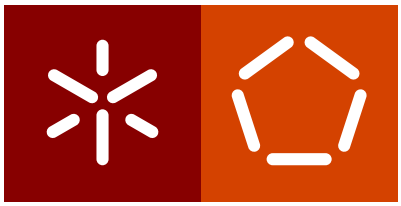
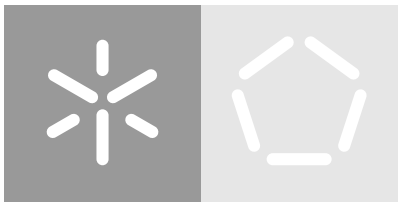


Universidade do Minho
Escola de Engenharia
Departamento de Informática

Rui Nuno Borges Cruz Oliveira

Scientific data management

October 2022



Universidade do Minho
Escola de Engenharia
Departamento de Informática

Rui Nuno Borges Cruz Oliveira

Scientific data management

Master dissertation
Master Degree in Integrated Master's in Informatics Engineering

Dissertation supervised by
José Orlando Roque Nascimento Pereira

October 2022

Copyright and Terms of Use for Third Party Work

Acknowledgments

Throughout the writing of this dissertation, I have received a great deal of support and assistance.

I would like to express my sincere gratitude to my supervisor, Professor José Orlando Pereira. Without his guidance and assistance throughout the process, the outcome of this dissertation would not be the same.

While doing this work, I had a Research Grant from INESC TEC funded by National Funds, so I also want to thank FCT (Fundação Portuguesa para a Ciência e Tecnologia) and INESC TEC. In particular, a word of thanks to Susana Barbosa, researcher at INESC TEC and one of the coordinators of the SAIL Project, her assistance was invaluable.

To my family, I would like to express my deep gratitude for always being there and for always supporting me throughout my academic journey, providing everything I needed. Certainly, without them, these last five years would not have been possible.

A special thanks to my friends, in particular those who have lived this experience closely, they have made all the effort rewarding. These five years will always be remembered for the beautiful stories we lived together.

The last word of appreciation goes to my dear Lídia, for her unconditional support at all times. I am thankful that she appeared on this journey and never let me down. Thank you for always encouraging me to be the best version of myself.

Statement of Integrity

I hereby declare having conducted this academic work with integrity.

I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

Abstract

Universally, it is known that data is constantly changing, new concepts emerge and ways to represent them, and sometimes many of these concepts become deprecated. In the context of scientific research, data management is no exception, as it becomes more challenging to deal with the growing volume of data and with the complexity of the surrounding techniques. Furthermore, it is common in companies, as well as in scientific research, a huge dependence on those who are responsible for the management of all the information, leading to the possibility that team changes are not as efficient as one would expect. It is therefore crucial to analyse, organise and document data.

There are several techniques developed over the last years to deal with this problem, and the main focus of this dissertation is to adapt and apply the most appealing in the scrutiny of the currently known systems to handle the management of scientific data. These systems will be compared considering the following criteria: architecture, interoperability, metadata, usability and security. Finally, after adapting one of the solutions, a performance evaluation will be required in order to extrapolate its functioning in a real-world context.

Keywords Scientific data management, research, CKAN, EUDAT

Resumo

Universalmente, sabe-se que os dados estão em constante mudança, surgem novos conceitos e formas de os representar, e por vezes muitos destes conceitos são depreciados. No contexto da investigação científica, a gestão de dados não é excepção, pois torna-se mais desafiante lidar com o volume crescente de dados e com a complexidade das técnicas que os rodeiam. Além disso, é comum nas empresas, bem como na investigação científica, uma enorme dependência daqueles que são responsáveis pela gestão de toda a informação, levando à possibilidade de as mudanças de equipa não serem tão eficientes como seria de esperar. É, por conseguinte, crucial analisar, organizar e documentar os dados.

Há várias técnicas desenvolvidas nos últimos anos para lidar com este problema, e o foco principal desta dissertação é adaptar e aplicar o mais apelativo na análise dos sistemas actualmente conhecidos para lidar com a gestão de dados científicos. Estes sistemas serão comparados considerando os seguintes critérios: arquitectura, interoperabilidade, metadados, usabilidade e segurança. Finalmente, após a adaptação de uma das soluções, será necessária uma avaliação de desempenho a fim de extrapolar o seu funcionamento num contexto do mundo real.

Palavras-chave Gestão de dados científicos, investigação, CKAN, EUDAT

Contents

List of Figures

List of Tables

Glossary

API Application Programming Interface.

CKAN Comprehensive Knowledge Archive Network.

DMS Data Management System.

EUDAT European Association of Databases for Education and Training.

GNSS Global Navigation Satellite System.

HPC High Performance Computing.

INESC TEC Institute for Systems and Computer Engineering, Technology and Science.

MACC Minho Advanced Computing Centre.

NMEA National Marine Electronics Association.

SAIL Space-Atmosphere-Ocean Interactions in the marine boundary Layer.

SDMS Scientific Data Management System.

Chapter 1

Introduction

Collaboration plays a crucial role in scientific innovation because the problems faced are of such complexity that no single individual or collective has all the relevant information, knowledge or resources (Loshin, 2010). Scientific research depends on an efficient and organised means of technological communication so that research can be exploited and improved to its full potential (Wilkinson et al., 2016). This requires special consideration of data resulting from investigation, regardless of whether they are large or small.

The better an organisation understands its data, the better it can use it and consequently share it with the scientific community and allow the same data to be repeated and improved. Hence, we can effectively say that the speed at which science evolves today depends on how and whether data sharing is done (Candela et al., 2015).

While at first glance it may seem obvious that all research collectives would like to make global collaboration a widespread practice, since we are talking about the evolution of science, this is not necessarily the case (Tenopir et al., 2015). In essence, science is also an industry, and the investment of time and money is something that has to be put on the table and made as efficient as possible (Rafes and Germain, 2015). As such, instruments to implement information sharing have to meet certain criteria for research institutions to invest in them.

1.1 Problem

Every year new topics appear on the agenda of researchers and scientists, and all over the world research teams dedicate themselves publicly or privately to the most varied

problems. The same research may be carried out by different organisations, which, with different resources and ways of working, produce huge volumes of data and make subjective analyses of them.

This reality led to the emergence of the problem of how the management of the work produced should be carried out and, with this, generated the question of the need for mechanisms and systems that allow the way of working to be universalised (Gabelica et al., 2022). Thus, the issue better known as scientific data management arose.

There are several known solutions to this problem, and the systems responsible for this management may or may not have research data as a priority, the important thing is to understand if the functionalities they support are the most useful in solving the problem. Basically, the choice of a solution depends on the particularities of the research of the promoting entities, so before choosing a system, it is necessary to define the aspects to be evaluated (Lee and Stvilia, 2017).

Specifically, this dissertation will take as a case study the SAIL project carried out by INESC TEC on board the School-Ship Sagres, the object of study of the project being atmospheric electricity and climate change, and aims to develop a scientific observation platform to collect data on the ocean and atmosphere (Barbosa and Karimova, 2021). This will allow us to have a concrete idea of what a system should provide for a specific research case.

Despite the significant growth of existing solutions in the market, most scientific institutions, especially those linked to universities, still have their own data repository. This is due to the fact that performance capacity is something to be considered, since unlimited fast access at any time requires a certain amount of computing power, particularly if there is the premise of growing the repository and with it the accesses to it (Armbruster and Romary, 2010). Therefore, it would be interesting to understand how a solution can be integrated into supercomputing and big data infrastructures, more specifically, with the help of the MACC, which is developing the installation of the new Portuguese supercomputer Deucalion at AvePark, in Caldas das Taipas, Guimarães. In this case, we have to consider that the data stored in the system to be developed should be accessible through compute servers. For example, a File System (FS) such as Lustre (an open source parallel FS that supports many requirements of HPC simulation environments) should be used for its data storage in a seamless and transparent manner.

1.2 Objectives

Firstly, the objective of this dissertation is to understand how some of the existing software systems and international cooperation networks work to solve the problem mentioned, and, more than understanding how they work, to understand, concretely, what they offer.

With this in mind, once the evaluation is done, the objective is to adapt and apply an existing system so that it can correspond in the most efficient way to scientific research, with special attention to the particularities promoted by the case study of this dissertation, the SAIL project.

Finally, it will be necessary to evaluate the performance of the developed system, important to understand how it might behave in a real context. For this, ideally we would use MACC, which, by promoting and supporting open scientific initiatives on advanced computing, data science and visualisation, could play a fundamental role in this project in terms of understanding how to adopt a system to real scenarios of considerable computing power. However, in the event that it is not possible for us to use MACC's resources, since the installation of Deucalion may still take some time, we will use the resources of INESC TEC, which, although it does not allow us to have the understanding that MACC would make possible, serves to give us a notion of how the system we develop works and will allow us to extrapolate its use, after the writing of this dissertation, in an infrastructure like MACC.

1.3 Structure

This dissertation is divided into several chapters, and the first provides the context, problem and general objectives of the work. Next, Chapter 2 allows us to review the state of the art, with the background and related work. In Chapters 3 and 4 we expose all the details of the implemented solution, how to use it and the results of the performance evaluation. Finally, the Chapter 5 presents the conclusions drawn and guidelines for future work.

Chapter 2

State of the Art

This chapter introduces the state of the art of this dissertation, which is divided between two sections: background and related work.

Firstly, we provide an overview of the use of DMSs, as well as their different models and approaches. Furthermore, we provide a contextualisation regarding the case study under consideration, describing its particularities. Finally, we discuss the different criteria for evaluating a SDMS as a response to the problem of scientific data management.

Secondly, we thoroughly analyse each of the three systems considered in this work, that is, *CKAN* (ckn), *EUDAT* (eud) and *Magda* (mgd), and subsequently we perform an evaluation and comparison between them to understand which is best suited for managing scientific data, with special attention to our case study.

2.1 Background

2.1.1 DMS

DMS provides a self-service environment for users to access, define, and manipulate data. The powerful tools in such a system use metadata management to automate data organisation, to show relationships between data, and some even include functionality for government, business, or scientific data (Gray et al., 2005).

Databases are the most common platform for data management, but file systems or cloud object storage services are also used, which generally offer greater flexibility over databases since they organize data in a less structured way.

DMSs have evolved and now present several options regarding the data model. A data

model is an abstract model that organises data elements, documents how data is stored and retrieved and designs the necessary response to information system requirements. The main existing models are as follows:

- Hierarchical data model, each component has a child/parent relationship with another component;
- Network data model, each component can have multiple relationships;
- Relational data model, each component has attributes that are linked to their identities through a table structure.

Other data models include graph database, relation-entity, and object-oriented.

This dissertation will focus on SDMSs, which aim to increase the productivity of any research without neglecting any sharing and collaboration efforts. SDMS is a better approach to handle unstructured data, which includes images, PDF files, spreadsheets, geographic coordinates, and other data formats resulting from an investigation.

2.1.2 SAIL Project

The case study focus on measuring the electric field of the atmosphere over the ocean and monitoring gamma, solar and cosmic radiation or GNSS signals, as well as taking into account other aspects such as temperature and pH. That being said, we can see that the study has a wide diversity of data, and it is also important to consider the fact that this campaign collected data every second, with hourly records for 371 days.

We needed to understand what data would need to be considered and how they are organised in order to adapt the system we intended to develop. Therefore, in order to find an answer to this point, we tried to meet with the project's research team, and we were able to meet with one of the members who informed us about the campaign data we need to work with:

- GNSS data, which refers to a constellation of satellites providing signals from space that transmit positioning and timing data to GNSS receivers. GNSS data can be represented in different formats – binary, RAWIMUX, and NMEA – with NMEA being the one to pay particular attention to when testing the volume of data;

- Atmospheric data, specifically electric field (E1 and E2), gamma radiation (GA), visibility (VI) and radiation (inSL and outSL), which constitute a smaller sample compared to the first type, referring to 30/60 days. These data are more interesting and easier to work with, since they were previously processed and subjected to quality control procedures.

In order to organise the data, a conventional nomenclature was adopted, consisting of: activity type + infix data group + infix measured parameter. Regarding the GNSS data we will explore, these are divided into 2 groups: ship data (SHIP) and sensor data (SD), and atmospheric data are in the pre-processed data (PD) group. Therefore, the following is the nomenclature that will be used in the datasets and system files:

- GNNS - SAIL_SHIP_GNSS_*
- GNNS NMEA standard - SAIL_SD_GNSS_NMEA_*
- Atmospheric fields - SAIL_PD_E1_*, SAIL_PD_E2_*, SAIL_PD_GA_*, SAIL_PD_VI_*, SAIL_PD_inSL_*, SAIL_PD_outSL_*

The * above denotes two possibilities – yyyymmddd for compressed files or yyyymmddd_hh for files – where yyyy is the year, mm is the month, dd is the day and hh is the hour.

There were two antennas collecting information from the Sagres ship and each compressed file includes information collected per day by those antennas. Each compressed file includes files with data associated with each hour of the day, i.e. a total of 48 files per day.

2.1.3 Key Criteria for SDMSs

Key criteria are defined by three dimensions: architecture, interoperability & metadata, and usability & security.

Table 2.1: Key criteria by dimension

Architecture	Interoperability & metadata	Usability & security
Installation and deployment	APIs	Popularity
Open source	Harvesting protocols	Advanced search
Storage replication	Standard schemas	Geospatial tools
Storage location	Validation	Authorised authentication
Customisation	Federation	Curation workflow
	Digital Object Identifier (DOI)	

Architecture

The installation and setup of any system should be as easy and flexible as possible. This work can usually be done through some repository of the institution that owns the platform or by outsourcing an external service (*Docker*, for example). Moreover, free access to the source code allows a better study and, consequently, understanding of the developed software. Furthermore, the fact that the code is not open source exposes the possibility that the platform's support is not done indefinitely and depends on third parties (Carvalho Amorim et al., 2016).

In a specific case for any DMS, if its database fails for a moment, it will not be a problem if we consider database replication, otherwise we may lose all the information. As such, storage replication will also be an important aspect to consider.

Finally, other interesting aspects to consider are the location where the data is kept (remotely or locally), and the ability of the system to be customised as well as extended. The latter allows the community to develop scientific data management solutions that are not restricted by the existing functionalities.

Interoperability and metadata

On the one hand, the existence of APIs in a scientific data management solution, allows its integration into external systems, which grants greater visibility and subsequent improvement of its content. On the other hand, metadata interoperability can be facilitated through compliance with standard protocols for harvesting metadata, such as the Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH) (Carvalho Amorim et al., 2016), with harvesting metadata referring to the automated collection of metadata description from several places to generate useful aggregations of their information.

Along with protocols such as OAI-PMH, metadata standardisation can be useful, because if everyone uses different standards it can be very difficult to compare and analyse data (Gray et al., 2005). Moreover, to ensure that the data to be analysed is correct, data validation is advantageous when creating any dataset.

Federation with a web service should be possible, since the creation of a federated network of data portals that share and disseminate information between each other is very useful, as it provides the interoperability, consistency, and access controls needed to enhance the cooperation and security policies between the different research centers across multiple networks.

From a long-term perspective, assigning a persistent unique identifier to the resources that are stored, like Digital Object Identifier (DOI), is also very important, as it persists with each resource forever, even if its location or form of storage changes over time, which also makes it easier to cite resources that could be extremely useful to any researcher.

Finally, the resources available in the system may be used freely or not, since any user holding a given dataset has the right to protect the use of its content. Therefore, the platform must guarantee the possibility of granting a licence to any data.

Usability and security

Lastly, there are also a few more criteria in a more general overview that also deserve attention. First, as any solution to any problem on the market is also evaluated by its popularity, SDMSs are no exception, as it is natural to assume that greater popularity leads to greater use and validation (or not) of the product. In addition, it enhances the diffusion of the solution and increases the size of the community, which allows for an improvement in knowledge and consequent usability for the system in question.

