



A Cloud-Based Solution to Industrial Internet of Things Server Monitoring

Harry Hawkins

184528

Supervised by Dr George Parisi

BSc Computing for Business and Management

School of Engineering and Informatics

University of Sussex

2021

Word count: 10,909

Statement of originality

This report is submitted as part requirement for the degree of BSc Computing for Business and Management (with an industrial placement year) at the University of Sussex. It is the product of my own labour except where indicated in the text. The report may be freely copied and distributed provided the source is acknowledged. I hereby give permission for a copy of this report to be loaned out to students in future years.

Signature:

A handwritten signature in black ink, appearing to read 'Harry Hawkins', with a stylized, cursive script.

Harry Hawkins

Acknowledgements

I would like to thank my supervisor Dr George Parisi for his continued assistance, guidance, and knowledge throughout the development of this project. I would also like to thank my family for providing me with support during my studies.

Abstract

The problem of monitoring Industrial Internet of Things (IIoT) devices is increasingly prevalent with the transition to the fourth industrial revolution. IIoT devices are often used to perform critical functions such as sensor readings for industries including power and healthcare. Due to the essential nature of these devices, ensuring maximum stability and system health is of paramount importance. This can be achieved by monitoring the devices from a central server, indicating the current state of IIoT devices to system operators, facilitating the quick resolution of issues and accelerated return to operation.

This project aims to address the monitoring problem with a cloud-based solution, providing an open-source system with a user-friendly interface to bypass the need for complicated technical processes associated with traditional methodologies of server monitoring. This solution allows businesses to save a considerable amount of time and money, which they can reinvest into technological advancement.

This project provides a web application that allows users to quickly and easily add or remove IIoT devices to a custom-built, open-source, cloud-based monitoring system. This monitoring system requires far less technical knowledge to deploy and operate compared to competing solutions.

Table of Contents

1	Introduction.....	1
1.1	Problem area and motivation	1
1.2	Aims and objectives.....	2
1.2.1	Aims.....	2
1.2.2	Objectives	2
1.3	Professional considerations.....	3
2	Related work	3
2.1	Literature review	3
2.2	Existing products and solutions	5
2.2.1	AppDynamics	5
2.2.2	BASEAPP Swarmsense	6
2.2.3	PandoraFMS	6
2.2.4	Nagios XI.....	7
3	Requirements analysis.....	7
3.1	Functional requirements.....	8
3.1.1	Mandatory requirements	8
3.1.2	Desirable requirements	9
3.2	Non-functional requirements	9
3.2.1	Mandatory requirements	9
3.2.2	Desirable requirements	10
3.3	Domain requirements.....	11
4	Design	11
4.1	Sequence diagram	11
4.2	Architecture design	12
4.3	Components	12
4.3.1	Dashboards.....	13
4.3.2	Metric exporter.....	13
4.3.3	Monitoring service	13
4.3.4	Programming language and web development framework.....	14
4.3.5	Database.....	14
4.3.6	Containerisation tool	15
4.3.7	Testing tools.....	15
4.4	User interface design.....	15
5	Implementation	15
5.1	System architecture	16
5.2	Data flow.....	16

5.3	Web application	16
5.3.1	Endpoint model	17
5.3.2	Add endpoint page	17
5.3.3	Remove endpoint page.....	18
5.3.4	Dashboards.....	18
5.4	Automation (back-end code).....	20
5.5	Workflow examples	20
5.5.1	Adding an endpoint.....	20
5.5.2	Deleting and endpoint	23
5.6	Long term data solutions.....	23
5.7	Deployment.....	24
5.8	Containerisation	24
5.9	Cloud deployment.....	25
5.10	System requirements	25
5.10.1	Host requirements	25
5.10.2	Endpoint requirements	26
5.11	Scalability	26
5.11.1	Prometheus.....	26
5.11.2	Grafana.....	26
5.11.3	Database.....	26
6	Testing.....	27
7	Conclusion	27
7.1	Future developments	27
7.1.1	Kubernetes	28
7.1.2	Machine learning.....	28
7.1.3	Auto determination of endpoint OS	28
7.1.4	Exclusion of metrics.....	28
7.1.5	Deploying the system in the cloud.....	28
7.2	Critical Evaluation	28
7.3	Conclusion	29
8	Bibliography	29
9	Appendix.....	33

1 Introduction

The Internet of Things (IoT) is the concept of connecting any electrical device to the internet and other connected devices [1]. These devices are becoming commonplace in our daily lives, ranging from wearable fitness devices to self-driving cars and fully connected smart homes [2].

The Industrial Internet of Things (IIoT) is the use of IoT devices and procedures in an industrial setting. As we become a more interconnected society, the use of IIoT devices is increasingly prevalent, with the global Industrial IoT market expected to reach approximately USD 751.3 billion by 2023 [3]. IIoT devices are commonly found in engineering projects such as wind farms, electric vehicle charging stations, power stations, factories, and a wide array of industrial environments. These devices often carry out critical tasks such as reporting data gathered by precise sensors. An example of this is using IIoT for monitoring substations or wind turbines in a power grid. Because of this, security and stability are of paramount importance. To ensure the stability of these IIoT devices, they should be monitored. Monitoring an IIoT server involves gathering data on CPU, disc, memory resources and network traffic, as well as a whole assortment of optional and customisable metrics. By monitoring the devices, the gathered data can assist decision-making and management to achieve optimum efficiency.

IIoT devices are often spread across vast geographical areas; therefore, having a monitoring system that is both scalable and secure is of utmost importance. The best way to ensure this is by hosting it in the cloud. The cloud is becoming the new normal for server solutions due to the low cost, ease of deployment, high flexibility, and low maintenance. According to Gartner, the worldwide public cloud services market grew by 17.5 per cent in 2019 to a total of \$214.3 billion, up from \$182.4 billion in 2018 [4].

Many cloud-based products are based upon an architecture known as Microservices. In [5], microservices is defined as “a cloud-native architecture that aims to realise software systems as a package of small services. Each service is independently deployable on a potentially different platform and technological stack”. When [5] refers to “small services”, these often take the form of containers. Containers are typically ran using a platform as a service (PAAS) product such as Docker. Docker defines a container as a standard unit of software that packages up code and all its dependencies, resulting in an application that runs quickly and reliably from one computing environment to another [6]. Using a microservices architecture composed of various containers, we can create a quick to deploy, cloud-ready solution for IIoT monitoring. Using Docker, we ensure that the necessary software for the monitoring system is stable on a variety of different operating systems and cloud providers.

By creating an intuitive and simple web application, we provide users of the monitoring system with a way to manage their environment quickly and easily, simply adding or removing IIoT devices and configure dashboards with no more than a few mouse clicks. This solution bypasses the usual manual configuration and installation process of a monitoring system, requiring technical knowledge and understanding of each component and data flow.

The novel combination of a purpose-built web application, automation and deployment code and an open-source technology stack comprises the Cloud-based Solution to IIoT Server Monitoring (CBSISM).

1.1 Problem area and motivation

Many available solutions to IIoT server monitoring are provided by expensive enterprise applications and frameworks, which require licensing on a per-node basis, as well as yearly maintenance fees. These enterprise applications frequently involve upfront fees and annual costs and often tie customers into a large ecosystem of additional proprietary applications, modules, adaptors, and extensions. These issues make it difficult for industrial businesses to innovate and adapt freely due to the lack of freedom and flexibility.

As previously mentioned, IIoT devices often carry out system-critical tasks. Monitoring is a great way to ensure the stability of critical digital infrastructure. Instead of purchasing a license for an expensive monitoring application, an IIoT centred business can develop its own monitoring system. However, this involves a vast amount of specialist knowledge, research, time, and manual technical configuration, especially with many endpoints.

A notable use case for IIoT monitoring is offshore wind farms due to the critical nature of power systems data. The average operational offshore wind farm in the UK consists of 59 turbines (Appendices A). In a self-made monitoring system, this many endpoints would result in a time-consuming manual configuration process for each IIoT device when adding the device as an endpoint to the monitoring system, costing the business valuable man-hours. Not only is it a slow process for a business to manually design and create a proprietary monitoring system, but they would also need to hire specialist staff to manage their system, as well as for ongoing support, management, and configuration. This can be expensive, with the average Senior Systems Administrator in the UK costing around £48,000/yr [7].

