Final Year Project

Datacenter Management Tool

Daniel Houlihan

Student ID: 18339866

A thesis submitted in part fulfilment of the degree of

BSc. (Hons.) in Computer Science

Supervisor: Damian Dalton

UCD School of Computer Science

University College Dublin

April 30, 2022

Abstract

The web application created as part of this project aims to provide datacenter administrators with tools to lower their carbon emissions and operational costs. Using software to measure the energy consumed by entities within a datacenter, useful metrics and visualisations will be conveyed to the administrator. With the ever-increasing amount of data that needs to be stored globally, the amount of datacenters required is also increasing. These datacenters are producing carbon emissions and until renewable energy is readily available, the best way to tackle this is to encourage and facilitate proper datacenter management. In most small to medium-sized datacenters, such as at a small company or university, there is little to no feedback from the physical hardware about which servers are running efficiently and which need reconfiguration.

Table of Contents

Chapter 1: Project Specification

The purpose of this project is to enable owners of a small datacenter environment to reduce their carbon footprint and save money. This will be achieved by highlighting inefficiencies and possible reconfigurations that can be made to their datacenter. The Papillon software developed by Beeyon will be used to measure and evaluate multiple components of the datacenters consumption and emissions [1].

According to McKinsey and Company, typical server utilisation in a business and enterprise datacenter rarely exceeds 6% over the course of a year [2]. A separate study by the same entity revealed that up to 25% of physical servers were "comatose" [3]. Comatose servers are those that have an average server utilisation of 3% or less. There is a unique opportunity for datacenters to substantially reduce their emissions and operational costs here without incurring any capital expenditure at all. Other ways to reduce emissions and costs are more often than not very expensive, such as upgrading the existing hardware.

In the last few years, it has become a top priority for companies and organisations to reduce their carbon footprint. This is due to an increasing awareness of the damage that carbon emissions are doing to the planet as well as pressure from governments and the general public. A carbon tax was introduced in Ireland in 2010 which incentivises not just datacenters but everyone to reduce their carbon footprint [4].

Project Objectives:

Core	Advanced
• Create an easy to use interface that provides all information that a datacenter administrator requires to make informed decisions.	• Implement a budget system which alerts the administrator if their carbon emissions are likely to, or have, exceeded their specified budget.
• Allow for both set audit periods and live audit periods (updatable)	• Allow the service to connect to multiple Papillon master servers (multiple different datacenters).
• Calculate and graph at a daily level the cost, carbon footprint, and energy consumption of the datacenter and individual servers.	• Provide the option to use the current carbon intensity in Ireland to calculate the carbon footprint on a rolling basis.
• Employ an industry standard development and test cycle to provide a stable release.	

Links:

GitLab: https://csgitlab.ucd.ie/danielhoulihan/fyp_datacenter_management

Demo Video: https://drive.google.com/file/d/1Ki3sNgRQNWIsZqvzZYCo3Nslu1Wu_WTN/view?usp=sharing

Chapter 2: **Introduction**

This project is being done as part of a Computer Science degree at University College Dublin. My hopes are that the end product can be used by the university to reduce the carbon footprint and costs of their datacenter.

This software will provide meaningful insights to administrators, such as which servers are being underutilised. Reducing carbon footprint and costs in a datacenter of any size requires a system in place that will provide detailed and accurate information about how each piece of hardware is being utilised. Without a system like this, it is a guessing game as to which servers are the main offenders of costs and carbon emissions. This software will make it easier for administrators to reduce their footprint while not compromising the security or reliability of the data that they have stored as they will still maintain full control over the datacenter.

In 2010, the International Data Corporation (IDC) calculated that 1.2 zettabytes of new data had been created globally, 50% higher than in 2009 [5]. In 2020, the amount of data created was approximately 59 zettabytes. The IDC now predict that by 2025, 175 zettabytes of data will be created annually [6]. Between 2018 and 2020, more data was created than all of human history previous to 2018. With this huge increase of data comes an increase in capacity requirements in the form of new datacenters being built all over the world. Until renewable energy is fully available globally, these datacenters will be contributing to greenhouse gases. In the meantime, we can encourage proper management and techniques for maximising efficiency.

The most important aspect of the application is the ability to create live audit periods. For example, you have a medium sized datacenter and you know that in the previous year the costs were 10,000€ and that it produced roughly 1,000 kilograms of CO2 to run. You are starting the new year and you have decided to cut down on your costs and emissions. Once you configure your datacenter on the application, you can update it daily and you can see your costs and emissions in real time as the year goes on. You will be able to see clearly through metrics and visualisations:

- How much your datacenter has consumed
- Exactly how much each piece of hardware has consumed
- Which servers are being underutilised
- Which servers are the most expensive/ emit the most CO2

The web application will be built using Python Django and will gather data from multiple API's including those available through Papillon. The web application will be able to run on any operating system so long as it has Python installed. It only needs to be available on the local network and so will not require a hosting server or https. This application will only be available to administrators of a datacenter and so it will not be available publicly.

Chapter 3: Related Work and Ideas

This chapter outlines in detail the main concerns and challenges facing datacenter efficiency and the technologies available to remedy these. The first item to be discussed is the extent of datacenter energy consumption but the main topic of discussion will be the inefficiencies in datacenters, how to calculate these, and what techniques can be applied to reduce them.

3.1 Energy Usage in Datacenters Globally

The European Commission released a paper [7] in 2017 highlighting the trends of datacenter consumption across Europe. It found that the Information and Communication Technology sector (ICT) generates up to 2% of all CO2 emissions worldwide [7]. A large portion of this is taken up by datacenters. It's estimated that datacenters have the fastest growing carbon footprint of all subsectors of ICT. This of course is a very worrying statistic and should be addressed with some degree of urgency. Studies of this scale take a lot of manpower, time, and money and are therefore not done very regularly. In 2011, it was estimated that datacenters accounted for 1.4% of global energy consumption and that this would increase by 4.4% each year compounded[7]. This would mean that in 2021, datacenters would be consuming 2.15% of all global energy. There have been no such reports in the last few years that are extensive enough to prove or deny this.

The European Code of Conduct for datacenter Energy Efficiency [8] was introduced in 2000 to address the increasing power usage and to promote best practices. Organisations can register their datacenters and provide monthly energy statistics. In the European Commission report[7], 289 datacenters were included across Europe. The study found that the participating datacenters had a PUE [9] average of 1.64 at the end of 2016. PUE is the Power Usage Effectiveness and describes the efficiency with which a datacenter uses energy. This is lower than in previous years and concludes a downward trend in PUE values from 2009 - 2016. When recorded in 2009, the average PUE value was 1.87. Only 289 large datacenters were included in this survey which is only scratching the surface of the thousands of datacenters across Europe, small and large. Interestingly, the Nordic countries have a lower PUE than any other region in Europe, while the Southern Europe/ Mediterranean areas have the worst PUE [9]. This is due to the fact that cooling is much cheaper in cold climates as they need only pump air into the datacenter, whereas in hotter climates they need to condition the air first.

It has been estimated that US datacenters in 2017 alone consumed 90 billion kilowatt-hours (90 terawatt-hours) of electricity[10]. Globally, this figure amounts to 416 terawatt-hours of electrical usage for datacenters[10]. To put that into context, the country of Ireland only consumed 26 terawatt-hours of electricity in 2017[11]. The vast majority of this energy is, unfortunately, originating from fossil fuels. The amount of renewable energy available simply is not enough at this time. A datacenter's architecture could quite easily be constructed in a way that would see a massive reduction in electricity usage. However, this equipment is expensive and most organisations will not use such servers because of their price. It is much cheaper to use commodity hardware but it does result in less efficient datacenters which ultimately leads to more carbon emissions. The tool I develop will promote the proper utilisation of such commodity hardware machines.

3.2 Datacenter Costs

The total cost of running a datacenter can be split up into three separate elements. These elements are; capital, reliability, and operating costs[12] as shown in Figure 3.1. Capital costs include the procurement of the hardware, facilities, and infrastructure. Reliability costs are completely up to the owners of the datacenter. Spending less on this will make your datacenter prone to outages or failures so if you want your datacenter to be fault tolerant you may need to invest more in backups, redundancy, and detection software/ hardware. Operating costs include consumables such as energy as well as staff.

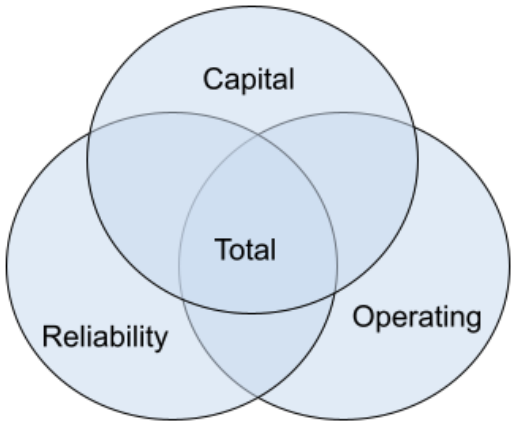


Figure 3.1: Main costs in a datacenter

For my project, I will be focusing on the operating costs as this is the only element that can be brought down without incurring any additional expenditure.

3.3 Zombie / Comatose Servers

Koomey Analytics undertook an analysis of datacenters in 2017 which comprised of 16,000 servers across 10 facilities over a period of 6 months[3]. This analysis showed that up to one quarter of servers were comatose in companies where no actions were taken to remove them or shut them off. It also found that around 30% of virtual servers were comatose as well. A server is labeled as zombie / comatose if it has been using less than 3% CPU utilization for the last 6 months but remains switched on and consuming energy. This is the second such study by Koomey, the first in 2014 where they found up to 30% of physical servers were comatose. The newer study includes four times as many servers and twice as many facilities which may be why this value was higher for the first study.[2].

As mentioned, only 16,000 servers across 10 datacenters were analysed in the 2017 study[3]. This is a tiny sample size as compared to the global amount of datacenters and servers so these results are not to be used to draw concise conclusions. They are, however, a good indication of how datacenters can reduce their carbon footprint and costs by simply reconfiguring their architecture. There is no need to buy more efficient servers or upgrade their datacenter. My tool will effectively eliminate this problem altogether as it will show clearly to any administrator which servers can be safely shut down or moved.

3.4 PUE

"PUE is determined by dividing the amount of power entering a datacenter by the power used to run the computer infrastructure within it."[9] Power Usage Effectiveness is a metric that I will be referring to quite a bit in this paper. It is essentially a measure of how much energy is being used in a datacenter including both computational energy and overheads such as cooling, lights, and security. PUE is now the industry standard for calculating energy efficiency in datacenters. GreenGrid published a set of rules for calculating PUE to ensure consistency when comparing datacenter metrics [13]. PUE does not represent the productivity of a datacenter, it is an indication of how efficient it is with its energy. There are many metrics that will need to be used in this paper to effectively quantify the carbon footprint of a datacenter and the cost of running it as PUE is not a comprehensive efficiency metric.

$$PUE = \frac{TotalFacilityEnergy}{ITEquipmentEnergy} \tag{3.1}$$

3.5 Characteristics of Low Carbon Datacenters

The carbon footprint of a datacenter is calculated from 3 major components. These components are; datacenter operations, manufacture of new/replacement of old equipment, and disposal/recycling of old equipment. The latter two are termed embodied emissions because the emissions are produced when making or destroying the physical devices [14]. My initial assumptions were that the carbon footprint of a datacenter is made up primarily of operational rather than embodied components, simply because embodied emissions are only once off for each hardware device.

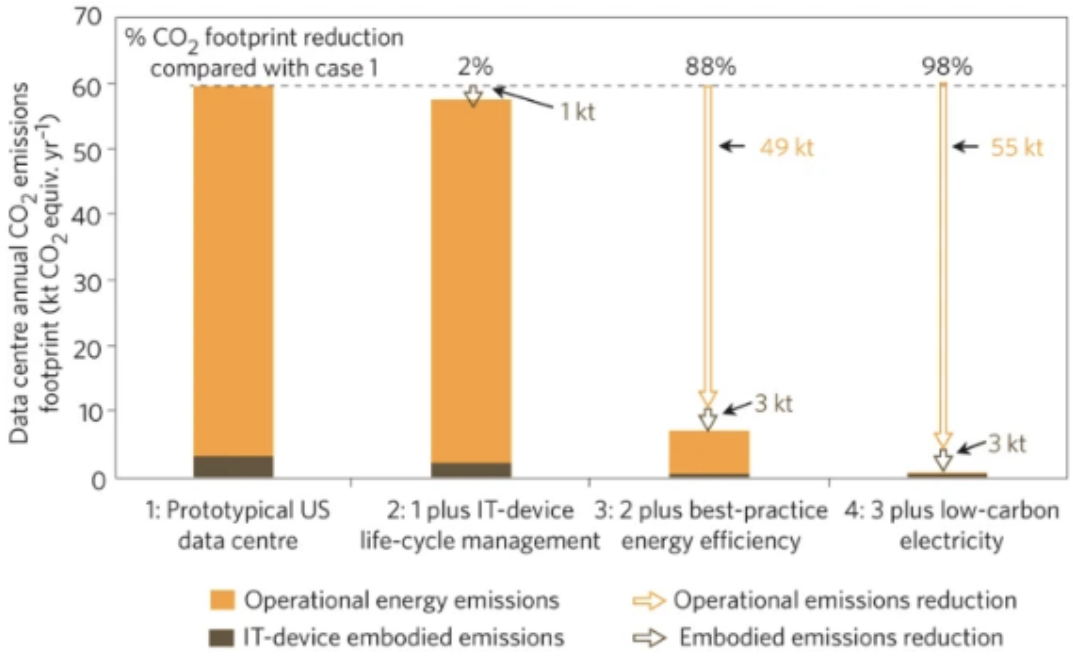


Figure 3.2: Possible Emission Reductions in datacenters [15]

Figure 3.2 above shows the possible reductions of CO2 emissions in a prototypical datacenter located in the US with 20,000 servers, 40,000 hard disk drives, 2,060 network switches, and the US average PUE [9] which was 1.8 at the time of this paper (2013).

Bar chart columns

1. Carbon emissions of a prototypical datacenter in a US enterprise-class facility.
2. Possible reduction of emissions with 100% recycling and proper lifetime extension of the devices added.
3. Possible reduction of emissions by utilising top of the range devices in terms of efficiency and deploying proper energy management.
4. Possible reduction of emissions by using renewable/ low-carbon electricity

We can conclude that reducing the operational emissions is much more effective at reducing the overall footprint than the proper management of embodied emissions. Proper utilisation of devices will lead to fewer devices being needed, which in turn will lower the embodied emissions as well. This was published in 2013 and the overall efficiency of hardware has increased substantially since then so the reductions won't be as substantial as seen in this graphic. The tool that I develop will be used to ensure proper and efficient utilisation of devices, lowering mostly the operational but also indirectly the embodied emissions.

3.6 Attitude Change

In recent years there has been a big shift in the attitude of companies in relation to their carbon footprint. If it is a medium to large sized tech company/ organisation, you can assume that they have their own datacenter which has operational costs and carbon emissions. Surveys, which were carried out as part of a report in 2020 [16] by 451 Research[17], a member of S&P Global Market Intelligence[18], show how companies have ranked the importance of efficiency and sustainability in 2020 and how they will rank it in 2023.

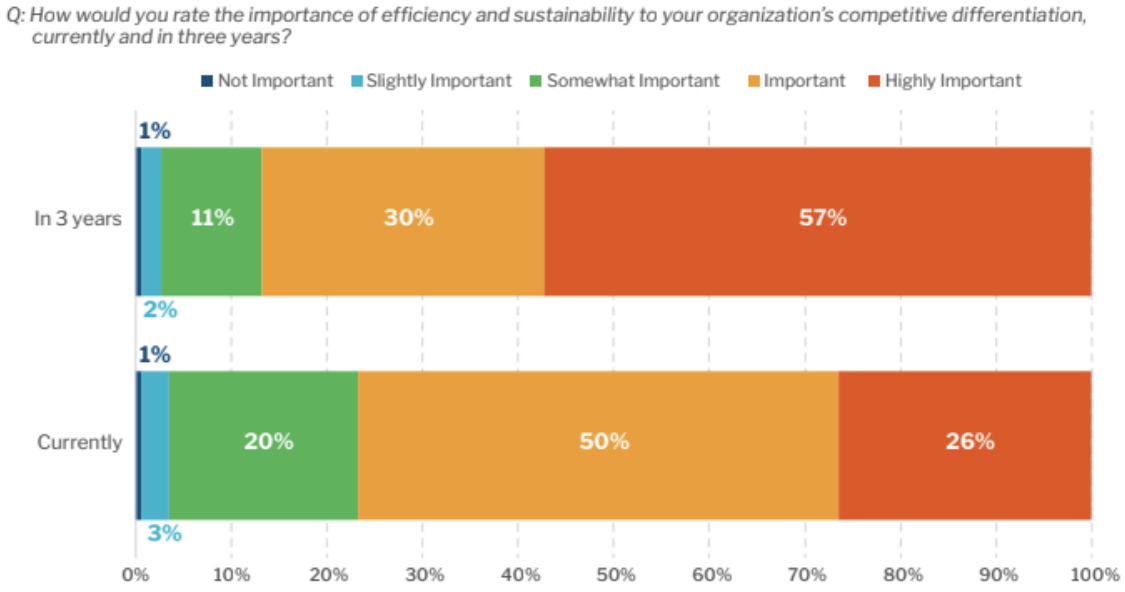


Figure 3.3: Present and Future Attitude of Organisations with respect to efficiency and sustainability [16]

In Figure 3.3 above, you can clearly see that organisations are taking sustainability and efficiency more seriously, and will be doing so at an increasing rate year on year. There are many different

factors driving this including; customer requirements, regulatory requirements, public opinion, long-term operational resiliency, etc.