Another relevant factor to usability is its search, the aim being as intuitive and intelligent as possible. More than allowing a simple search, it is important to consider advanced search features, as it offers a greater number of tools for any researcher to find what they want more easily. Likewise, advanced geospatial features can be advantageous to use in the context of scientific investigations.

Some security policies can be offered to organisations that use data management solutions, one of which is authorised authentication, in other words, it is possible that these same organisations control who can carry out actions on their resources. For example, authorisation systems can manage who can register/delete new users, or who can edit/create/delete datasets.

Usually, researchers are not experts in data management, therefore, they need intuitive and efficient tools to produce the intended results without the need for complex learning about the functioning of an effective solution to the problem of managing scientific data. Not to mention that the available workflow has to be viable both in the immediate and long term to scientific research, that is, curation workflow.

In short, any solution that is easy and safe to use is closer to being adopted in the daily work of an investigator.

2.2 Related Work

2.2.1 *CKAN*

CKAN is an open-source data management solution whose main objective is to allow the creation, access, and dissemination of knowledge through tools for the use and reuse, publication, search and share of information. Generally, it is implemented by organisations and is used by several national governments to manage their data, but it can also be used by companies, research institutions, or even individuals. Currently, *CKAN's* design demonstrates a versatile range of options, and can be used as a catalog, repository, datastore, or as a combination of all three (Winn, 2013).

Architecture

CKAN follows the classic *Model View Controller (MVC)* pattern. There is a clear distinction between the various layers, where the data representation and client interaction,

i.e. the View, are separated from the methods that interact with the databases, i.e. the Model, with the Controller mediating between the two through the changes that the user interacting with the system wants. This architecture is shown in and the Figure 2.1.

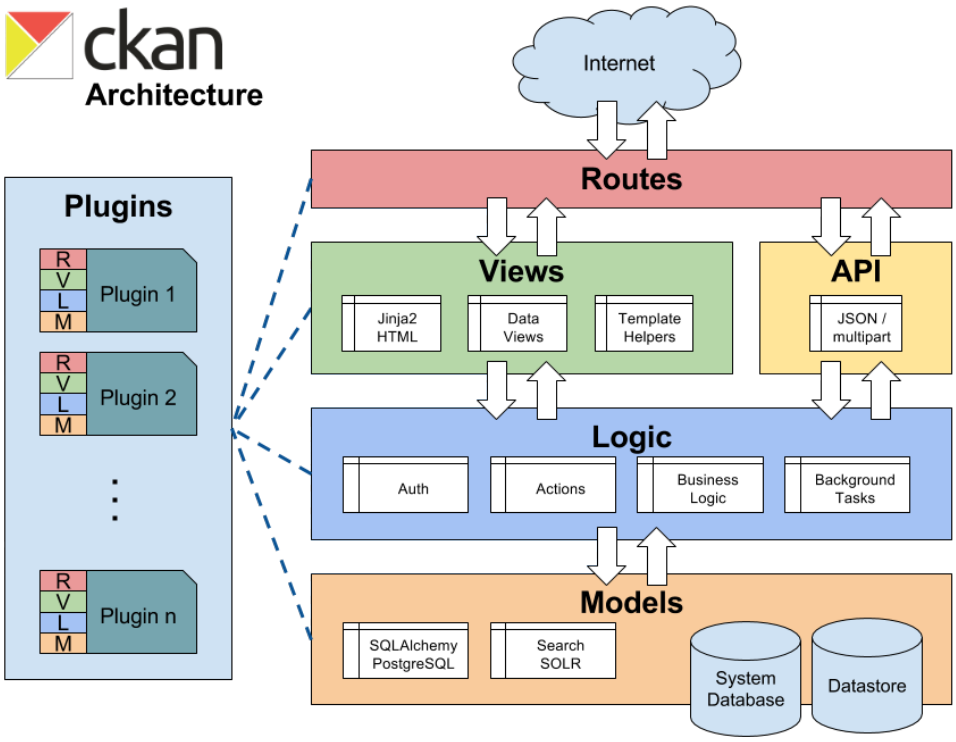


Figure 2.1: *CKAN* Code Architecture (http://docs.ckan.org/en/2.9/contributing/architecture.html)

CKAN is highly customisable and encourages the extension of content to the code base to adapt the system to different needs. The community and the *CKAN* development team already have a vast list of extensions available.

Installation

The best option for *CKAN's* installation, which is currently in version 2.9 (version by which the evaluation of this platform will be guided), depends on the intended use of it. According to the documentation that the system provides, there are three ways to install it:

- Install from an operating system package;
- Install from source;

- Install from *Docker Compose*, a tool for defining and running multi-container *Docker* applications;

Installing from the package is the fastest and easiest way to get *CKAN*, as it requires either the 18.04 64-bit or 20.04 64-bit version of Ubuntu. It is still necessary to install *Solr* for the research tools and *PostgreSQL* for the database. Moreover, this system is primarily written in *Python*.

When the installation is well completed, two web applications will be running, *CKAN* itself and the *Datapusher*, a distinct service for automatically importing data to *CKAN's DataStore* extension.

Datasets, Role management and Authorisation

With *CKAN*, data is published in units called datasets, which contain two things: information relating to the data, in other words, metadata, and resources that contain the data, which can be stored as relational database entries or separate files (XML files, images or linked data in RDF format, for example).

Unlike other cloud services, *CKAN* does not use a distributed data storage, since multi-user concurrency is not a major requirement in the way this software handles information. Instead, data integrity is generally favoured by data publishers to avoid data conflicts. Hence, *CKAN*'s team decided to implement a central data store to meet their needs (Aué et al., 2016).

Typically, each dataset belongs to an organisation and each *CKAN* instance can have numerous organisations. Considering, for example, the case of a scientific investigation that is being carried out by a research center that is using *CKAN*, there may be different departments publishing data. In this case, each organisation can define its way of working and specific authorisations, providing tools to manage its data ingestion process. Thus, the platform becomes very versatile and there is no requirement for centralised management.

Within each organisation, there can be different levels of access, where admins can add users and specify the level of access for each one.

Despite the above, it is also possible to configure *CKAN* with the objective that the datasets do not belong to any organisation, which opens up the possibility of a wiki-like *DataHub*.

Metadata

By default, *CKAN* offers the following basic set of metadata:

- Title;
- ID;
- Groups;
- Description;
- Revision history;
- License;
- Tags;
- Formats;
- API key.

It also allows other attributes, such as location data through the geographic feature or customised information relevant to the publisher or dataset. Although the *CKAN* metadata does not follow any standard schema, the system, through the above-mentioned customisation, grants the inclusion of arbitrary key/value pairs, which can only be represented by strings (Carvalho Amorim et al., 2016).

FileStore and *DataStore*

CKAN's FileStore allows users to upload files, which by default are stored locally on the server, so that *CKAN* has more control over the files and manages access to them. However, some extensions allow the user to save files remotely, such as on *Amazon S3* (Winn, 2013). Presently, only one storage location can be used.

The *DataStore* extension provides a database to store structured data of *CKAN* resources, and data can be extracted from files and saved in the *DataStore*, whether initially in the *FileStore* or external links. The upload work of both possible resource sources is from *Datapusher*.

Therefore, there are two distinct ways of storing *CKAN* resources. On the one hand, there is the *FileStore* which provides a storage of files where access or filtering to specific

parts is not possible. On the other hand, the *DataStore* is like a database where accessing and filtering data elements is possible. For example, a file of the CSV type, in *FileStore* would be stored as a whole and to access its content would be necessary a download, while in *DataStore* it would be possible to access each line of the content of a CSV spreadsheet individually via an API, as well as filtering content. Although *CKAN* does not support data replication, it is possible to say that a partial replication of certain datasets can often occur under the simultaneous use of these two storage components.

Interoperability

Through its APIs, this system offers tools for developers who want to interact with *CKAN* sites or their data. In addition to the *CKAN* Action API, an RPC-style API that exposes the core functionalities of a *CKAN* website to any external code, *FileStore*, and *DataStore* have their APIs.

Despite the interoperability that allows via APIs, *CKAN* fails to support, by default, compliance with standard protocols used by the scientific community such as the OAI-PMH. Furthermore, metadata records do not follow any specific schema, despite allowing the use of an extension that exposes and consumes metadata using Resource Description Framework (RDF) documents serialized using Data Catalog Vocabulary (DCAT), which is an RDF vocabulary developed to facilitate the interoperability between data catalogs (Carvalho et al., 2016; Winn, 2013). Thus, *CKAN* makes it possible to expose and consume metadata from other data catalogs with standards schemas, but at no point does it include their validation, which in this case is done by the entities that originate the data.

This system can be used to create a federated network of data portals that share and manage data with each other. This is achieved through the *CKAN* harvesting extension, and it is even possible to federate data from *non-CKAN* catalogs with the aforementioned DCAT extension.

Search, preview and visualization tools

CKAN provides a search interface similar to *Google*, allowing search by word, delimited or not by quotation marks. Users can easily get a list of available datasets and perform searches on them. There are several search options, which can all be performed via API:

- Search on all dataset attributes;

- Full-text search;
- Fuzzy-matching, option to find terms close to the one entered instead of exact terms;
- Faceted search, in other words, searching through specific fields, such as tags.

This platform can automatically preview some data types and extend an interactive analysis before downloading. The main features of content visualization include graphics, tables, maps (via advanced geospatial features), or images, and it is also possible to create new forms of preview for other types of data through extensions.

2.2.2 *Magda*

Magda is an open-source data catalog system that provides a single space for an organisation's data to be stored, cataloged, searched, and enriched, whether large datasets or small files. This system is designed around the concept of federation and with the flexibility to be used as a catalog of big data in a data lake, an easily searchable repository for small data files, an aggregator of data from multiple external sources, or all in one.

Architecture

One of *Magda's* main characteristics is its architecture, which is presented in a diagram in Figure 2.2 and consists of an extensible set of micro services that are distributed as *Docker* containers, in the sense that new services can be added allowing organisations to go beyond the standard system. Any extension to collect data or enrich metadata can be written in any language and added/removed without significant changes to the operation of the core system. The architecture of *Magda* is presented in a diagram in Figure 2.2.

The use of *Kubernetes*, for orchestrating *Docker* containers, and *Helm*, which helps manage *Kubernetes* applications, means that the configuration of a customisable *Magda* instance can be stored and tracked as plain text, and instances with identical configuration can be replicated quickly.

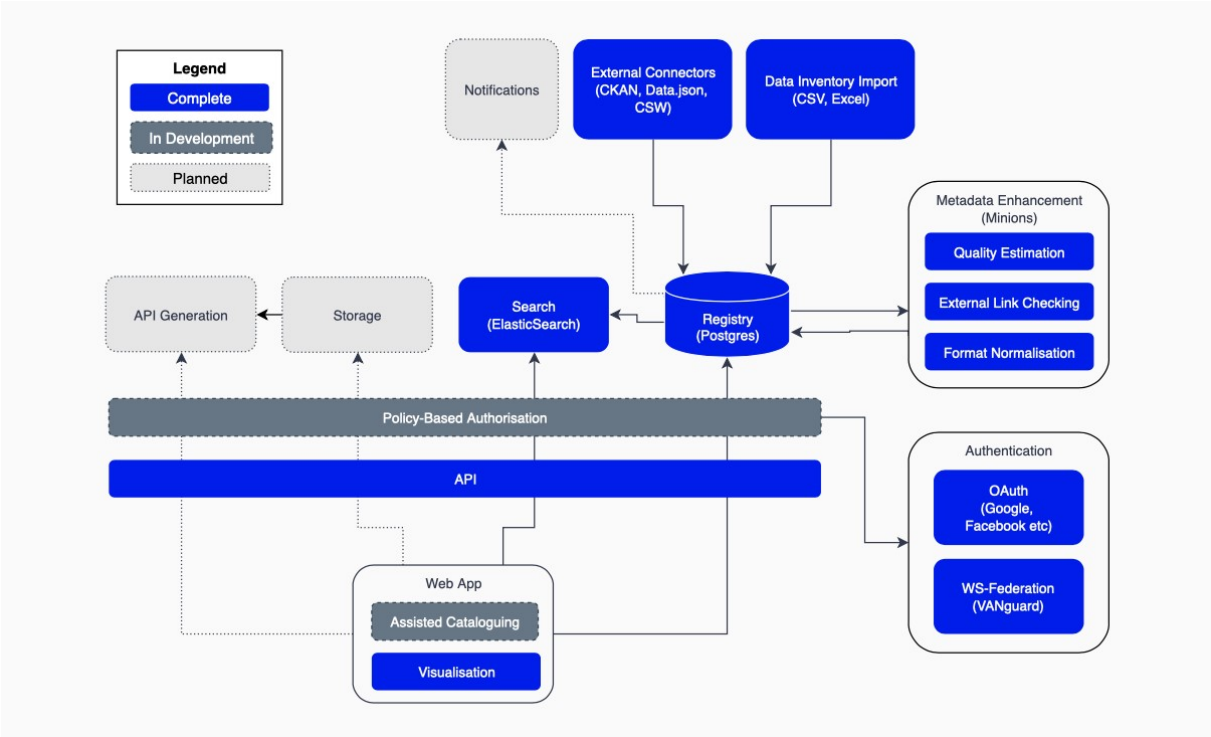


Figure 2.2: *Magda* Architecture (https://magda.io/)

Installation

Magda can be installed in two ways, with the one chosen depending on the priorities of the user or organization:

- Install from source;
- Install from *Terraform*, a tool that provides a workflow for managing cloud services as *Google Cloud*, or via another local or cloud environment.

Using *Kubernetes* and *Helm* enables simple installation and minimal system downtime. Configuration via pre-built images, i.e. from the second option in the list above, is simple in a setup as an open data search engine. However, using other features like adding datasets or the admin UI. setup is not so simple and *Kubernetes* skills are usually required.

Magda is currently under development, with their team looking to make the open source side more robust and adding new features to their implementations, which at the moment are not fully documented or require specific configurations to work.

Authorisation and authentication

At the moment a customisable authorisation system based on Open Policy Agent is being integrated, which will allow:

- Restricting datasets based on the establishment of access control frameworks (through roles, for example) or policies specified by the organization;
- Control user access to content when searching for it;
- Federated authorisation.

The latter not only allows data to be collected from external sources but also allows to mimic the same authorisations policies.

Magda's authentication system, which is based on *PassportJS*, an authentication middleware for *NodeJS*, integrates a wide range of providers, such as *Google*, *Facebook*, or even the aforementioned *CKAN*. In addition, it is also possible to develop authentication plugins to customise this process.

Registry

Magda works on the basis of a *Registry*, an opinionless datastore built on top of *PostgreSQL*, which stores records as a set of JSON documents called aspects. A dataset is represented as a record with several aspects - basic records containing the name or description, for example, or more complex ones like the quality or license of the data. Similarly, the actual data files or the URLs that link to them are also worked out as records, with their own sets of aspects.

Aspects can be declared dynamically by other services through a call with name, description, and JSON schema. This allows the user to save extra information by declaring new aspects. Also, since the system is opinionless, it is possible to create new entities in the system that connect to each other.

Metadata and *Minions*

The default metadata format used natively is the aspects system. *Magda* has an unopinionated central metadata store and therefore can provide most metadata schemas but none as standard, i.e. it supports partial standardisation.

Magda is ready to automatically enhance metadata without ever needing the data to be transmitted to a *Magda* server. The process of adding datasets for those that are cataloged directly is able to derive data from the files directly in the user's internet browser.

A *Minion* is a service that observes new records or changes to existing ones, performs some operation and writes the result back to the *Registry*. There are several aspects that are written by various *Minions*, such as the quality aspect that contains content evaluations from different sources, which are averaged out and used by search. For both internal or external dataset, the *Minions* framework is ready to act.