This project recognises the necessity for technological advancement; therefore, a primary high-level objective is to promote innovation and development within industrial and engineering businesses. By using a suite of open-source, customisable and specially curated software, programming languages and tools, we can build a solution to give companies a quick, customisable and easy-to-deploy system that provides the same, if not better, experience than that of an expensive enterprise product, combined with the ability to extend or reconfigure at low cost, additionally removing the need for a business to create their own monitoring system at great expense.

1.2 Aims and objectives

1.2.1 Aims

The overall aim of this project is to use a specifically curated selection of technologies to design and develop a deployable cloud-based solution to IIoT server monitoring. By carefully selecting software, programming languages and frameworks, a system can be designed and built, allowing IIoT servers/devices to be monitored easily from a cloud-based central server.

An intuitive administrator user interface is implemented to enable users to connect and configure the IIoT endpoints quickly and easily. This is a crucial aspect of the solution, as one of the main advantages compared to competitors is the ease of use and configuration. The administration page is connected to a dashboard page, displaying the monitored metrics in a customisable format.

To develop this solution, we researched IIoT and cloud technologies to find the resources that enable the best solution possible.

To ensure the software is fully functional, research into available testing methodologies and technologies has taken place to find a solution for large scale testing and simulation of IIoT devices. This is a necessary part of the software development lifecycle to ensure that the solution meets intended requirements and is secure and reliable.

1.2.2 Objectives

- The system collects server metrics from remote IIoT devices and displays them in a customisable graphical interface. This will provide the user with important information about their endpoints, allowing them to monitor their status, security, and stability.
- The system uses a dashboarding tool to allow users to create customisable dashboards and alarms based on gathered metrics; this should be integrated with the administration page.
- The system allows the user to add IIoT device endpoints easily and quickly to the system through a simple user interface. This requires automating the complex and individual node

configuration that would otherwise be necessary. The system will use automation technology to install, configure and scale the required resources.

- The system uses a containerised architecture [6] to maintain scalability and stability. Containerisation results in a secure and ready to deploy program.
- The system collects time-series data stored in a well-designed database that can scale as required with the number of endpoints.

1.3 Professional considerations

There are four fundamental principles of the BCS Code of Conduct [8]:

You make IT for everyone

This principle is regarding allowing wide access to IT. This project uses open-source languages and resources, which promotes equal access to IT for everyone.

Show what you know, learn what you don't

This principle encourages continuous learning and not taking on tasks that you don't have the skills to complete. This project is within my skill set, and anything that I do not know, I will research and learn.

Respect the organisation or individual you work for

This principle ensures that you act in the best interest of your organisation or client, including taking responsibility for actions. This project aligns with the interests of the University of Sussex.

Keep IT Real. Keep IT professional. Pass IT on

This principle relates to acting as an ambassador for the IT industry. It encourages the positive promotion of the industry to the world. This project acts in BCS's best interest to uphold its reputation and support the IT industry.

Another professional consideration regarding this project is ensuring the security and privacy-related components. It is assumed that the system will be deployed on the cloud, with all necessary security provisions in place configured by the system administrator managing the deployment, including firewalls, access restriction and encryption of sensitive data. It is a requirement for the CBSISM to have all necessary ports opened/closed, depending on the user's endpoints and system design; it is not the responsibility of this project to provide this. The system administrator is expected to have the required knowledge of network security to protect against unwanted access.

2 Related work

This section explores work related to the domain of this project. Firstly, the literature review analyses academic research into IIoT, cloud technologies, monitoring and other areas related to this project. The existing products and solutions section discusses current solutions to IIoT server monitoring, with their benefits and drawbacks.

2.1 Literature review

With the ever more ubiquitous use of the Industrial Internet of Things and cloud-based technologies, IIoT and monitoring of IIoT implementations have been the subject of extensive academic research in the last two decades.

The fourth industrial revolution concept is presented in [9], which is characterised by its reliance on the use of Cyber Physical Systems capable of communication with one another and autonomous decisions, increasing overall industrial efficiency and safety. This is commonly considered synonymous with IIoT

due to the inclusion of IoT devices within modern industrial systems. A robust definition of IIoT is put forward in [9], identifying gaps in literature and understanding of IIoT, providing use cases of IIoT and information regarding current infrastructure. The apparent necessity for IoT devices across various industries makes the CBSISM an important development for modern industrial systems.

It can be challenging to see how IIoT can fit into so many industries; however, there is a clear academic interest in developing IIoT for widespread use cases, from manufacturing [10] to electric vehicle charging [11].

Healthcare is an industry that benefits significantly from IIoT devices, for example, IoT enabled electrocardiogram (ECG) machines. The authors of [12] state that smart healthcare plays a significant role in healthcare applications due to the use of embedded sensors and actuators to monitor and track patients. Because of the criticality of these devices, monitoring them to ensure stability is a high priority. The value of monitoring IIoT devices in the healthcare industry (commonly referred to as HealthIIoT) is expressed extensively in [13]. The paper proposes a system for monitoring sensor readings from HealthIIoT devices such as ECG. It highlights the benefits of hosting a monitoring system in the cloud, such as “seamless access by healthcare professionals” and faster overall processing, contributing to making the CBSISM a cloud-ready system. In the paper [14], the further benefits of using IoT with the cloud are discussed, stating that cloud computing provides an ideal back-end solution for large data streams and processing them in real-time for the unprecedented number of IoT devices and users [14]. There are numerous business benefits when combining IoT and cloud-based technologies.

Data security is of paramount importance for IIoT deployments, particularly within national infrastructure. Blockchain is one way of ensuring a high standard of security within IIoT, “Blockchain is widely considered as a promising solution, which can build a secure and efficient environment for data storing/processing/sharing in the IIoT” [15]. Blockchain has numerous benefits for IIoT deployments, such as improved data security. However, the scalability for systems with high data throughput is challenging to achieve with Blockchain. Therefore, the cloud is a suitable choice.

Field monitoring is another primary use case of IIoT monitoring; the importance of flexibly and dynamically monitoring field servers is explored in [16]. Field servers are akin to IIoT devices, as they are used to gather information from sensors in an industrial setting. The importance of monitoring is expressed in [16], by discussing its benefits in the context of agriculture.

Agriculture is an example of an industry where IIoT can be utilised to improve business processes. The authors in [17] discuss how IoT can ensure the quality of produce by controlling factors such as soil pH, humidity, nutrients, and monitoring temperatures of produce throughout the supply chain. Not only can the live data be used to aid decision making, in [17], the writers explain how historical data can be analysed to assist farmers in choosing the most lucrative crops depending on the time of the year. Historical data analysis is a significant benefit of monitoring IIoT devices. As shown in [18], the Process Query System (PQS) is a server monitoring system that can monitor the state of processes and an entire system. From this, [18] creates models to estimate the system state to identify issues with systems. Being able to develop predictive models based on historical monitoring data is a brilliant by-product of server monitoring. One of the future developments (Section 7.1) for the CBSISM is the possibility of using Machine Learning to predict and identify failures by analysing gathered system metrics.

Power systems are becoming increasingly dependent on IoT. Two primary examples of IoT in power systems are Wide Area Measurement Systems (WAMS) [19] and Supervisory Control and Data Acquisition (SCADA) [20] [21] [22]. Additionally, there is the concept of the Smart Grid [23]. WAMS systems are comprised of tens, hundreds or thousands of sensors and computers used to gather power systems information from a power grid. One of these devices is known as a Phasor Measurement Unit (PMU) [19]. PMU’s run at a very high data rate, meaning any downtime or system issue could cause significant data loss. This is an example of an IIoT device that would benefit from being monitored due

to the high criticality and importance of its operation. Power Systems are used as the primary use case for the CBSISM, as there is a large amount of public information regarding typical deployment sizes and architectures, which provide an example of the ideal number of devices the CBSISM should support, as well as a way to put the project into context with a particular industry.