3.7 Virtualisation

"Server virtualisation is the process of dividing a physical server into multiple unique and isolated servers by means of a software application"[19]. There are many different types of virtualisation, each suited for different use cases. Operating system level virtualisation virtualises the physical server at the operating system level. The host OS becomes a modified kernel that can execute multiple isolated containers. This type is widely used and is quite efficient - but it does not support multiple kernels. Para-virtualisation adds instructions that override those of the physical machines instruction set architecture. This allows for multiple kernels but means the kernel of the guest OS must be modified. Full virtualisation is a method where operating systems and software run on virtual hardware, meaning computer service requests are separated from the physical hardware. Full virtualisation is the most common for sharing hardware servers so this is the one which will be promoted most in this project. A virtual machine monitor (VMM), also known as the hypervisor [20], is software that creates and runs virtual machines. The hypervisor manages the resources which allow a machine to run multiple VMs by virtually sharing memory and processing resources. These are all important concepts for this project as the focal point of the tool is conveying to the administrator of a datacenter how they can better utilise their datacenter using virtualisation of the servers.

One concern regarding virtualisation is its multi-tenancy nature. Some users may not want to share a hardware device with another due to security concerns. This is not too much of an issue in this scenario as my tool is to be used in a small datacenter environment. It is assumed that this will be used only internally within an organisation. Even so, using full virtualisation does provide a high level of security as each guest operating system is encapsulated.

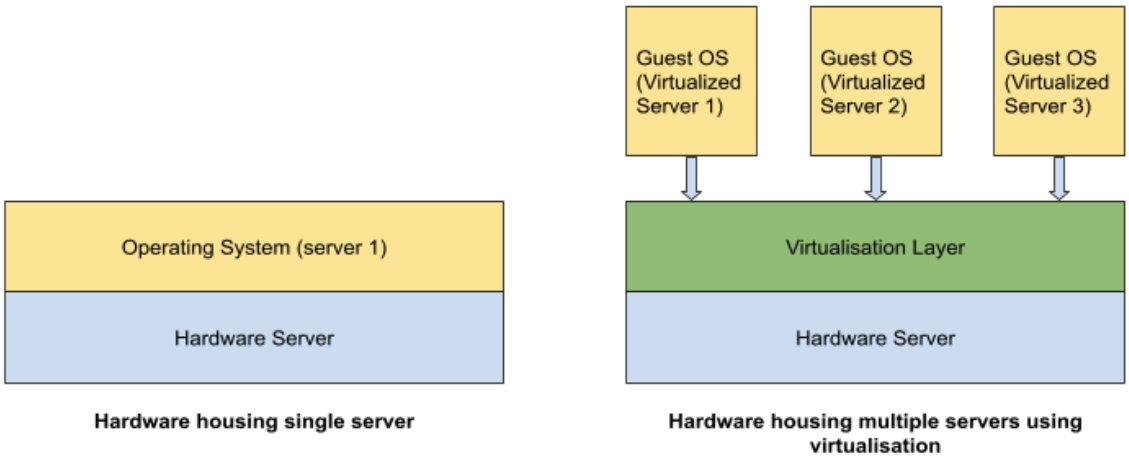


Figure 3.4: Virtualising multiple servers on one piece of hardware

Figure 3.4 shows how full virtualisation can be used to host multiple servers on a single hardware device.

Promoting virtualisation and sharing server space is a feature of the application. Showing which servers are using only a fraction of their capacity will allow administrators to decide which hardware is potentially suitable for this

3.8 Virtualisation in Datacenter Case Study

'An Investigation into Energy Efficiency of datacenter Virtualisation'[21], published in 2010, includes the results of tests that were performed on a set of servers to see how much energy could be saved through the utilisation of virtualisation. In this case, a physical power meter was used to track the power consumption of the devices while the CPU was monitored by performance software. Full virtualisation, as discussed earlier, was used. VMware was used to create the virtual machines

Using virtualisation we can host many virtual servers on one physical server. As mentioned earlier, it's estimated that datacenters rarely deliver more than 6% of their maximum computing power over the course of a year[2] and that up to 25% of physical servers are comatose[3]. Comatose servers can be switched off while other servers which are only utilising a small amount of their capacity can and should be virtualised together with others on a single piece of hardware. This way we can store two or more 'servers' on a single piece of hardware. Figure 3.5 below shows the overall CO2 emissions of 21 physical servers versus 12 physical servers which have 9 of the 21 servers virtualised. Not all servers are suitable for virtualisation - such as backup servers or servers which are already at >60% CPU capacity The two infrastructures are performing the same tasks and no computational power is lost.

The different servers in this test were running a range of different operating systems which do vary slightly in terms of energy consumption. The servers of a small datacenter will predominantly be Linux based.

	Evaluation - Physical vs. Virtualised Infrastructure	
	Physical Infrastructure	**Virtualised Infrastructure**
Number of physical servers	**21 Physical Servers**	**12 Physical & Virtualised**
Power (Servers)	7.6 kW	*5.3 kW*
BTU/hour	26 kilo-BTUs /hr	*18 kilo-BTUs /hr*
Power (Cooling)	4.3 kW	*2.9 kW*
Annual kWh (Servers)	67.1 MWh	*46.8 MWh*
Annual kWh (Cooling)	37.7 MWh	*25.4 MWh*
Annual CO_2 (Servers)	63.7 tonnes	*44.5 tonnes*
Annual CO_2 (Cooling)	35.8 tonnes	*24.1 tonnes*
Total Annual kWh	104.7 MWh	*72.3 MWh*
Total Annual CO_2	99.5 tonnes	*68.6 tonnes*

Figure 3.5: Physical Vs Virtualised [21]

3.8.1 Carbon Intensity

Carbon intensity is the number of grams of carbon dioxide (CO2) produced to make one kilowatt-hour of electricity[22]. When electricity is created through the burning of fossil fuels, the carbon intensity is high as carbon is emitted when generating the electricity. Using renewable forms of energy such as hydro, wind, and solar produces almost no carbon. This figure is needed to accurately calculate the carbon footprint of a datacenter and so this must be input by the administrator when using the application. An API will be used to get the national average if the figure is unknown [23].

3.9 Papillon

Papillon (Profiling and Auditing of Power Information in Large and Local Organisational Networks) [1] is a software tool developed by Beeyon. Beeyon specialises in datacenter energy management and performance technology [24]. Papillon uses no physical meters or any physical hardware. It is a complete software design solution for measuring power and efficiency in datacenters. Rather than physical meters, it uses patented modeling technology which delivers power analysis that is over 98% accurate compared to industry standard physical metering. Each server type has its own power model which is created by Beeyon and allows for such high accuracy.

Papillon employs a client-server architecture. In a datacenter, there is one master server and any number of client servers to be monitored. Agents are installed on all of the client servers and any data that is acquired is communicated to the master server which hosts the power models and a database. The master server then calculates and saves all metrics received from the servers.

One major advantage of using Papillon over physical metering is that it can measure the power consumption of all applications on the server separately. Leveraging this, you can even monitor virtual machines as separate entities. In the Papillon API, A server entity is named a 'host'. A host can be either a physical server or a virtual machine with its own IP address.

Papillon constantly monitors the servers and saves data every 60 seconds. The granularity is incredibly fine and allows for accurate and insightful metrics to be calculated.

Integrating Papillon into existing architecture is easy as it adopts a complete REST-ful API [25]. Papillon can be downloaded directly from the internet and installed onto a server. It operates securely behind the datacenters firewall and the agents only communicate directly with the master server. Each client in the figure below shows a host (VM or physical server).

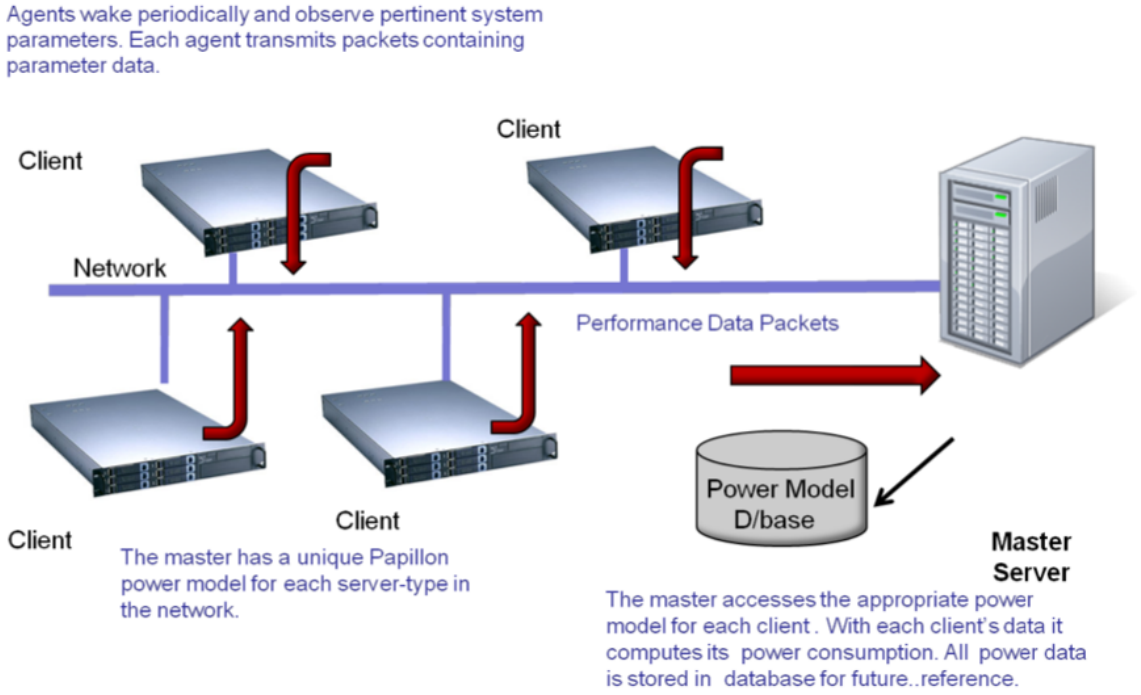


Figure 3.6: Papillon client master communication [26]

3.10 Django

Django, developed between 2003 and 2005, is a high level Python web framework that enables the rapid development of secure, maintainable, and scalable websites [27]. It is free and open source. Django is used by a surprising number of well known websites such as; Instagram, Spotify, YouTube, DropBox, Eventbrite, and Quora [28].

I chose to use Django for this web application for the following reasons.

1. Complete: Follows a 'batteries included' philosophy. Most things a developer could hope for come included and there's little need for imports and 3rd party libraries.
2. Versatile: Django can be used to build any type of website - It gives the freedom to pivot the web application as I see fit during the development
3. Secure: although security will not be of major concern for this web application as it is hosted only on the local network, Django has built in protection against many exploits by default, including; SQLi, XSS, CSRF.
4. Maintainable: As opposed to many Java approaches of building web applications, Django requires much less boilerplate code and encourages a DRY approach (Don't Repeat Yourself).
5. Portable: This web application will need to run on all major operating systems (Windows, Linux, MacOS). Being built in Python means this is possible.

Django web applications work in the following way.

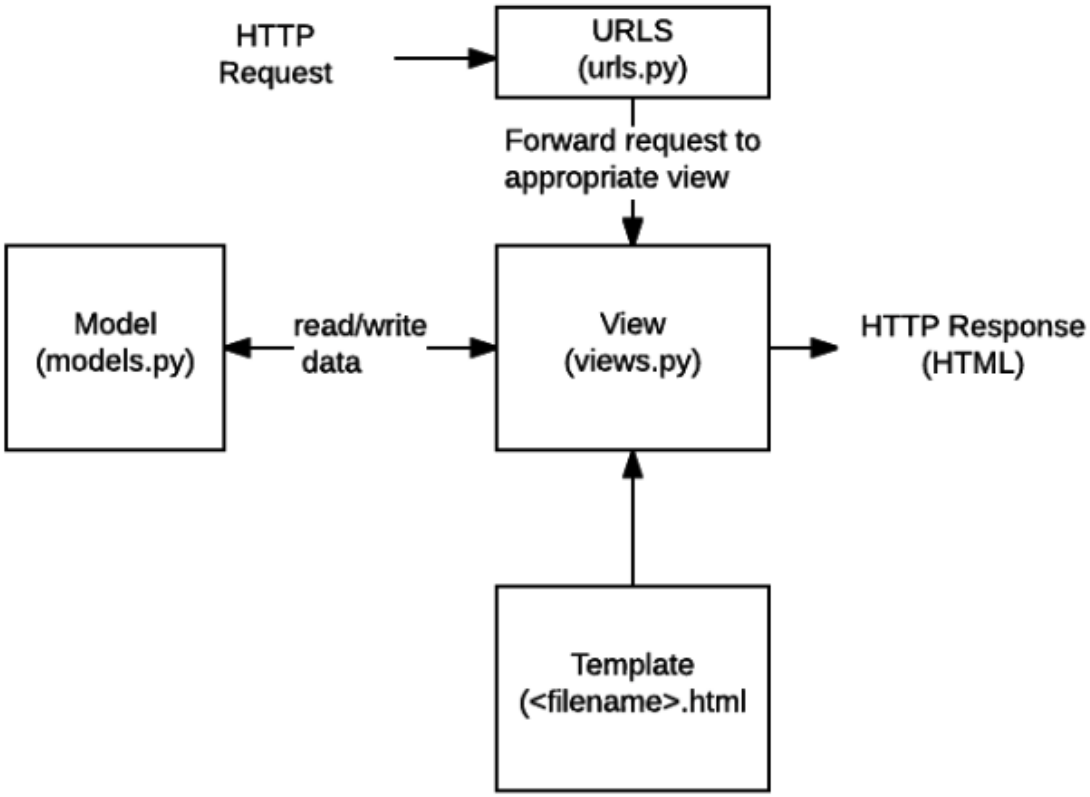


Figure 3.7: Django Architecture [29]

A URL mapper (urls.py) redirects http requests to the specified view based on the request URL. Django runs through each URL pattern in order and stops at the first one which matches the

requested URL. When a match is found, Django imports and calls the given view. If no matches are found, an exception is raised. A view function (views.py) is a Python function that takes a web request and returns a web response. The response will normally be HTML content for a page but it can also be a redirect, a 404, an XML file, an image, etc. The view will contain any logic necessary to return the response with any data required. Often this data is retrieved from a model.

Models (models.py) are Python objects containing fields and behaviours of data stored. Each model maps to a single database table. The default database for Django is SQLite3.

Templates are used to generate HTML dynamically. Templates contain static parts as well as dynamic variables which will be filled by the view.

3.11 Conclusions

My background research has allowed me to come to a few conclusions. First, and most importantly, is that the best element of a datacenter to target for reducing carbon emissions and costs is the operating component. The main reason for this is that it is the cheapest and easiest solution for administrators of a small datacenter to reconfigure. Papillon will be used to gather the relevant information on a datacenter. Django is the framework I have chosen to use to build the application.

Chapter 4: Data and Context

4.1 Papillon API

This web application relies heavily on data from the Papillon REST API[25]. Data from Papillon is needed to discover the available datacenters, floors, racks, and hosts. Once these have all been identified and mapped out, we can then use the properties to make new API calls, this time to find the power and energy used by each host. datacenters are formed in a tree-like structure of nodes going from datacenter -> floor -> rack -> host. This relationship is shown in Figure 4.1.