The framework for enhancement can be extended so that the user/organisation can build its enhancement process using any language that can be deployed as a *Docker* container.

Interoperability and *Connectors*

Magda is designed with the ability to pull data from external sources into an easily searchable catalog in which all datasets are all one entity that supports all the operations available in the others, regardless of where they come from. However, it does not use any harvesting protocol such as OAI-PMH.

For the data collection process there are so-called *Connectors*, *Docker-based* micro services invoked as jobs, which go to external data sources and copy their metadata to the *Registry*. Thus, the metadata can be searched and have other aspects attached to it.

This system can accept metadata from its own cataloging process, and there are Excel or CSV-based data inventories and metadata APIs such as those from *CKAN* or from its own REST API. Moreover, the data here is combined into a single search index with history tracking and notifications when metadata records change.

Search, preview and visualisation tools of data

When users search for something they expect the result to be the best and closest to what they want, not just the one that matches the most keywords. This system is ready to return better quality datasets over lower quality ones, understand acronyms and synonyms, and even search for geospatial or time.

Datasets and distributions (the actual data files or URLs that link to them) in the

Registry are fed into an *ElasticSearch* (which is a distributed, scalable and open data search and analysis engine for all kinds of data) cluster, which indexes some key aspects of each and exposes an API.

For previewing datasets *Magda* provides tools to verify the usefulness of their representation through graphs, automatic charting of data in tables, and spatial previewing with the open framework called *TerriaJS* for building spatial data federated web platforms.

A User Interface (UI) is provided for these tools, which is served through its own micro service and consumes APIs. Moreover, there are plans to make this UI extensible in the future.

2.2.3 *EUDAT*

EUDAT is a pan-European Collaborative Data Infrastructure (CDI) initiative (i.e. it involves all or almost all the nations of Europe), which consists of a network of nodes that provide a range of services to integrate, search, share, store, and replicate research data, more some additional services. These nodes are substantially data centers or computing centers that contain a data repository and provide a stack or part of the *EUDAT* services to manage the stored data, and these service providers can be divided into two categories: with specific research theme or generic.

EUDAT Services

Currently, *EUDAT* CDI has the following services:

- *B2ACCESS*, user authentication and authosisation;
- *B2DROP*, store, synchronise and exchange data;
- *B2FIND*, metadata harvesting and cataloging;
- *B2HANDLE*, persistent identification management;
- *B2HOST*, deploy and operate applications and data-oriented services on machines next to the data storage location;
- *B2NOTE*, for easily and intuitively create annotations on data;
- *B2SAFE*, safe data replication;

- *B2SHARE*, store, share and publish data from smaller datasets;
- *B2STAGE*, data transfer to high-performance computers.

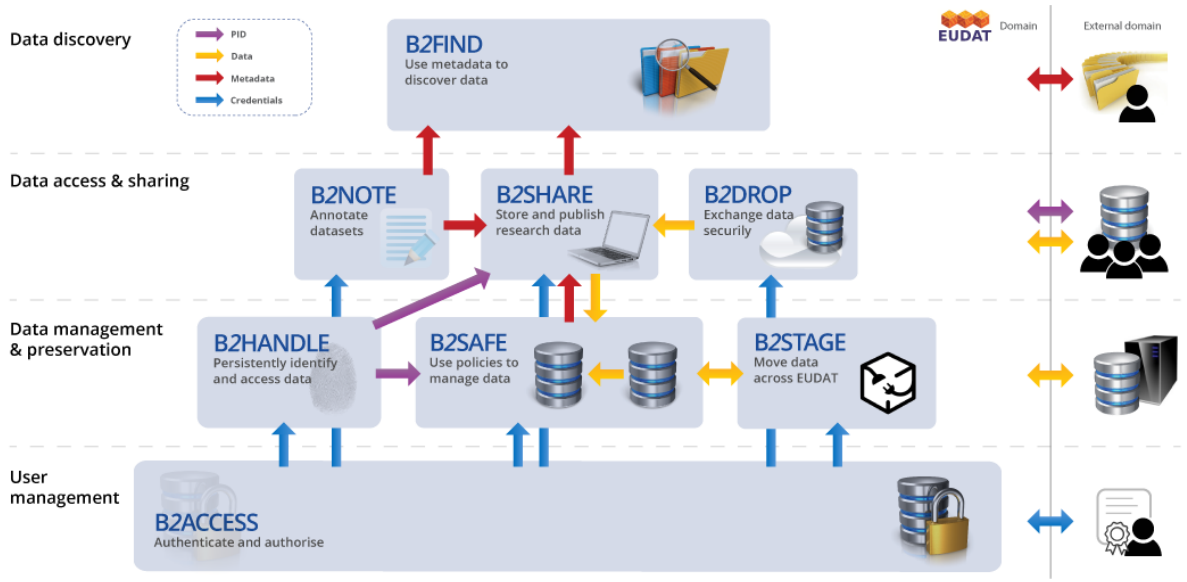


Figure 2.3: *EUDAT* services (https://eudat.eu/services/userdoc/eudat-primer)

Interaction with *EUDAT* CDI

The users of the EUDAT common services appear in two layers: researchers as end-users and experts from the scientific communities/organisations (Lecarpentier et al., 2013).

There are two ways in which scientific communities/organisations can integrate with *EUDAT*:

- Join: community becomes a data centre or computing centre part of the *EUDAT* network, which is more demanding to develop, but allows close use with all core services. This requires a minimal installation and configuration of the *EUDAT* CDI, and the provision of some resources, such as computing and storage, by the community;
- Use: an existing *EUDAT* data centre or computing centre ingests the community data and ensures its storage and also its replication, if desired. Unlike the former, due to the looser interaction, users will have little or no control over the development of core features and their configurations. *EUDAT* exposes several UIs (to aggregate

them, a Common Services Layer Interface (CSLI) is currently under development) and APIs for using its services.

End-users can also use the system in a similar way to community use, however, data replication is not possible for them.

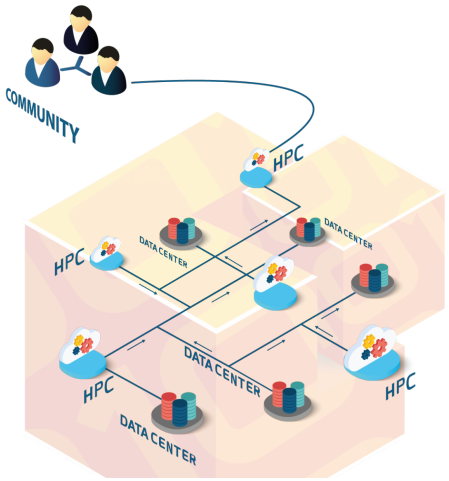


Figure 2.4: Using the *EUDAT* services (https://www.eudat.eu/eudat-cdi/using)

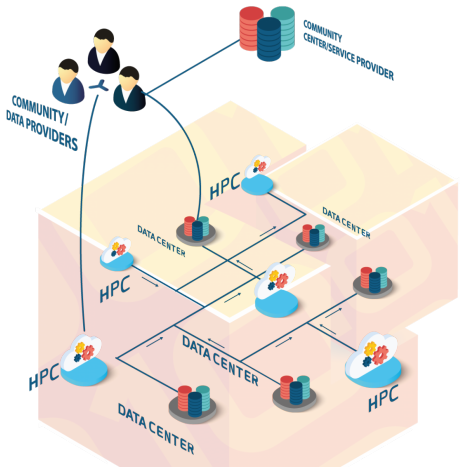


Figure 2.5: Joining the *EUDAT* services (https://eudat.eu/eudat-cdi/joining)

In both cases, the community can access *EUDAT's* primary services, which will be analysed in the next subsection. The following image clearly illustrates the distinction between the two forms of interaction.

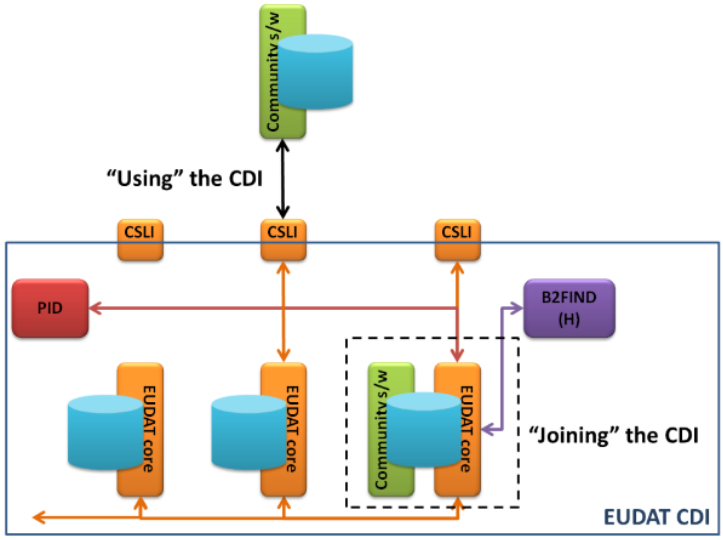


Figure 2.6: Interaction with the *EUDAT* CDI (https://eudat.eu/services/userdoc/eudat-primer)

B2ACCESS

B2ACCESS is a federated cross-infrastructure framework for user authentication and authorisation that controls their access to other services and resources.

Users can authenticate in several ways:

- User's home organisation identity provider;
- Social account, like *Google* or *Facebook*;
- *B2ACCESS* id.

The *B2ACCESS* process is based on two aspects - attributes and groups. Attributes are key-value pairs that describe basic user information, such as their name, and keep information for authorisation purposes, such as membership to communities. Groups are used to manipulate and control user's access to other services, with each service having its own set of groups.

B2DROP

B2DROP is a trustworthy service used by investigators to exchange data and thus keep data synchronised and up-to-date with each other, being based on *NextCloud*. *B2DROP's* main intention is to be used for volatile data that can be easily change and still subjective for the research and to manage this provides the version of the inserted files but does not maintain persistent identifiers for them. It can be used via a user-friendly web UI or implemented as a drive on the desktop machine via WebDav.

B2FIND

Metadata is a key part of data management, as they describe the dataset. Systematically structured and searchable metadata enables efficient discovery of scientific data, and this is how *B2FIND* service works, using a comprehensive joint metadata catalog and a search portal respectively, aided by *CKAN* functionality and API.

This service can be used through the discovery portal via the web, an API, or an intuitive command-line tool, and encompasses advanced functions like presenting datasets through tables or filtering by location using geospatial features. To search by metadata there are three options, which can even be used together:

- Free text search, without any restriction;
- Faceted search, through specific fields or properties;
- By location or time.

B2FIND aggregates diverse metadata information from heterogeneous sources within *EUDAT*. Regarding protocols for metadata, the standard is OAI-PMH, but other formats are also possible.

The metadata schema is based on *DataCite* schema but is compatible with others, such as DublinCore. Currently it includes 26 elements with different levels of obligation: mandatory if applicable; recommended; optional. Some of these elements incorporate important particularities related to the content, such as the use of persistent identifiers of resources (the DOI being specifically intended to make citation available), or information about its licensing and consequent permission status for use by other researchers.

The use of a metadata schema and collection protocols such as these add a validation component in the process of improving metadata quality, a process in which *B2FIND* is embedded.

B2HANDLE

B2HANDLE is a distributed service that handles the management of Persistent Identifiers (PIDs) based on the *Handle* system, a trusted and scalable software infrastructure used by data curators focused on identifier registration services to create and control identifiers of their resources. The *Handle* system for referencing PIDs uses a string as a prefix and a suffix separated by a "/".

Communities that join *EUDAT* and want to provide this service can do so in two ways: either the data center itself runs *Handle* or they pass their prefix to a *EUDAT* partner to manage the control. Thus, *B2HANDLE* improves interoperability and makes operations of other services easier.

Other *EUDAT* services use this to ensure access to data from a long-term perspective, as well as its maintenance. Specifically, *B2FIND* and *B2STAGE* use *B2HANDLE* to query objects and *B2SHARE* and *B2SAFE* to create and manage the PIDs of the data objects they store.

B2HOST

Some members of *EUDAT* service providers – Tieteen Tietotekniikan Keskus Oy (CSC) , Jülich Research Centre (JUELICH), Max Planck Gesellschaft (RZG) and SURFsara Bv (SURFsara) – offer computing resources from their data centres through a service hosting framework called *B2HOST*. These resources provide communities the ability to deploy and work their data-driven applications and services on machines close to the location of the data storage, since extra computing power may be required when the volume of data is too large to be transferred or there may be licensing restrictions that do not allow data to be copied to third parties.

B2NOTE

B2NOTE is *EUDAT's* latest service and offers creation and control of annotations on online resources, which can be comments and/or pointers that provide extra information about a resource. Three types of annotations are possible: a semantic tag; a free-text keyword for when a specific semantic term is not found; a free-text comment. These are stored and can later be searched, and the results can be viewed via UI and refined before being exported or as JSON-LD (JSON for linked data).

At the moment this service is still being finalised and is not integrated into the other services.

B2SAFE

B2SAFE is *EUDAT*'s service that provides long-term data preservation through replication to multiple backup sites to ensure data redundancy.

This service is implemented as an *iRODS (Rule-Oriented Data System)* – data management software used in research worldwide module – with *Handle* REST API or the EPIC API (system with the same purpose as *Handle*) that serves for the creation and management of PIDs and also uses the *iRODS* middleware to replicate datasets from a datacenter as a source. Communities can ask another *EUDAT* site to host this service and responsibility.

Lastly, each data object, original or replica, is assigned with a PID, which even contains information connecting it to its parents if it is a replica, or to a list of direct children if it is the original.

B2SHARE

B2SHARE web-based service is a solution that ensures the long-term persistence of locally stored data and enables researchers to publish, store and share small or medium-sized datasets. This service uses other *EUDAT* services for data reliability and retention, while storage is managed by trusted nationally supported repositories.

Any user, registered or not, can search for data on the *B2SHARE* homepage. However, it is also possible to deploy *B2SHARE*, requiring a web server or cloud infrastructure as a well as a way to store the published data, and this same deployment can be done through *Docker* containers. In addition, this service is the only one that is open-source.

This service has the particularity of being integrated with *B2DROP*, which allows data to be added from it to *B2SHARE*. Besides, the existing data in *B2SHARE* can be searched through the search portal provided by *B2FIND*, which harvests the metadata present in this service.

B2STAGE

The *B2STAGE* service offers data staging, in other words, the transfer of data between nodes for processing. There are two protocols for this: GridFTP for large datasets and HTTP for small to medium files.

Big Data computing manages a high volume of data and when there are several supercomputers involved is necessary to take data transfer even more seriously. The implementation of this service is currently limited to files managed by *B2SAFE*, handling the dynamic replication of data to the HPCs. It also provides a *B2STAGE* HTTP API for dynamic access to data for integration into specific community applications.

2.2.4 Evaluation

The evaluation of the three systems analysed in the previous section is done based on the criteria also listed in the previous section, and is provided in Table 2.2 to 2.4. It should be noted that for the last table, the first field regarding popularity instead of being evaluated with a (✓) or (✗) is evaluated in order of popularity.

CKAN is already sucessful in several cases of government data platforms such as the US, corporate data platforms such as Lego, and is also used by the European Data

Portal's, whose commitment to open research data and learning is paramount, with over 35 countries involved. *EUDAT*, meanwhile, has more than 25 member, all data centers and research organizations, such as CERN (European Organization for Nuclear Research). Finally, *Magda* has as its only case of success its initial goal, which was to develop a federal government data portal for the government of Australia.

