



Escola d'Enginyeria de Telecomunicació i  
Aeroespacial de Castelldefels

UNIVERSITAT POLITÈCNICA DE CATALUNYA

# FINAL DEGREE PROJECT

**TITLE:** Design, planning, deployment and operation of a learning platform

**DEGREE:** Bachelor's degree in Network Engineering

**AUTHOR:** Miguel Mateos Luque

**DIRECTOR:** Toni Oller Arcas

**DATE:** September 8, 2023

**Title:** Design, planning, deployment and operation of a learning platform

**Author:** Miguel Mateos Luque

**Director:** Toni Oller Arcas

**Date:** September 8, 2023

## Overview

This work explores in depth the effective management of Odoo-based systems in educational and business environments, with a special focus on the experience of the aUPaEU educational project. Odoo, an open source business management system, has proven to be an essential tool for managing a wide range of business processes. Successful implementation of Odoo involves sound management and appropriate approaches to critical issues such as backup, version migration and continuous monitoring.

This work addresses these fundamental aspects of Odoo system administration. For backups, it proposes the use of Minio, a scalable cloud storage solution that ensures the integrity of enterprise data. Version migration is addressed through the use of OpenUpgrade, a tool that automates this complex process and minimises the associated risks. In terms of system monitoring, a set of tools including Prometheus, Grafana and Loki are used, enabling constant and effective control of the Odoo infrastructure.

In the context of aUPaEU, this work also examines how these solutions and best practices are specifically applied to the management of Odoo systems in education. It highlights how aUPaEU has used these tools to improve the efficiency and reliability of its systems, resulting in a more robust user experience and more effective management of educational resources.

The work not only presents these tools, but also highlights best practices for their successful implementation in the Odoo environment, with a focus on how these practices benefit aUPaEU. In addition, future directions are explored to further improve Odoo systems management and its impact on aUPaEU, making it a valuable resource for both working professionals and students venturing into this ever-evolving field.

Ultimately, this work makes a contribution to the field of Odoo systems management by providing guidance and essential tools that respond to the evolving needs of companies and organisations using Odoo to drive their business operations, including educational cases such as aUPaEU.

**Título:** Diseño, planificación, despliegue y operación de una plataforma de aprendizaje

**Autor:** Miguel Mateos Luque

**Director:** Toni Oller Arcas

**Fecha:** 8 de setiembre de 2023

## Resumen

Este trabajo, centrado en la gestión eficaz de sistemas basados en Odoo en los sectores educativo y comercial, pone de relieve la experiencia del proyecto educativo aUPaEU. El sistema de gestión empresarial de código abierto Odoo se ha consolidado como una herramienta crucial para supervisar diversas operaciones corporativas.

En este trabajo, se presentan facetas importantes de la administración de sistemas Odoo, incluidas las copias de seguridad, la migración de versiones y la supervisión continua. Para las copias de seguridad, se propone el uso de Minio, una solución escalable de almacenamiento en la nube. La migración de versiones se aborda mediante el empleo de OpenUpgrade, una herramienta que automatiza este proceso complejo. En cuanto a la monitorización del sistema, se recurre a herramientas como Prometheus, Grafana y Loki para un control constante y eficaz de la infraestructura de Odoo.

En el contexto de aUPaEU, este trabajo examina cómo estas soluciones y mejores prácticas se aplican específicamente a la gestión de sistemas Odoo en el ámbito educativo, mejorando la eficiencia y confiabilidad de sus sistemas.

El trabajo no solo presenta estas herramientas, sino que también subraya las mejores prácticas para su implementación exitosa en el entorno Odoo, con un enfoque en cómo benefician a la aUPaEU. Además, se exploran direcciones futuras para mejorar aún más la gestión de sistemas Odoo y su impacto en aUPaEU.

En última instancia, este trabajo hace una contribución al campo de la gestión de sistemas Odoo, al proporcionar una guía y herramientas esenciales que responden a las necesidades cambiantes de las empresas y organizaciones que utilizan Odoo para impulsar sus operaciones comerciales, incluyendo casos educativos como el de aUPaEU.

# Table of contents

<b>INTRODUCTION</b>	<b>1</b>
<b>CHAPTER 1. CONTEXT</b>	<b>3</b>
1.1. aUPaEU	3
1.2. Odoo	4
1.2.1. OCA	4
1.3. Objectives	5
<b>CHAPTER 2. ARCHITECTURE</b>	<b>6</b>
2.1. Odoo requirements	6
2.2. aUPaEU requirements	7
2.3. Resources	8
<b>CHAPTER 3. ODOO INSTALLATION</b>	<b>9</b>
3.1. Installation	9
3.2. Docker based installation	10
3.2.1. Automation	14
3.3. Addons Installation	15
3.3.1. Automation	17
<b>CHAPTER 4. ODOO DEPLOYMENT</b>	<b>20</b>
4.1. Deployment	20
4.1.1. Reverse proxy	21
4.1.2. HTTPS	21
4.1.3. Docker networking	22
4.2. Odoo deployment	22
4.2.1. Automation	25
4.2.2. Addressing infrastructure problems	28
<b>CHAPTER 5. ODOO BACKUP</b>	<b>30</b>
5.1. Backup	30
5.2. Odoo backup	30
5.2.1. Automation	31
5.3. MinIO	32
5.3.1. Automation	34
<b>CHAPTER 6. ODOO UPGRADE</b>	<b>36</b>
6.1. Upgrade	36
6.1.1. OpenUpgrade	36
6.2. Odoo upgrade	37
6.2.1. Automation	41
<b>CHAPTER 7. ODOO MONITORING</b>	<b>43</b>
7.1. Monitoring	43
7.1.1. Metrics	44
7.1.2. Logs	49
7.1.3. Alerts	52
<b>CHAPTER 8. CONCLUSIONS AND NEXT STEPS</b>	<b>57</b>
8.1. Conclusions	57

8.2. Next steps	58
<b>REFERENCES</b>	<b>59</b>
<b>BIBLIOGRAPHY</b>	<b>63</b>

## List of tables and figures

<b>Table 2.1</b> Odoo hardware requirements based on active users	6
<b>Fig. 2.1</b> Scenario diagram	8
<b>Fig. 3.1</b> Commands for Docker installation	10
<b>Fig. 3.2</b> Configuration file "odoo.conf"	11
<b>Fig. 3.3</b> Environment file ".env"	11
<b>Fig. 3.4</b> Docker compose file "docker-compose.yml"	12
<b>Fig. 3.5</b> Commands for Docker management	13
<b>Fig. 3.6</b> Odoo initial configuration screen	13
<b>Fig. 3.7</b> Updated configuration file "odoo.conf"	14
<b>Fig. 3.8</b> Odoo login screen	14
<b>Fig. 3.9</b> Odoo Apps Screen	15
<b>Fig. 3.10</b> Odoo "Developer Mode" activation button Screen	15
<b>Fig. 3.11</b> Odoo "Update Apps List" button Screen	16
<b>Fig. 3.12</b> Command for restart Odoo container	16
<b>Fig. 3.13</b> Updated configuration file "odoo.conf"	17
<b>Fig. 3.14</b> Command to recreate a container	17
<b>Fig. 3.15</b> Commands for Python addon installation	17
<b>Fig. 3.16</b> Execute the script from CLI	18
<b>Fig. 4.1</b> Overall picture of the system	20
<b>Fig. 4.2</b> Dockerfile "nginx-proxy.Dockerfile"	23
<b>Fig. 4.3</b> Updated Docker compose file "docker-compose.yml"	23
<b>Fig. 4.4</b> Current scenario on an unrestricted server	24
<b>Fig. 4.5</b> Secure connection check screen on the server	24
<b>Fig. 4.6</b> Command to recreate the acme container	25
<b>Fig. 4.7</b> GitHub Actions usage example	25
<b>Fig. 4.8</b> Commands to configure a runner	26
<b>Fig. 4.9</b> Workflow configuration file "deploy.yml"	27
<b>Fig. 4.10</b> GitHub Action workflow status screen	27
<b>Fig. 4.11</b> Pre-production server console screen activating runner daemon	28
<b>Fig. 4.12</b> GitHub repository runner status screen	29
<b>Fig. 5.1</b> Odoo database management screen	31
<b>Fig. 5.2</b> Parameter configuration of "Database auto-backup"	32
<b>Fig. 5.3</b> MinIO main screen	33
<b>Fig. 5.4</b> Basic AWS CLI commands	34
<b>Fig. 5.5</b> Parameter configuration of the modified "Database auto-backup"	34
<b>Fig. 6.1</b> OpenUpgrade Flow	37
<b>Fig. 6.2</b> Screen of the module coverage on the OpenUpgrade page	38
<b>Fig. 6.3</b> Odoo database management screen with an error in one database	39
<b>Fig. 6.4</b> Command to install openupgradelib	39
<b>Fig. 6.5</b> Commands to run OpenUpgrade	39
<b>Fig. 6.6</b> Commands to run OpenUpgrade	40
<b>Fig. 6.7</b> Console error screen after running OpenUpgrade	40
<b>Fig. 7.1</b> Display of cAdvisor graphs	41
<b>Fig. 7.2</b> Configuration file "prometheus-config.yml"	45
<b>Fig. 7.3</b> Display of cAdvisor exported metrics	46
<b>Fig. 7.4</b> Screen of the Prometheus targets	47
<b>Fig. 7.5</b> cAdvisor dashboard screen in Grafana	48
<b>Fig. 7.6</b> Odoo dashboard screen in Grafana	48
<b>Fig. 7.7</b> Postgres dashboard screen in Grafana	48
<b>Fig. 7.8</b> Prometheus Flow	50
<b>Fig. 7.9</b> Configuration file "promtail-config.yml"	50

<b>Fig. 7.10</b> Display of Loki metrics	51
<b>Fig. 7.11</b> Loki Explore screen in Grafana	52
<b>Fig. 7.12</b> Configuration file "alertmanager-config.yml"	53
<b>Fig. 7.13</b> Updated configuration file "prometheus-config.yml"	53
<b>Fig. 7.14</b> Alerts configuration file "alert-rules.yml"	54
<b>Fig. 7.15</b> Alertmanager main screen	54
<b>Fig. 7.16</b> Alert screen received by mail	55
<b>Fig. 7.17</b> Prometheus main screen	55
<b>Fig. 7.18</b> Grafana alerting screen	56

# INTRODUCTION

The successful administration of business systems has become essential for the success of organisations of all kinds in a world that is becoming more and more technologically oriented. From project management to accounting to human resource management, Odoo stands out in this environment as a flexible, open source option that covers a wide variety of corporate applications. Due to its versatility and flexibility to different demands, this software package has become more and more popular in both corporate and educational settings.

This work focuses on the management of Odoo systems in the context of aUPaEU (A University Partnership for Acceleration of European Universities), an innovative educational platform that has adopted Odoo to manage its academic and administrative operations. Throughout the following pages, key practices and strategies to ensure the successful deployment and efficient maintenance of Odoo systems in education and business environments will be explored, with a particular focus on aUPaEU.

The work is divided into eight parts:

- **Context:** This chapter sets the context of the work, presenting Odoo as a key tool in the aUPaEU project, and further defines the objectives.
- **Architecture:** This chapter focuses on understanding the Odoo architecture and its requirements. It mentions the hardware and software requirements for Odoo and explores the specific needs of aUPaEU in terms of infrastructure.
- **Initial configuration and deployment:** It begins by exploring the process of initial configuration and deployment of Odoo in the aUPaEU environment, highlighting the key design and configuration decisions that drive the success of the system.
- **Backup Strategies:** Data security is an undisputed priority in managing Odoo systems. Backup strategies are discussed in detail, including the implementation of Minio as a cloud storage solution.
- **Version Migration:** Upgrading Odoo to newer versions is essential to take advantage of the latest features and security fixes. A detailed approach to version migration using OpenUpgrade is presented.
- **Monitoring Tools:** In an ever-changing world, constant monitoring is essential. The use of tools such as Prometheus, Grafana and Loki to maintain optimal performance in aUPaEU Odoo systems is explored.
- **Conclusions and next steps:** In this section, possible future directions for the administration of Odoo systems in aUPaEU are discussed and the main conclusions drawn from the work are given.



- Bibliography and References: The list of references and bibliographic sources utilised in the work is provided at the end.

Throughout this work, the focus has been on contributing to the continued success of aUPaEU and other educational and business environments that have adopted Odoo as their system of choice. With the goal of sharing best practices and valuable insights, this work serves as a resource for practitioners and students interested in the efficient management of Odoo systems in similar contexts.

## CHAPTER 1. CONTEXT

This chapter lays the foundation for the work, presenting an overview of Odoo and its importance in business environments. In addition, the contribution of the Odoo Community (OCA) is examined, and the overall goals of the project are established. This chapter's goal is to provide a solid understanding of the context in which this work is situated.

### 1.1. aUPaEU

The EU-funded aUPaEU (A University Partnership for Acceleration of European Universities) project [1] is based on the concept of the ancient Greek agora, a meeting place where citizens exchanged ideas and formed alliances. Today, it seeks to create a shared space where diverse stakeholders can offer and receive services while sharing their knowledge.

This project involves the collaboration of two European alliances, EPiCUR and Unite! working together to integrate and provide support services in the context of higher education. These institutions have experience in modernising research and innovation and wish to serve as an example for other institutions, networks and university alliances.

European University Alliances [2] are transforming higher education in Europe through transnational collaboration. These alliances, made up of diverse institutions across Europe, focus on sustainability, excellence and long-term European values. They offer joint student programmes on multiple campuses, promoting student mobility and an interdisciplinary approach to tackling European challenges.

The main objective of aUPaEU is to develop an acceleration space where higher education institutions, university networks and alliances can achieve lasting transformations in key areas of education and research. This includes sharing resources, enhancing researchers' careers, collaborating with diverse research actors, promoting open science, contributing to society and promoting gender equality.

In essence, the project seeks to create an inclusive and accessible space where all stakeholders can collaborate and benefit from each other. The success of this agora will serve as an example for other institutions and university alliances wishing to undertake similar initiatives.

The basis of this agora is going to be Odoo.

## 1.2. Odoo

Odoo [3] is an open source software platform used for managing businesses in a number of sectors, including e-commerce, customer relationship management (CRM), sales, inventories, and more. Organisations of all sizes and sectors find it to be particularly appealing because to its adaptability and versatility.

The open source edition of Odoo that is free is referred to as the community version or Odoo Community Edition. Small and medium-sized organisations as well as bigger organisations can benefit from its extensive set of features and functions. The corporate edition of Odoo, which costs money and provides greater support and capabilities, is an option for businesses as they expand and need more sophisticated features or particular customizations.

Odoo stands out for its adaptability and flexibility in meeting the individual demands of every business. To do this, "addons" are used, which are modules that may be added or withdrawn to increase the capability of Odoo. These add-ons enable significant customization and can be created by the Odoo user community or by other developers.

In the context of aUPaEU, Odoo is used as a central tool to optimise and manage various operations and processes. Its versatility allows aUPaEU to tailor Odoo to its specific needs, including efficient resource management, automation of administrative tasks and improved decision making. In addition, Odoo's ability to manage multiple languages and adapt to country-specific regulations makes it a suitable solution for an international organisation such as aUPaEU.

### 1.2.1. OCA

The open source company management software Odoo is promoted and developed by the OCA (Odoo Community Association) [4], a nonprofit organisation. A diverse variety of modules and extensions are developed and maintained by its community of users, which is made up of Odoo developers, consultants, and fans, to increase the software's functionality in fields including accounting, human resources, e-commerce, and logistics. The OCA focuses on maintaining high quality standards and offers these open source resources free of charge to benefit the Odoo community. It also serves as a meeting and collaboration point for the community, providing forums and resources for knowledge sharing and problem solving related to Odoo.

Odoo communities [5] around the world are organised into country and region-specific groups to adapt and localise Odoo according to local needs, such as legal regulations and accounting. This allows Odoo to be highly adaptable and customisable, which is essential for use in different locations. These groups are essential to Odoo's international growth since they make it simpler for businesses to operate efficiently and adhere to local laws.

### 1.3. Objectives

The following aims can be used to describe this document:

- The first objective is to gain an in-depth understanding of Odoo as an open source business management platform, including its structure, features and flexibility.
- Implement Odoo in aUPaEU: Apply the acquired knowledge to effectively implement Odoo in the aUPaEU organisation, taking advantage of its versatility to improve the management of educational resources and processes.
- Automation and Efficiency: Develop automation strategies for Odoo installation, deployment and backup, with the aim of improving operational efficiency.
- Migration to New Versions: Establish an efficient migration process to newer versions of Odoo using OpenUpgrade, ensuring that aUPaEU is always up to date with the latest functionality.
- Monitoring and Control: Implement monitoring tools to capture metrics, logs and alerts, improving visibility and control of aUPaEU's Odoo systems.
- Community Contribution: Share acquired knowledge and developed solutions with the Odoo community, contributing to the growth and continuous improvement of this open source platform.

By addressing these objectives, it will contribute to the success of aUPaEU by leveraging Odoo as an effective tool for the management and optimisation of its educational and administrative operations.

## CHAPTER 2. ARCHITECTURE

Effective Odoo management starts with a clear understanding of its architecture. This chapter focuses on the requirements and environments needed to implement Odoo. It explores the current scenario, and details the connection to the infrastructure, which is essential for a successful implementation.

### 2.1. Odoo requirements

The infrastructure required to support an alliance of up to 44 alliances using Odoo would depend on a variety of factors, such as the size of the user base, the number of modules and applications being used, and the amount of data being generated.

Odoo is a straightforward system [6]. A 2 CPU 2 RAM server would be adequate for businesses with 5 employees, while for those with 20 employees, a 4 CPU 8 RAM server would be required. For 90 employees, it is advisable to separate application and database servers. A corporation with more than 250 employees would require load balancing (LB) of the application server. A summary can be found in Table 2.1.

**Table 2.1** Odoo hardware requirements based on active users

Active users	Hardware	RAM
10	2	2
50	4	16
100	16	32
150	25	128
200	32	128
250	LB + 2×12 CPU	128
300	LB + 2×12 CPU	128
350	LB + 2×32 CPU	256
400	LB + 2×64 CPU	256
450	LB + 2×64 CPU	256
500	LB + 2×64 CPU	256

The size of the database and the number of modules to be installed will determine the amount of storage required. For the program and database files, it is often advised to have at least 10 GB of free disc space. In addition, 2-4 GB per active user should be considered.

On the software side, deploying Odoo requires careful planning of operating systems, databases and other software components. It is essential to choose an operating system that is compatible with Odoo and provides good performance, such as Ubuntu, which is suggested by the business. Furthermore, databases like PostgreSQL need to be set up and optimised to provide effective data management and high availability.

## 2.2. aUPaEU requirements

What is to be implemented will be determined once some numerical calculations are made. There is a macro-project initially consisting of two associations, but there are many options for growth.

The basic thing to define is a Postgres instance for each Odoo instance, leaving no room for doubt. The next step is whether to separate each alliance into a different instance, or to somehow group them into one, which considering that they need to be kept in sync in case of software changes, the second option is the most viable. Now, everything is in a single set of Postgres plus Odoo, but must be separated in some way. Here the options are to use, within the same Postgres, different databases, or to apply a multi-company, where this option is the winner, because although both options allow customising each alliance (in this case), a multi-company [7] allows sharing data and configurations between alliances, as may be the case of users belonging to several, but the most important thing is still the fact of not having to be synchronising software changes, since several databases are just that, different.

Even if there is only a single server for the entire project, this only serves as a precedent for applying vertical scaling techniques, initially, and horizontal scaling techniques, to offer greater availability.