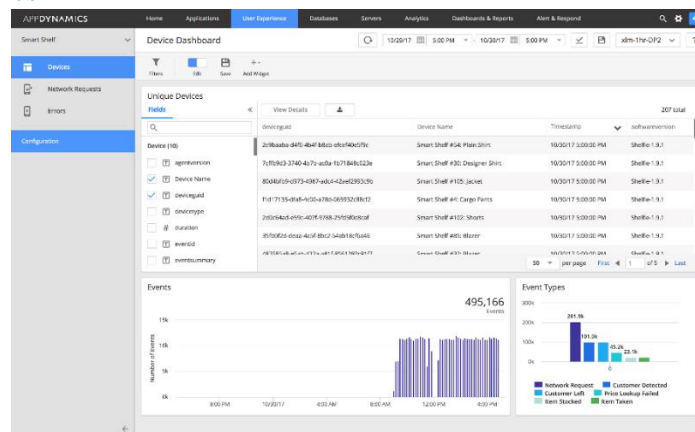
Monitoring is considered integral in many types of computer system architectures; the paper [24] puts forward designs for a Virtual Machine performance monitoring system, which can be used to identify bottlenecks in performance and stability. Additionally, a system for monitoring performance within a client-server architecture is presented in [25]. Although these papers do not discuss the use of IIoT devices, they contain invaluable information regarding the monitoring of computer systems in general.

As expressed in [26], servers are the core of a network. The paper explores how to monitor servers in a lightweight methodology using the Simple Network Management Protocol (SNMP). The authors of [25] discuss the process of monitoring server resources and put forward four key areas to be monitored: “static information (hardware description, software description, administrator, physical location, etc.), dynamic information (interface traffic, usage of CPU, memory and disk, etc.) network services (HTTP, FTP, DNS, SMTP, POP3, SQL Server database, etc.) and network performance” [26]. Although their monitoring methodology is different from the CBSISM, the aforementioned key monitoring areas helped to decide upon meaningful metrics that this projects solution aims to support.

2.2 Existing products and solutions

There are several existing solutions to server monitoring on the market, including products catered towards IoT. However, many of these products tie customers into a technological ecosystem, forcing users to pay for licenses, as well as maintenance and support plans. Not only are these solutions costly, but some businesses (especially national infrastructure operators) could also consider it a security risk to allow a third-party company to manage their server monitoring data completely. The CBSISM is a fully open-source, novel solution to IIoT server monitoring, bringing together new and improved features compared to available alternatives. Below are examples of the most popular server monitoring products.

2.2.1 AppDynamics

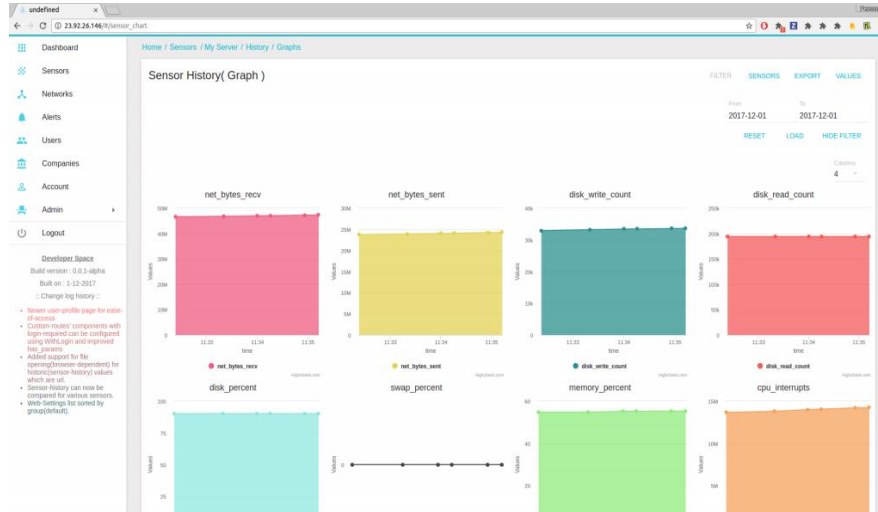


AppDynamics dashboard [27]

AppDynamics [27] allows you to monitor and manage C/C++ and Java apps by providing complete visibility into every line of code. It uses their machine learning platform “Cognition Engine” [28] to solve network issues, auto-detect performance problems and identify root causes through root cause analysis (RCA), as well as visualise revenue paths and correlate customer and application experience. The advantages of AppDynamics are that it has a wide range of advanced features, it is part of Cisco (a trustworthy business), and it is suitable for large enterprises. However, AppDynamics is closed-source and is a paid product with pricing available on request. In conclusion, AppDynamics is an

industry-leading application intelligence platform that provides monitoring, management, and systems analysis. As the product is paid for and contains many features, this could be viewed as overly complicated for a customer who only wants the monitoring aspect. It is an excellent product for large enterprises with big budgets; however, it could be considered more than required for many customers.

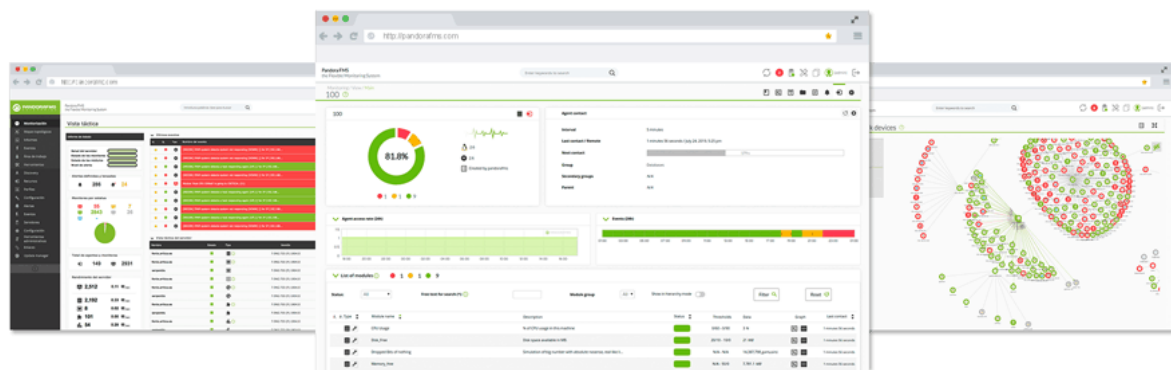
2.2.2 BASEAPP Swarmsense



Swarmsense dashboard [29]

SwarmSense is a fully equipped, self-hosted IoT platform for monitoring any type of time-series data, including server metrics and stats [29]. It supports time-series data which allows real-time monitoring with a wide range of optional metrics. To connect an IoT device to the system, a Python server script must be written and ran as a cron task to expose the monitoring data on an HTTP endpoint. Unfortunately, it is a paid service, and the requirement of python knowledge to configure the system could put off potential users. The functionality of Baseapp is like the CBSISM. However, it is a paid service that requires technical knowledge to set up. The need for the user to write Python scripts to configure endpoints could slow down the deployment and configuration process and make it overly complex for the average IT administrator. The CBSISM also uses Python for endpoint configuration. However, this is hidden from the user through the use of the front-end web application, and users are not required to have any programming knowledge.

2.2.3 PandoraFMS

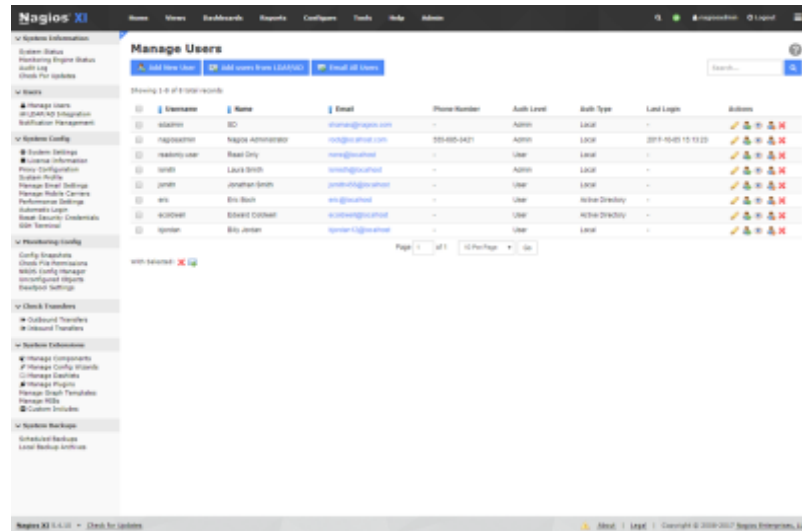


PandoraFMS dashboard [30]

PandoraFMS is similar to the previous two solutions; however, it is a much larger, all-in-one solution, providing an extensive array of features such as UX monitoring, remote control and IoT monitoring [31]. Due to the large number of features, a paid licensing structure is used, making this product

undesirable compared to the CBSISM, especially for smaller businesses who want a simple solution to monitor their devices.

