Fotovoltaic Excess Management and Visualization System

Bachelor Thesis
Information Technology Specialization

Pablo Esteban Baquero

Director: Luis Velasco Esteban
(Computer architecture department)

27th June, 2022

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

Contents

1 Context and scope

1.1 Context

This first section aims to provide a general overview of the project as a whole. First, the context of the project will be laid out. Then, some key concepts will be defined, followed by a short statement of the problem to solve. Finally, a brief description of the stakeholders will be given.

1.1.1 Project context

The present document describes the work done for a Bachelor Thesis in Informatics Engineering in the Facultat d'Informàtica de Barcelona (FIB), faculty of the Universitat Politècnica de Catalunya (UPC). More precisely, it can be framed as part of the Information Technology specialization curriculum, within the FIB's Informatics Engineering degree.

The project is done within the framework of the university (A modality), although with the collaboration of Ontec Energy, and is directed by Luis Velasco Esteban.

1.1.2 Concepts

Internet of Things (IoT)

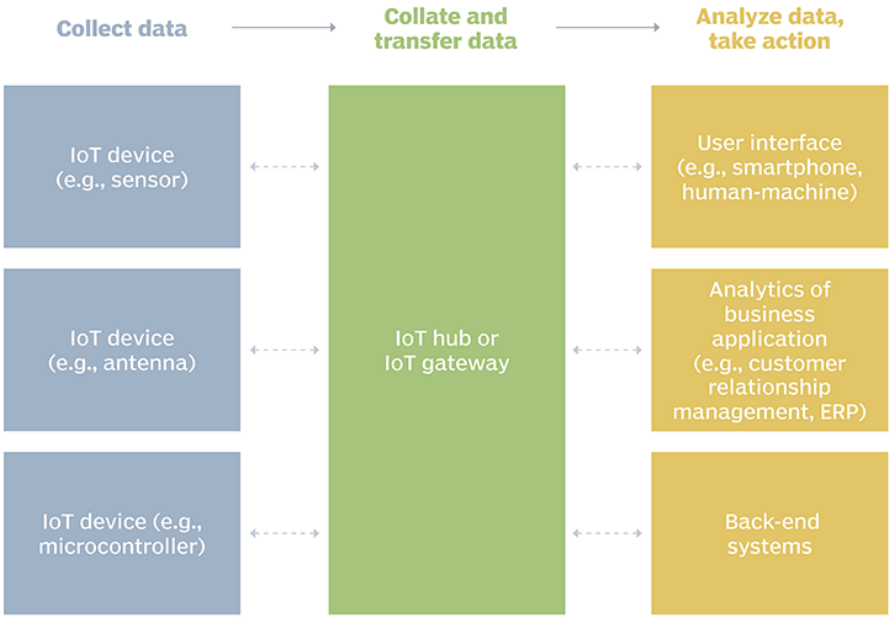


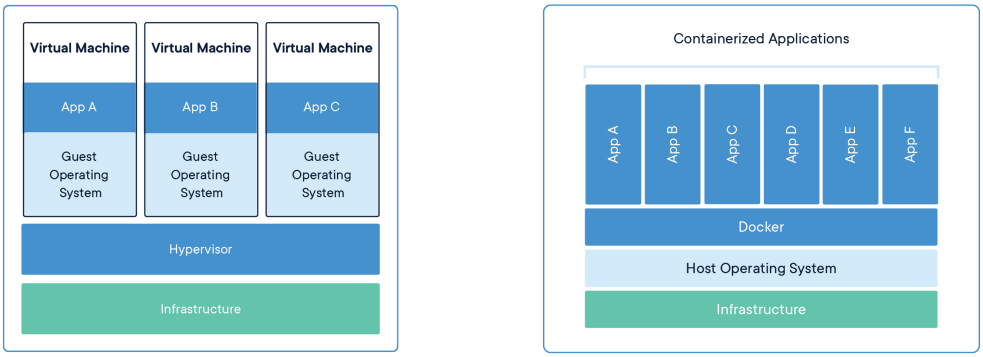
Figure 1: Example of an IoT system [71]

The Internet of Things is a rather broad term used to describe the growing network of devices able to communicate over the Internet or other communications networks without the need for human interaction [158]. Many of these devices are embedded

with sensors processing ability or software, but their use cases range from built-in sensors in vehicles to heart monitor implants [71].

Domotics
Also known as home automation, domotics can be summarized as the subset of Internet of Things found in smart homes. These automation systems are able to monitor and control a wide variety of home attributes, such as lighting or temperature, as well as taking care of security or task automation [30].

Containerization
A container is a standard unit of software used to run applications quickly and reliably from one computing environment to another [63]. Although similar to virtual machines at a first glance, they serve different purposes: containers are designed to be lightweight and easily replaceable, while virtual machines are mainly used to provide isolation and hardware virtualization.



(a) Virtualized operating systems (b) Containerized applications

Figure 2: Comparison between containers and virtual machines [63]

1.1.3 Problem to solve

This project's main objective, as may be inferred from its title, is the development of a working prototype for a photovoltaic excess management and visualization system based on the integration of already existing technologies. In doing so it aims to solve a practical requirement for the stakeholders.

This description might be a bit vague, but it will be expanded upon in further sections and should give the reader a general idea for the time being.

1.1.4 Stakeholders

The main stakeholders for this project are twofold: one being the director for this thesis, Luis Velasco Esteban, researcher at the FIB's Computer Architecture department, and the other one being Ontec Energy as a direct collaborator. The first one will benefit from the research done during the project, and the second one from the developed final product.

Aside from the main stakeholders, the results of the project will be of interest to some other groups of people as well. These will mainly be smart home owners, domotic companies or researchers sharing similar requirements.

1.2 Justification

Whenever a project deals with a software system a decision must be made regarding how it will be realized: whether it will be developed from scratch, modify an existing solution or some other alternative. This section will detail some of the explored existing solution and technologies and justify the final decision made.

1.2.1 Existing solutions

Rather than reinventing the whole wheel, many times what a project needs to achieve its goal is to find an already existing solution that fits its requirements and configure or adapt it to its particular needs.

Home assistant [149] could be one such solution for this project. It is a free and open-source software for home automation designed to be a smart home's main control system. Many IoT devices and software are supported as additional modules, which makes it highly customizable.

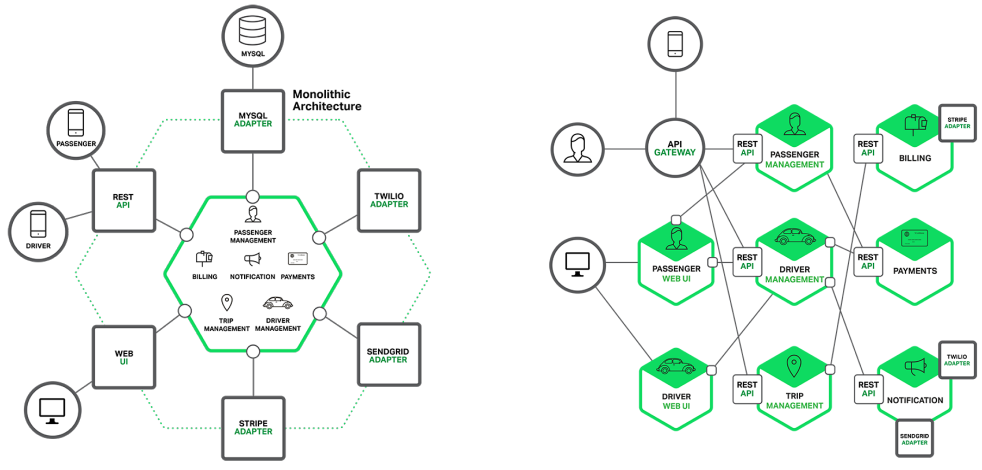
1.2.2 Existing technologies

It can also be the case that no perfect solution exists for the problem at hand, so that the best solution is to combine independent technologies to achieve the desired functionality. As we shall see later on, there are many perfectly useful technologies out there that can be leveraged to build a solution to our problem.

1.2.3 Conclusion

In order to not be tied to a particular stack and some other reasons, the design decision made for this project is to build the system by integrating existing technologies.

To implement this we'll make use of a microservice architecture, a way of designing applications as suites of independently deployable services which has gained traction in recent years and can bee seen in opposition to a monolithic architecture [107, 147].



(a) Monolithic architecture

(b) Microservice architecture

Figure 3: Comparison between the two architectures [138]

1.3 Scope

So far the project has been described in a general manner only. In order to narrow down its scope some specifics need to be addressed.

First of all, the main objectives of the project will be described together with more fine-grained sub-objectives. Then, requirements, both functional and non-functional, will follow. Finally, obstacles and risks will be considered.

1.3.1 Objectives

This project's main objective, as may be inferred from its title, is the development of a working prototype for a photovoltaic excess management and visualization system based on the integration of already existing technologies.

In more concrete terms, this means the implementation of a software-based solution capable of managing excess power from a smart home or similar installation (tracking use of imported/exported power, deciding when and how to use excess power, etc.), as well as visualizing it (consumption of different devices, computation of power costs/benefits, hardware resources, etc.).

1.3.2 Sub-objectives

In order to attain the above-mentioned objectives, they will be further subdivided into simpler, more concrete sub-objectives:

- Study the IoT devices that will be used in the project as well as the protocols used to interface with them.
- Study the various software components that will be used within the project and how they interact with each other.
- Explore and understand how electrical import and export prices are determined by electrical companies and develop algorithms to minimize costs.
- Install and configure the hardware devices according to the network topology.
- Implement the system by deploying and configuring the required components and ensure proper communication between them.

1.3.3 Requirements

To fulfill all of the objectives some requirements need to be met. These are further divided into hardware, functional and non-functional requirements.

Hardware requirements
These are some of the hardware devices that will be needed for the project:

- **Management unit**: The system should by driven by a central device with an additional requirement that it be low-powered, as it may need to be embedded into other systems. The chosen device is an Asus **MiniPC**, although it could easily be replaced by some other Linux low-power device such as a Raspberry.
- **Solar inverter**: The chosen device is an **Autarco** inverter.

Functional requirements
These are the main components of the system to be built:

- **IoT wiring**: The hub of the system, framework used to coordinate all devices and act as a middleman for the other components. **Node-RED** [23] will be used.
- **Messaging protocol**: Communication protocol able to communicate different IoT devices in an efficient manner. **MQTT** [126] will be used.
- **Database**: Fast and lightweight persistent storage able to store all the data generated by devices. **InfluxDB** [92] will be used.
- **Visualization tool**: Tool capable of displaying and transforming the information stored in the database. **Grafana** [104] will be used.
- **Container engine**: Component used to deploy and manage all other components as containerized services. **Docker** [62] will be used.

Non-functional requirements
During the development of the system, some quality attributes should be kept in mind to ensure a good final result. Following are the main such attributes in no particular order:

- **Reliability**: To provide a good user experience the system should be able to provide diagnostic information and tolerate failures. In particular it should be able to tolerate failure on services, devices and the network.
- **Performance**: Since most IoT devices have limited computational resources performance should be kept in mind when designing the system.
- **Reusability**: Modularity in software not only allows for a better understanding of the system by reducing its complexity but also make reusing existing pieces in other projects feasible.
- **Security**: Nowadays vulnerabilities are being found increasingly both in software and hardware, for this reason, systems should be designed with security in mind. This is specially important for IoT devices, since a compromised device could affect many others.

1.3.4 Obstacles and risks

As with every other project, some precautions must be taken to ensure that it can continue even in the face of unexpected events. What follows is a non-exhaustive list of obstacles and risks, short enough to be useful while remaining faithful to the actual project setting (otherwise, it would most likely be infinite).

- **Inexperience**
- **Security threats**
- **Data loss**
- **Hardware failure**
- **Network problems**

1.4 Methodology and rigor

Because a project without methodology is not such, this section will deal with the development methodology that will be applied throughout the project to ensure its proper planning and execution, as well as the development principles that will be followed so that all requirements are met with rigor.

1.4.1 Methodology

Traditionally, development and IT operations teams have carried out their work separately from each other, which often results in unexpected problems or a subpar end product due miscommunication between the teams. DevOps aims to solve this issue by integrating both aspects into a single methodology [110, 51].

In addition to integrating the development and administration of applications the DevOps methodology also tries to go beyond that and include stakeholders (end-users, customers, etc.) into the life cycle of the project.

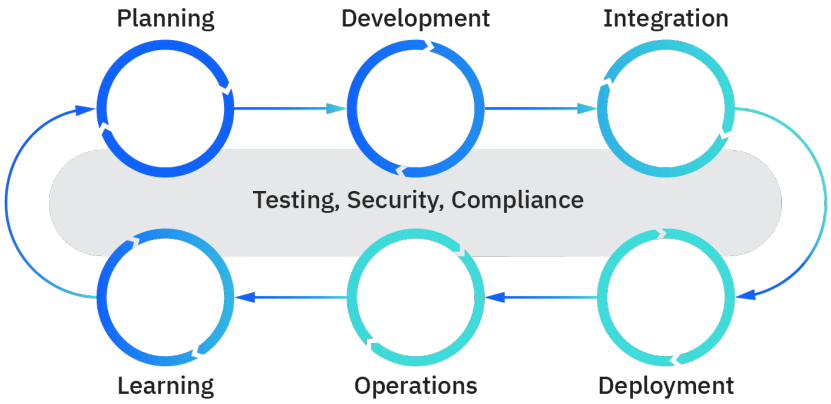


Figure 4: DevOps methodology [51]

Although this project is lead by a one-man team, its core principles remain applicable and its focus in integrating both development and administration should prove beneficial for our security and reliability requirements.

1.4.2 Rigor

Rigor will be enforced by following industry best practices: application development guidelines such as **Twelve-Factor App** from Heroku [155] or NGINX's **Microservices Reference Architecture** [147] and work will be tracked using **git** as a revision control system on **GitHub**.

In addition to that meetings with both the tutor and Ontec will be scheduled regularly to discuss updates in the project or notify changes in planning.

2 Time planning

2.1 Task description

Defining a time planning for a project requires a good understanding of the individual tasks that will need to be carried out during its course. This section aims to define tasks reflecting the objectives and requirements mentioned previously as well as to describe their dependencies on resources and other tasks.

The project was officially inscribed the 15th of December 2021, although actual work started some time after, and is expected to finish around June 2022. This leaves us with about 150 days and 500 hours of work, or about 3 hours of work per day.

2.1.1 Task definition

The project will be divided into four main tasks, some of which will be executed in parallel with others. These tasks will be briefly described and further divided into more manageable sub-tasks, which will be more precisely defined.

Project management
The appropriate management of a project is of utmost importance. The following are tasks that will allow the project to develop in a focused and controlled manner:

- **Meetings**: Meetings scheduled with the tutor to give updates on the implementation of the project and get feedback to ensure its proper progress.
- **Context and scope**: Description of the project's context and scope, including an overview of concepts used and a definition of its objectives and requirements.
- **Time planning**: Definition of the tasks to be carried out during the project, the resources required to fulfill said tasks and the dependencies between them.
- **Budget and sustainability**: Creation of a budget for the project and analysis of its sustainability considering economic, environmental and social dimensions.
- **Initial report**: Devising of a preliminary report based on the initial management tasks that will serve as basis for the final thesis report.

Documentation
As the project progresses changes and improvements made will need to be documented. Given its importance and changing nature, this task will be present all throughout most part of the project.

- **General documentation**: Documentation for developers and stakeholders.
- **Thesis report**: Creation of the document that will serve as a written record of the work done in the project as well as the problems faced during its course.
- **Thesis defense**: Creation of a presentation for the thesis defense aiming to provide a brief summary of the thesis report in spoken form.

Initial prototype
This task could be considered the first stage of the project, essentially the development of a minimum viable product with the essential features required by the main stakeholders. This stage of the project will be comprised of the following tasks:

- **Software study**: Study of the various software technologies that will be used in the project together with the possible interactions between them.
- **Device study**: Study of the various devices that will be used in the project as well as their software requirements and communication methods.
- **Environment setup**: Installation of the required combination of operating system and software on the initial devices and configuration of the system.
- **Prototype implementation**: Implementation of the actual system. It should be able to monitor system resources, keep track of the exported and imported power and distribute excess power among devices.

Final product
If the initial prototype was the first stage of the project, this is the second and last stage. The final product stage will consist in the addition of extra features, quality control and refinement of the prototype developed during the first stage.

- **Additional features**: Features not present in the original prototype that the stakeholders consider necessary for the final product.
- **Prototype testing**: Testing of the developed product to ensure that both functional and non-functional requirements are met.
- **Prototype polishing**: Improvement of the final product (e.g., user experience) and fixing of any flaws found through testing.

2.1.2 Resources

Many of the tasks depend on resources to be accomplished, these will be grouped into human and material resources and named accordingly to increase readability.

Human resources
These resources include the following people and organizations:

- **Student** [**H1**]: I, Pablo Esteban Baquero, will be in charge of developing, troubleshooting and documenting the project for the most part.
- **Tutor** [**H2**]: The tutor, Luis Velasco Esteban, will mentor the project by scheduling meetings with H1 and H3, giving advice on technical aspects and providing feedback for the overall project.
- **Collaborators** [**H3**]: The collaborators, Ontec Energy, will be in charge of defining the project requirements and providing most of the material resources included in M4 and M6.

Material resources
These resources include the following software and devices:

- **Documentation software** [**M1**]: The software used to document the project will consist in the LaTeX typesetting suite together with **Biber** as the bibliography backend and **GanttPro** for the creation of Gantt diagrams.
- **Version control software** [**M2**]: All code and documentation in the document will be tracked using the **git** version control system and **GitLab** will be used as a remote storage location to prevent data loss if something goes wrong.

- **Development software** [**M3**]: As mentioned in previous sections, the project will follow a microservice architecture managed with **Docker** and the main software components will be those mentioned in it. Additionally, **OpenVPN** will be used to allow remote access to the environment.
- **Development devices** [**M4**]: During the development of the prototype the only required devices will be the **ASUS MiniPC**, used as the development environment, together with some IoT devices provided by H2 and H1's personal computer.
- **Additional software** [**M5**]: Software required to carry out the implementation of additional features should the stakeholders require it.
- **Additional devices** [**M6**]: Devices required to carry out the implementation of additional features should the stakeholders require it.

2.1.3 Task summary

According to the FIB's academic regulations a thesis is worth 18 ECTS, which roughly corresponds to 500 hours of dedication. This section will give a rough time estimate for every task by distributing the total dedication time of the project among them.

Besides time estimates, dependencies on tasks and resources will also be indicated, but with some caveats. Dependencies of dependencies aren't explicitly mentioned to avoid clutter. For the same reason, resource H1 (i.e., myself) will be assumed to be implicitly required by all tasks.

ID	Task	Time	Dependencies	Resources
T1	**Project management**	**80h**		
T1.1	Meetings	20h		H2
T1.2	Context and scope	15h		
T1.3	Time planning	15h	T1.2	
T1.4	Budget and sustainability	15h	T1.3	
T1.5	Initial report	15h	T1.4	
T2	**Documentation**	**70h**		
T2.1	General documentation	25h		M1,M2
T2.2	Thesis report	25h	T1.5	M1,M2
T2.3	Thesis defense	20h	T2.1	M1,M2
T3	**Initial prototype**	**200h**		
T3.1	Software study	50h		M3,H2
T3.2	Device study	50h		M4,H2
T3.3	Environment setup	50h		M3,M4,H2
T3.4	Prototype implementation	50h	T3.3	M2-M4,H2
T4	**Final product**	**150h**	**T3**	
T4.1	Additional features	50h	T3.4	M2-M6,H3
T4.2	Prototype testing	50h	T4.1	M2-M6,H3
T4.3	Prototype polihsing	50h	T4.1	M2-M6,H3
Total		**500h**		

Table 1: Summary of the tasks

2.2 Gantt estimates

Figure 5 shows an estimated timeline for the project with all the tasks previously described. The start date is taken from the first meeting with the director and ends with the approximate date of the thesis defense.

Figure 5: Gantt diagram of the tasks

2.3 Risk management

Previously, risks and obstacles were only mentioned, in this section they will be revisited, and for each of them a solution will be proposed that accounts for their impact on the project, alternative tasks or additional resources that may be required.

2.3.1 Inexperience

Because I haven't previously worked on any IoT project and am not very familiar with the field any problems that arise related to it could pose a threat to the time planning. If such a problem was serious enough it could cause a delay in some of the tasks and as result affect the deadline of the project.

Hopefully my own study together with the tutor's guidance will be enough to avoid this type of situation. At any rate, to mitigate such an issue more time could be allocated to its troubleshooting while sacrificing some additional features.

2.3.2 Security threats

According to the DevOps methodology, security should be inherent to the development process and not separate process or an afterthought. As a result, if the methodology is followed, any security threats should be minimized.

Again, reality is often unpredictable and such contingencies must be accounted for. In the event that any security concerns arise they could be addressed by adding an additional task focused on the securing of the system as a whole.

2.3.3 Data loss

Any project that runs for long enough must at some point face the common problem of data loss. For this reason, the project will backup all of its code in a **GitLab** repository from day one. For the time being other types of data (e.g., resource monitoring data) won't be backed up as they can easily be regenerated.

Once the project enters its second stage, however, data loss could prove fatal for the project's progress. With that in mind, a new resource such as Amazon's **S3** might be required for the purpose of general data backups.

2.3.4 Hardware failure

Another common problem found in any project dealing with devices is that of hardware failure. This is specially true for projects involving multiple devices (as indicated by the MTTF formula), such as an IoT project.

If a hardware failure were to occur on a device it would most likely need to be replaced by another one. This would imply the purchase of an additional resource, which will have an impact on the budget and sustainability of the project.

2.3.5 Network issues

Many of the tasks in the project will be done remotely, as the physical devices will be in installations belonging to either H2 or H3. Connectivity issues due to bad Internet access or VPN misconfiguration could have an impact on productivity.

To get around this problem other tasks not requiring remote access could be prioritized while the problem persits or the remote enviornment be locally replicated.

3 Budget and sustainability

3.1 Budget

Once a planning has been laid out for the project, it becomes necessary to draw up a budget matching the contents outlined in it. This section will deal with the identification of costs by dividing them into human and material resources, relating them to the tasks that constitute the project as well as taking amortization, contingencies and incidentals into account.

3.1.1 Human resources

In order to compute costs derived from human resources, we must first define a list of roles together with their corresponding salary. Resources H2 and H3 will each correspond to a single role, but H1 will be divided into three roles that mirrors the main groups of tasks in the project. Table 3 shows these roles together with estimated hourly salaries based on data obtained through **Glassdoor** (except for H2, and H3). Table 2 shows the staff cost computed from the list of tasks according to the estimated dedication time of each role and their respective salaries.

ID	Task	Time per role					Cost (€)
		H1.1	H1.2	H1.3	H2	H3	
T1	**Project management**	**80h**			**20h**		**2180.00**
T1.1	Meetings	20h			20h		920.00
T1.2	Context and scope	15h					315.00
T1.3	Time planning	15h					315.00
T1.4	Budget and sustainability	15h					315.00
T1.5	Initial report	15h					315.00
T2	**Documentation**	**70h**					**1470.00**
T2.1	General documentation	25h					525.00
T2.2	Thesis report	25h					525.00
T2.3	Thesis defense	20h					420.00
T3	**Initial prototype**		**200h**		**20h**		**3900.00**
T3.1	Software study		50h				850.00
T3.2	Device study		50h				850.00
T3.3	Environment setup		50h		10h		1100.00
T3.4	Prototype implementation		50h		10h		1100.00
T4	**Final product**		**75h**	**75h**		**20h**	**2825.00**
T4.1	Additional features		50h				850.00
T4.2	Prototype testing			50h		10h	950.00
T4.3	Prototype polihsing		25h	25h		10h	1025.00
Total		**150h**	**275h**	**75h**	**40h**	**20h**	**10375.00**

Table 2: Staff cost estimates by task

ID	Role	Cost (€/hour)
H1.1	Junior Project manager	21.00
H1.2	Junior DevOps engineer	17.00
H1.3	Junior QA engineer	14.00
H2	Project tutor	25.00
H3	Project collaborator	25.00

Table 3: Hourly salary estimates for staff according to roles

3.1.2 Material resources

This section identifies costs that do not qualify as human resources and classifies them as either software or hardware resources.

Software

All software that will be used during the project is either open source or free to use, as a result software costs are nonexistent. Products marked with an asterisk include paid versions, but for our purposes its free version is enough. Tables 4, 5 and 6 show costs for development, storage and backup, and miscelaneous software respectively.

ID	Software	Cost (€)
SW1.1	Docker	0.00
SW1.2	MQTT	0.00
SW1.3	Node-RED	0.00
SW1.4	Grafana	0.00
SW1.5	InfluxDB	0.00

Table 4: Cost estimates for development software

ID	Software	Cost (€)
SW2.1	Amazon S3*	0.00
SW2.2	GitLab*	0.00
SW2.3	Duplicity	0.00

Table 5: Cost estimates for storage and backup software

ID	Software	Cost (€)
SW3.1	git	0.00
SW3.2	LaTeX	0.00
SW3.3	OpenVPN	0.00

Table 6: Cost estimates for other miscelaneous software

Hardware

Throughout the course of the project many hardware devices will be used, but for the purpose of this preliminar examination only those that are being used at the moment will be taken into account.

To account for amortizations, which will be computed using the following formula, HW1 and HW2 are assumed to be used for the duration of the project (500h) and HW3 and HW4 to always be on after being set up (4 months):

$$amortization = cost(euros) \cdot \frac{usage(hours)}{lifetime(hours)}$$

ID	Hardware	Lifetime	Cost (€)	Amortization (€)
HW1	ASUS Aspire 3	4 years	499.99	7.13
HW2	Logitech M220	3 years	25.99	0.49
HW3	ASUS MiniPC	4 years	354.00	29.10
HW4	Autarco SX-MII	10 years	500.00	16.44
Total				**53.16**

Table 7: Hardware cost estimates

3.1.3 Incidentals

Incidental costs are those related to the risks detailed in previous sections, more precisely, those having to do with the replacement of hardware devices due to failures or the necessity of overtime work (estimated as 50h) for a given role (due to inexperience, security threats, data loss or network problems). The probabilites given to each of the incidentals are rather arbitrary and could be enhanced by a more careful analysis.

Incidental	Probability (%)	Cost (€)	Estimate (€)
HW1 failure	5	499.99	25.00
HW2 failure	5	25.99	1.30
HW3 failure	5	354.00	17.70
HW4 failure	5	500.00	25.00
H1.1 overtime	15	1050.00	157.50
H1.2 overtime	15	1350.00	202.50
H1.3 overtime	15	700.00	105.00
Total			**534.00**

Table 8: Incidental cost estimates

3.1.4 Final budget

Defining an accurate budget is a difficult task, even more so when one does not have experience, and, as a result, deviations from the budget may arise from miscalculations. Table 9 shows the final budget together with an additional contingency cost that oversizes the orginal budget a 10% to account for possible deviations.

Concept	Base (€)	Contingency (€)	Final (€)
Human resources	10375.00	1037.50	11412.50
Material resources	53.16	5.32	58.48
Incidentals	534.00	53.40	587.40
Total	**10962.16**	**1096.22**	**12058.38**

Table 9: Final budget cost estimates

3.1.5 Management control

As it has been mentioned previously, the real budget might end up deviating from the computed one, even with the 10% oversize provided by the addition of contingencies. To facilitate control of the budget whenever deviations occur, some numerical indicators will be defined.

Concept	**Indicator**
Task cost	$(estimated\_cost - real\_cost) \cdot real\_hours$
Human resources cost	$estimated\_cost - real\_cost$
Material resources cost	$estimated\_cost - real\_cost$
Hour number	$estimated\_hours - real\_hours$

Table 10: Budget numerical indicators

3.2 Sustainability report

Every project needs to analyze its sustainability in a way that accounts for the complex reality it resides in. Following are a self-assessment regarding sustainability together with an analysis of the economic, environmental and social dimensions of the project.

3.2.1 Self-assessment

This section should contain a summary of the self-evaluation survey regarding sustainability, but that will not be the case. In its place, I, the author of this botch, will use this space to briefly complain about the academic system.

Most of modern academic production could be caracterized as tedious paperwork that nobody will ever bother to read. It comes as no surprise, then, that a great deal of the time spent on the TFG is to be dedicated to writing a report that no one cares about, instead of doing actual research or developing something worth the time.

As I see it, the way the TFG currently operates encourages the mass production of half-assed report papers that will just sit wasting space in the university's database and, perhaps, even hinder the discovery of noteworthy research due to an overabundance of mediocre work (this one being no exception).

Students, for the most part, will go on to work a normal job and not do any kind of academic research, so why should they be required to invest so much time into something that will be of little to no use in their career instead of focusing on the practical side of things?

It could be argued that the elaboration of a report helps the student develop some soft skills such as time management, critical thinking or communication competence, but I think that is far from reality. By the time the TFG must be done, most students have already attained such skills, and if in more than three years they have not, what makes somebody think that they will have by the end of it?

Overall, I think it is a huge time sink that, in most cases, serves no real purpose other than checking the corresponding box off on the university syllabus, and that both faculty staff and students would be better off without it.

3.2.2 Economic dimension

Have you estimated the cost for the completion of the project?

The cost of the project has been roughly estimated following some of the guidelines provided by the university. However, due to a lack of knowledge regarding economics and project management the quality achieved is probably quite low.

How will your solution improve economic issues with respect to other existing solutions?

The proposed solution will most likely not improve previous solutions but rather build on top of already working ones.

3.2.3 Environmental dimension

Have you estimated the environmental impact of the project? Did you plan to minimize its impact, for example, by reusing resources?

Although the environmental impact of the project has not been estimated as such, its impact will most likely be favorable. As the central idea of the project is that of measuring and managing power excess from photovoltaic generation, it could easily be expanded to account for environmental issues.

Many of the devices used throughout the project were already owned by some of the parties involved, so in a sense they are already being reused. In addition to that, most of them are low-powered, which reduces the energy consumption of the project.

How will your solution improve environmental issues with respect to other existing solutions?

The proposed solution will most likely not improve previous solutions but rather build on top of already working ones.

3.2.4 Social dimension

What do you think you will achieve, in terms of personal growth, from working on this project?

By the end of the project I expect to have learned more about the Internet of Things, microservices and how they relate to one another as well as hands-on experience regarding containerization technologies.

How will your solution improve environmental issues with respect to other existing solutions?

The proposed solution will most likely not improve previous solutions but rather build on top of already working ones.

Is there a real need for the project?

The main stakeholders do have a need for the project, but in more general terms, most likely no, as there already are alternative solutions to the one presented herein.

4 Planning changes

As it was previously mentioned, any project is subject to unexpected events that are bound to have an impact on its original planning decisions. While minor changes were to be expected, some noticeable changes have been necessary.

The following section is written to account for said changes, making amends to the scope, time planning and budget definitions outlined in the previous sections.

4.1 Scope

Lack of experience, remote work, and miscommunication, among many other factors, have impacted and somewhat limited the extent to which the project could progress. This has required a change in its scope to better reflect reality.

Although the core of the project remains the same, that is the development of a photovoltaic excess management and visualization system, its main objective has shifted from the creation of a fully working prototype to the creation of a more limited proof of concept, together with the documentation of obstacles and solutions.

Sub-objectives previously described are broad enough that they can still be used as long as one keeps in mind the new main objective described above.