In addition to all this, there is the need to keep backups, which should not be on the same server. In the case of not using an external service, a dedicated server should be added to store the backups.

So far, it has been discussed what is basically the minimum for an end-user, but this is preceded by a series of developments. To be up to date and to include improvements in an Odoo instance, it is not possible to work on the same server as the users. Three environments are defined:

- Local / Development: This is the physical machine of each developer, or failing that, an own server, which is rarely necessary. Here the developers will create and test the new contributions they will make on Odoo.
- Pre-production: An own server, where the changes from development will be uploaded, and everything will be checked to make sure that everything works as it should.
- Production: Another own server, where the final users are located. The changes will be uploaded from pre-production, already verified, and any errors that may arise will be followed up.

## 2.3. Resources

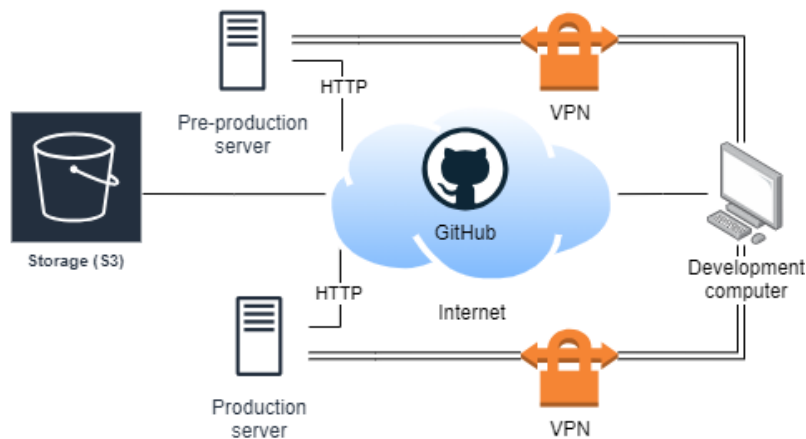
For the realisation of this project, the local environment consists of a virtualised installation of Ubuntu 22.04.2 LTS, to which 2 cores of a consumer CPU, 4 GB of ram and 50GB of storage have been assigned. As for the OS, the choice is Ubuntu Desktop 22.04 LTS.

For the rest, the infrastructure is provided by IThinkUPC [8]. The servers currently available are virtualised, in this case using VMware ESXi. Each server has been assigned a socket, which in turn only contains one core, which in turn only has one thread, all coming from an Intel(R) Xeon(R) Platinum 8280 CPU with a base frequency of 2.70GHz. 4GB of RAM and 512GB of storage space are available. The operating system is Ubuntu Server 22.04.02 LTS.

In addition, several domains have been registered for use with these machines, including: "aupaeu.pre.upc.edu" and "aupaeu.upc.edu".

The fact that accessing this infrastructure requires the use of a VPN. These issues will be discussed later with a little more context on basic concepts (see 4.2.2).

The scenario is a 4-server scenario, consisting of a storage server, the two provided servers and the developer's computer. The code exchange between the different environments will be done through a GitHub repository. To access the resources of the provided servers, a VPN tunnel will be established. The resulting scheme is shown in Fig. 2.1.



**Fig. 2.1** Scenario diagram

## CHAPTER 3. ODOO INSTALLATION

Installing Odoo is a critical step in the implementation. This chapter dives into the different installation methodologies available, including installation using Docker. It also discusses the automation of these processes to speed up the deployment and reduce potential errors.

### 3.1. Installation

Odoo can be installed in various ways [9]. The choice of installation method depends on factors like the technical expertise, infrastructure requirements, scalability needs, and deployment preferences. Some different types of installations for Odoo that can be considered are:

- **Source:** The source installation method involves downloading the Odoo source code and manually configuring the system. This approach provides maximum control and customization options, allowing advanced configurations and modifications tailored to specific needs. However, it necessitates technical proficiency and system administration skills. Additionally, this approach might take a while, especially for complicated settings, and managing dependencies manually can be challenging.
- **Package managers:** The package installation method utilizes package managers like apt-get or yum to install Odoo directly from the distribution's repositories. This approach simplifies the installation process by handling dependencies and package management automatically. It offers the advantage of easier updates and management through the package manager. However, it is limited to supported distributions and repositories. Furthermore, as compared to the official version, there could be delays in getting the most recent Odoo upgrades. Additionally, using this in production contexts is not intended.
- **Docker:** The Docker installation method utilizes containerization technology to create an isolated environment for Odoo. It provides portability and scalability, allowing deployment across different environments with ease. Docker [10] simplifies the deployment and management of Odoo instances, offering advantages such as isolation and straightforward application updates. However, this method requires familiarity with Docker and containerization concepts. It introduces additional complexity for managing networking, data persistence, and resource allocation.

The choice of Docker as an installation method for Odoo is justified due to its exceptional portability, scalability and ease of management. Docker allows for the creation of isolated Odoo containers, making it easy to deploy across multiple environments without worrying about differences in the configuration of the underlying system. In addition, the ability to efficiently create and delete containers streamlines upgrades and deployments of new Odoo versions.

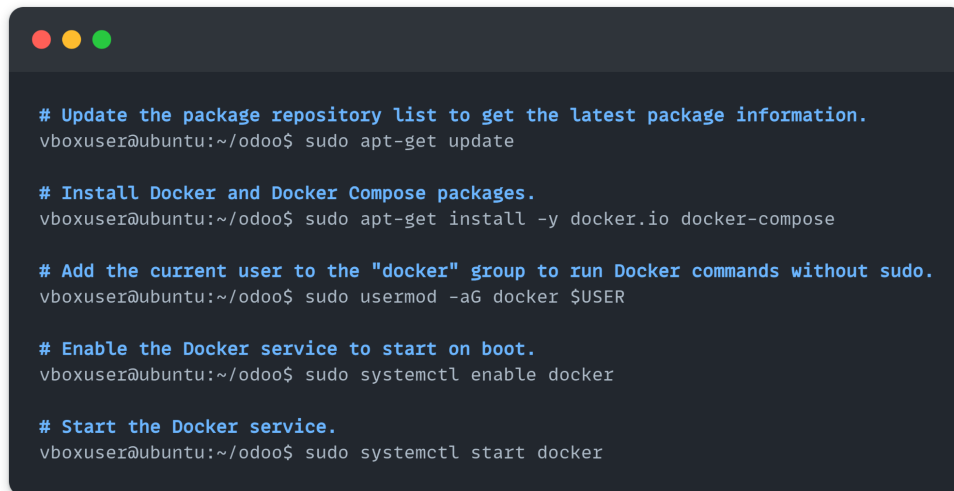


Compared to other methods, such as installing from packages or from source code, Docker offers a more efficient and versatile solution, ideal for enterprise and development environments that require flexibility and efficiency in managing Odoo.

## 3.2. Docker based installation

Everything described in this document is done, tested and working in a real environment. The writing has been done on the basis of experience.

To start working with Docker, the first thing to do is to install it. This involves a few commands (see Fig. 3.1):

A terminal window with a dark background and light text. It shows a series of commands and their outputs for installing Docker on Ubuntu. The commands are: 1. Update package repository: `sudo apt-get update`. 2. Install Docker and Docker Compose: `sudo apt-get install -y docker.io docker-compose`. 3. Add current user to the 'docker' group: `sudo usermod -aG docker $USER`. 4. Enable Docker service to start on boot: `sudo systemctl enable docker`. 5. Start the Docker service: `sudo systemctl start docker`. Each command is preceded by a comment explaining its purpose.

```
# Update the package repository list to get the latest package information.
vboxuser@ubuntu:~/odoo$ sudo apt-get update

# Install Docker and Docker Compose packages.
vboxuser@ubuntu:~/odoo$ sudo apt-get install -y docker.io docker-compose

# Add the current user to the "docker" group to run Docker commands without sudo.
vboxuser@ubuntu:~/odoo$ sudo usermod -aG docker $USER

# Enable the Docker service to start on boot.
vboxuser@ubuntu:~/odoo$ sudo systemctl enable docker


# Start the Docker service.
vboxuser@ubuntu:~/odoo$ sudo systemctl start docker
```

**Fig. 3.1** Commands for Docker installation

Once everything related to Docker is installed, the next step is to define the structure. Organising the files needed for a Docker-based installation [11] [12] is important to be able to maintain the project easily. The basic components are:

- **addons:** The "addons" directory serves as a repository for custom Odoo modules or extensions. Each module can be organised within subdirectories.
- **config:** Configuration files, often placed in the "config" folder, dictate how services behave. These files encompass environment-specific settings such as database connections, security configurations, and logging preferences.

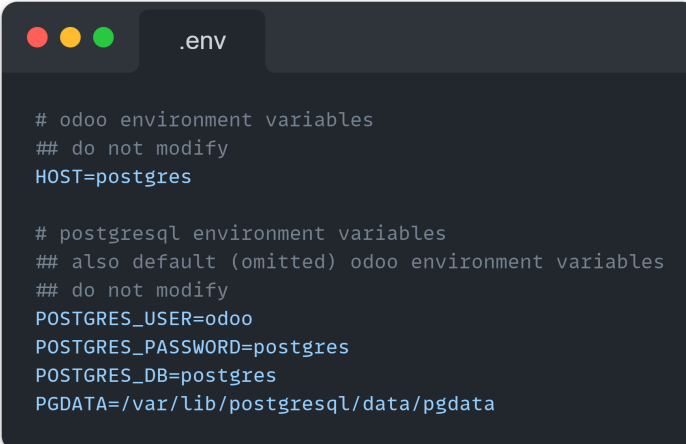
- `odoo.conf`: The "`odoo.conf`" file [13] (see Fig. 3.2) consolidates crucial runtime settings for the Odoo instance. It encompasses database details, security parameters, and more.

A screenshot of a terminal window showing the contents of the `odoo.conf` file. The window title is `odoo.conf`. The content is as follows:

```
[options]
# do not modify
addons_path = /mnt/extra-addons
## postgresql environment variables
db_host = postgres
db_password = postgres
```

**Fig. 3.2** Configuration file "`odoo.conf`"

- `.env`: The `.env` file (see Fig. 3.3) contains environment variables required by Docker Compose or other components. Storing sensitive information like credentials in a separate file enhances security and promotes ease of management across different environments.

A screenshot of a terminal window showing the contents of the `.env` file. The window title is `.env`. The content is as follows:

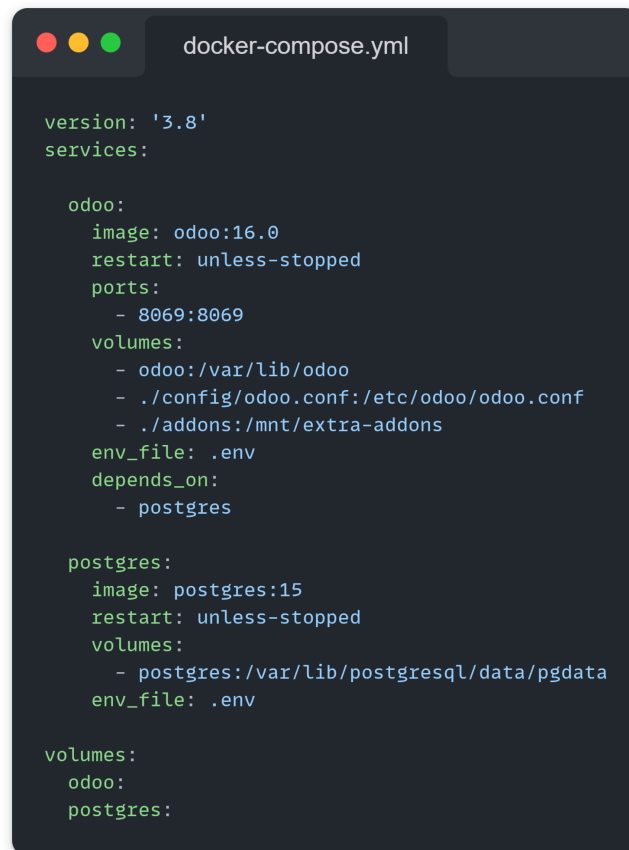
```
# odoo environment variables
## do not modify
HOST=postgres

# postgresql environment variables
## also default (omitted) odoo environment variables
## do not modify
POSTGRES_USER=odoo
POSTGRES_PASSWORD=postgres
POSTGRES_DB=postgres
PGDATA=/var/lib/postgresql/data/pgdata
```

**Fig. 3.3** Environment file "`.env`"

For a basic example, default values have been used, but it is better to modify these values. Note that even if they are not referenced in the Docker composition, all containers can access them if this file is passed, thus saving declarations.

- `docker-compose.yml`: The "Docker Compose" file [14] (see Fig. 2.4) is a core component that orchestrates Docker containers. It defines services, networks, and volumes necessary for smooth collaboration between different containers. This simple file, allows to easily define the environment, without the need to use the command line.

A screenshot of a code editor window titled "docker-compose.yml". The code is written in a dark theme with light green and white text. It defines a Docker Compose configuration for two services: 'odoo' and 'postgres'. The 'odoo' service uses the 'odoo:16.0' image, restarts unless stopped, listens on port 8069, and mounts several volumes including '/var/lib/odoo', './config/odoo.conf:/etc/odoo/odoo.conf', and './addons:/mnt/extra-addons'. It also uses an '.env' file and depends on the 'postgres' service. The 'postgres' service uses the 'postgres:15' image, restarts unless stopped, and mounts a volume for data storage at '/var/lib/postgresql/data/pgdata'. It also uses an '.env' file. A separate 'volumes' section at the bottom lists 'odoo' and 'postgres' as defined volumes.

```
version: '3.8'
services:
  odoo:
    image: odoo:16.0
    restart: unless-stopped
    ports:
      - 8069:8069
    volumes:
      - odoo:/var/lib/odoo
      - ./config/odoo.conf:/etc/odoo/odoo.conf
      - ./addons:/mnt/extra-addons
    env_file: .env
    depends_on:
      - postgres

  postgres:
    image: postgres:15
    restart: unless-stopped
    volumes:
      - postgres:/var/lib/postgresql/data/pgdata
    env_file: .env

volumes:
  odoo:
  postgres:
```

**Fig. 3.4** Docker compose file "docker-compose.yml"

This Docker Compose configuration orchestrates two services: "odoo" for Odoo version 16.0 and "postgres" for PostgreSQL version 15. The Odoo service, accessible at port 8069, mounts volumes for data and configuration files, utilizing environment variables from the `.env` file and depending on the PostgreSQL service. Similarly, the PostgreSQL service uses a named volume for data storage. This setup enables containerized deployment of Odoo and PostgreSQL instances while maintaining data persistence and inter-service coordination.

To manage the Docker-based project, the following commands will be helpful (see Fig. 3.5):

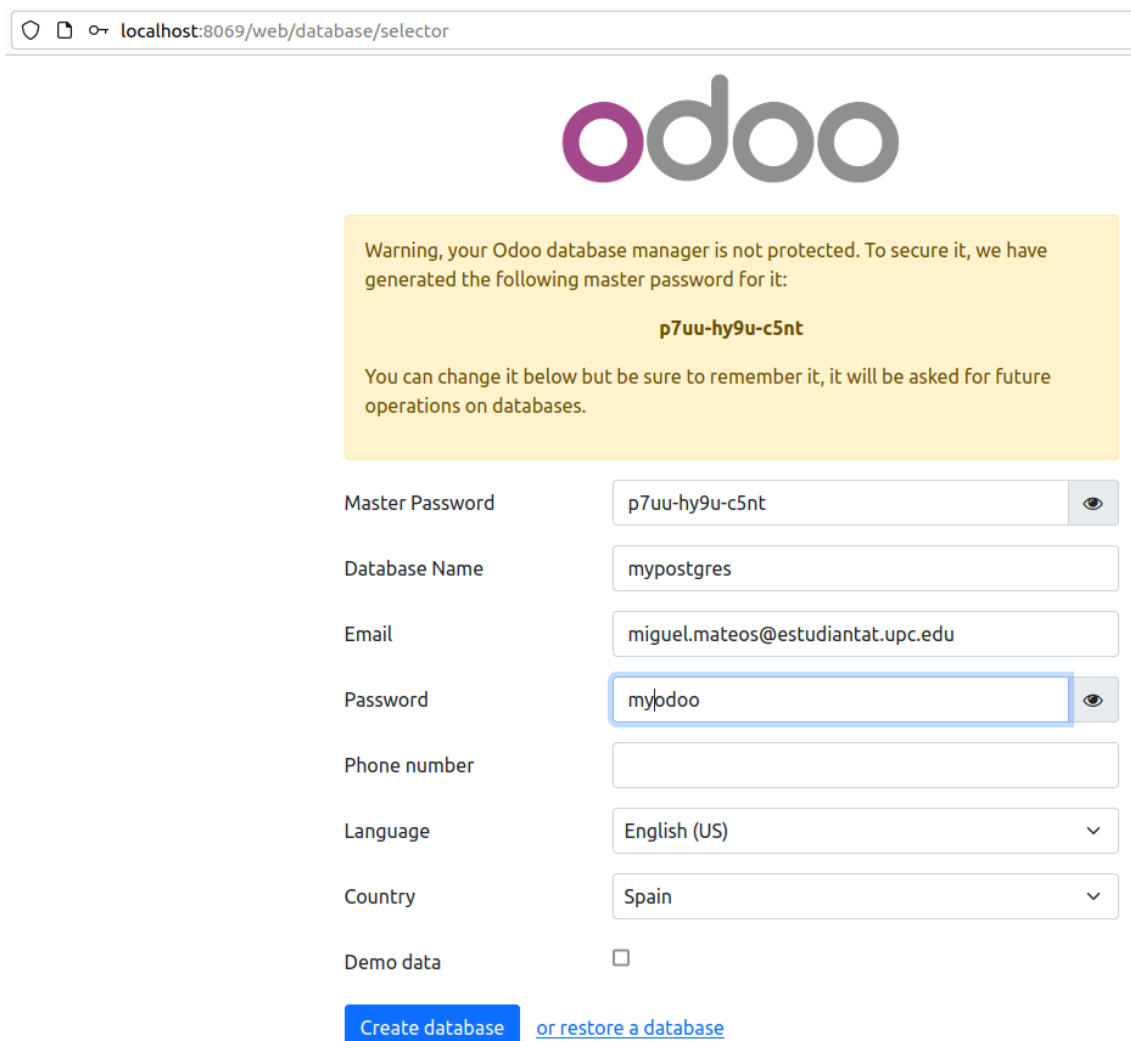
```
# Start the Docker Compose services in detached mode (background).
vboxuser@ubuntu:~/odoo$ docker-compose up -d

# Restart the "odoo" service defined in the Docker Compose configuration.
vboxuser@ubuntu:~/odoo$ docker-compose restart odoo

# Stop all services defined in the Docker Compose configuration.
vboxuser@ubuntu:~/odoo$ docker-compose stop
```

**Fig. 3.5** Commands for Docker management

Finally, by accessing localhost:8069 through the browser (see Fig. 3.6), a new database can be created for the project, or a backup can be imported.



localhost:8069/web/database/selector

# odoo

Warning, your Odoo database manager is not protected. To secure it, we have generated the following master password for it:

**p7uu-hy9u-c5nt**

You can change it below but be sure to remember it, it will be asked for future operations on databases.