Table 2.2: Architecture comparison

Platform	Installation and deployment	Open Source	Storage Replication	Storage Location	Customisation
CKAN	Installation package or service	✓	×	Local or remote[1]	✓
Magda	From source or service	✓	×	Local or remote	✓
EUDAT	Service	×[2]	✓	Remote	×

[1] Remote only available through extensions

[2] Only one service supports this feature

Table 2.3: Interoperability & metadata comparison

Platform	APIs	Harvesting protocols	Standard schemas	Validation	Federation	DOI	Licensing
CKAN	✓	✓[1]	✓[1]	×	✓[1]	✓[1]	✓
Magda	✓	×	×[2]	×	✓	×	✓
EUDAT	✓	✓	✓	✓	✓	✓	✓

[1] Only available through extensions

[2] Only partially

Table 2.4: Usability & security comparison

Platform	Popularity	Advanced search	Geospatial tools	Authorised authentication	Curation workflow
CKAN	1º	✓	✓	✓	×
Magda	3º	✓	✓	×	×
EUDAT	2º	✓	✓	✓	✓

After the analysis and evaluation, *CKAN* is the one that stands out the most. Although it has more cases of proven success in the use made by government institutions for processing their data, the fact that it is an open-source system, has a wide range of features, powerful APIs, and the possibility of customisation and extensions of functionalities make it possible to use it as a SDMS (Carvalho Amorim et al., 2016). In addition, it gives the option (even incentive) to organisations to have full control over the storage of their data, which can be a key decision for many scientific institutions.

An interesting fact is also the use of *CKAN* by the other two systems addressed, *EUDAT* in the use of the search portal and *Magda* in metadata aggregation, which demonstrates its value as a solution for data management.

There would have been other criteria that could have been taken into account, but this dissertation has focused on those we considered to be the most important in the current choice of a system that is as efficient for large institutions as for smaller ones, whose community support in the development of the platform is somewhat advantageous.

Apart from the evaluated systems, i.e. *CKAN*, *Magda* and *EUDAT*, we enunciate other solutions that would be interesting to analyse, such as:

- *DSpace*, which focuses on creating open source repositories for scholarly and published digital content and shows a few similarities to what *CKAN* offers. This software package was developed by MIT and HP Labs and has notorious examples of its use such as Apollo (Cambridge University repository) or the Open Knowledge Repository (World Bank repository);

- *Invenio*, an open source repository software offering tools for managing digital content in institutional repositories and research data management systems (RDMSs). It was initially developed by CERN, but is also used by other institutions such as the

SLAC National Accelerator Laboratory, a U.S. Department of Energy laboratory operated by Stanford University;

- *Dataverse*, an open source web application similar to *CKAN*, but with a greater focus on research data. This project was developed by an institute at Harvard University and of particular note is its use as national solutions for university repositories, proven, for example, in the Netherlands and currently being developed in Norway.

Finally, before we proceed to the development of a *CKAN* based system and its consequent qualitative and quantitative evaluation, we will qualitatively evaluate a *EUDAT* system adapted to our case study. As we have seen in this chapter, the potential of *EUDAT* in the future, if made available for development, could be even more powerful than the *CKAN* system, so it is in our interest to understand how we could set it up.

EUDAT

We now address how our system, which will be developed through *CKAN*, could have been developed in *EUDAT*. It would also be interesting for us to make a quantitative evaluation, but after contacting the *EUDAT* team, they were inflexible in providing unlimited access to their services for local development purposes, since their integration policy is somewhat rigid. Despite this, they have some workloads for some of their services to be developed locally, but mostly without any support or with the need to get in touch with communities of other software.

Through the experience we have gained through the use of *EUDAT* services in production mode, and through some local reproductions of some of its services, we have been able to understand how we can take advantage of the system considering our case study. We therefore examined which services would be interesting to implement:

1. *B2ACCESS*: in order to take advantage of most *EUDAT* services, it is necessary to register on the platform responsible for simple and secure authentication and authorisation. There are several possibilities for authentication, however, we can take advantage of INESC TEC's existence as an organisation already inserted in the system and authenticate ourselves by providing the identity of the user's organisation of origin. Therefore, it is already possible to access the remaining services without any restriction;

2. *B2DROP*: after accessing *EUDAT*, the first service to use would be *B2DROP*, which is used to store and exchange data between colleagues in the research team. In this case, the data to be stored in this service would be GNSS and NMEA, as these formats do not allow their usability and would need to be processed. This platform allows the insertion of files with a maximum of 2 GB and each user has a maximum quota of 20 GB, which we believe would fit the amount of information resulting from the SAIL Project;

3. *B2SHARE*: Once we have considered the *B2DROP*, it makes sense to understand where we could store and publish the already processed data, this is where *B2SHARE* comes in, which serves that purpose. In addition, it guarantees the long-term preservation of that same data (through a PID associated to the dataset), which is ideal for those we already consider as final such as the CSV files of the measurements of atmospheric conditions. Thus, while *B2DROP* does not allow metadata, *B2SHARE* already does, and so we could enjoy the use of time and location as some of the fields. When it comes to values, *B2SHARE* allows to upload files up to 10GB and records up to 20GB;

4. *B2HANDLE*: the acquisition and management of the PIDs made by this service can be entrusted to the *B2HANDLE* team, it is only necessary to acquire a production prefix, which is requested to the team through a form;

5. *B2FIND*: Although *B2SHARE* shares data and we can search through its platform, this is limited, since its purpose is not that. However, this component is important in the research project of SAIL and, then, the use of *B2FIND* arises, which harvests metadata from different sources, including *B2SHARE*, and includes a data portal (based on *CKAN*) that allows searching them. To publish the metadata in *B2FIND* it is not enough to have them in *B2SHARE*, we must take into account the protocol we offer to provide the metadata, which is preferably the OAI-PMH protocol, and their schema. Regarding the first point, the OAI-PMH protocol has already been taken into account when preparing the SAIL data management plan and, regarding the second point, one of the schemas considered in that same plan is Dublin Core, which is one of the formats recognised by *EUDAT*. As such, no change would be necessary and it would be sufficient to have the metadata present in *B2SHARE*.

After evaluating these services, we summarize the reasons why we would not consider the rest in the integration of a system adapted to *EUDAT*:

1. *B2NOTE*: the possibility to have annotations to resources is something interesting and that we could contemplate in the system, but it is not yet integrated in *EUDAT* services and is therefore unavailable;

2. *B2SAFE*: this service emerges as a solution for communities without archiving facilities or without considerable computing resources, which basically need to replicate their data in different data stores. This is not the case in our situation, because an infrastructure like MACC would have the resources to take control of the system;

3. *B2STAGE*: required to transfer data in an HPC environment, between different data stores, which would be useful when using the *B2SAFE* service.

Chapter 3

Research data repository

This chapter covers the solution developed for the problem presented in this thesis, from how it is implemented, addressing the particularities of the case study and its data, to the obstacles encountered and the improvements made to the *CKAN* base system.

After the definition of *CKAN* as the system to be adopted, the investigation of the functionalities to be considered for the SAIL Project had a new contribution from its team, who informed us about the key elements to be considered in the elaboration of the system, taking into account the nature and usability of the data:

1. Understand how best to organize the data;
2. View data geographically;
3. Filter the data geographically;
4. Restrict content according to user type;
5. Seek to integrate *EUDAT*, more specifically the *B2FIND* service.

3.1 Deployment

First of all, we describe the requirements for deploying the software on a new machine, as well as a step-by-step approach to try it out.

For the installation of the system it is necessary to meet certain prerequisites, which are listed below:

- Unrestricted access to external domains, such as *GitHub* or *DockerHub*;
- Have docker, docker-compose and git installed;

Once the prerequisites listed above are met, we need to download the software as well as some custom images to run certain docker containers. For this, we need to run the following commands:

- Clone *CKAN* into a directory of choice:

```
cd /path/to/directory
git clone https://github.com/priest110/ckan.git
```

- Pull the required docker images from the *DockerHub*:

```
docker pull priest110/docker_db:1.0.0
docker pull priest110/docker_ckan:1.0.0
```

Reaching this point, we have a working system, however, the *Docker* installation has a problem regarding the recognition of localhost as a valid address (where the main service *CKAN* is running) for other services, in this case for the *Datapusher*, which makes it impossible to run well. As such, some changes need to be made to the hosts file:

1. First, find the IP 0f the docker0 bridge:

```
ip addr show | grep docker0
```

2. An example of the possible output:

```
10: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state
DOWN group default
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0:
```

3. Next, we can then edit /etc/hosts/ file:

```
172.17.0.1 dockerhost
```

4. Restart the *Docker* daemon:

```
sudo service docker restart
```

5. Run:

```
docker-compose -f docker-compose.yml -f docker-compose.override.yml up -d
```

After this, we have all the services working well, just navigate to http://dockerhost:5000 and explore.

3.2 Data

The organisation of the data was an important aspect to consider when developing the solution, taking into account the structure and nomenclature defined in Subsection 2.1.2.

To take advantage of the usability, before inserting the datasets we tried to match the features of *CKAN* to what we wanted from the data. As such, we first took the preprocessed data sample and converted it to a CSV format, since *CKAN* tools allow geospatial availability and interaction with it from structured data, and provide other formats that are also interesting for us to analyse the data, such as table and graph. Thus, we are able to test the usability of the data with geospatial fields, with the help of extensions, which we will talk about later. With regard to the raw GNSS and NMEA data, it would also be possible for us to convert it to CSV and make it viewable, however, it is purely data associated with location and time, whereas the structure data associates location with other data relating to atmospheric conditions, so we have chosen to keep the GNSS and NMEA data in the original format.

We have organised the information into 3 different types of datasets, whose titles are as follows:

- Structured atmospheric measurements;
- Raw GNSS measurements (YYYY/MM/DD);
- Raw NMEA measurements (YYYY/MM/DD).

For each type of dataset, we have several datasets encompassing daily information regarding GNSS and NMEA measurements, and one dataset for the structured data since our preprocessed sample has only 6 files, each one on a different atmospheric condition. However, it would make sense to adopt the same behaviour for the structured data with a sample of the same size as the other measurements, and consider a different daily dataset for each atmospheric condition.

At this point, we can already see that we will have dozens and dozens of datasets, since they are daily datasets. As such, we chose to insert the date in the title so that not all datasets have the same name and it also allows us to find any dataset through a search in *CKAN's* search engine, for example, if we search for "title:23", all datasets that have 23 in the title will appear, including the datasets that have the day 23. However, if we choose to search for "title:10" will already appear datasets with month equal to 10 and/or day equal to 10, therefore, we chose to add 3 key-value pairs to the metadata, being the keys "year", "month" and "day", so if we search for "day:10 month:01" will effectively only appear the datasets associated with day 10 of month 01.

Finally, besides the datasets and respective files, we also included in the system 2 organisations – INESC TEC and High-Assurance Software Laboratory (HASLab), which is one of INESC TEC's integrated R&D centres – and 1 group – SAIL Project. While it is necessary in *CKAN* to have an organisation that owns and manages the information, in this case study, INESC TEC, there is no need for groups, however, they can be used in order to catalogue a team, theme or project, in this case SAIL Project.

3.3 User Interface (UI)

We have chosen to make some changes in the User Interface available in the front-end in order to take advantage of the features provided by *CKAN*. In what concerns the pages alluding to datasets and resources, as well as those of groups and organisations, we consider that the information is already provided in an intuitive way and, in our opinion,

does not need any changes. Nevertheless, we consider that the homepage could suffer some changes, and, as such, we opted to work on it and obtained the result shown in the figure 3.1.

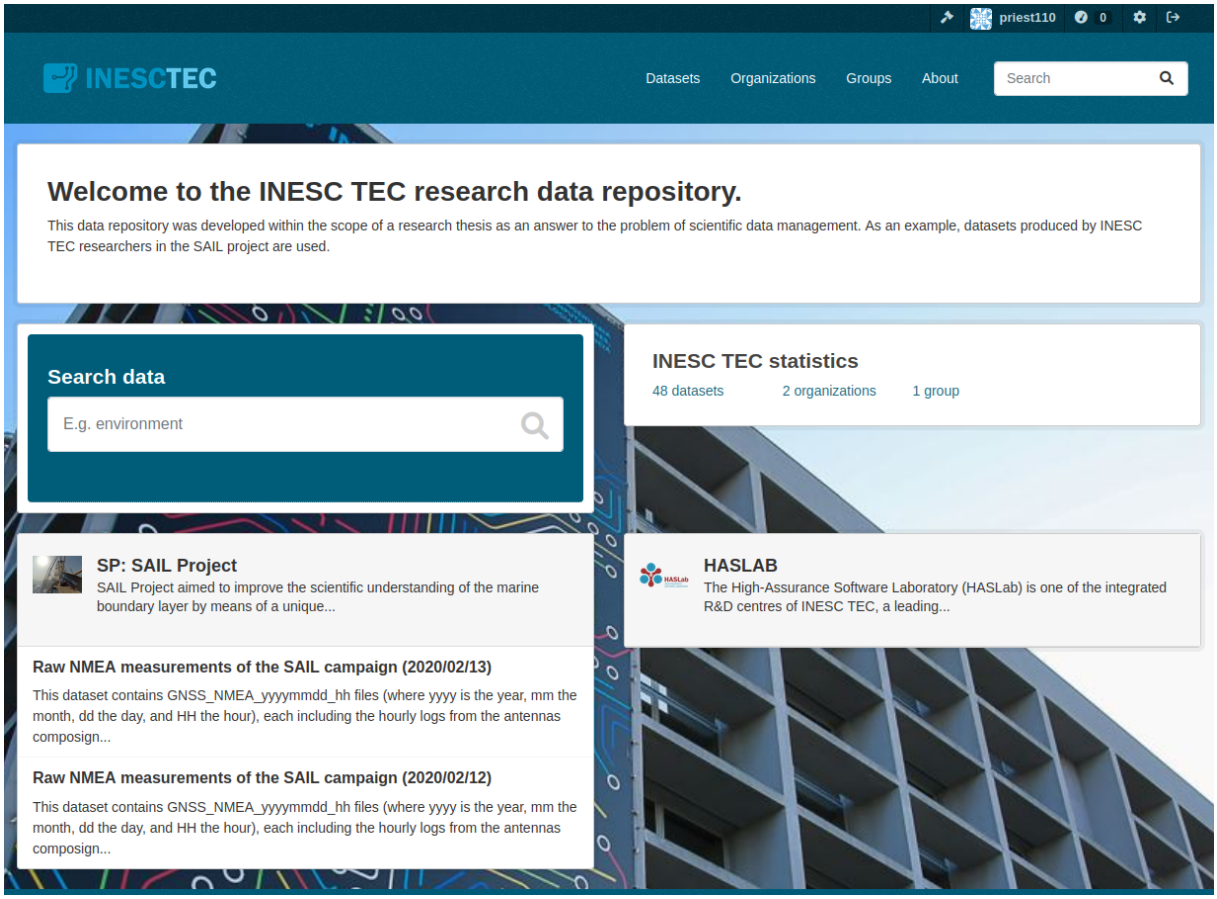


Figure 3.1: Homepage

Thus, we have improved the initial aesthetics of the main page (which ends up being the first thing that users will see in the system) and we have customised it to fit in with the institution that uses its services. Furthermore, as the primary reason for the change, we have been able to transmit more useful information, as previously only the welcome statement, the search engine, and the statistics were considered, and now it is also possible to access two additional components:

- Latest group in the system, in this case associated with SAIL Project, and its two latest resources;
- The most recent organization in the system, which is HASLab, and which in turn does not yet have any associated resources.