4.2 Time planning

ID	Task	Time	Dependencies	Resources
T1	**Project management**	**80h**		
T1.1	Meetings	20h		H2
T1.2	Context and scope	15h		
T1.3	Time planning	15h	T1.2	
T1.4	Budget and sustainability	15h	T1.3	
T1.5	Initial report	15h	T1.4	
T2	**Documentation**	**70h**		
T2.1	General documentation	25h		M1,M2
T2.2	Thesis report	25h	T1.5	M1,M2
T2.3	Thesis defense	20h	T2.1	M1,M2
T3	**Initial prototype**	***275h***		
T3.1	Software study	*75h*		M3,H2
T3.2	Device study	*75h*		M4,H2
T3.3	Environment setup	*75h*		M3,M4,H2
T3.4	Prototype implementation	50h	T3.3	M2-M4,H2
T4	**Final product**	***75h***	**T3**	
T4.1	Additional features	*25h*	T3.4	M2-M6,H3
T4.2	Prototype testing	*25h*	T4.1	M2-M6,H3
T4.3	Prototype polihsing	*25h*	T4.1	M2-M6,H3
Total		**500h**		

Table 11: Summary of the tasks after reassessment

The aforementioned change in scope also implies a change to the the way work hours are split between tasks. The number and order of tasks will however stay the same, as the sub-objectives and, therefore, logical units of work remain mostly the same

Because a lot more time than expected has been devoted to finding, debugging and solving both hardware and software issues, more hours have been allotted for both hardware and software study tasks as well as environment setup. The hours given to these tasks will be deduced from those related to improving the final product.

4.3 Budget

After reworking the distribution of hours, the budget must be modified accordingly. In addition to that, some new devices have been acquired during the course of the project, increasing costs and, as a result, requiring a reassessment of the budget.

Since the number of hours is unchanged and the only other factor having an impact on human resources is the hourly wage of a given resoruces a similar cost is to be expected.

ID	**Task**	**Time per role**					**Cost (€)**
		H1.1	**H1.2**	**H1.3**	**H2**	**H3**	
T1	**Project management**	**80h**			**20h**		**2180.00**
T1.1	Meetings	20h			20h		920.00
T1.2	Context and scope	15h					315.00
T1.3	Time planning	15h					315.00
T1.4	Budget and sustainability	15h					315.00
T1.5	Initial report	15h					315.00
T2	**Documentation**	**70h**					**1470.00**
T2.1	General documentation	25h					525.00
T2.2	Thesis report	25h					525.00
T2.3	Thesis defense	20h					420.00
T3	**Initial prototype**		***275h***		**20h**		***5175.00***
T3.1	Software study		*75h*				*1275.00*
T3.2	Device study		*75h*				*1275.00*
T3.3	Environment setup		*75h*		10h		*1525.00*
T3.4	Prototype implementation		50h		10h		1100.00
T4	**Final product**		***37.5h***	***37.5h***		**20h**	***1162.50***
T4.1	Additional features		*25h*				*425.00*
T4.2	Prototype testing			*25h*		10h	*600.00*
T4.3	Prototype polihsing		*12.5h*	*12.5h*		10h	*637.50*
Total		**150h**	***312.5h***	***37.5h***	**40h**	**20h**	***10487.50***

Table 12: Staff cost estimates by task after reassessment

As will be explained in later sections, new devices were acquired after having finished the initial budget estimate. Although they were not bought at the exact same time as previous devices, it was a close enough date that ammortization will be computed over the same period of time, i.e. 4 months.

ID	Hardware	Lifetime	Cost (€)	Amortization (€)
HW5	Shelly 3EM	2 years	119.88	19.71
HW6	Shelly Plug S (3)	2 years	57.24	9.41
HW7	OpenEVSE	4 years	300	24.66
Total				**53.78**

Table 13: Additional hardware cost estimates

We also need to account for inicidental costs related to the new hardware devices:

Incidental	Probability (%)	Cost (€)	Estimate (€)
HW5 failure	5	119.88	5.99
HW6 failure	5	57.24	2.86
HW7 failure	5	300	15
Total			**23.85**

Table 14: Additional incidentals cost estimates

After adding everything up we end up with a relatively similar final budget:

Concept	Base (€)	Contingency (€)	Final (€)
Human resources	10487.50	1049.80	*11537.30*
Material resources	53.16	5.32	58.48
Incidentals	534.00	53.40	587.40
Additional material resources	53.78	5.38	*59.16*
Additional incidentals	23.85	2.39	*26.24*
Total	***11152.29***	***1115.23***	***12267.52***

Table 15: Final budget cost estimates after reassessment

If we now compute the differences between the previous and current budget we get $10487.50 - 10375.00 = 112.5$ (1% increase) for human resources, 59.16 (101% increase) for material resources and $12267.52 - 12058.38 = 209.14$ (2% increase) for the total budget. Considering the original 10% budget oversize it seems a somewhat reasonable deviation.

5 Development process

The following section aims to document the development process of the project, including some of the problems found during it as well as the solutions or workarounds for said problems. Some particular details might have been left out for not being specially relevant or because of time constraints.

The narration more or less follows the order in which events happened during actual development. Explanation of the events are intertwined with relevant sections of logs or code, unless they are too long, in which case they can be found in the appendix, at the end of this document.

Finally, in addition to the content described above, the appendix also contains a user manual (A) which describes in a straightforward manner the installation and setup of the system, targeted at a potential developer wanting to test it.

5.1 First steps

5.1.1 Central hub

The first thing that had to be done was to decide the physical device that would be in charge of receiving data, processing it and delivering the necessary signals to other devices, essentially providing a central hub for communications.

At the start of the project the hardware in mind was some kind of low-power such as a **Raspberry**, as it would not only provide a cost-effective and well-tested device, but also one that is thoroughly documented and backed by a large user community, which would make development easier. This idea, however, was quickly discarded, as, at the time of writing, these devices are not easily available [153] due to a prevalent global chip shortage [24].

In its place, it was decided that a **MiniPC** [14] from **ASUS** would be used, which, although more expensive, had more computing power and runs an `x86` processor instead of an `ARM`, which perhaps could have made things more complicated in terms of installing packages and troubleshooting.

Operating system
Once the appropriate hardware device had been chosen the next task would be to select an operating system and install it. The choice was restricted to **Linux** distributions, as that is what is most commonly used in these types of installations and what I am most familiar with.

Because the system would need to be stable and easily maintainable, rolling release distributions were discarded, and because the target hardware is somewhat limited in computing power the decision made was `Ubuntu Server`. The main difference between the server and desktop versions boils down to some configuration aspects as well as a headless (i.e. no GUI) approach to the server version to reduce unnecessary workload. If sometime later there was a need for a graphical interface it would be possible to install it nevertheless.

The installation was carried out with a `USB` formatted with the `Ubuntu` live CD image [150]. During the process a few problems arose which will be described together with the solutions devised.

MiniPC network card

The first problem encountered when trying to install the operating system was the lack of internet connection. According to the specifications [14], the `MiniPC PN41` had both a wired network interface (`RTL8125B-CG` [137]) and an integrated wireless network interface, neither of which was recognized. Even if one decided to proceed with the installation, it would eventually run into problems, so it had to be fixed.

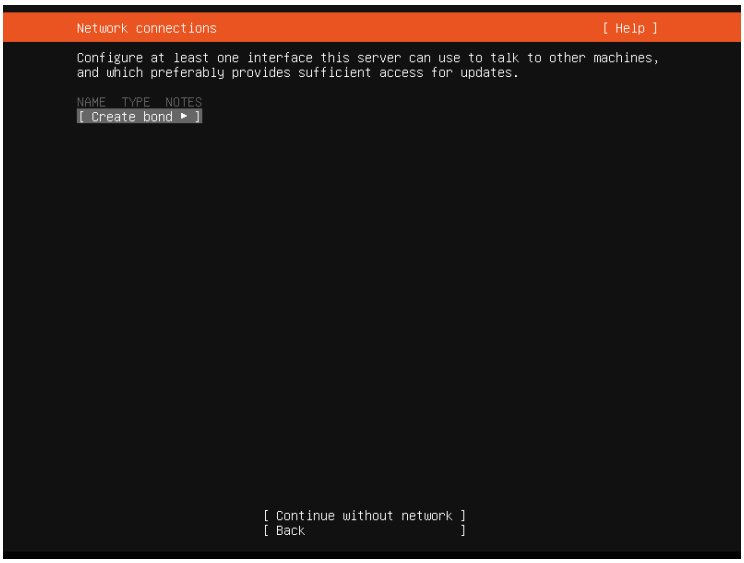


Figure 6: Ubuntu installer missing network error (recreated on **VirtualBox**)

As it turns out, the **Ethernet** network interface was not recognized because of a missing driver on the live CD, as it was a somewhat recent addition to the mainline kernel and had not yet made its way to the kernel shipped in the default `Ubuntu 20.04` version [133].

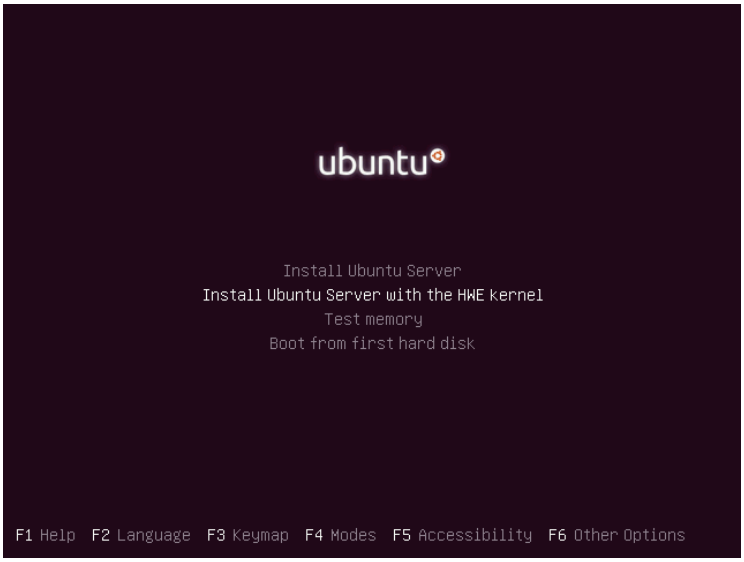


Figure 7: Ubuntu installer `HWE` option (recreated on **VirtualBox**)

Indeed, the command `uname -a` reported a `5.4.0` kernel on the default installer while the driver for the network card in question was first introduced in the `5.9.0` version [133], so it makes sense that it did not work.

The proposed solutions were to either install Ubuntu with Hardware Enablement (`HWE`), a stack providing support for a more recent kernel [151], or to manually install **Realtek**'s driver [136]. After booting the live CD with the `HWE` and executing `uname -a` again the kernel version returned was `5.13.0`. Running `lshw` now displayed the expected output, internet was accessible and the installation could proceed.

```
*-pci:1
     description: PCI bridge
     product: Intel Corporation
     (···)
   *-network
        description: Ethernet interface
        product: RTL8125 2.5GbE Controller
        vendor: Realtek Semiconductor Co., Ltd.
        (···)
```

Listing 1: Relevant output of `lshw` after selecting the `HWE` installer

MiniPC SSD

Right after the network problem was solved another more concerning one arose. According to the installer the system had no storage devices connected to it, which meant it would be unable to continue unless one was supplied.

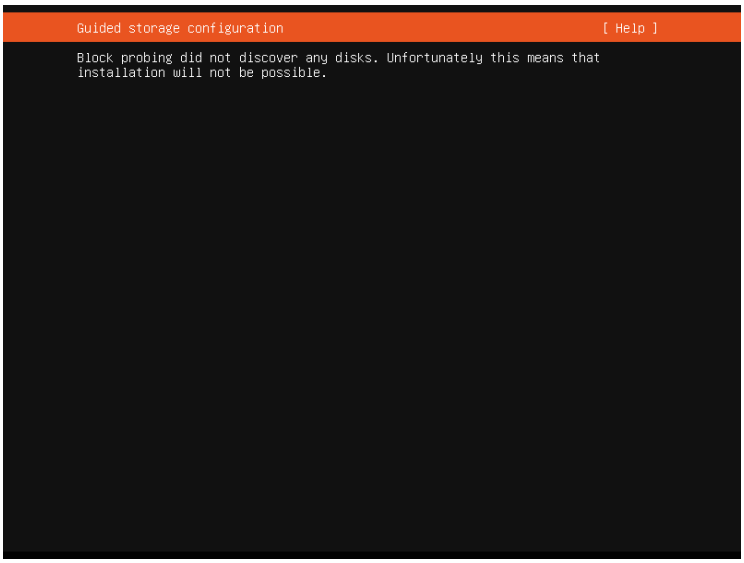


Figure 8: Ubuntu installer missing disk error (recreated on **VirtualBox**)

This time the specifications [14] allowed for a variety of devices, but according to the retailer who sold it, it should have had an `NVMe SSD` attached by default. The computer's `UEFI`, however, agreed with the installer in not detecting any storage devices. This made it even stranger, since if it could not see it either there must have been something wrong with the hardware.

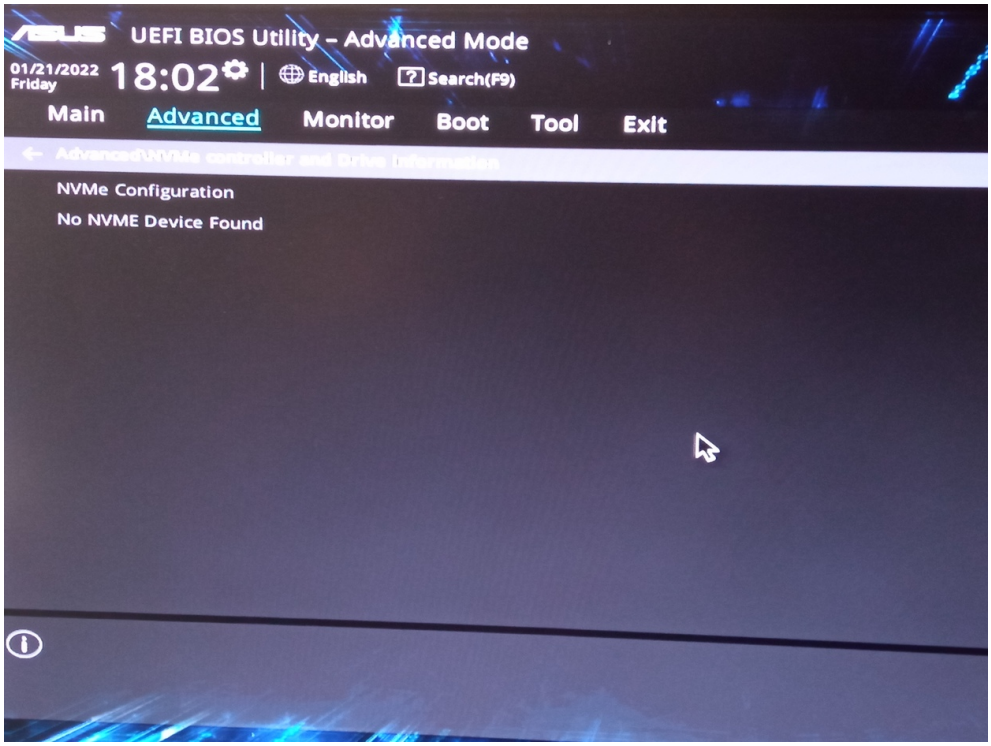


Figure 9: UEFI showing no NVMe devices

For the time being, the operating system was installed on a USB stick, and while everything else worked as it should have, lsblk and lshw still gave no clue as to what could be wrong with the SSD device, the system seemed unable to detect it.

```
NAME   MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
loop0    7:0    0 55.4M  1 loop /snap/core18/2128
loop1    7:1    0 32.3M  1 loop /snap/snapd/12704
loop2    7:2    0 70.3M  1 loop /snap/lxd/21029
sda      8:0    1  7.4G  0 disk
+-sda1   8:1    1  512M  0 part /boot/efi
`-sda2   8:2    1  6.9G  0 part /
```

Listing 2: Output of lsblk after installation on a USB

Fearing that the issue might have been a hardware one, I decided to take a look inside the computer's case and, surprisingly, at a first glance nothing seemed out of place: the SSD was properly connected where it should have been, there was no apparent damage to any of the components, and, most importantly, nothing smelled burnt.

After noticing that the installed SSD was labeled as a SATA and doing looking around the problem was found out to be an incompatibility at the interface level. As the UEFI had previously reported, no NVMe was found, because the inserted device was not one. Even if the form factor is the same M.2 [100] the underlying protocol must also match the one the hardware interface is expecting, NVMe in this case [101].

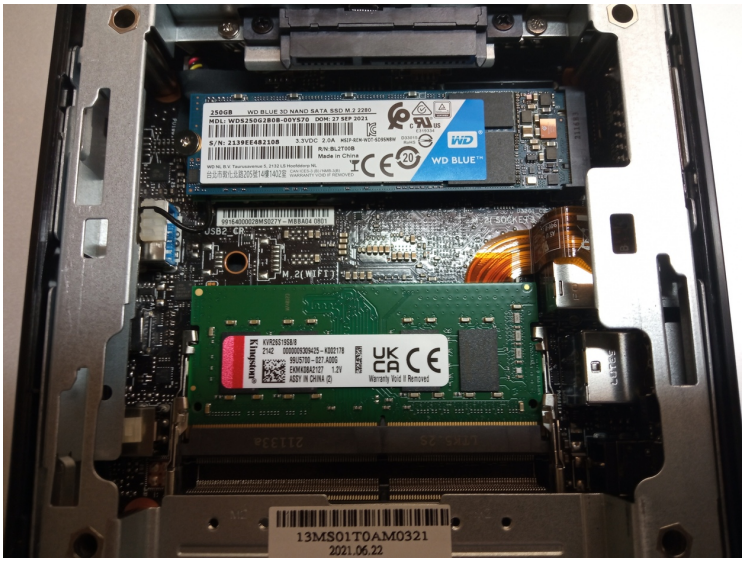


Figure 10: SATA SSD inside the MiniPC

As a result, the machine had to be brought back to the retailer so the incompatible SSD could be replace with an appropriate one. Finally, after having the brand new NVMe installed everything was working as expected. lshw now showed an entry for the device and, during installation, partitions were created without any problems, as can be seen in the output of lsblk.

```
*-pci:0
     description: PCI bridge
     product: Intel Corporation
     (...)
   *-storage
        description: Non-Volatile memory controller
        product: Sandisk Corp
        (...)
      *-nvme0
           description: NVMe device
           product: WDC WDS500G2B0C-00PXH0
           (...)
```

Listing 3: Relevant output of lshw after installing NVMe

```
NAME        MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
loop0         7:0    0  55.5M  1 loop /snap/core18/2344
loop1         7:1    0  44.7M  1 loop /snap/snapd/15534
loop2         7:2    0  67.8M  1 loop /snap/lxd/22753
nvme0n1     259:0    0 465.8G  0 disk
+-nvme0n1p1 259:1    0   512M  0 part /boot/efi
`-nvme0n1p2 259:2    0 465.3G  0 part /
```

Listing 4: Output of lsblk after installation on the NVMe

5.1.2 Initial software overview

After having installed the operating system it was time to start testing locally the different technologies that would be involved in the project, as, for the most part, I had never used them before.

What follows is a diagram (this one and others in future sections have been created using the free graph editor tool **yEd**) of the main software services being run on the device, either containerized in Docker or as native system services, some of the relationships among them as well as the port mapping of containers in the case of Docker services or the exposed ports in the case of native ones:

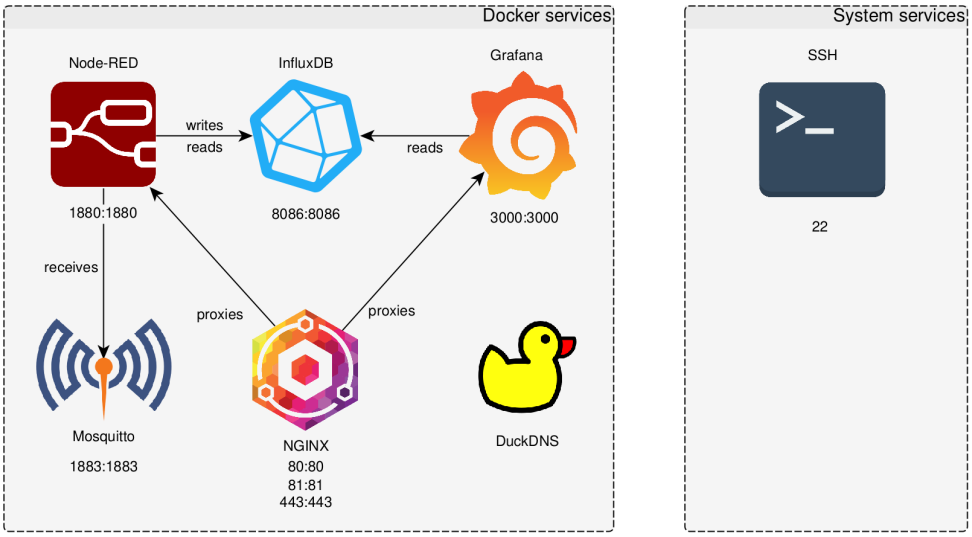


Figure 11: Diagram of the local software stack

The following sections aim to describe in some detail the role that each technology plays within the whole stack. The initial setup process is also skimmed over, but proper installation and configuration instructions can be found in the appendix, either in the user manual (A) or in scripts following the manual.

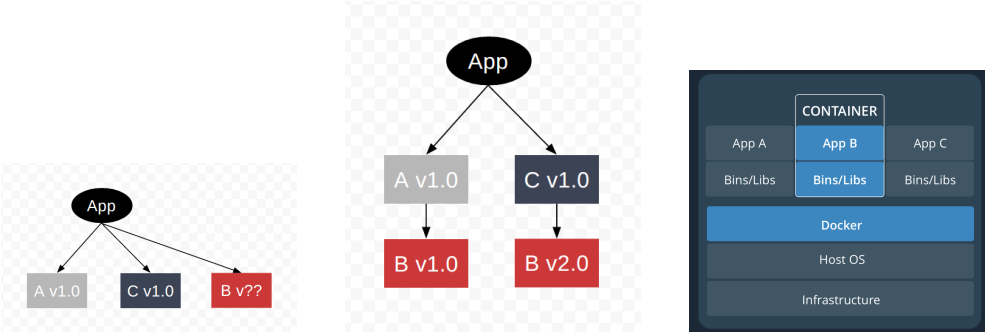
Note, however, that this section describes the software used in the initial local setting, which will be improved and modified in later versions, including the removal or replacement of some technologies due to changes in design.

Throughout the explanation some pieces of code will be referenced which, unless they appear inlined with the rest of the explanation, can be found in the appendix's listings section (B). Some fragments containing sensible information, such as passwords or tokens in configuration files, have been marked as redacted, but when replaced with valid values they should still be functional.

Docker

Historically, software has been developed in the exact same environment as the rest of the operating system and by using the same tools (e.g. package managers) for the dependency management of both. This coexistence often was the cause for the so-called dependency hell [27, 124], an aptly named phenomenon consisting in the clash of software packages due to reliance on different versions of the same dependency.

More recently, different programming languages/frameworks have tried to tackle the issue by means of different methods, the most well known relying on the isolation of system and development directories, such as `python`'s virtual environments [135] or `npm`'s local packages [123]. The list of examples could go on and on, but all of them suffer from a common issue, that of being tied to a particular software.



(a) Package manager (b) Module manager (`npm`) (c) Container engine

Figure 12: Dependency conflict management with different abstractions [124]

As the cloud gained traction and DevOps methodologies became predominant, the increasing number of deployments proved that the need for faster, cheaper and more reproducible environments could not be covered by virtual machines and CI/CD technologies alone, and that a new solution was required to tackle the problem.

Containers (or a primitive ancestor) initially appeared as a means to isolate processes in order to provide a restricted environment to the user [156]. Eventually, they evolved into what they are today, and, leveraging some of the same concepts mentioned above, they implement the concept of operating system virtualization [159], essentially providing a per-application virtual operating system without incurring the cost of traditional virtual machines.

As it was been mentioned throughout the document, **Docker** [56] is the container engine that will be used throughout the project. Although many other container engines are available, **Docker** is one of the most well known as it provides a robust **CLI** fronted allowing the developer to use it without having to bother about details.

Although the **CLI** might initially provide a good starting point for individually testing containers, as the number of required containers grows, it becomes unwieldy to deploy them by hand. **Docker compose** [58] was created for this purpose, allowing users to define a list of services with their configurations in `YML` format, specifying dependencies among services, metadata and other parameters.

What now follows is a short overview of the technologies that were initially used for local development, most of which will still be used later on. The appendix contains the compose file (B.1) used at the time together with other relevant files.

Node-RED

The first problem that one might think of when dealing with IoT might be that of interconnecting hardware devices, and while that is certainly a rather important one, there is another even more crucial: the interconnection of logical components.

After hardware devices are set up, there still is a big barrier preventing them from being used together in a useful manner. After all, what use are sensors if their data can not be easily processed or switches if they can not be switched from a simple user interface. There is a need for a central tool that is able to centralize both the management of device data as well as the issuing of commands.

In this project, the tool in charge of said task will be **Node-RED**, a web based programming application able of wiring hardware devices with different protocols, interacting with databases, creating dashboards to visualize information and many other interesting features [23].

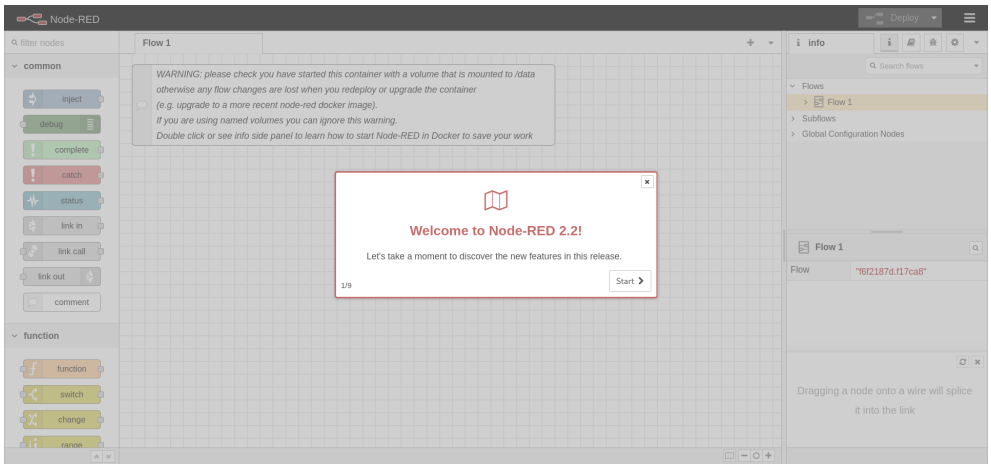


Figure 13: Node-RED initial welcome screen

Node-RED belongs to the family of visual programming languages [160], which makes it more accessible for beginners and non-programmers. More precisely, it is a flow-based programming language [157], allowing for the creation of isolated components (named nodes) that can be reused and connected to other existing components. In addition to the above, it makes of use of `JavaScript` as the underlying language for its components. All of these traits make for a quite user-friendly framework with a large and active community.

As with most services used throughout the project, the software required is distributed as a docker image [89] that can be run either manually or as a service in a `docker-compose` deployment. For our purposes, no special configuration is required, although the image offers support for credentials and some other configuration through environment variables. Additional information on the image can be found on the official website [122].

Mosquitto

Whenever two or more devices need to communicate, a messaging protocol must be used. This could be anything ranging from physical layer wire communication, such as serial protocol, to application layer `HTTP` request based communication.

Since many possible solutions to the problem exist it is important that we consider all parameters at play that could make us choose on or another protocol. The main requirements that will shape the final decision are those of IoT devices, namely a tendency towards wirelessness (mainly `Wi-Fi` based) as well as the provision of leeway for growth in the number of devices and, as a result, the volume of communications among those.

These restrictions make the previously mentioned protocols completely unfeasible. Wired connections such as serial port communication are out of the question, as they would make installation a nightmare. Classic point to point protocols such as `TCP` or even `HTTP` are also not ideal, as when the number of devices increased, the maximum number of connections required would grow exponentially (granted not all devices will talk to one another, but as long as they speak to more than one the number can quickly get out of hand).

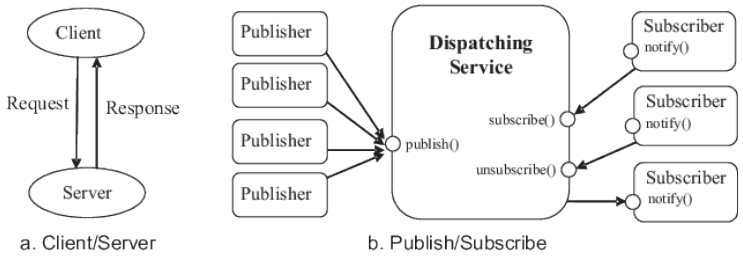


Figure 14: Comparison of the client-server and publish-subscribe models [139]

Fortunately, the `MQTT` [126] protocol provides exactly what we need. Using the `TCP` protocol at the transport layer it provides a publish-subscribe [38] model that allows clients to communicate by sending (publishing) topic-based messages to a broker server that will then deliver them to those devices that were previously subscribed to the topic of the message.

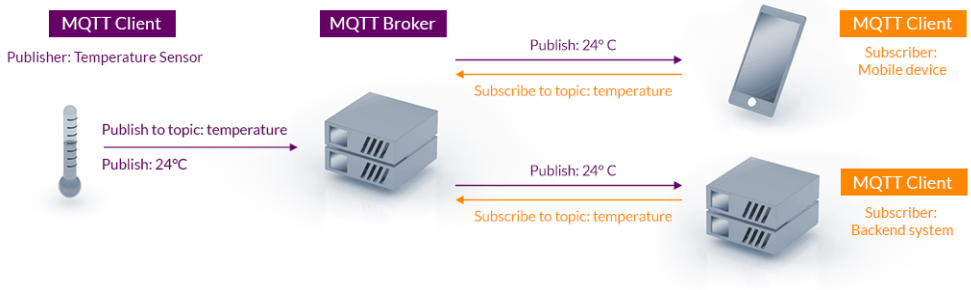


Figure 15: Diagram of client devices communicating through an `MQTT` broker [126]

In our stack the role of the broker server will be played by **Mosquitto** [117], which is also provided as a docker image [82]. Although the service offers additional configuration such as authentication, it will not be necessary for our purposes.

InfluxDB

With devices being able to send data through `MQTT`, the next obvious requirement is a way to store said data in a simple and efficient manner, essentially a database of some kind. However, different types of databases exists and many implementation exist for each one of them, so we must first decide which one fits our data the best.

Traditional relational databases [39] such as **MySQL** [119] or **PostgreSQL** [120] are based around the notions of tables and relations among the columns of one or more of them. More recent `NoSQL` databases [37] focus mainly on scalability with numerous database replicas, with some of the most well known being document-oriented databases such as **MongoDB** [115] or **CouchDB** [13]. While many databases belonging to these categories can be repurposed to manage IoT workloads, they are not the perfect fit for them, as they were not devised with the nature of IoT in mind.