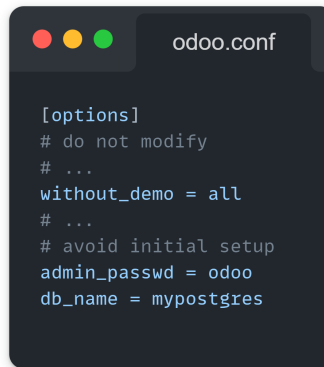
Master Password	<input type="text" value="p7uu-hy9u-c5nt"/>
Database Name	<input type="text" value="mypostgres"/>
Email	<input type="text" value="miguel.mateos@estudiantat.upc.edu"/>
Password	<input type="text" value="myodoo"/>
Phone number	<input type="text"/>
Language	<input type="text" value="English (US)"/>
Country	<input type="text" value="Spain"/>
Demo data	<input type="checkbox"/>

[Create database](#) [or restore a database](#)

**Fig. 3.6** Odoo initial configuration screen

### 3.2.1. Automation

Installing and setting up Odoo is relatively straightforward. Once Docker is installed, the files are generated, and Docker Compose is up, it's all done. In the case of a completely new installation, there are a series of parameters that can be added to "odoo.conf" that can avoid a step (see Fig. 3.7):



```
odoo.conf

[options]
# do not modify
# ...
without_demo = all
# ...
# avoid initial setup
admin_passwd = odoo
db_name = mypostgres
```

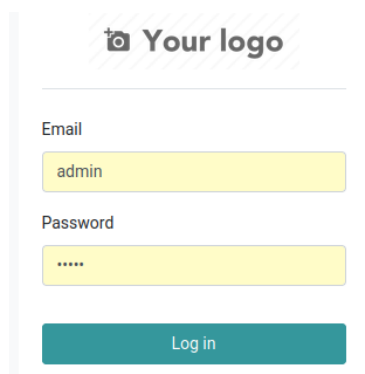
**Fig. 3.7** Updated configuration file "odoo.conf"

The first parameter is used to establish not to add test data.

Regarding credentials, on the one hand, there is the master password (admin\_passwd), which is used to manage the entire Odoo instance, which for security reasons cannot be "admin". On the other hand, there are the user credentials, which are specific to each database, and should not be confused with the Postgres connection user. When the initial setup is avoided in this way, a generic user is generated with "admin" and "admin" credentials, which are highly recommended to change.

With "db\_name", besides creating a new database, if it does not exist, it is defining the default database, so it will not give the option to change the database, if this parameter is set.

This leads directly to the login page (see Fig. 3.8).



The screenshot shows the Odoo login interface. At the top, there is a logo placeholder with the text "Your logo". Below this, there are two input fields: "Email" with the value "admin" and "Password" with the value "\*\*\*\*\*". At the bottom, there is a teal "Log in" button.

**Fig. 3.8** Odoo login screen

### 3.3. Addons Installation

Odoo has a section called "Apps", where one can find its "modules", base list and the option to install them (see Fig. 3.9).

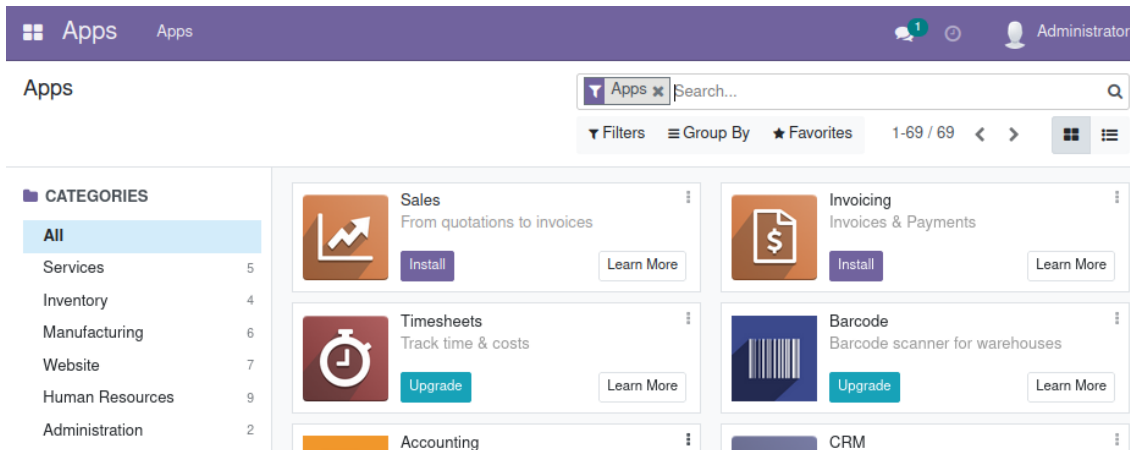


Fig. 3.9 Odoo Apps Screen

If addons (custom modules) need to be installed, a search can be done in the Odoo Apps Store [15], GitHub, or Google itself. The result of the download should be a folder that should be saved in the "addons" folder.

To be able to install addons, as it does not appear by default, "Developer tools" must be activated, available at the bottom of the configuration tab (see Fig. 3.10).

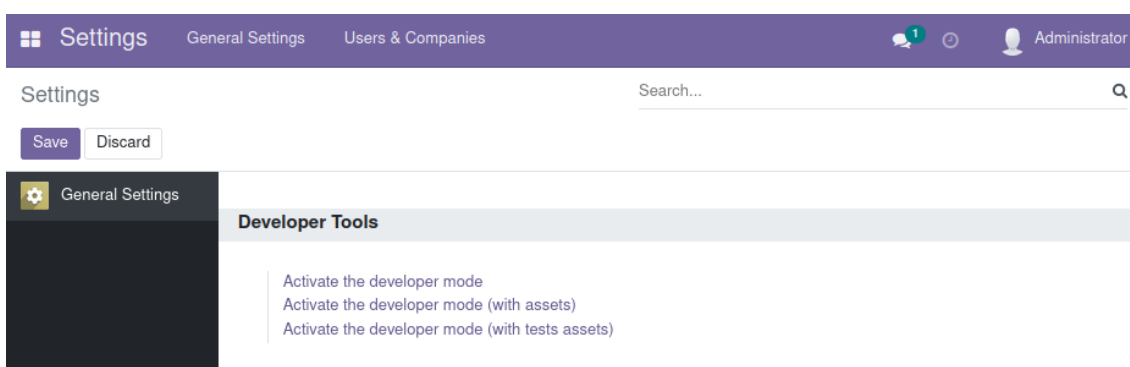
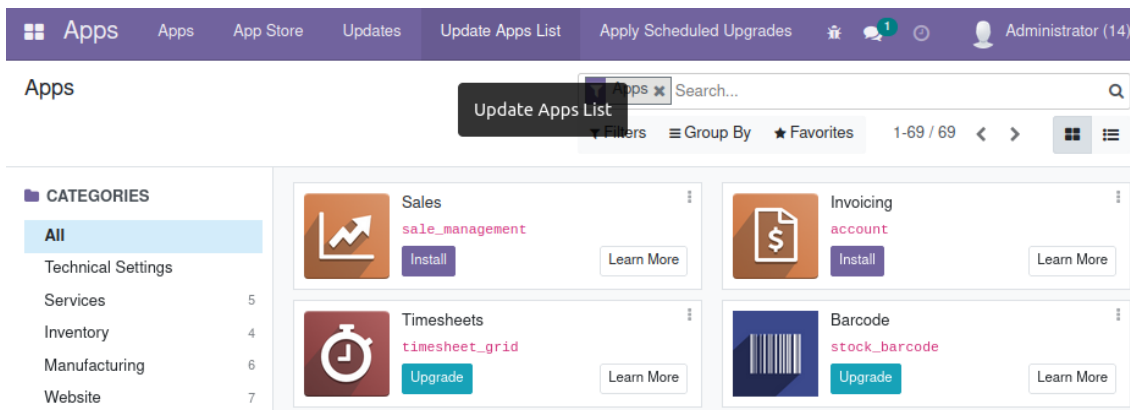


Fig. 3.10 Odoo "Developer Mode" activation button Screen

This will enable, among many other options, "Refresh App List", and addons will now be listed and can be installed in the same way as modules (see Fig. 3.11).



**Fig. 3.11** Odoo “Update Apps List” button Screen

In case of addon updates, if they only involve changes in the "XML" view, they will be detected automatically, but if they involve changes in the functions, it will be necessary to restart the Odoo container (see Fig. 3.12).

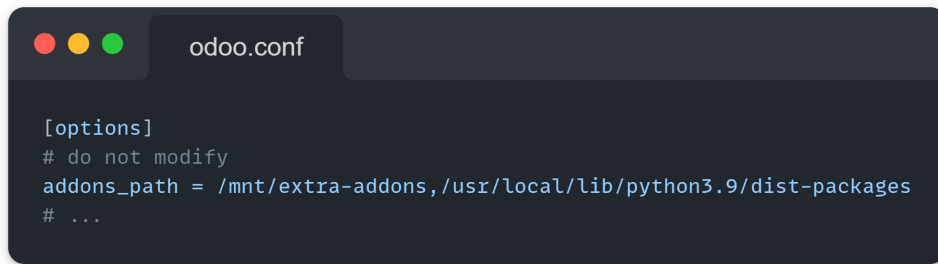
```
# Restart the "odoo" service to apply the changes
vboxuser@ubuntu:~/odoo$ docker-compose restart Odoo
```

**Fig. 3.12** Command for restart Odoo container

There is also a way to install addons, which may seem easier at first, is to install Python packages. One thing to note here is that addons are installed in the Odoo instance, and not in the database, so it is outside the usual flow, so they should be treated as such.

The Python ecosystem has a package management system known as "pip", which already comes with Odoo, since it is also based on Python, with which the addons will be installed. "pip" consumes from a repository of Python ecosystem software known as "PyPI", where it is also possible to search graphically (on its website) for the addons needed. Most of the OCA modules [16] are also found in this repository.

First, the path where these packages are installed must be added in the "odoo.conf", so that Odoo can recognise them (see Fig. 3.13):

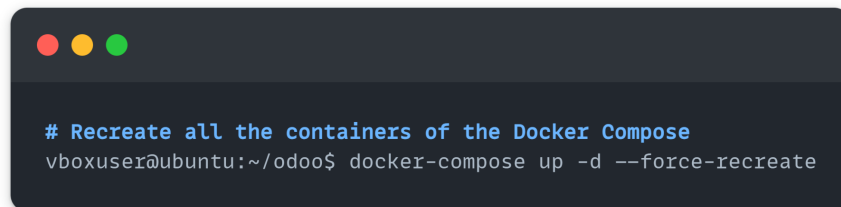
A terminal window showing the contents of the 'odoo.conf' file. The file contains configuration options for addons\_path and other settings.

```
odoo.conf

[options]
# do not modify
addons_path = /mnt/extra-addons,/usr/local/lib/python3.9/dist-packages
# ...
```

**Fig. 3.13** Updated configuration file "odoo.conf"

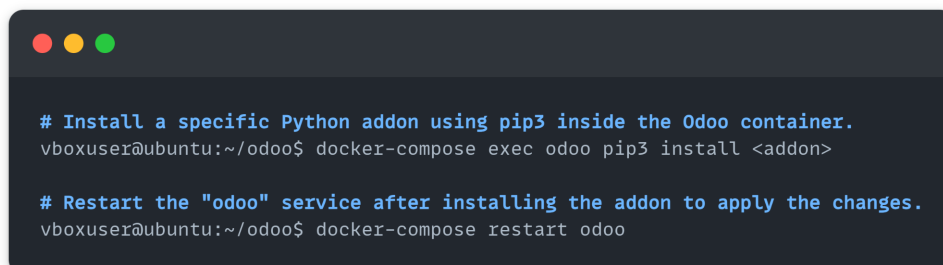
For the changes to be applied, the Odoo container must be recreated (see Fig. 3.14):

A terminal window showing a command to recreate all Docker Compose containers.

```
# Recreate all the containers of the Docker Compose
vboxuser@ubuntu:~/odoo$ docker-compose up -d --force-recreate
```

**Fig. 3.14** Command to recreate a container

Now the desired addon has to be installed, and the container has to be restarted to reflect the changes (see Fig. 3.15):

A terminal window showing two commands: one to install a Python addon inside the Odoo container and another to restart the service.

```
# Install a specific Python addon using pip3 inside the Odoo container.
vboxuser@ubuntu:~/odoo$ docker-compose exec odoo pip3 install <addon>

# Restart the "odoo" service after installing the addon to apply the changes.
vboxuser@ubuntu:~/odoo$ docker-compose restart odoo
```

**Fig. 3.15** Commands for Python addon installation

### 3.3.1. Automation

Automation can already be applied at this point. A number of scripts have been developed that will help automate many tasks that will be mentioned throughout this document.



These scripts are written in Bash, except for a couple that are in Python, due to the need to use modules (Python in this case, not to be confused with Odoo's). Given their length, only what they mainly do will be discussed. They can be downloaded from the GitHub repository [17], and added to a new folder called "scripts" in the root of the project.

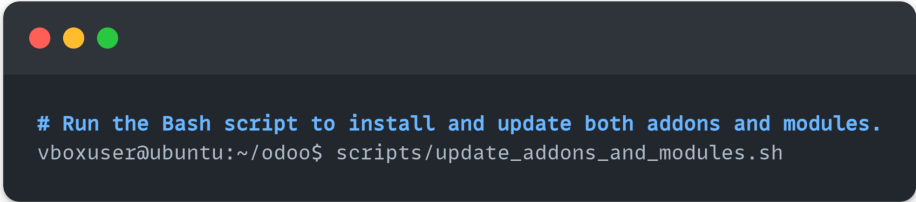
For this occasion, there are two scripts, "update\_addons\_and\_modules.sh" which in turn calls "get\_addons\_requirements.py".

These scripts are supported by two text files that must be created inside the "addons" folder:

- `extra_modules.txt`: Includes the name of the addons (PyPI packages), one per line, that are to be installed. For OCA modules, until Odoo version 14, in PyPI they were listed as "odooXX-...", so in important version changes, it should be taken into account. From version 15 onwards, the version has been removed from the name, and the package repository also includes major version changes.
- `extra_requirements.txt`: Include the name of the PyPI packages, one per line, that are needed for some reason.

The addons have a file called "`__manifest__.py`" [18] in which the information about them is included, including the Odoo modules and Python modules that they require to work. In the case of addons installed through "pip", these two types of modules are not installed at the same time, but in the case of addons that are in the "addons" folder and are installed in the classic way, the second type of modules, the Python ones, must be previously installed or it will give an error.

This script, written in the Bash scripting language, is designed to automate several tasks for managing an Odoo instance within a Docker environment. It can be run as shown in Fig. 3.16.

A terminal window with a dark background and three colored window control buttons (red, yellow, green) at the top left. The terminal text shows a comment and a command being executed.

```
# Run the Bash script to install and update both addons and modules.  
vboxuser@ubuntu:~/odoo$ scripts/update_addons_and_modules.sh
```

**Fig. 3.16** Execute the script from CLI

See below for a breakdown of its functionality:

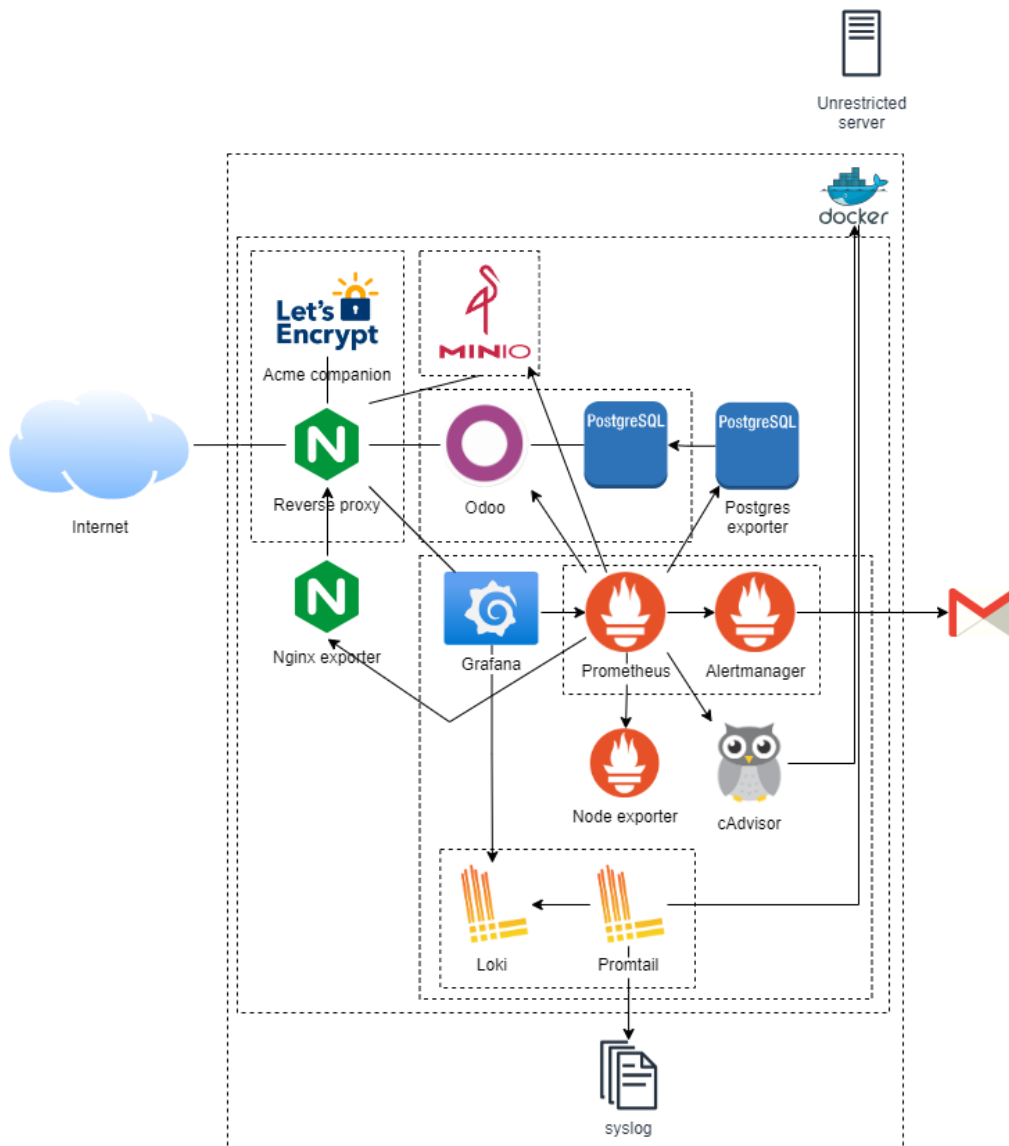
1. Through the second script, the Python one, the content of the different "manifest" is reviewed, the requirements in terms of Python modules are collected, and the file "addons\_requirements.txt" is created in the "addons" folder.
2. The script then proceeds to install requirements for both the addons and the extra ones, using pip.
3. Afterwards, the script installs the extra modules, via pip. It then goes on to use Odoo command-line interface [19] to install and update both the addons in the folder and the base modules.
4. After successfully installing and updating addons/modules, the script restarts the Odoo service to apply the changes.

## CHAPTER 4. ODOO DEPLOYMENT

Once installed, Odoo needs to be properly deployed. Here, the deployment of Odoo is explored, including setting up a reverse proxy and enabling HTTPS to improve security. Deployment automation is considered essential to maintain a stable environment.

### 4.1. Deployment

Throughout this document, starting with the basics, the different points will be studied in order to finally obtain and understand the complete system presented in Fig. 4.1.