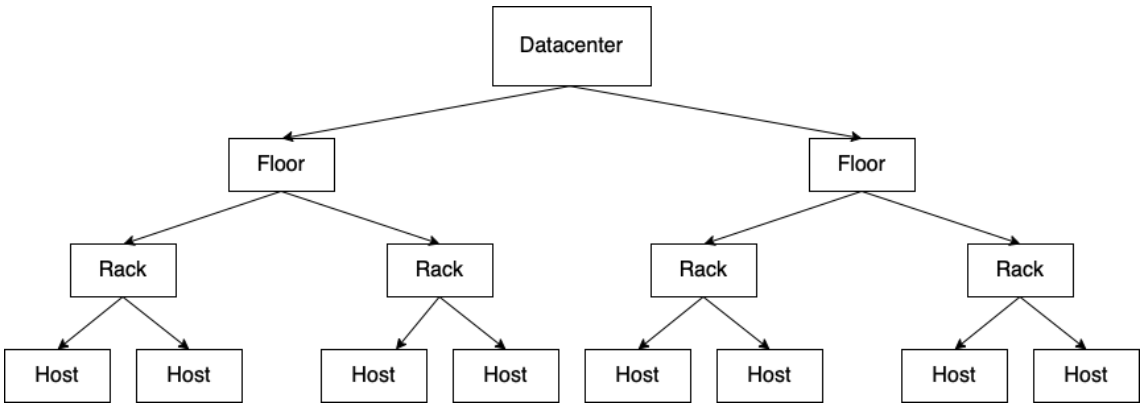


Figure 4.1: Papillon datacenter layout

Papillon is designed in a RESTful way. Once Papillon is installed and running on the master server, energy and power information about the master and clients are reachable via an API endpoint on the local network in real time. The base URL for the Papillon API is:

http://<master>:8080/papillonserver/rest

In order to have the API return JSON objects, it must be specified in the request headers. There are 4 different Papillon endpoints that need to be accessed to gather all required data for the calculations. These endpoints are:

1. http://<master>:8080/papillonserver/rest/datacenters

   This endpoint returns all datacenters available on the master IP address. It includes additional information about the datacenter such as description and location.

Code Listing 4.1: API response containing datacenters

```
{"hostExtended": [
    {
    "id": 1,
    "name": "dc1",
    "description": "dc1 description",
    "latitude": 2.0,
    "longitude": 0.0
    }, ... ]
}
```

2. http://<master>:8080/papillonserver/rest/datacenters/<datacenterId>/allhosts

   This endpoint returns all hosts available on the specified master IP and specified datacenter. It includes information about the datacenter, floors, racks, and hosts. A host can be either a physical server or a virtual machine with its own IP address.

Code Listing 4.2: API response containing host locations

```
{"hostExtended": [
    {
    "datacenterId": 1,
    "datacenterName": "dc1",
    "datacenterDescription": "dc1 description",
    "floorId": 2,
    "floorName": "floor1",
    "floorDescription": "floor1 description",
    "rackId": 3,
    "rackName": "rack1",
    "rackDescription": "rack1 description",
    "hostId": 4,
    "hostName": "host1",
    "hostDescription": "host1 description",
    "modelGroupId": 5,
    "modelGroupName": "Dell-PowerEdge",
    "hostType": "SERVER",
    "IPAddress": "127.0.0.1",
    "processorCount": 4,
    "VMCount": 1
    }, ... ]
}
```

3. http://<master>:8080/papillonserver/rest/datacenters/<datacenterid>/floors/<floorid>/racks/<rackid>/hosts/<hostid>/power?starttime=<start>&endtime=<end>

   Given a master IP address, datacenter ID, floor ID, rack ID, host ID, start time, and end time, this API returns power information for the host each minute it's active, accompanied by the timestamp. Power information is the amount of watt-hours the host consumed for that minute.

Code Listing 4.3: API response containing host power

```
{"power": [
    {
    "power": 1.7,
    "timeStamp": 1649116855
    }, ... ]
}
```

4. http://<master>:8080/papillonserver/rest/datacenters/<datacenterid>/floors/<floorid>/racks/<rackid>/hosts/<hostid>/activity?starttime=<start>&endtime=<end>

   Given a master IP address, datacenter ID, floor ID, rack ID, host ID, start time, and end time, this API returns activity information for the host each minute it's active, accompanied by the timestamp. Stat1 is the average CPU utilisation over the last minute (percentage).

Stat2 is the Disk I/O for the last minute (pageins and pageouts). Stat3 is the Network I/O for the last minute. (bytesin and bytesout)

Code Listing 4.4: API response containing host activity

```
{"activity": [
    {
    "id": "1",
    "hostId": 200,
    "power": 1.7,
    "powerMode": ""AC_DEFAULT",
    "stat1": 0.1,
    "stat2": 20000,
    "stat3": 30000,
    "timeStamp: 1649116855,
    "allApps": 200.1,
    "apps": {
        "app"[ {
            "appId": 13,
            "activityId": 294185,
            "name": "Java",
            "cpu": 4
        }, ... ]}
    }, ... ]
}
```

4.2 CO2 Signal

CO2 Signal provides a free, limited API for educational purposes which returns, in real time, the carbon intensity and fossil fuel percentage for a given country [23]. The carbon intensity is required by the user so if they have no figure for their own datacenter they can take the national average. Fossil fuel percentage will also be useful for the user as they can decide when the best time is for heavy processes.

```
{
  "disclaimer": "",
  "status": "ok",
  "countryCode": "IE",
  "data": {
    "datetime": "2022-04-11T17:00:00.000Z",
    "carbonIntensity": 234,
    "fossilFuelPercentage": 38.72
  },
  "units": {
    "carbonIntensity": "gCO2eq/kWh"
  }
}
```

Chapter 5: Project Aproach

5.1 Dummy Datacenter

In order to use the Papillon API, I first needed a server running Papillon. Unfortunately, I had no access to any such servers. I was not granted SSH permission to the UCD datacenter for obvious security reasons. The solution to this problem was to create two virtual machines on my laptop, both with Papillon installed on them. These virtual machines were running Linux Mint 18 (Papillon is not compatible with MacOS, only Linux and Windows). One virtual machine is the master server which stores the database and provides the API. This master server is also a client and calculates usage on itself. The second virtual machine has a client installed on it. This host runs the Papillon client software which calculates usage and sends this to the master. This setup mimics a real datacenter with two hosts (set up as two physical servers rather than VMs). An arbitrary amount of virtual machines could be set up on my device to simulate a larger datacenter but RAM limits me to two while the device still functions properly for other tasks. Once the application is complete, it will be run on an actual datacenter. The two 'servers' ran on my laptop for more than 2 months, periodically being turned off for various reasons.

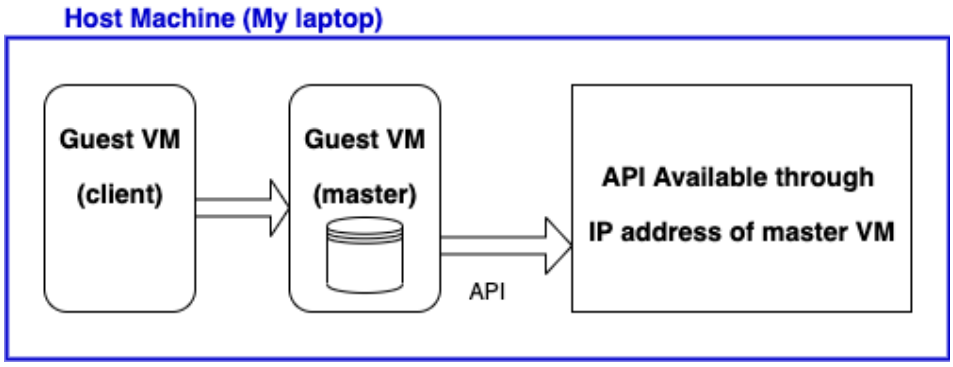


Figure 5.1: Dummy Datacenter consisting of two Virtual Machines

Figure 5.2 shows a diagram of the layout of the datacenter. Both hosts, the two virtual machines, are set up on a single rack. This is the only rack in the only floor of the datacenter.

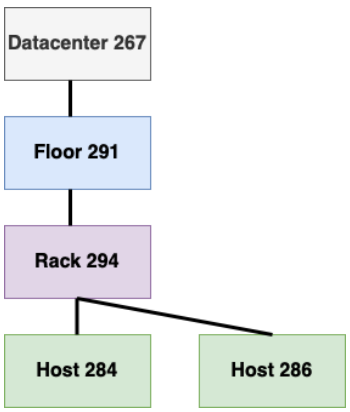


Figure 5.2: Layout of dummy datacenter (two hosts)

5.2 Measures

In order to identify which hardware in the datacenter is running efficiently and which is not, we need different metrics which cover the consumption of energy, the emission of carbon dioxide, and the operational costs of the datacenter. In order to get a comparable figure for each asset, the measures will be extrapolated over a 3 year period. This long period will allow the datacenter administrator to see the true cost of the servers over their lifetime.

1. Operational Consumption for 3 years: The energy consumption of the server extrapolated over a 3 year period. It is shown in kWh (kilowatt-hours). This metric is required for the TCO calculation. This figure is calculated by finding the average kilowatt-hour that the machine is consuming over the specified audit period and multiplying it by 3 years (3 years x 52 weeks x 7 days x 24 hours)

$$ops\_cons_3 = AVG(kWh) \times 3 \times 52 \times 7 \times 24$$

2. Total Cost of Ownership for 3 years: TCO amortises the initial cost of buying hardware plus the average power usage to estimate the total cost incurred over 3 years. This excludes expenses for floor space, cooling overheads, etc. The capital cost must be provided by the user as it is not specified in Papillon and hardware ranges so much in price that using a standard figure is not an option. ops_cons3 is the operational consumption for 3 years.

$$TCO_3 = capital + (energy\_cost \times ops\_cons_3)$$

3. Operational Cost for 3 years: An estimate of the cost accrued by the specific server over a 3 year period. This metric takes into account the PUE (Power Usage Effectiveness [9]) of your datacenter to give a more accurate figure. Energy cost is given by the user when configuring the datacenter. ops_cons3 is the operational consumption for 3 years and is multiplied by the PUE, that figure is then multiplied by the energy cost of 1 kWh (kilowatt-hour) in euro.

$$op\_cost_3 = (ops\_cons_3) \times PUE \times energy\_cost$$

4. Carbon Footprint for 3 years: Based on the energy consumed by the server. The result is the amount of CO2 produced by the server in kilograms. Carbon intensity is required from the user upon configuration. We multiply the estimated kilowatt-hours consumed over 3 years by the grams of CO2 produced.

$$carbon\_footprint_3 = (ops\_cons_3) \times carbon\_intensity$$

5. Average CPU Utilisation: The CPU utilisation in percentage for each minute is calculated by Papillon. To get the average for the audit period, we divide the sum by the number of responses.

$$av\_cpu = \frac{sum(cpu\_percentage)}{count(responses)}$$

6. Apparent Wastage for 3 years: Apparent wastage is the cost you have accrued by not utilising your server fully. For example, if your server only uses 10% of its capacity and is costing 100 euro a day to run, then 90% (90€) of your costs could be avoided with better utilisation of your hardware.

$$app\_waste\_cost_3 = (op\_cost_3) \times (1 - av\_cpu)$$

5.3 Testing

Due to the nature of nearly all the data shown on this site coming from an API, it means that a lot of methods can't be tested as it would require a connection to the Papillon master, which is not always available. However, there are still many other functions and methods which can be tested. I have adopted the 'if it can break, test it' rule to ensure the site is as reliable as possible. A test suite of 150+ tests is located in the folder 'webapp/tool/test/'. Acceptance testing was completed to ensure the functionality works correctly for the use case of the application. This was carried out by myself when the application was connected to the UCD datacenter for analysis.

5.3.1 Unit Tests

The unit tests are set up to test the services individually. Services are classes of functions that the views use in order to create, update, and delete the objects in the database through accessing API's and manipulating the data. Unit tests ensure the functionality of each method regardless of the state of the application. All tests follow the rule of 'zero, one, many' where applicable. This means that tests on a single method should be executed three times - once where the input is null/0, once where the input is 1, and once where the input is greater than 1. This should catch all possible scenarios. Unit tests are carried out on the models to ensure correct data types and attributes. Unit tests are also carried out on the forms (forms.py) which sanitise any input from the user. Unit tests are filed according to the service they provide. For example, 'asset_services.py' includes methods used in the 'Asset' tab and the tests for these are in 'tests/test_asset_services.py'.

5.3.2 Integration Tests

Integration testing ensures that all components of the application run together and do not cause any bugs or errors when interacting with each other. Integration testing is split into groups of related methods from one services file that relies in some way on methods from another services file. Unit tests alone are not sufficient for testing as they do not test the way in which the different modules interact.

5.3.3 Regression Testing

Regression testing has been incorporated into the existing unit and integration tests. Any bugs which have been fixed during development are included. This ensures that with every new update to the application, no previously known bugs or errors have resurfaced. Regression testing is run regularly and before any push is made to the GitLab master repository. This ensures consistency and maintainability in the software.

5.3.4 UI Testing

Selenium is an open-source automation tool for web-based testing [30]. The Python API allows you to connect with a browser through selenium. Using selenium in the Django test suite, it autonomously opens Google Chrome and clicks buttons/ fills fields/ submit forms as specified in the test. Testing in this way ensures that the GUI works as expected and speeds up development time as you do not need to click every possible button or form when you make a change to make

sure you haven't broken anything. The web application must be running and available at the specified URL. A 'chromedriver' file is located in the root of the web application to allow Selenium to open Google Chrome and execute the specified tests. The Selenium tests are skipped by Django when testing unless specified by removing the '@unittest.skip("skip UI tests")' at the top of the 'test_UI.py' class. This is because the application must be running and connected to a Papillon master server for the tests to pass.

5.4 Workflow

To ensure best practice testing is properly maintained and executed, I have devised a project workflow to follow (Figure 5.3). Following this plan at each step will result in a release which is extensively tested to an industry standard level.

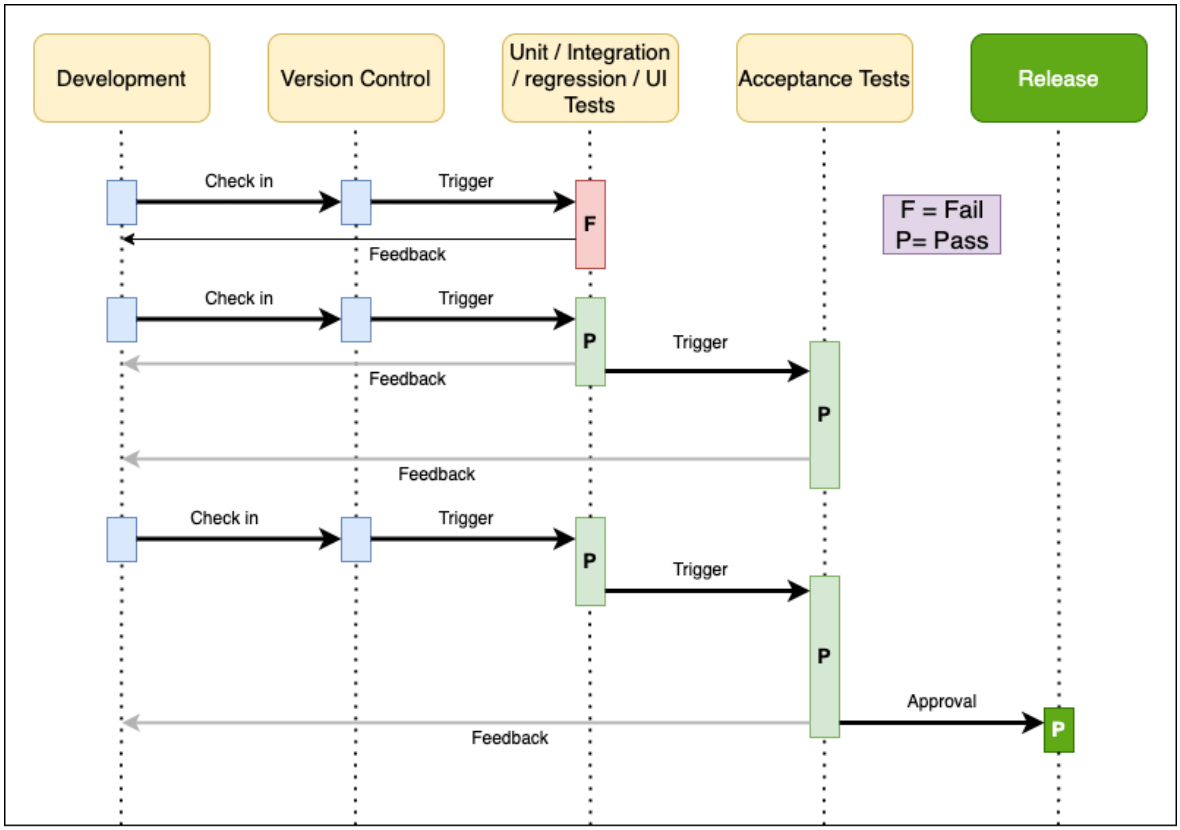


Figure 5.3: Project Workflow

Chapter 6: Design and Implementation

6.1 Architecture

The web application will consist of several tools that will each convey useful energy consumption information as well as graphics to show consumption visually. The four pages will be; Home, Assets, Budget, and TCO (Total Cost of Ownership). Each page will contain specific information and will contain forms that the user can submit to modify the web application database.