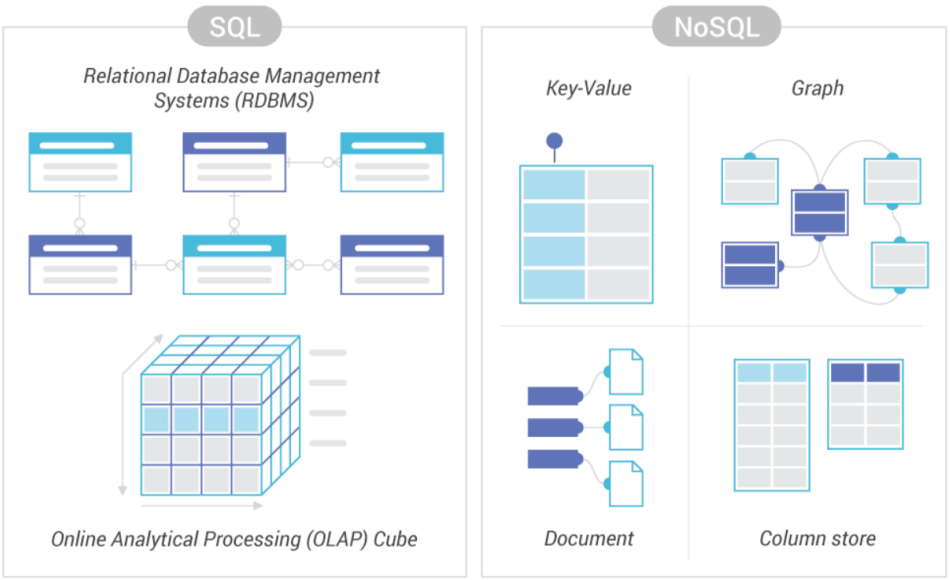


Figure 16: Comparison of key differences between relational (`SQL`) and `NoSQL` [140]

According to some, the ideal solution for IoT devices that generate many periodic measurements such as sensors appears to be a new category of emerging databases named time series databases [48]. These kind of databases are optimized for large and uniform datasets that usually emerge from periodic measurements composed of data with an associated timestamp. In addition to optimizations based on specialized compression algorithms these databases allow the configuration of the time to live of measurements, which helps with storage space management.

The chosen database was **InfluxDB** [92], which has two main versions: `1.X` and `2.X`. Out of the two, the latter was chosen, as it provides a more capable query language and offers some degree of compatibility with previous versions. Although the different versions make finding configuration information somewhat confusing at first, the project is very thoroughly and well documented.

The first thing that must be done to interact with **InfluxdDB** from another application is to configure the initial bucket, user and organization. The configuration was initially carried out manually by using the `influx setup` command [93].

```
# influx setup
> Welcome to InfluxDB 2.0!
? Please type your primary username username
? Please type your password ********
? Please type your password again ********
? Please type your primary organization name organization
? Please type your primary bucket name bucket
? Please type your retention period in hours, or 0 for infinite 0
? Setup with these parameters?
  Username:          username
  Organization:      organization
  Bucket:            bucket
  Retention Period:  infinite
 Yes
User            Organization    Bucket
username        organization    bucket
```

Listing 5: **InfluxDB** configuration for default user, organization and bucket

After having configured the basic elements, we need to set up a database mapping to enable backwards compatibility with `InfluxQL` queries [94, 95], which were the default in versions `1.X` and allow `Grafana` to use an `SQL`-like query builder.

```
# influx v1 dbrp create --db database --rp policy --bucket-id
↪  cda4c0ac4521217f --default
ID                      Database        Bucket ID
↪  Retention Policy         Default Organization ID
097819efc2aff000        database        cda4c0ac4521217f
↪  policy                   true    7a144e35e679ecb6
# influx v1 auth create --read-bucket cda4c0ac4521217f --write-bucket
↪  cda4c0ac4521217f --username username
? Please type your password ********
? Please type your password again ********
ID                      Description     Username        v2 User Name
↪  v2 User ID               Permissions
09781c33b7aff000                        username        username
↪  09781999746ff000
↪  [read:orgs/7a144e35e679ecb6/buckets/cda4c0ac4521217f
↪  write:orgs/7a144e35e679ecb6/buckets/cda4c0ac4521217f]
```

Listing 6: **InfluxDB** configuration for compatibility with `InfluxQL`

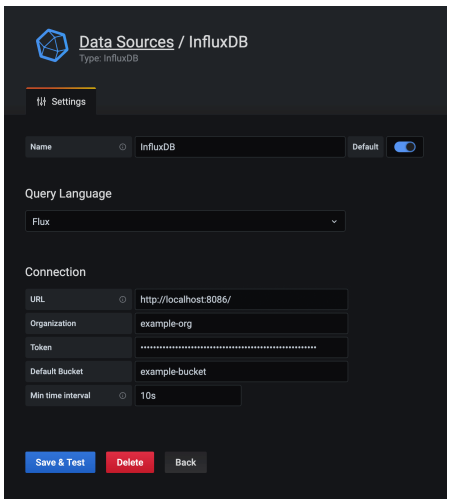
InfluxDB will be deployed using its official docker image [84], which allows for a lot of automatic configuration through environment variables as well as scripts. Initially all configuration was done manually but, as we shall see later on, the tools provided are very handy to perform the initial setup on a new device or when trying changes locally.

Grafana

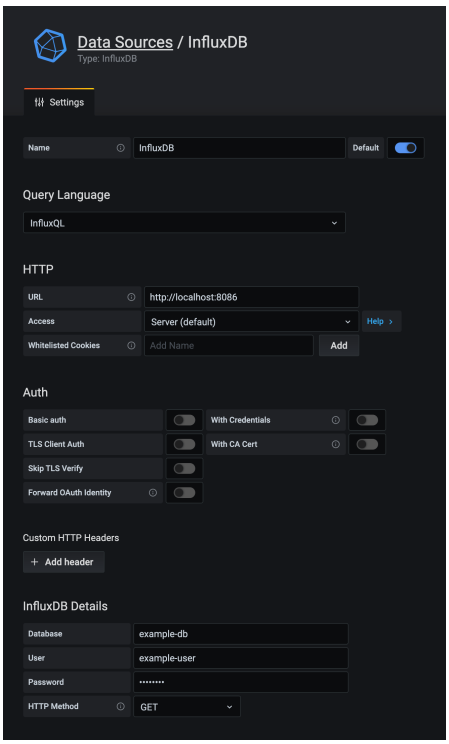
Once the storage of generated data has been taken care of, there needs to be a way of visualizing it. Some of the desired properties of the software in question are the ability to easily extend it with new components, an easy learning curve for new users and reliability.

With the restriction that it be user-friendly and compatible with `InfluxDB` two main options appeared to be widely adopted: **Grafana** [104] and **Chronograf** [91]. Since **Grafana** has a large following and is not tightly coupled with any particular data source (unlike **Chronograf**, which is tied to **InfluxDB**), it was chosen.

After creating a user and logging in, data sources for **InfluxDB** must be configured before any data can be queried. The official website for **Grafana** has limited information [105, 103], while its **InfluxDB** [95] counterpart provides a much more detailed guide on configuring both the data sources and database.



(a) Configuration for `Flux`



(b) Configuration for `InfluxQL`

Figure 17: **Grafana** InfluxDB data source configuration [95]

As with the other services, **Grafana** [83] comes packaged as a docker image. Configuration options are available for credentials and many other settings, with the official website detailing many of them [106]. At a later time some of them will be used to allow users to view dashboards anonymously as well as some other default behaviors.

DuckDNS

If we want to allow remote connections into our device, there are two main criteria that must be fulfilled if we want it to be practical. First, our device must somehow be accessible from an external identifier (IP or domain name), and, second, that identifier must be static.

Details on how an external IP can map to an internal IP will be given on the next section, so let us focus on the latter criterion. We need that our identifier be static so that changes in the state of the network do not impact how we access the device. Ideally, we could use the router's external IP as a static identifier, but since IPs are a scarce [32] resource nowadays ISP mostly assign IPs dynamically, unless they are paid for. Unless we want additional costs, this leaves us with the second alternative: domain names.

A domain name, or more precisely a DNS record, maps a string of characters to an external IP [28, 22] (technically not all DNS records map to IPs [33]). Since DNS records can be made static for as long as the owner of said records wants we can use them to identify our device. There is a catch, however; in order to own DNS records associated to a domain name one must first buy it from a regsitrar [21].

A third alternative, free Dynamic DNS subdomains, gives us a solution with no additional costs and the added benefit of simplicity. Dynamic DNS [29, 20] is a service offered by some DNS providers by which a user is allowed to modify the IP mapping of their domain name by means of some software solution. Combining Dynamic DNS with a free subdomain provider such as **DuckDNS** [65] we are able to obtain a static identifier for our remote device.

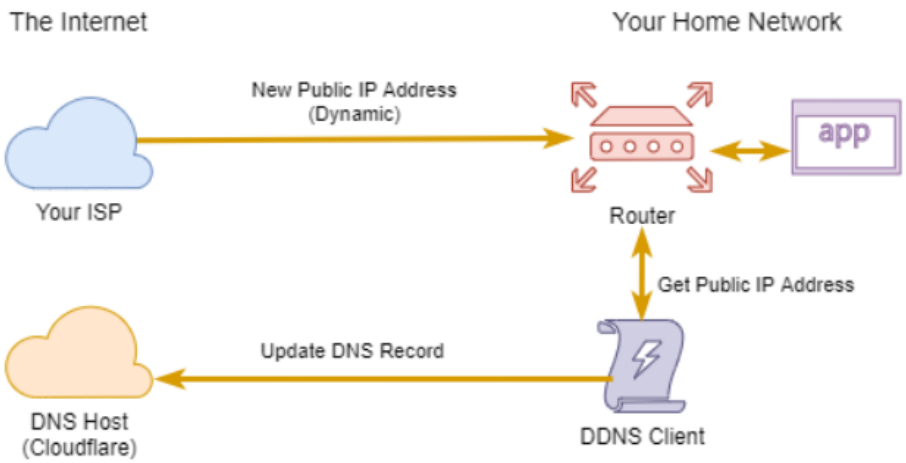


Figure 18: Network communications involved in Dynamic DNS [19]

In the case of DuckDNS, the Dynamic DNS service is provided via a `REST` API that requires a user token generated after singing up with an account. Although it would not be too difficult to setup either a `cron` or `systemd` service, a docker image [81] providing scripts configurable through environment variables already exists.

NGINX

If we want to provide remote access to web services without worrying about forwarding ports for all of them, the easiest solution is to use reverse proxy [40]. A reverse proxy, together with the corresponding DNS configuration, allows a single machine to provided multiple services on a single port by treating different domain names as virtual hosts.

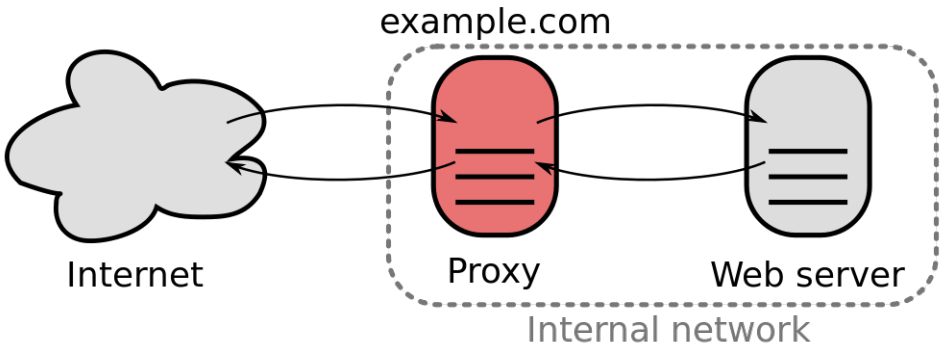


Figure 19: Example of a reverse proxy with a single web server [40]

Beside being a well known web server, **NGINX** also has reverse proxy capabilities [121]. By default it is managed through configuration files, which requires some additional knowledge, so instead a graphical web application alternative called **NGINX Proxy Manager** [96] will be used to make configuration easier. The application is provided as a docker image [88] and offers support for many of the features in **NGINX**.

In order to create the proxied hosts, first a valid domain name is required, for the example a dummy domain obtained from **DuckDNS** (`tfgtest.duckdns.org` will be used. Because we want traffic to be encrypted we also need to generate a valid `TLS` certificate, which can be done automatically through the application using the **Let's Encrypt** service [67].

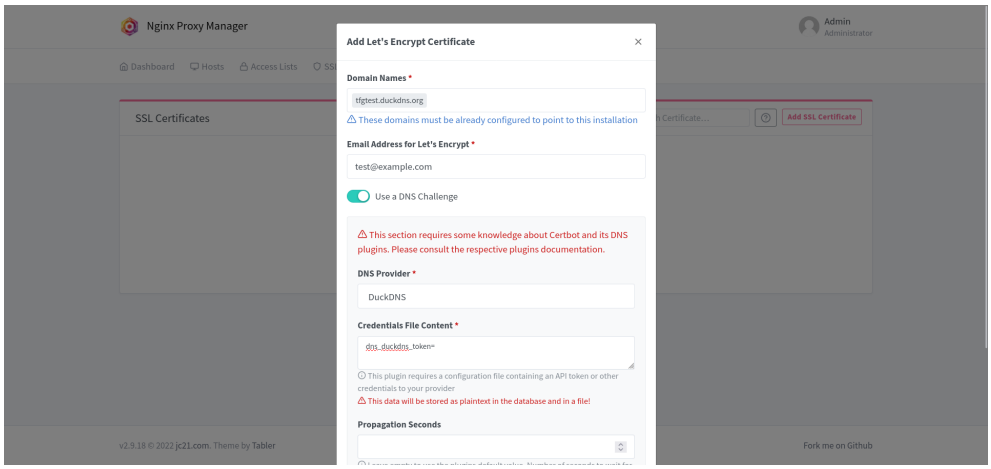


Figure 20: Configuration of a **Let's Encrypt** certificate through **DuckDNS**

Once the configuration form for the certificate is submitted a `DNS` challenge [66] is issued to **DuckDNS**, which verifies that the issuer is indeed the legitimate owner of the domain (in this case by means of a user token) and after waiting for the **Let's Encrypt** server to respond, a fully working certificate is automatically generated.

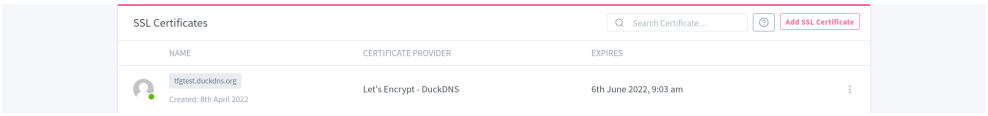


Figure 21: List of `TLS` certificates after creation

After having generated the certificate creating the proxy hosts is as simple as deciding a further subdomain and mapping it to a port on the base domain. In the example domains are created for the **Node-RED** `nodered.tfgtest.duckdns.org` and **Grafana** `grafana.tfgtest.duckdns.org` services.

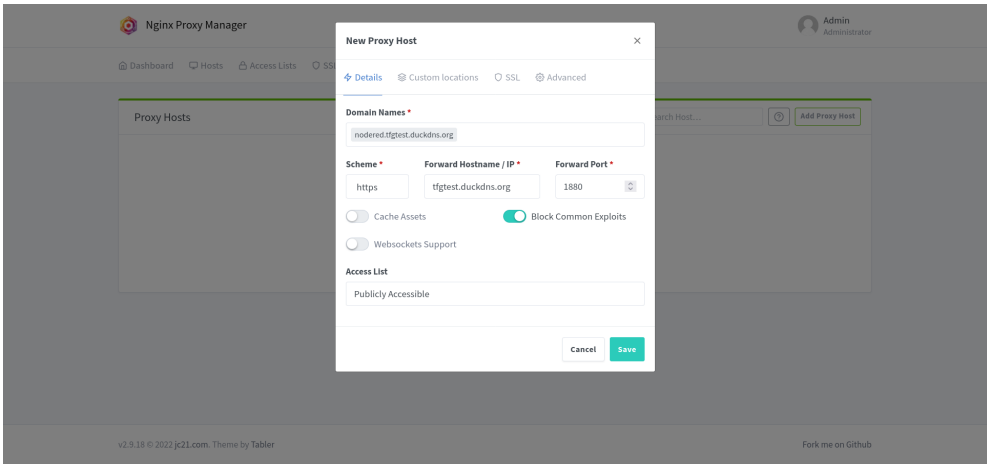


Figure 22: Configuration of a proxy host

Figure 23: List of proxy hosts after creation

Initially, providing public access to the services was considered an important use case, so that if a user wanted to access any of its services from a remote location it would be possible. As it will be explained at a later section, although still possible, this method was deemed unnecessary and removed.

SSH

Finally, we need a way to execute commands in the remote device in order to manage it in an effective manner. Historically, protocols based on unencrypted `TCP` connections were used, with `telnet` [47] being the most well known. Not providing encryption (nor authentication), however, proved to be shortsighted, and as the number of users with Internet access increased so did security attacks taking advantage of the technology.

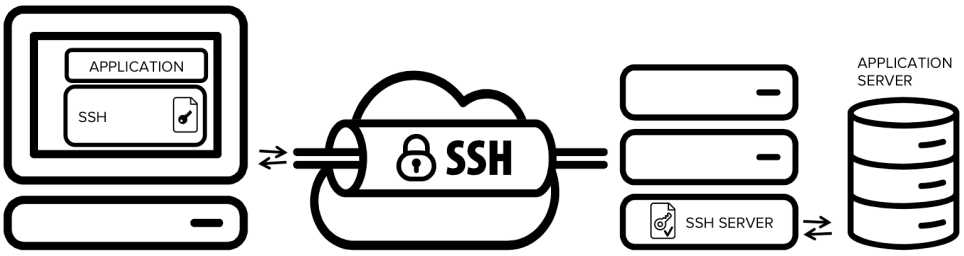


Figure 24: Two devices communicating communicating through an `SSH` tunnel [146]

Soon after, the Secure Shell (`SSH`) protocol [42] was born as means to provide access to remote terminals while ensuring authentication, confidentiality and integrity. By creating an end-to-end encrypted tunnel, traffic can neither be read nor tampered with by an external attacker. In addition to command execution, the flexibility of the `SSH` protocol allows for other uses such as tunneling of `X server` traffic and proxying.

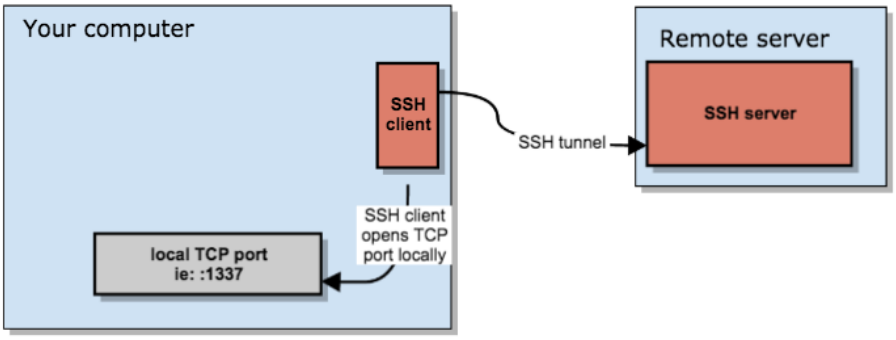


Figure 25: Redirection of traffic through an `SSH` tunnel [70]

Because the `OpenSSH` [128] implementation is bundled with nearly every Linux distributions we need only configure it to start using it. Although many security options exist, for our purposes forcing the user to verify via public key cryptography, setting `PasswordAuthentication no` and `PubkeyAuthentication yes`, on the `/etc/ssh/sshd_config` configuration file should be enough. Additionally, the router needs to be configured to allow connections to port on the device 22 by establishing a port mapping.

5.1.3 Initial network overview

Having considered what the software stack looks like and what functionality it should offer it is worth the time to describe what the network looks like and how it is configured. Although at this point it is rather simple, it will gain relevance in future sections as the complexity of the network increases

The following diagram depicts the relation between devices in the local network, excluding those deemed not relevant to the project:

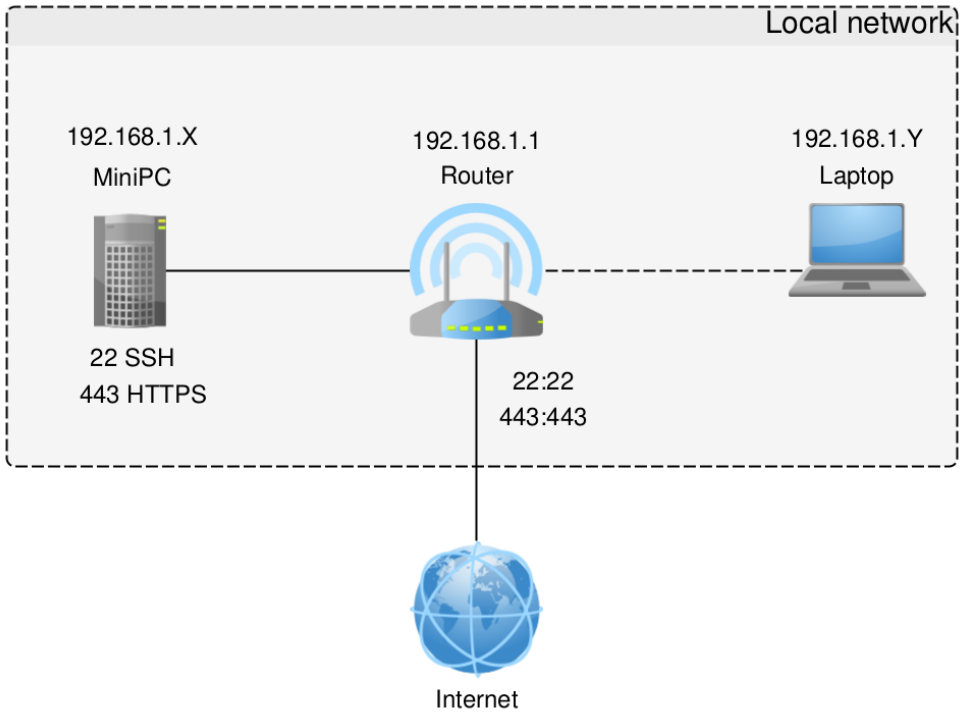


Figure 26: Diagram of the local network

In addition to the name describing the devices, the diagram includes their IPs (or placeholders if they were not remembered at the time of writing), the MiniPC's relevant port numbers together with the type of service provided, as well as the router's mapped ports. As it was explained in the previous section, the HTTPS port is used by **NGINX** to act as a reverse proxy for the other services.

Because this router was used only briefly, it will only appear in the present section, most of the configuration details are not relevant to the overall project. The only configurations worth mentioning are the mapping of ports 22 and 443 on the router to their counterparts on the MiniPC, in order to allow external traffic to be forwarded to the MiniPC, and the assignment of static IPs to the interfaces of the MiniPC, to ensure that the port mappings do not break after the device is restarted.

5.1.4 MiniPC monitoring integration

After having the whole stack set up and running, it was time to try it out with some real devices and data obtained from them. At that moment the simplest and most useful option was to start monitoring the MiniPC's resources, as no other devices where available yet and it would have to be done at some point either way.

The solution makes use of a `bash` script (A.1) deployed as a `systemd` service (A.2) that gathers device information from the operating system, which is then transmitted via `MQTT` and fetched by Node-RED. There, `JSON` is parsed into an object an then values are converted to floats and sent to the InfluxDB database. Finally, the values are split and sent to the corresponding visual elements in the dashboard.

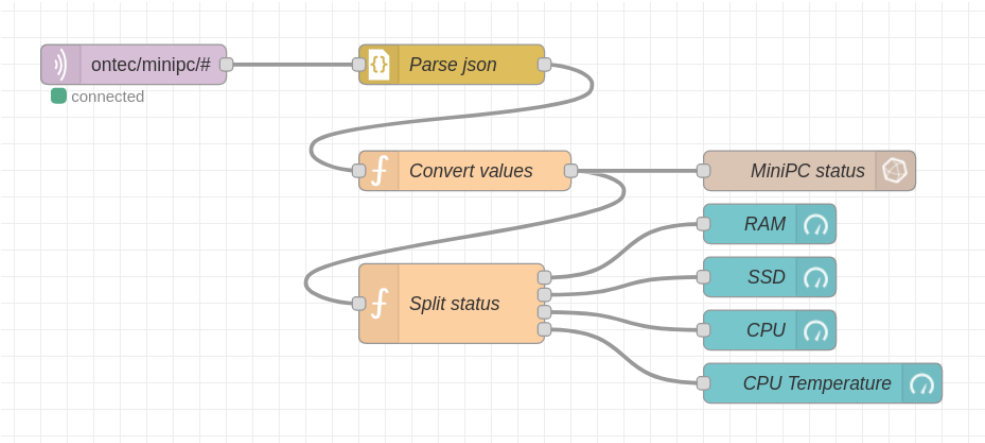


Figure 27: MiniPC monitoring Node-RED flow

```
var status = msg.payload;
status.ram_usage = parseFloat(status.ram_usage);
status.disk_usage = parseFloat(status.disk_usage);
status.cpu_usage = parseFloat(status.cpu_usage);
status.cpu_temp = parseFloat(status.cpu_temp);
return { payload: status };
```

Listing 7: JavaScript code for the `Convert values` node

```
var status = msg.payload;
var ram_usage = { payload: status.ram_usage };
var disk_usage = { payload: status.disk_usage };
var cpu_usage = { payload: status.cpu_usage };
var cpu_temp = { payload: status.cpu_temp };
return [ram_usage, disk_usage, cpu_usage, cpu_temp];
```

Listing 8: JavaScript code for the `Split status` node

Although the resulting flow and `JavaScript` code are arguably simple, the development process took some time, as there were multiple moving parts and they had to be properly tied together in order for it work. Most of the time was spent writing the script and then figuring out how to get the values to Node-RED and then InfluxDB.

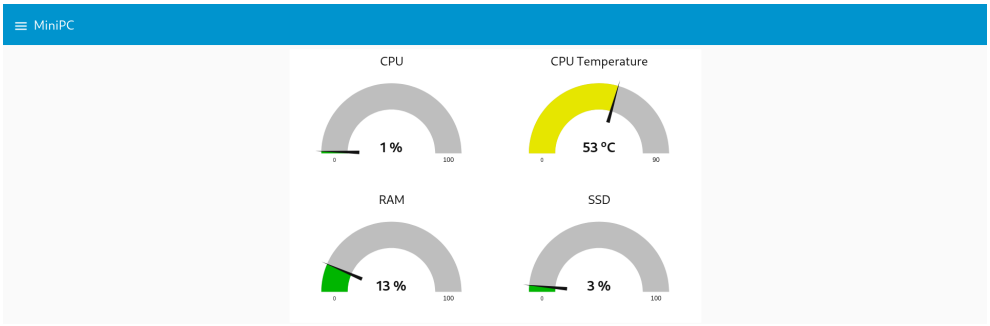


Figure 28: MiniPC monitoring Node-RED dashboard

After managing to get the monitoring data to the database, it was necessary to create a Grafana dashboard that would display the data in a more visual way than simple gauges. By comparison, this was quite simple, as once the data sources had been configured, the only remaining step was to create a panel for each of the values, configure it, and then write the corresponding database query.

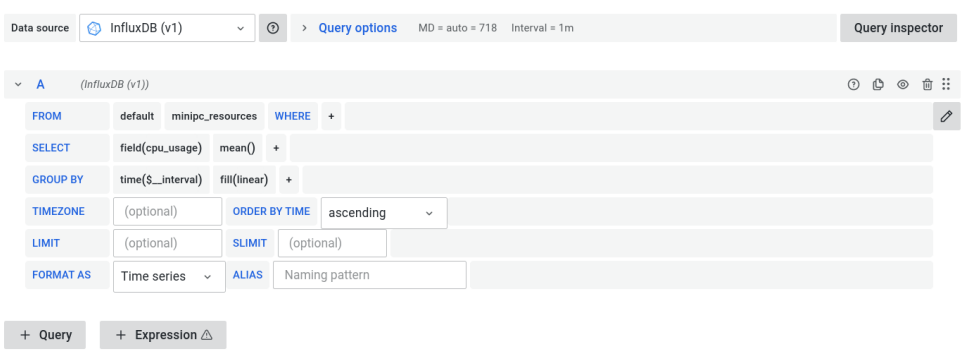


Figure 29: Query definition with Grafana's `InfluxQL` query builder

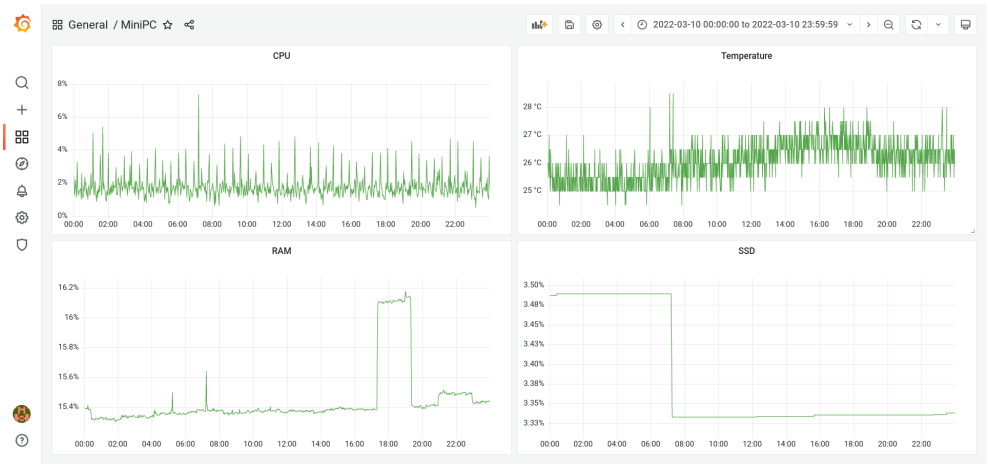


Figure 30: MiniPC monitoring Grafana dashboard

5.2 Main functionality

5.2.1 Solar inverter

After having performed the first integration test with the monitoring of the MiniPC, it was time to start thinking about implementing some of the more important functionality that would be required for the prototype, namely, that of integrating the required devices for the monitoring of a photovoltaic system. For this purpose a partnership was established with **Ontec Energy**, a company in the business of solar energy, which would provide the required installations and physical devices for the project.

Photovoltaic installations include some well known components such as solar panels, electrical cables or distribution boards, but, perhaps, a lesser known component is the solar inverter [46]. Usually hidden away from public view, an inverter is the component in charge of transforming the direct current output generated by solar panels into an alternating current, the type used by most consumer appliances.

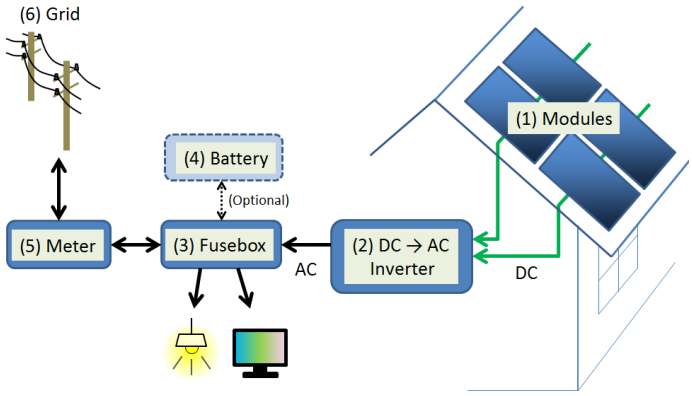
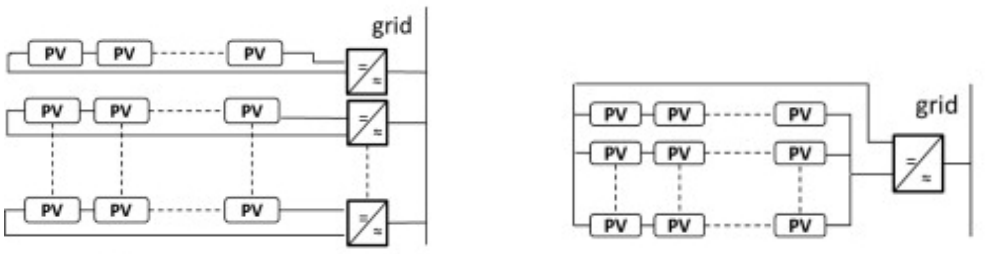


Figure 31: Grid-connected residential photovoltaic power system [46]

When it comes to solar inverters used in residential or medium-sized commercial systems two types of inverters dominate the market: string and central inverters [46]. In a nutshell, string inverters use a distributed architecture, which grants them high fault tolerance at the expense of of less power, while central inverters, as their name implies, work in a centralized fashion and produce larger amounts of power at a reduced cost [116].