2.2.4 Nagios XI



Nagios XI dashboard [32]

NagiosXI [32] is a well-established, popular tool with thousands of available plugins. Because of this, it is costly (\$3,495 a year). This is a multifaceted, advanced product for large enterprises. Not only is the tool expensive, but the enterprise edition licenses also require annual renewal of maintenance and support contracts. Relying on a third party to manage all maintenance and support can be risky for businesses running national infrastructure or critical systems. Due to the high cost and M&S requirements, Nagios XI may be inaccessible to smaller businesses and undesirable to businesses who wish to control their own system maintenance. A free version of NagiosXI supports up to 7 nodes, which is not enough for most IIoT use cases. However, this could be a good option for small businesses to monitor their on-site servers or research purposes.

3 Requirements analysis

Several features make this project a novel solution. The system is comprised of only open-source software, this is different to many competing solutions that operate on a paid enterprise licensing model and do not let users access the source code.

Furthermore, this solution aims to be easier to configure and operate compared to competitors. As seen in Section 2.2, many current products require specialist knowledge to configure, including having to write configuration scripts in programming languages and having advanced knowledge of computer networks. This project aims to be usable by anyone with the ability to use web forms and a basic understanding of TCP/IP. In contrast to competitors, which require computer scientists and engineers to operate. The web administration page aims to abstract the technical configuration process, making the solution more accessible.

Scalability and security are a considerable influence on this project. Using a microservices architecture and being cloud-ready, security and scalability can be achieved, making this system stand out amongst competitors.

The requirements below were used when designing and implementing this solution to ensure all necessary features are in place and formalise the system's functionality. The functional requirements relate to the behaviour of the system. Non-functional requirements define the attributes or constraints

to the system [33]. Domain requirements are requirements reflecting the environment in which the system operates [34], as opposed to the system itself.

This section explores the project's requirements, with analysis detailing their level of fulfilment, indicating where to find further information on the requirements related work within this project.

The requirements are in the following format:

Reference: Short description

Information: Use case/further description

- Has the requirement been met / The location of further information within the report

3.1 Functional requirements

3.1.1 Mandatory requirements

Mandatory requirements are imperative to the success of this project.

F1: The system shall enable users to add a new IIoT device as an endpoint to the monitoring system.

Information: A user has a new IIoT device they wish to monitor. To do this, firstly, the user must ensure the device is reachable by the host server, then simply enter the device information into the web application.

- The web interface allows the simple addition of a new IIoT endpoint. See section 5.5.1 for more information regarding how this is achieved.

F2: The system shall store information on the endpoints in a database, allowing endpoints to be added and deleted.

Information: When a user adds a new endpoint, the information for this is stored in a database. This can be deleted when the endpoint is no longer required.

- Endpoints can be added and deleted from the system. See Section 5.5.

F3: The system shall automate the installation and configuration of exporter software on IIoT endpoints.

Information: When a new endpoint is added to the system, the metric exporter is installed on the device, ready to expose metrics for the system to scrape.

- Automation scripts install NodeExporter on the endpoints when an endpoint is added. Section 5.4 covers the automation scripts used within the project.

F4: The system shall automate the configuration of new endpoints on the monitoring service.

Information: When a new endpoint is added to the system, the monitoring service is updated to scrape the exposed metrics on the new device.

- Automation scripts automatically provide the configuration of Prometheus when a new endpoint is added. See section 5.4.

F5: The system shall provide customisable dashboards with real-time monitoring data.

Information: The user can view and customise dashboards showing monitoring data in real-time. When endpoints are added or removed, the dashboards update to reflect this.

- See section 5.3.4 for information regarding the dashboards.

F6: The system shall be able to show historical data as well as current data.

Information: The user can change the time frame of the data in the dashboards to view either real-time or historical data.

- The date range can be adjusted in Grafana to allow viewing of any time period of data. This can be seen in Figure 10.

F7: The system shall provide the ability to set warnings and alarms based on gathered metrics, as well as add severity to alerts to allow a reduction in alert noise, which could be a significant issue with large deployments.

Information: To ensure specific critical metrics are stable, the user may want to set an alarm to be warned when a metric goes over a threshold, for example, if the RAM has been at total capacity for a long time, the user can set warnings and alarms within the dashboarding tool to flag this visually. If there are many alerts set, it is beneficial for these alerts to be assigned a severity so that the most critical alerts can be addressed first.

- Grafana provides this functionality.

F7: The system shall allow the grouping of endpoints within dashboards

Information: If a user has thousands of nodes, it is not beneficial to see them all at once, by grouping endpoints by region, this can provide a more user-friendly way to see their system.

- Grafana provides this functionality through the ability to create your own Prometheus queries.

3.1.2 Desirable requirements

Desirable requirements are not essential to the project's success; however, they provide additional functionality that is desirable to users.

F8: The system should allow users to predict possible future issues, as well as complex analysis of the metrics.

Information: The user may want to predict issues with hardware ahead of time. This can aid with decision making for a business, as they can see when devices need maintenance or replacement ahead of time.

- Users can use the gathered data however they desire; however, the system provides no functionality to predict issues itself.

3.2 Non-functional requirements

3.2.1 Mandatory requirements

NF1: The system shall be suitable for cloud deployment.

Information: To improve overall availability, scalability and stability, the system should be deployable to a cloud environment.

- The system uses a containerised architecture, making it suitable for cloud environments. See Section 5.8, 5.9 and Figure 23.

NF2: The communication within the system shall be as secure as possible.

Information: A national infrastructure project such as a wind farm may want their data to be as secure as possible, therefore ensuring secure processes are used within the cloud can prevent security concerns. An advantage of this project is that the open-source technologies used to allow the security to be more easily assured than closed-source code.

NF3: The deployment shall be able to scale automatically based on the number of endpoints.

Information: The system should handle a wide range of business sizes, for example, a small business with only 2 IIoT devices and a large business with 200 IIoT devices should both be able to use the same system with no additional configuration.

- Section 5.1.1 discusses scalability within this project.

NF4: The administration web interface shall be created using Django - a Python Web framework.

Information: The administration web interface is the front end of the system, where the user can add and remove endpoints.

- Django was used to create the web application for this system.

NF5: The database shall be able to scale to store time-series data to provide quick and easy access to data.

Information: Much like NF3, the database should be scalable to cope with a range of deployment sizes, in addition to providing availability and ease of access.

- See section 5.11.

3.2.2 Desirable requirements

NF6: The system should use a microservices architecture to improve scalability, security, and ease of deployment.

Information: Businesses will benefit from high levels of scalability, security, and ease of deployment as it will allow them to adapt and innovate their deployments freely, without cost. By using a microservices architecture, these features inherently stronger compared to a traditional monolithic application architecture.

- This project uses Docker. See section 5.8.

NF7: The system should be self-healing. Meaning that if a component goes down, it fixes itself with minimal downtime.

Information: Businesses benefit when downtime is minimised. If part of the monitoring system fails (either due to hardware issues or unexpected external circumstances), self-healing and automatically restarting processes can save significant maintenance time.

- See section 7.1.1 for discussion regarding self-healing through the use of Kubernetes.

NF8: The system should require minimal initial configuration for deployment, reducing the need for in-depth technical knowledge.

Information: Initial configuration refers to the installation and deployment of the system. Installing a typical monitoring system requires specialist skills and knowledge of computer networks and

programming. This system aims to be accessible to less skilled users, reducing costs for businesses using the monitoring solution.

- The system uses an installation shell script, explained in section 5.7.

NF9: The system should support the use of HTTPS for communication between endpoint and server.

Information: To improve the security of the system, an encrypted protocol is preferred. The data passed from the devices to the monitoring system is purely system metrics, this does not contain personal data that would need to be covered by GDPR. This solution uses HTTP for communication between endpoints and servers. Section 7.2 discusses TLS within this system and why HTTP is used instead of HTTPS.

3.3 Domain requirements

D1: The system shall be suitable for enterprise IIoT implementations – to do this, we ensure the system is well tested, usable, and scalable

D2: The average IT employee should be able to operate the system by using a simple UI with abstracted technical areas.

D3: There must be a variety of possible endpoint devices, making the system suitable for various IIoT use cases by allowing the support of popular operating systems and devices platforms

All three domain requirements have been met.