The back-end architecture is designed using Python Django[27]. The front-end is designed using HTML, JavaScript, and CSS. The front-end is completely original and no bootstraps were used. The deployment of the Django application is handled by Docker and Gunicorn using WSGI (Web Server Gateway Interface). The database is SQLite3. There is also support for a virtual environment to be set up when running the application and any required libraries are installed within (if you don't want to use docker). This avoids installing Python libraries globally on the machine which can cause incompatibilities.

Figure 6.1 shows a high level diagram of the system architecture.

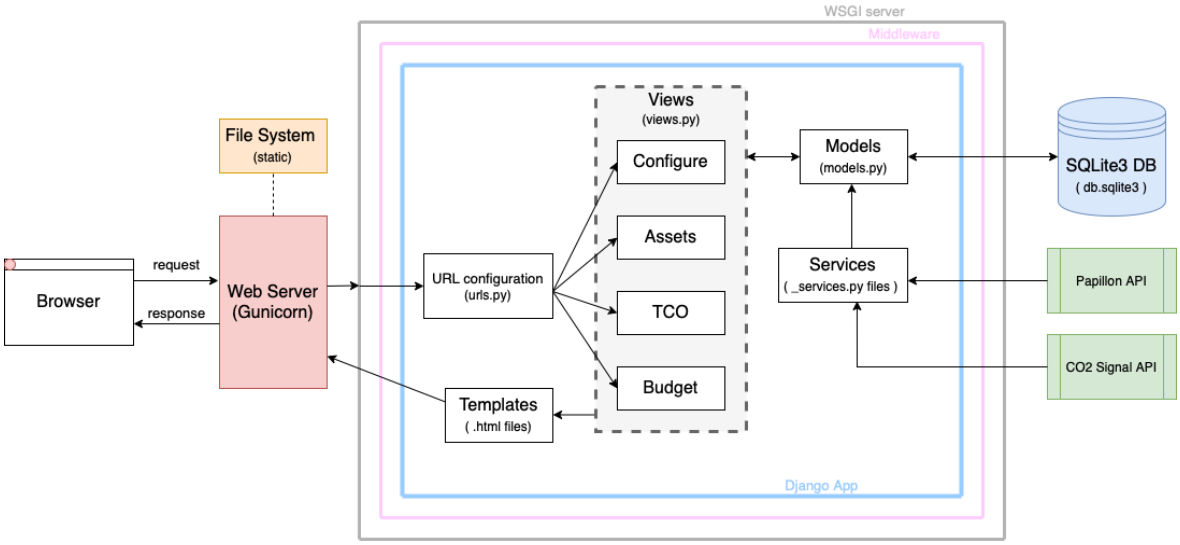


Figure 6.1: Web Application System Architecture

6.2 Database

Django provides support for SQLite3 by default. The application itself must have default properties which can be modified by the user but must remain constant once the application has been run and until the user decides to change them. These constants are; the IP address of the master server (input by the user), the name of the currently selected datacenter (selected by the user), and the upper and lower threshold for CPU usage if you want to be warned when they are too high or too

low. All configured datacenters are saved to the database, even if it is not your currently selected instance or on the master IP address. The only elements provided by the user are; current master IP address, currently selected instance, and the configurations table. Everything else is retrieved and calculated from the Papillon API.

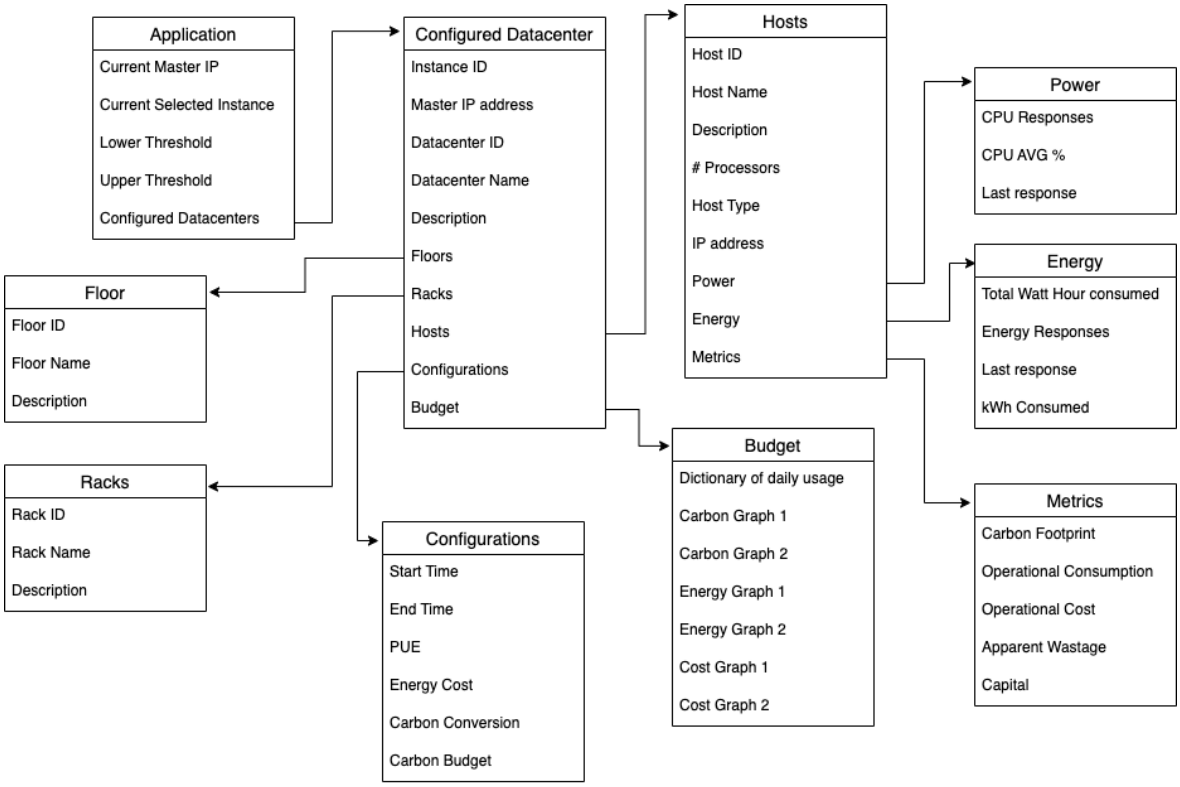


Figure 6.2: Web Application Database

- Configured datacenter: A configured datacenter is comprised of; an instance ID, a master IP address, a datacenter ID, a name, a description, any amount of floors, any amount of racks in each floor, any amount of hosts in each rack, configurations input by the user, and a budget.
- Floors and racks each include an ID, a name, and a description.
- Configurations: Set by the user when configuring a new datacenter. (See section 6.5.2)
- Hosts: Each host is a server. It can be virtualised on a piece of hardware along with others or can be an entire hardware server. It comprises; an ID, a name, a description, a number of processors, a host type, an IP address, power data, energy data, and metrics.
  - Power: Power stores the total CPU % usage for each minute. It also stores the number of responses. These can be used to find the average for the audit period. It also stores the last response time - which will be used for updating.
  - Energy: Energy stores the total watt-hours consumed by the host over the specified audit period. It also stores the number of responses, the last response, and the kWh consumed (kilowatt-hours)
  - Metrics: These metrics are calculated using the formulas in section 5.2.
- Budget: The budget model stores a dictionary of daily power usage to be graphed using matplotlib. Each graph is stored in the database encoded in base64.

The schema for the database is defined in models.py (webapp/tool/models.py). Each model maps to a single database table stored in db.sqlite3 (webapp/db.sqlite3).

6.3 Retrieval of Data

6.3.1 Initial Configuration

When a user navigates to the homepage, which you must do to configure a datacenter, the application uses the API to map out the structure of the datacenter as in shown Figure 4.1, and saves any changes to the database (if it's the first time, the whole datacenter will be saved - otherwise only updates will be saved). The master IP address is needed to find all datacenters hosted on the master server, the default master IP address is localhost and can be changed by the user at any time. Once these datacenters are found, another API call is used for each of them to get the layout of each datacenter. Hosts, racks, floors, and even datacenters can be added to Papillon running on a single IP address at any point in time, so this is done automatically (before configuring) to pick up on any changes.

The datacenter, floor, rack, and host objects have been saved to the database containing only descriptive values such as ID, name, and description. No power or energy data has been retrieved yet. When the user goes to configure a datacenter, the map of the datacenter is used to create API calls to the Papillon master. The variables needed are; Master IP address, Datacenter ID, Floor ID, Rack ID, Host ID, Start time, and End time. All of these variables except start and end time are retrieved by iterating over the tree and creating a URL for each host node using its values (host id) along with its parents values (floor id, rack id, datacenter id). The start time and end time are specified by the user. These calls must be made twice for every host in the datacenter - once for power (JSON response 4.3) and once for activity (JSON response 4.4). We can find the CPU utilisation using the activity data and we can find the amount of energy consumed using the power metrics (watt-hours).

The data retrieved for activity has a very fine granularity. We must add up all of the 'stat1' values and divide that by the number of responses made by the server. This will calculate the average CPU utilisation over the specified period. We can update the existing host object with this value. The 'host' variable in the code snippet below is the Django model for the specified host which was used in the API call. This code snippet is found in 'webapp/tool/services/asset_services.py'.

Code Listing 6.1: Calculating CPU utilisation from raw data and updating database

```
cpu_total = 0
cpu_count = 0
if isinstance(data['activity'], list):
    last_response = data['activity'][-1:][0]['timeStamp']
    for activity in data['activity']:
        cpu_total += float(activity['stat1'])
        cpu_count += 1
    avg_cpu = cpu_total/cpu_count
else:
    last_response = data['activity']['timeStamp']
    cpu_total = float(data['activity']['stat1'])
    cpu_count = 1
    avg_cpu = cpu_total/cpu_count

host.update(cpu_usage=avg_cpu, cpu_responses=cpu_count, total_cpu=cpu_total)
```

Using a similar approach for the power data, we can find the amount of watt-hours consumed by the specific host as well as the number of responses and the last timestamp we receive. Using the watt-hours consumed, we can calculate the kilowatt-hours consumed and the average watt-hour

consumed per minute. Using these values, along with the PUE, energy cost, and carbon intensity as provided by the user, we can calculate the TCO measures as described in section 5.2 (TCO).

The figures for the measures are for the entire audit period. The budget figures, however, must be calculated at a daily level for both each server and the entire datacenter in order to produce graphs and visualisations. Below is a small section of the logic that shows how we divide the watt-hours consumed into days (86400 seconds in one day). The start time in this case will always be 00:00:00. The daily consumption for all hosts is stored in a pandas DataFrame. Django models have static fields and since there is an arbitrary amount of hosts in each datacenter, it is not possible to store a daily figure for each host in a model in its raw state. For this reason, I elected to convert the DataFrame to a Python dictionary object, encode this dictionary as a JSON object, and store that in the model.

Code Listing 6.2: Calculating watt-hours consumed on a daily bases

```
for power in data['power']:
    if int(power['timeStamp']) >= start:
        if int(power['timeStamp']) >= start+86400:
            start+=86400
            energy[start].append(float(power['power']))
            continue
        energy[start].append(float(power['power']))
```

6.3.2 Updating an Existing Datacenter

The technique behind updating an existing datacenter is simple yet difficult to implement. For every host in the datacenter, and for both power and activity, we find the last response time. This response time becomes the new start time for the period and the end time becomes the current time (for live audit periods) or remains the date specified by the user. Now we have the new data for this period, not including any data which has been processed already. Rather than updating the model with this data, we must append it onto the existing data and then save that. Below is a snippet for updating the activity of a host. The method for updating the power data is similar.

Code Listing 6.3: Updating CPU Usage of an existing host

```
if isinstance(data2['activity'], list):
    last_response = data2['activity'][-1:][0]['timeStamp']
    for activity in data2['activity']:
        cpu_total += float(activity['stat1'])
        cpu_count += 1
    updated_responses = host.values().get()['cpu_responses'] + cpu_count
    updated_cpu_total = host.values().get()['total_cpu'] + cpu_total
    updated_avg_cpu = updated_cpu_total/updated_responses

else:
    last_response = data2['activity']['timeStamp']
    updated_responses = host.values().get()['cpu_responses'] + 1
    updated_cpu_total = host.values().get()['total_cpu'] + float(data2['
                                            activity']['stat1'])
    updated_avg_cpu = updated_cpu_total/updated_responses

host.update(cpu_usage=updated_avg_cpu, cpu_responses=updated_responses,
                total_cpu=updated_cpu_total,
                cpu_last_response=data2['activity']['timeStamp'])
```

Updating the budget information is not as straightforward. The data already stored in the model is encoded in a JSON object and so needs to be decoded first. Once decoded, we can append the data for the first day of the updating period to the last day of the existing data, and add any subsequent days if applicable.

Code Listing 6.4: Decoding the daily energy consumption from the budget model

```
budgeted = budget.values().get()['energy_dict']
decoded_data = json.loads(budgeted)
```

6.4 URLs

There are four URLs that make up the entire site. These URLs are specified in the urls.py file (webapp/tool/urls.py). These four URLs are:

1. tool : Homepage - Contains forms for configuring and updating datacenters
2. tool/assets : Assets tab - Contains an interactive map of the datacenter
3. tool/budget : Budget tab - Contains graphs of cumulative consumption
4. tool/tco : TCO tab - Contains measures outlined in section 5.2 (Measures)

Code Listing 6.5: urls.py - Setting URLs with corresponding view from views.py

```
urlpatterns = [
    path('', views.configure, name = "configure"),
    path('budget/',views.budget, name = "budget"),
    path('tco/',views.tco, name = "tco"),
    path('assets/',views.assets, name = "assets"),
]
```

6.5 Homepage

The homepage is where the configured datacenters and available datacenters will be shown to the user (Figure 6.3). The user will also be able to: change the master IP address, configure a new datacenter, update an existing datacenter, delete an existing datacenter, and view live CO2 figures for Ireland. Originally, the plan was to have a homepage showing configured datacenters and a separate configure page where you could make the below changes using forms. In the philosophy of making things simple for the user, it was decided to include all of this functionality in the homepage.

The first table includes any configured datacenters, showing the values that the user has inputted when configuring a datacenter. There is no limit to how many datacenters you can configure. The highlighted one (in grey) is the currently selected datacenter, which is also shown on the top right-hand side of all pages. The second table shows the available datacenters on the Papillon master. This is necessary as it's possible to set up two different datacenters on the same master server.

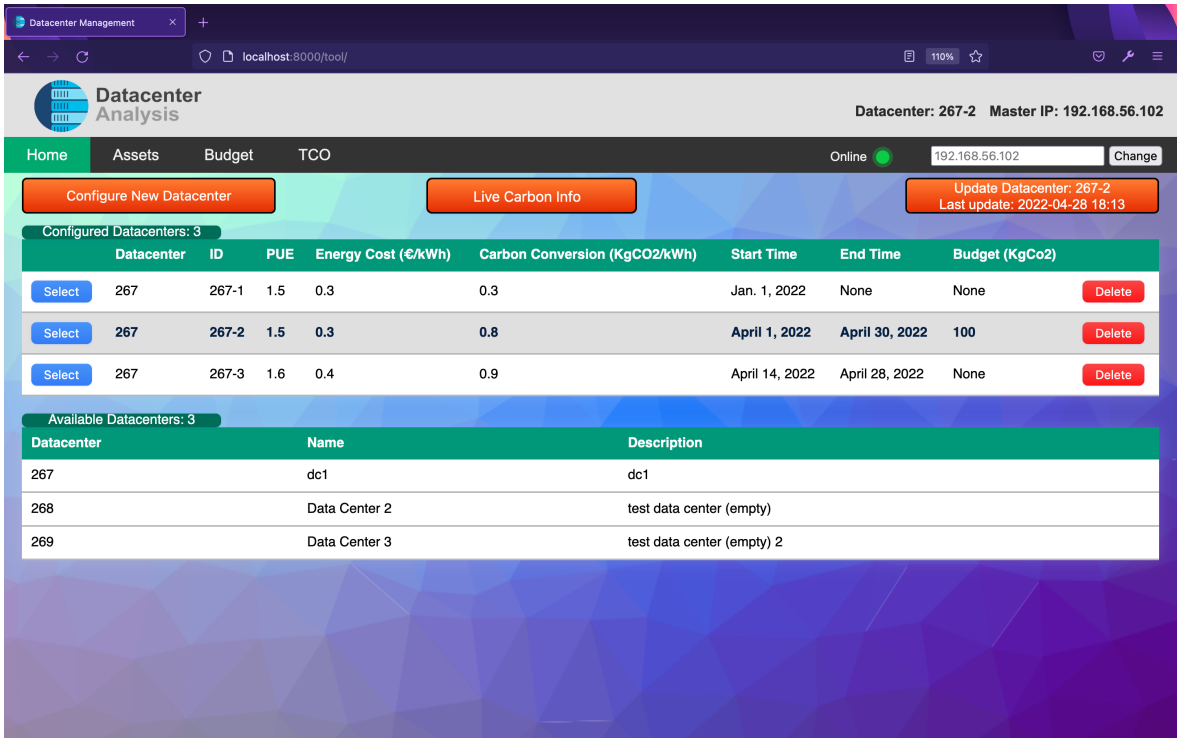


Figure 6.3: Homepage of Web Application

6.5.1 Master IP

The IP address of the master node of Papillon can be changed here (Figure 6.4). By default it is localhost. While on the same network as the master, inputting the IP address will set it as the new default. The IP address can be changed in the case that you have two or more masters on the same network. Changing the master IP address will not remove any configured datacenters, you can simply change back and your configured datacenters will still be saved. This master IP address will be used in API calls to Papillon to get information on the servers. If the master IP is reachable and contains a Papillon master on it, there will be a green light showing that it is online. If the IP address entered either does not have a Papillon master running on it or the master server is not responding there will be a red light indicating offline.