(a) String solar inverter

(b) Central solar inverter

Figure 32: Schematics of the two most used types of inverters [53]

Autarco SX-MII

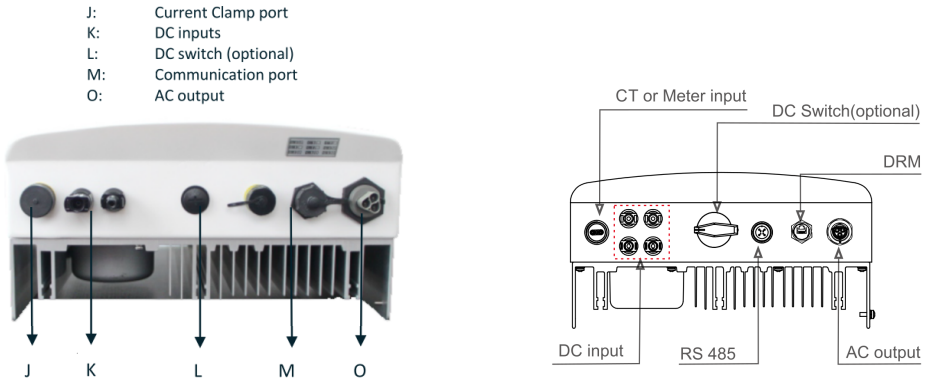
Although many alternatives exist in the market of solar inverters, **Ontec** uses **Autarco**, a manufacturer of solar equipment based on the Netherlands, as a provider for theirs. As a result, the one used for the project ended up being one of their products, more precisely, the **Autarco SX-MII** model, a monophasic string inverter [16].

(a) Front side view (b) Bottom side view

Figure 33: **Autarco SX-MII** solar inverter [16]

Although the inverter is very well built and offers strong guarantees in terms of solar production, the documentation offered by the provider in the form of a datasheet is quite lacking and at times even appears to intentionally omit details to prevent user from tampering with the device or customizing it in any form.

Luckily, after asking around for information, the inverter was found out to be based on another one by a different manufacturer, in particular the **S6 Mini** monophasic inverter from **Solis** [145]. While the documentation itself is not much more detailed, it does offer more precise descriptions of the user management and the different parts of the inverter, for instance, **Solis** marks an `RS485` port where **Autarco** simply makes a veiled reference to a communication port.



(a) **Autarco SX-MII** manual [16] (b) **Solis S6** manual [145]

Figure 34: Bottom side description from the two vendors

Inverter meter

By default most inverters are only concerned with converting the current from direct to alternating, so in order to provide data about measurements such as voltage or power another device is required, an inverter meter. Most meters measure current by attaching a circular clasp around the measured cable and analyzing the electromagnetic field generated by the flow of electricity.

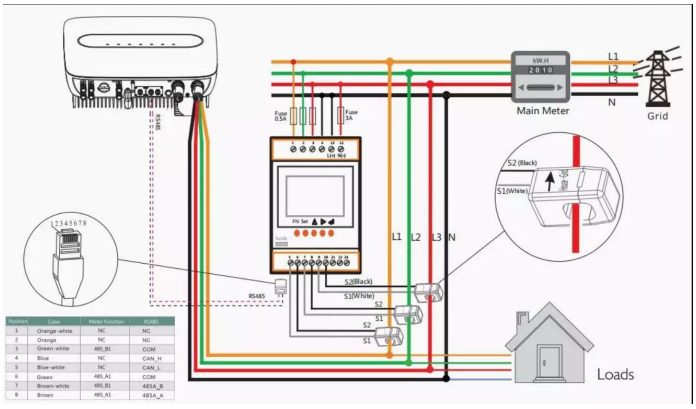


Figure 35: Installation diagram for **ACR10R** [1]

Initially, the **ACR10R** meter from **Acrel** [1] was to be used, as it is one of the few meters officially supported by the inverter. As per the installation instructions, the meter, which communicates using `RS485` [41], has to be connected to the leftmost port on the inverter (meter port), where it feeds the data it measures to the inverter so that it can be shown on the front panel display.

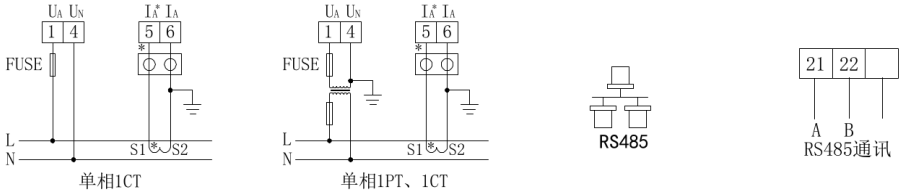
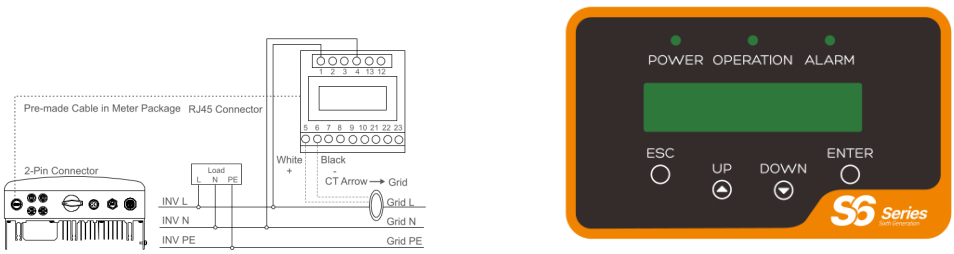


Figure 36: Diagram depicting individual connections on the **ACR10R** [1]



(a) Meter connected to `RS485` via `RJ45`

(b) Front panel display

Figure 37: Figures from the **Solis** inverter manual [145]

After contacting **Autarco** for information regarding the extraction of data sent by the meter directly from the inverter, the only thing provided was a short document containing the configuration of the `Modbus` [36] `RS485` line and a list of registers used.

Because the documentation provided little to no information on how to extract the collected data for integration with other devices, third party resources were used to figure things out. One particular article contained detailed instructions on reading data through the `RS485` port and then transmitting it with `MQTT` [52]. Although the original article is aimed at a **Solis** inverter, most relevant points should still hold for our particular inverter.

Modbus Protocol:

Baudrate:	9600bps
Parity checking:	None
Databits:	8
Stopbits:	1
	LSB/MSB

Data frame:

Slave Address	Function code	Data	CRC Check
8-Bits	8-Bits	Nx8-Bits	16-Bits

Slave Address: It is the corresponding slave address and must match the slave address of the inverter.

Reading data:

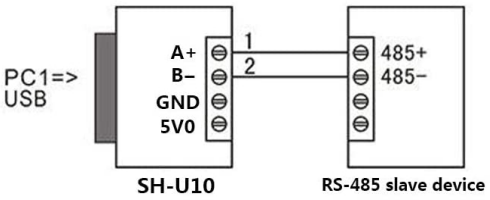
The function code is 0x04, the register address needs to offset one bit.

Example: register address: 3000, the send address is 2999.

Figure 38: **Modbus** information provided by **Autarco**

For the purpose of interacting with the inverter, an `RS485` to `USB` converter was attached to the communication port of the inverter and a `USB` port in the MiniPC, creating a serial device (`/dev/ttyUSB0`). In order to sent messages with the appropriate **Modbus** format, the `PyModbus` [64] python library was used.

(a) **SH-U10** converter



(b) Converter to `RS485` slave diagram

Figure 39: `RS485` to `USB` converter [5]

Despite trying different configurations, the serial device could not be read, and after some weeks were spent trying to make the **ACR10R** work and failing, it was decided to find an alternative solution to the problem. In the future, however, it might be worth giving it another shot, as later, some resources were found that could perhaps solve the observed issues [69].

(a) **ACR10R** meter [1] (b) **Shelly 3EM** meter [141]

Figure 40: Inverter meters used with the solar inverter

The alternative solution was to use an external meter that could directly communicate through `MQTT`, which would be easier to configure and most likely yield the same results. For this purpose, the `Shelly 3EM` [141] was used. Since the inverter was monophasic, a simple `Shelly EM` could have sufficed, but, in case measurements from more than one source were needed later on, the former was preferred.

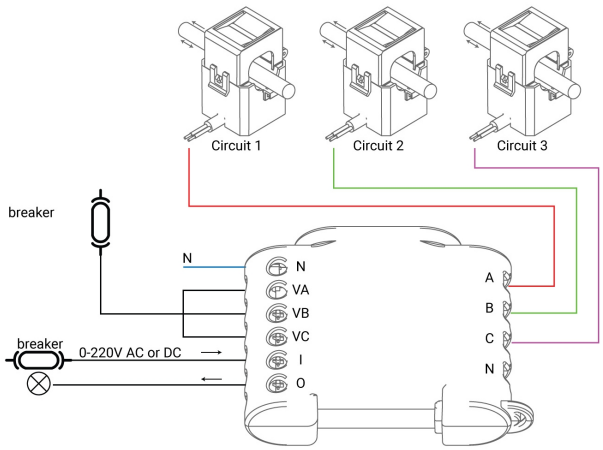


Figure 41: Installation diagram for **Shelly 3EM** [1]

Initially there were some issues getting the meter to work. The first problem was that the meter was being added to the `MOVISTAR_XXXX_PLUS` wireless network, a network operating on the 5GHz band, while the device only supports networks using the 2.4GHz, and so should have been connected to the `MOVISTAR_XXXX` network. After the correct network was used, however, the device did not seem to be accessible on the network for some hours, after which it appeared to magically fix itself, probably due to some caching on the router side.

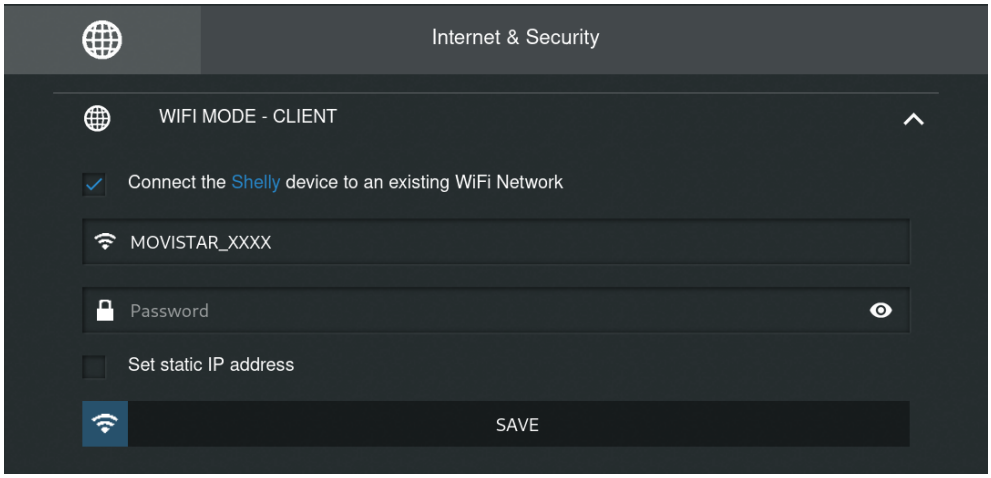


Figure 42: Network configuration for **Shelly 3EM**

Figure 43: **Shelly 3EM** web application displaying real-time data

Figure 44: Meters installed in a distribution board at Ontec

5.2.2 Backup system

At this point in the project it was time to set up a backup system that did not rely on manually copying files over from one machine to another. This would not only allow restoring the remote environment should anything happen to it, but also would grant the local development environment the ability to work with a previous version of the remote environment without need to tinker with the remote one.

Backup storage
With the advent of cloud computing [26] backups are rarely carried out manually or stored in a private machine, the same way most companies do not actually host their application or web servers on premises. Instead, there is a growing trend of using third party resources for storage, mostly renting it in a pay-as-you-go manner.

Nowadays many companies have emerged whose business model resides in the renting of cloud resources, but only a handful of them share significant market shared. In the last decades three big cloud platforms, owned by companies which were already leading the tech industry before, have emerged: **Amazon** Web Services (**AWS** [3]), **Google** Cloud Platform (**AWS** [80]) and **Microsoft** Azure (**Azure** [114]). Some other companies worth mentioning are **IBM** [90] **Oracle** [130] and **Digital Ocean** [54], which although used are nowhere near the other three.

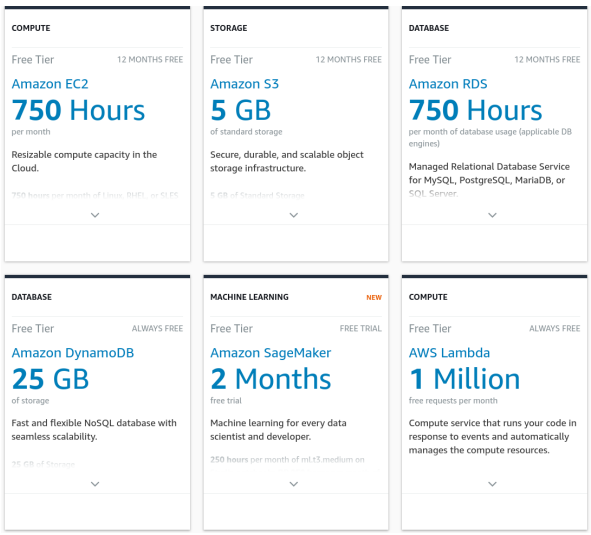


Figure 45: **Amazon AWS** free tier offerings [6]

Most of the cloud platforms mentioned above offer free tiers which include some limited amount of storage which can be used for any purpose. After some research, general consensus appeared to favor the big three platforms, not only on account of their popularity, but also because of their larger community and more extensive documentation. Out of those three **Azure** is not held in a very high regard by some, so it was discarded. Because **AWS** offered a good free tier and I had previously heard good things about it, that ended up being the final choice.

The service that will allow us to store files is called **S3**. Following the official documentation [7], we first need to create a bucket, a uniquely identified storage container that is replicated across multiple availability zones [8] to provide a reliable storage.

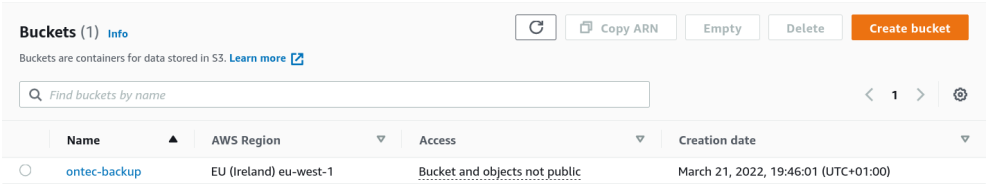


Figure 46: **S3** bucket that will store backups

After that we need a way to give applications access to the **S3** bucket without exposing all of **AWS**. The **IAM** [10] authentication mechanism allows the creation of groups providing access through policies and users belonging to said groups. Once the user is created, an access ID and key [9] are generated which can be passed to applications to authenticate as the given user.

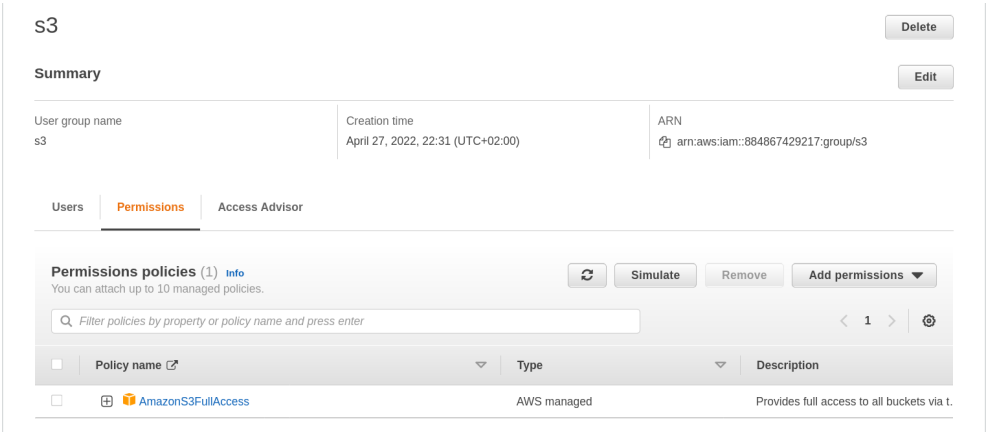


Figure 47: **AWS** group granting permissions to the **S3** service

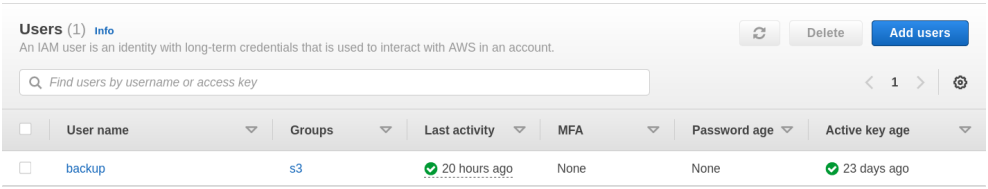


Figure 48: **AWS** user that will be used to access the **S3** service

After setting everything up we can test that everything is working as expected by installing the **AWS CLI** [11] and uploading a test file and listing the contents of the bucket. Profiles can be configured using `aws configure --profile`.

```
$ aws --profile ontec s3 cp test.txt s3://ontec-backup
upload: ./test.txt to s3://ontec-backup/test.txt
$ aws s3 ls s3://ontec-backup
2022-05-08 17:12:20  4 test.txt
```

Listing 9: Copying a file and listing of a bucket via the `AWS CLI`

Mail delivery

In addition to having a place in which to store backups, when implementing a backup system it is also wise to create some form of notification for the administrator to know the result of the backup. The most frequent form of notification is via mail, as it is ubiquitous and easier to implement than other alternatives.

`SMTP` [44] is the de facto protocol for electronic mail transmission. Oversimplifying many details of the protocol, the transmission process consists of two users (Mail User Agent `MUA`), the sender and receiver of the message, two termination servers, one used by the sender (Mail Submission Agent `MSA`) and one by the receiver (Mail Delivery Agent `MDA`), and one or more mail relay servers (Mail Transmission Agent `MTA`) between those two endpoints.

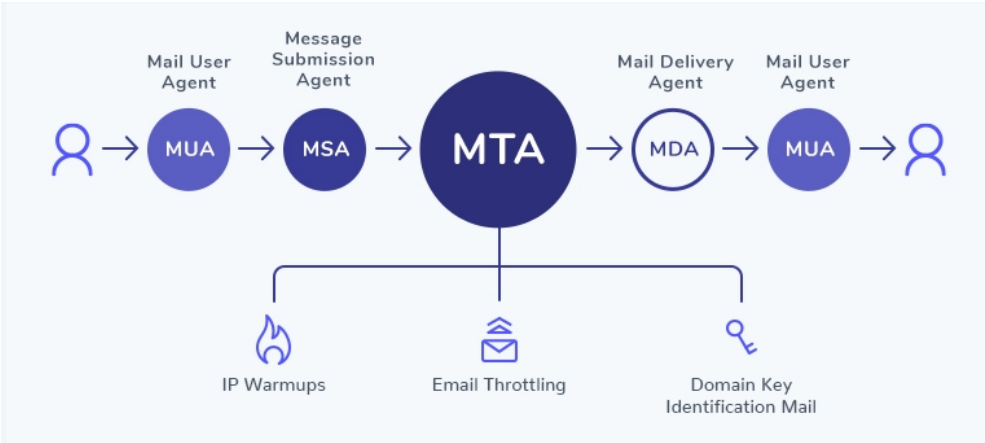
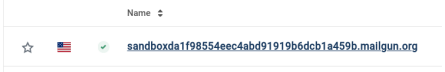


Figure 49: Diagram depicting a simplified view of the `SMTP` architecture [102]

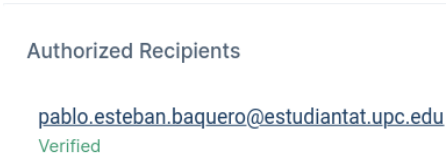
To send mails from an application, an `MTA` [35] server is required that will forward them to their destination. Although setting up a self-hosted mail relay is not that difficult by using existing solutions such as **Postfix** [154], it certainly requires a lot more work than simply using an already existing server [112, 15].

Many of cloud providers mentioned previously also offer mail relay services, such as **Google**'s `Gmail google:smtp` or **Amazon**'s `SES` [2]. Those services however, are either not provided for free or are somewhat complex to set up. After some research for free alternatives, **Mailgun** [111] was found to be a good candidate.

After creating an account for **Mailgun** we are granted access to a mail sandbox [144]. This essentially is a private DNS domain which we can access without the need to setup a DNS entries for a real one, with the restriction of only being able to send mails to 5 authorized parties.



(a) **Mailgun** private sandbox



(b) **Mailgun** authorized recipients

Figure 50: **Mailgun** service restrictions

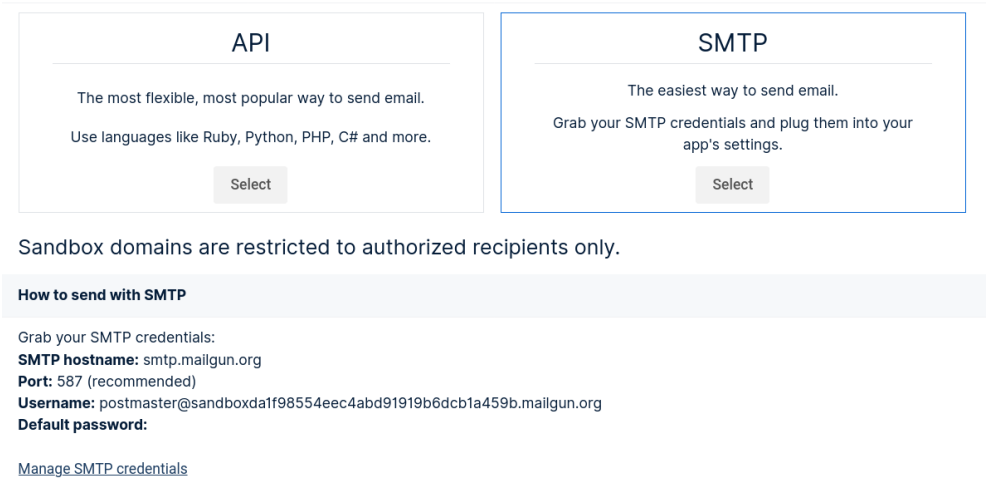


Figure 51: Instructions and credentials for sending a mail with SMTP

In order to test the newly created sandbox we can install `swaks` [97], an application for interacting with SMTP servers, and send a mail with it by using our Mailgun credentials, a valid mail account and the content and headers of the message.

```
sudo apt install swaks
swaks --auth                                    \
    --server smtp.mailgun.org                   \
    --au <USERNAME>                             \
    --ap <PASSWORD>                             \
    --to pablo.esteban.baquero@estudiantat.upc.edu  \
    --from 'test@test.com'                      \
    --h-From '"Test" <test@test.com>'           \
    --h-Subject 'Test mail'                     \
    --body 'Hello world'
```

Listing 10: Installation of `swaks` and sending of test mail

After sending the mail through `swaks` it should arrive at the **Mailgun** sandbox and be forwarded by MTA servers until it hit the **Google** mail server. If we check the inbox of the recipient we can see that it did indeed arrive and was not flagged as spam, as it came from a reputable source.

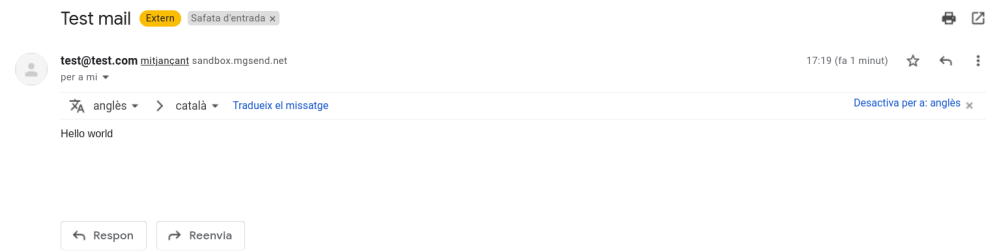


Figure 52: Mail received on the university Gmail account

5.2.3 Software overview

Having moved the MiniPC to a remote location meant that the software stack had to change to adapt to the new circumstances. Besides the setup of the backup system described in the previous section, there was now also a need to keep two separate environments, one for local development and one for deploying changes remotely.

This section will present the changes produced in three parts. First, changes concerning docker such as the deployment of containers through docker compose will be explained. Then, both local and development environments will be depicted in diagrams. Finally, newly added services will be described together with old services that were removed.

Docker overhaul

Before starting this project I had only used docker sporadically to test a certain software without having to install it locally or to run existing compose files. As project progressed, many questions arose, which required reading the official documentation as wells as solutions from other people.

After some time working on it, many changes were made to the initial compose file, and requiring a way to interact with different environments was the last straw. At that point, I decided to completely rework the compose file by replacing non-idiomatic sections and following the best practices that I knew of at the time.

The first thing that had to be done was devising some way of separating files related to the local and remote environments. Initially two independent compose files were created, one for each of environment, but that resulted in duplicated `YML` code, which made changes error-prone as many of them had to be reflected on both files. Eventually, searching for solutions to the problem lead to a feature of docker compose enabling sharing of configuration between files [59].

```
.-- docker-compose.override.yml -> env/compose/override.yml
|-- docker-compose.yml
|-- env -> environments/*
`-- environments
    |-- devel
    |   |-- compose -- override.yml
    |   |-- config --- *.env
    |   `-- scripts -- internal
    `-- prod
        |-- compose -- override.yml
        |-- config --- *.env
        `-- scripts -- internal
```

Listing 11: Project directory structure after the rework

The compose specification provides three major version numbers from 1 to 3, with the first one being deprecated. Versions 2 of the specification provides a somewhat cumbersome method, although quite flexible, of extending services from another compose file [59]. Version 3, on the other hand, provides a more simple override mechanism by which a second compose file (`docker-compose.override.yml`) is merged with the base file [59]. It is with this second method in mind that the directory structure was modified so that environments could be switched by means of a `symlink` to the corresponding override file. As a result, the compose version was changed to 3.

Although the override mechanism partially solves the issue of multiple environments, there still is an issue with updating the remote machine after changes to the services. Thankfully, docker's client-server architecture [57] plays nicely with our needs. By either setting the `DOCKER_HOST` environment variable or creating a docker context [55] we can specify a remote docker engine that will be used instead of the local one.

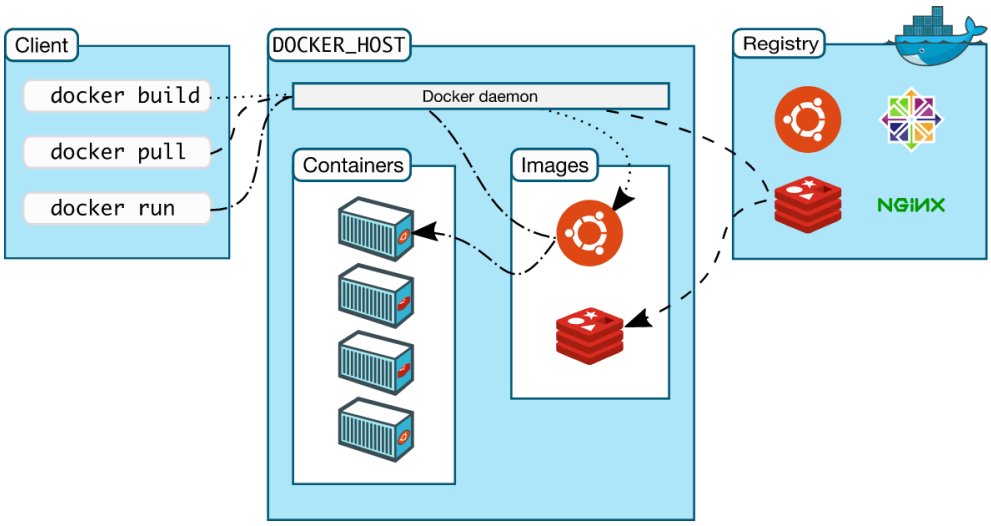
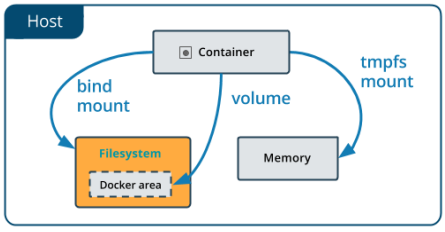
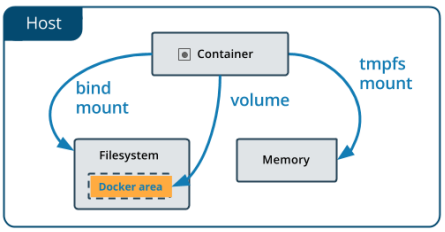


Figure 53: Diagram of the Docker client-server architecture [57]

Finally, there were various configuration changes made, perhaps the most significant one being the usage of named volumes [60] instead of bind mounts [61]. Numerous reasons exists why the first are preferred over the second [60, 17], but the strongest arguments are a better handling of permissions, a tighter integration with docker and portability. In order to migrate from bind mounts to named volumes (and some other extensions to the docker `CLI`) I wrote a small helper library.



(a) Bind mount [61]



(b) Named volume [60]

Figure 54: Comparison of the two methods exposing the host filesystem

In the development section of the appendix the resulting compose and configuration files for the local (B.3) and remote (B.4) environments can be found. Keep in mind that only the definitive version of each file is provided in the appendix to prevent cluttering it. As a result, some sections might mention features that did not make it to the files in the appendix.

Remote software environment

As it has been made apparent in the previous sections, there were various changes made in the software stack when the MiniPC was moved to a remote location. All of the core components of the system remain essentially the same, with only configurations being added or altered in some form. On the other hand, secondary services were both added and removed as the use cases and design changed.

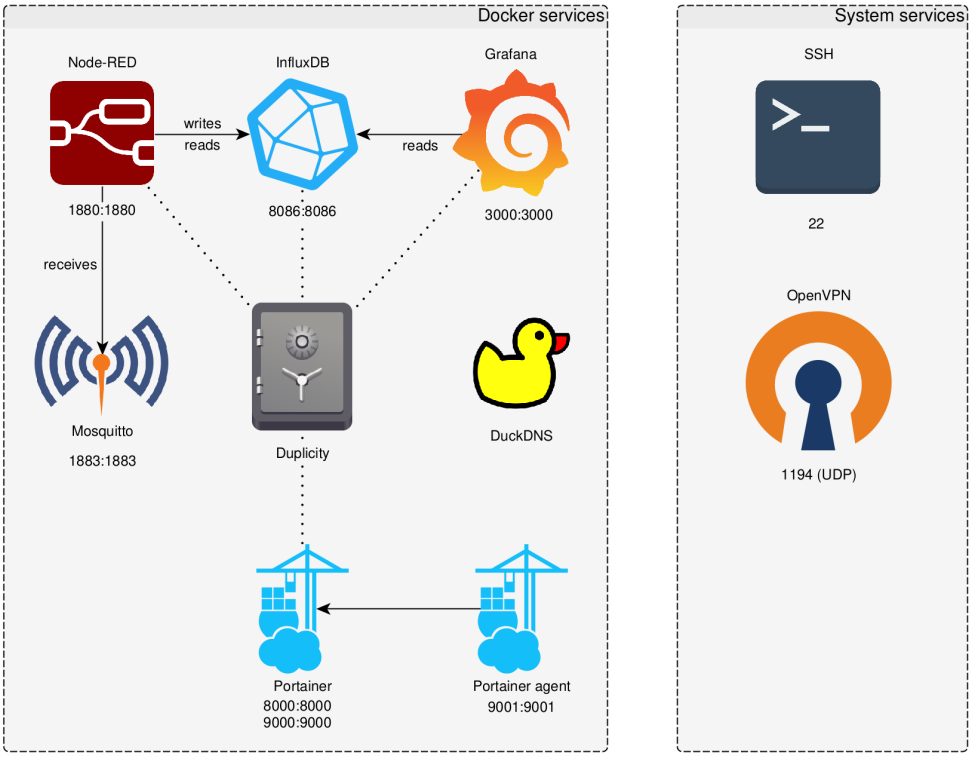


Figure 55: Diagram of the reworked remote software stack

First of all, the **NGINX** reverse proxy was removed, since it was not the best fit for the use cases that were in mind, as it will be explained later on. In its place, it was decided that **OpenVPN** would be used to interact with the remote network, providing access to the services hosted in the MiniPC and to other physical devices in the network.