4 Design

This section explores how the system was designed, including designing the system architecture and selection of components. Firstly, a sequence diagram is used to portray a typical use case within the system, combining this with the requirements in section 3 to develop the architecture and identify necessary areas within the system. The components section details the different areas of the overall solution, with research into possible technologies and solution stacks for this project.

4.1 Sequence diagram

Figure 1 shows a sequence diagram detailing the typical processes within the system. This diagram was used when designing the architecture and selecting components, as well as for referencing when programming to verify functionality.

The use case shown in this diagram is the process of adding an Endpoint to the system. A System operator adds the endpoint information into the web interface, which passes the parameters of the device, such as IP address and credentials, to the automation code. The automation code then installs the metric exporter on the IIoT device. While this occurs, the endpoint information is stored in the database, and the dashboard is updated to show the new endpoint.

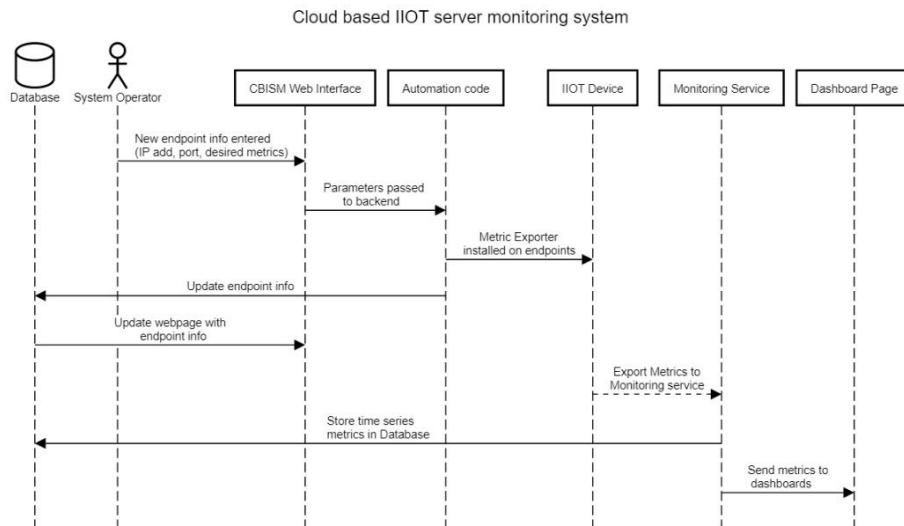


Figure 1 - Sequence diagram

Although this is a very high-level diagram, it helps to build a picture of the system, and by following this diagram as a reference in the design process, it ensured that all functionality and components are present, aiding with the decision process for determining specific components.

4.2 Architecture design

Figure 2 shows a high-level overview of the architecture of the CBSISM, detailing the separate areas of the system and how they interact.

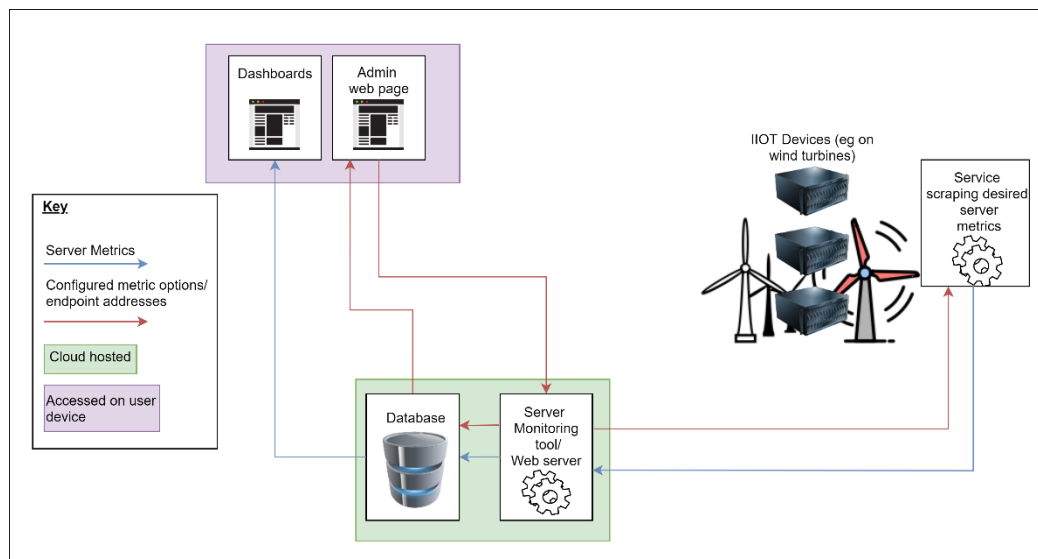


Figure 2 - High-level architecture

Figure 2 is a high-level diagram with little detail regarding specific technologies. This Figure was utilised to solidify an initial architecture, allowing decisions to take place about particular components. The Figure was designed as a development from Figure 1, translating the sequence diagram use case into a general-purpose diagram showing data flow and system constituents.

4.3 Components

All chosen components within this system are open source. This means that the software is free to use and has source code that anyone can inspect, modify, and enhance [35]. This has many advantages. One advantage of open source software is that it can be considered more stable and secure than closed source.

This is because anyone can view and modify the source code, resulting in the possibility of someone spotting errors or exploits. This is often a faster process compared to bug fixes in proprietary software. Another advantage is that “Because programmers publicly distribute the source code for open-source software, users relying on that software for critical tasks can be sure their tools won't disappear or fall into disrepair if their original creators stop working on them.” [35]. These advantages contribute to the decision to use only open source components.

Below are the areas of the system which required decisions regarding options of software and tools. To ensure the best selection was chosen for this project, research into available alternatives for each area of the system was conducted. The components can relate to areas within Figure 2, as this was used to identify the need for the chosen tools and technologies.

4.3.1 Dashboards

A tool is required to provide customisable dashboards; this is used to show gathered metrics and monitoring information, as well as alarms and analysis. The dashboard component can be seen in Figure 2 in the purple box. For the dashboard tool, the decision was made to use Grafana [36]. Grafana is industry-leading and open-source. It has a Docker [6] distribution, making it easy to plug into the system. As well as this, Prometheus is supported as a data source, which automatically updates when new endpoints are configured (see 4.3.3 for more on Prometheus).

Grafana is very customisable and has thousands of downloadable plugins. As well as the ability to create your own Grafana dashboards, there are many online to download and use. Dashboards can be stored in JSON format, allowing sharing and version control of dashboards. For the CBSISM, we have decided to use the Node Exporter Full dashboard [37] on Grafana Labs. This uses Prometheus data sources and is pre-configured to support the other components within the CBSISM. Despite the use of this dashboard, users are free to create and customise as many dashboards as required. Furthermore, Grafana allows the creation of alarms and alerts that warn a user when specific metrics hit specified levels.

4.3.2 Metric exporter

A metric exporter is a service that gathers metrics from a system and exposes the data via a chosen protocol. This service is located on the IIoT device server itself – as seen in Figure 2 in the white area captioned “Service scraping desired server metrics”. The metric exporter is automatically installed on the IIoT device when it is added to the system.

For this service, there are two industry-leading options, Prometheus NodeExporter [38] or collectd [39]. Prometheus NodeExporter was chosen as the more suitable option. This is because NodeExporter is produced by Prometheus, which is a well-established monitoring technology. NodeExporter and Prometheus are designed to work together. By using both in this project, we reduce compatibility issues between the exporter and monitoring service.

4.3.3 Monitoring service

A monitoring service is required to scrape metrics that have been exported from the IIoT device by NodeExporter, and this acts as the entry point for server metrics to the overall monitoring system. The way this service fits into the system can be seen within the green area of Figure 2, in the box titled “Server monitoring tool”.

The most popular tools that provide this service are the following [40]:

- Prometheus
- Nagios
- Graphite
- InfluxDB
- OpenTSDB

Prometheus is seen as the industry standard for cloud-based monitoring, which can be deduced from [40], which shows its comparison to other monitoring tools. As described in [40]:

- Nagios is based on exit codes, with no provided storage solution.
- Graphite is less detailed than Prometheus and harder to integrate, more suited for long term data storage.
- InfluxDB has the same constraints as Graphite
- OpenTSDB has the same scope issues as Graphite, coupled with the high complexity and cost of using Hadoop and hbase, which is overly complex for the project.

As a result of these comparisons, Prometheus was chosen. Prometheus is great for cloud-based deployments. The compatibility with NodeExporter and Grafana makes it the ideal choice.