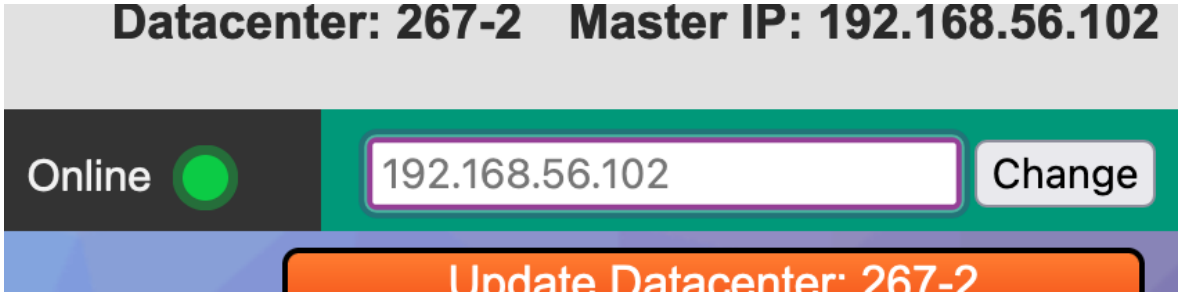


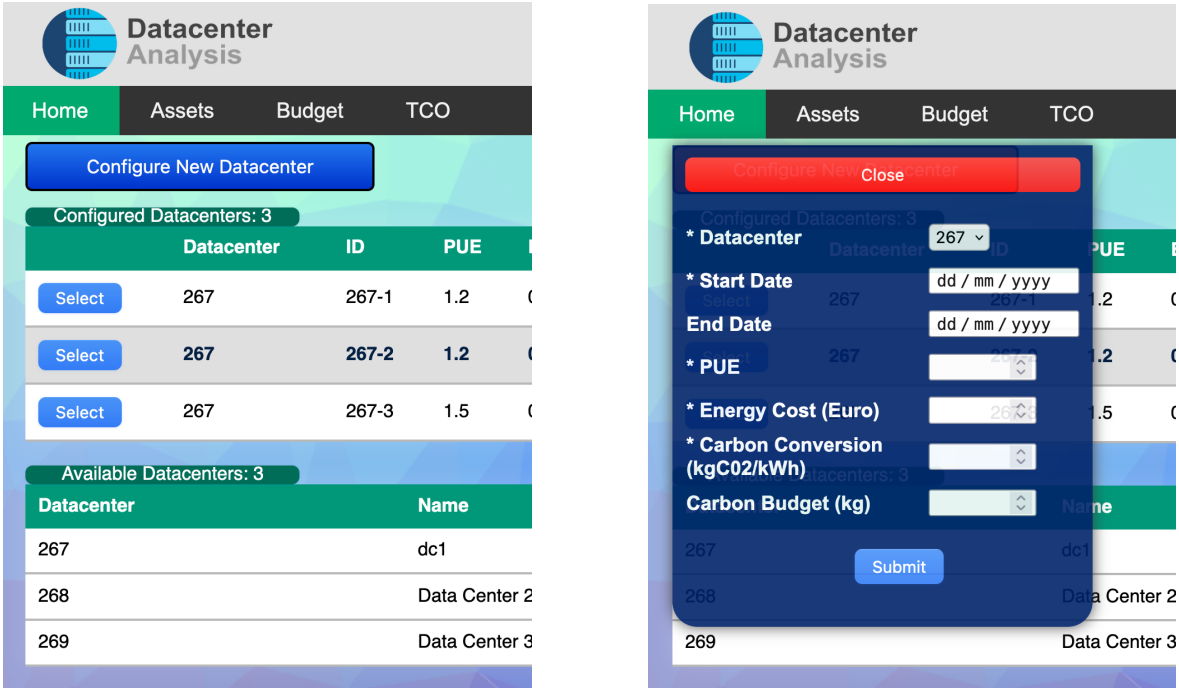
Figure 6.4: Changing the IP Address of the Master Server

6.5.2 Configuration

To configure a new datacenter, the user must click the 'Configure New Datacenter' button which will open a form (Figure 6.5). Certain properties of your datacenter must be specified before energy information can be retrieved for it - they must be entered by the administrator. These

properties include:

1. Datacenter: You can have multiple datacenters on one Papillon master so you must select which datacenter you want to focus on. This is a drop-down field and is whitelisted to only include datacenters available on the current master IP.
2. Start Date: The date that you want the audit period to begin. This can be any date and is not confined to the date that Papillon first started running on the datacenter assets. This is a datetime field.
3. End Date: This property is not required, but you can include it if you want to create a fixed audit period. If the end date is not specified then the audit period end date is the current time and will update automatically. End date must be after the start date. This is a datetime field.
4. PUE: The Power Usage Effectiveness [9] of your datacenter must be specified. This will be used in calculations to quantify the carbon footprint and operational costs of your datacenter. The web application will only accept floats from 1 to 3 and in steps of 0.01.
5. Energy Cost: The price of energy in kWh (kilowatt-hour) must be specified for calculations of operational costs. The price is in euro. Inputting 0.3 here will mean that 1 kWh of energy costs €0.30. The price of electricity depends on the provider and can fluctuate so it is up to the administrator to input this.
6. Carbon Intensity: Carbon intensity is the number of grams of carbon dioxide (C02) that is produced per kWh produced. It is required to calculate the carbon footprint of your assets.
7. Carbon Budget: This field is optional. You can specify your carbon budget (kgCo2) for the period and the user will be able to see their consumption in relation to their budget.



(a) Configure New Datacenter Button

(b) Configure New Datacenter Form

Figure 6.5: Configuring a New Datacenter

6.5.3 Select / Delete

You can configure multiple instances of datacenters. Selecting a datacenter on the table will change the active datacenter. Deleting a datacenter on the table will delete all information relating to that configured datacenter from the applications database.

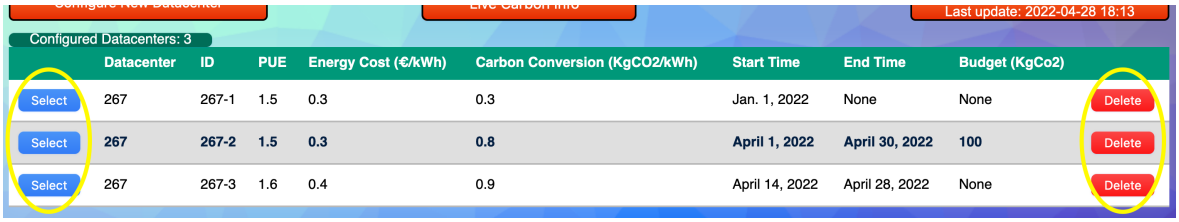


Figure 6.6: Selecting and Deleting a Datacenter

6.5.4 Live Carbon Info

CO2 Signal provides a free, limited API that returns some live CO2 metrics for a selected country [23]. The API returns: carbon intensity, fossil fuel percentage, and last update time, all of which will be shown on the homepage. Hovering over the 'Live Carbon Info' button will show a tooltip containing this information.

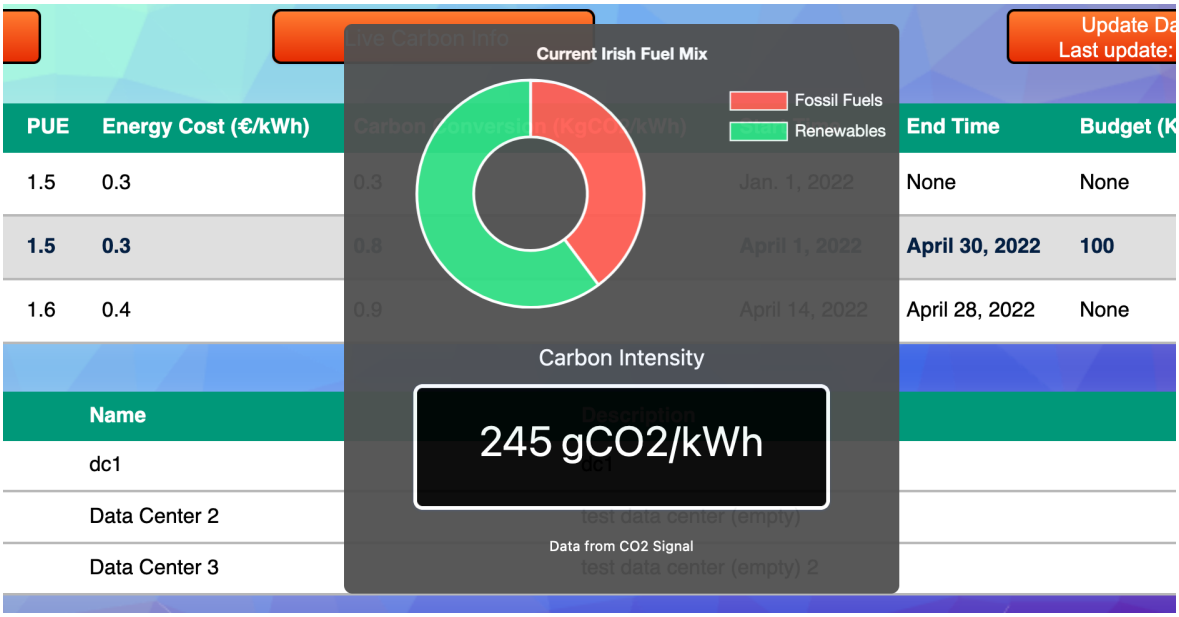


Figure 6.7: Live Carbon Information (Ireland)

6.5.5 Update Datacenter

When you configure a datacenter, all relevant information is retrieved from the Papillon API and saved to the database to allow for exploring the application without loading times, which can sometimes be quite long depending on the amount of data stored on the master server. To update your existing datacenter, you simply need to press a button on the homepage which will send new API requests that will append to the existing database (Figure 6.8). The update will take significantly less time than the original configuration if the period to be updated is shorter than the existing audit period. If you have configured a datacenter for an audit period that is no longer

active, the button will not be available as there is no new data to update it. For example, if your audit period is January 2022 - June 2022 and you configured this in July 2022.

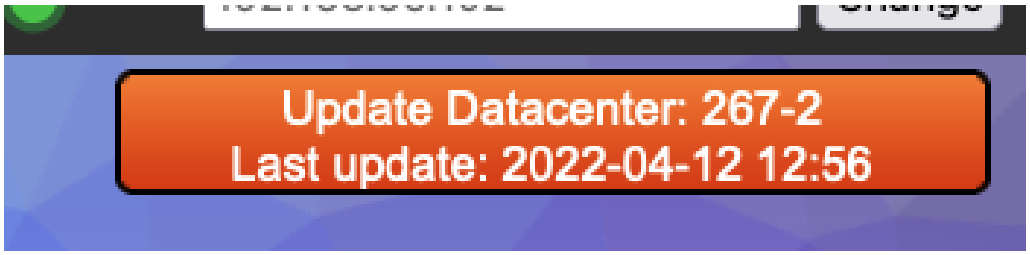


Figure 6.8: Button for updating selected datacenter

6.6 TCO (Total Cost of Ownership)

The TCO tab will allow the user to calculate the Total Cost of Ownership of their servers along with other metrics (Figure 6.9). The TCO tab includes all servers on one page, irregardless of their respective location in terms of floors and racks. The relative locations of hardware can be found by looking at the Floor ID and Rack ID or by navigating to the Assets tab. Some metrics are extrapolated over a 3 year period to make a comparable estimate. For example, if you know the average kWh consumed per minute of a server that has been running for 3 weeks, you can leverage that to estimate what it would be for 3 years by scaling it.

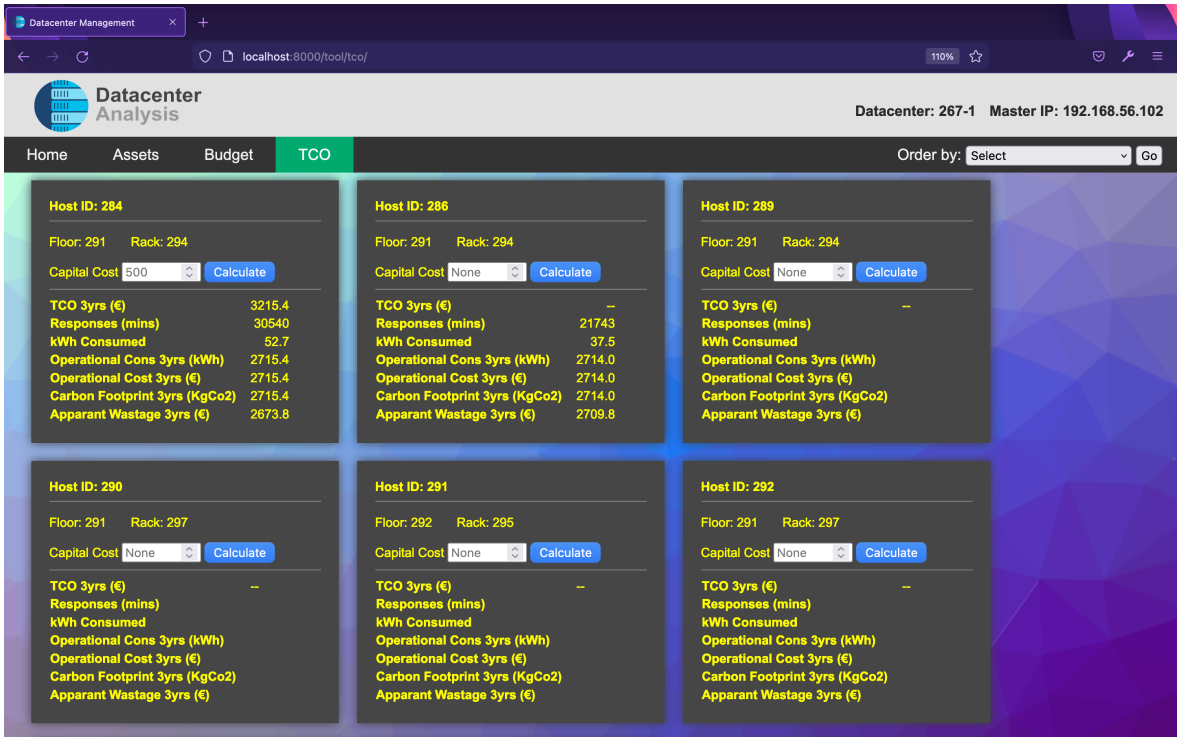
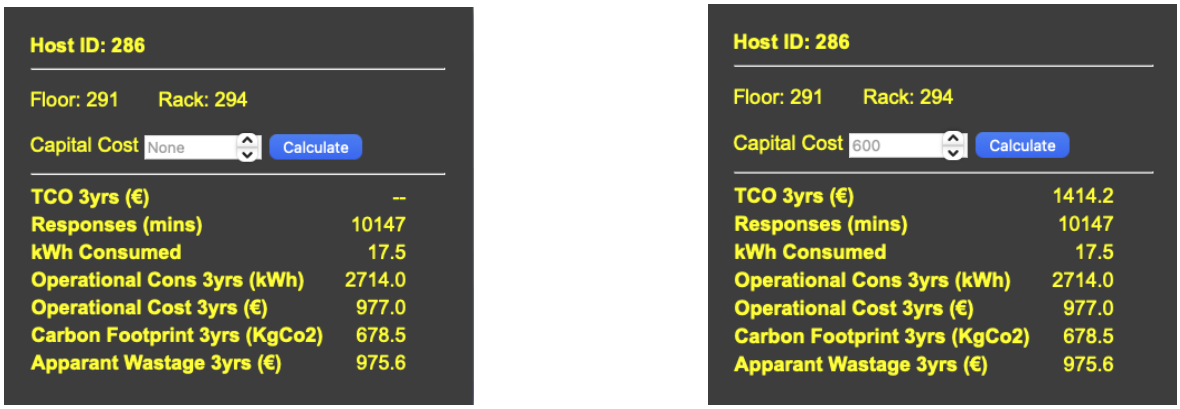


Figure 6.9: TCO Tab

For the total cost of ownership, the user will need to input the capital cost of the hardware. Using this information along with the energy usage and energy cost, the application calculates the estimated TCO for 3 years. The administrator can then identify the most expensive servers to run. Other metrics, which do not require the capital cost, are shown before capital is inputted and include; the estimated carbon footprint for 3 years, the estimated total operational cost for 3

years, the estimated total consumption for 3 years, the apparent wastage cost for 3 years, and the average CPU utilisation over the audit period. The formulas for these calculations are outlined in section 5.2.