In order to create backups **Duplicity** was introduced. Because most of the data to backup is found within docker volumes, it made sense to have it be a docker service as well. As hinted at previously, the remote environment is configured to create backups using **Amazon**'s S3 storage service and sending mail notifications through **Mailgun**.

Finally, after a meeting with the director it was decided that a potential user might want to interact with docker in a more visual way. **Portainer** is web application providing a graphical interface for docker, and an additional agent service provides information required by the application.

Local software environment
The new local environment was designed to match the remote environment as closely as possible while removing undesired side effects while testing locally. As a result, most of the services run on the remote environment are also present here.

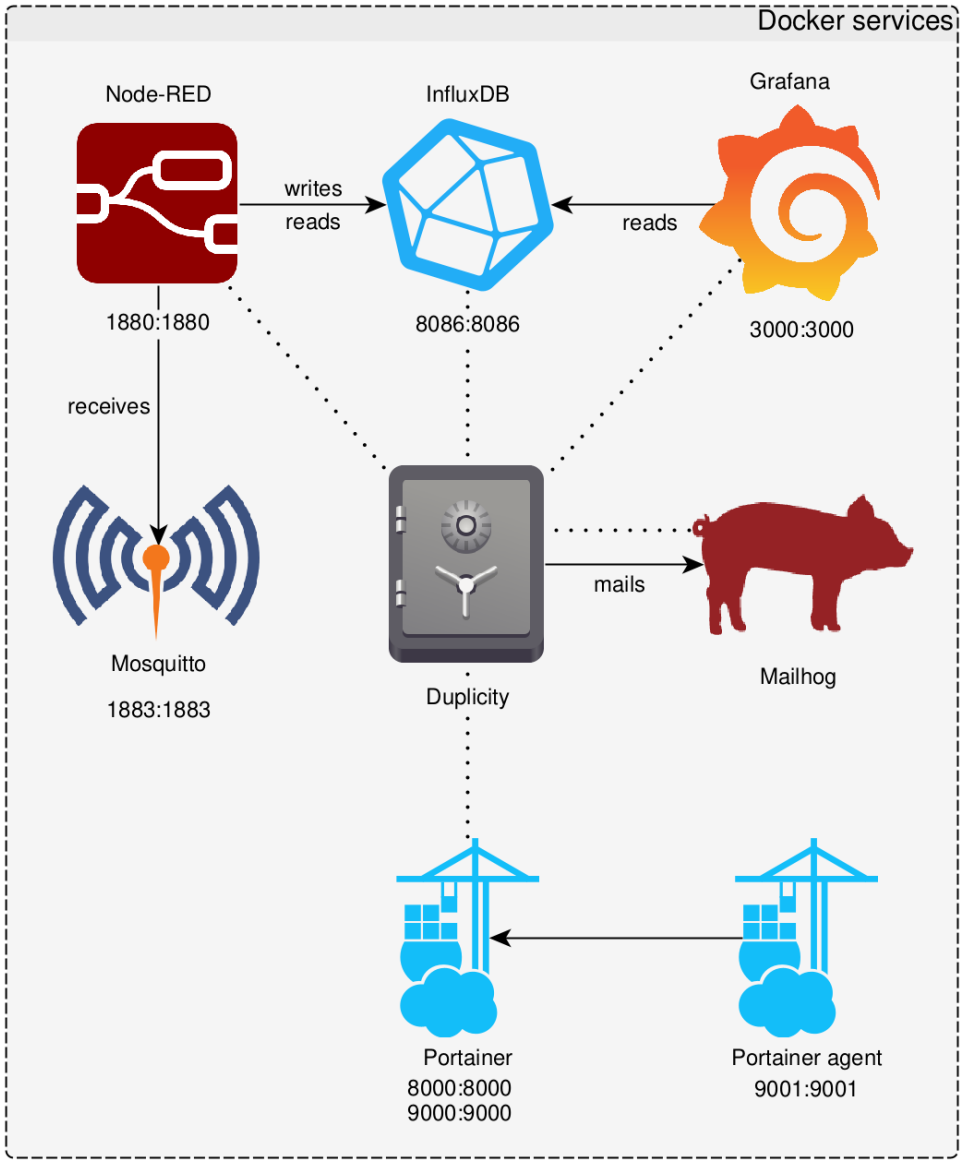


Figure 56: Diagram of the reworked local software stack

To provide a locally testable backup system that does not use external services **Duplicity** was configured to use a local volume as the destination for backups and a new local service called **Mailhog** that functions as a fake SMTP server.

Finally, the **DuckDNS** service was simply removed, as it is not needed for local development. In fact, using it would be harmful to the project, as both the local and remote machines would constantly fight over the DNS entries.

Duplicity

Earlier the storage and mail services that would be used for the backup system were described in some detail. However, without a particular backup software, the most important piece of the puzzle, nothing would actually be backed up, so the next step would be to find a piece of software that satisfied our needs while also fitting in the current software stack.

Traditionally backups were performed either manually, by copying files from one place to another, or by repurposing already existing software [148], but it was error-prone and not very user friendly. A well known tool for performing backups is **duplicity** [79], which is based on the original `rsync` utility [76, 72] and supports multiple storage backends including **S3**.

For this project we shall use a docker image by **Tecnativa** [78] that builds on top of it while providing a high degree of customizability through the definition of periodic jobs executed by `cron`. Also, since it is a docker image we can take advantage of named volumes to select the data to backup. Configuration for the service can be found on the appendix.

```
====================================
Job 200: `docker container exec influxdb influx backup
↪  /var/lib/influxdb2/backup`
Started: 2022-05-12 02:00:00.209485
Finished: 2022-05-12 02:00:03.046382
Success: True

2022/05/12 00:00:00 INFO: Downloading metadata snapshot
(...)
====================================
Job 300: `backup`
Started: 2022-05-12 02:00:03.046565
Finished: 2022-05-12 02:00:24.433660
Success: True

+ exec dup /mnt/backup/src boto3+s3://ontec-backup
(...)

--------------[ Backup Statistics ]--------------
(...)
TotalDestinationSizeChange 138691593 (132 MB)
Errors 0
-------------------------------------------------
```

Listing 12: Output generated by a daily backup

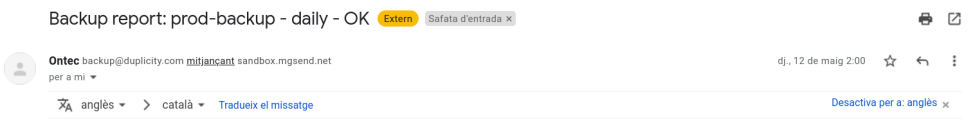


Figure 57: Mail generated by **Duplicity** on a daily backup

Mailhog

As it was mentioned earlier, during local development one might one to test an application that uses an `SMTP` server to send mails without sending the mail to its real destination. **Mailhog** [73], which has a docker image [85], provides that functionality as well as web UI and the possibility of forwarding mail to a real `SMTP` server.

In the project we configure **Duplicity** to use the service on the local environment, so that messages with the state of backups can be viewed and relevant information retrieved in case of failure.

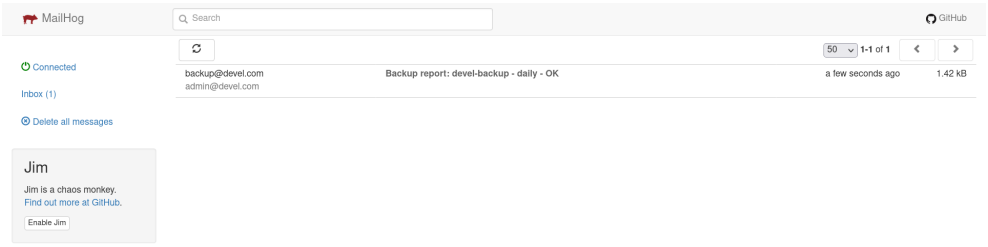


Figure 58: **Mailhog** showing a successful mail generated by **Duplicity**

Portainer

In order to allow users to view centralized info about the state of the services it was determined that a graphical UI to Docker would be necessary. **Portainer** [86] is a docker native web application providing a centralized monitoring dashboard, and some other features, which also works with multiple instances in swarm mode. Initially both the **Portainer** server [86] and agent [87] services were used, but as it will be explained later only the server was required.

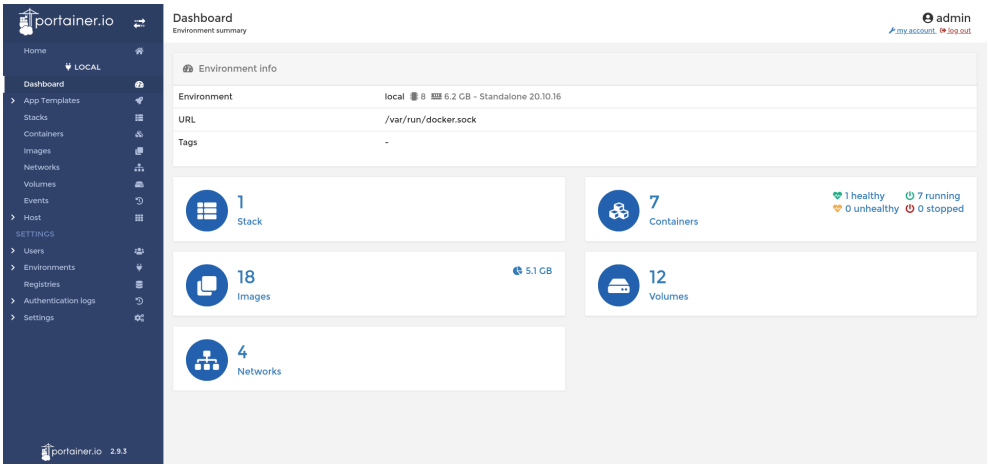


Figure 59: **Portainer** monitoring dashboard

OpenVPN

A VPN [49] enables remote users to send and receive data from a private network. Although at first it might appear similar in concept to remote access protocols such as SSH, the two serve quite different purposes. While SSH aims to grant a user access to a particular remote machine, VPNs allows the user to communicate with remote devices as if it were directly connected to their physical network.

Various protocols exist that offer VPN capabilities such as IPsec [31], but they are not easy to set up. For our purposes, **OpenVPN** [129] , an open source VPN client-server application, will be used. Since we will have direct access to the network, there will be no need to expose the services publicly, and so the **NGINX** service will be removed.

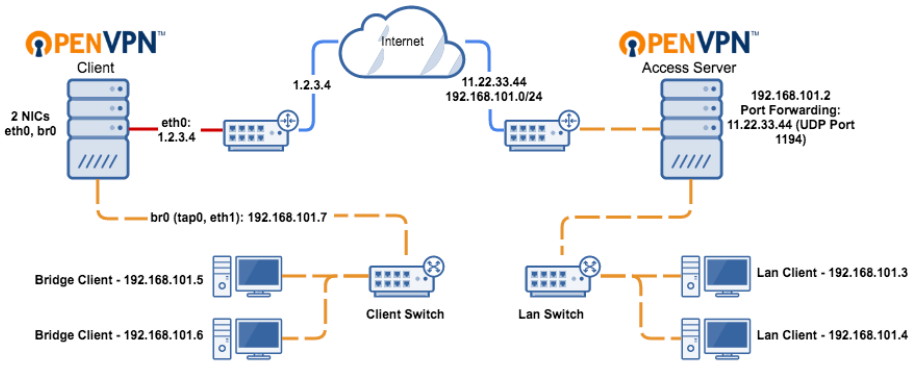


Figure 60: Diagram of two private networks connected through a VPN

Because the service can be considered critical, it is required to have access to the devices in the remote network, it shall be run as a system service outside of Docker, so that if the Docker daemon fails we retain VPN access. Installation and configuration of the service will be done through a helper script that guides the setup [75].

```
client
dev tun
proto udp
remote <REDACTED>.duckdns.org 19966
resolv-retry infinite
nobind
persist-key
persist-tun
remote-cert-tls server
auth SHA512
cipher AES-256-CBC
ignore-unknown-option block-outside-dns
block-outside-dns
verb 3
(...)
```

Listing 13: Client file generated by OpenVPN

5.2.4 Network overview

Having moved all the main devices to **Ontec**, there were now two principal networks, the remote one and the local network. The local network is used for development and to connect to devices in the remote network, while the remote network is used to deploy the system with real devices.

In addition to those two networks, the backup system also interacts with external services, so the network diagram depicts an approximation of the services and the actions that are performed by devices in relation to them. **Amazon S3** is represented as a service replicated accross different Availability Zones (AZ) and is acted upon in order to perform backups or restore them. **Mailgun** and **Gmail** services are also shown as part of an `SMTP` circuit, used to send mails with information of backups.

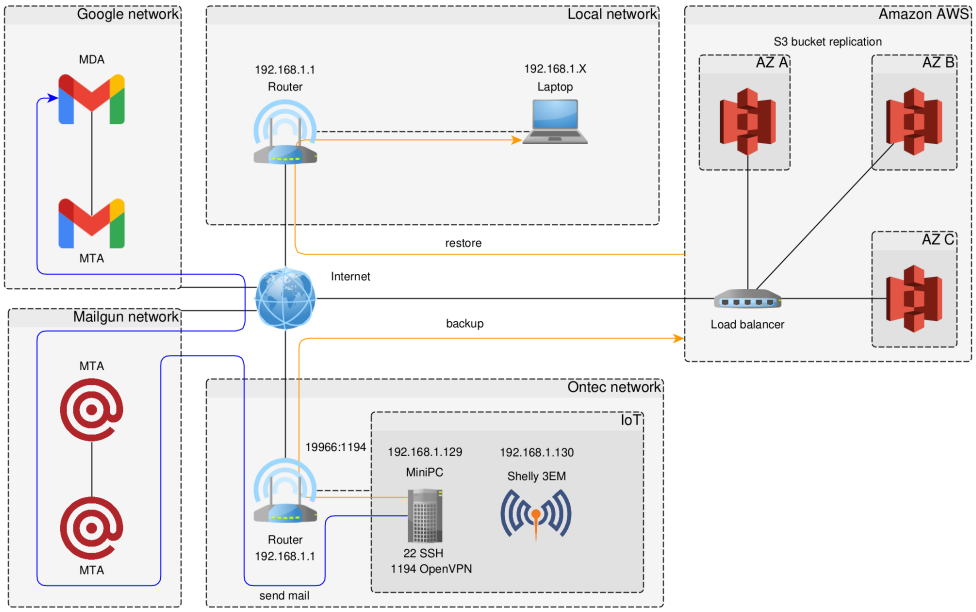


Figure 61: Diagram of the local and remote network

RTF8115VW router

Whenever changing environments, it usually is a good idea to get acquainted with differences before getting to work, and routers, specially consumer ones, vary quite a lot in quality and usability. The router at Ontec is a **Askey RTF8115VW**, one of the routers installed by the Spanish telecommunications provider **Movistar**. This particular model does not have very a good reputation [161, 12], as its web interface is quite restricted and not very responsive. Furthermore, information is very lacking, and is mostly obtained from forums dedicated to networking [162, 118] instead of official documentation, which seems to be nonexistent.

As it will be seen later on, many issues occurred in relation to the router. At some point acquiring a new neutral router or configuring the MiniPC to work as one was considered, but ultimately was not carried out due to time constraints and other factors. In the future, however, it will probably be done, as it would make reproducible installations possible and avoid issues any particular router might present.

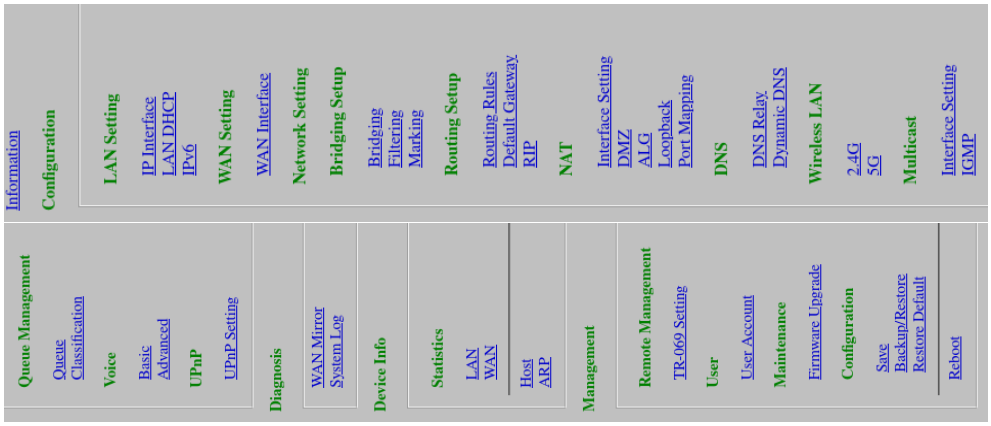


Figure 62: Router configuration menu

Router configuration

The configuration required at this points is similar to that of the previous local setup with the difference on different services being exposed to the outside world.

The first step was to configure static IPs for the new devices, so that even after being reset they would remain the same and no applications required any reconfiguration. The router defines 4 `DHCP` pools for different purposes, and each one offers the possibility to configure the range of addresses in the pool and static leases among others things. The default pool for home devices is number 3, and the rest are reserved for specific **Movistar** services [118].

To avoid having to configure a subnet, the IP subrange `192.168.1.128/25` is chosen for all the IoT devices, and because not many devices exist in the network only devices assigned through static leases should end up in it. Both the MiniPC's wireless card `MAC` and the Shelly 3EM's default `MAC` are given an IP in the range.

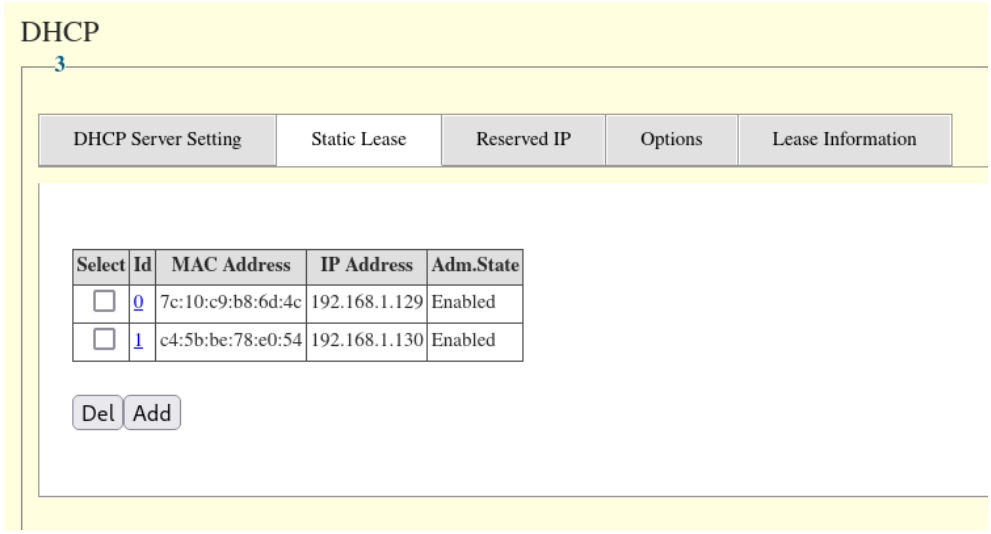


Figure 63: Configuration of static `DHCP` entries

Using the previous IP assignment, a port mapping rule is created to forward traffic coming to UDP port `19966` to the MiniPC's OpenVPN (`1194`) UDP port. The configuration for the particular router requires the selection of an interface from which to forward traffic, which in our case is the `PPPoE` interface.

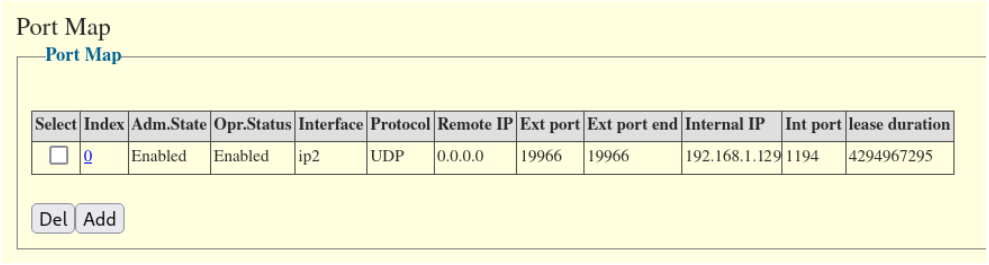


Figure 64: Configuration of port mappings

MiniPC network service

During this stage of the development there was an issue with the MiniPC related to networking. Essentially, the machine got stuck waiting for a `systemd` service for a few minutes after booting if no available network was found.

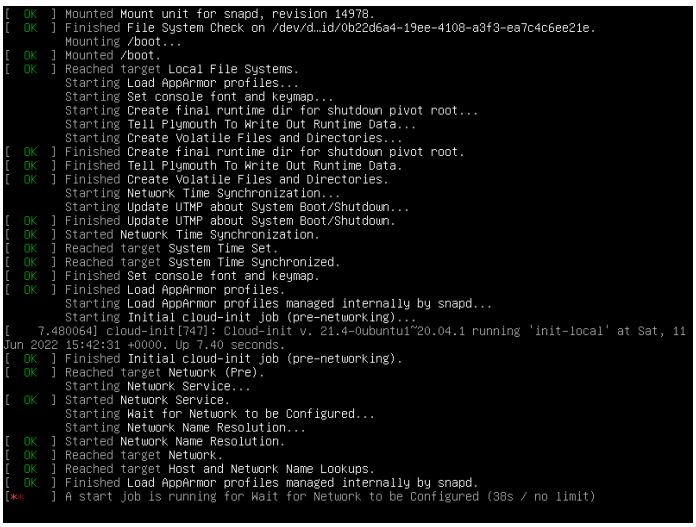


Figure 65: Ubuntu `systemd` waiting for network (recreated on **VirtualBox**)

After looking for similar issues online, the problem was found out to be caused by [18], a utility for network configuration created by **Canonical**, which is installed in Ubuntu server by default. The solution to the issue was to modify one of the configuration files (`/etc/netplan/00-installer-config.yaml`) to indicate that some devices might not be available by setting `optional: true` on their properties [152].

5.2.5 Shelly 3EM integration

Once the devices were set up and the remote network configured, the next thing to be done was to integrate the meter into the existing system to collect information sent by it. The resulting integration ended up being pretty similar to the initial MiniPC save the way data was obtained from `MQTT` and processed. Information regarding the format of messages was obtained from the official API documentation [143].

Because the Shelly 3EM is designed to send messages for each one of the three phases, the first step is to filter the one in use. Additionally, each measurement within a phase is sent in a different message, so data is put together in groups of 7, the number of measurements. After that, the message's topic is set to the value that will be written to the database and, using the same idea as in the first integration, relevant measurements are then split and sent to the dashboard.

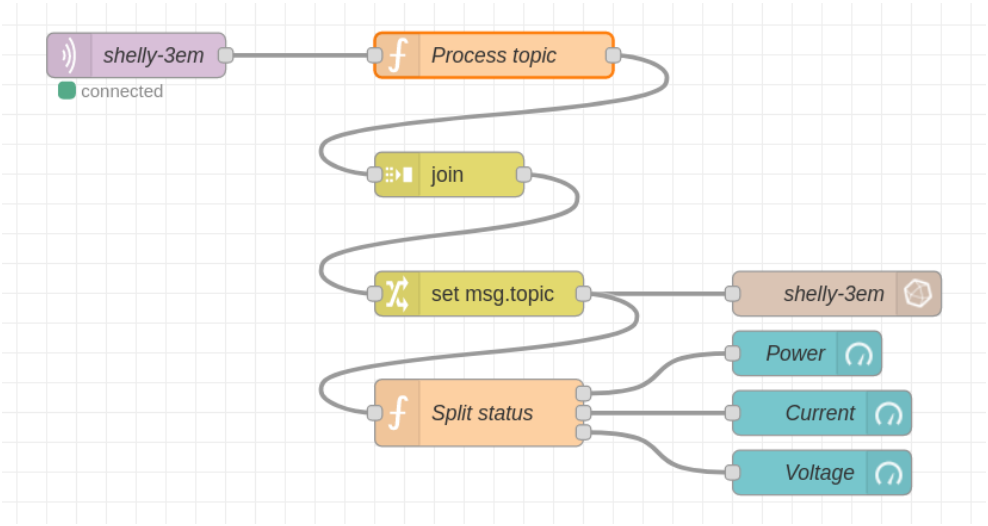


Figure 66: Shelly 3EM Node-RED flow

```
// topic := shellies/shellyem3-<deviceid>/emeter/<i>/<data>
var topic = msg.topic.split("/")
if (topic[4] !== undefined && topic[3] == 0) {
    msg.key = topic[4]
    msg.payload = Number(msg.payload)
    return msg;
}
```

Listing 14: JavaScript code for the `Process topic` node

```
var status = msg.payload;
var power = { payload: status.power };
var current = { payload: status.current };
var voltage = { payload: status.voltage };
return [power, current, voltage];
```

Listing 15: JavaScript code for the `Split status` node

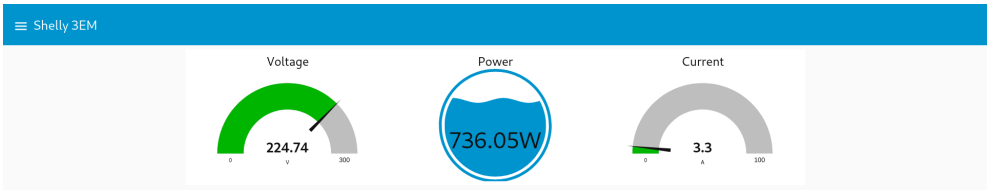


Figure 67: Shelly 3EM Node-RED dashboard

Again, after getting all the measurements into the database, a Grafana dashboard was created with 4 panels depicting some of the important ones via InfluxQL. Although some measurements such as the total energy are not that relevant to the final user, this is what could be obtained through raw data alone; some processed measurements will be considered later on.

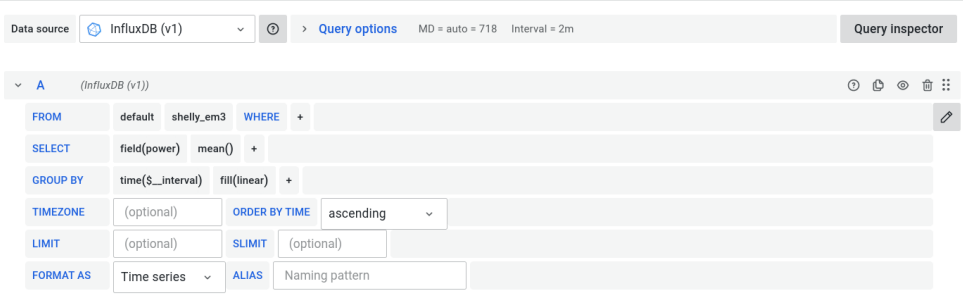


Figure 68: Query definition with Grafana's `InfluxQL` query builder

With the obtained data we can clearly visualize how, as one would expect, the amount of generated current and power is determined by the time of the day, forming a bell with the highest point centered around 13:00 and lower values moving outwards.

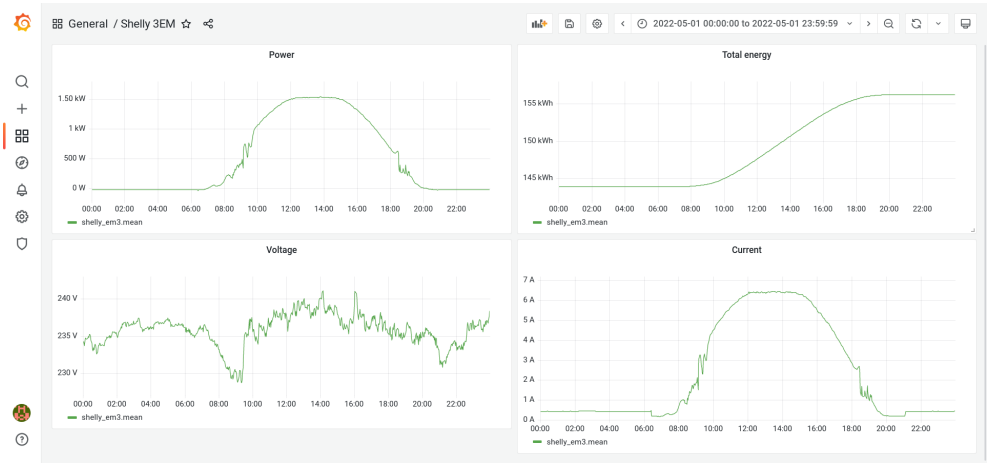


Figure 69: Shelly 3EM Grafana dashboard

5.3 Final additions

5.3.1 Active devices

The next challenge after the integration of the inverter was to allow the management of excess with other devices as well as testing and polishing the system. However, due to the previously mentioned issues with meters and some issues faced after the installation of these devices and network rework, not much time was left to complete the initially designated tasks, as the project was nearing the end of its duration.

Shelly Plug S
The first device acquired was the **Shelly Plug S** [142], a smart plug that can be controlled and monitored remotely with some additional capabilities. For this final stage, three of them were bought and installed on electric outlets, which would be used to perform simple tests with manual and scheduled management operations.

(a) **Shelly Plug S** power outlet [142]

(b) **Shelly Plug S** installed at Ontec

Figure 70: Inverter meters used with the solar inverter

Configuration for the device is the same as for other **Shelly** products, such as the previously seen **Shelly 3EM**, and the same can be said for its web interface.

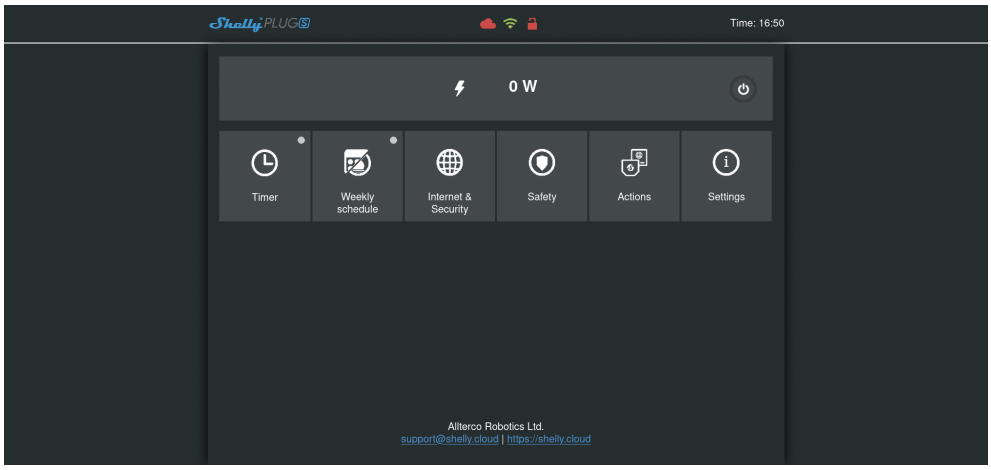
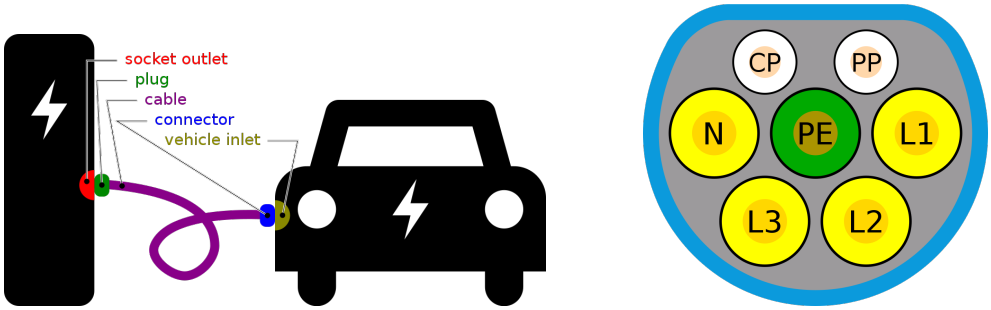


Figure 71: **Shelly Plug** web application displaying real-time data

OpenEVSE

With the recent surge in popularity of electric cars, it would probably be interesting to consider adding integration support for devices such as charging stations [25]. Although initially limited in scope to specialized vendors, with the emergence of consumer stations and open source solutions, more and more electric car owners have taken to installing them at their own home.