**Fig. 4.1** Overall picture of the system

So far, it has simply been shown how to install Odoo in a local environment. Moving on to production environments, it is advisable to avoid exposing ports directly on a Docker Compose for security and scalability reasons. Instead, it is recommended to use internal networks and load balancers to control access and traffic distribution to containers. This protects against threats and simplifies management in large-scale, changing environments.

#### 4.1.1. Reverse proxy

A reverse proxy is a server that acts as a middleman, receiving requests from clients and forwarding them to a single or more backend servers, managing routing, security, and traffic optimisation. This allows hiding the infrastructure behind the proxy, improving performance by distributing the load and adding layers of security, such as SSL encryption and authentication, before requests reach the end servers.

Different software solutions are available on the market for this purpose. Nginx is known for its stability and performance, Traefik is ideal for container environments and Caddy stands out for its ease of configuration and security. The choice depends on the needs of the project. Although the Traefik option is very tempting and some Odoo-based projects use it, the tried and trusted Nginx will be used.

For this implementation, "jwilder/nginx-proxy" [20], a Docker container that provides an automated reverse proxy solution, will be used. This container automatically detects other containers that expose ports and configures an Nginx proxy server to route traffic to those containers based on hostnames or Docker tags. It simplifies the configuration of multiple web services by providing automatic routing based on hostnames and facilitates the implementation of SSL/TLS automatically through solutions such as Let's Encrypt.

#### 4.1.2. HTTPS

The "nginxproxy/acme-companion" [21] container will be added, which is used as part of a reverse proxy solution with automatic TLS certificate support. To obtain and renew TLS (Transport Layer Security) certificates securely and automatically, it works with a web server such as Nginx and the ACME (Automated Certificate Management Environment) client. This container acts as an intermediary, taking care of the acquisition and renewal of SSL/TLS certificates via the ACME protocol (e.g. Let's Encrypt [22]), and redirecting secure requests to the backend web server, such as Nginx. Online discussions can now be automatically configured for end-to-end encryption without the need for lengthy or costly configuration.

### 4.1.3. Docker networking

When utilising Docker Compose, the services defined in the same `docker-compose.yml` file are automatically connected through a bridge network. In this way, the "odoo" and "postgres" services are automatically linked to the same bridge network, allowing them to talk to each other.

Based on its container name, Docker generates a distinct DNS name for each service. In this case, the "postgres" service will be reachable from the "odoo" service using the hostname "postgres". This DNS resolution is done automatically within the Docker network [23].

Building a good foundation will involve segmenting the communication between the different containers, using Docker networks.

Docker internal networks are isolated environments where containers can communicate with each other without direct exposure to the external network. These networks allow for the segmentation and protection of communications between containers, providing greater security. Containers on the same internal network can communicate via host names and ports, as if they were on a private local network, facilitating the development and deployment of complex applications.


It should be noted that this is a separate functionality from the above-mentioned with respect to nginx, not to be confused.

## 4.2. Odoo deployment

The containers are added and the relevant configurations are made. The complete configurations can be found in the GitHub repository, because once the base has been set up, it is only interesting to comment on a few points.

Basically, two new services have been added. It is important to note that it is necessary to name, not just declare (anonymous) a new volume for the certificates, otherwise new certificates will be generated again if the containers are deleted or recreated, reaching the free quota of Let's Encrypt.

It will be necessary to make some modifications to the nginx-proxy. This can be done in several ways, such as mounting configuration files on the service, which has already been done with Odoo, or creating a custom image. The latter requires a "Dockerfile" [24], which is a text document containing instructions for building a Docker container image, describing how to set up a container environment, which applications or services to include and how to configure them. This file has been saved as "nginx-proxy.Dockerfile" (see Fig. 4.2), in the new "dockerfiles" folder.



```
nginx-proxy.Dockerfile

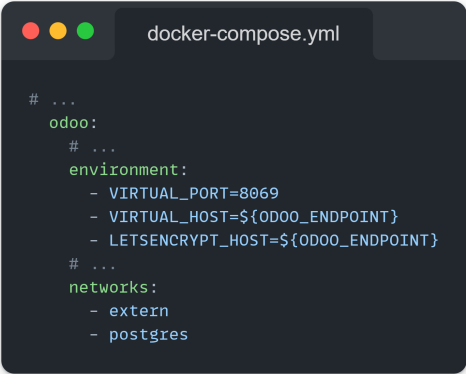
FROM jwilder/nginx-proxy

# Create a custom configuration file
RUN { \
    echo 'client_max_body_size 100m;'; \
} > /etc/nginx/conf.d/my_proxy.conf
```

**Fig. 4.2** Dockerfile "nginx-proxy.Dockerfile"

There are four environment variables for these two new services. The port to which incoming HTTP traffic should be directed, the domain to which the traffic should be directed and the SSL certificate generated, and an email address required to be able to use Let's Encrypt. The environment file ".env" shall be used, so that it can have different domains depending on the environment.

If the networks required by the Odoo service, "extern" (proxy) and "postgres", are added, the updated configuration is shown in Fig. 4.3.

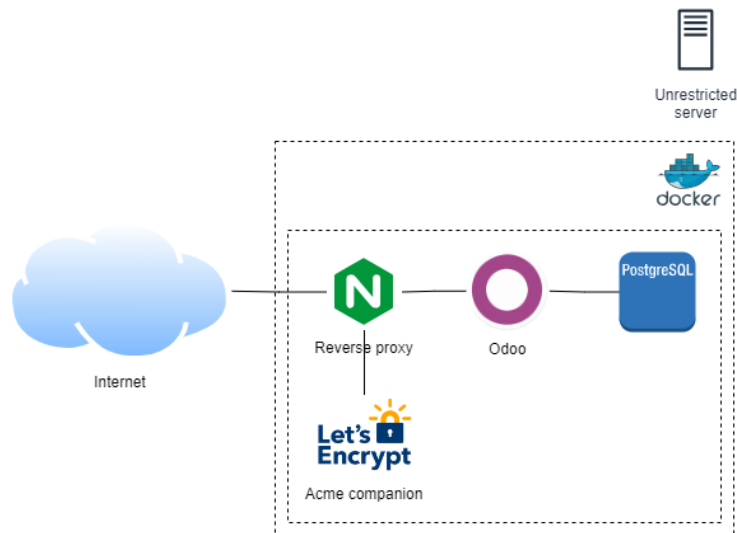


```
docker-compose.yml

# ...
odoo:
# ...
  environment:
    - VIRTUAL_PORT=8069
    - VIRTUAL_HOST=${ODOO_ENDPOINT}
    - LETSENCRYPT_HOST=${ODOO_ENDPOINT}
# ...
  networks:
    - extern
    - postgres
```

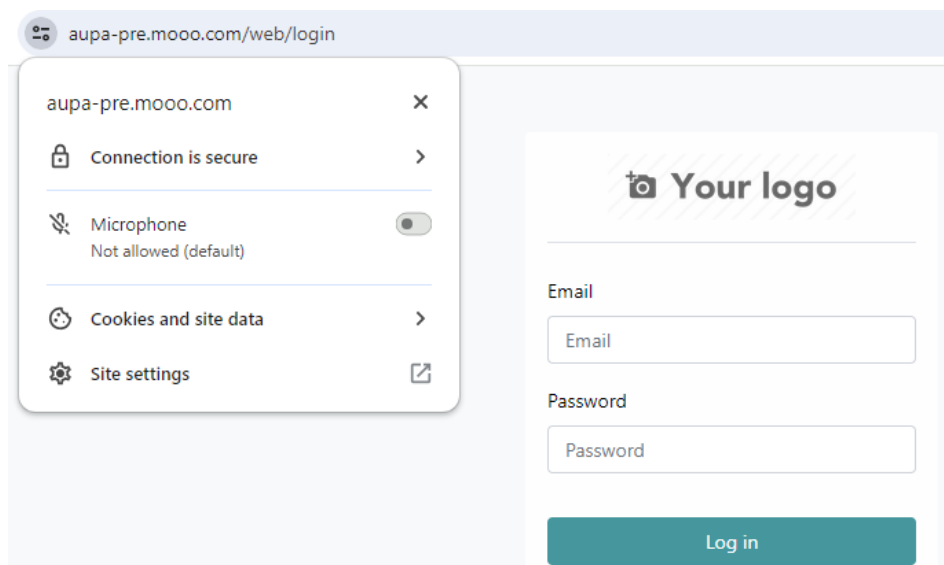
**Fig. 4.3** Updated Docker compose file "docker-compose.yml"

Assuming a server without technical limitations, although this is not the case, the scenario would be as follows (see Fig. 4.4):



**Fig. 4.4** Current scenario on an unrestricted server

Now, the page can be accessed from the browser and the connection can be checked for security (see Fig. 4.5).



**Fig. 4.5** Secure connection check screen on the server

It is possible that, in the event of changes to the domains covered by acme-companion, no new certificates are generated. This can be solved with a new container, for example with "docker-gen", which is a tool that automatically generates configuration files for reverse proxy servers, such as Nginx, based on Docker container events, thus facilitating the administration and routing of traffic in Docker container environments. For this project, it has not been implemented, as the recreation of the acme-companion container has the same effect (see Fig. 4.6):

```
# Recreate acme-companion container
vboxuser@ubuntu:~/odoo$ docker-compose up -d --force-recreate acme-companion
```

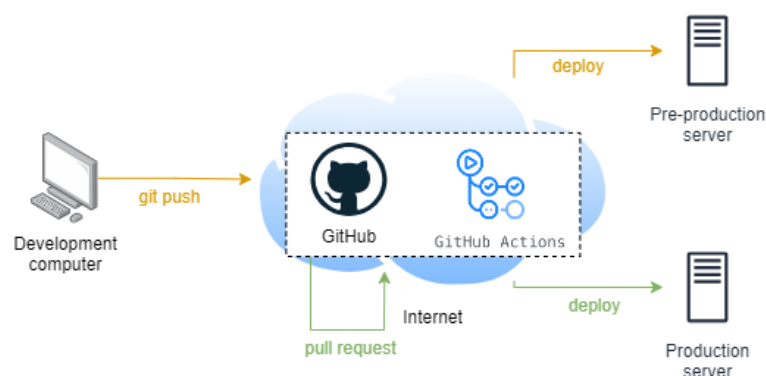
**Fig. 4.6** Command to recreate the acme container

### 4.2.1. Automation

It is assumed, at this point, that the project is already being managed with Git and uploaded to GitHub.

GitHub Actions [25] is a tool that allows automating workflows and actions in projects maintained on GitHub. Certain events, such as code changes, build pull requests or releases, can be configured to trigger certain actions. These actions simplify development automation and CI/CD (continuous integration/continuous delivery) operations from a GitHub repository. These actions can do things like build, test, deploy and more. "Actions" are defined through "Workflows", which are automation scripts that define specific tasks and workflows in a GitHub repository.

In the following example (see Fig. 4.7), there is a typical use case. Upon a push to the repository, an action is initiated to deploy the new changes to PRE, and upon a successful pull request from PRE to PRO, the changes are deployed to PRO.



**Fig. 4.7** GitHub Actions usage example

In addition, "secrets" can be set up, which are confidential values, such as passwords or API keys, that are stored securely and can be used in GitHub Actions workflows without exposing sensitive information. They help maintain security and privacy in CI/CD development and automation on GitHub.

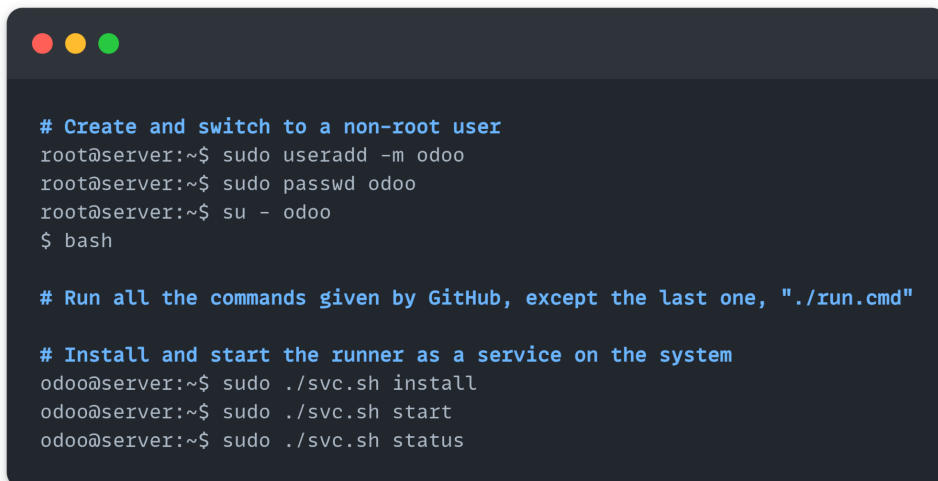


A runner is an instance or agent that runs automated tasks in GitHub Actions. It can be provided by GitHub (self-hosted runner) or configured and hosted by the user (self-hosted runner). Its function is to execute specific workflows and tasks in response to events in a GitHub repository.

GitHub Actions' own runner is provided by GitHub and runs on GitHub's infrastructure, while a self-hosted runner is a runner that is configured and hosted by the user on its infrastructure (such as a server or local machine). The main difference is control and customisation: GitHub's own runner is easier to use, but the self-hosted runner gives more control and can execute actions on an own infrastructure.

For specific cases, such as performing testbeds on the code or repetitive creation of custom Docker images, it is advisable to use a GitHub runner, since the resources of the production machines will not be used.

A connection is established via SSH to the root user of one's own server. Docker must be installed with the aforementioned commands. To create a self-hosted runner, from the GitHub repository, "Configuration" - "Actions" - "Runners" - "New self-hosted runner". Then (see Fig. 4.8) the base commands are modified to configure the runner as a service.



```
# Create and switch to a non-root user
root@server:~$ sudo useradd -m odoo
root@server:~$ sudo passwd odoo
root@server:~$ su - odoo
$ bash

# Run all the commands given by GitHub, except the last one, "./run.cmd"

# Install and start the runner as a service on the system
odoo@server:~$ sudo ./svc.sh install
odoo@server:~$ sudo ./svc.sh start
odoo@server:~$ sudo ./svc.sh status
```

**Fig. 4.8** Commands to configure a runner

The options offered by GitHub actions are endless, but this time, the focus is on the automatic deployment of a Docker-based project on one's own server. It is necessary to create the following file and folders in the root of the project: ".github/workflows/deploy.yml" (see Fig. 4.9).

```

name: Deploy Docker Compose

on:
  push:
    branches:
      - master # Triggered when changes are made to the "master" branch.

jobs:
  deploy:
    runs-on: self-hosted # Run on a "self-hosted" runner, configured by the user.

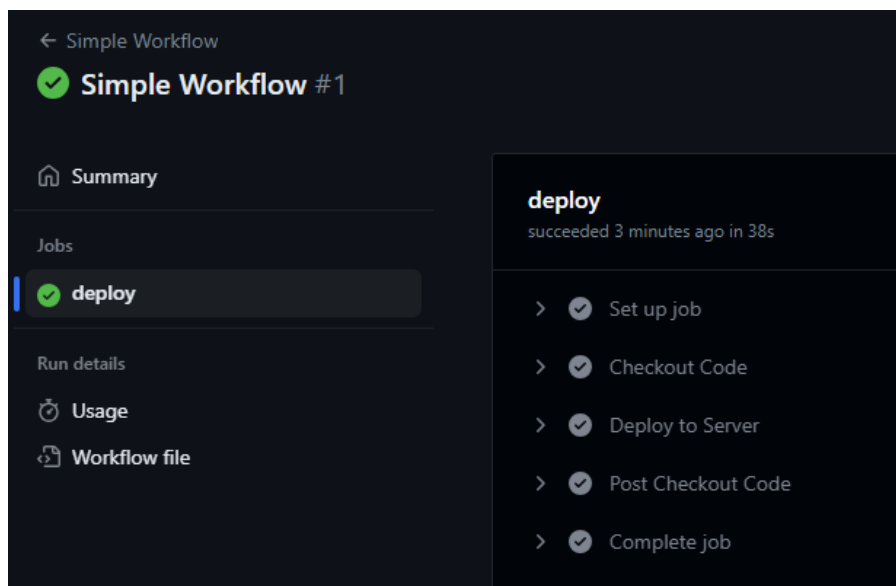
    steps:
      - name: Checkout Code
        uses: actions/checkout@v3 # Get the code from the repository.

      - name: Run commands on the server
        run: |
          docker-compose up -d
    
```

**Fig. 4.9** Workflow configuration file "deploy.yml"

With the second action, a console is available on the server, in order to be able to execute the scripts that will be mentioned later, in an automated way.

In case of a push, the workflow will be executed, and the status can be consulted in the "Actions" section of the GitHub repository (see Fig. 4.10) or in the e-mail that will be sent to an specified mailbox.



**Fig. 4.10** GitHub Action workflow status screen

Upon updates to the project, services will be updated and added. Updates to addons have already been discussed above, so the script in question could be added.

The question arises whether it could be cost-effective to create a customised Odoo image that already incorporates the addons, via a Dockerfile. This could also be applied to the rest of the services. It would consist of an action that generates the images in a GitHub server and uploads them to a private repository in Docker Hub, and another action that takes them out of that repository.

Since the updated Docker Compose would still need to be copied to the production server, it is complicating things more than they already are. Also, it would require restarting containers that often don't need to be restarted to apply the changes, as mentioned above, so although uploads to production should take place during peak service usage hours, this would add more downtime.

#### 4.2.2. Addressing infrastructure problems

Although the machines are exposed on ports 80 and 443 to serve the content, in this case, to access the servers, a VPN connection must be established.

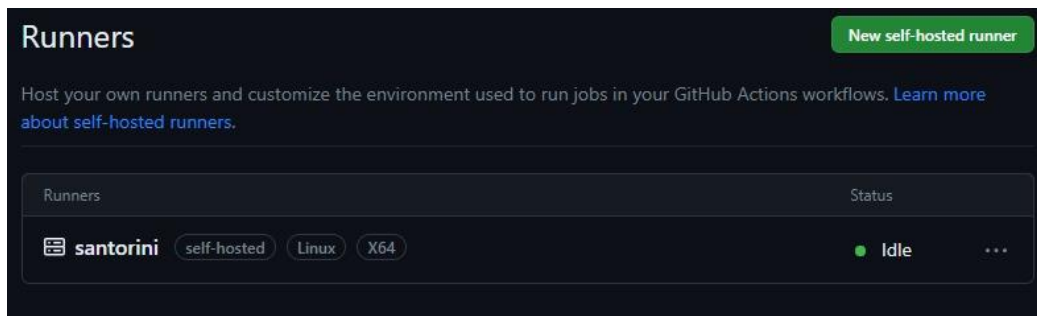
The developers use the “FortiClient VPN” client [26] for this purpose. The system is very similar to the one recently implemented at the university itself, with credentials, to which a 2FA is followed, to add an extra layer of security.

Self-hosted runners use HTTPS long poll with a 50 second connection timeout that when it times out, a new connection is opened. This means that there is no need to allow GitHub to establish incoming connections to the runner. Runners connect back to GitHub, no need for holes in the inbound firewall, outbound HTTPS 443 is all that is needed.

In the following example of the pre-production machine, the connection has been established by bypassing the VPN and without the need to open ports (see Fig. 4.11 and Fig. 4.12).

```
odoo@santorini:~/actions-runner$ ./run.sh
✓ Connected to GitHub
Current runner version: '2.308.0'
2023-09-06 11:37:36Z: Listening for Jobs
```

**Fig. 4.11** Pre-production server console screen activating runner daemon



**Fig. 4.12** GitHub repository runner status screen

The given environment is limited, as is to be expected for security reasons. The reverse proxy is provided by the provider and routes the traffic to the virtual machine through port 8000, the internal port of the virtual machine, which is the one they have opened and consume in the virtualisation management.