For data storage, Prometheus includes a local on-disk time-series database but also optionally integrates with remote storage systems [41].

4.3.4 Programming language and web development framework

Python [42] was chosen as the primary programming language for this solution, as it has many packages and libraries that provide useful functionalities needed for this project. Python is required to perform a lot of the automation within this system, such as the installation of Node exporter on IIoT devices, as well as adding new endpoints to Prometheus. This automation process requires Python packages such as the SSHv2 package “Paramiko” [43] and scp [44] to communicate with the IIoT devices, which make the development of automation scripts simple and effective, bypassing the need to write proprietary SSH and SCP client.

A web development framework is required to build the front end of the system, where the user’s main point of interaction with the endpoints takes place. An administration web page (the purple area of Figure 2) allows the addition of endpoints, as well as a link to the Grafana dashboard page.

Django was chosen as the Framework as it is based on Python, which is used for back-end automation. Django is more advanced than other Python-based web frameworks such as Flask [45], due to its abundance of features such as testing, administration portal and native database integration. By using a Python-based web framework, the overall system complexity is reduced. “Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design” [46]. Integrating the front-end code with the back-end scripts is vital to this project, and this was made simple by using Django and Python.

4.3.5 Database

A database (see the green area of Figure 2) is needed to store both the application data and monitoring data. The monitoring data is gathered on a high-frequency basis. As Prometheus is often used in conjunction with TimescaleDB [47], TimescaleDB is used. Prometheus is used as the primary storage solution for the gathered time-series data; however, Prometheus can integrate with remote storage solutions, including TimescaleDB. TimescaleDB is an open-source implementation of a PostgreSQL time-series database, which enables secure and safe storage of time series data, as well as typical PostgreSQL functionality. Grafana and Prometheus both have native integration with TimescaleDB. Django has native integration with PostgreSQL, which allows TimescaleDB to be used as the Django database for application data.

The monitoring data is not actually stored in TimescaleDB in the CBSISM, as it was not necessary for this project due to Prometheus’s time-series database proving adequate. However, it is possible to adapt the storage solution to use TimescaleDB as primary data storage, using the Prometheus PostgreSQL

Adapter [48] or the new tool Promscale [49]. It is likely that in the cloud, a user may want to integrate a storage solution such as Amazon S3 [50] to facilitate long term archiving, scalability and performance.

4.3.6 Containerisation tool

Docker [6] is used to facilitate Containerisation. Docker is the industry standard for Containerisation technologies, it has the strongest support and plentiful documentation. Containers are used to manage the deployment and orchestration of services. Within Figure 2, some of the cloud-hosted components are containers.

4.3.7 Testing tools

Ideally, the system would be tested on a cloud platform such as AWS or Azure, with many real or simulated IIoT endpoints; however, this is very costly and out of the scope of this project.

The Django test execution framework is able to verify if the application is functional by using physical test nodes and writing tests in the test suite. The testing process is automated throughout the development lifecycle.

4.4 User interface design

Add new node

Device name:

IP Address:

Admin username:

Password:

Check desired metrics

☐ CPU

☐ RAM

☐ Temperature

Choose a protocol/format:

Choose an operating system:

Figure 3 - Simple front-end design

Requirements F1 and D2 state that the user must be able to add a new endpoint easily, and the system must be suitable for the average IT person. To achieve this, a simple UI is used to reduce complexity and make the endpoint addition process quick and unambiguous. Figure 3 shows a design that was used as a starting point for the web application. It is quite minimalistic, with no unnecessary features. By allowing the user to enter the endpoint information and click one button, the process to add an endpoint is extremely simple, requiring minimal IT skills. The design uses a form structure, which is familiar to users of web applications.

5 Implementation

This section explores how the final project solution is implemented based on the previously mentioned designs. Firstly, by showing the final system architecture and data flows. Followed by an explanation of the web application and back-end automation. Workflow examples are provided to demonstrate the system's functionality, with additional discussions regarding long term data solutions, deployment of the system, the containerised architecture and cloud deployment.

5.1 System architecture

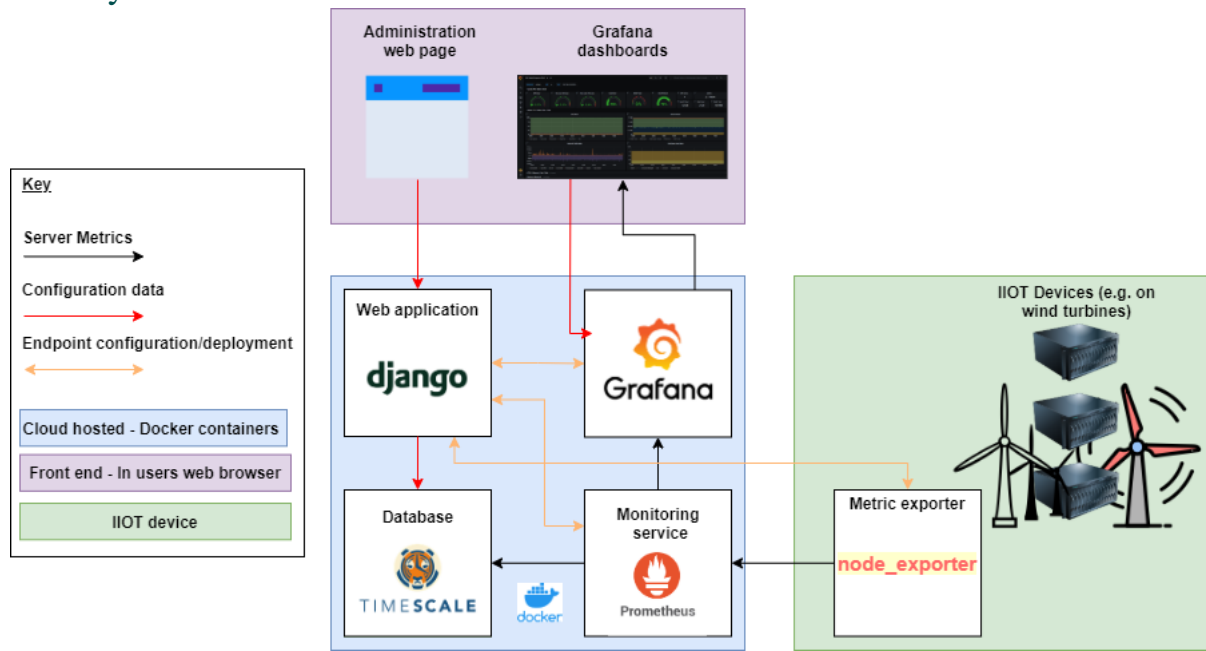


Figure 4 - Final system architecture

As an adaptation from Figure 2, Figure 4 shows the components of the system, with detail regarding specific technologies used, considering the decisions made in section 4.3. This helps to demonstrate the final chosen tech solution stack as well as detail the data flows and tools used within the system.

The purple area shows areas of the system visible to the user – the administration web page and the Grafana dashboards. The blue area shows the components hosted in the cloud in the form of Docker containers. These are the back-end areas of the system that facilitate the management of the database, web application, automation, and monitoring. Finally, the green area is the IIoT endpoints with the metric exporter service on each device within the system.

5.2 Data flow

As shown in Figure 4, the three arrow types indicate how the components of the system communicate in various ways.

The administration web application sends configuration data to Django, which is then stored as an Endpoint model within the database. The red arrow denotes configuration data, this is data used to configure the system, such as Endpoint information and dashboard configurations. Once the Endpoint configuration data is sent to the web application and database, the web application invokes automation scripts that install `node_exporter` on the IIoT device and reconfigures Prometheus to include the new device as a scraping target (as shown with the orange arrows). After the required software is installed and configured, the monitoring of metrics begins, and server metrics data (black arrow) is distributed throughout the system in real-time, showing data within the Grafana dashboards. These server metrics are exposed on the IIoT device on port 9100 with HTTP. Prometheus scrapes this data, and Grafana uses Prometheus as a data source for the dashboards.

5.3 Web application

This section discusses the different aspects of the web application, including how pages integrate, how the web application can be used and an explanation of features. The web application is the user's main point of interaction with the system.

5.3.1 Endpoint model

A Django model is the definitive source of information about your data. Each model maps to a single database table [51], an Endpoint model is created to allow Endpoints to be added to the database while invoking various automation scripts upon the creation of the individual Endpoint objects.