3.4 Extensions

One of the great advantages of *CKAN* is effectively its customisation and the existence of numerous solutions developed for use as extensions to its base system. As such, we examined which extensions could be interesting to explore in our case study, as well as the outcomes and consequences of their implementations.

ckanext-spatial

This extension, along with ckanext-geoview, are undoubtedly the ones that had the greatest impact meeting the requirements of the SAIL Project, because they allowed answers to two of the five questions posed at the start of the chapter. More specifically, the geospatial capabilities that this extension added to the system include:

- A new field ("spatial") in the dataset metadata schema, which allows this same field to be available through a map on the front-end of the dataset's respective page, and also geospatial queries, such as filtering by geographic area;
- Harvesters to import geospatial metadata from other sources into *CKAN*.

As for the same dataset we have several files with different latitudes and longitudes, we have chosen to collect the information regarding the maximum and minimum values of each measure. Thus, we obtain an area relative to the investigation of each day, this area being the value of the "spatial" key, which after adding to the metadata is ready to be viewed.

ckanext-geoview

This extension contains view plugins to display geospatial files and services in *CKAN*. Thus, it made it possible for all resources with structured data with latitude and longitude fields to be viewed through a geographical map, and also provided filtering capabilities, for example, in the number of entries to be viewed in a given file, and interaction, for example, in the ability through the map to access a particular climatic condition associated with a geographical point.

ckanext-harvest

Although this is not something essential to consider for a project like the one we are addressing, we believe that this extension will add quality to any system that manages data, as it allows us to generate datasets from other sources.

For instance, considering that some research team outside INESC TEC would like to access some of the data provided by the SAIL project in order to explore it further or in a different way, new information could emerge that INESC TEC would consider interesting to make available on its platform. For that, it would be enough to use a harvester to fetch the remote dataset from where the new data would be into our system.

After adding the harvester, we can define if we want to run it weekly, daily, annually, and each time we do it the following happens:

- New datasets will be added locally, if any;
- Updates that are done on the remote datasets, will also be done locally;
- Datasets that no longer exist remotely will be deleted locally.

ckanext-doi

We have added this extension to assign a DOI when a dataset is created and made public. This requires an account at *DataCite*, since it is the service that handles the assignment. Note that the DOI is in the format:

```
https://doi.org/[prefix]/[8 random alphanumeric characters]
```

Again, for this project this extension would not have the best of uses, as it would assign different DOI's to each dataset, when they are all part of the same investigation. As such, it would be useful to have a DOI, but in this case this would be the same for all datasets and would have to be assigned outside the system (whereby we could use the existing https://doi.org/10.5281/zenodo.5797919 referring to the SAIL Project). However, this extension would be extremely useful in most cases.

ckanext-dataspatial

This extension is no longer maintained and unfortunately no longer works for the version of *CKAN* we are using, however, like others mentioned, while interesting, it is not core.

It would provide geospatial awareness of *DataStore* data, which includes geospatial searches within datasets and the spatial extension of *DataStore* searches. However, the first of the two capabilities can be achieved through ckanext-geoview.

ckanext-hierarchy

Often there are relationships between organizations, like the one we contemplate in our system, for example, since HASLab is an organization that belongs to INESC TEC. However, in the default *CKAN* system there is no way to make explicit the relationship between two or more organizations, so we chose to use this extension, which allows us to:

- Create relationships between organisations;
- Hierarchize the relationships;
- For each organisation, on its datasets page, in addition to having available the ones it owns, we can choose to also provide the datasets of the organisations it owns.

ckanext-restricted

Through this extension we were able to answer one more of the points listed at the beginning of the chapter, since it provides the possibility of restricting access to resources in a dataset, while still being able to view the metadata but not the data itself. However, this extension is currently out of date, which does not allow proper use.

The access levels that we can assign to the resources in a dataset are as follows:

- Public;
- Registered users;
- Members of an organization;
- Members of the organization that owns the dataset.

ckanext-oaipmh

Aside from the sources provided by the ckanext-harvest extension, this extends the *CKAN* harvester to parse OAI-PMH metadata sources as well as import datasets. It supports metadata schemas such as oai_dc (Dublin Core), used in the SAIL project.

This extension is outdated and, since it is not supported by the *CKAN* team (it was developed by the community), it should not be updated again for the new *CKAN* version. As such, there are some bugs that do not allow its proper use.

ckanext-b2find

The *B2FIND* service, already discussed in Section 2.2.3 provides a user-friendly data discovery portal to search for research information stored in various repositories.

This extension allows the use of the *B2FIND* service on *CKAN*, which in essence ends up allowing a different approach to metadata aggregation, since the search portal is based on *CKAN*.

Like others already mentioned, this extension is outdated, but it is currently in transition progress, and we estimate that in a few months it will be possible to enjoy its functionalities.

ckanext-ord-hierarchy

Finally, this last extension is similar to ckanext-hierarchy, however, it is used to create and provide relationships between datasets through a hierarchy. Looking at the way we have structured the datasets, we believe that there are no ownership relations between them, however, it is another extension that brings quality to the system and could easily be considered for other cases.

3.5 Discussion

There were several obstacles encountered during the development of the solution, and some have already been addressed throughout the chapter, but we opted to dedicate this section to explore in more detail those in which we invested more time or which had more consequences on the final result.

The first challenge occurred in installing *CKAN*, since, on the one hand, the latest stable version of it installed via *Docker* uses *Python* 2, which is no longer supported, while, on the other hand, the master version is in transition to *CKAN* 2.10. We ended up choosing the second option because, after contacting the *CKAN* team, they informed us that they believed the master version was stable enough, and, in addition, had the advantage of working with *Python* 3.

As far as extensions are concerned, there were several that did not work, since they are outdated (ckanext-restricted, ckanext-dataspatial, ckankext-ord-hierarchy, ckanext-oaipmh), or have limited functionality (ckanext-b2find), and therefore could not be used. However, although they added quality to the software they were not central to what we wanted to develop, and we believe that the most important extensions are present in the system.

Finally, the biggest obstacle was not being able to take advantage of the partnership with MACC, for reasons unrelated to the research. For this reason, it was not possible to recreate the best scenario regarding an HPC environment. As such, we resorted to using machines from INESC TEC to evaluate the performance of the developed system, which did not allow us to recreate an effective approximation to the real world, as would happen with the MACC resources.

Chapter 4

Performance Evaluation

In order to understand how the system behaves in a real-world context, especially to extrapolate its behavior on a large-scale system with thousands of requests per second, we need to understand how the system works in a much smaller, controlled world. To this end, we performed tests to obtain the server execution profile and, in this section, we will explain the test conditions, the methodology used, and the results we obtained from our adapted *CKAN* system.

4.1 Test conditions and Methodology

To test the system, we used one of the machines from the INESC TEC, specifically described in Table 4.1.

CPU	Intel® Xeon® E5-2698 v4
#Sockets	2
#Cores per socket	20
#Threads per core	2
CPU Frequency	2.2 GHz (base), 3.60 GHz (max.)
Memory	32 GB DDR4-2400

Table 4.1: Description of the machine used

We used the *Locust* (loc) tool and the iostat command for testing. On the one hand, *Locust* is a load testing tool developed in *Python* that allows us to run tests in a distributed way across multiple machines and simulate thousands of users making requests

concurrently. It produces two metrics: throughput, i.e. the number of requests the server can handle in a unit of time, and the response time, i.e. the time the server takes from the time it receives a request until it answers it (these two metrics seem to be the inverse of each other, but they are not, since throughput is very much limited by the number of requests we can launch per unit of time, while response time will be limited by the server's processing and resource management capacity). On the other hand, the iostat command is used for monitoring system input/output devices by observing the time they are active in relation to their respective average transfer rates, and allowed us to extract information regarding two other metrics: CPU and Disk utilisation.

The tests were done using a variable number of clients, specifically from 0 to 1000, and the increment was divided into 3 stages:

- 0 to 100 concurrent users with a ramp up speed of 2 users/second;
- 100 to 500 concurrent users with a ramp up speed of 4 users/second;
- 500 to 1000 concurrent users with a ramp up speed of 6 users/second.

The interval from 0 to 1000 was chosen since from 1000 users on, and even before that, in some cases, there were already system faults (and not failures) due to the considerable weight of certain requests, which had the usual consequence of exceeding the timeout that we consider to be adequate in the response to a read request made to the server – 5 seconds. Nevertheless, this interval proved to be more than enough to understand how the system responds, taking into account the hardware and the different types of requests used (shown in Table 4.2), where not all possible ones were contemplated, we neglected those that we considered to have no real relevance to reliably measure the performance of the system on a large scale. When it comes to the choice of using different stages, the goal was to understand the system's ability to adapt to different quantities and growths of users.

Type	URL (with prefix /api/3/action)	Description	Probability
GET	/current_package_list_with_resources	Return a list of the site's datasets and their resources	5
GET	/package_show?id	Return the metadata of a dataset and its resources	5
GET	/resource_show?id	Return the metadata of a resource	4
GET	/group_list	Return a list of the names of the site's groups	3
GET	/organization_list	Return a list of the names of the site's organisations	3
POST	/package_create	Creates a new dataset	1
POST	/package_revise	Revise a dataset selectively with match, filter and update parameters	
POST	/resource_create_default_resource_views	Creates the default views (if necessary) on the provided resource	
POST	/datastore_create	Adds a new table to the *DataStore*	

Table 4.2: Types of requests

All the core functionality of the site provided by *CKAN* can be used through external code that invokes the API, which will be extremely useful in testing. As such, the requests listed in the table above make use of the strong API that *CKAN* provides, which unlike requests made through the web UI, do not need to load any views or templates. Also, for testing purposes, POST requests can only be recreated through the API, so in order to get

a fair comparison between requests, we chose to use only the API to evaluate performance.

Until we got to a realistic case, developing the tests required a cycle of optimizing/measuring the results in order to build a real test of an efficient server. As such, we divided the evaluation moments into three distinct ones:

1. Individual case, to understand how the system worked with just one individual type of request;

2. Preliminary case, where we worked only with read-only requests, since in our view they are the ones that exist more frequently in the system and allowed us to get a sense of how it works while analysing other influencing factors based on the server and test tool settings:

   (a) waiting time between requests

   (b) load distribution

   (c) connection pool

3. Realistic case with read and write requests, each request with a specific probability of occurring, since there are requests that are more frequent than others.

The instrumentation methodology, is as follows: we run a script with the *Locust* that creates users and each one executes tasks, one at a time, until a voluntary stop is made, more specifically, about 10s after we get the 1000 users (each test took, on average, 4 minutes and 30 seconds). Each read request is included in a different task, while the four write requests are included in a single task, since for each insertion of new non empty dataset, we need one to create the dataset and another to add a certain resource, and besides different requests for each insertion, it is necessary to create default views for the resource, as well as add it to the *DataStore* (the latter two allow the resources to be viewed and exploitable). It should be noted that each test was repeated several times in order to understand if the values presented were valid or suffered from some extraneous anomaly.

4.2 Individual case

Before we understand how the requests work together, it makes sense to understand their individual behavior, i.e., to consider the case where we make only a certain request, since certain decisions in later testing may be influenced by the behavior we observe in these first tests. For the POST requests, since we encompassed them all in one task, we consider all 4 as one type, and due to the heavy computation (4 different requests, as well as the random generation of the dataset and file to be uploaded to the platform), we chose to run only up to 100 users, which will serve to fully understand how they work.

4.2.1 Throughput

The first metric we evaluated was the throughput of requests that the server can answer, per second. This metric provides a notion of the system's behavior for a varied amount of clients, which will give us an idea of what would be the ideal number of simultaneous users corresponding to the highest number of answered requests per second.

In Figures 4.1, 4.2 and 4.3 we present the data collected in the tests with regard to throughput, and we chose to split the information between 3 different graphs (requests to read datasets and resources, requests to read organizations and groups, and write requests), since everything in the same graph would greatly reduce the perception of the results.

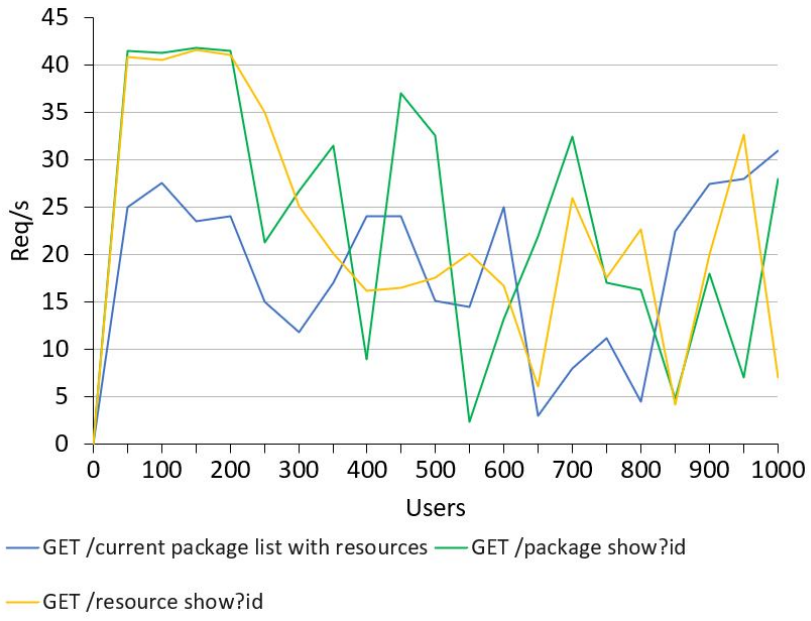


Figure 4.1: Throughput for dataset and resource requests

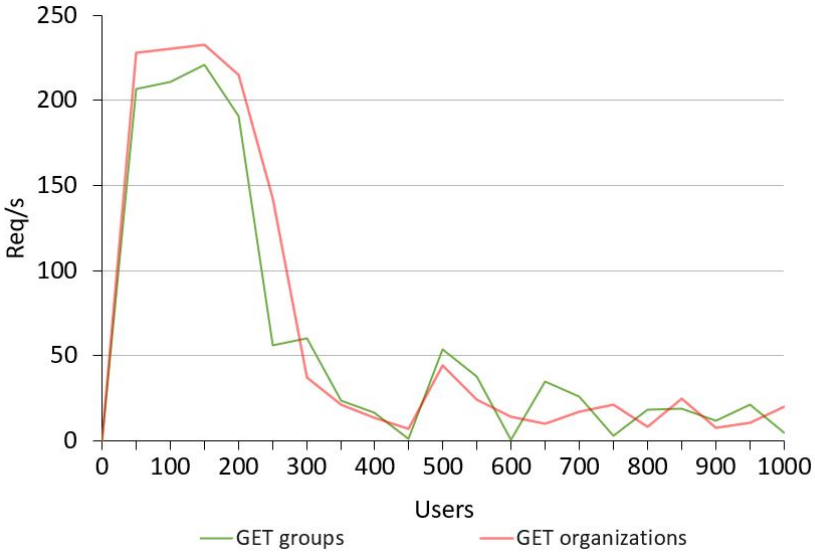


Figure 4.2: Throughput for group and organization requests

From the graphs in Figure 4.1 and 4.2, we can draw some conclusions:

- The requests per second increase up to 50 concurrent users, and remain without significant changes up to 200 users, which for now shows to be an interesting data to identify the limit of clients per server from which we will have finished the diminishing returns, and which would be the ideal number of clients to establish as a limit of clients per server. Furthermore, we can notice that the throughput of the requests to the set of groups and/or organisations is much higher, which lies in the fact that this data is much lighter;

- After the 200 user mark, there is a significant decrease in the system's ability to handle that many users, with the system's lower ability to manage the number of users suggesting a greater range in requests made per second, getting several inconsistent peaks. This happens most relevantly for requests to read datasets and resources, since the proportion of requests before and after the 200 users is not as noticeable as for organisations or groups;

- It would make sense for the throughput to be higher for reading a given resource compared to a given dataset, since its average response size is smaller than the latter. However, *CKAN* forces the resource request to an extra computation, i.e., it accesses the dataset it belongs to and checks if it is in the dataset's resource list, if the resource is no longer present, then it is non-existent.