Actually, ports 80 and 443 leave the physical machine, through its reverse proxy, which may or may not be virtualised. In this way, they are also managing the ACME through Let's Encrypt.

In the case of needing to open more ports, internal to the virtualised machine, it is necessary to contact the provider, but even if it has not been considered so far, there is an easy solution.

Based on the Docker compose generated so far, the one currently used in the ERP environment is simpler, but fully compatible with the one seen.

Without going into details, one might tend more towards one configuration or the other, the most important thing is to keep the name of the volumes already in use. All references to ACME or certificates are removed, and all exposed ports are also removed, except for nginx. Port 443 will not be used, and the container's internal port 80 should be exposed to the virtualised machine's internal 8000, which is the one that is enabled. The environment variables of ports and virtual hosts must also be configured for the containers that must go outside. This way it is possible to host several services accessible from the internet in a machine with limitations.

To access the Postgres, although it has an internally exposed port, which in itself is not a good practice, the VPN limits access from the outside, fortunately. It would not be advisable to make it publicly accessible, although it is always password protected. It can be understood that in pre-production environments, having a SQL client such as DBeaver can make development tasks easier.

## CHAPTER 5. ODOO BACKUP

Data loss can be catastrophic, so this chapter focuses on the importance of Odoo backups. Automated backup methods and the configuration of Minio, a cloud storage system for backups, are covered.

### 5.1. Backup

One of a project's most crucial components is the security copies. The ability to restore data and systems in the event of loss, damage, or errors ensures service continuity, protects sensitive data, and shortens downtime, all of which are reasons why security backups are essential.

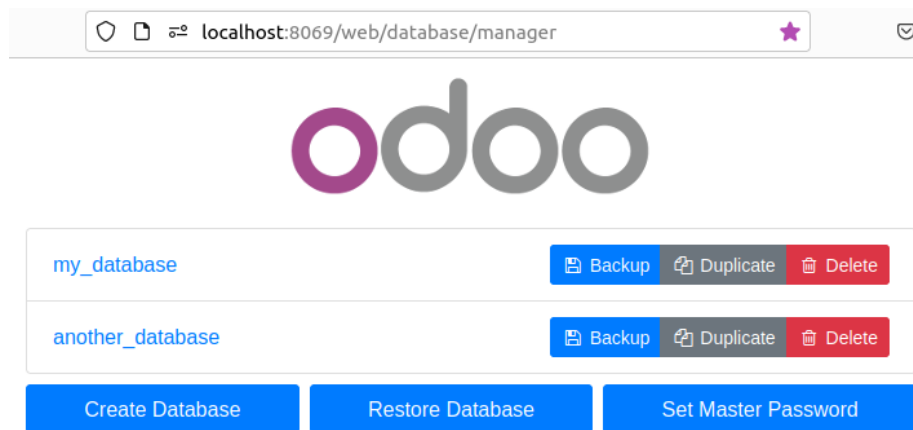
The "3-2-1" rule, which should be kept in mind, refers to maintaining three copies of the data, at two different locations, with at least one of them outside the primary site, to ensure the availability and recovery of the data in a variety of circumstances.

Data online archiving may become reliable and scalable with the use of cloud storage services like Amazon S3, Google Cloud Storage, Microsoft Azure Blob Storage, and IBM Cloud Object Storage. Each of them offers unique benefits including competitive prices, options for long-term data storage, connectivity with certain platforms, and services. The choice between these services depends on the specific requirements of an organisation, such as the desired cloud platform, the amount of storage required, and the demand for a particular function.

Using the "Simple Storage Service" (S3) [27] as an example, it is a cloud storage service from Amazon Web Services (AWS) that offers a location to store and retrieve data safely and flexibly online. It provides an affordable solution to store and manage a range of data, from basic files to backups and media assets, and is incredibly reliable. S3 is organised into "buckets" (containers) and "objects" (files), and is widely used in web applications, data backup and static content storage.

### 5.2. Odoo backup

Odoo has a backup management tool (see Fig. 5.1), which can be accessed by clicking on the "Manage databases" button on the login page.



**Fig. 5.1** Odoo database management screen

Current databases can be managed (backup, duplicate or delete), new databases can be created or a database can be restored from a backup.

The backups are downloaded as a .zip file, which includes:

- **filestore:** This folder contains the attachments and documents stored in an Odoo instance.
- **dump.sql:** This file contains a copy of the Odoo database in SQL format, which is essential to restore data in case of need.
- **manifest.json:** This file contains information about the configuration and modules installed on the Odoo instance.

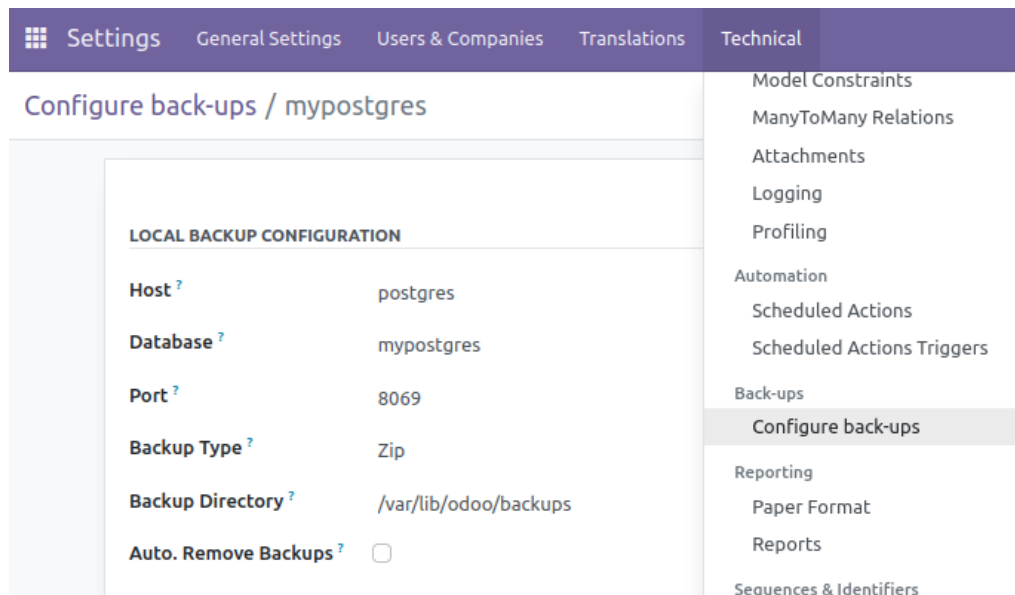
These files are essential for full backups and allow to restore an Odoo instance in case of data loss or technical problems.

Once a backup is made, it should be stored in a safe place.

### 5.2.1. Automation

The Odoo App Store has a lot of add-ons that cover backups. Possibly the most prominent is "Database auto-backup" [28].

Once installed, "Configuration" - "Technical" - "Backup" - "Configure backups". The previously configured parameters must be filled in. To be able to easily access the backups from the host machine, the "backups" folder must be created in the root of the project. Then, Docker compose must be modified to mount that folder in "/var/lib/odoo/odoo/backups" of the container, which is a very recognisable directory.



**Fig. 5.2** Parameter configuration of "Database auto-backup"

Then, from "Configuration" - "Technique" - "Automation" - "Scheduled actions" - "Backup scheduler", it is possible to activate and define how often this task will be performed.

In addition, a small script, "backup.sh", has been created, which asks Odoo for a backup through a POST, and saves it in the backups folder. In case the parameters it needs are not available, it asks for them, but ideally they should be added, if they are not already, in the environment file, so that by calling the script through a cronjob or the same workflow seen previously, it could be executed periodically. Unlike the addon, the script is run from the client machine, so it opens up a lot of options.

### 5.3. MinIO

Since there is currently no S3 available to store backups also in the cloud, one option is to set up one.

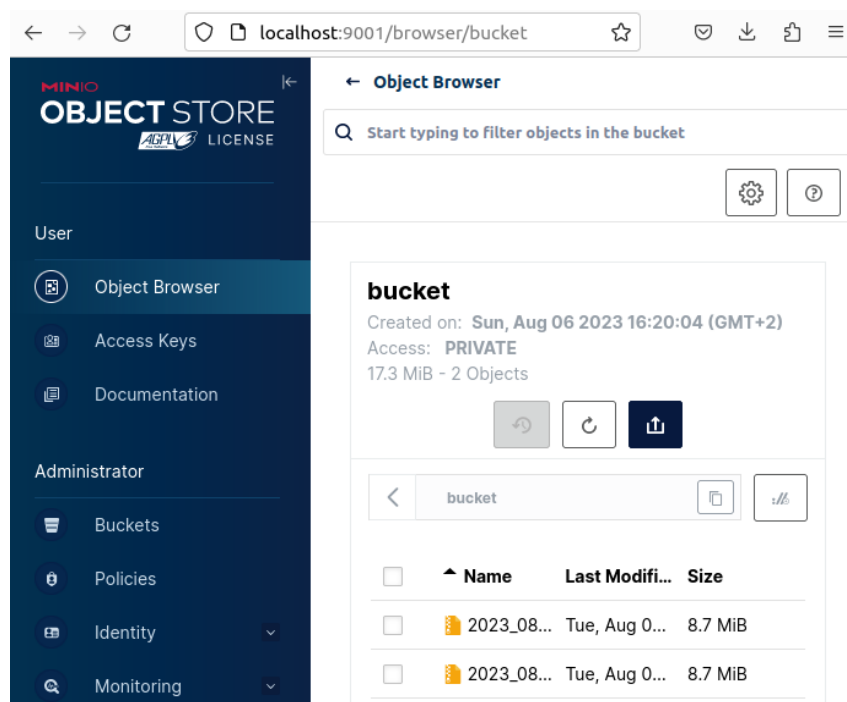
Similar to Amazon S3, Minio [29] is an open source object storage server that enables the creation of a secure cloud storage system for unstructured data and files.

Although it doesn't make much sense to have both the Odoo instance and Minio on the same server, for the purpose of this project it has been done that way. A Minio could be deployed on another server, or a real S3 could be used, the only thing that will change is the URL to attack.

For this new part, two new services have been added to the Docker compose.

The first service refers to Minio itself, where it should be noted that it runs on two ports. On 9000, it gives access to an API that simulates an S3, to the extent that it can be used from S3 clients. On 9001, it gives access to the "console" or web interface. Here it should be noted that, if accessed from the browser, on port 9000 if it is directly exposed, or with the reverse proxy, but also pointing to 9000, it will redirect to 9001, provided it is open, to provide the web interface.

To illustrate the "console", 9001 can be temporarily exposed, but in the production version, neither port will be exposed, the reverse proxy will only point to 9000, which is the important thing, as using two virtual ports for the same container would mean modifying the reverse proxy.



**Fig. 5.3** MinIO main screen

The second service is used to, through a temporary Minio client called "mc" [30], although it could be any S3 client, create a bucket called "bucket", in order to start using it.

Now that the bucket is in place, an S3 client can be installed on the host machine, and any file can be copied to the bucket. That said, any S3 client will do, as the endpoint that Minio generates simulates an S3. From the official Minio client, "mc", to an alternative such as "S3cmd", or the official Amazon client, "AWS CLI" [31]. Each of these has been tested, but the most relevant thing is to see it working with the Amazon client. A small drawback is that the endpoint has to be added to the command, as it is non-standard (see Fig. 5.4). For this, root is used, although it can also be installed by pip, but "mc" without root has also been contemplated, more on that later.



```

# Install AWS CLI
vboxuser@ubuntu:~/odoo$ sudo apt install awscli

# Run AWS CLI setup
vboxuser@ubuntu:~/odoo$ aws configure

# List buckets
vboxuser@ubuntu:~/odoo$ aws s3 ls --endpoint-url <minio_url>

# Copy a file to a bucket
vboxuser@ubuntu:~/odoo$ aws s3 cp <local_file> s3://<bucket_name>/<object_key> --endpoint-url <minio_url>

# Remove a file from a bucket
vboxuser@ubuntu:~/odoo$ aws s3 rm s3://<bucket_name>/<object_key> --endpoint-url <minio_url>

```

**Fig. 5.4** Basic AWS CLI commands

### 5.3.1. Automation

Following the thread of using an addon to make recurring backups, it seemed that for the latest version of Odoo, version 16, there was no free module, and this is important, available. Migrating addons to newer versions of Odoo will be discussed later, as "Database Auto-Backup to S3" [32] is available for Odoo version 12.

In this situation, using "Database auto-backup" as a base, which in addition to making local copies, allows the use of an FTP server, the code has been modified to attack an S3. Here it is necessary to modify, on the one hand, the model (file.py), in which the "Boto3" package [33], which is official from AWS, has been used, and on the other hand, the view (file.xml), which here is to change the name of the variables.

In the same menu above, the S3 references can now be seen. In this case, the endpoint is using Docker's internal DNS, but it could point to another server.

#### Configure back-ups / mypostgres

**S3**

**Warning:** Use S3 with caution! This writes files to external servers under the path you specify.

Write to external server  (S3) <sup>?</sup>

S3 endpoint URL <sup>?</sup> http://minio:9000

S3 bucket name <sup>?</sup> bucket

S3 access key <sup>?</sup> odoo

S3 secret key <sup>?</sup> .....

Remove S3 after x days <sup>?</sup> 30

**Fig. 5.5** Parameter configuration of the modified "Database auto-backup"

In addition, another small script has been created, "s3\_backup.sh", which is based on the previous one to upload backups to an S3, in this case, using "mc", installing it if it is not already available. Like the previous one, it can be run manually, or by adding it, for example, to a GitHub Actions workflow. As a note, the "mc" package in the Ubuntu repositories does not refer to the package in question.

Add that, although it is not directly related, there are also several addons, such as "Storage Backend S3" [34] from the OCA, which allows the use, as its name suggests, an S3 as storage backend, so the storage of the Odoo volume would be pleasantly reduced. This is a good practice if users are used to attach very large files, instead of saving them in the Odoo volume, they will be linked from S3.

## CHAPTER 6. ODOO UPGRADE

Keeping Odoo up to date is crucial to benefit from the latest features and fixes. This chapter introduces OpenUpgrade as a tool for smooth upgrades. Automation of this process is also included to simplify upgrades.

### 6.1. Upgrade

Keeping Odoo up to date it is important due to several key factors. Firstly, updates often include critical security fixes that protect data and systems from potential threats. In addition, new versions of Odoo often offer performance improvements and additional features that can increase the efficiency and functionality of the business system. Keeping up to date also facilitates integration with other applications and services, which is essential for a constantly evolving business environment.

To ensure that Odoo images are using the most recent releases, updates are made often, a new release of each version of Odoo is created every night.

Upgrading from an old release to the latest one provided of the same major version is a straightforward process. In contrast, upgrading from a major version to another is a much more complex process, requiring elaborated migration scripts.

At the time, Odoo Community Edition (CE) included a series of scripts to migrate an instance to the new version. In reality, these scripts have not disappeared, they have simply been moved to the Enterprise version, and the migration is being managed by Odoo's own technical team.

An alternative is to export the data from the old instance, install the new version of Odoo, and import the data into the new version. Please note that in this case not all data will be recovered, e.g. history and links between data will be lost.

#### 6.1.1. OpenUpgrade

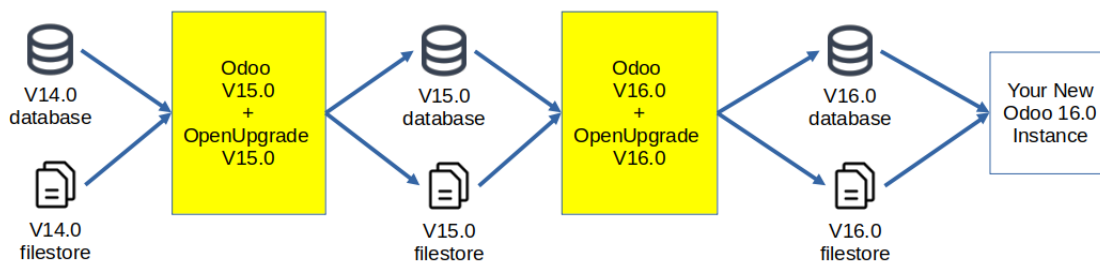
For this case, appeal is made to the Open Source community, and when it comes to migrating Odoo, it is OpenUpgrade [35], which is also under the umbrella of the OCA. As mentioned, Odoo CE has the option to upgrade within the same major version, what it cannot do is make the jump. OpenUpgrade uses this feature of Odoo as a base.

The OpenUpgrade project is hosted as two separate GitHub projects:

- OpenUpgrade contains:
  - the framework which is a set of a few Odoo patches to ensure that upgrading to a major version can be performed

- The database analysis which lists the differences between one version of Odoo and the next
- The migration scripts which contains database transformation scripts for each installed Odoo module.
- openupgradelib contains a library with helper functions. It can be used in the migration of any Odoo module

In Fig. 6.1 [35], the flow according to the official documentation can be seen:



**Fig. 6.1** OpenUpgrade Flow

## 6.2. Odoo upgrade

When migrating an instance with an active mail server, it is advisable to deactivate it during the process. Remember, this can be done in "Configuration" - "Technical" - "Incoming/outgoing mail servers".

Before touching anything, it is important to make sure that the modules in use have migration scripts for the version to which the migration is to be done.

The list of installed modules in the "Apps" section. In the OpenUpgrade github.io it must be checked whether the migration is possible or not. This has to be checked for each version upgrade desired, as it is done one at a time.



In the event that a module is not supported, one can choose to do without it by uninstalling it, or one can try to correct the errors that may arise, which is not a very recommendable option. The OCA encourages users to contribute the missing migration scripts, a process that is very well documented.



## Module coverage 14.0 -> 15.0

A module can have one of the following statuses:

- **Nothing to do:** Analyse has been done, and there is nothing to do or there's no change between the old edition and the new one;
- **Done:** Scripts are ready to run and can be found in this distribution;
- **<Empty>:** Not covered yet. Please see the 'openupgrade\_analysis.txt' file of the module and start hacking the migration scripts!

You will always find migration scripts that are ready for review or work-in-progress at <https://github.com/OCA/OpenUpgrade/pulls>. Please have a look to see if you can help out.

In the table below, modules that are new in this release are marked with . Modules that are missing in the new release are marked with .

Module	Status	Extra Information
account	Done	
account_check_printing	Done	No DB layout changes.
account_debit_note		No DB layout changes.
account_edi	Done	
 account_edi_extended		Merged into account_edi.
account_edi_facturx	Nothing to do	
account_edi_proxy_client		No DB layout changes.
account_edi_ubl	Nothing to do	
 account_edi_ubl_bis3		

**Fig. 6.2** Screen of the module coverage on the OpenUpgrade page

Now addons (custom modules) come into play, but in this case there is no list. In the same way, the user can either do without them, or investigate how to migrate them, which, except for a specific version jump, sometimes it is not necessary to do anything at all.

Once OpenUpgrade is started, if something goes wrong, the error must be corrected, otherwise the database becomes unusable, so be sure of what is being done.

If this is the way to go, the first thing to do is to make a backup. Next, one should make a copy of the root folder of the project, where the Odoo version will be modified in the Docker compose, the project needs to be started, so the backup can be imported, although Odoo will show a warning, if something is not right. If there are several version jumps, the process will have to be repeated several times.