(a) TCO before capital cost is inputted

(b) TCO after specifying the capital cost

Figure 6.10: TCO for one server

The servers are, by default, ordered by their Host ID in ascending order, as shown in Figure 6.9. The option to order them by any of the metrics is provided (Figure 6.11). For example, if you wanted to see which hosts were generating the most amount of carbon, you can order them by their estimated carbon footprint in descending order. This is to make it easy for the administrator to compare the different hosts using whatever metric is most important to them at any given time.

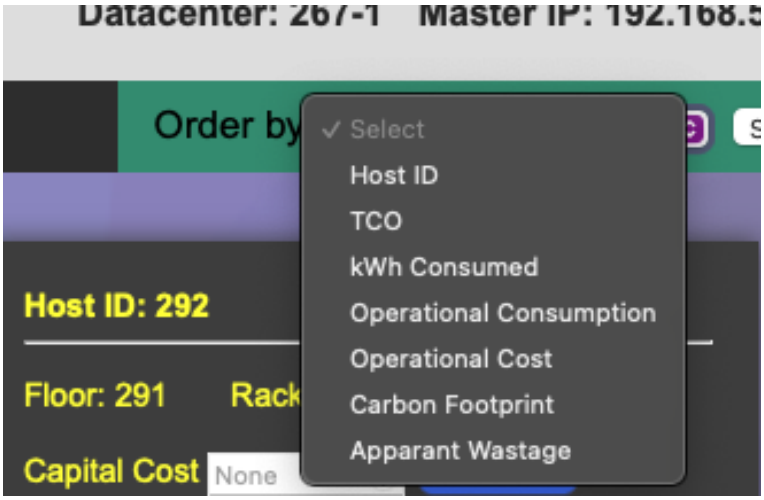


Figure 6.11: TCO Tab

6.7 Assets

This tab will allow the user to view all of the assets available in their configured datacenter (Figure 6.12). It acts as an interactive visual map of the datacenter as it can be hard to remember locations just based on the floor, rack, and host ID. It is a single URL that uses JavaScript to view assets in a tree-like structure, expanding and closing as required. The depth of the tree is 4 as shown in Figure 4.1. The first node, datacenter, is hidden as the datacenter is already selected. The user can expand nodes to show only the desired floors/ racks/ hosts. When installing Papillon onto a

group of servers, the floors, and racks can be specified, meaning you can organise your hosts into separate logical groups, whether that be the users of specific hosts or the applications that are being run. In many organisations, groups of servers will be run by teams or individuals. This will allow the administrator to analyse usage at a higher level rather than just single hosts.

Code Listing 6.6: JavaScript for interactive tree showing assets

```
var toggler = document.getElementsByClassName("box");
var i;

for (i = 0; i < toggler.length; i++) {
toggler[i].addEventListener("click", function() {
  this.parentElement.querySelector(".nested").classList.toggle("active");
  this.classList.toggle("check-box");
});
}
```

While viewing the hosts, the user will be shown the CPU % usage of each asset. They will be able to set thresholds for a traffic light system showing how critical the level of usage is. The default will be set to: green: > 30, amber: >= 15 & <= 30, red: < 15. If the host has Papillon installed but is not returning any data because it is powered off, the CPU % will be empty and the box will be grey. In the figure below, the thresholds has been set to green: > 1.5, amber: 1 - 1.5%, red: < 1% for demonstration. This datacenter only contains two idle VMs for development leading to the low CPU usage. The grey hosts are servers which have been added to Papillon but do not actually exist and so are not sending data to the master.

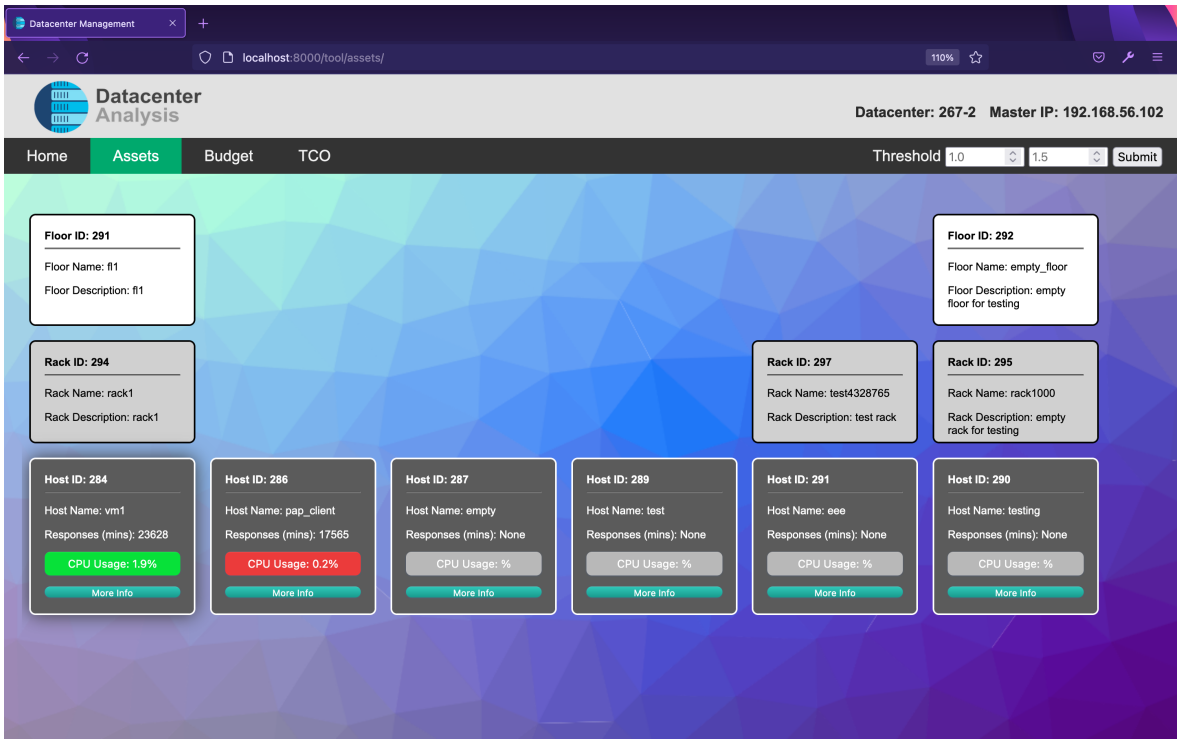


Figure 6.12: Assets Tab

When hovering your cursor over the 'More info' button (Figure 6.13) for each host, additional information will be shown, including; the host description, the host type, the number of processors, and the IP address.

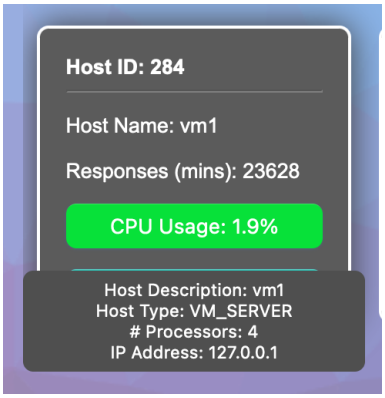


Figure 6.13: Additional information of host

6.8 Budget

The budget tab will contain cumulative graphs of the carbon consumption (kgCo2), energy consumption (kWh), and operational costs (€) (Figure 6.14). This page is a single URL and JavaScript is used to open tabs for viewing each element. Graphs for each of these will show both total consumption and also consumption of each host individually. If you have set a budget for your datacenter, this will be shown on the carbon graph making it easy for the user to identify the trend and if any changes will need to be made to keep below their budget for the period.

Figure 6.14: Budget graphs

If the end date has been set by the user, the audit period will be set and unchangeable - this is reflected in the graphs in Figure 6.15 below. If the end date has not been set, the audit period will be dynamic. When you update your existing datacenter, the audit period end date will become the time in which you have updated. If a budget has been set, it will be shown as a horizontal red line on the y-axis as shown in Figure 6.14 above and Figure 6.15 below.

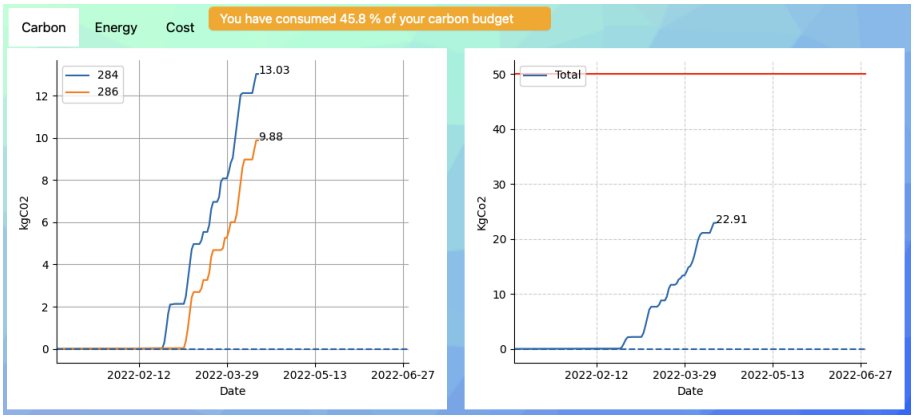


Figure 6.15: Budget with set audit period (January 1st - June 30th)

I had attempted multiple times to create interactive graphs for the live carbon emissions, energy usage, and operational costs in the Budget tab. I was unsuccessful in part due to my lack of experience with chart.js. I had to settle for graphs created in matplotlib which do communicate the information well but do not cater for zooming or other interactions.

6.9 Invalid User Input

Any input from the user is processed in Django to make sure it is valid. This sanitisation happens in forms.py (webapp/tool/forms.py). If any input is classified as incorrect, the cause of the error will be shown on the webpage. Using forms, we can generate error-specific messages and send them to the HTML templates dynamically. All input from the user is checked extensively. For example, if you try to configure a datacenter with an end that is before the start date, you will be shown this error in Figure 6.16. This sanitisation of the data ensures that no invalid data is saved to the database and prevents the web application from crashing.

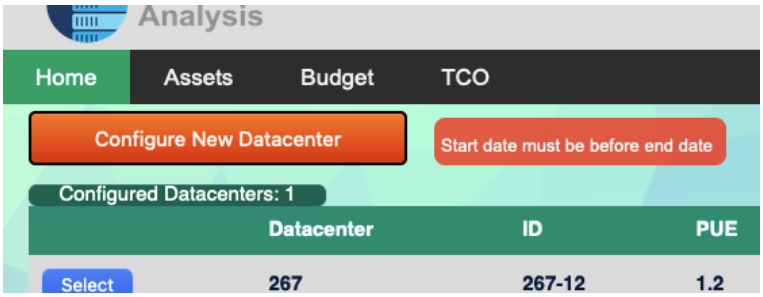


Figure 6.16: Invalid input error

6.10 Deployment

This application must be able to run on all types of hardware. The application can be run on the master server alongside Papillon or else on any other device on the local network. This is the reason why the IP address of the master is customisable, you are not limited in where or how you run the application. The only system requirement is Python 3.7+. Lower versions of Python may

work but are untested. Python 3.7 was released in 2016 and most hardware will be able to run it unless it is dated.

The web application can be run in three different ways, depending on the use case and your hardware/ software.

1. Docker: A docker container has been implemented which uses Gunicorn to run the web application on port 8000. Gunicorn is a Python WSGI server which has no dependencies. It is installed in the container and is used to run the server.

   ```
   $ docker compose build
   $ docker compose up
   ```

2. Python Virtual Environment: A bash script (startup.sh), when run, will create and activate a Python virtual environment that installs all required libraries to the virtual environment using the 'requirements.txt' file. Fabric is a Python library and command line tool which allows for custom application deployment and admin tasks to be run. Fabric allows for the use of SSH when deploying which may be needed in some edge cases. I have implemented four Fabric tasks which can be run in the virtual environment.

   (a) fab setup: Collects all static files and creates a directory rather than Django serving these files. This does not need to be run as this directory already exists. In the case that a static file is changed for whatever reason in the future, this command will update it and in turn, update the GUI as specified.

   (b) fab run: Runs the WSGI server using Gunicorn on port 8000 as a daemon. The PID of this process is stored in pid.txt.

   (c) fab kill: Stops the server running using the PID that was previously saved to pid.txt.

   (d) fab reset: Completely wipes the application database in the case that the user wants a fresh application

3. Django debug mode: Using Django debug mode, you will see useful outputs in the command line when running the application.

   ```
   $ python3 manage.py runserver
   ```

6.11 Changes from Original Spec

6.11.1 Automatic Updating

Originally, my hope was to build the application to update from the Papillon API whenever you used the application. For example, whenever you click into the 'Assets' tab, the API was called and the CPU usage % figures were updated. Likewise when you navigate to 'TCO' or 'Budget', the API would be automatically called to get any new data and you never would have to click an update button. This was implemented fully and worked well on the dummy datacenter I had set up on my laptop.

The problem with this arose when testing began on the UCD datacenter. Papillon has been running on the servers in UCD for multiple years now and because of this, calling the API can be very slow. Each time you call the API, SQL queries the database, which contains minute by minute data for 8 servers over multiple years. This results in a relatively slow response and makes the usability of

the application not up to standard. With such a large database, you need to wait a few minutes when clicking into each tab, even just to update one minute of data.

To overcome this problem it was decided to add an update button that gathered all new information from the API and stored it in the database. Navigating the application is now effectively instant unless you want to configure a new datacenter or update an existing one.

6.11.2 Rolling Carbon Intensity

One advanced objective was to provide a rolling carbon intensity to be used in the carbon footprint calculations. This would give a more accurate estimate rather than taking an average over the audit period. Using daily carbon intensity figures would also lead to a much more informative carbon emission graph which would show periods of days/ weeks/ months where carbon output was lower for whatever reason. Unfortunately, this was not possible to implement for two reasons.

1. Time constraint: Implementation of this feature would require a good portion of time as it would involve integrating the database with daily CO2 intensity figures. This integration would result in a redesign of the Budget and Metrics database tables.
2. Data constraint: CO2 signal is an excellent source for getting the current carbon intensity in Ireland. However, it is limited and will lead to problems if the application is being used often. There are other, paid API's but I was not granted a key for educational purposes.

Chapter 7: Results and Analysis

7.1 Web Application Speed

The speed of the application is important to ensure a seamless and on-demand service for the datacenter administrator. All data is cached (except live CO2 figures) which means once you have a datacenter configured, traversing the different tools in the application is essentially instant. There are two instances in which you will have to wait for the application to consume data and save it. These are; when you configure a new datacenter, and when you update an existing datacenter. To measure the speed, I used the Python time library to calculate the time before the methods are called and then found the time elapsed after. For long audit periods with many hosts, the loading time can be a few minutes. For this reason, I added a loading visual so that the user is assured that the application is running correctly (Figure 7.1).