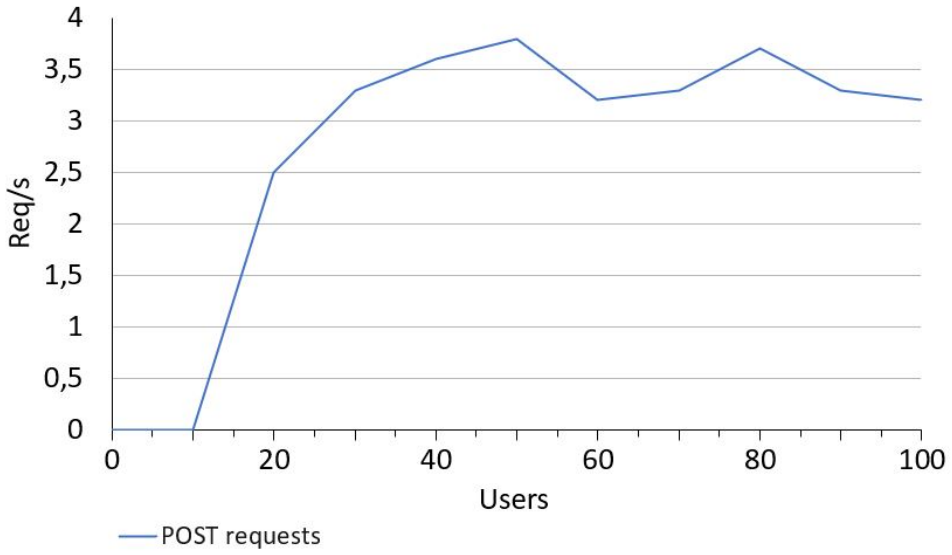


Figure 4.3: Throughput for write requests

The graph in Figure 4.3 shows the huge demand for write requests, being the most relevant bottleneck in this set of requests. Note that the system does not allow to inserting more than one file (resource) at a time, which makes this process even more complex.

4.2.2 Response time

The second metric we sought to analyse was the time it takes the client to receive a response to a given request, distinguishing between each of the types and understanding what factors can influence this response time. As such, we present information in Table 4.3.

We chose to analyse only the average time, since, for now, the average time allows us to draw the necessary conclusions for the elaboration of the tests in the following sections.

Request	# Requests	Average (ms)	Average size (bytes)	RPS
GET /current_package_list_with_resources	5178	12704	216203	19.5
GET /package_show?id	7309	9891	10603	26.8
GET /resource_show?id	7056	10459	772	25.0
GET /organization_list	24350	3324	129	92.5
GET /group_list	22222	3190	116	80.6
POST requests	179	9496	2178	3.5

Table 4.3: Individual Response Time

Analysing Table 4.3 we can draw three conclusions:

- The average response time of the server to a given request depends mainly on the size of the response, that is, the smaller the size of the content we want to read, the faster the response will be (even suggesting that we are facing a logarithmic growth) and, consequently, the higher the throughput will also be;
- Although for the case of POST requests we consider a smaller sample, it manages to present an average response time almost equal to the reading of datasets with random id's in a sample with 10 times more users. This reinforces the impact that this type of request will have in a real scenario;
- The number of requests to the list of organisations is slightly higher than for groups, despite having a larger average size as a response, and we believe this is due to extra calculation that the latter performs, however, we cannot see where this actually happens.

4.2.3 Summary

Finally, after these first tests, we can make an overall analysis and make some decisions for those that will follow:

- All clients are launched with the same process (running on a single core) by *Locust*, which can end up being limiting when it comes to exploiting the maximum capacity

of the available resources, and therefore also when it comes to getting a realistic sense of how the system works. Furthermore, all clients share the network bandwidth, which also limits the throughputs that we collect in these first attempts. However, even though we cannot do anything about the bandwidth sharing, when it comes to using only one core, we will try to exploit the load distribution feature *Locust* has;

- Requests that require a random ID of a resource or dataset are time expensive compared to those that use a specific ID, and obviously in a real world scenario this would not happen, but for testing purposes the extra computation generating a random ID is needed. The system may have numerous resources and datasets and each with completely different sizes, so going the other way, such as a default ID, would be misleading;

- In a realistic world scenario a user will not be making requests all the time with no break in between, so we will explore the wait between requests;

- *CKAN* uses pagination when making datasets available to the user through the UI, as such it only loads x number of datasets (by default 20) which enables the insertion of a larger volume of data. We were able to reproduce the same pagination behavior in a similar way, and although we considered increasing the number of datasets in the system, we chose not to do so because as we can see from these early graphs, there is already a considerable amplitude difference between requests early on. However, due to paging, we believe that we could increase the number of data, with slight consequences up to 200 users and bigger problems thereafter.

4.3 Preliminary case

In this section, we will analyse the behavior of the system with the variety of all read requests and expose the system to the factors listed in the Section 4.1, i.e., waiting between requests, load distribution, and connection pool.

4.3.1 With variety

After in the previous section we understood how the system reacts if there is only one type of request being executed, in this section we will analyse its behavior if there are

several types of requests being executed, more specifically, all the reading requests listed in Table 4.2. Furthermore, we will not consider any restrictions on the way requests are made, nor any changes in the way the system handles them, since the results of the tests in this section will be used as a comparison to the remaining ones in this performance evaluation.

Throughput

It is to be expected that the evaluation of this metric will only corroborate the conclusions drawn in individual testing, since the only change from that testing is the request variety.

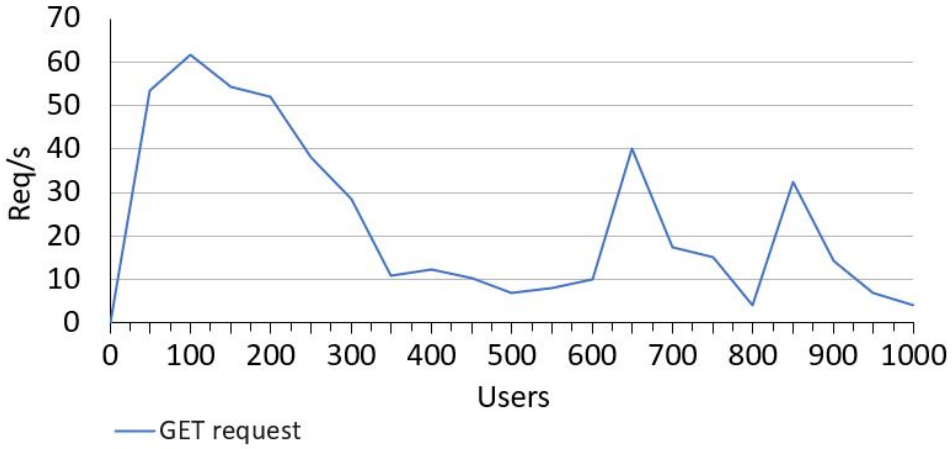


Figure 4.4: Throughput

We can then prove what we expected from the graph in Figure 4.4, that is, that the 200 users remains the point at which a significant drop in the number of requests executed per second begins to appear. Moreover, the large amplitude between requests exists from the 200 concurrent user mark onwards, with the same irregular peaks present in the individual case.

Next, we present the table 4.4 with general information regarding the number of requests executed per second, as well as other information that may be useful for making comparisons with other tests.

# Requests	# Fails	Average size (bytes)	RPS
8096	355	50200	28.8

Table 4.4: Request statistics

Of the 8096 requests made to the system, 335 fail (note that this failure is not harmful to the system), that is, 335 exceed the timeout in terms of time for the server to respond to something, which is due to the higher traffic of requests, but we will address more clearly the occasions when this happens afterwards.

Response Time

For this metric, we will look at the median response time, i.e., the time value that separates the highest and lowest half of our sample of server response times to requests made to it, and the 95% percentile, i.e., how long it takes the server to respond to requests 95 percent of the time. It is to be expected that from 200 users onwards, these values will start to increase with some significance, and that, as with throughput, there will also be a certain range between users and irregular peaks.

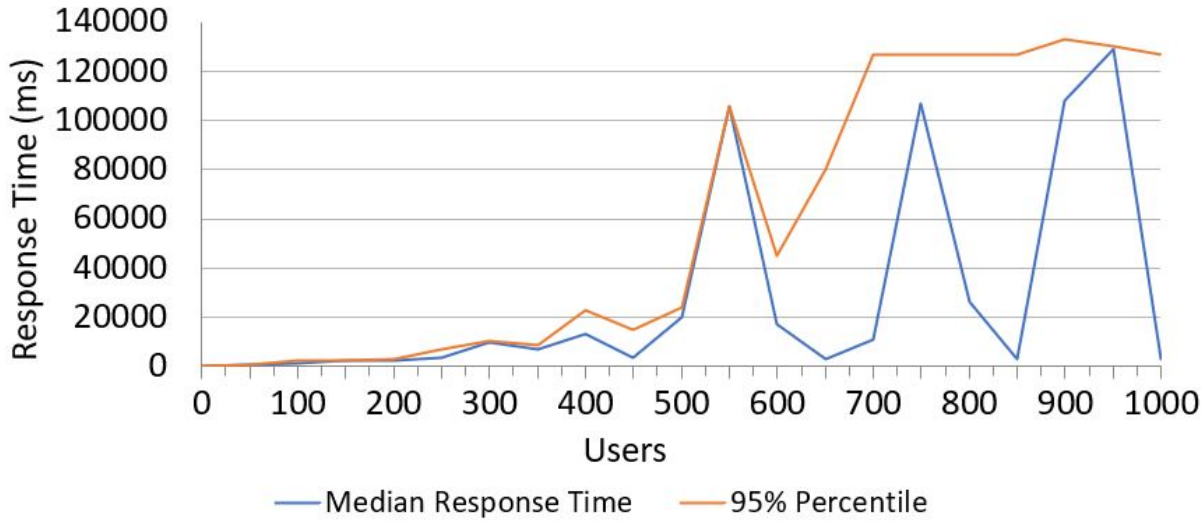


Figure 4.5: Response Time

Looking at the graph in figure 4.5, we can then confirm the expected:

- Until the 200 users mark the system responds in an acceptable way, more specifically, the server has a median of 2430 ms and 95% of the time it can respond in a maximum of 3170 ms to the requests that are made to it. However, although we take into account that most of the requests are somewhat heavy, 3s is something that we believe we can improve;
- After 200 concurrent users, the system begins to behave somewhat unpredictably, however, always in an unbearable way. It is worth mentioning the 500 users mark, since from that point on, the system starts to suffer more aggressively, since it goes from being able to respond 95% of the time in a maximum of 24000 ms to values higher than 100000s, that is, an increase of more than 400%, which results in the growth of faults in the system, since the timeout used for read requests is 5s.

To add an interesting statistic to compare later, the average response time was 9156 ms.

CPU and Disk Utilisation

For this evaluation, with the help of the Linux iostat command, we will make an analysis of the percentage of CPU usage that occurred during the execution at the user level (i.e. application) and, when it comes to disk usage, we chose to compile the information (tps - number of transfers per second to the device, kb_read/s - amount of data read from the device, kb_wrtn/s - amount of data written to the from the device) of the devices into one, these being sda, sdb and sdd, which were the ones that showed changes during the execution of the test. As far as the CPU is concerned, since we have 80 CPUs, the result is an average of the usage of each one.

We provide the above information for 100, 500 and 1000 users in the Table 4.5, i.e. the CPU and disk utilisation from the initial moment up to 100 users, then the utilisation between 100 and 500 users, and finally between 500 and 1000.

Users	%CPU	Devices		
		tps	kB_read/s	kB_wrtn/s
100	1,5	2,1	0	36,2
500	0,8	2,2	0	35,5
1000	0,6	2,3	0	27,8

Table 4.5: CPU and Disk utilisation

This way we realize that the fact that *Locust* uses only one process results in almost zero CPU usage and, in turn, Disk usage is also residual. Note that both metrics are influenced, in a minimal way, by other processes than those used by the tests or the server. Furthermore, no information is read from the devices, since it is already in the CPU cache, loaded at the beginning of the test.

4.3.2 With variety and waiting between requests

At this point, let's evaluate the system taking into account that there are waits between requests. To do this, we changed our script by adding an attribute that *Locust* provides for the User class, more specifically, wait_time, which introduces a delay after the execution of each task.

```
wait_time = constant_throughput(0.25)
```

The above assignment guarantees that each User runs a task 0.25 times per second, i.e., 1 task every 4 seconds, which seems to us a realistic estimate of system usage by a client on average.

Throughput

Something we can expect before testing this metric is that the initial growth of requests may not be as steep as for a no-wait test. As such, it should also have an influence on the speed with which the system will reach the optimal concurrency situation, more specifically, it should take longer to reach that state.

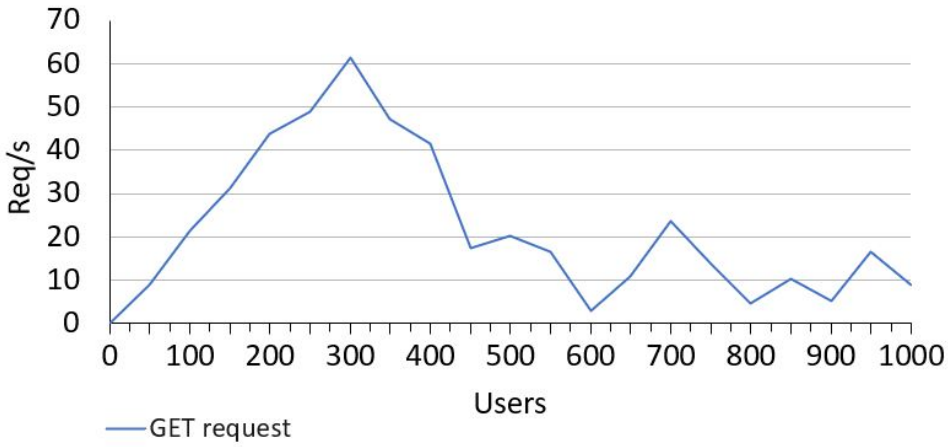


Figure 4.6: Throughput with waiting between requests

From the Figure 4.6 we can conclude the following:

- The wait between requests causes what would be expected, that is, a less gradual growth until diminishing returns are reached, which in this case we can point to end up at 400 users. This is good news, since by considering a case closer to reality, we managed to double the ideal competition limit;
- Passing the mark of the ideal number of concurrent users, in contrast to the large range between requests when there is no wait, here the range, while still existing, exists to a lesser extent.

From the same test performed for the elaboration of the graph of the Figure 4.6, we were able to extract information regarding the total values of the requests made, among other values. We present this information available in the following Table 4.6.

# Requests	**# Fails**	**Average size (bytes)**	**RPS**
5967	237	51139	21.6

Table 4.6: Request statistics with waiting between requests

We then see that fewer requests are made in total relative to a system without waits, which makes perfect sense. Without wanting to fall into redundancy, more waiting, fewer requests, however, this is not something negative, just a consequence of the reality of a system in which all users are not constantly making requests.

Response Time

Now, we try to understand how the system performs in terms of time to process and respond to a request. We again expect that, just as happened with the throughput, the response times should start to increase later, less gradually.