**Fig. 6.3** Odoo database management screen with an error in one database

Installation of `openupgradelib` is done directly from PyPI, since the package exists (see Fig. 6.4).

```
# Install openupgradelib inside the odoo container
vboxuser@ubuntu:~/odoo$ docker-compose exec odoo pip3 install openupgradelib
```

**Fig. 6.4** Command to install `openupgradelib`

Now, depending on the version to migrate to, the files needed are different:

- From Openupgrade 5.0 to Openupgrade 13.0, the OpenUpgrade repository branches contain copies of the main Odoo project, but with extra commits that include the framework, and the analysis and migration scripts for each module. In fact, the migration will be executed using a modified version of Odoo (see Fig. 6.5).

```
# Clone the branch, without the GitHub history
vboxuser@ubuntu:~/migrate$ git clone -b <odoo_version>
--depth 1 https://github.com/OCA/OpenUpgrade.git
addons/OpenUpgrade

# Run OpenUpgrade
vboxuser@ubuntu:~/migrate$ docker-compose -f exec odoo
/mnt/extra-addons/OpenUpgrade/odoo-bin
--database=<your_db>--update all --stop-after-init
```

**Fig. 6.5** Commands to run OpenUpgrade

- Since Openupgrade 14.0, the branches contain an `openupgrade_framework` module that gathers all the odoo patches, and a module called `openupgrade_scripts` that contains the analysis and migration scripts.

As they are addons, one could download the two folders in the addons folder, and it is not necessary to install them, they just have to be available through the addons-path. There is also the option of the packages in PyPI, but they are intended for testing, and don't work as they should. The quickest option is to clone the branch from the repository (see Fig. 6.6).

```

# Clone the branch, without the GitHub history
vboxuser@ubuntu:~/migrate$ git clone -b <odoo_version>
--depth 1 https://github.com/OCA/OpenUpgrade.git
addons/OpenUpgrade

# Run OpenUpgrade
vboxuser@ubuntu:~/migrate$ docker-compose -f exec odoo odoo --database=<your_db>
--addons-path=/mnt/extra-addons/OpenUpgrade/
--upgrade-path=/mnt/extra-addons/OpenUpgrade/openupgrade_scripts/scripts
--load=base,web,openupgrade_framework
--update all --stop-after-init

```

**Fig. 6.6** Commands to run OpenUpgrade

After countless operations, if all has gone well, the danger symbol will have disappeared, and the migrated database will be accessible.

It is advisable to restart the Odoo container once the migration has been successfully completed, as sometimes, the danger symbol disappears, but when entering the database, errors appear that may suggest otherwise.

If a critical error has occurred (see Fig. 6.7), action must be taken.

```

2023-08-22 16:36:34,426 27 INFO o12-p11-old odoo.modules.loading: loading 21 modules...
2023-08-22 16:36:34,469 27 WARNING o12-p11-old odoo.addons.auto_backup_aws_s3.models.s3_backup: pip3 install --upgrade boto3
2023-08-22 16:36:34,469 27 CRITICAL o12-p11-old odoo.modules.module: Couldn't load module auto_backup_aws_s3
2023-08-22 16:36:34,469 27 CRITICAL o12-p11-old odoo.modules.module: module 'odoo.api' has no attribute 'multi'
2023-08-22 16:36:34,470 27 ERROR o12-p11-old odoo.modules.registry: Failed to load registry
Traceback (most recent call last):
  File "/mnt/extra-addons/OpenUpgrade/odoo/modules/registry.py", line 87, in new
    odoo.modules.load_modules(registry_db, force_demo, status, update_module)
  File "/mnt/extra-addons/OpenUpgrade/odoo/modules/loading.py", line 489, in load_modules
    force, status, report, loaded_modules, update_module, models_to_check, upg_registry)
  File "/mnt/extra-addons/OpenUpgrade/odoo/modules/loading.py", line 368, in load_marked_modules
    upg_registry=upg_registry,
  File "/mnt/extra-addons/OpenUpgrade/odoo/modules/loading.py", line 195, in load_module_graph
    load_openerp_module(package.name)
  File "/mnt/extra-addons/OpenUpgrade/odoo/modules/module.py", line 405, in load_openerp_module
    __import__('odoo.addons.' + module_name)
  File "/mnt/extra-addons/auto_backup_aws_s3/__init__.py", line 3, in <module>
    from . import models
  File "/mnt/extra-addons/auto_backup_aws_s3/models/__init__.py", line 2, in <module>
    from . import s3_backup
  File "/mnt/extra-addons/auto_backup_aws_s3/models/s3_backup.py", line 29, in <module>
    class S3Backup(models.Model):
  File "/mnt/extra-addons/auto_backup_aws_s3/models/s3_backup.py", line 65, in S3Backup
    @api.multi
AttributeError: module 'odoo.api' has no attribute 'multi'

```

**Fig. 6.7** Console error screen after running OpenUpgrade

To put it in context, this error has occurred in the migration attempt from Odoo 12 to 13, an instance that included the "Database Auto-Backup to S3" addon that was produced previously. A Google search for the error "AttributeError: module 'odoo.api' has no attribute 'multi'", shows that in the Odoo 13 version the

attribute "@api.multi" has been removed, so removing the references in all the files where it is, paths indicated above the error, and re-running OpenUpgrade, the result should be satisfactory. Subsequently, the migration has been continued up to version 16 without any problems, so the addons are really easy to migrate.

There is a module called "database\_cleanup", hosted in the OCA server-tools project, which makes it easy to remove unnecessary traces after migration.

### 6.2.1. Automation

Automating this process is a very helpful addition that makes the job much easier. A script, "upgrade.sh", has been created that follows the following flow:

1. A function is defined, so that in case of an error the original state of the project is restored.
2. The directory to work on is defined, independently of where the script is called from.
3. The environment variables are defined and queried.
4. Ask for the availability of an S3 bucket and runs "s3\_backup.sh" or "backup.sh", as appropriate.
5. Two copies of the project folder are made, one as a backup and one to work with. It is necessary to be careful with the volumes, because if the project folder has the same name, even if it is in another directory, the volumes are the same. Also, if the services have the same name as others that are already running, it will not allow the services to be created.
6. "restore\_backup.sh" is executed, which, as its name indicates, imports a database through a POST.
7. It asks if all addons (custom modules) are to be migrated, and if so, it returns the name across folders, or it asks for the addons in question.
8. "check\_modules\_coverge.sh" is executed, which in turn relies on "get\_modules\_coverage.py". The second, import "BeautifulSoup4" [36] to pull the modules with coverage from the table on the OpenUpgrade page. The first one, pulls the list of installed modules directly from the database, and compares it with the coverage.
9. "migrate\_addons.sh" is executed, which uses the PyPI package "odoo-module-migrator" [37] from the OCA, to migrate the addons. This package is not mentioned in the official OpenUpgrade guide.
10. As explained in the manual process, it installs openupgradelib, clones the OpenUpgrade repository and runs OpenUpgrade differently



depending on the version. In case of errors in the process, there is the option to correct them and try again without losing the process.

11. If more than one is needed, the checks are repeated. If the process is successful, the previous step is repeated for the new version. If not, the upgrade will be to the version that has already been successfully migrated in the previous step.
12. The resulting database is backed up locally or in an S3, based on the initial question.
13. The migration folder is cut and pasted over the original folder.
14. It asks if the Postgres version is also to be upgraded, and if so, to the version lower than the Odoo version, as Postgres is one version number behind.
15. Based on the previous answer, either the original database is previously deleted through a POST with "drop\_database.sh", or the new one is restored directly, to avoid error when repeating the name.
16. Finally, the original updated project is available as well as a backup, in case anything happens.

All the scripts it supports can be run independently, and in case they don't have the parameters it needs, they ask for them, but ideally they should be added, if they are not already, in the environment file.

The main script is interactive, since it asks several questions, but it is very easy to adapt it so that no intervention is required. In that case, it is recommended to check the output for possible errors, which has been taken into account, and the outputs that do not really contribute anything have been removed.

With this script, any instance of Odoo version 5.0 can be easily migrated. Actually, there are only a few conflicting versions, which have incorporated more changes. However, the more version jumps, the more chances for errors.

## CHAPTER 7. ODOO MONITORING

Constant monitoring is essential to ensure optimal performance. Metrics, logs, and alerts are addressed here to detect and resolve problems efficiently.

### 7.1. Monitoring

Monitoring Odoo is essential to ensure its smooth operation and optimal performance. Proper monitoring helps identify issues, improve system efficiency, and provide a better user experience.

The classical point in monitoring is metrics, which is critical to maintaining the performance and health of an Odoo system. There are some key areas of monitoring:

- **Server Monitoring:** This aspect of monitoring is concerned with keeping track of server resources including CPU, RAM, disc space, and network activities within reasonable bounds.
- **Uptime Monitoring:** Odoo's uptime is monitored by a system, guaranteeing that the instance is always reachable and functioning correctly.
- **Container Monitoring:** When using Docker containers to deploy Odoo, it is crucial to monitor their performance and resources to ensure efficient operation.
- **Application Monitoring:** This area focuses on monitoring Odoo processes, response times and transaction rates to assess and optimise application performance.
- **Database Monitoring:** The underlying database, typically PostgreSQL, is critical to Odoo. Monitoring its health and performance, including database size, query execution times and connection counts, is essential for optimal performance.

All this must be accompanied by log monitoring, which involves regularly reviewing Odoo's log files for errors, warnings, and other relevant information. This makes it possible to identify and solve potential problems.

All this can be improved through alerts and notifications, which are configured for critical events or abnormal behaviour in the system. These alerts can be sent through a variety of channels, such as email, SMS, or collaboration platforms.

All this can be improved through, it is essential to regularly monitor and verify the success of backup processes to ensure data integrity. Automating backup monitoring and notifications in case of failures are important steps to keep data safe and recoverable in case of loss.

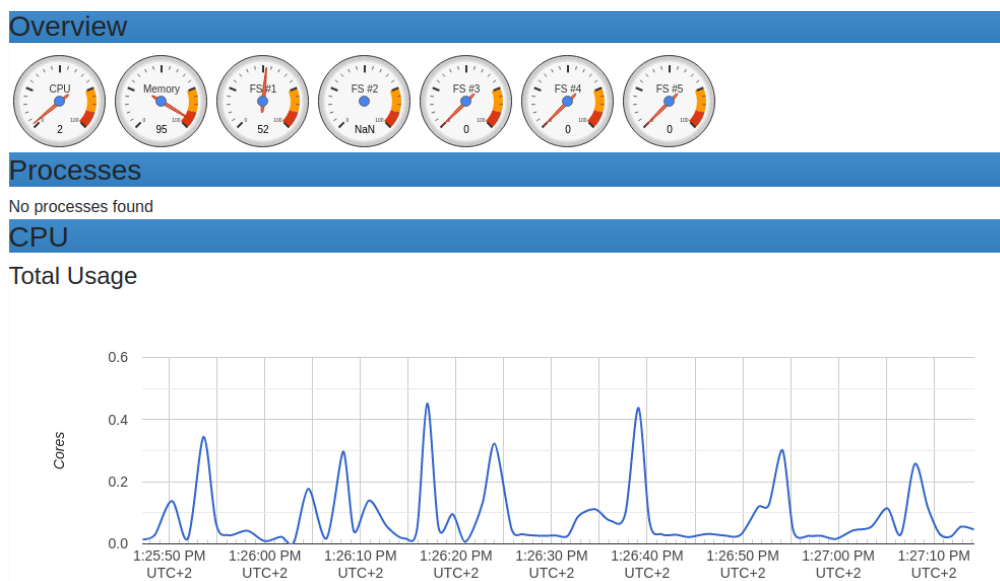
After all this introduction to monitoring, since the backup part is already covered by addons and scripts, three main blocks are defined: metrics, logs and alerts. Together, these three elements provide a complete picture of the performance and health of a system. Priority has been given to the use of open source technologies.

### 7.1.1. Metrics

Metrics are numerical statistics that are used to assess a system's state and performance. This data is gathered periodically and offers details on resource utilisation, application behaviour, and other pertinent system characteristics. CPU use, RAM that is accessible, network traffic, application response times, and other metrics are examples of metrics. Metrics are necessary to comprehend a system's performance and health both in the present and over time.

The easiest method to obtain certain metrics in a Docker-based project is to utilise cAdvisor (Container Advisor) [38]. This Google open source tool is used to track metrics from Docker containers and other container systems. It offers details on the functionality and resource usage of containers, including CPU, memory, storage, and networking. For real-time monitoring and the production of metrics that help with decision-making and the identification of issues with containerized applications and services, cAdvisor is frequently used in container settings.

The configuration of this service is relatively simple, but it should be noted that for maximum compatibility, a community-modified image has been used, and that reading kernel messages will probably not work at first. If port 8080 is exposed temporarily, as there are other ways to display the information, its web interface can be seen (see Fig. 7.1).

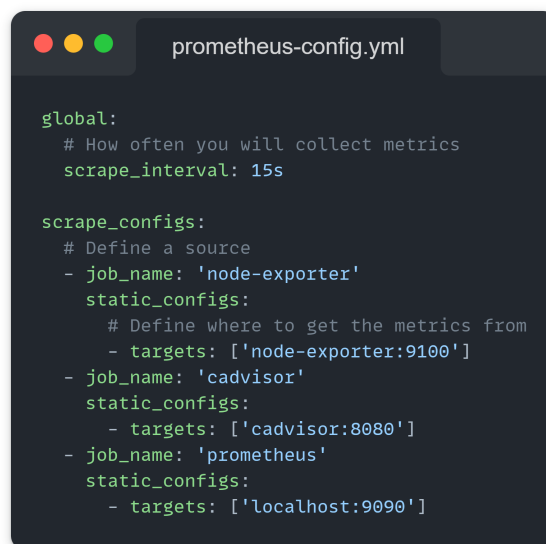


**Fig. 7.1** Display of cAdvisor graphs

As it is also desirable to have the metrics of the applications, of which there may be many, an attempt will be made to centralise everything in one place.

An open source platform called Prometheus [39] is used to track and notify services and systems. Operations and development teams are able to rapidly monitor and diagnose systems because to its expertise in the gathering and storage of performance indicators. Prometheus is ideal for real-time monitoring of applications and systems as well as for the production of alerts based on specific criteria since it is extremely scalable and has a flexible query mechanism.

For the example case (see Fig. 7.2), only three sources are consumed, one of which has not yet been mentioned.

A screenshot of a terminal window with a dark background and light text. The window title is 'prometheus-config.yml'. The content is a YAML configuration file for Prometheus. It includes a 'global' section for 'scrape\_interval' and a 'scrape\_configs' section with three jobs: 'node-exporter', 'cadvisor', and 'prometheus', each with its own 'static\_configs' and 'targets' list.

```
global:
  # How often you will collect metrics
  scrape_interval: 15s

scrape_configs:
  # Define a source
  - job_name: 'node-exporter'
    static_configs:
      # Define where to get the metrics from
      - targets: ['node-exporter:9100']
  - job_name: 'cadvisor'
    static_configs:
      - targets: ['cadvisor:8080']
  - job_name: 'prometheus'
    static_configs:
      - targets: ['localhost:9090']
```

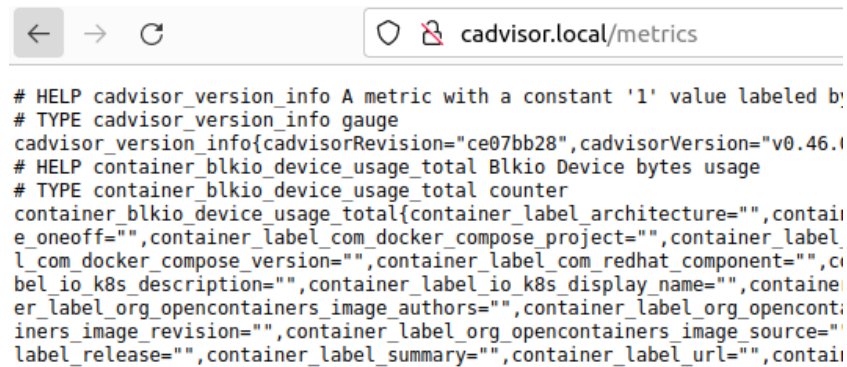
**Fig. 7.2** Configuration file "prometheus-config.yml"

Node Exporter [40] is a popular exporter used in the Prometheus ecosystem. Its primary function is to collect metrics from a machine's operating system and hardware and expose them in a format that Prometheus can collect and store. Some of the metrics that the Node Exporter can collect include CPU usage, memory, disk space, network performance and file system statistics.

Node Exporter is a further service added to Docker Compose. "Exporters" in the context of Prometheus are applications or services that collect system, application or component specific metrics and expose them in a format that Prometheus can collect and store. These exporters are essential to enable monitoring of custom metrics in a variety of environments. A collection of the most commonly used exporters is available on the official Prometheus website [41].

Prometheus uses a specific metrics format known as the Prometheus Metric Format or Prometheus Exposition Format. This format is quite simple and is designed to be readable by both humans and machines.

There are some services that directly export in this format, just like the three sources previously defined. For example, cAdvisor exports a large number of parameters (see Fig. 7.3).



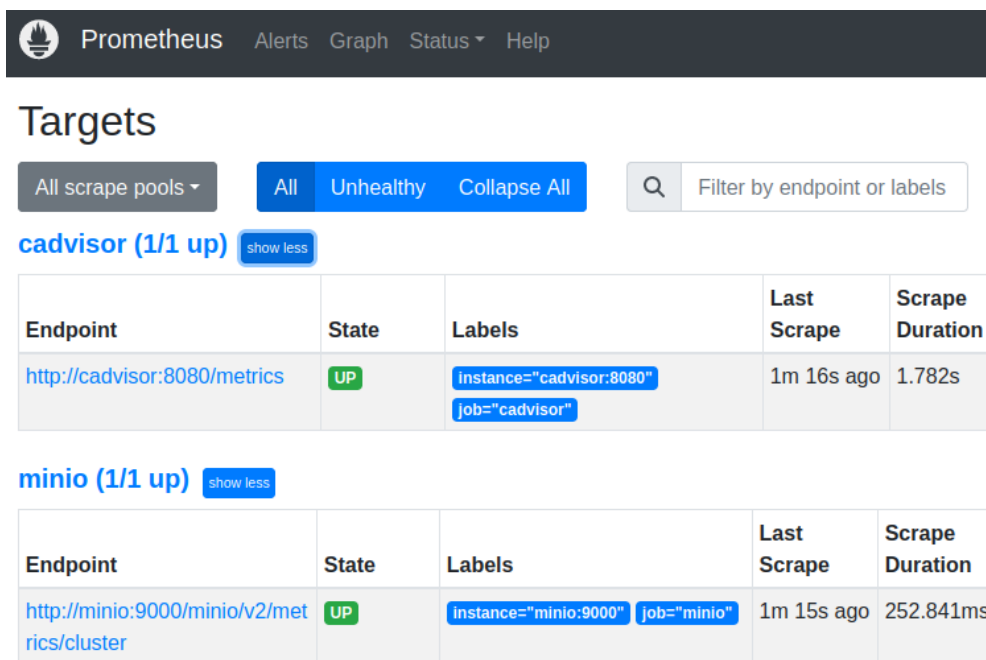
```
# HELP cadvisor_version_info A metric with a constant '1' value labeled by
# TYPE cadvisor_version_info gauge
cadvisor_version_info{cadvisorRevision="ce07bb28",cadvisorVersion="v0.46.1"} 1
# HELP container_blkio_device_usage_total Blkio Device bytes usage
# TYPE container_blkio_device_usage_total counter
container_blkio_device_usage_total{container_label_architecture="",container_label_com_docker_compose_project="",container_label_com_docker_compose_version="",container_label_com_redhat_component="",container_label_io_k8s_description="",container_label_io_k8s_display_name="",container_label_org_opencontainers_image_authors="",container_label_org_opencontainers_image_revision="",container_label_org_opencontainers_image_source="",container_label_release="",container_label_summary="",container_label_url=""}
```

**Fig. 7.3** Display of cAdvisor exported metrics

But this is not always the case:

- Ngnix: Requires an exporter [42] in addition to enabling metrics, which can be done using a modified image, and since one is already in use, the process is simplified.
- Oddo: Requires the "Prometheus Exporter" addon [43] to be installed.
- Postgres: Requires an exporter [44].
- Minio: Requires enabling the metrics [45], being easy if it is indicated with an environment variable that is made public and adding a specific path in the Prometheus configuration.