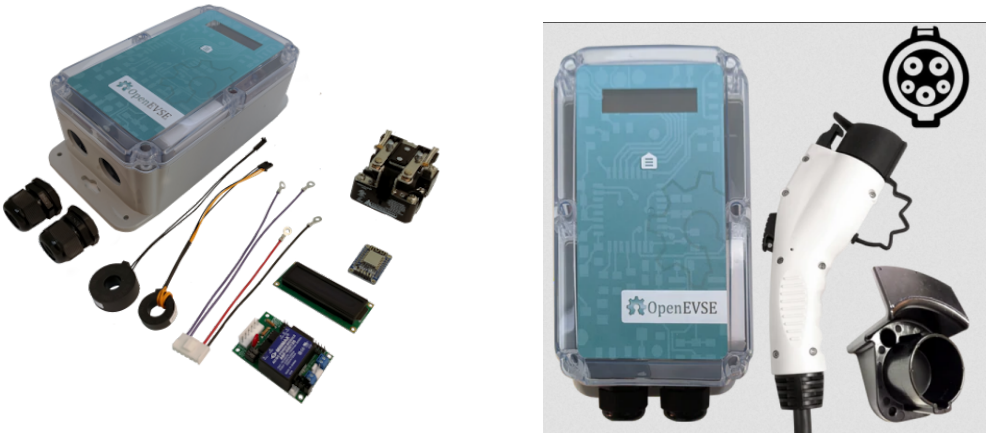


(a) Components in a charging station

(b) European triphasic connector

Figure 72: Diagrams of electric vehicle supply equipment (EVSE)

For the purpose of the project **OpenEVSE** [127], an open source software and hardware for charging stations, was considered. Besides technical assistance and installation, **OpenEVSE** offers various DIY kits, of which the advanced bundle, containing al required components, was acquired.



(a) **OpenEVSE** DIY kit

(b) **OpenEVSE** assembled station

Figure 73: **OpenEVSE** charging station [127]

However, as mentioned earlier, not much time was left at this time of the project, and because not much was known about the installation and integration process it was too late to actually set it up. Nevertheless, some time was dedicated to studying its architecture and reading up on documentation, as it will certainly be the next step in the project.

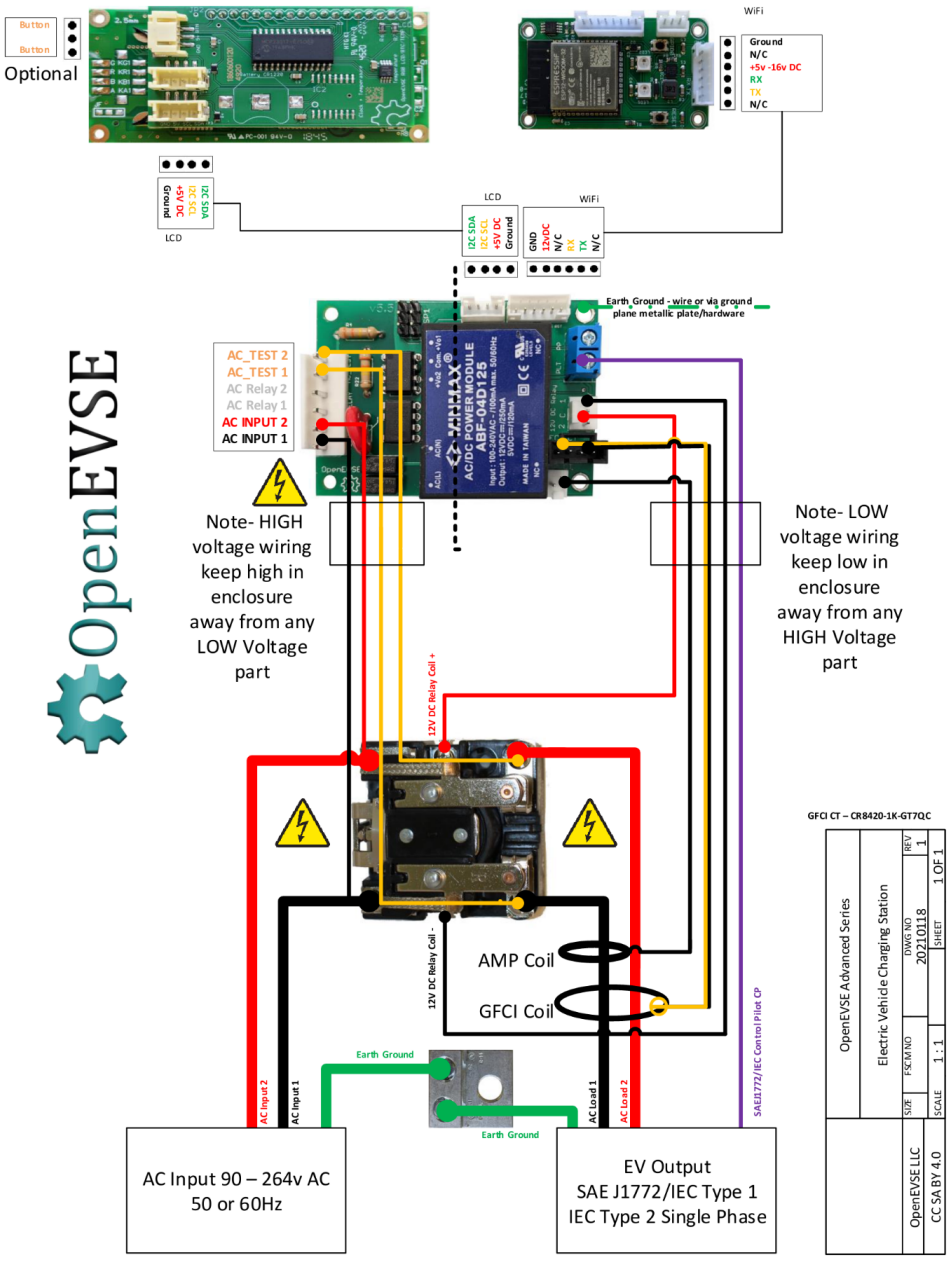


Figure 74: **OpenEVSE** assembling diagram [74]

5.3.2 Final software overview

Remote software environment
In this final stage of the project not much functionality was added in terms of the software stack save for a few changes. Since the system was working as expected most of the effort went to the making changes to the network and writing documentation.

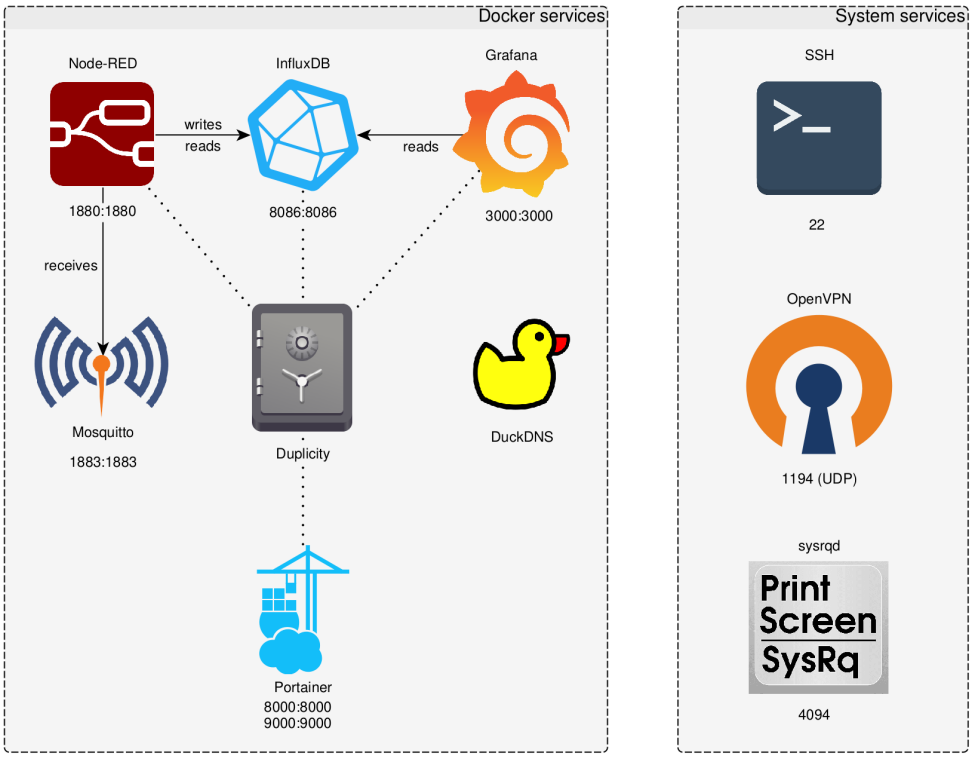


Figure 75: Diagram of the final remote software stack

One of the only changes made to the remote environment was the removal of the **Portainer** agent docker service [87]. Initially both the server and agent services were thought to be necessary in order to run the monitoring software, but after a closer look at the documentation [134] it was found out that the service agent was only required for docker instances running in `swarm` mode, which apparently requires special access to the underlying node resources.

As it will be explained later on, while making changes to the network some issues arose that made it impossible to work without restarting the machine and, at the same time, to connect remotely using the usual `SSH` service. To solve the issue the `sysrqd` [77], which did allow to reboot the system even in those conditions, was installed. Because it is a critical service that should work at all times it was installed as a system service.

Local software environment
The local software environment also received minimal changes only. One being the modification of some of the configuration files, and the other one the removal of the **Portainer** service, as it was only really used in the remote environment.

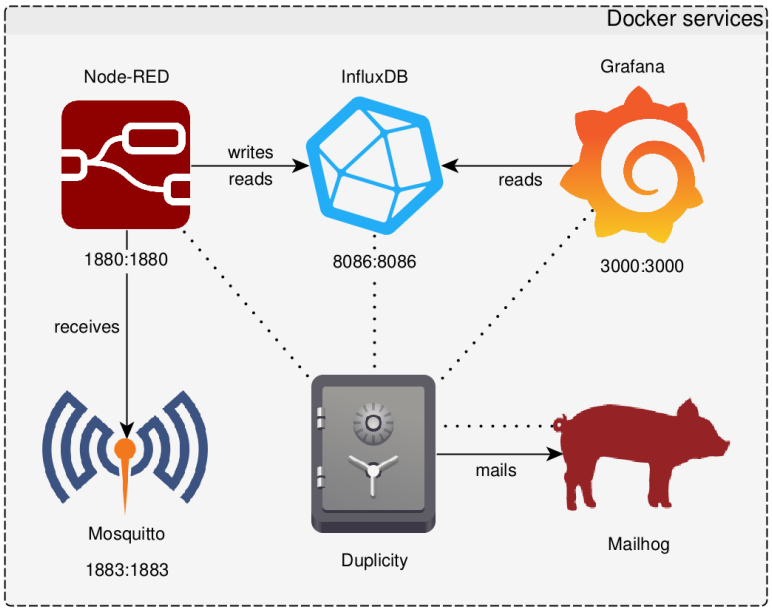


Figure 76: Diagram of the final local software stack

Improved backup
Keeping an eye on the backup system and routinely testing it to ensure that everything works as intended is as important as setting it up in the first place.

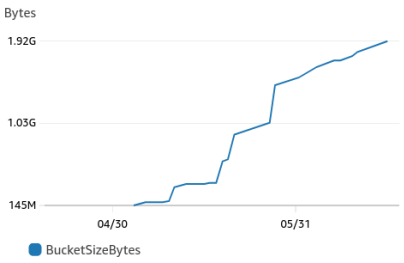


Figure 77: **CloudWatch** [4] dashboard monitoring bucket size

After some time doing backups the size of each backup appeared to be increasing more than it should have, and looking at data provided by Amazon corroborated the observation. After looking around in the docker **Duplicity** container it was found out that after backing up the **InfluxDB** database generated files were not deleted, and so the size kept on increasing. The solution was to create a new Duplicity job that deleted previous backup files before generating new ones.

Another improvement that was in the works but not fully implemented was that of backing up only specific files instead of entire docker volumes. For instance, saving **Node-RED** `json` configuration files only, and discarding NodeJS modules in the same volume. Some scripts were created for the purpose (A.5, A.6) but were not integrated into the backup/restore process.

5.3.3 Final network overview

On this last stage of the project only **Shelly Plug S** smart plugs were added to the network, as not enough time was left to assemble and test the **OpenEVSE** charging station. As with the previous sections, the idea was not so much about adding functionality as it was about improving the existing one.

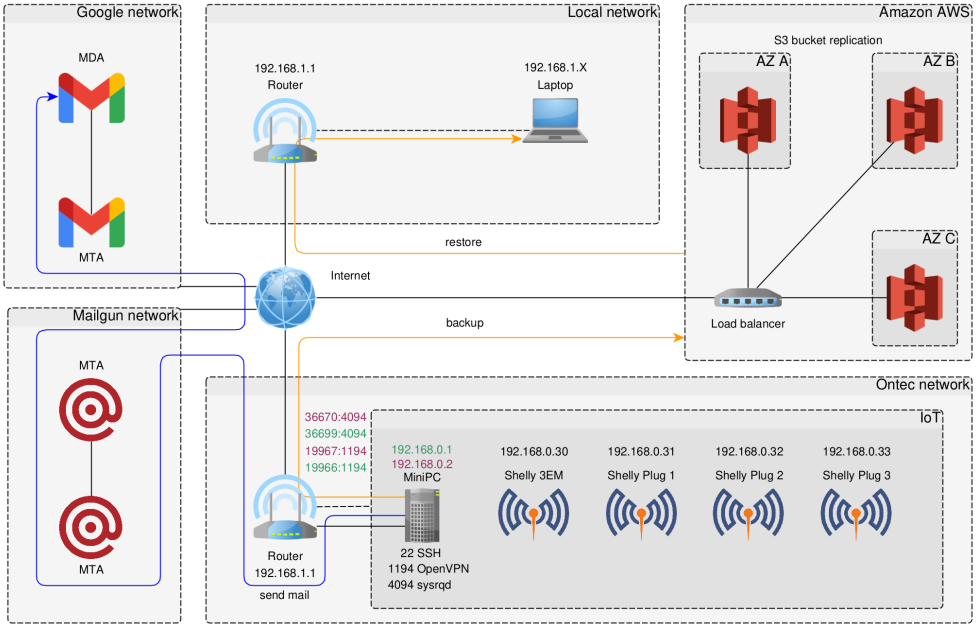


Figure 78: Diagram of the final network

In the case of the network, the planned improvements were the rework of the remote network topology by creating a dedicated `IoT` subnetwork which was isolated from the rest of devices in the Ontec network through firewall rules as well as the creation of a wireless network on through which `IoT` devices could communicate without interferences with devices outside of their subnetwork.

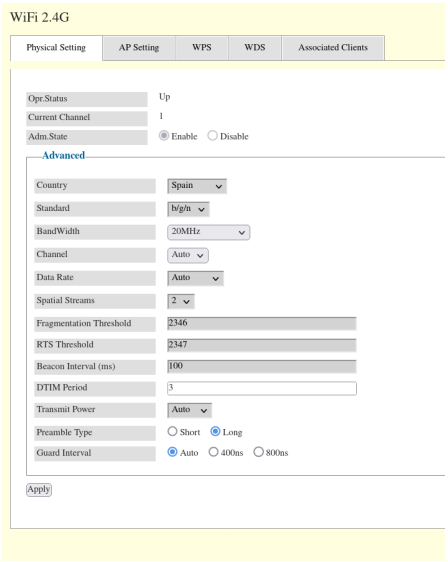
Reality, however, is not as pretty as one would like, and router limitations, hardware issues and other problems did not allow to perform all of the desired changes and ended up requiring a simplification of the requirements stated above.

Access point configuration

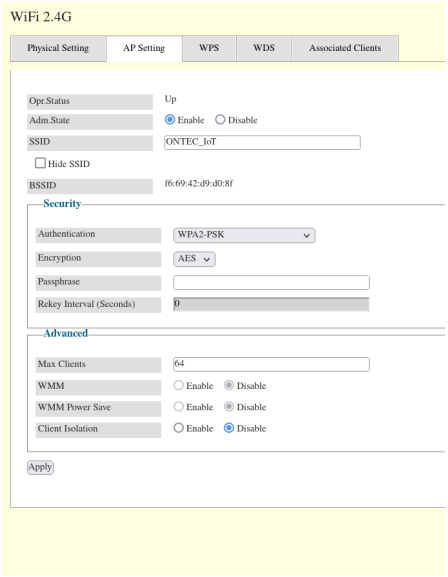
The first configuration change attempted was the creation of a wireless network dedicated to devices that would belong to the `IoT` network. This was done in an attempt to prevent collisions between normal devices and `IoT` devices, which although not noticeable with the current amount, could become a problem as the number grew.

Because of the way wireless local area networks work, multiple groups, known as service sets [43], can be configured to work on the same area without any interferences through the use of different frequency channels [34] defined for use with wireless networks. In practice, these different service sets can be configured in routers by using different `SSID`. In our case the configuration was done in the `2.4G` band, as it is the one used by **Shelly** devices.

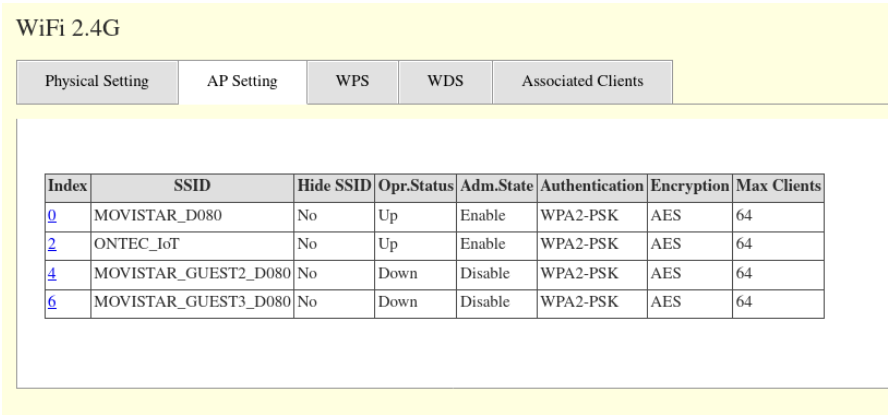
By default, the router offers 4 different wireless networks, the default one and 3 guest networks. After configuring one of the guest networks and connecting some devices it was observed that they could communicate with the Internet but not among themselves, which was mandatory to allow integrating them with Node-RED. At first it appeared to be caused by AP isolation [109, 108], a mechanism used in guest untrusted networks to improve security, but even after disabling it the problem persisted.



(a) Configuration of the 2.4G band



(b) Configuration of the **IoT** wireless network



Index	SSID	Hide SSID	Opr.Status	Adm.State	Authentication	Encryption	Max Clients
0	MOVISTAR_D080	No	Up	Enable	WPA2-PSK	AES	64
2	ONTEC_IoT	No	Up	Enable	WPA2-PSK	AES	64
4	MOVISTAR_GUEST2_D080	No	Down	Disable	WPA2-PSK	AES	64
6	MOVISTAR_GUEST3_D080	No	Down	Disable	WPA2-PSK	AES	64

(c) List of available wireless networks

Figure 79: **RTF8115VW** access point configuration

After searching around not much information was found, but because of previous issues with the router, it was assumed that it probably had to do with the router hardcoding some default values for guest networks. It would also make sense, as for some time the web interface did not even allow to save the disabled option for client isolation. In the end, the default network was used for all devices.

Subnet configuration

Previously the two devices in the `IoT` network had been placed in the `192.168.1.128/25` subrange, but it was considered that it would be cleaner to create a completely separate subnetwork, `192.168.0.0/24` for instance, and then create firewall rules that kept the original network `192.168.1.0/24` apart from the rest of `IoT` devices.

However, once again, the tight router settings did not allow for an easy solution, as the firewall could not be configured from the web interface and no way was found to assign different IPs to the router interfaces. Additionally, manual IP assignment on **Shelly** devices did not work, so the only way left of configuring them was by using the `DHCP` server on the router.

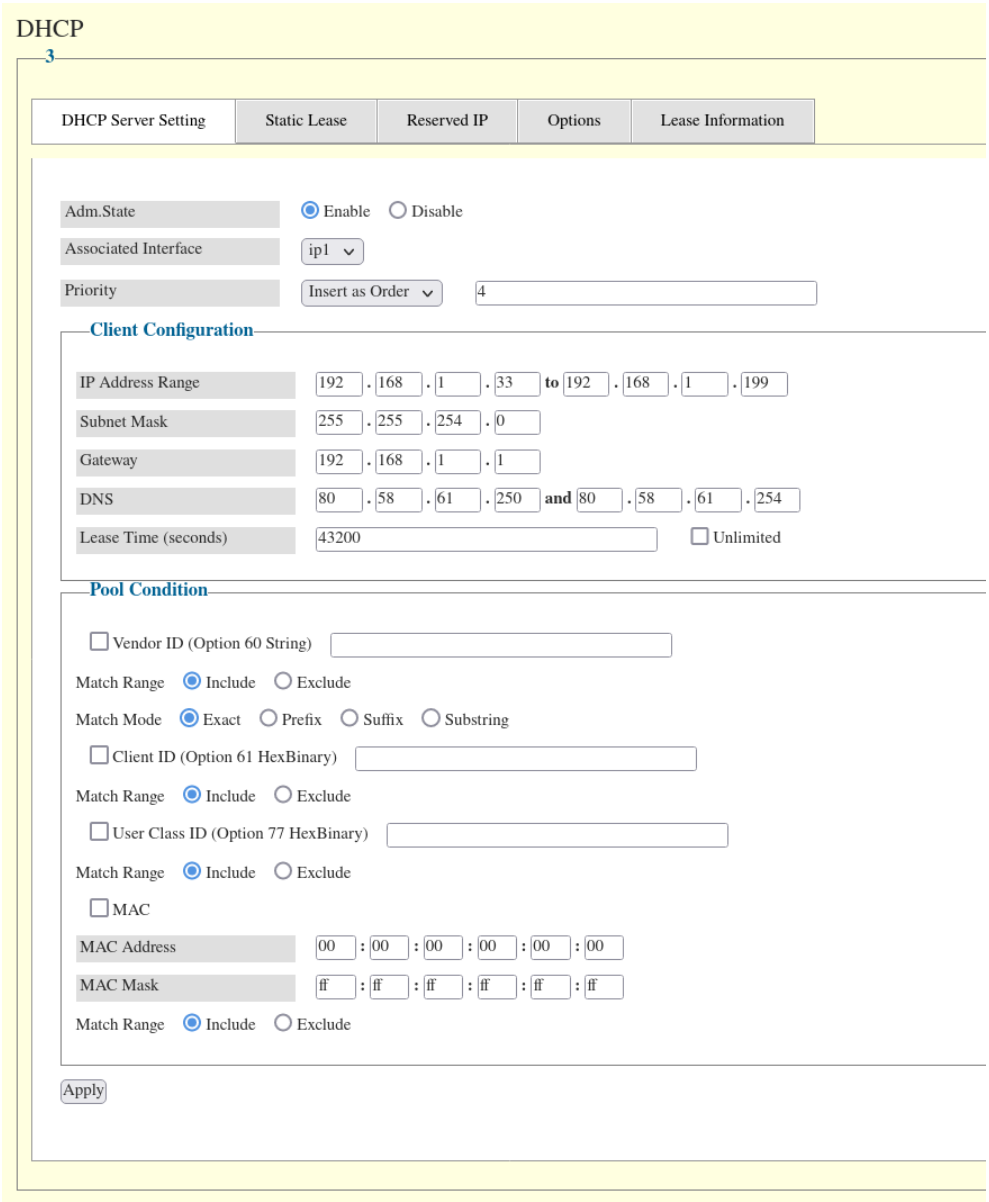


Figure 80: `DHCP` configuration menu

Again, there was probably some way of achieving the desired solution, for example the configuration of the MiniPC as a secondary router for the subnetwork, but because not much time could be spared on finding the appropriate setting, an easier solution was chosen. Instead of creating two separate networks the main network was modified to take `255.255.254.0` as the subnet, creating the larger `192.168.0.0/23` range.

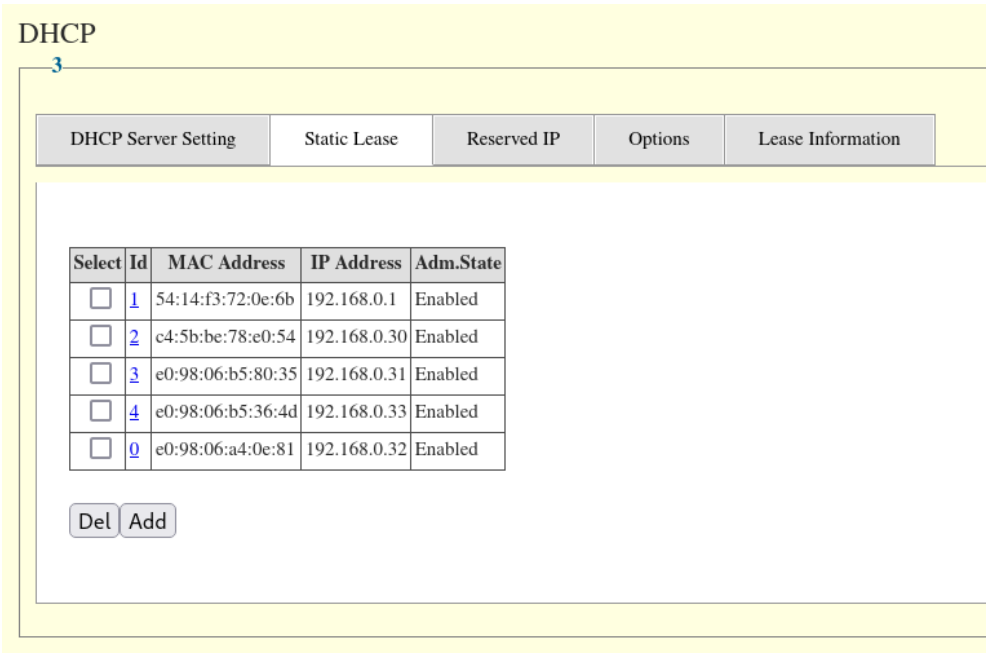


Figure 81: Configuration of static `DHCP` entries

For the changes to take effect, the `DHCP` server's range was modified to the one mentioned above and all of the devices were assigned an static lease in it. Besides that, the default router interface was also modified to to account for it.

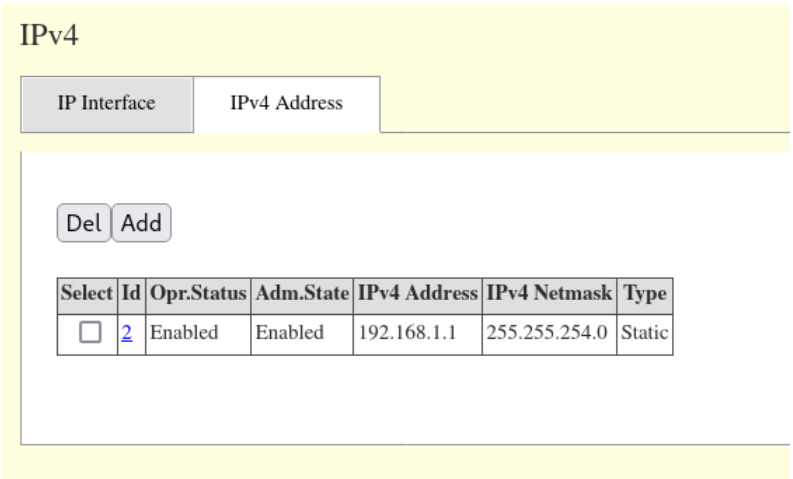


Figure 82: `IPv4` interface configuration

MiniPC crashes

At some point during the project the MiniPC started rejecting `SSH` connections with a `kex_exchange_identification` error message and the problem would not be solved until after resetting the physical machine. Some people hinted at the `SSH` server malfunctioning and a possible relation to voltage problems [132, 131]. After a closer inspection it appeared that whenever the issue occurred no new processes could be created, as if the system were overloaded, which would explain why `SSH` was refusing connections, as it could not create a new shell.

Initially the problem was noticed by seeing that backups had stopped being uploaded to S3. After restoring a backup, or by checking short after the remote machine was reset, one could see in the Grafana dashboard for the MiniPC that some anomalies were present in the temperature graph. Since the MiniPC had recently been relocated, it was assumed that bad ventilation in the new location was causing the system to overheat and become unresponsive, so it was moved to previous location.

Figure 83: MiniPC monitoring dashboard (30 days)

Initially appearing solved, soon enough crashes returned, and the temperature issue did not go away. After some pondering, it was found that the drastic temperature values had been brought by a change to the linux kernel boot parameters. At some point the system logs had been flooded with messages regarding `PCIe` errors, and the solution had been to update the `/etc/default/grub` file with `pcie_aspm=off` as a boot parameter [99], disabling Active State Power Management.

```
AER: Multiple Corrected error received: 0000:00:1c.0
PCIe Bus Error: severity=Corrected, type=Physical Layer, (ReceiverID)
  device [8086:4dbc] error status/mask=00000001/00002000
    [ 0] RxErr
```

Listing 16: `PCIe` error message flooding the `systemd` logs

Although at first this could appear to be the actual cause for overheating, after further analysis it was found that only the values reported by the monitoring script changed, while the physical temperature of the device and one reported by another utility did not, hinting at a possible issue with the sensors in charge of reporting temperature, not temperature itself.

Once again, the MiniPC was relocated to another location, and this time it finally appeared to be solved. Whether the problem was originally caused by bad ventilation, a faulty Ethernet connection or any other issue is still unknown, but to make sure that if it did ever happen again something could be done without being physically there, some mechanisms were put in place to improve remote access.

The first mechanism was the addition of a service that allowed the user to restart the system even when unresponsive. Based on the linux `sysrq` mechanism [98], `sysrqd` [77]. runs as a daemon allowing password-protected remote connections that give access to commands that cannot be ignored by the kernel, such as rebooting, terminating all processes, etc. For the service to work one must first enable the desired `sysrq` functionality on the system, which in Ubuntu can be done by writing `kernel.sysrq=X` (see [98] for values of `X`) to `/etc/sysctl.d/99-sysctl.conf`. To secure the access to the service a password can be placed in the `/etc/sysrqd.secret` file either in plaintext or as a hashed password following the `crypt` library format.