5.3.2 Add endpoint page

A simple Django template is used for the add endpoint page, this maps the Endpoint model fields to an HTML page, creating a simple way to add a new Endpoint object to the system.

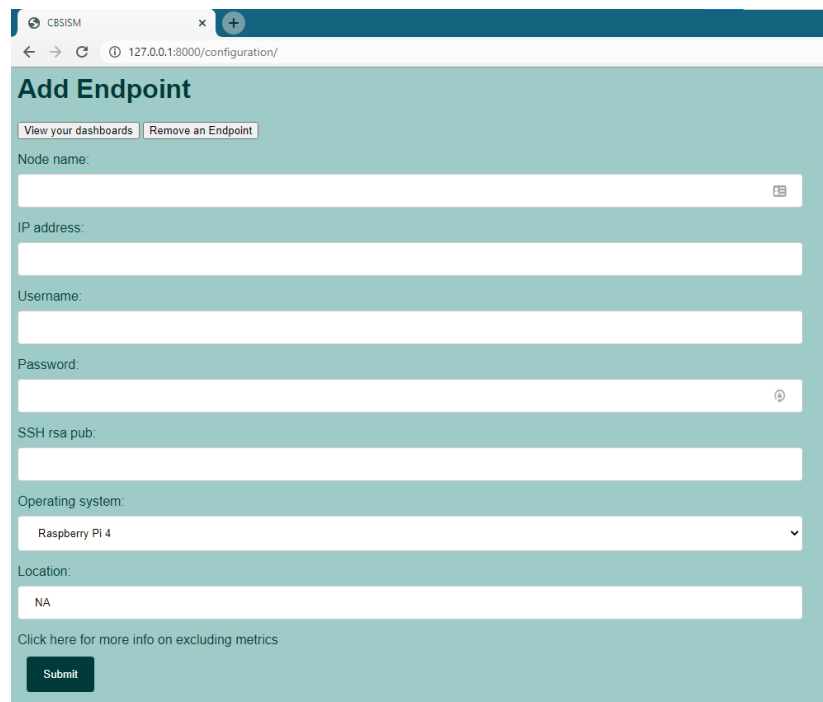


Figure 5 - Add endpoint web page

When an Endpoint is added through the form in Figure 5, the information is saved to a database, and the form information is passed to the automation scripts.

As seen in Figure 5, the add endpoint page has a button linking to the Dashboard page, allowing the user to quickly navigate to their dashboards after adding a new endpoint. Additionally, there is a button to navigate to the remove Endpoint page.

If the user enters correct details, the page buffers while the system installs Node Exporter on the Endpoint and adds the target information to Prometheus, then a confirmation message is displayed.

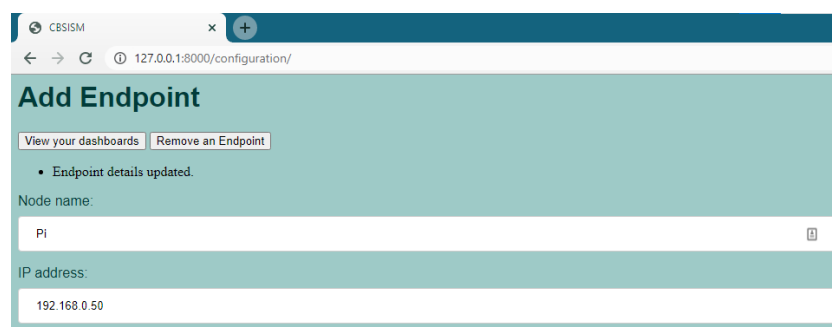


Figure 6 - Endpoint details updated

If a user tries to add a duplicate Endpoint, they are shown an error, and the form is not processed.

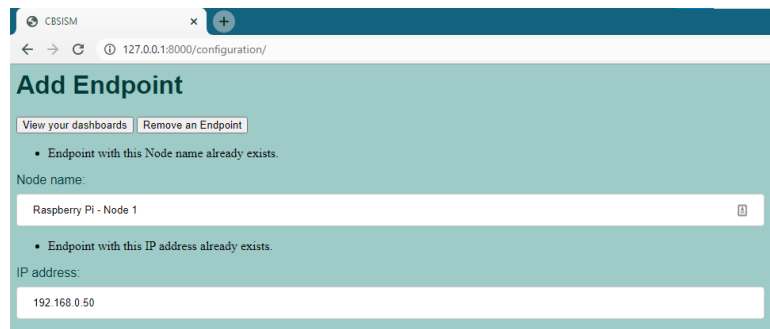
A screenshot of a web browser showing the 'Add Endpoint' page. The browser's address bar displays '127.0.0.1:8000/configuration/'. The page has a teal header with 'CBSISM' and a navigation bar with 'View your dashboards' and 'Remove an Endpoint'. A message states 'Endpoint with this Node name already exists.' Below this, the 'Node name' field contains 'Raspberry Pi - Node 1'. Another message states 'Endpoint with this IP address already exists.' Below this, the 'IP address' field contains '192.168.0.50'.

Figure 7 - Duplicate endpoint warning

5.3.3 Remove endpoint page

The user can navigate to the Endpoint removal page using the “Remove an Endpoint” button. This page allows a user to enter the IP address of an Endpoint and remove it from the system. By doing this, the Endpoint model object is removed from the Django database, and the monitoring target is removed from Prometheus and Grafana. When the removal process has taken place, the message “Endpoint removed” is shown.

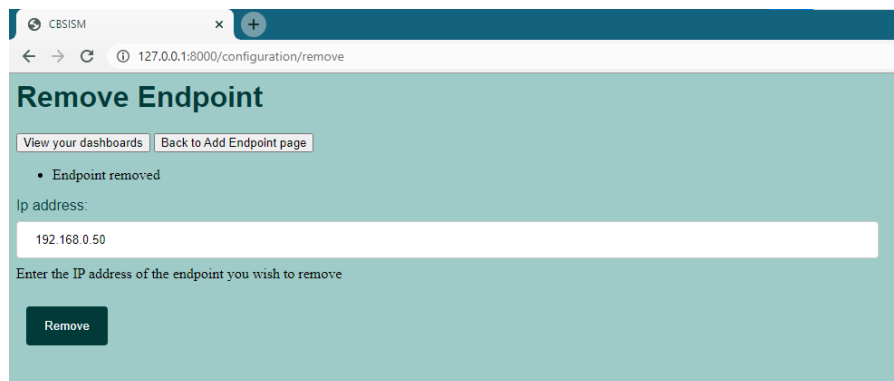
A screenshot of a web browser showing the 'Remove Endpoint' page. The browser's address bar displays '127.0.0.1:8000/configuration/remove'. The page has a teal header with 'CBSISM' and a navigation bar with 'View your dashboards' and 'Back to Add Endpoint page'. A message states 'Endpoint removed'. Below this, the 'Ip address' field contains '192.168.0.50'. A label reads 'Enter the IP address of the endpoint you wish to remove'. A 'Remove' button is at the bottom.

Figure 8 - Endpoint removal page

5.3.4 Dashboards

Below in Figure 9 is the Grafana dashboard showing the Node Exporter Full [37] dashboard. The user is able to switch between Endpoints using the “Job” drop-down box. As seen in Figure 9, the job “asus” is currently selected. This is a test device that has been used within the system. In the top right corner, there is the functionality to change the time window of the data (Figure 10), as well as the refresh rate (Figure 11).



Figure 9 - Grafana dashboard

Figure 9 shows a preconfigured dashboard [37]. However, the user has the ability to create and customise any dashboard they want, to present their data in a way that they find meaningful in the context of their use case.