Temporarily, as there is no authentication, port 9090 will be exposed to see the Prometheus interface. Under "Status" - "Targets", all configured sources are listed, and if they are active (see Fig. 7.4).



**Fig. 7.4** Screen of the Prometheus targets

All available parameters, including graphics, can be consulted here, but that is not what Prometheus was designed for.

Grafana [46] is an open-source platform that allows the monitoring and visualisation of data. It enables the creation of dashboards, which include interactive panels and command grids that display data from a variety of sources, such as databases, online services, monitoring systems, etc. Grafana is widely used in IT operations and software development to create personalised data visualisations and make data-based decision-making easier.

Once started and port 3000 is exposed, or via the reverse proxy, the Grafana web interface can be accessed, where login with "admin" - "admin" is required.

A big option to add a data source will be displayed, but Prometheus will be selected. It is enough to enter the URL "http://prometheus:9090", the rest is left as default.

All that remains is to visualise the data. On the Grafana website there is a "Dashboards" section [47], where the community publishes its creations. They can be easily imported through an ID. The most interesting ones for this project are listed below:

- Node Exporter: 315
- cAdvisor: 10619 (see Fig. 7.5)

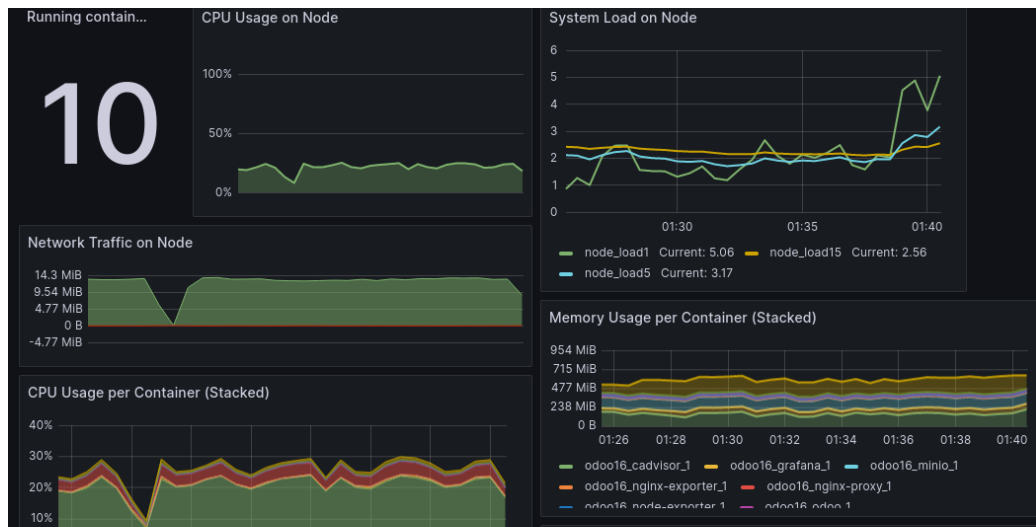


Fig. 7.5 cAdvisor dashboard screen in Grafana

- Prometheus: 3662
- Nginx: 12708
- Odoo: There are none, and there aren't many metrics either, but it's very easy to do (see Fig. 7.6):

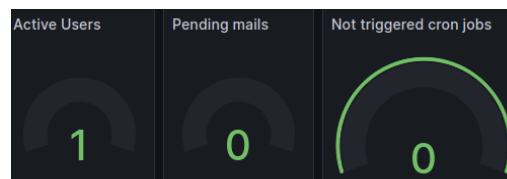


Fig. 7.6 Odoo dashboard screen in Grafana

- Postgres: 9628 (see Fig. 7.7)

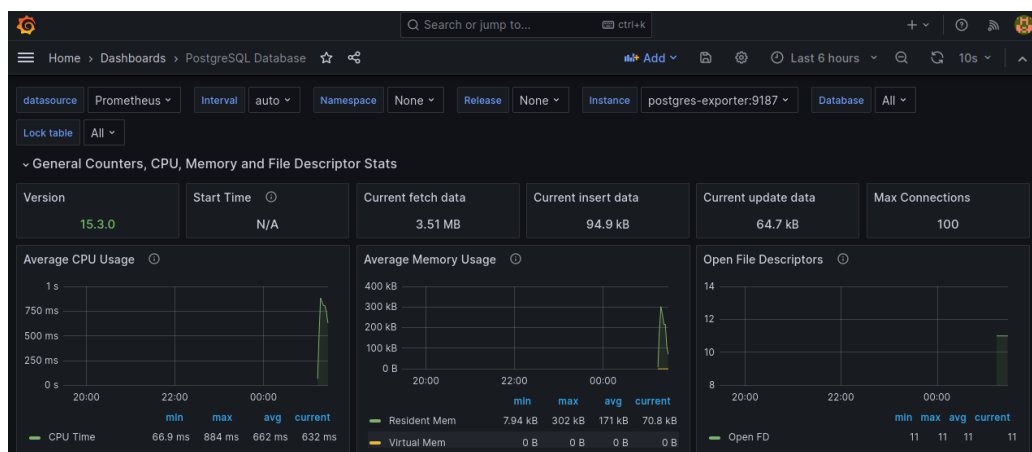


Fig. 7.7 Postgres dashboard screen in Grafana

- MinIO: 13502

With all this information available, it is useful to remix it all, creating a dashboard with the information needed.

To monitor several machines at the same time, as a more realistic environment, one would simply be consuming exporters on the different machines from a single Prometheus, or one Prometheus per machine, depending on the user's choice. The advantage is that with a single Grafana, all metrics could be centralised, even in a single Dashboard, depending on the configuration.

### 7.1.2. Logs

Logs are detailed records of events and activities occurring on a system. These logs can contain information about errors, transactions, user activities, configuration changes and much more. Logs are essential for diagnosing problems and tracking events in complex systems. They are often stored in text files or databases and can be searched, filtered and analysed for useful information.

Until recently, the industry standard for these issues was the ELK Stack [48], a toolset consisting of Elasticsearch, Logstash and Kibana, which focuses on the collection, indexing and visualisation of logs and log data, making it ideal for log analysis and troubleshooting. It covers a lot, but that does not detract from the fact that it is one of the best options when it comes to indexing content.

A relatively new stack is the Grafana Loki stack [49], consisting of, by way of comparison, Promtail, Loki and Grafana. Loki is the distributed log storage and query system, which is like Prometheus for logs, while Promtail [50] is the agent used to collect and send logs to Loki. Since an environment with Grafana is already available, it is advisable to take advantage of it.

To set up Loki, simply use the template they provide, there is not much provided, there is not much to explain. The file will be called "loki-config.yml".

Promtail has a configuration file with a very similar structure to Prometheus, little more than defining the targets. Here it should be noted that unlike Prometheus who pulls the different targets, here it is Promtail who pushes Loki. This is visible in the configuration (see Fig. 7.8).



```
promtail-config.yml

server:
  http_listen_port: 9080
  grpc_listen_port: 0

positions:
  filename: /tmp/positions.yaml

clients:
  - url: http://loki:3100/loki/api/v1/push

scrape_configs:

# local machine logs
- job_name: local
  static_configs:
    - targets:
      - localhost
    - labels:
        job: varlogs
        __path__: /var/log/*log

# docker logs
- job_name: docker
  pipeline_stages:
    - docker: {}
  static_configs:
    - labels:
        job: docker
        __path__: /var/lib/docker/containers/*/*-json.log
```

**Fig. 7.8** Configuration file "promtail-config.yml"

With this configuration, the logs of the local machine and the Docker logs are being consumed, but for that second point, some extra steps are required. A Docker Driver will have to be configured, so that the logs reach Loki (see Fig. 7.9).

```
# Install the Docker Loki Driver
vboxuser@ubuntu:~/odoo$ docker plugin install grafana/loki-docker-driver:latest
--alias loki --grant-all-permissions

# Modify the docker daemon
vboxuser@ubuntu:~/odoo$ sudo nano /etc/docker/daemon.json

{
  "debug": true,
  "log-driver": "loki",
  "log-opts": {
    "loki-url": "http://localhost:3100/loki/api/v1/push",
    "loki-batch-size": "400"
  }
}

# Reboot Docker and recreate the containers to apply the settings
vboxuser@ubuntu:~/odoo$ sudo systemctl restart docker
vboxuser@ubuntu:~/odoo$ docker-compose up -d --force-recreate
```

**Fig. 7.9** Docker driver configuration

The use of this driver, depending on the version, breaks the "docker-compose" command, so this must be taken into account.

Looking at port 3100, it can be seen that Loki metrics are also being exported (see Fig. 7.10).

```

loki_request_message_bytes_count{method="gRPC",route="/schedulerpb.SchedulerForQuerier/QuerierLoop"} 11
# HELP loki_response_message_bytes Size (in bytes) of messages sent in response.
# TYPE loki_response_message_bytes histogram
loki_response_message_bytes_bucket{method="GET",route="loki_api_v1_labels",le="1.048576e+06"} 1
loki_response_message_bytes_bucket{method="GET",route="loki_api_v1_labels",le="2.62144e+06"} 1
loki_response_message_bytes_bucket{method="GET",route="loki_api_v1_labels",le="5.24288e+06"} 1
loki_response_message_bytes_bucket{method="GET",route="loki_api_v1_labels",le="1.048576e+07"} 1
loki_response_message_bytes_bucket{method="GET",route="loki_api_v1_labels",le="2.62144e+07"} 1
loki_response_message_bytes_bucket{method="GET",route="loki_api_v1_labels",le="5.24288e+07"} 1
loki_response_message_bytes_bucket{method="GET",route="loki_api_v1_labels",le="1.048576e+08"} 1
loki_response_message_bytes_bucket{method="GET",route="loki_api_v1_labels",le="2.62144e+08"} 1
loki_response_message_bytes_bucket{method="GET",route="loki_api_v1_labels",le="+Inf"} 1
loki_response_message_bytes_sum{method="GET",route="loki_api_v1_labels"} 47
loki_response_message_bytes_count{method="GET",route="loki_api_v1_labels"} 1
loki_response_message_bytes_bucket{method="GET",route="other",le="1.048576e+06"} 2
loki_response_message_bytes_bucket{method="GET",route="other",le="2.62144e+06"} 2
loki_response_message_bytes_bucket{method="GET",route="other",le="5.24288e+06"} 2
loki_response_message_bytes_bucket{method="GET",route="other",le="1.048576e+07"} 2
loki_response_message_bytes_bucket{method="GET",route="other",le="2.62144e+07"} 2
    
```

**Fig. 7.10** Display of Loki metrics

The Grafana web interface will be accessed and a data source of type "Loki" will be added. It is sufficient to enter the URL "http://loki:3100", the rest is left as default.

With this, all that remains is to query the logs from the "Explore" section, selecting "Loki" as the source (see Fig. 7.11). Loki is a relatively complex query language, LogQL. However, Grafana has an interactive search where parameters can be added and filled in with the values proposed by a drop-down menu. For example, it is possible to query all the logs of the Docker compose. It can also be integrated with other Dashboards.

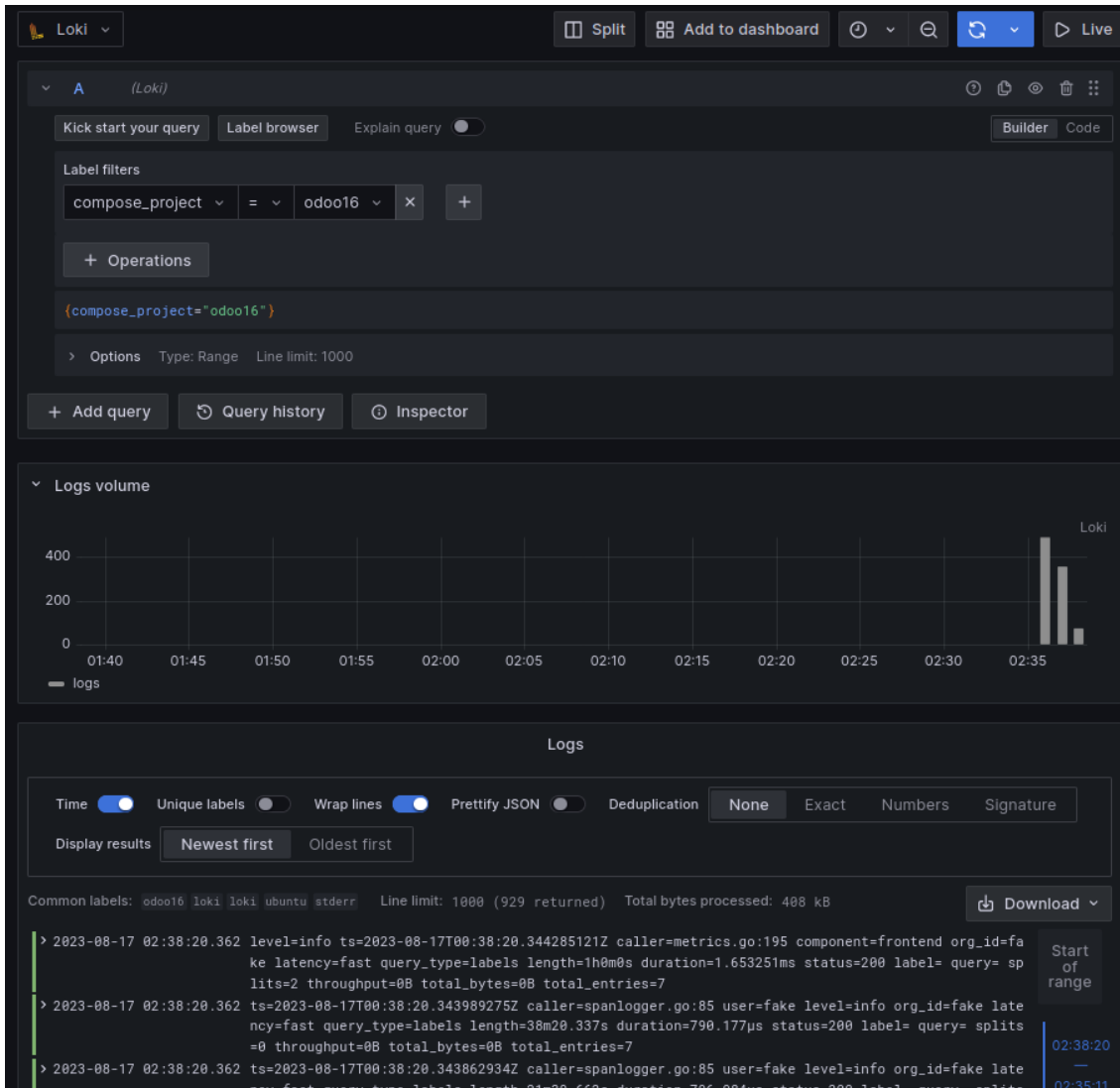


Fig. 7.11 Loki Explore screen in Grafana

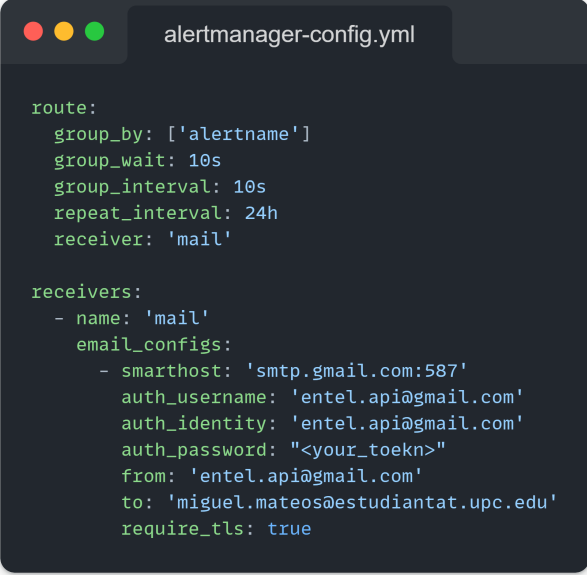
### 7.1.3. Alerts

Alerts are automatic notifications that are generated when anomalous conditions or events are detected in a system. These abnormal conditions are usually based on predefined thresholds or business rules. When a threshold is exceeded or a rule is met, an alert is generated to inform managers or operations staff. Alerts are crucial for early detection of problems and taking corrective action before they significantly affect the performance or availability of a system.

To continue within the ecosystem, Prometheus Alertmanager will be used, an essential part of the Prometheus ecosystem, designed to handle and manage alerts generated by Prometheus and other sources. Its main function is to take the generated alerts, apply deduplication, muting and enhancement rules, and then route them to specific notification channels, such as email, messaging

systems, chat services, among others. Integration with Grafana is also available.

Alertmanager has a configuration file (see Fig. 7.12) that is quite easy to understand. In it one can define intervals, and where one wants to send the alerts, either by email or Slack, among others.

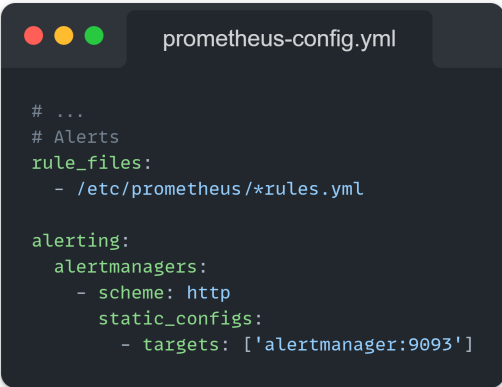
A screenshot of a code editor window titled "alertmanager-config.yml". The code is in a dark theme and shows the following configuration:

```
route:
  group_by: ['alertname']
  group_wait: 10s
  group_interval: 10s
  repeat_interval: 24h
  receiver: 'mail'

receivers:
- name: 'mail'
  email_configs:
  - smarthost: 'smtp.gmail.com:587'
    auth_username: 'entel.api@gmail.com'
    auth_identity: 'entel.api@gmail.com'
    auth_password: "<your_toekn>"
    from: 'entel.api@gmail.com'
    to: 'miguel.mateos@estudiantat.upc.edu'
    require_tls: true
```

**Fig. 7.12** Configuration file "alertmanager-config.yml"

In addition, the configuration of Prometheus (see Fig. 7.13) will have to be modified, so that it sends alerts, and uses some rules.