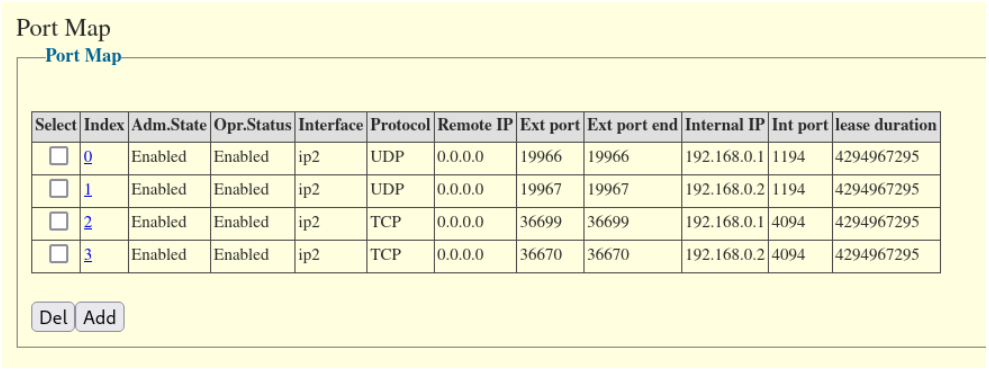


Figure 84: Configuration of port mappings

Because there were also issues with network interfaces sometimes not being available, such as the wireless interface failing to be assigned an IP by `dhclient`, the second mechanism was to perform pseudo-load-balancing of the exposed `OpenVPN` and `sysrqd` services, so that if after a reboot one of the interfaces became unavailable they could still be accessed from another external port.

Select	Id	MAC Address	IP Address	Adm.State
☐	1	54:14:f3:72:0e:6b	192.168.0.1	Enabled
☐	2	c4:5b:be:78:e0:54	192.168.0.30	Enabled
☐	3	e0:98:06:b5:80:35	192.168.0.31	Enabled
☐	4	e0:98:06:b5:36:4d	192.168.0.33	Enabled
☐	0	e0:98:06:a4:0e:81	192.168.0.32	Enabled
☐	5	7c:10:c9:b8:6d:4c	192.168.0.2	Enabled

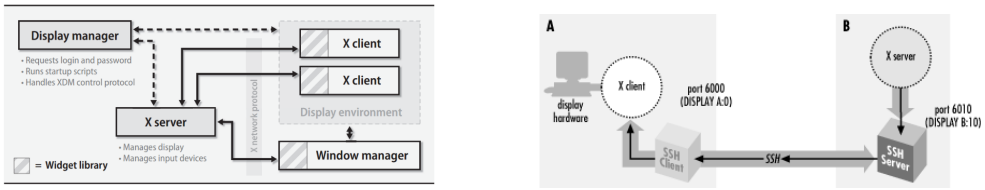
Figure 85: Configuration of backup static `DHCP` entries

This was achieved by adding a new static lease entry on the `DHCP` server for the wired interface and creating duplicated port mappings for both services. In the case of `OpenVPN` it was also necessary to modify the `/etc/openvpn/server/server.conf` file by commenting the line binding the local IP (`;local 192.168.0.1`) so that it could be chosen dinamically at boot time.

OpenVPN unreachable machines

Once the network had been reworked, unsurprisingly, another issue was found, this time related to access to the remote network through `OpenVPN`. When accessing the VPN and trying to ping devices on the remote network some responded and some did not, being reported as unreachable. Although most certainly caused by a misconfiguration of either the router or `OpenVPN`, with no knowledge of `OpenVPN` the latter and little time to look into the real cause of the problem a workaround was devised instead.

Since from within the `SSH` session the MiniPC could access all the other devices, the first idea that came to mind was to take advantage of the client-server architecture of the `X` window system [50] to establish a remote graphical session via `X` forwarding. This can easily be done by configuring appropriately the server (`/etc/ssh/sshd_config`) and client (`~/.ssh/config`) files and using using `ssh -X <HOST>` when connecting to the remote host [68]. The final result, however, was slow to the point of not being usable, so another solution was needed.



(a) `X` client-server architecture [113] (b) `X` forwarding via `SSH` [125]

Figure 86: Connection to remote `X` server

The second method tried involved using a proxy to route browser traffic through the VPN and into the MiniPC. Traditionally specific programs have been used to create `HTTP` proxies, but modern browsers support the use of the `SOCKS` [45] protocol, which is designed for this kind of task, to forward traffic. We can leverage `SSH` for this exact purpose by using the command `ssh -D 1337 -q -C -N <HOST>`, which creates a `SOCKS` proxy on port 1337 and tells `SSH` to compress traffic and work as a proxy instead of executing remote commands [70]. By executing the command and configuring the browser to use the proxy all devices could be accessed through their web interface once again.

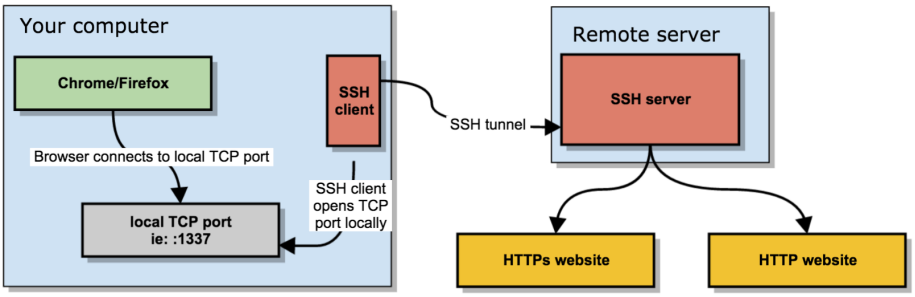


Figure 87: Web browsing through a `SOCKS5` proxy with `SSH` [70]

6 Conclusions

This section focuses on results. It analyzes the obstacles encountered during the project, some of the mistakes made during it and how they could have been avoided, and, finally, the lessons that can be drawn from both of them.

6.1 Obstacles

Remote management
The first obstacle, and probably the one that had the biggest impact on the project, was the need for remote management. Not having physical access to a device might not seem that problematic at a first glance, as technologies such as `SSH` or `OpenVPN` already provide solutions to the basic problems of remote access.

Reality, however, is often more complex than theory, and, as I could experience firsthand, network and hardware issues are a common occurrence in the real world. If something happens to the physical device or network you are trying to connect to, there is little you can do to fix the issue besides waiting for it to fix itself or fixing it yourself, and the latter option often involves spending time commuting.

Physical distance
The previous point leads directly into this one: the distance from home to the work environment. Even if the hurdle of remote management was not that bad by itself and surmountable by careful planning and some extra effort, when combined with a long distance it becomes a rather noticeable time sink. As at the start of the project the location was not yet known, not much could be done in terms of planning.

Delayed products
Another obstacle was related to the acquisition of hardware products. Some of the components took more time to be delivered than was expected, time during which the project was not able to proceed according to the expected timeline. Most of the items arrived in time, but some that did not ended up being delayed for some weeks.

Unexpected issues
Somewhat related to the first obstacle is the more general one of unexpected issues. Besides that of remote management, during the course of the project numerous hardware and software issues arose that hindered its progress. Some of them could be solved relatively quickly, others took more time, but for some of them workarounds had to be created to avoid wasting time.

6.2 Mistakes

Broad scope
The biggest mistake made, and probably a cause of some other ones, was defining a too ambitious and broad scope. Even if it might have seemed feasible on paper, and certainly achievable given more time, the time restrictions should have been taken more into account, as ultimately that was one of the limiting factors at play. However, as with most of the other mistakes, there was probably little to be done initially.

Risk management
Tied to the unexpected issues mentioned previously, another mistake was made, underestimating the time cost associated to them. Although they were initially considered, they were somewhat dismissed as not that critical, but the effect they had on the project far exceeded the estimates. On that note, tasks that may have appeared simple at a first glance ended up taking more time than expected to figure out.

Time management
At some points during the project I found myself overwhelmed with the amount of work that had to be done and the apparent little time that was available. Probably, this was partly cause by a lack of experience but also because of sloppy time management, which could have helped with the overall organization of the project.

Development focus
Finally, perhaps not a mistake but simply a tendency, too much focus was put on the operations side of **DevOps** and much less so on the development side of it. Being an IT student I am probably biased towards it, and during the project I might have been too fixated on creating a manageable and reproducible infrastructure and less on developing the actual functionality.

6.3 Lessons

Knowledge integration
One of the main takeaways from this project is the integration of knowledge from various fields. Because of the nature of IoT, many areas from both inside and outside of computer science come together in a single project. As a result, the project allowed me to not only hone IT related skills, but also learn new concepts from various disciplines.

Problem solving
In addition to having learned particular technologies and concepts, there were also multiple situations that required more general problem solving skills. Because nobody knows everything, even about a particular subject, one of the most valuable skills is that of figuring out the nature of a given problem, searching for information relating to it, and, finally, reaching a solution from the extracted information.

Journey before destination
The final lesson, and perhaps the most important one, had to do with the objective of the project. Even if the original goal was not actually reached, progress was made, and with it came the understanding of various technologies and devices. In that sense, the goal for the project should not have been considered an end by itself, but rather the means to an end, the learning process which leads to knowledge of some form.

7 Future work

As it has been mentioned before, various factors have reduced the final scope of the project and some planned features did not make it to the prototype. Although the project is finished, work remains to be done in the future. This section provides a rough overview of features and changes that could guide a potential roadmap.

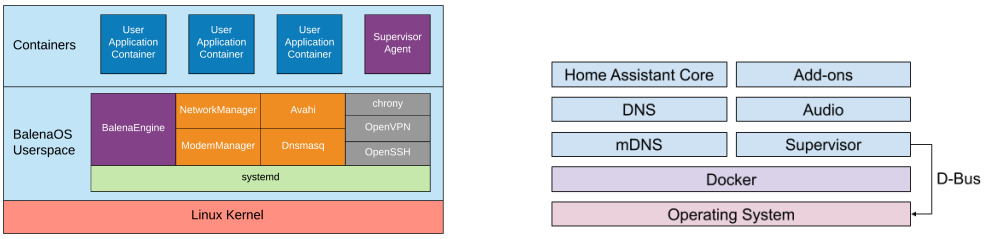
Improve capabilities
The first point on the list would be to improve the current capabilities of the system, getting closer to the original outline of the project. This would include adding functionality for automatic excess management, improving the visualization available by adding additional relevant information and fixing some of the existing problems.

Device support
After having sorted out the current system, it would be a good idea to add support for additional devices, both active and passive. The first step would be to carry out the integrations for **Shelly Plug S** and **OpenEVSE** that were left unfinished. After that, additional ones could be considered as the requirements of clients changed.

Software alternatives
Once enough devices have been integrated the understanding of the whole system will have improved considerably, allowing for better insight on which parts of it work as expected and which ones could be improved or should simply be replaced by others fulfilling the role better. Some initial thoughts on the matter would be to evaluate the possibility of using other visualization methods, such as **Chrongraf** instead of **Grafana**, or considering if the addition of **Home Assistant** would help with the growing number of devices to be supported.

Consumer solution
Finally, once the system prototype reached a mature enough point where everything worked as expected and functionality solved most of the proposed use cases, it would be possible to design a real consumer solution that took into account the resulting system and expected workloads and optimized the hardware and software components.

One possible solution could consist of an affordable consumer hardware, a **Raspberry Pi** for instance, installed in a practical manner, such as a wall mounted touchscreen or at least placed inside a compatible casing. On the software side of things, it would be worth the time studying available operating systems and their respective advantages, for example **BalenaOS** and **Home Assistant OS** (both focused on IoT workloads).



(a) **BalenaOS** architecture (b) **Home Assistant** architecture

Figure 88: Comparison of the two operating system candidates

8 References

[1] Acrel. *ACR10R energy meter for solar inverter*. URL: https://www.acrel.uk/acr10r-energy-meter-for-solar-inverter.html (visited on 22/04/2022).

[2] Amazon. *Amazon Simple Email Service*. URL: https://aws.amazon.com/ses/ (visited on 06/05/2022).

[3] Amazon. *Cloud Computing Services - Amazon Web Services*. URL: https://aws.amazon.com/ (visited on 06/05/2022).

[4] Amazon. *CloudWatch - Application and Infrastructure Monitoring*. URL: https://aws.amazon.com/cloudwatch/ (visited on 03/06/2022).

[5] Amazon. *DSD Tech SH-U10 Convertidor USB a RS485*. URL: https://www.amazon.es/DSD-TECH-SH-U10-Convertidor-Compatible/dp/B078X5H8H7 (visited on 22/04/2022).

[6] Amazon. *Free Cloud Computing Services - AWS Free Tier*. URL: https://aws.amazon.com/free/ (visited on 06/05/2022).

[7] Amazon. *Getting started with Amazon S3*. URL: https://docs.aws.amazon.com/AmazonS3/latest/userguide/GetStartedWithS3.html (visited on 06/05/2022).

[8] Amazon. *Global Infrastructure Regions & AZs*. URL: https://aws.amazon.com/about-aws/global-infrastructure/regions_az/ (visited on 06/05/2022).

[9] Amazon. *Managing access keys for IAM users*. URL: https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_access-keys.html (visited on 06/05/2022).

[10] Amazon. *What is IAM?* URL: https://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html (visited on 06/05/2022).

[11] Amazon. *What is the AWS Command Line Interface?* URL: https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-welcome.html (visited on 06/05/2022).

[12] Banda Ancha. *Firmwares para Askey RTF8115VW que permitan aprovechar su hardware*. URL: https://bandaancha.eu/foros/askey-rtf8115vw-cacharro-digno-1995-1740559 (visited on 27/05/2022).

[13] Apache. *Apache CouchDB*. URL: https://couchdb.apache.org/ (visited on 25/03/2022).

[14] ASUS. *Mini PC PN41 - Tech Specs*. URL: https://www.asus.com/Displays-Desktops/Mini-PCs/PN-PB-series/Mini-PC-PN41/techspec/ (visited on 18/03/2022).

[15] Jeff Atwood. *So You'd Like to Send Some Email (Through Code)*. URL: https://blog.codinghorror.com/so-youd-like-to-send-some-email-through-code/ (visited on 06/05/2022).

[16] Autarco. *SX Mark II series solar inverters*. URL: https://www.autarco.com/products/inverters/sx-mark-ii-series-solar-inverters/ (visited on 22/04/2022).

[17] BMitch. *docker - volumes vs mount binds. what are the use cases?* URL: https://serverfault.com/a/996804 (visited on 20/05/2022).

[18] Canonical. *Netplan - Backend-agnostic network configuration in YAML*. URL: https://netplan.io/ (visited on 27/05/2022).

[19] June Castillote. *How to Setup Cloudflare Dynamic DNS*. URL: https://adamtheautomator.com/cloudflare-dynamic-dns/ (visited on 01/04/2022).

[20] Cloudflare. *Dynamic DNS*. URL: https://www.cloudflare.com/learning/dns/glossary/dynamic-dns/ (visited on 01/04/2022).

[21] Cloudflare. *What is a domain name registrar?* URL: https://www.cloudflare.com/learning/dns/glossary/what-is-a-domain-name-registrar/ (visited on 01/04/2022).

[22] Cloudflare. *What is DNS? - How DNS works*. URL: https://www.cloudflare.com/learning/dns/what-is-dns/ (visited on 01/04/2022).

[23] OpenJS Foundation & Contributors. *Node-RED*. URL: https://nodered.org (visited on 27/02/2022).

[24] Wikipedia contributors. *2020–present global chip shortage — Wikipedia, The Free Encyclopedia*. 2022. URL: http://en.wikipedia.org/w/index.php?title=2020%5C%E2%5C%80%5C%93present%5C%20global%5C%20chip%5C%20shortage&oldid=1088492023 (visited on 18/03/2022).

[25] Wikipedia contributors. *Charging station — Wikipedia, The Free Encyclopedia*. 2022. URL: https://en.wikipedia.org/w/index.php?title=Charging_station&oldid=1093552234 (visited on 03/06/2022).

[26] Wikipedia contributors. *Cloud computing — Wikipedia, The Free Encyclopedia*. 2022. URL: https://en.wikipedia.org/w/index.php?title=Cloud_computing&oldid=1091198609 (visited on 22/04/2022).

[27] Wikipedia contributors. *Dependency hell — Wikipedia, The Free Encyclopedia*. 2022. URL: https://en.wikipedia.org/w/index.php?title=Dependency_hell&oldid=1077517071 (visited on 25/03/2022).

[28] Wikipedia contributors. *Domain Name System — Wikipedia, The Free Encyclopedia*. 2022. URL: https://en.wikipedia.org/w/index.php?title=Domain_Name_System&oldid=1090882376 (visited on 01/04/2022).

[29] Wikipedia contributors. *Dynamic DNS — Wikipedia, The Free Encyclopedia*. 2020. URL: https://en.wikipedia.org/w/index.php?title=Dynamic_DNS&oldid=984726018 (visited on 01/04/2022).

[30] Wikipedia contributors. *Home automation — Wikipedia, The Free Encyclopedia*. 2021. URL: https://en.wikipedia.org/w/index.php?title=Home_automation&oldid=1060659571 (visited on 27/02/2022).

[31] Wikipedia contributors. *IPsec — Wikipedia, The Free Encyclopedia*. 2022. URL: https://en.wikipedia.org/w/index.php?title=IPsec&oldid=1090301371 (visited on 20/05/2022).

[32] Wikipedia contributors. *IPv4 address exhaustion — Wikipedia, The Free Encyclopedia*. 2022. URL: https://en.wikipedia.org/w/index.php?title=IPv4_address_exhaustion&oldid=1090374520 (visited on 01/04/2022).

[33] Wikipedia contributors. *List of DNS record types — Wikipedia, The Free Encyclopedia*. 2022. URL: https://en.wikipedia.org/w/index.php?title=List_of_DNS_record_types&oldid=1090675362 (visited on 01/04/2022).

[34] Wikipedia contributors. *List of WLAN channels — Wikipedia, The Free Encyclopedia*. 2022. URL: https://en.wikipedia.org/w/index.php?title=List_of_WLAN_channels&oldid=1093319262 (visited on 03/06/2022).

[35] Wikipedia contributors. *Message transfer agent — Wikipedia, The Free Encyclopedia*. 2022. URL: https://en.wikipedia.org/w/index.php?title=Message_transfer_agent&oldid=1083371486 (visited on 06/05/2022).

[36] Wikipedia contributors. *Modbus — Wikipedia, The Free Encyclopedia*. 2022. URL: https://en.wikipedia.org/w/index.php?title=Modbus&oldid=1092284871 (visited on 22/04/2022).

[37] Wikipedia contributors. *NoSQL — Wikipedia, The Free Encyclopedia*. 2022. URL: https://en.wikipedia.org/w/index.php?title=NoSQL&oldid=1087561833 (visited on 25/03/2022).

[38] Wikipedia contributors. *Publish–subscribe pattern — Wikipedia, The Free Encyclopedia*. 2022. URL: https://en.wikipedia.org/w/index.php?title=Publish%E2%80%93subscribe_pattern&oldid=1073380138 (visited on 25/03/2022).

[39] Wikipedia contributors. *Relational database — Wikipedia, The Free Encyclopedia*. 2022. URL: https://en.wikipedia.org/w/index.php?title=Relational_database&oldid=1088987450 (visited on 25/03/2022).

[40] Wikipedia contributors. *Reverse proxy — Wikipedia, The Free Encyclopedia*. 2022. URL: https://en.wikipedia.org/w/index.php?title=Reverse_proxy&oldid=1089196899 (visited on 01/04/2022).

[41] Wikipedia contributors. *RS-485 — Wikipedia, The Free Encyclopedia*. 2022. URL: https://en.wikipedia.org/w/index.php?title=RS-485&oldid=1090820751 (visited on 22/04/2022).

[42] Wikipedia contributors. *Secure Shell — Wikipedia, The Free Encyclopedia*. 2022. URL: https://en.wikipedia.org/w/index.php?title=Secure_Shell&oldid=1090532743 (visited on 01/04/2022).

[43] Wikipedia contributors. *Service set (802.11 network) — Wikipedia, The Free Encyclopedia*. 2022. URL: https://en.wikipedia.org/w/index.php?title=Service_set_(802.11_network)&oldid=1073483485 (visited on 03/06/2022).

[44] Wikipedia contributors. *Simple Mail Transfer Protocol — Wikipedia, The Free Encyclopedia*. 2022. URL: https://en.wikipedia.org/w/index.php?title=Simple_Mail_Transfer_Protocol&oldid=1089433561 (visited on 06/05/2022).

[45] Wikipedia contributors. *SOCKS — Wikipedia, The Free Encyclopedia*. 2022. URL: https://en.wikipedia.org/w/index.php?title=SOCKS&oldid=1091130060 (visited on 10/06/2022).

[46] Wikipedia contributors. *Solar inverter — Wikipedia, The Free Encyclopedia*. 2022. URL: https://en.wikipedia.org/w/index.php?title=Solar_inverter&oldid=1091477297 (visited on 22/04/2022).

[47] Wikipedia contributors. *Telnet — Wikipedia, The Free Encyclopedia*. 2022. URL: https://en.wikipedia.org/w/index.php?title=Telnet&oldid=1091662448 (visited on 01/04/2022).

[48] Wikipedia contributors. *Time series database — Wikipedia, The Free Encyclopedia*. 2022. URL: https://en.wikipedia.org/w/index.php?title=Time_series_database&oldid=1073518068 (visited on 25/03/2022).

[49] Wikipedia contributors. *Virtual private network — Wikipedia, The Free Encyclopedia*. 2022. URL: https://en.wikipedia.org/w/index.php?title=Virtual_private_network&oldid=1091787826 (visited on 20/05/2022).

[50] Wikipedia contributors. *X Window System — Wikipedia, The Free Encyclopedia*. 2022. URL: https://en.wikipedia.org/w/index.php?title=X_Window_System&oldid=1090677160 (visited on 10/06/2022).

[51] Andrea Crawford. *DevOps a complete guide*. 2019. URL: https://www.ibm.com/cloud/learn/devops-a-complete-guide (visited on 27/02/2022).

[52] DezeStijn. *Reading Ginlong Solis inverter over serial and importing in Home Assistant over MQTT*. URL: https://sequr.be/blog/2021/08/reading-ginlong-solis-inverter-over-serial-and-importing-in-home-assistant-over-mqtt/ (visited on 22/04/2022).

[53] M. Díez-Mediavilla et al. "Performance of grid-tied PV facilities based on real data in Spain: Central inverter versus string system". In: *Energy Conversion and Management* 86 (2014), pp. 1128–1133. URL: https://www.sciencedirect.com/science/article/pii/S0196890414006128 (visited on 22/04/2022).

[54] DigitalOcean. *DigitalOcean - The developer cloud*. URL: https://www.digitalocean.com/ (visited on 06/05/2022).

[55] Docker. *Docker Context*. URL: https://docs.docker.com/engine/context/working-with-contexts/ (visited on 20/05/2022).

[56] Docker. *Docker Engine overview*. URL: https://docs.docker.com/engine/ (visited on 25/03/2022).

[57] Docker. *Docker overview*. URL: https://docs.docker.com/get-started/overview/ (visited on 20/05/2022).

[58] Docker. *Overview of Docker Compose*. URL: https://docs.docker.com/compose/ (visited on 25/03/2022).

[59] Docker. *Share Compose configurations between files and projects*. URL: https://docs.docker.com/compose/extends/ (visited on 20/05/2022).

[60] Docker. *Use bind mounts*. URL: https://docs.docker.com/storage/volumes/ (visited on 20/05/2022).

[61] Docker. *Use volumes*. URL: https://docs.docker.com/storage/bind-mounts/ (visited on 20/05/2022).

[62] Inc. Docker. *Docker*. URL: https://docker.com (visited on 27/02/2022).

[63] Inc. Docker. *What is a Container?* URL: https://www.docker.com/resources/what-container (visited on 27/02/2022).

[64] Read the Docs. *PyModbus - A Python Modbus Stack*. URL: https://pymodbus.readthedocs.io/en/latest/readme.html (visited on 22/04/2022).

[65] DuckDNS. *DuckDNS - About*. URL: https://www.duckdns.org/about.jsp (visited on 01/04/2022).

[66] Let's Encrypt. *Challenge Types*. URL: https://letsencrypt.org/docs/challenge-types/ (visited on 01/04/2022).

[67] Let's Encrypt. *Getting Started*. URL: https://letsencrypt.org/getting-started/ (visited on 01/04/2022).

[68] Stack Exchange. *How to forward X over SSH to run graphics applications remotely?* 2016. URL: https://unix.stackexchange.com/q/12755 (visited on 10/06/2022).

[69] Marcos G. *Modbus Error: [Invalid Message] Incomplete message received, expected at least 2 bytes (0 received)*. URL: https://stackoverflow.com/a/56923891 (visited on 22/04/2022).

[70] Mattias Geniar. *Create a SOCKS proxy on a Linux server with SSH to bypass content filters*. 2017. URL: https://ma.ttias.be/socks-proxy-linux-ssh-bypass-content-filters/ (visited on 01/04/2022).

[71] Alexander Gillis. *What is internet of things (IoT)?* 2021. URL: https://internetofthingsagenda.techtarget.com/definition/Internet-of-Things-IoT (visited on 27/02/2022).

[72] GitHub. *librsync*. URL: https://github.com/librsync/librsync (visited on 20/05/2022).

[73] GitHub. *Mailhog*. URL: https://github.com/mailhog/MailHog (visited on 20/05/2022).

[74] GitHub. *OpenEVSE PLUS*. URL: https://github.com/OpenEVSE/OpenEVSE_PLUS (visited on 03/06/2022).

[75] GitHub. *openvpn-install*. URL: https://github.com/Nyr/openvpn-install (visited on 20/05/2022).

[76] GitHub. *rsync*. URL: https://github.com/WayneD/rsync/ (visited on 20/05/2022).

[77] GitHub. *sysrqd*. URL: https://github.com/jd/sysrqd (visited on 03/06/2022).

[78] GitHub. *Tecnativa - Docker Duplicity*. URL: https://github.com/Tecnativa/docker-duplicity (visited on 20/05/2022).

[79] GitLab. *duplicity*. URL: https://duplicity.gitlab.io/ (visited on 20/05/2022).

[80] Google. *Cloud Computing Services - Google Cloud*. URL: https://cloud.google.com/ (visited on 06/05/2022).

[81] Docker Hub. *DuckDNS*. URL: https://hub.docker.com/r/linuxserver/duckdns/ (visited on 01/04/2022).

[82] Docker Hub. *Eclipse Mosquitto*. URL: https://hub.docker.com/_/eclipse-mosquitto/ (visited on 25/03/2022).

[83] Docker Hub. *Grafana*. URL: https://hub.docker.com/r/grafana/grafana/ (visited on 25/03/2022).

[84] Docker Hub. *InfluxDB*. URL: https://hub.docker.com/_/influxdb/ (visited on 25/03/2022).

[85] Docker Hub. *Mailhog*. URL: https://hub.docker.com/r/mailhog/mailhog (visited on 20/05/2022).

[86] Docker Hub. *Mailhog*. URL: https://hub.docker.com/r/portainer/portainer (visited on 20/05/2022).

[87] Docker Hub. *Mailhog*. URL: https://hub.docker.com/r/portainer/agent (visited on 20/05/2022).

[88] Docker Hub. *Nginx Proxy Manager*. URL: https://hub.docker.com/r/jc21/nginx-proxy-manager (visited on 01/04/2022).

[89] Docker Hub. *Node-RED*. URL: https://hub.docker.com/r/nodered/node-red/ (visited on 25/03/2022).

[90] IBM. *IBM Cloud*. URL: https://www.ibm.com/cloud (visited on 06/05/2022).

[91] InfluxData. *Chronograf: Complete Dashboard Solution for InfluxDB*. URL: https://www.influxdata.com/time-series-platform/chronograf/ (visited on 25/03/2022).

[92] InfluxData. *InfluxDB*. URL: https://docs.influxdata.com/influxdb (visited on 27/02/2022).

[93] InfluxData. *Install InfluxDB*. URL: https://docs.influxdata.com/influxdb/v2.1/install/ (visited on 25/03/2022).

[94] InfluxData. *Query data with InfluxQL*. URL: https://docs.influxdata.com/influxdb/v2.1/query-data/influxql/ (visited on 25/03/2022).

[95] InfluxData. *Use Grafana with InfluxDB OSS*. URL: https://docs.influxdata.com/influxdb/v2.1/tools/grafana/ (visited on 25/03/2022).

[96] jc21. *Nginx Proxy Manager*. URL: https://nginxproxymanager.com/ (visited on 01/04/2022).

[97] John Jetmore. *Swaks - Swiss Army Knife for SMTP*. URL: https://jetmore.org/john/code/swaks/ (visited on 06/05/2022).

[98] The Linux Kernel. *Linux Magic System Request Key Hacks*. URL: https://www.kernel.org/doc/html/latest/admin-guide/sysrq.html (visited on 10/06/2022).

[99] Colin Ian King. *PCIe Bus Error: severity=Corrected, type=Physical Layer, id=00e5(Receiver ID)*. 2016. URL: https://askubuntu.com/a/863301 (visited on 10/06/2022).

[100] Kingston. *2 Types of M.2 SSDs: SATA and NVMe - Kingston Technology*. URL: https://www.kingston.com/en/blog/pc-performance/two-types-m2-vs-ssd (visited on 18/03/2022).

[101] Kingston. *NVMe vs SATA: What is the difference? - Kingston Technology*. URL: https://www.kingston.com/en/blog/pc-performance/nvme-vs-sata (visited on 18/03/2022).

[102] Adam Kościelak. *Mail Transfer Agent: advantages*. URL: https://elasticemail.com/blog/advantages-of-elastic-email-mail-transfer-agent (visited on 06/05/2022).

[103] Grafana Labs. *Flux support in Grafana*. URL: https://grafana.com/docs/grafana/latest/datasources/influxdb/influxdb-flux/ (visited on 25/03/2022).

[104] Grafana Labs. *Grafana*. URL: https://grafana.com (visited on 27/02/2022).

[105] Grafana Labs. *InfluxDB data source*. URL: https://grafana.com/docs/grafana/latest/datasources/influxdb/ (visited on 25/03/2022).

[106] Grafana Labs. *Run Grafana Docker image*. URL: https://grafana.com/docs/grafana/latest/administration/configure-docker/ (visited on 25/03/2022).

[107] James Lewis and Martin Fowler. *Microservices: a definition of this new architectural term*. 2014. URL: https://martinfowler.com/articles/microservices.html (visited on 27/02/2022).

[108] TP-Link. *Brief Introduction of AP Isolation*. URL: https://www.tp-link.com/es/support/faq/2089/ (visited on 10/06/2022).

[109] Linksys. *Getting to know the AP Isolation function*. URL: https://www.linksys.com/gb/support-article?articleNum=135098 (visited on 10/06/2022).

[110] Mike Loukides. *What is DevOps?* 2012. URL: http://radar.oreilly.com/2012/06/what-is-devops.html (visited on 27/02/2022).

[111] Mailgun. *Transactional Email API Service For Developers.* URL: https://www.mailgun.com/ (visited on 06/05/2022).

[112] Mailgun. *Why not just use Sendmail + Postfix + Courier IMAP?* URL: https://documentation.mailgun.com/en/latest/faqs.html#why-not-just-use-sendmail-postfix-courier-imap (visited on 06/05/2022).

[113] MangoHot. *X Window System.* 2015. URL: https://nasa.cs.nctu.edu.tw/sap/2015/slides/X_Window_System.pdf (visited on 10/06/2022).

[114] Microsoft. *Cloud Computing Services - Microsoft Azure.* URL: https://azure.microsoft.com (visited on 06/05/2022).

[115] MongdoDB. *MongdoDB - Build faster. Build smarter.* URL: https://www.mongodb.com/ (visited on 25/03/2022).

[116] Troy Morlan. *String Versus Central Inverters for Solar PV Projects.* URL: https://blog.norcalcontrols.net/string-versus-central-inverters-solar-pv (visited on 22/04/2022).

[117] Mosquitto. *Eclipse Mosquitto.* URL: https://mosquitto.org/ (visited on 25/03/2022).