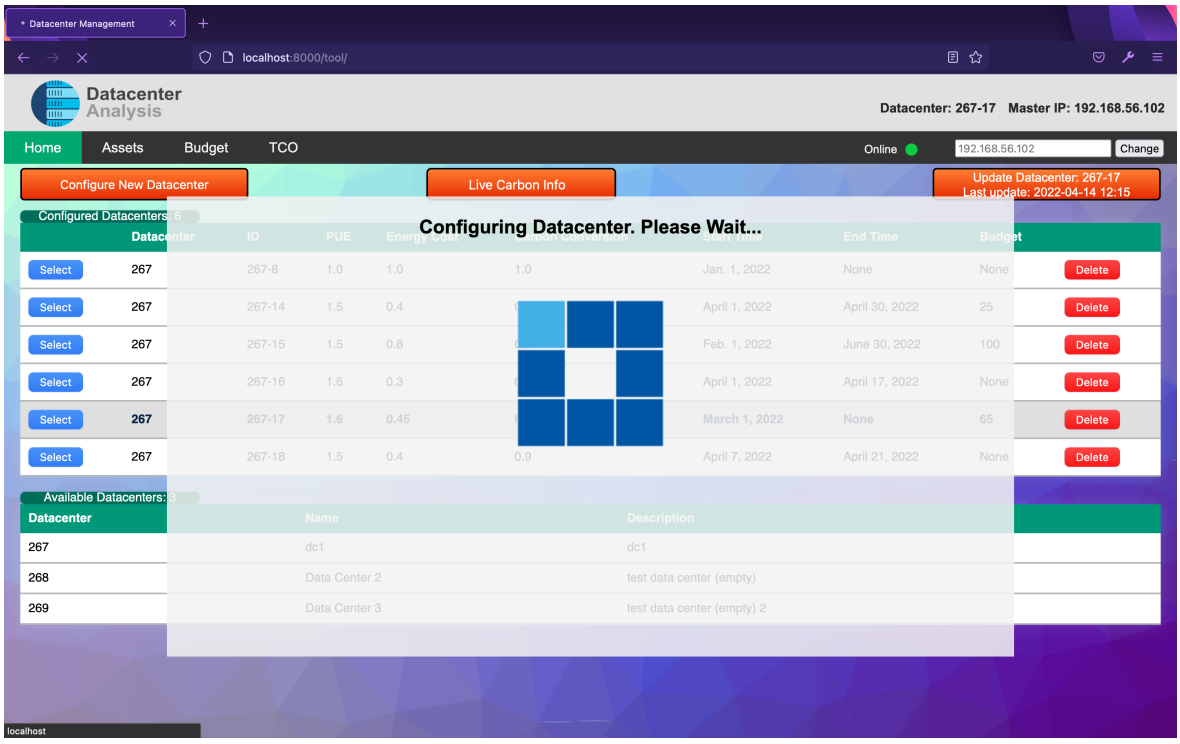


Figure 7.1: Loading screen when configuring/updating a datacenter

- **Initial Configuration**

  For 1000 hours of data for 1 host, the time taken to configure was 3.807 seconds. It should be noted that for every hour of data, two API endpoints are being hit (power and activity). So 1000 hours of data for 1 host is 2000 hours of responses (120,000 responses) for that host. This may seem like a very quick loading time, and it is, but this is only a 42 day audit period for one host. 2.145 seconds of this time is just getting a response from the API as Papillon needs to query its own database to get the data as well. For 1000 hours of data for 2 hosts, the time elapsed was 6.751 seconds - with 4.093 seconds to get the responses.

- **Updating Datacenter**

  To update the datacenter with exactly 1 days worth of new data for 1 host takes 2.297 seconds. The reason it takes quite a relatively large amount of time is because it still has to hit all the endpoints which takes time as Papillon must query its database. The real speed advantage of updating will be shown when updating a large audit period with many hosts. For example, if you have a 6 month period for 10 hosts saved and you update after 1 day.

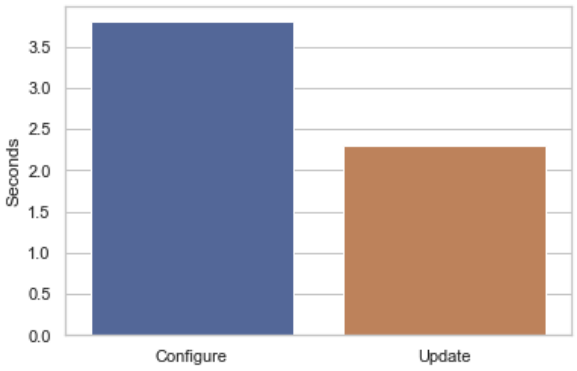


Figure 7.2: Speed of configure vs update for one host with 1000 hours of data

7.2 Practical Use in Real World Datacenter

The web application was installed and run in UCD on a datacenter that has been running Papillon and gathering data for years. The hope is that we can use the web application to help improve the datacenters efficiency. By using the three tools available; Assets, Budget, and TCO, we can identify any possible ways to decrease the carbon emissions and operational costs of the datacenter.

Below is the Homepage of the application running in UCD (Figure 7.3). The master server is located at 192.168.0.76. There is only one datacenter set up on Papillon and only one datacenter configured on the application. For the month of March, there has been a budget set of 100 kilograms of CO2 emitted.

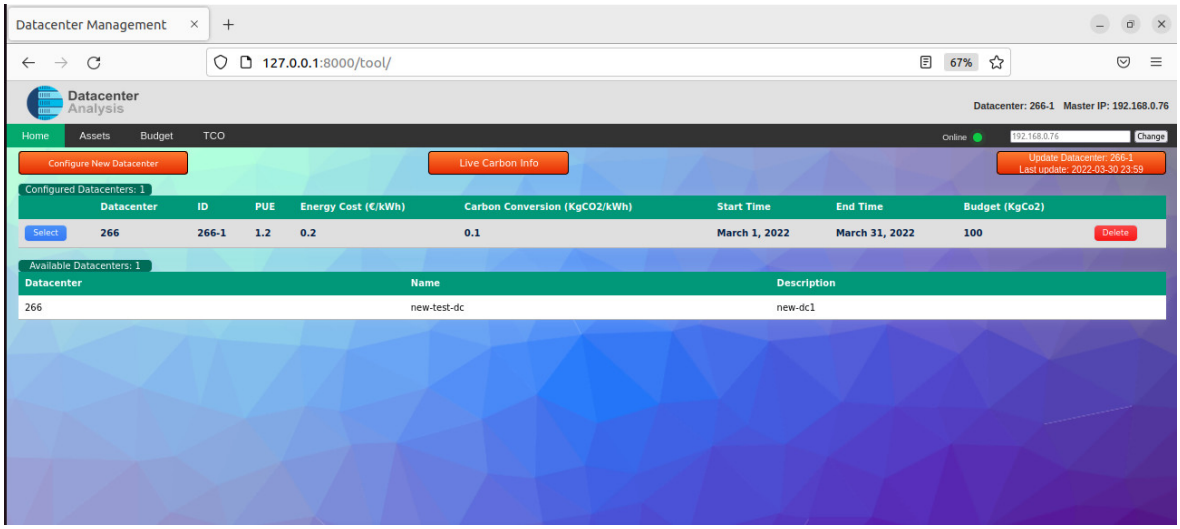


Figure 7.3: Configured UCD datacenter

7.2.1 Utilisation of Assets

Using the Assets tab, we can identify what hardware (if any) is being underutilised. We decided to use the default thresholds for CPU usage. Below 15% usage will be red, between 15%-30% is amber, and 30%+ is green. This is the traffic light system that will show the least efficient servers visually.

The UCD datacenter contains only one floor with two racks. In rack 293 (rack1) we have 5 hosts connected while in rack 294 (rack2) we have only 3. The tree in Figure 7.4 has been fully extended and all hosts in the datacenter are currently being shown. Four of the eight hosts in this datacenter are being underutilised in terms of CPU usage potential - 283, 287, 289, 290. Three hosts are being utilised efficiently - 284, 285, 288. One host sits in the middle at 23.1% usage (286), this could be improved but is not urgent.

By identifying which hosts are being underutilised, it is now up to the datacenter administrator to take action. By aiming to have all hosts in the green, the carbon emissions and costs can be lowered. For example, running 10 servers at 5% will consume a lot of energy. Virtualisation can be used to share resources and effectively have 2 or more hosts on one machine. 5 hosts running at 30% capacity is still a significant decrease in wasted CPU potential.

Any hosts using a very low CPU % should be investigated. It may come to light that the server is not performing any productive tasks at all. This is a huge problem in datacenters with many servers. If all tasks are stopped on a server and it's not switched off after, the server is sitting idle and is still consuming valuable energy. This can often happen through poor management and a lack of communication. When there is no system like the application implemented in this report in place, an administrator would have to connect to each server individually and check what is running. This may not be too large of a task for a small datacenter with only a few servers but as the amount of hardware grows, so does the time to analyse. It would take a lot of effort to estimate the CPU usage which would still be very inaccurate unless you fitted the server with a physical hardware meter.

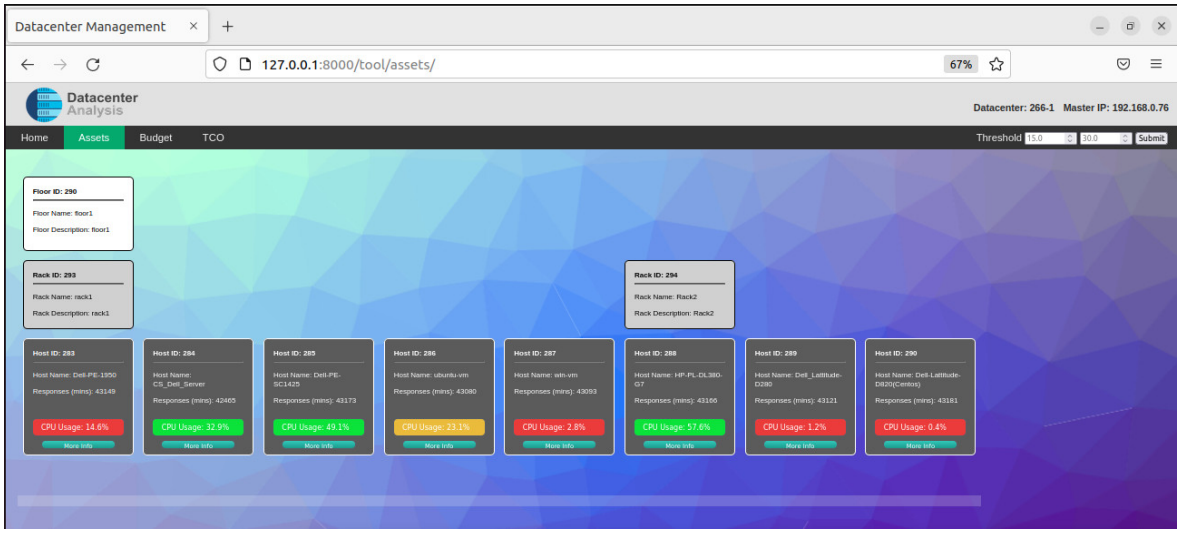


Figure 7.4: UCD Datacenter Assets Tab

7.2.2 Main causes of carbon emissions and costs

Using the TCO tab, we can compare the different metrics of the servers to accurately identify which servers are the worst offenders in terms of; carbon footprint, electricity consumption, operational costs, apparent wastage cost, and total cost of ownership. The calculations for these metrics are provided in section 5.2 (Measures). The capital cost of each server has been inputted by the administrator and ranges from €200 to €1000.

It is possible to order the hosts in descending order of whichever metrics you like. For this example, I have left it ordered by host ID as we will be looking at all the metrics together (Figure 7.5).

1. Energy Consumption: Host 285 (Dell-PE-SC1425) consumes the most amount of energy. This host consumes nearly 4.5 times more energy than host 290 (Dell-Latitude-D820(Centos)).
2. Carbon Footprint: Host 285 (Dell-PE-SC1425) has the highest carbon footprint. Host 283 and 284 also have high carbon footprints (above 500 KgCO2 over 3 years).
3. Operational Costs: Host 285 (Dell-PE-SC1425) has the highest operation cost over 3 years.
4. Apparent Wastage: Host 283 (Dell-PE-1950) has the largest apparent wastage over a 3 year period.
5. TCO: Host 285 (Dell-PE-SC1425) has the highest Total Cost of Ownership over 3 years at 1287.3.

Host 285 has the highest carbon footprint, consumes the most amount of energy, and has the highest operational cost. However, it's apparent wastage is relatively low compared to host 283. This is because, with host 283, it is consuming quite a lot of energy (5197 kWh over 3 years) but, if we look back at the Assets tab, we see that it is only utilising 14.6% of its potential CPU usage. This is a very inefficient server. Host 285, on the other hand, is utilising 49.1% of it's CPU potential.

The hosts in rack 294 (hosts 288, 289, 290) provide another example of efficient vs inefficient servers. Host 288 consumes twice as much energy as hosts 289 and 290 but its apparent wastage is lower than the other two. Host 288 is utilising 57.6% of it's CPU while the other two are both utilising less than 2%.

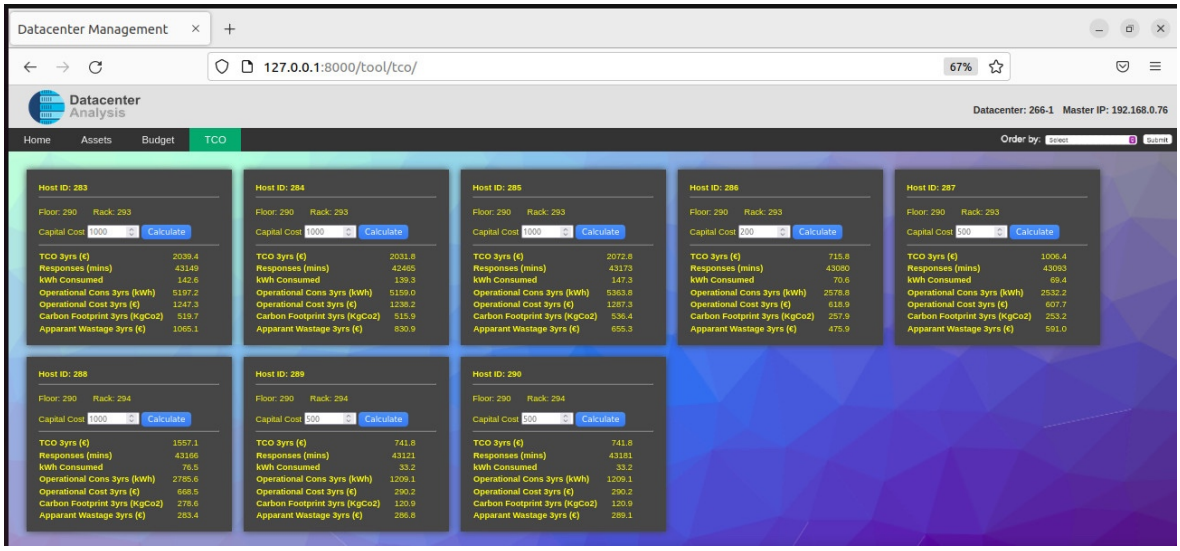


Figure 7.5: UCD Datacenter TCO Tab

7.2.3 Budget use

Figure 7.6 below shows the UCD datacenter which has been configured for an audit period between 01/04/2022 and 31/05/2022. The date that this screenshot was taken was 25/04/22. This is a set audit period and so the budget graphs will show the full period with a cumulative line showing how much has been consumed in days that have passed. Unfortunately, I was not able to implement a rolling carbon intensity (Section 6.11 - Changes from Original Spec). If this was implemented, both graphs would show much more variation from day to day. However, if a good estimate of the average carbon intensity was supplied by the user, the end values would still be the similar.

The right hand graph shows the total carbon emissions of the entire datacenter. The red line at y=100 is the budget that was specified by the user upon configuration. 51.54 KgCO2 has been emitted by the datacenter for the days between 01/04/2022 and 25/04/2022. By inspecting the graph we can see that, at current usage, the datacenter will exceed its budget before the end of the period. The graph on the left hand side shows the carbon emissions for each host individually. Host 285 has the highest emissions while host 290 has the lowest.

There are two more sets of graphs for Energy and Cost. These tabs can be opened by the user (and handled by JavaScript). The curve of the graphs will be identical since the energy price and carbon intensity are static values. The y-axis will show different values and different scales though. For example, the Energy tab shows euro on the y-axis and goes from 1-22 (host 283 consuming 21.14€ so far over the period).

At standard zoom the graphs below (Figure 7.6) fill the entire page. The zoom is at 67% so that all hosts can be shown without scrolling in the Assets and TCO tab. This is for demonstration only and in normal use standard zoom should be used. In datacenters with 6+ hosts, you will have to scroll down in the TCO tab to view them all. This is by design as you can have an arbitrary amount of hosts in a datacenter and attempting to fit them all on a single page without scrolling is not feasible.