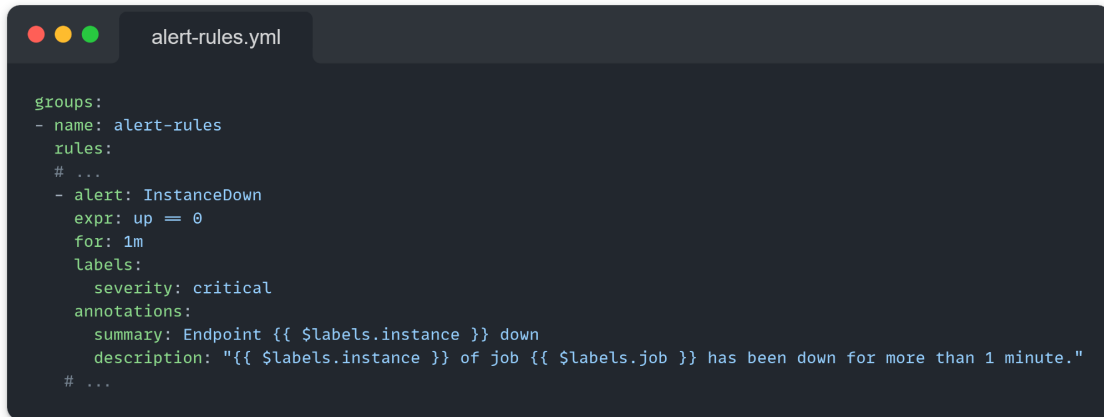
A screenshot of a code editor window titled "prometheus-config.yml". The code is in a dark theme and shows the following configuration:

```
# ...
# Alerts
rule_files:
- /etc/prometheus/*rules.yml

alerting:
  alertmanagers:
  - scheme: http
    static_configs:
    - targets: ['alertmanager:9093']
```

**Fig. 7.13** Updated configuration file "prometheus-config.yml"

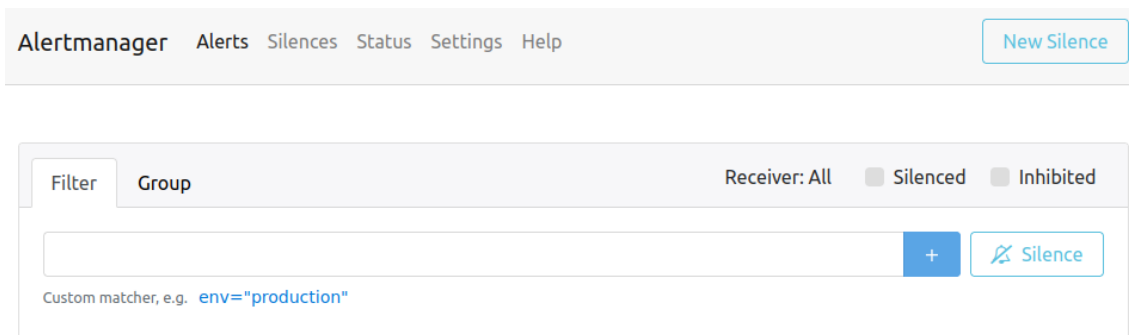
Next, a file with the rules under which the alerts will be sent must be created (see Fig. 7.14). Note that this file must be mounted in the Prometheus service, and not in the Alertmanager service.



```
groups:
- name: alert-rules
  rules:
  # ...
  - alert: InstanceDown
    expr: up == 0
    for: 1m
    labels:
      severity: critical
    annotations:
      summary: Endpoint {{ $labels.instance }} down
      description: "{{ $labels.instance }} of job {{ $labels.job }} has been down for more than 1 minute."
  # ...
```

**Fig. 7.14** Alerts configuration file "alert-rules.yml"

If it is desired to touch other configuration parameters, such as adding mutes, port 9093 can be exported (see Fig. 7.15).



**Fig. 7.15** Alertmanager main screen

In the following example, the MinIO container has been intentionally stopped, so that in less than a minute, an alert has been received in the mail (see Fig. 7.16).

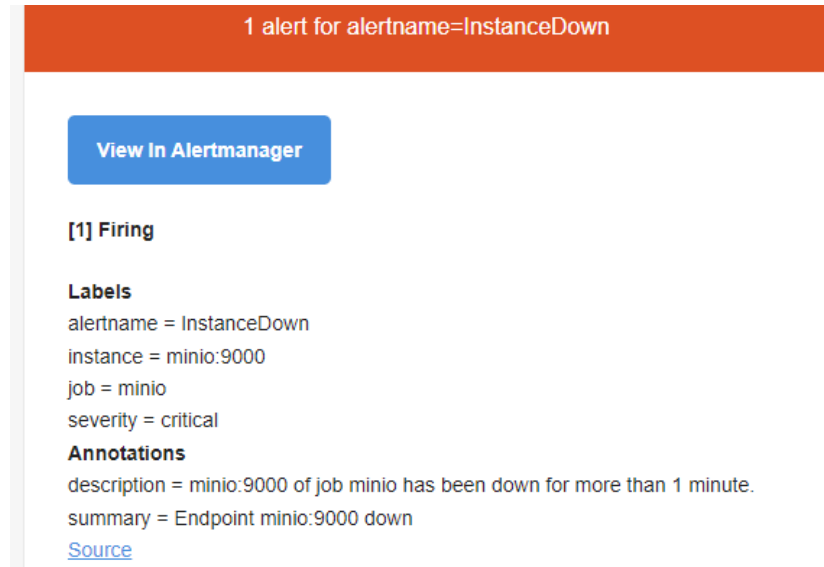


Fig. 7.16 Alert screen received by mail

Accessing Prometheus (see Fig. 7.17), the rules can also be viewed.

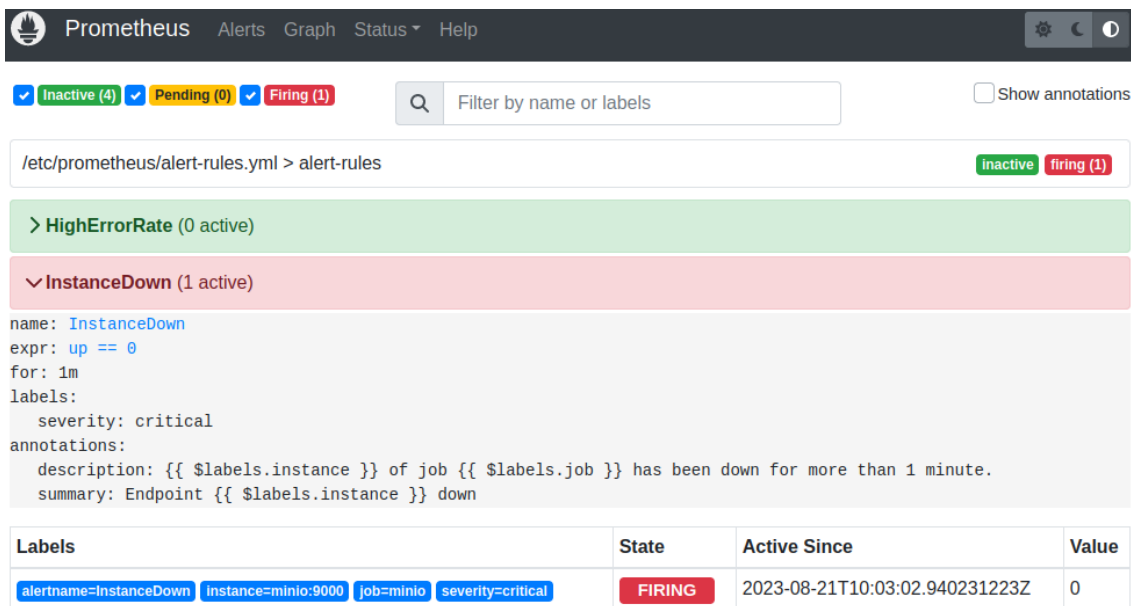
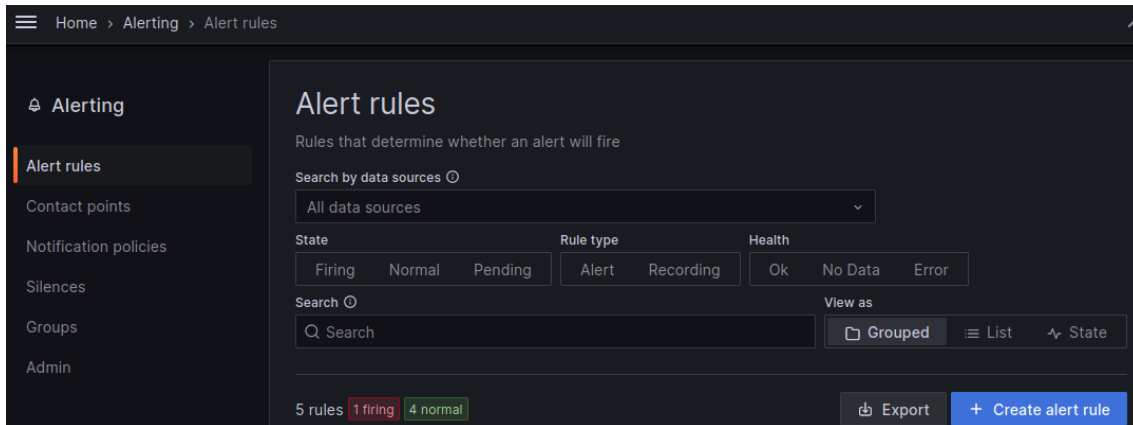


Fig. 7.17 Prometheus main screen

If Alertmanager is added as a data source in Grafana, in the same way as before, through "Alerting" it will have read permissions on the Alertmanager configuration (see Fig. 7.18). If it were managed by a Mimir [52], another Grafana service, the configuration could be modified. It should be noted here that Grafana also allows the creation of its own alerts, but these are independent of the rest.



**Fig. 7.18** Grafana alerting screen

## CHAPTER 8. CONCLUSIONS AND NEXT STEPS

In this final chapter, the conclusions drawn from the work are presented and possible future directions for the management of Odoo systems in educational and business environments are considered.

### 8.1. Conclusions

This work has provided a comprehensive overview of Odoo system management, focusing on critical aspects such as backup, version migration and monitoring. Key findings include:

- **Importance of Upgrading:** The importance of keeping Odoo systems up-to-date to take advantage of the latest features and security fixes has been highlighted. Regular updates ensure optimal performance and data security.
- **Effective Data Backup:** Implementing a robust backup strategy is essential. Minio, as a cloud storage solution, provides an effective and scalable way to back up critical Odoo data.
- **Version Migration with OpenUpgrade:** A detailed approach has been developed to migrate Odoo instances to new versions using OpenUpgrade. The ability to fix bugs in the process is critical to the success of the migration.
- **Constant Monitoring:** Continuous monitoring of an Odoo system is essential to detect and resolve problems efficiently. The collection of metrics, logs and alerts provides a complete picture of the performance and health of the system.
- **Monitoring Tools:** Prometheus, Grafana and Loki have been presented as valuable tools for monitoring. Alertmanager is used to manage and direct alerts to specific notification channels.

This work has contributed to the field of Odoo systems management in the aUPaEU educational environment by providing guidance and key tools to address common challenges. Specific contributions include:

- A detailed approach to version migration using OpenUpgrade, which facilitates the upgrade of Odoo instances in aUPaEU.
- The implementation of monitoring tools such as Prometheus, Grafana and Loki to improve visibility and efficiency in the management of Odoo systems in the context of aUPaEU.
- The presentation of best practices in Odoo data backup, with a focus on scalability and security for the aUPaEU educational platform.



Importantly, some of these contributions have been successfully incorporated into the project, demonstrating their relevance and value in improving the management of Odoo systems in the aUPaEU educational environment.

## 8.2. Next steps

The field of Odoo systems management is constantly evolving, and there are several promising directions for future research and improvements in the management of Odoo systems in educational and enterprise environments. Some possible future directions include:

- **Improving Migration Methods:** Investigating new tools and approaches that further facilitate Odoo version upgrades in the context of aUPaEU, especially in environments with extensive customisation.
- **Automation and Machine Learning:** Explore the implementation of artificial intelligence solutions for early problem detection and response automation in the management of Odoo systems in aUPaEU.
- **High Availability and Scalability:** Investigate high availability and scalability solutions to ensure uninterrupted performance in high demand aUPaEU educational environments.
- **Advanced Security:** Dive into security best practices and tools to ensure the protection of critical data on Odoo systems used in aUPaEU.
- **Detailed Documentation:** Create detailed guides and documentation for managing Odoo systems in aUPaEU, which can serve as resources for other professionals and students in the field.
- **DevOps and Continuous Delivery:** Explore the implementation of DevOps and continuous delivery practices to streamline the administration and maintenance of Odoo systems in aUPaEU.

These future tracks offer exciting opportunities to further advance the management of Odoo systems in aUPaEU and to address the evolving challenges faced by educational organisations in their implementation and maintenance of Odoo-based solutions.

## REFERENCES

- [1] AUPAEU - University Association for the Promotion of Education in Europe. Official website. Available at: <https://aupaeu.widening.eu>
- [2] European Universities Initiative. European Commission - Education and Training. Available at: <https://education.ec.europa.eu/education-levels/higher-education/european-universities-initiative>
- [3] Odoo. Official website. Available at: <https://www.odoo.com>
- [4] Odoo Community Association (OCA). Official website. Available at: <https://odoo-community.org>
- [5] OpenERP Spain Users Group. OpenERP users group in Spain. Available at: <https://groups.google.com/g/openerp-spain-users>
- [6] Odoo. "Hardware Requirements for Odoo 11". Odoo Forum. Available at: <https://www.odoo.com/forum/help-1/hardware-requirements-for-odoo-11-138936>
- [7] Odoo. "Multi-Company". Odoo Slides. Available at: <https://www.odoo.com/slides/slide/multi-company-1005>
- [8] IthinkUPC. Official website. Available at: <https://www.ithinkupc.com>
- [9] Odoo. "Odoo Installation Guide". Official documentation. Available at: <https://www.odoo.com/documentation/16.0/administration/install.html>
- [10] Docker. Official website. Available at: <https://www.docker.com>
- [11] Docker. Official Odoo image on Docker Hub. Available at: [https://hub.docker.com/\\_/odoo](https://hub.docker.com/_/odoo)
- [12] Docker. Official PostgreSQL image on Docker Hub. Available at: [https://hub.docker.com/\\_/postgres](https://hub.docker.com/_/postgres)
- [13] GitHub. "Docker Compose for Odoo". GitHub Gist. Available at: <https://gist.github.com/Guidoom/d5db0a76ce669b139271a528a8a2a27f>
- [14] Docker. "Docker Compose". Official documentation. Available at: <https://docs.docker.com/compose>
- [15] Odoo App Store. Odoo App Store. Available at: <https://apps.odoo.com/apps>
- [16] Odoo Community Association (OCA) on PyPI. OCA's page on Python Package Index. Available at: <https://pypi.org/user/OCA>

- [17] GitHub. "Odoo Docker". GitHub repository. Available at: <https://github.com/miguelmalu/odoo-docker>
- [18] Odoo. "Backend Module Reference". Official documentation. Available at: <https://www.odoo.com/documentation/16.0/developer/reference/backend/module.html>
- [19] Odoo. "Command Line Interface (CLI)". Official documentation. Available at: <https://www.odoo.com/documentation/16.0/developer/reference/cli.html>
- [20] Docker. Nginx Proxy image on Docker Hub. Available at: <https://hub.docker.com/r/jwilder/nginx-proxy>
- [21] Docker. Nginx Proxy Companion image on Docker Hub. Available at: <https://hub.docker.com/r/nginxproxy/acme-companion>
- [22] Let's Encrypt. Official website. Available at: <https://letsencrypt.org>
- [23] Docker. "Networking Overview". Official documentation. Available at: <https://docs.docker.com/network>
- [24] Docker. "Dockerfile reference". Official documentation. Available at: <https://docs.docker.com/engine/reference/builder>
- [25] GitHub Actions. GitHub Actions documentation. Available at: <https://docs.github.com/en/actions>
- [26] Fortinet. Product Support and Downloads. Available at: <https://www.fortinet.com/support/product-downloads>
- [27] Amazon S3. Amazon Simple Storage Service. Available at: <https://aws.amazon.com/s3>
- [28] GitHub. "Odoo Automatic Backup". GitHub repository. Available at: [https://github.com/Yenthe666/auto\\_backup](https://github.com/Yenthe666/auto_backup)
- [29] Docker. MinIO image on Docker Hub. Available at: <https://hub.docker.com/r/minio/minio>
- [30] Docker. MinIO Client (mc) image on Docker Hub. Available at: <https://hub.docker.com/r/minio/mc>
- [31] AWS CLI. Amazon Web Services Command Line Interface. Available at: <https://aws.amazon.com/cli>
- [32] Odoo App Store. "Auto Backup to AWS S3". Available at: [https://apps.odoo.com/apps/modules/12.0/auto\\_backup\\_aws\\_s3](https://apps.odoo.com/apps/modules/12.0/auto_backup_aws_s3)
- [33] Boto3 Documentation. Boto3 - The AWS SDK for Python. Available at: <https://boto3.amazonaws.com/v1/documentation/api/latest/index.html>

- [34] Odoo App Store. "Storage Backend S3". Available at: [https://apps.odoo.com/apps/modules/15.0/storage\\_backend\\_s3](https://apps.odoo.com/apps/modules/15.0/storage_backend_s3)
- [35] OpenUpgrade. Official OpenUpgrade documentation. Available at: <https://oca.github.io/OpenUpgrade>
- [36] Python Package Index (PyPI). BeautifulSoup4. Available at: <https://pypi.org/project/beautifulsoup4>
- [37] Python Package Index (PyPI). Odoo Module Migrator. Available at: <https://pypi.org/project/odoo-module-migrator>
- [38] Docker. Google cAdvisor image on Docker Hub. Available at: <https://hub.docker.com/r/google/cadvisor>
- [39] Docker. Prometheus image on Docker Hub. Available at: <https://hub.docker.com/r/prom/prometheus>
- [40] Docker. Node Exporter image on Docker Hub. Available at: <https://hub.docker.com/r/prom/node-exporter>
- [41] Prometheus. "Exporters and Integrations". Available at: <https://prometheus.io/docs/instrumenting/exporters>
- [42] GitHub. "Nginx Prometheus Exporter". GitHub repository. Available at: <https://github.com/nginxinc/nginx-prometheus-exporter>
- [43] Odoo App Store. "Prometheus Exporter". Available at: [https://apps.odoo.com/apps/modules/16.0/prometheus\\_exporter](https://apps.odoo.com/apps/modules/16.0/prometheus_exporter)
- [44] Docker. PostgreSQL Exporter image on Docker Hub. Available at: <https://hub.docker.com/r/prometheuscommunity/postgres-exporter>
- [45] MinIO. "Collect MinIO Metrics Using Prometheus". MinIO documentation. Available at: <https://min.io/docs/minio/linux/operations/monitoring/collect-minio-metrics-using-prometheus.html>
- [46] Docker. Grafana image on Docker Hub. Available at: <https://hub.docker.com/r/grafana/grafana>
- [47] Grafana. Grafana Dashboards. Available at: <https://grafana.com/grafana/dashboards>
- [48] Elastic. Elastic Stack. Available at: <https://elastic.co/elastic-stack>
- [49] Docker. Loki image on Docker Hub. Available at: <https://hub.docker.com/r/grafana/loki>

[50] Docker. Promtail image on Docker Hub. Available at:  
<https://hub.docker.com/r/grafana/promtail>

[51] Docker. Alertmanager image on Docker Hub. Available at:  
<https://hub.docker.com/r/prom/alertmanager>

[52] Grafana. "Mimir Documentation". Available at:  
<https://grafana.com/docs/mimir/latest>

## BIBLIOGRAPHY

Greg Moss, *Working with Odoo*, Packt Publishing, 2015

Parth Gajjar, Alexandre Fayolle, Holger Brunn, Daniel Reis, *Odoo 14 Development Cookbook*, Packt Publishing, Fourth Edition, 2020