[118] Comunidad Movistar. *HGU RTF8115VW: ¿Para qué sirven los servers DHCP 0, 1, 2?* URL: https://comunidad.movistar.es/t5/Soporte-Fibra-y-ADSL/HGU-RTF8115VW-Para-qu%C3%A9-sirven-los-servers-DHCP-0-1-2/td-p/4631406 (visited on 27/05/2022).

[119] MySQL. *MySQL.* URL: https://www.mysql.com/ (visited on 25/03/2022).

[120] MySQL. *PostgreSQL: The World's Most Advanced Open Source Relational Database.* URL: https://www.postgresql.org/ (visited on 25/03/2022).

[121] NGINX. *NGINX Reverse Proxy.* URL: https://docs.nginx.com/nginx/admin-guide/web-server/reverse-proxy/ (visited on 01/04/2022).

[122] Node-RED. *Running under Docker.* URL: https://nodered.org/docs/getting-started/docker (visited on 25/03/2022).

[123] Node.js. *npm global or local packages.* URL: https://nodejs.dev/learn/npm-global-or-local-packages/ (visited on 25/03/2022).

[124] npm. *Dependency Hell - How npm Works.* URL: https://npm.github.io/how-npm-works-docs/theory-and-design/dependency-hell.html (visited on 25/03/2022).

[125] O'Reilly. *SSH: The Secure Shell, The Definitive Guide.* 2016. URL: https://docstore.mik.ua/orelly/networking_2ndEd/ssh/ch09_03.htm (visited on 10/06/2022).

[126] OASIS. *MQTT.* URL: https://mqtt.org (visited on 27/02/2022).

[127] OpenEVSE. *OpenEVSE Kits - Electric Vehicle Charging Solutions.* URL: https://www.openevse.com/kits.html (visited on 03/06/2022).

[128] OpenSSH. *OpenSSH.* URL: https://www.openssh.com/ (visited on 01/04/2022).

[129] OpenVPN. *Business VPN - Next-Gen VPN.* URL: https://openvpn.net/ (visited on 20/05/2022).

[130] Oracle. *Oracle Cloud Infrastructure.* URL: https://www.oracle.com/cloud/ (visited on 06/05/2022).

[131] Stack Overflow. *How to fix? kex_exchange_identification: read: Connection reset by peer*. URL: https://stackoverflow.com/q/69394001 (visited on 10/06/2022).

[132] Stack Overflow. *SSH kex_exchange_identification: read: Connection reset by peer*. URL: https://stackoverflow.com/q/61185751 (visited on 10/06/2022).

[133] Pilot6. *Can't get RTL8125B working on 20.04*. URL: https://askubuntu.com/a/1260626 (visited on 18/03/2022).

[134] Portainer. *Mailhog*. URL: https://docs.portainer.io/start/architecture (visited on 20/05/2022).

[135] Python. *venv - Creation of virtual environments*. URL: https://docs.python.org/3/library/venv.html (visited on 25/03/2022).

[136] Realtek. *Gaming Ethernet Family Controller Software*. URL: https://www.realtek.com/en/component/zoo/category/network-interface-controllers-10-100-1000m-gigabit-ethernet-pci-express-software (visited on 18/03/2022).

[137] Realtek. *RTL8125BG(S)-CG*. URL: https://www.realtek.com/en/products/communications-network-ics/item/rtl8125bg-s-cg (visited on 18/03/2022).

[138] Chris Richardson. *Microservices: a definition of this new architectural term*. 2015. URL: https://www.nginx.com/blog/introduction-to-microservices (visited on 27/02/2022).

[139] Ozgur Sahingoz and Ahmet Sonmez. "Agent-Based Fault Tolerant Distributed Event System." In: *Computing and Informatics* 26 (Jan. 2007), pp. 489–506. URL: https://www.researchgate.net/publication/220106069_Agent-Based_Fault_Tolerant_Distributed_Event_System (visited on 25/03/2022).

[140] ScyllaDB. *ScyllaDB - NoSQL vs SQL*. URL: https://www.scylladb.com/learn/nosql/nosql-vs-sql/ (visited on 25/03/2022).

[141] Shelly. *Products - Shelly 3EM*. URL: https://shelly.cloud/products/shelly-3em-smart-home-automation-energy-meter/ (visited on 22/04/2022).

[142] Shelly. *Products - Shelly Plug S*. URL: https://shelly.cloud/products/shelly-plug-s-smart-home-automation-device/ (visited on 03/06/2022).

[143] Shelly. *Shelly 3EM - API Reference*. URL: https://shelly-api-docs.shelly.cloud/gen1/#shelly-3em (visited on 27/05/2022).

[144] Stanislav Shymanskyi. *How Does Mailgun Sandbox Work?* URL: https://mailtrap.io/blog/mailgun-sandbox-tutorial/ (visited on 06/05/2022).

[145] Solis. *S6 string inverter*. URL: https://www.ginlong.com/solarinverter1/700-3600w_mini_s6_global.html (visited on 22/04/2022).

[146] SSH. *SSH Tunnel*. URL: https://www.ssh.com/academy/ssh/tunneling (visited on 01/04/2022).

[147] Chris Stetson. *Introducing the Microservices Reference Architecture from NGINX*. 2016. URL: https://www.nginx.com/blog/introducing-the-nginx-microservices-reference-architecture (visited on 27/02/2022).

[148] Gabby Taylor. *Taming the tar command: Tips for managing backups in Linux*. URL: https://www.redhat.com/sysadmin/taming-tar-command (visited on 20/05/2022).

[149] Home Assistant Core Team and Community. *Home assistant*. URL: https://www.home-assistant.io (visited on 27/02/2022).

[150] Ubuntu. *Ubuntu 20.04.4 LTS (Focal Fossa)*. URL: https://ubuntu.com/kernel/lifecycle (visited on 18/03/2022).

[151] Ubuntu. *Ubuntu kernel lifecycle and enablement stack*. URL: https://ubuntu.com/kernel/lifecycle (visited on 18/03/2022).

[152] Ask Ubuntu. *A start job is running for wait for network to be configured*. 2017. URL: https://askubuntu.com/q/972215 (visited on 27/05/2022).

[153] Eben Upton. *Supply Chain, shortages, and our first-ever price increase*. 2021. URL: https://www.raspberrypi.com/news/supply-chain-shortages-and-our-first-ever-price-increase/ (visited on 18/03/2022).

[154] Wietse Venema. *The Postfix Home Page*. URL: https://www.postfix.org/ (visited on 06/05/2022).

[155] Adam Wiggins. *The twelve-factor app*. 2017. URL: https://12factor.net (visited on 27/02/2022).

[156] Wikipedia contributors. *Chroot — Wikipedia, The Free Encyclopedia*. 2022. URL: https://en.wikipedia.org/w/index.php?title=Chroot&oldid=1090719602 (visited on 25/03/2022).

[157] Wikipedia contributors. *Flow-based programming — Wikipedia, The Free Encyclopedia*. 2022. URL: https://en.wikipedia.org/w/index.php?title=Flow-based_programming&oldid=1091088877 (visited on 25/03/2022).

[158] Wikipedia contributors. *Internet of things — Wikipedia, The Free Encyclopedia*. 2022. URL: https://en.wikipedia.org/w/index.php?title=Internet_of_things&oldid=1073917710 (visited on 27/02/2022).

[159] Wikipedia contributors. *OS-level virtualization — Wikipedia, The Free Encyclopedia*. 2022. URL: https://en.wikipedia.org/w/index.php?title=OS-level_virtualization&oldid=1090462427 (visited on 25/03/2022).

[160] Wikipedia contributors. *Visual programming language — Wikipedia, The Free Encyclopedia*. 2022. URL: https://en.wikipedia.org/w/index.php?title=Visual_programming_language&oldid=1091089785 (visited on 25/03/2022).

[161] ADSL Zone. *Sobre el router hgu Askey*. URL: https://www.adslzone.net/foro/o2.188/sobre-router-hgu-askey.580551/ (visited on 27/05/2022).

[162] Redes Zone. *Configura el router Askey RTF8115VW de Movistar en modo bridge o puente*. URL: https://www.redeszone.net/tutoriales/configuracion-routers/configurar-askey-rtf8115vw-movistar-bridge-puente/ (visited on 27/05/2022).

Appendices

A User manual

This document will detail the steps required to deploy a visualization system of photovoltaic power consumption in an installation with an **Autarco SX-MII** solar inverter, a **Shelly 3EM** meter and an **ASUS MiniPC** as the main hub.

Although the documentation is written for the previously mentioned hardware, it should be fairly straightforward to adapt it to other hardware combinations.

Installation

Operating system

The operating system used for the main hub will be an **Ubuntu 20.04** with Hardware Enablement (HWE) enable to ensure that the latest drivers are available, as some are needed by the **ASUS MiniPC**. Installation should be straightforward, but in case of doubt check the official tutorial.

Software dependencies

Docker will be used as an orchestrator for our containers, which requires both **Docker Engine** and **Docker Compose**. They can either be installed by following the official instructions listed on the website or through the package manager:

```
$ sudo apt install docker.io docker-compose
```

Although **MQTT** will be executed as a container, the following package can be used to send local data to a broker and test network connection to other devices:

```
$ sudo apt install mosquitto-clients
```

To enable the monitorization of the main hub the following packages are required:

```
$ sudo apt install msr-tools jq
```

In case remote access to the main hub is required **OpenVPN** can be used to connect to it from outside the local network. The following command downloads and executes a script that will walk the user through the installation process:

```
$ wget https://git.io/vpn -O openvpn-install.sh
$ bash openvpn-install.sh
```

Software utilities

This section contains a list of tools that might be useful to more easily manage the main hub.

Remote management through `ssh` can be tedious for complex tasks such as development, `tmux` can be used to run multiple terminal instances in a single session:

```
$ sudo apt install tmux
```

Network tools such as `netstat` or `route` can be used to help debug network problems, they are included in the `net-tools` package:

```
$ sudo apt install net-tools
```

By default Ubuntu server does not ship with a graphical interface, but it can be enabled if needed by installing `ubuntu-desktop-minimal`:

```
$ sudo apt install ubuntu-desktop-minimal
```

Firewall configuration is a very important task if a machine must be exposed to the Internet, in that case `ufw` allows a user to easily configure it:

```
$ sudo apt install ufw
```

Configuration

System settings

After installing the operating system and packages one might want to configure some of its services. Here we describe some optional configuations together with a required service that will provide **Node-RED** with status information.

Configure SSH

Configuring `SSH` is a must if we want to access a device remotely. To do so we need a cryptographic key pair on the machine that will have access, which can easily be generated with `ssh-keygen`.

Once generated a `.pub` public key file will be generated. To grant access on the remote machine either copy it manually to `~/.ssh/authorized_keys` or use following command (replacing variables as needed):

```
$ ssh-copy-id -i ~/.ssh/key.pub user@host
```

It usually is a good idea to disable password authentication as soon as public keys are in place. This can be achieved by setting `PasswordAuthentication no` in the `SSH` deamon configuration file `/etc/ssh/sshd_config`. Once modified it is necessary to restart the service:

```
$ sudo systemctl restart sshd
```

Configure time zone

Time zones are useful for making sense of logs and essential to ensure that services executing periodic jobs such as `cron` are run when they are supposed to.

Timezones follow the format described in the **tz** database, and available ones can be obtained with `timedatectl list-timezones`. They can be controlled with the following command:

```
$ sudo timedatectl set-timezone timezone
```

Activate `systemd` monitoring service

Finally, we need to enable the service in charge of monitoring systems resources that will be sent via **MQTT**:

```
$ sudo cp minipc-status.sh /usr/local/bin/minipc-status.sh
$ sudo cp minipc-status.service /etc/systemd/system/minipc-status.service
$ sudo systemctl enable minipc-status
```

Devices

As more devices are added to the network they can quickly become difficult to manage unless they are easily identifiable. For this purpose we can create a subnet (or at least reserve a range of IPs) for our IoT devices, for instance `192.168.10.0/24`. For specific information on configuration see your router's manual or its manufacturer's site.

Shelly

All **Shelly** devices are configured pretty much the same way. Once they are turned on they will create a wireless access point named after the device, after joining the network they can be accessed at `192.168.33.1`. Specific details can be found on the **Shelly**'s website, for instance see **Shelly 3EM**'s product page and API documentation. Information for other devices can be found in a similar way.

Services

To deploy the set of services a `docker-compose.yml` file is provided together with environment configuration files. The following table describes the list of services, files associated to them that require configuration (if any) and relevant links to their respective official sites:

Service	File	Documentation
Node-RED		[1], [2]
InfluxDB	`influxdb.env`	[1], [2]
Grafana	`grafana.env`	[1], [2]
Portainer		[1], [2]
Duckdns	`duckdns.env`	[1], [2]

Once the services have been started with `docker-compose`, the provided files describing **Node-RED** flow and **Grafana** dashboards and datasources can be either manually imported, via web the UI, or automatically through by executing the provided `import-configuration.sh` script.

Finally, **Node-RED** requires access to the REE API to fetch information about prices. As explained in their website, the public token changes periodically, so it is necessary to request a personal token by sending a mail to `consultasios@ree.es`.

A.1 minipc-status.sh

```
#!/bin/bash

period=30

modprobe msr

compute_ram_usage()
{
    awk '$1=="MemAvailable:" {available=$2}; $1=="MemTotal:"
    ↪  {total=$2}; END {print 100*(total-available)/total}'
    ↪  /proc/meminfo
}

compute_disk_usage()
{
    df / | tail -1 | awk '{print 100*$3/$4}'
}

compute_cpu_usage()
{
    top -bn 2 -d 0.5 | grep '^%Cpu' | tail -n 1 | awk '{print
    ↪  $2+$4+$6}'
}

compute_cpu_temp()
{
    echo $((98 - $(rdmsr --bitfield 22:16 -u 0x1B1)))
}

while sleep $period
do
    ram_usage=$(compute_ram_usage)
    disk_usage=$(compute_disk_usage)
    cpu_usage=$(compute_cpu_usage)
    cpu_temp=$(compute_cpu_temp)

    message=$(jq -n \
        --arg ru "$ram_usage" \
        --arg du "$disk_usage" \
        --arg cu "$cpu_usage" \
        --arg ct "$cpu_temp" \
        '{ram_usage: $ru, disk_usage: $du, cpu_usage: $cu, cpu_temp:
        ↪  $ct}')

    mosquitto_pub -h localhost -t "minipc" -m "$message"
done
```

A.2 minipc-status.service

```
[Unit]
Description=MiniPC resource monitorization

[Service]
ExecStart=/usr/local/bin/minipc-status.sh
Restart=on-failure

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

A.3 *.json configuration files

Configuration files for Grafana dashboards and datasources as well as Node-RED flows are not included because they are too long and not very interesting in terms of content.

A.4 influxdb-setup.sh

```
#!/bin/bash
set -euo pipefail

influx v1 dbrp create                              \
  --org ${DOCKER_INFLUXDB_INIT_ORG}                \
  --bucket-id ${DOCKER_INFLUXDB_INIT_BUCKET_ID} \
  --db ${DOCKER_INFLUXDB_INIT_BUCKET}              \
  --rp ${DOCKER_INFLUXDB_INIT_BUCKET}-policy     \
  --default

influx v1 auth create                                    \
  --org ${DOCKER_INFLUXDB_INIT_ORG}                      \
  --username ${DOCKER_INFLUXDB_INIT_USERNAME}        \
  --password ${DOCKER_INFLUXDB_INIT_PASSWORD}        \
  --write-bucket ${DOCKER_INFLUXDB_INIT_BUCKET_ID}  \
  --read-bucket ${DOCKER_INFLUXDB_INIT_BUCKET_ID}
```

A.5 import-configuration.sh

```
#!/bin/bash

set -euo pipefail

global_test_or_read()
{
    [[ -v $1 ]] || read -p "${2:-$1}: " "$1"
}

nodered_import_flows()
{
    curl -X POST localhost:1880/flows \
        -H "Content-Type: application/json"
        --data-binary @nodered/flows.json
}

grafana_import_datasources()
{
    for file in grafana/datasources/*
    do
        curl -X POST localhost:3000/api/datasources     \
            -H "Content-Type: application/json"         \
            -u "$GRAFANA_USERNAME:$GRAFANA_PASSWORD"    \
            --data-binary "@$file"
    done
}

grafana_import_dashboards()
{
    for file in grafana/datasources/*
    do
        curl -X POST localhost:3000/api/dashboards/db   \
            -H "Content-Type: application/json"         \
            -u "$GRAFANA_USERNAME:$GRAFANA_PASSWORD"    \
            --data-binary "@$file"
    done
}

global_test_or_read GRAFANA_USERNAME
global_test_or_read GRAFANA_PASSWORD

nodered_import_flows
grafana_import_datasources
grafana_import_dashboards
```

A.6 export-configuration.sh

```
#!/bin/bash

set -euo pipefail

global_test_or_read()
{
    [[ -v $1 ]] || read -p "${2:-$1}: " "$1"
}

nodered_export_flows()
{
    curl -s localhost:1880/flows > nodered/flows.json
}

grafana_export_datasources()
{
    curl -s localhost:3000/api/datasources          \
        -u "$GRAFANA_USERNAME:$GRAFANA_PASSWORD" |  \
        jq -c '.[]' | split -l 1 - grafana/datasources/
}

grafana_export_dashboards()
{
    local jq_query='.[] | select(.type == "dash-db") | .uid'
    for id in $(curl -s localhost:3000/api/search           \
                -u "$GRAFANA_USERNAME:$GRAFANA_PASSWORD"    \
                | jq -r "$jq_query")
    do
        curl -s "localhost:3000/api/dashboards/uid/$id"     \
            -u "$GRAFANA_USERNAME:$GRAFANA_PASSWORD"        \
            > "grafana/dashboards/$id"
    done
}

global_test_or_read GRAFANA_USERNAME
global_test_or_read GRAFANA_PASSWORD

mkdir -p nodered
nodered_export_flows
mkdir -p grafana/datasources
grafana_export_datasources
mkdir -p grafana/dashboards
grafana_export_dashboards
```

A.7 docker-compose.yml

```yaml
services:
    nodered:
        image: nodered/node-red:2.2.2
        ports:
            - "1880:1880"
        volumes:
            - nodered-data:/data
        depends_on:
            - influxdb
            - mosquitto
        restart: always
        container_name: nodered

    grafana:
        image: grafana/grafana:8.2.6
        env_file:
            - ./env/config/grafana.env
        ports:
            - "3000:3000"
        volumes:
            - grafana-data:/var/lib/grafana
        depends_on:
            - influxdb
        restart: always
        container_name: grafana

    influxdb:
        image: influxdb:2.1
        env_file:
            - ./env/config/influxdb.env
        ports:
            - "8086:8086"
        volumes:
            - influxdb-data:/var/lib/influxdb2
            - influxdb-config:/etc/influxdb2
            - influxdb-backup:/var/lib/influxdb2/backup
        restart: always
        container_name: influxdb

    mosquitto:
        image: eclipse-mosquitto:2.0.14
        command: mosquitto -c /mosquitto-no-auth.conf
        ports:
            - "1883:1883"
        volumes:
            - mosquitto-config:/mosquitto/config
            - mosquitto-data:/mosquitto/data
            - mosquitto-log:/mosquitto/log
        restart: always
```

```
        container_name: mosquitto

    portainer:
        image: portainer/portainer-ce:2.9.3
        command: --admin-password "${PORTAINER_PASSWORD}"
        ports:
            - "8000:8000"
            - "9000:9000"
        volumes:
            - /var/run/docker.sock:/var/run/docker.sock
            - portainer-data:/data
        restart: always
        container_name: portainer

     duckdns:
         image: lscr.io/linuxserver/duckdns
         env_file:
             - ./env/config/duckdns.env
         restart: always
         container_name: duckdns

volumes:
    nodered-data:
    grafana-data:
    influxdb-data:
    influxdb-config:
    influxdb-backup:
    mosquitto-data:
    mosquitto-config:
    mosquitto-log:
    portainer-data:
```

A.8 *.env environment files

.env

```
# Generate with htpasswd
# https://portainer.readthedocs.io/en/stable/configuration.html
PORTAINER_PASSWORD=
```

influxdb.env

```
# Enable automated influxdb setup
DOCKER_INFLUXDB_INIT_MODE=setup
# Username of the main influxdb user
DOCKER_INFLUXDB_INIT_USERNAME=
# Password of the main influxdb user
DOCKER_INFLUXDB_INIT_PASSWORD=
# Name of the main influxdb organization
DOCKER_INFLUXDB_INIT_ORG=
# Name of the main influxdb bucket
DOCKER_INFLUXDB_INIT_BUCKET=
# Fix issues when restoring
# https://github.com/influxdata/influxdb/issues/22890
TMPDIR=/var/lib/influxdb2
```

grafana.env

```
# Password of the admin user
GF_SECURITY_ADMIN_PASSWORD=
# Allow anonymous access without login
GF_AUTH_ANONYMOUS_ENABLED=true
# Set permissions for anonymous access
GF_AUTH_ANONYMOUS_ORG_ROLE=Viewer
```

duckdns.env

```
# Account token provided by duckdns
TOKEN=
# Comma separated list of subdomains to be updated
SUBDOMAINS=
# Timezone for logging purposes
# https://en.wikipedia.org/wiki/Tz_database
TZ=
```

B Development listings

B.1 Initial `docker-compose.yml`

```yaml
version: "2"

services:
  nodered:
    image: nodered/node-red:2.2.2
    volumes:
      - ./nodered/data:/data
    ports:
      - "1880:1880"
    depends_on:
      - nginx
    restart: always
    container_name: nodered

  grafana:
    image: grafana/grafana:8.2.6
    user: "1000"
    volumes:
      - ./grafana/data:/var/lib/grafana
    ports:
      - "3000:3000"
    depends_on:
      - influxdb
      - nginx
    restart: always
    container_name: grafana

  influxdb:
    image: influxdb:2.1
    volumes:
      - ./influxdb/data:/var/lib/influxdb2
      - ./influxdb/config:/etc/influxdb2
    ports:
      - "8086:8086"
    restart: always
    container_name: influxdb

  mosquitto:
    image: eclipse-mosquitto
    volumes:
      - ./mosquitto/config:/mosquitto/config
    ports:
      - "1883:1883"
    restart: always
    container_name: mosquitto

  nginx:
```

```
    image: jc21/nginx-proxy-manager
    volumes:
      - ./nginx/data:/data
      - ./nginx/letsencrypt:/etc/letsencrypt
    ports:
      - "80:80"
      - "81:81"
      - "443:443"
    restart: always
    container_name: nginx

  duckdns:
    image: lscr.io/linuxserver/duckdns
    env_file:
      - duckdns.env
    restart: always
    container_name: duckdns
```

B.2 Final `docker-compose.yml`

```
version: "3"

services:
    nodered:
        volumes:
            - nodered-data:/data
        depends_on:
            - influxdb
            - mosquitto
        restart: always
        container_name: nodered

    grafana:
        env_file:
            - ./env/config/grafana.env
        volumes:
            - grafana-data:/var/lib/grafana
        depends_on:
            - influxdb
        restart: always
        container_name: grafana

    influxdb:
        env_file:
            - ./env/config/influxdb.env
        volumes:
            - influxdb-data:/var/lib/influxdb2
            - influxdb-config:/etc/influxdb2
            - influxdb-backup:/var/lib/influxdb2/backup
        restart: always
        container_name: influxdb
```

```
    mosquitto:
        volumes:
            - mosquitto-config:/mosquitto/config
            - mosquitto-data:/mosquitto/data
            - mosquitto-log:/mosquitto/log
        restart: always
        container_name: mosquitto

    duplicity:
        env_file:
            - ./env/config/duplicity.env
        volumes:
            - /var/run/docker.sock:/var/run/docker.sock:ro
            - duplicity-root:/root
            - nodered-data:/mnt/backup/src/nodered-data
            - grafana-data:/mnt/backup/src/grafana-data
            - influxdb-backup:/mnt/backup/src/influxdb-backup
        depends_on:
            - influxdb
        restart: always
        container_name: duplicity

volumes:
    nodered-data:
    grafana-data:
    influxdb-data:
    influxdb-config:
    influxdb-backup:
    mosquitto-data:
    mosquitto-config:
    mosquitto-log:
    duplicity-root:
```

B.3 Local

B.3.1 docker-compose.override.yml

```yaml
version: "3"

services:
    nodered:
        image: nodered/node-red:2.2.2
        ports:
            - "1880:1880"

    grafana:
        image: grafana/grafana:8.2.6
        ports:
            - "3000:3000"

    influxdb:
        image: influxdb:2.1
        ports:
            - "8086:8086"

    mosquitto:
        image: eclipse-mosquitto:2.0.14
        command: mosquitto -c /mosquitto-no-auth.conf
        ports:
            - "1883:1883"

    duplicity:
        image: ghcr.io/tecnativa/docker-duplicity-docker
        volumes:
            - backup:/mnt/backup/dst
        hostname: devel-backup

    mailhog:
        image: mailhog/mailhog
        command:
            - -storage=maildir
            - -maildir-path=/home/mailhog
        volumes:
            - mailhog:/home/mailhog
        ports:
            - "1025:1025"
            - "8025:8025"
        restart: always
        container_name: mailhog

volumes:
    backup:
    mailhog:
```

B.3.2 *.env environment files

influxdb.env

```
DOCKER_INFLUXDB_INIT_MODE=setup
DOCKER_INFLUXDB_INIT_USERNAME=admin
DOCKER_INFLUXDB_INIT_PASSWORD=development
DOCKER_INFLUXDB_INIT_ORG=ontec
DOCKER_INFLUXDB_INIT_BUCKET=data
TMPDIR=/var/lib/influxdb2
```

grafana.env

```
GF_SECURITY_ADMIN_PASSWORD=devel
GF_AUTH_ANONYMOUS_ENABLED=true
GF_AUTH_ANONYMOUS_ORG_ROLE=Viewer
```

duplicity.env

```
# Duplicity
DST=file:///mnt/backup/dst
PASSPHRASE=devel
# Mail
SMTP_HOST=mailhog
SMTP_PORT=1025
EMAIL_FROM=backup@devel.com
EMAIL_TO=admin@devel.com
TZ=Europe/Madrid
# InfluxDB job
JOB_190_WHAT="rm -f /mnt/backup/src/influxdb-backup/*"
JOB_190_WHEN="daily weekly"
JOB_200_WHAT="docker container exec influxdb influx backup
↪  /var/lib/influxdb2/backup"
JOB_200_WHEN="daily weekly"
```

B.3.3 internal.sh

```
#!/bin/bash

set -euo pipefail
[[ -v DEBUG ]] && set -x

restore()
{
    [[ -n "${1:-}" ]] ||
    { echo 'Missing argument'; return 1; }
    declare -F "_restore_$1" > /dev/null ||
    { echo "Invalid environment '$1'"; return 3; }
    _restore_$1
}

#############################################################################
# restore
#############################################################################

_restore_devel()
{
    docker container exec duplicity \
        restore --force
    docker container exec influxdb \
        influx restore --full /var/lib/influxdb2/backup
}

_restore_prod()
{
    local env_file=./env/config/restore_prod.env
    docker container exec --env-file "$env_file" duplicity \
        restore --force
    docker container exec influxdb \
        influx restore --full /var/lib/influxdb2/backup
}

_restore_prod_old()
{
    _tmp_setup

    local ret=0
    local names=("nodered-data" "grafana-data"
                 "influxdb-data" "influxdb-config")
    local prefix_prod=ontec_prod
    local prefix_devel=ontec_devel
    for name in ${names[@]}
    do
        local volume_prod="${prefix_prod}_${name}"
        local volume_devel="${prefix_devel}_${name}"
        echo "Cloning volume $name"
        sudo DOCKER_HOST="ssh://minipc@192.168.0.1" \
```

```
        dockext volume get "$volume_prod" "$name" &&
        sudo DOCKER_HOST="unix:///var/run/docker.sock" \
        dockext volume put "$volume_devel" "$name" ||
        { ret=$?; break; }
    done

    _tmp_cleanup
    return $ret
}

##############################################################################
# tmp
##############################################################################

_tmp_setup()
{
    trap '_tmp_trap $? $LINENO' ERR
    tmp_dir=$(mktemp -d)
    cd "$tmp_dir"
}

_tmp_cleanup()
{
    cd /tmp
    rm -rf "$tmp_dir"
}

_tmp_trap()
{
    set +e
    echo "Exit status $1 at line $2"
    _tmp_cleanup
    exit 1
}

declare -F $1 > /dev/null || exit 3
"$@"
```

B.4 Remote

B.4.1 docker-compose.override.yml

```yaml
services:
    nodered:
        image: nodered/node-red:2.2.2
        ports:
            - "1880:1880"

    grafana:
        image: grafana/grafana:8.2.6
        ports:
            - "3000:3000"

    influxdb:
        image: influxdb:2.1
        ports:
            - "8086:8086"

    mosquitto:
        image: eclipse-mosquitto:2.0.14
        command: mosquitto -c /mosquitto-no-auth.conf
        ports:
            - "1883:1883"

    duplicity:
        image: ghcr.io/tecnativa/docker-duplicity-docker-s3
        hostname: prod-backup

    duckdns:
        image: lscr.io/linuxserver/duckdns
        env_file:
            - ./env/config/duckdns.env
        restart: always
        container_name: duckdns

    portainer:
        image: portainer/portainer-ce:2.9.3
        command: --admin-password "${PORTAINER_PASSWORD}"
        volumes:
            - /var/run/docker.sock:/var/run/docker.sock
            - portainer-data:/data
        ports:
            - "8000:8000"
            - "9000:9000"
        restart: always
        container_name: portainer

volumes:
    portainer-data:
```

B.4.2 *.env environment files

.env

```
PORTAINER_PASSWORD=<REDACTED>
```

influxdb.env

```
DOCKER_INFLUXDB_INIT_MODE=setup
DOCKER_INFLUXDB_INIT_USERNAME=admin
DOCKER_INFLUXDB_INIT_PASSWORD=<REDACTED>
DOCKER_INFLUXDB_INIT_ORG=ontec
DOCKER_INFLUXDB_INIT_BUCKET=data
TMPDIR=/var/lib/influxdb2
```

grafana.env

```
GF_SECURITY_ADMIN_PASSWORD=<REDACTED>
GF_AUTH_ANONYMOUS_ENABLED=true
GF_AUTH_ANONYMOUS_ORG_ROLE=Viewer
```

duplicity.env

```
# Duplicity
DST=file:///mnt/backup/dst
PASSPHRASE=devel
# Mail
SMTP_HOST=mailhog
SMTP_PORT=1025
EMAIL_FROM=backup@devel.com
EMAIL_TO=admin@devel.com
TZ=Europe/Madrid
# InfluxDB job
JOB_190_WHAT="rm -f /mnt/backup/src/influxdb-backup/*"
JOB_190_WHEN="daily weekly"
JOB_200_WHAT="docker container exec influxdb influx backup
↪  /var/lib/influxdb2/backup"
JOB_200_WHEN="daily weekly"
```

duckdns.env

```
TOKEN=<REDACTED>
TZ=Europe/Madrid
SUBDOMAINS=<REDACTED>
```

B.4.3 internal.sh

```
#!/bin/bash

set -euo pipefail
[[ -v DEBUG ]] && set -x

restore()
{
    [[ -n "${1:-}" ]] ||
    { echo 'Missing argument'; return 1; }
    declare -F "_restore_$1" > /dev/null ||
    { echo "Invalid environment '$1'"; return 3; }
    _restore_$1
}

###############################################################################
# restore
###############################################################################

_restore_prod()
{
    docker container exec duplicity \
        restore --force
    docker container exec influxdb \
        influx restore --full /var/lib/influxdb2/backup
}

declare -F $1 > /dev/null || exit 3
"$@"
```