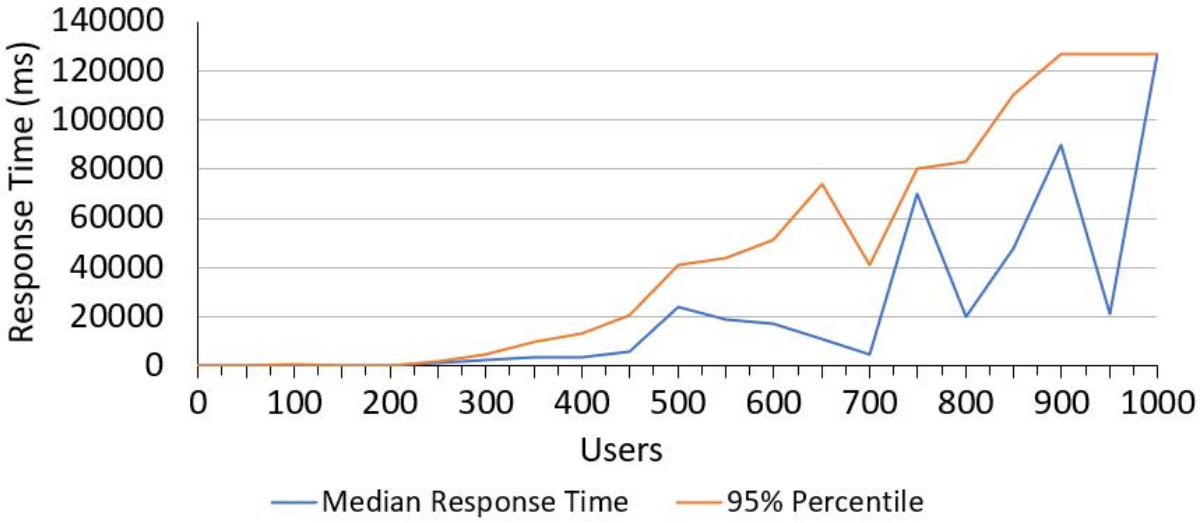


Figure 4.7: Response time with waiting between requests

Thus, with the help of Figure 4.7 we can confirm what we theorize. On the one hand, the 95% percentile and the median start showing signs of increase even before 200 users in a system without waits between requests, and the 95% percentile reaches the maximum value of 127000 ms at the 700 users mark. On the other hand, for the system we are considering in this subsection the median starts to increase considerably after 450 users, and the 95% percentile start increasing after 300 users and reaches the maximum value at the 900 users mark. From the same test, we also took the average response time, this was 10190 ms.

CPU and Disk Utilisation

We expect that, unlike the first scenario, there will be a greater CPU usage after 100 users, since until this mark the ideal number of concurrent users in the system has not yet been reached. Regarding the use of the Disk, we believe that it will again be residual.

Users	%CPU	Devices		
		tps	kB_read/s	kB_wrtn/s
100	0,5	3,2	0	34,4
500	1,1	2,5	0	33,5
1000	0,4	2,1	0	23,9

Table 4.7: CPU and Disk utilisation

We confirm the expected through Table 4.7.

4.3.3 With variety, waiting between requests and distributed load

We now consider two factors, namely, the wait between requests, since it showed to be a factor to consider in a realistic case, and, additionally, the load distribution that *Locust* allows.

While a single process running Locust can simulate a significant number when it comes to throughput, for more complex or higher load tests, scaling to multiple processes, and even multiple machines, is something to strongly consider. In our case, since we have hardware with higher capacity than usual, we will explore this feature. To do so, we use the same script as in the previous section, we just modify the way we run it, as well as the number of instances created, that is, we run one instance of Locust in one terminal using the master flag and 40 other instances (we use 40 since, typically, the Locust team suggests using one worker instance per processor core) in different terminals using the worker flag. The master instance tells the workers when to stop or start and does not run Users, while the workers run Users and send back statistics to the master.

Throughput

For this metric, we think that the distribution load will result in better overall throughput, because the process cores will be less heavy than a single would be, and for that very reason, probably the moment of diminishing returns will last longer.

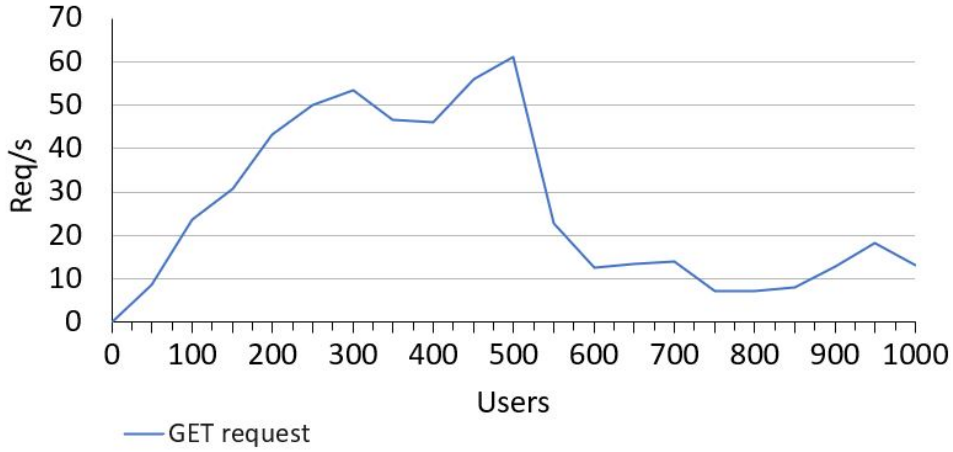


Figure 4.8: Throughput with waiting between requests and distributed load

Analysing Figure 4.8 we conclude that:

- Our predictions were true, that is, the moment of diminishing returns lasts a little longer, from 200 (as in the test of waits between requests) to 500 concurrent users, which is a record number within the tests we ran;
- Similar to the test in the previous section, the range is smaller than the initial tests due to the wait, which allows to have fewer peaks.

More general statistics regarding the response to requests are presented below in Table 4.8.

# Requests	**# Fails**	**Average size (bytes)**	**RPS**
8342	68	54369	29.4

Table 4.8: Request statistics with waiting between requests and distributed load

So we broke another record, this time for the total number of requests answered, although it is residual. On the contrary, we managed to decrease the number of failures, which is the result of an improved system capacity, and now, in the analysis of response times, we will understand why.

Response Time

In the analysis of this metric we expect to see a lower median response time, and consequently fewer peaks relative to the previous tests due to the removal of the overhead from one process, spreading it across multiples. Also, for the 95% percentile, we expect that it will not reach stabilization, since it would take more users for that moment to happen.

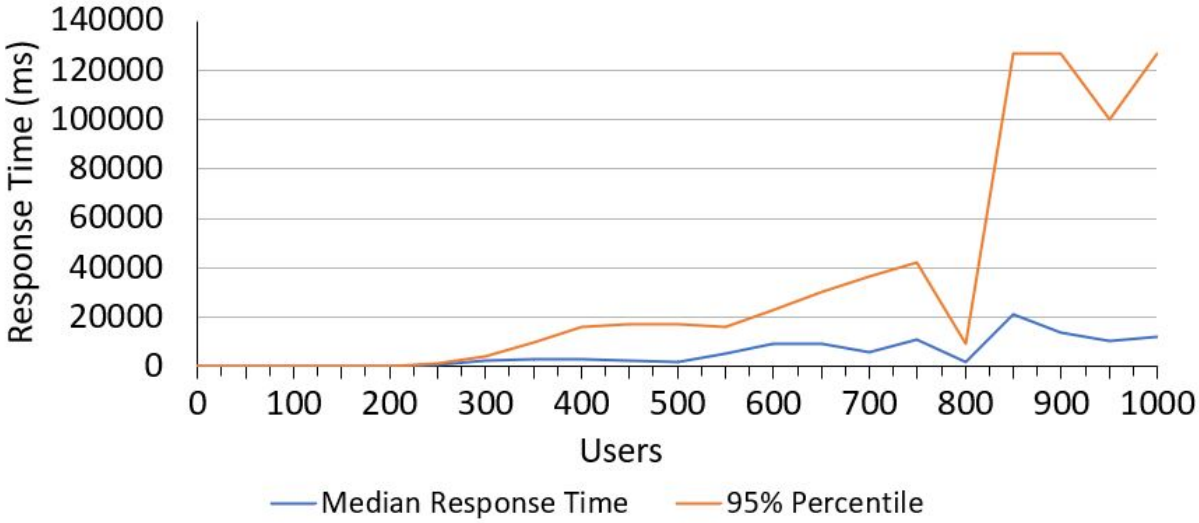


Figure 4.9: Response time with waiting between requests and distributed load

Observing the graph in the figure above, we draw the following conclusions:

- The median remained until 1000 users without significant variations in its amplitude compared to the other tests, nor did we detect any peak of relevance. In addition, we can observe that the median, as for the previous situations, started to increase with greater intensity from the ideal number of concurrent users (500), this makes sense, since a lower ability to deal with concurrency results in a longer time to respond to requests;

- The fact that the median is below 5000s more than half the time explains the low number of failed requests, which is also positive to draw from this metric;

- For the 95% percentile, we can confirm that it effectively does not stabilize at any maximum, and so we can say with certainty that it would take a few more users for this to happen.

We also took the average response time from the same test, this was 5684 ms.

CPU and Disk Utilisation

We anticipate that the load distribution will increase CPU usage in percentage, however, we are unable to predict how much the increase will be. Furthermore, we believe that Disk usage will again be residual. We present the relative information in the Table 4.9.

Users	%CPU	Devices		
		tps	kB_read/s	kB_wrtn/s
100	9,8	3,1	0	32,7
500	10,2	2,3	0	46,2
1000	8,6	1,9	0	35,5

Table 4.9: CPU and Disk utilisation

For this scenario, we also ran the Linux top command to get a sense of the CPU utilisation of each processor involved in the testing, and realised that on average each processor was using 20% of its capacity.

The CPU utilisation increases about 10 times, which we can explain by the following: each processor uses on average one fifth of its capacity, and since we use 40 processors in the load distribution, which corresponds to half of the possible CPUs, that is, 50% of the maximum machine capacity, then we conclude that on average the total usage should be:

```
%CPU = (0,2 * 0,5) * 100 % = 10 %
```

4.3.4 With variety, waiting between requests, distributed load and connection pool

In addition to the factors considered earlier in this section, we will now add a connection pool, a configuration enabled by *CKAN*, which uses *SQLAlchemy* (a *Python SQL* toolkit and Object Relational Mapper that gives application developers the full power and flexibility of *SQL*) to take advantage of this.

The connection pool technique is used to keep long-running connections in memory for efficient reuse. In addition, it provides management of the total number of connections that the system must use concurrently. To do this we had to add to our *CKAN* configuration file:

```
sqlalchemy.pool_size = 40
sqlalchemy.max_overflow = 20
```

So we have a pool with size 40, i.e. the number of core processors in our machine, and we allow 20 more connections to be used if necessary, which gives a total of 60 connections, and does not jeopardize the full utilisation of our hardware.

Throughput

Regarding this metric, we expect similar behavior to the previous case, but that the better management by the connection pool may be visible at some point in the test.

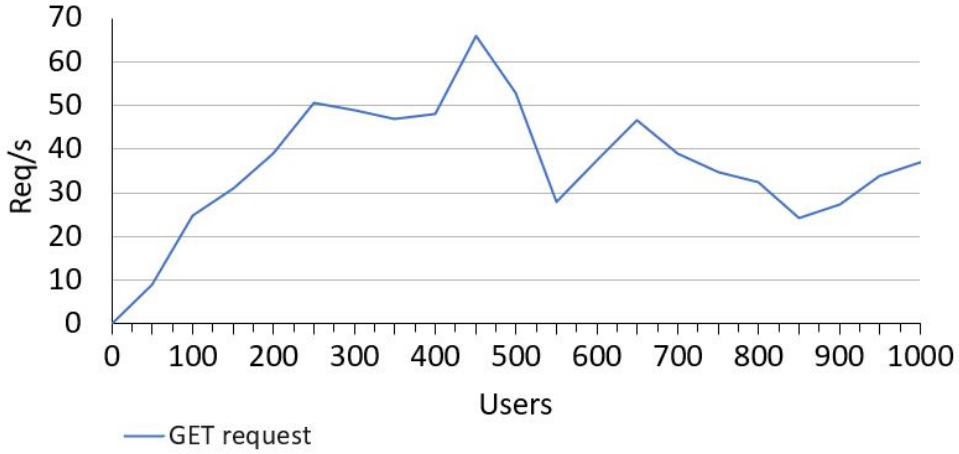


Figure 4.10: Throughput with waiting between requests, distributed load and connection pool

Thus, we confirmed that the system behaves in a similar way, but that after 500 users, which remains as the final moment of diminishing returns, it manages to maintain a throughput between 25 and 45 requests per second until the final moment of the test. This proves the efficiency of the pool management.

In order to get a general overview of this test, we can see more information about it in the Table 4.10.

# Requests	# Fails	Average size (bytes)	RPS
9545	71	52278	35.0

Table 4.10: Request statistics with waiting between requests, distributed load and connection pool

As we can see, the better management carried out by the connection pool allowed the total number of requests to increase slightly.

Response Time

In Figure 4.11 we will see a graph of the response times for the latter case. At the outset, after improving the throughput, we believe that the response time will also show better results, leveraged by the efficient use of connections.

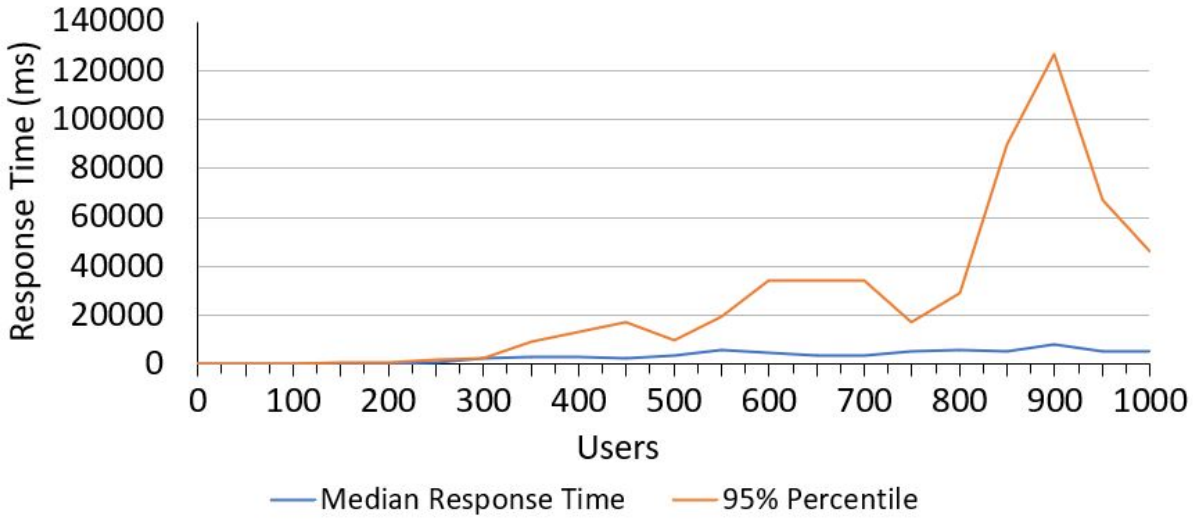


Figure 4.11: Response time with waiting between requests, distributed load and connection pool

Viewing the graph above, we then know that, for the first time, we get a system that can reach 1000 users without ever getting a median greater than 10000ms, demonstrating an almost flat line from start to finish. Certainly, this system can handle an even larger number of users. However, there is no point in exploring further, since it is already unthinkable to even consider requests with a response of about 10s.

As for the average response time, this was 5785 ms.

CPU and Disk Utilisation

CPU and Disk usage should be similar to the previous scenario, with a slight difference in CPU usage, which should be more balanced due to the connection pool. The information is presented below in table 4.11.

Users	%CPU	Devices		
		tps	kB_read/s	kB_wrtn/s
100	9,2	5,9	0	65,4
500	10,0	2,91	0	49,3
1000	9,3	2,0	0	38,2

Table 4.11: CPU and Disk utilisation

We verified what we had predicted, however, the values corresponding to the transfers per second up to 100 users, as well as the number of kB written cause some surprise, and we do not know exactly what caused this increase in relation to the previous scenario.

