Home Page” [Online]. Available: https://www.mysql.com/.

[27] InfluxData, “InfluxData Home Page” [Online]. Available: https://www.influxdata.com/.

[28] phpMyAdmin, “phpMyAdmin Home Page” [Online]. Available: https://www.phpmyadmin.net/.

[29] InfluxData, “InfluxData InfluxDB Documentation” [Online]. Available: https://docs.influxdata.com/influxdb/v1.7/.

[30] NextCloud, “NextCloud Home Page” [Online]. Available: https://nextcloud.com/.

[31] Postfix, “Postfix Home Page” [Online]. Available: http://www.postfix.org/.

[32] Dovecot, “Dovecot Home Page” [Online]. Available: https://www.dovecot.org/.

[33] R. Mail, “Roundcube Mail Home Page” [Online]. Available: https://roundcube.net/.

[34] NextCloud, “NextCloud Documentation” [Online]. Available: https://docs.nextcloud.com/.

[35] NextCloud, “NextCloud App Store” [Online]. Available: https://apps.nextcloud.com/.

[36] Postfix, “Postfix Basic Configuration” [Online]. Available: http://www.postfix.org/BASIC_CONFIGURATION_README.html.

[37] Postfix, “Postfix SASL Documentation” [Online]. Available: http://www.postfix.org/SASL_README.html.

[38] Dovecot, “Dovecot Documentation Wiki” [Online]. Available: https://wiki2.dovecot.org/.

[39] RoundCube, “RoundCube Options” [Online]. Available: https://github.com/roundcube/roundcubemail/wiki/Configuration.

[40] NGINX, “NGINX Home Page” [Online]. Available: https://www.nginx.com/.

[41] L. Encrypt, “Certbot Home Page” [Online]. Available: https://certbot.eff.org/.

[42] L. Encrypt, “Let’s Encrypt Home Page” [Online]. Available: https://letsencrypt.org/.

[43] NGINX, “NGINX Documentation” [Online]. Available: https://nginx.org/en/docs/.

[44] Certbot, “Certbot User Guide” [Online]. Available: https://certbot.eff.org/docs/using.html.

[45] L. Encrypt, “Let's Encrypt - How it Works” [Online]. Available: https://letsencrypt.org/how-it-works/.

[46] Grafana, “Grafana Home Page” [Online]. Available: https://grafana.com/.

[47] Bootstrap, “Bootstrap SB Admin 2 Page” [Online]. Available: https://startbootstrap.com/template-overviews/sb-admin-2/.

[48] Grafana, “Grafana Documentation” [Online]. Available: http://docs.grafana.org/.

[49] Grafana, “Grafana Plugins” [Online]. Available: https://grafana.com/plugins.

[50] InfluxData, “InfluxData Telegraf Agent” [Online]. Available: https://www.influxdata.com/time-series-platform/telegraf/.

[51] upsmon, “upsmon Home Page” [Online]. Available: http://www.ups-technet.com/upsmon.htm.

[52] N. U. Tools, “Network UPS Tools Home Page” [Online]. Available: https://networkupstools.org/.

[53] TeamSpeak, “TeamSpeak Home Page” [Online]. Available: https://www.teamspeak.com/en/.

[54] WebCollab, “WebCollab Home Page” [Online]. Available: https://webcollab.sourceforge.io/.

[55] MediaWiki, “MediaWiki Home Page” [Online]. Available: https://www.mediawiki.org/wiki/MediaWiki.

[56] MediaWiki, “MediaWiki LDAP Authentication Plugin” [Online]. Available: https://www.mediawiki.org/wiki/Extension:LDAP_Authentication.

[57] Samba, “Samba Home Page” [Online]. Available: https://www.samba.org/.

[58] Samba, “Samba Documentation Page” [Online]. Available: https://www.samba.org/samba/docs/.

5.3.3 David Part

[59] IBM, “IBM Watson Home Page” [Online]. Available: https://www.ibm.com/watson/.

[60] IBM, “IBM Cloud Page” [Online]. Available: https://www.ibm.com/cloud/.

[61] Amazon, “Amazon Echo Dot 3” [Online]. Available: https://www.amazon.com/dp/B0792KTHKJ/ref=fs_ods_aucc_dt.

[62] Amazon, “Amazon Alexa Skill Kit” [Online]. Available: https://www.amazon.com/alexa-skills/b?ie=UTF8&node=13727921011.

[63] Amazon, “Amazon Alexa Skill Kit” [Online]. Available: https://developer.amazon.com/it/alexa-skills-kit.

[64] Amazon, “Amazon - Custom Skill as Web Service” [Online]. Available: https://developer.amazon.com/it/docs/custom-skills/host-a-custom-skill-as-a-web-service.html.

[65] Amazon, “Amazon - Custom Skill as AWS Lambda Function” [Online]. Available: https://developer.amazon.com/it/docs/custom-skills/host-a-custom-skill-as-an-aws-lambda-function.html.

[66] IBM, “IBM Bluemix - Intent&Entities Documentation” [Online]. Available: https://console.bluemix.net/docs/services/conversation/intents-entities.html.

[67] IBM, “IBM Watson Assistant” [Online]. Available: https://www.ibm.com/watson/ai-assistant/.

[68] IBM, “IBM Watson Assistant - Getting Started” [Online]. Available: https://console.bluemix.net/docs/services/assistant/getting-started.html.

[69] IBM, “IBM Watson Assistant - API Reference” [Online]. Available: https://console.bluemix.net/apidocs/assistant.

[70] Apache, “Apache Tomcat Home Page” [Online]. Available: http://tomcat.apache.org/.

[71] Spring, “Spring Boot Home Page” [Online]. Available: http://spring.io/projects/spring-boot.

[72] Apache, “Apache Maven Home Page” [Online]. Available: https://maven.apache.org/.

[73] Apache, “Apache Tomcat Documentation” [Online]. Available: http://tomcat.apache.org/tomcat-8.0-doc/.

[74] SalaOlimpo, “SalaOlimpo - David SL Repository” [Online]. Available: https://github.com/SalaOlimpo/David-Alexa.

[75] Telegram, “Telegram Bot Reference” [Online]. Available: https://core.telegram.org/bots.

[76] Telegram, “Telegram BotFather Page” [Online]. Available: https://telegram.me/BotFather.

[77] Node-RED, “Node-RED Home Page” [Online]. Available: https://nodered.org/.

[78] Node-RED, “Node-RED Adding Nodes” [Online]. Available: https://nodered.org/docs/getting-started/adding-nodes.

[79] Node-RED, “Node-RED contrib-telegrambot Documentation” [Online]. Available: https://flows.nodered.org/node/node-red-contrib-telegrambot.

[80] IBM, “IBM Watson Speech-To-Text” [Online]. Available: https://www.ibm.com/watson/services/speech-to-text/.

[81] Grafana, “Grafana API Reference” [Online]. Available: http://docs.grafana.org/http_api/.

[82] IBM, “Watson Text-To-Speech” [Online]. Available: https://www.ibm.com/watson/services/text-to-speech/.

[83] Grafana, “Grafana Notification Documentation” [Online]. Available: http://docs.grafana.org/alerting/notifications/.

[84] Apache, “Tomcat 7 Maven Plugin” [Online]. Available: http://tomcat.apache.org/maven-plugin-trunk/tomcat7-maven-plugin/plugin-info.html.

[85] Amazon, “Amazon Request Types Reference” [Online]. Available: https://developer.amazon.com/it/docs/custom-skills/request-types-reference.html.

Turin, 1 December 2018