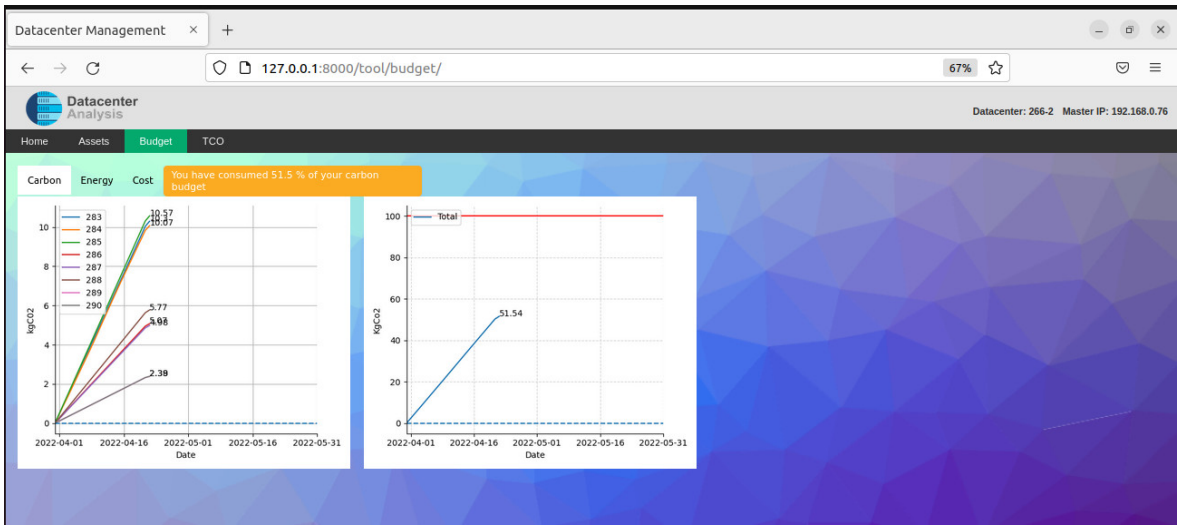


Figure 7.6: UCD Datacenter Budget Tab

7.3 Analysis Results

Installing and running the web application in the UCD datacenter in conjunction with Papillon has allowed us (the administrators) to come to some interesting conclusions about the efficiency

of our datacenter. The application was successful in every aspect that was required. We were able to identify servers being underutilised, inefficient servers, servers which were causing the most carbon emissions, servers consuming the most energy, most expensive servers, and provided a way for the administrator to plan and budget for short and long term goals. Using the tools provided, a datacenter administrator is provided with information that is invaluable in taking steps to reduce both their carbon footprint and operational costs.

Chapter 8: Conclusions and Future Work

This chapter outlines some potential areas for future work within the scope of the project. These ideas for future work include both small additions as well as large scale, more ambitious proposals. Finally, I will conclude with a discussion of the accomplishments of the project and whether or not the initial objectives have been met.

8.1 Future Work

The only advanced goal which was not met was the implementation of a rolling carbon intensity. The details of which are outlined in section 6.11 (Changes from Original Spec). This would be the first item to be added for any future work. As the project advanced and evolved, new ideas were coming to me constantly. Some of these are quite large scale and could be projects in themselves but tying it all up into one application would provide a very powerful tool for datacenter administrators to reduce their carbon footprint and operational costs.

8.1.1 Profile-based system

For larger organisations with dozens or even hundreds of servers, a profile-based approach could be implemented. A profile-based system would entail allowing teams/ groups/ individual users to create profiles and add any of their hosts to their profile. This way the datacenter could be split up into logical clusters of hosts. Analysis would be conveyed to only the owners of the servers. This service would still only be available to users on the network and so passwords for each profile will be optional. In companies with many teams, some may prefer to keep their energy/ cost consumption private.

Django provides architecture to add user accounts so this new feature would not be difficult to implement. Assigning servers to individual teams and allowing for the addition/ removal from a pool may be more difficult to design - much of the database would require restructuring. This is quite an exciting concept and adds a new layer of depth to the application which could be used by enterprise and larger organisations.

8.1.2 Recommendations

Personalised recommendations on how to reduce your carbon footprint and operational costs would be the next logical step in making this application more impactful. Through bench-marking the most common types of servers and the most common types of tasks, we can generate a model which would be able to accurately estimate how much you could save by virtualising servers. For example, if you have two servers in your datacenter, both running a simple task and running at 5% CPU utilisation, the application would then provide you with the estimated cost of running one server with both hosts virtualised on the single piece of hardware. The software implementation of this would be relatively easy, however, creating the model for so many types of servers, each with

multiple tasks, would be a lengthy project in itself.

8.1.3 Circular Economy

All servers have a life cycle. Newer servers may be more expensive but in general consume less energy for the same tasks. It can be difficult to judge when to invest in new servers for both reducing operational costs and reducing the carbon footprint of the datacenter. For example, upgrading a 15 year old diesel car to a brand new electric car will of course reduce your carbon footprint and over a long enough period of time, even your costs. However, upgrading that electric car every 6 months will raise both your costs and your carbon emissions drastically. A balance is required to upgrade at the ideal time. I would like to implement a system that calculates your potential carbon savings and your financial savings through buying new servers. As well as this, it would be able to identify any servers suitable for being recycled for various purposes and therefore extending their life cycle. An example of this would be outlining which servers can be switched to housing non-critical applications after being replaced.

8.2 Final Conclusion

The project has succeeded in its goal of developing an easy to use and informative web application for reducing the carbon footprint and operational costs of a small datacenter. The analysis on the UCD datacenter has proven its usefulness in identifying inefficient servers, helping to budget their consumption, and to compare servers from multiple perspectives. Users can configure multiple different instances of one datacenter allowing for different periods and budgets to be set which can overlap with each other. Users can also connect to multiple master servers by changing the IP address and the data retrieved is all saved to the database so you can easily switch between them. I am optimistic that UCD can continue to use this application in the future and that it will help in their goal of sustainability.

The development process for the application has followed an industry standard design flow making use of tools and methodology such as GIT, extensive testing, verification, and dockerisation of the final release. This has provided a stable and reliable release that is ready for use in a functioning datacenter.

Useful carbon dioxide emission statistics are relayed to the administrator. The carbon footprint of each server is estimated over a three year period allowing for comparisons to be made. For example, if you have a 6 week audit period starting with one server and you added a new server after 4 weeks, it would be difficult to compare the carbon emissions of the two servers over different period lengths. Real time graphs show estimates for how much carbon the servers have emitted up to the current minute or the end of the audit period specified, as well as a graph for the entire datacenter. The option is provided to set an overall carbon budget which will be shown on the graph. All of these tools have been implemented to help a datacenter administrator reduce the carbon footprint of their datacenter.

Similarly, metrics relating to the cost of the servers are also available. The operational cost over a 3 year period, the total cost of ownership over a 3 year period (includes capital cost), and the apparent wastage cost over a 3 year period are all calculated to provide insight into the cheapest/ most expensive servers as well as which servers are being underutilised (apparent wastage). A real time operational cost graph shows the estimated cost up to the current minute as well as a graph for the entire datacenter.

Other tools such as the interactive tree showing assets and their CPU usage % are included to help organise and promote proper management of the datacenter as well as identifying underutilised servers.

Unfortunately, one advanced objective was not met due to time and data constraints. This feature of a dynamic carbon intensity figure for the period would provide more accurate and informative carbon footprint and emissions figures. Only eight servers in UCD are running Papillon at the moment. I had hoped to test my application on a larger datacenter but this was out of my hands. The results we got from the datacenter are still valid and informative but I think a larger datacenter would have made for much more interesting analysis.

Acknowledgements

I would like to sincerely thank my supervisor Damian Dalton for his continued support and guidance throughout the completion of this project. His expertise in the field and feedback on the project has been invaluable.

I would also like to thank Abhay Vadher, a postdoc who works in UCD alongside Damian, for his assistance in setting up Papillon and providing me much needed advice on its use with precision and patience.

Bibliography

1. *Papillon* https://www.beeyon.com.
2. Koomey, J. & Taylor, J. New data supports finding that 30 percent of servers are 'Comatose', indicating that nearly a third of capital in enterprise data centers is wasted (2015).
3. Koomey, J. & Taylor, J. Zombie/Comatose Servers Redux (2017).
4. *Carbon Tax* 2021. https://www.citizensinformation.ie/en/money_and_tax/tax/motor_carbon_other_taxes/carbon_tax.html.
5. Gantz, J. & Reinsel, D. THE DIGITAL UNIVERSE IN 2020: Big Data, Bigger Digital Shadows, and Biggest Growth in the Far East. https://www.speicherguide.de/download/dokus/IDC-Digital-Universe-Studie-iView-11.12.pdf (2012).
6. Reinsel, D., Gantz, J. & Rydning, J. Data Age 2025, An IDC White Paper. https://www.import.io/wp-content/uploads/2017/04/Seagate-WP-DataAge2025-March-2017.pdf (2012).
7. Avgerinou, M., Bertoldi, P. & Castellazzi, L. Trends in Data Centre Energy Consumption under the European Code of Conduct for Data Centre Energy Efficiency. *Energies* **10.** https://www.mdpi.com/1996-1073/10/10/1470 (2017).
8. E3P, T. *Data Centres Code of conduct* July 2016. https://e3p.jrc.ec.europa.eu/communities/data-centres-code-conduct.
9. Fontecchio, M. *What is Power Usage Effectiveness (pue)? - definition from whatis.com* Apr. 2009. https://searchdatacenter.techtarget.com/definition/power-usage-effectiveness-PUE.
10. Delforge, P. America's Data Centers Consuming and Wasting Growing Amounts of Energy. https://www.nrdc.org/resources/americas-data-centers-consuming-and-wasting-growing-amounts-energy (2015).
11. *Sustainable Energy Authority of Ireland, Energy in Ireland Report* (2018). https://www.seai.ie/publications/Energy-in-Ireland-2018.pdf.
12. Flucker, S. & Tozer, R. Data Centre Energy Efficiency Analysis to minimize total cost of ownership. *Building Services Engineering Research and Technology* **34,** 103–117 (2013).
13. *PUE: A Comprehensive Examination of the Metric* (2014). https://www.seai.ie/publications/Energy-in-Ireland-2018.pdf.
14. Flucker, S., Tozer, R. & Whitehead, B. Data centre sustainability – Beyond energy efficiency. *Building Services Engineering Research and Technology* **39,** 173–182. eprint: https://doi.org/10.1177/0143624417753022. https://doi.org/10.1177/0143624417753022 (2018).
15. Masanet, E., Shehabi, A. & Koomey, J. Characteristics of low-carbon data centres. *Nature Climate Change* **3,** 627–630. https://doi.org/10.1038/nclimate1786 (2013).
16. Bizo, D. Multi-tenant Datacentres and Sustainability - Ambitions and Reality. *Schneider Electric* (2020).
17. *451 Research Overview* https://www.spglobal.com/marketintelligence/en/solutions/451-research.
18. *About SP Global Market Intelligence* https://www.spglobal.com/marketintelligence/en/about/.

19. Rodríguez-Haro, F. *et al.* A summary of virtualization techniques. *Procedia Technology* **3.** The 2012 Iberoamerican Conference on Electronics Engineering and Computer Science, 267–272. https://www.sciencedirect.com/science/article/pii/S2212017312002587 (2012).
20. Yfantis, V. *What is a hypervisor and what are its benefits?: Parallels explains* July 2021. https://www.parallels.com/blogs/ras/hypervisor/.
21. Pretorius, M., Ghassemian, M. & Ierotheou, C. *An Investigation into Energy Efficiency of Data Centre Virtualisation* in *2010 International Conference on P2P, Parallel, Grid, Cloud and Internet Computing* (2010).
22. *What is Carbon Intensity?* https://www.nationalgrideso.com/future-energy/net-zero-explained/what-carbon-intensity.
23. *CO2 Signal API documentation* https://docs.co2signal.com/#introduction.
24. *Beeyon* https://www.beeyon.com/about-beeyon/.
25. Dalton, D. & Vadher, A. *Papillon API Reference* 2016.
26. Dalton, D. & Vadher, A. *Papillon System Overview* 2019.
27. *Django Software Foundation* 2022. https://www.djangoproject.com/foundation/.
28. Korsun, J. *10 Popular Websites Built With Django* 2016. https://djangostars.com/blog/10-popular-sites-made-on-django/.
29. *Django Introduction* 2022. https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction.
30. *Python Selenium* 2018. https://selenium-python.readthedocs.io.

Appendix

Project Workplan

Figure 8.1 below shows my initial workplan as of December 2021. The project workplan was continuously changing during development because of unforeseen issues and setbacks, mainly to do with setting up a Papillon datacenter for me to develop on my machine (Figure 8.2).

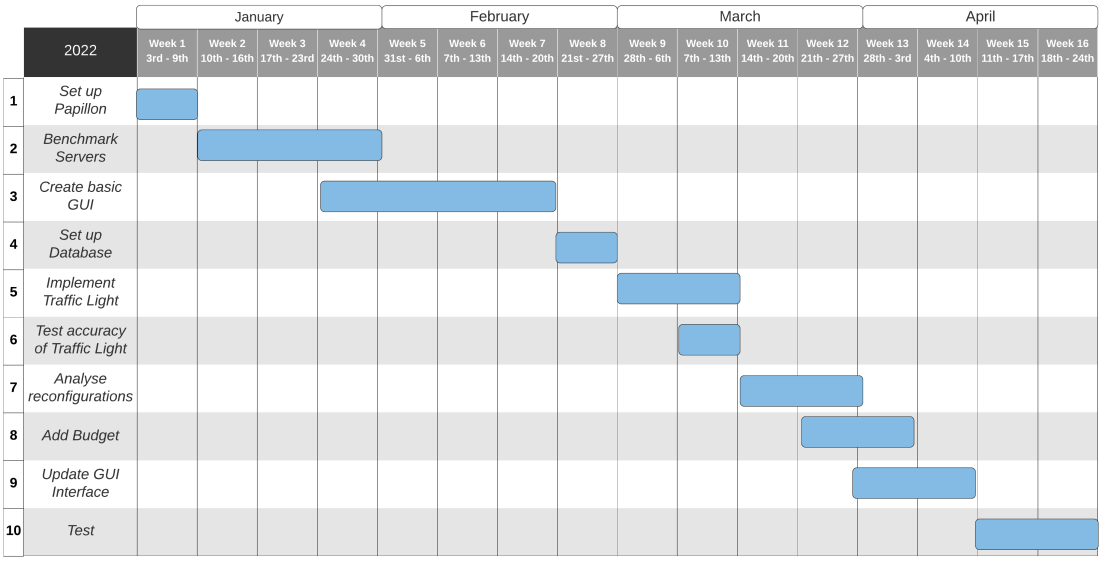


Figure 8.1: Initial Workplan Gantt Chart

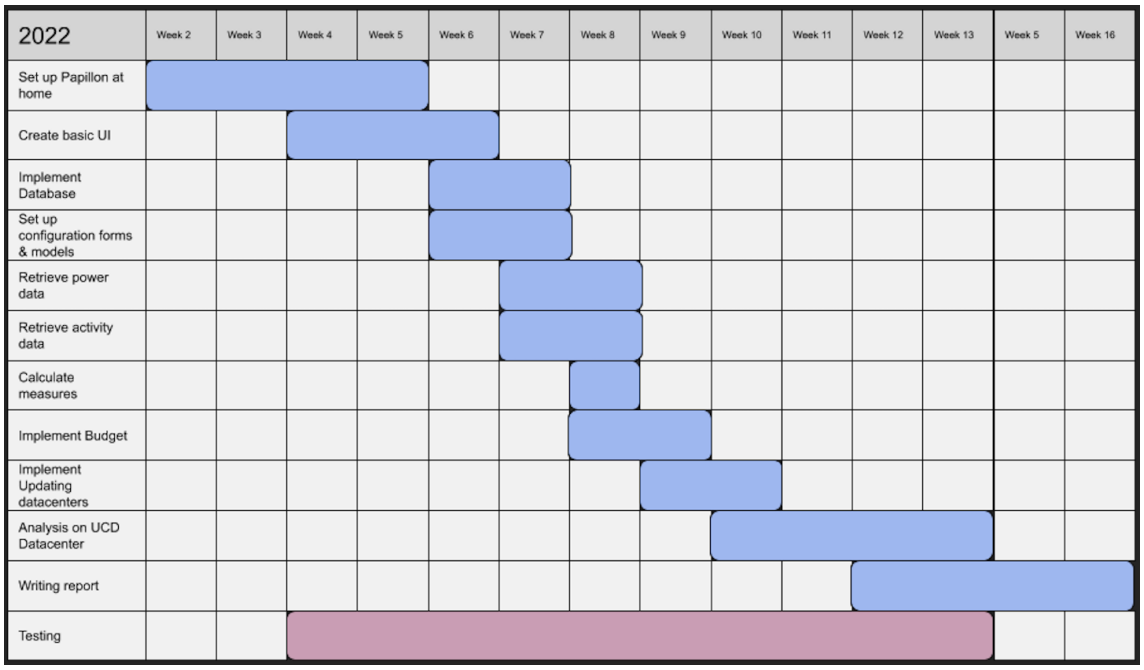


Figure 8.2: Revised Workplan Gantt Chart