4.3.5 Summary

After all the tests of the preliminary case are completed, we compare the former with the latter:

- 22 % gain in the total number of requests executed;
- The average response time was almost halved (down about 40%), which turned out to be the biggest gain;
- Decrease of 80% in the number of failed requests.

We can then state that we have achieved a more efficient system than the one we initially predicted. However, we emphasize that these tests are a pessimistic estimation of how our system will work, mainly due to bandwidth sharing and write requests. We look at this pessimistic estimation as a positive thing, since in a real scenario our system will be able to give an even better answer than the one we presented.

Finally, we conclude that in the realistic scenario that we will present next it will make sense to approach a system like the last one we tested, i.e. with waits between orders, load

distribution and connection pool, since they proved to be closer to reality and allowed a better use of the resources we had available.

4.4 Realistic Case

This section includes the analysis and evaluation of the realistic case, that is, the case where both reading and writing requests are possible, with their respective probabilities of occurrence listed in table 4.2. In addition, we considered the waiting between requests, load distribution and connection pool, as we just mentioned in the last section.

We believe it would be best to do an analysis of how each request influenced the big picture, before evaluating metrics regarding throughput and response time. As such, we present Table 4.12 on the next page.

Request	# Requests	# Fails	Average (ms)	Average size (bytes)	RPS
GET /current_package_list_with_resources	756	55	17890	369734	3.0
GET /resource_show?id	626	38	16830	763	2.5
GET /package_show?id	799	32	14279	10652	3.2
GET /organization_list	481	25	15864	124	1.9
GET /group_list	473	19	13284	113	1.9
POST /package_create	163	118	18232	1063	0.7
POST /package_revise	138	16	23257	4577	0.6
POST /resource_create_default_resource_views	109	5	12272	332	0.4
POST /datastore_create	101	5	15491	106	0.4
Aggregated	3646	313	16166	79395	14.5

Table 4.12: Request statistics for reading and writing requests

As we can already see, the two biggest times are the two for writing requests, and so we already have an idea of how influential these are going to be in the big picture. Also, the number of total requests, as we can see, is the lowest so far, which was also to be expected.

4.4.1 Throughput

For this metric, we already know that adding write requests will have impact on the number of requests executed per second.

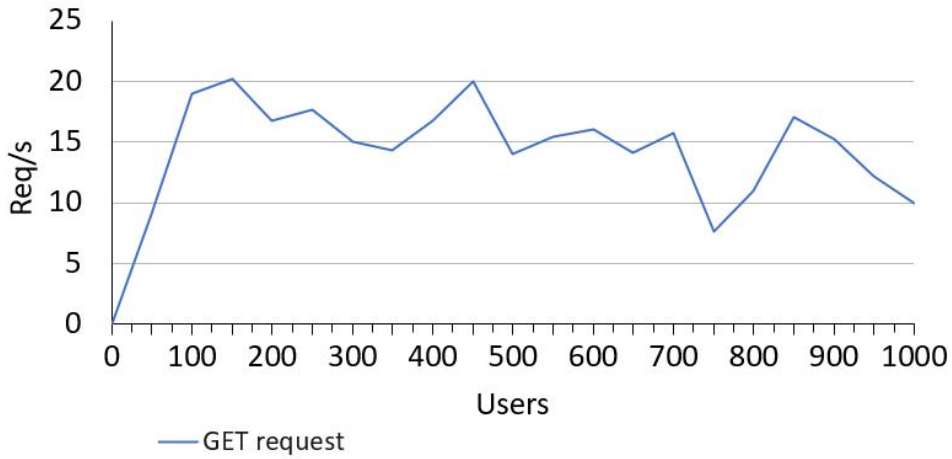


Figure 4.12: Throughput for reading and writing requests

We can then conclude from the Figure 4.12 the following:

- There is a certain balance in the values from start to finish, which highlights the factors considered (without them, we would certainly have to reduce the sample of users);
- The 100-user mark proves to be the optimal number of concurrency, since in addition to reaching the maximum peak of 19 requests per second, we consider that a user would make 1 request every 4 seconds, which making the proportion, 100 users would make a total of 25 req/s, which being a bit higher than the 19 req/s would have a slight affect at the level of response time.

.

4.4.2 Response time

For response time, we also expect the worst result so far. Nevertheless, we expect that the median response time variation, like throughput, will not show a considerable range between users.

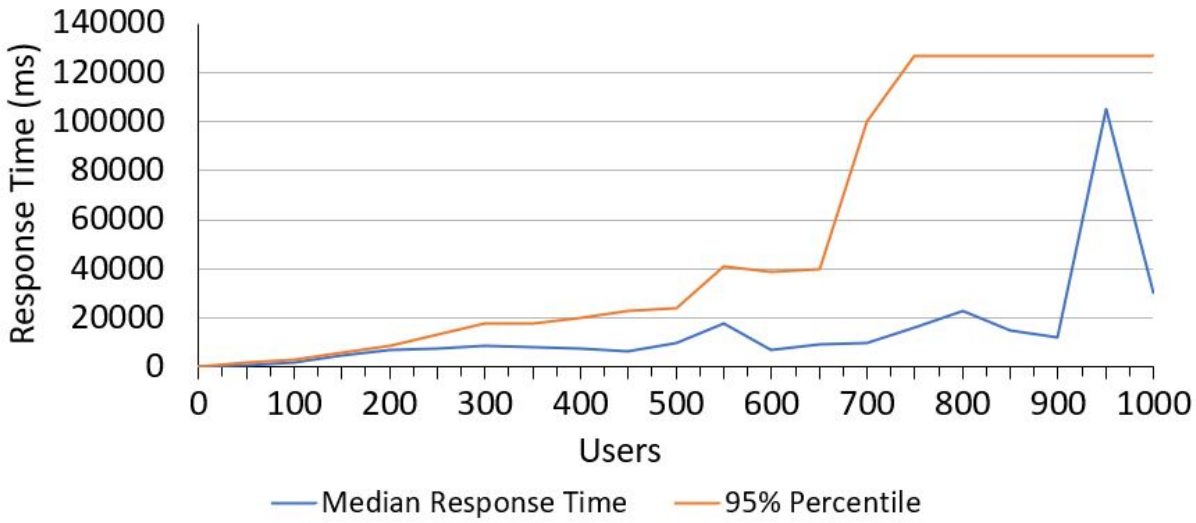


Figure 4.13: Response time for reading and writing requests

Figure 4.13 tells us that:

- In fact, the variation of the median is not as considerable as in most of the cases already worked on, although its sharpest rise starts earlier, from the 50 users, which registers a 420 ms mark as far as the median is concerned, while the 100 users that we had referenced as the competition ideal, registers 1900 ms;
- The 95% percentile ends up showing a balanced growth without peaks until it reaches a maximum peak of 127000 ms at 750 users and then stabilises.

CPU and Disk Utilisation

In a realistic scenario, we believe that the CPU utilisation will be the same as obtained from the scenarios with load distribution and that the Disk utilisation will show a marked difference from what we have analysed so far, due to the write requests. To scrutinise what we have stated, we present the Table 4.13.

Users	%CPU	Devices		
		tps	kB_read/s	kB_wrtn/s
100	10,0	111,8	0,2	3814,8
500	10,1	151,7	0,5	8023,8
1000	9,5	90,5	0,4	4743,1

Table 4.13: CPU and Disk utilisation

Thus, we conclude that what we predicted actually happens. The number of kB written per second increases considerably due to the existence of write requests on the devices, which consequently causes differences in the total data present in the system, and so when read requests are made, it is necessary to access the disk to access the new data, which are not yet in RAM and then in the CPU cache.

4.4.3 Summary

We conclude this section with a positive balance. Based on what we have just examined, we can consider that the ideal number of users of our system would be 100 users and would serve for research teams from an institute like INESC TEC to work on the platform. However, recognizing the pessimistic estimate we have, we are confident that in a real world scenario our system could possibly handle far more users, as well as enjoying an improvement of the available resources.

Chapter 5

Conclusions

This final chapter summarizes the main contributions of this dissertation and discusses guidelines for future work.

The work reported in this dissertation addressed the viability of using SDMSs in the daily life of scientific communities, proposing *CKAN* as the best solution at the moment. However, we cannot deny that *EUDAT* is an extremely interesting system and that if one day it is made available for local development it may become a more attractive solution than *CKAN*.

The solution produced led to the conclusion that it will be difficult to achieve a universal system that can meet the needs of all scientific institutions or individual researchers. Moreover, the SDMSs themselves do not appear to be seeking convergence towards universal use (Assante et al., 2016), but we should emphasize that *EUDAT* has the greatest potential to achieve this. As far as the SAIL project is concerned, we are confident that the system based on *CKAN* has been able to meet its requirements, in general.

The benchmarking performed showed promising results, and taking into account that our estimation is pessimistic, we believe that the system could respond even better in a real context. Nevertheless, it should be kept in mind that due to the hardware available for testing, we can only extrapolate its usability.

The final conclusion is that the future of research will increasingly depend on these systems, and on how, above all, they will manage to convince the entire scientific community to come on board (Poschen et al., 2012). This will require robust documentation of the systems and developer communities ready to help (Vardigan et al., 2008), as the biggest barrier remains a certain need for technical expertise to exploit the full potential

of the technologies (Gabelica et al., 2022).

5.1 Future work

One of the major goals of this dissertation was to recreate the system in an HPC environment with the help of an infrastructure such as MACC. As such, it would be worth exploring this possibility while keeping in mind the guidelines set out. In particular, it would also be interesting to consider for the MACC other hypotheses beyond the produced system, such as the possible insertion of Deucalion in the *EUDAT* network, which would allow customisation and full control of its services.

CKAN is very close to presenting its new version 2.10. Although we don't know to what extent it may affect the functionality of this system, we have the expectation that it may bring more quality and potential to its use. In addition, it will allow the stability of certain extensions, which is not possible with the current (not stable) master version. In the future, there are two main features that we suggest to be integrated into the system that are not planned for the new version: upload/delete of multiple files and statistics regarding the use of the datasets, such as qualitative evaluation and number of file downloads.

Finally, the strongest suggestion for future work is to turn the software that has been developed into production and explore its functionality in a real-world context. Ideally, it would be interesting to use the system in the service of a research institution like INESC TEC or HASLab.

Bibliography

Ckan. URL https://ckan.org/.

Eudat. URL https://www.eudat.eu/.

Locust. URL https://locust.io/.

Magda. URL hhttps://www.eudat.eu/.

Chris Armbruster and Laurent Romary. Comparing repository types - challenges and barriers for subject-based repositories, research repositories, national repository systems and institutional repositories in serving scholarly communication. March 2010.

M. Assante, L. Candela, D. Castelli, and A. Tani. Are scientific data repositories coping with research data publishing? *Data Science*, 15:6, 2016. doi: 10.5334/dsj-2016-006. URL http://doi.org/10.5334/dsj-2016-006.

Joop Aué, Arie van Deursen, Piotr Tekieli, Chengxin Ma, Johan Carvajal-Godinez, And0rz, ileontiuc, Marco di Biase, Maurício Aniche, tompeeters368, Kristín Fjóla Tómasdóttir, rbottema, Jasper Denkers, Sander van den Oever, robbertvs, mdunnewind, kogelnikp, bjohannsson, Daniël Vliegenthart, Vladimir Kovalenko, Valentine Mairet, Sambit Praharaj, Floris Verburg, Moumi, Manoj K, Lina He, LandDragoon, Kyriakos, JanZegers, and Jorden van Breemen. desosa2016: Desosa 2016, version 1.1, July 2016. URL https://doi.org/10.5281/zenodo.57924.

Susana Barbosa and Yulia Karimova. Sail data management plan. September 2021. doi: 10.5281/zenodo.5797919. URL https://doi.org/10.5281/zenodo.5797919.

Leonardo Candela, Donatella Castelli, Paolo Manghi, and Alice Tani. Data journals: A survey. *Journal of the Association for Information Science and Technology*, 66(9):

1747–1762, 2015. doi: https://doi.org/10.1002/asi.23358. URL https://asistdl.onlinelibrary.wiley.com/doi/abs/10.1002/asi.23358.

José Carvalho, Pedro Príncipe, Augusto Ribeiro, and Filipe Furtado. Relatório sobre ckan. November 2016. URL https://hdl.handle.net/1822/46094.

Ricardo Carvalho Amorim, João Aguiar Castro, João Rocha da Silva, and Cristina Ribeiro. A comparison of research data management platforms: architecture, flexible metadata and interoperability. *Universal Access in the Information Society*, page 851–862, June 2016. doi: 10.1007/s10209-016-0475-y. URL https://doi.org/10.1007/s10209-016-0475-y.

Mirko Gabelica, Ružica Bojčić, and Livia Puljak. Many researchers were not compliant with their published data sharing statement: a mixed-methods study. *Journal of Clinical Epidemiology*, 150:33–41, 2022. ISSN 0895-4356. doi: https://doi.org/10.1016/j.jclinepi.2022.05.019. URL https://www.sciencedirect.com/science/article/pii/S089543562200141X.

Jim Gray, David Liu, Maria Nieto-Santisteban, Alexander Szalay, David DeWitt, and Gerd Heber. Scientific data management in the coming decade. *SIGMOD Record*, 34, March 2005. doi: 10.1145/1107499.1107503.

Damien Lecarpentier, Peter Wittenburg, Willem Elbers, Alberto Michelini, Riam Kanso, Peter Coveney, and Rob Baxter. Eudat: A new cross-disciplinary data infrastructure for science. 8, 2013. doi: 10.2218/ijdc.v8i1.260. URL https://doi.org/10.2218/ijdc.v8i1.260.

Dong Joon Lee and Besiki Stvilia. Practices of research data curation in institutional repositories: A qualitative view from repository staff. *PLOS ONE*, 12(3):1–44, 03 2017. doi: 10.1371/journal.pone.0173987. URL https://doi.org/10.1371/journal.pone.0173987.

David Loshin. *Master data management*. Morgan Kaufmann, 2010.

Meik Poschen, June Finch, Rob Procter, Mary Goff, Mhorag McDerby, Simon Collins, Jon Besson, Lorraine Beard, and Tom Grahame. Development of a pilot data management

infrastructure for biomedical researchers at university of manchester – approach, findings, challenges and outlook of the madam project. 7, 2012. doi: 10.2218/ijdc.v7i2.234. URL https://doi.org/10.2218/ijdc.v7i2.234.

Karima Rafes and Cécile Germain. A platform for scientific data sharing. In *BDA2015 - Bases de Données Avancées*, Île de Porquerolles, France, September 2015. URL https://hal.inria.fr/hal-01168496.

Carol Tenopir, Elizabeth D. Dalton, Suzie Allard, Mike Frame, Ivanka Pjesivac, Ben Birch, Danielle Pollock, and Kristina Dorsett. Changes in data sharing and data reuse practices and perceptions among scientists worldwide. *PLOS ONE*, 10(8):1–24, 08 2015. doi: 10.1371/journal.pone.0134826. URL https://doi.org/10.1371/journal.pone.0134826.

Mary Vardigan, Pascal Heus, and Wendy Thomas. Data documentation initiative: Toward a standard for the social sciences. 3, 2008. doi: 10.2218/ijdc.v3i1.45. URL https://doi.org/10.2218/ijdc.v3i1.45.

Mark Wilkinson, Michel Dumontier, and IJsbrand Aalbersberg *et al.* The fair guiding principles for scientific data management and stewardship. *Sci Data*, 3, 2016. doi: 10.1038/sdata.2016.18. URL https://doi.org/10.1038/sdata.2016.18.

Joss Winn. Open data and the academy: an evaluation of ckan for research data management. May 2013.