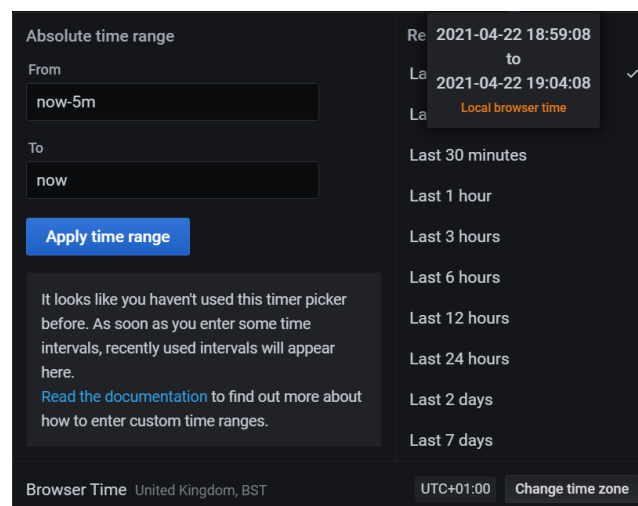


Figure 10 - Time range selection



Figure 11 - Refresh rate

5.4 Automation (back-end code)

This section explores two parts of the back-end code for this system, *install_NE_on_node.py* and *add_prometheus.py*.

install_NE_on_node.py is a Python script used to install the metric exporter (NodeExporter) on an endpoint. To achieve this, an SSH client is created using Paramiko [43], as well as an SCP client using the python package scp [44]. The script is invoked when a new endpoint is added to the system, using parameters from the form on the add endpoint web page. For the script to work, Paramiko requires the SSH RSA public key of the device it is connecting to. Therefore, there is a field in the add endpoint form for the key. It can easily be found on the endpoint device at `~/.ssh/ssh_host_rsa_key.pub`. Firstly, the script determines the operating system of the Endpoint, it then uses SCP to copy a bash install script (*install-amd64.sh* or *install-pi.sh*) to the Endpoint. Once the install script is copied over, the automation script connects to the endpoint with SSH and runs the install bash script.

The install bash script is simple, it uses curl to download the latest node exporter release from 'https://github.com/prometheus/node_exporter/releases', ensuring that the correct version is downloaded depending on the operating system of the endpoint, then configures and runs the Node Exporter service. "systemctl enable nodeexporter" is used to ensure the service is started at reboot. If the IIoT device is ever shut down, Node Exporter will automatically run on startup.

After the metric exporter is installed on the Endpoint has taken place, we need to add the new Endpoint to the monitoring service, Prometheus. Prometheus works by scraping a target in the format of *device_ip:9100*. It is possible to change the targets port, but we use the default node exporter port 9100 (it is imperative that this port is open on the firewall both on the host and the endpoints). The *add_prometheus.py* script is similar to *install_NE_on_node.py* in that it is ran using arguments from when a new endpoint is added to the system. Prometheus is run in the CBSISM as a Docker container. Containers are made from Docker images, which are created using a DockerFile. The Prometheus DockerFile is simple, it pulls Prometheus from a repository and copies a configuration file to the image. The configuration file (prometheus.yml) contains all the information regarding targets and scraping jobs, it is the only way to add new endpoints. The method *add_to_prometheus* in *add_prometheus.py* takes advantage of this by updating prometheus.yml with the new endpoint parameters and re-creates the Docker image while backing up the old one. Once the new Docker image contains the new targets, the container is restarted, and the new Endpoint is fully integrated into the monitoring system.

When removing an Endpoint from the system, we use the method *remove_from_prometheus* in *add_prometheus.py*; this uses the following command to remove the Endpoint target from Prometheus:

```
vim -e -c 'g/" + ip_address + "/.-3,.d' -c 'wq' prometheus.yml
```

The command removes the three lines associated with an Endpoint, then rebuilds and restarts Prometheus in a similar way to *add_to_prometheus*. Once this has run, the Endpoint is no longer monitored by Prometheus, therefore not showing any new data in Grafana.

5.5 Workflow examples

To demonstrate the functionality of this project, we will go through some example use cases of the system and show the workflow. The two typical actions that a user will take are:

- Adding an endpoint and viewing its data in the Dashboards
- Deleting an endpoint and seeing the Dashboard no longer showing real-time data

5.5.1 Adding an endpoint

Firstly, for this example, we use a clean system with no endpoints. As seen in Figures 12 and 13, no endpoints are present yet within the system:



Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://localhost:9090/metrics	UP	instance="localhost:9090" job="prometheus"	4.40s	9.013ms	

Figure 12 - Prometheus with no endpoints (By default Prometheus always has itself as a target) – this is not a user-facing UI, as the user will only access Grafana and the Web application

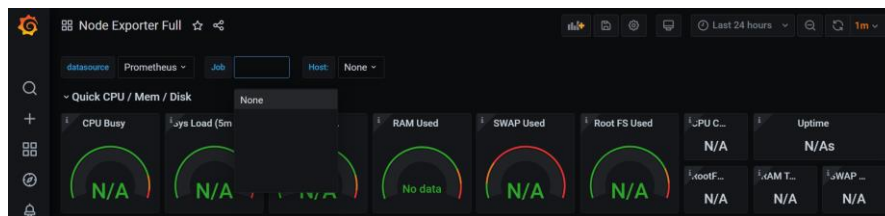
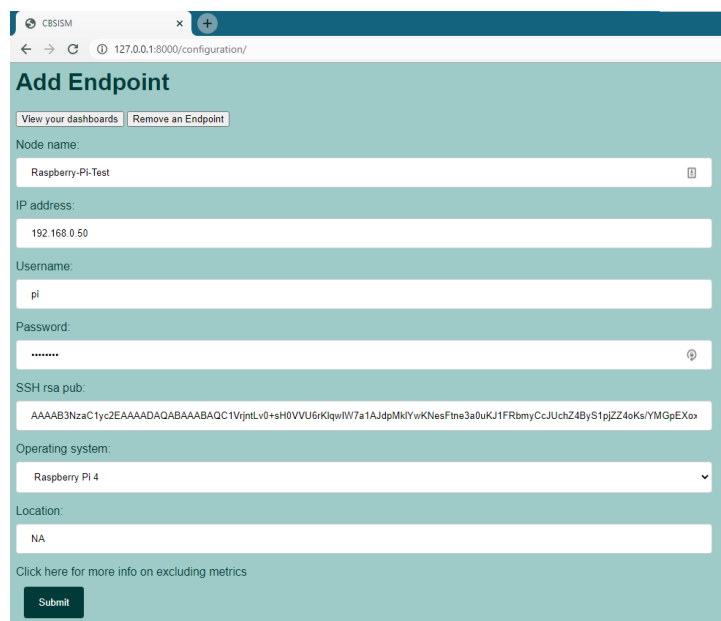


Figure 13 - Grafana with no endpoints

For this demonstration, we will use a Raspberry Pi 4b with Raspbian OS installed. This test device has the IP address 192.168.0.50 and is reachable by the host system.



Add Endpoint

View your dashboards Remove an Endpoint

Node name: Raspberry-Pi-Test

IP address: 192.168.0.50

Username: pi

Password:

SSH rsa pub: AAAAB3NzaC1yc2EAAAADAQABAAQCT1VjntLxv0+sH0VVU6rKlqwIW7a1AJdpMkIYwKNesFne3a0uKJ1FRbmyCcJuchZ4BySpjZZ4oKa/YMGpEXo

Operating system: Raspberry Pi 4

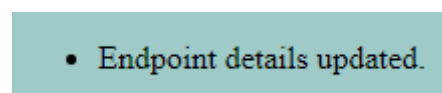
Location: NA

Click here for more info on excluding metrics

Submit

Figure 14 - Adding a test device to the system

As seen in Figure 14, we have filled the information of the Raspberry Pi into the add endpoint page. This is all we need to do to add the endpoint to the system. Once “Submit” is clicked, after a few seconds, we see the message:



• Endpoint details updated.

Figure 15 - Updated message

Meanwhile, on the server-side logs, we can see that NodeExporter is being installed on the new Endpoint, while the Prometheus container is rebuilt to include the new endpoint as a monitoring target:

```

this is the username pi
Connection and credentials verified
installing node
Output from server node_exporter-1.1.2.linux-armv7/LICENSE
Output from server node_exporter-1.1.2.linux-armv7/NOTICE
Output from server node_exporter-1.1.2.linux-armv7/node_exporter
Output from server install-done
done!
adding to prometheus
2021-05-04-prom
2021-05-04-prom
[+] Building 0.4s (7/7) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 37B
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load metadata for docker.io/prom/prometheus:latest
=> [internal] load build context
=> => transferring context: 1.06kB
=> CACHED [1/2] FROM docker.io/prom/prometheus
=> [2/2] COPY ./prometheus.yml /etc/prometheus/prometheus.yml
=> exporting to image
=> => exporting layers
=> => writing image sha256:ba51af99cd7612743f9351eabf24477da72a8c964880d536711b49d136692388
=> => naming to docker.io/library/prometheus
Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them
877bccc10ae267e9546eb464ce1d0557e46b2eb300a2f0d91a83c7a91ad5fe1d
prometheus updated

```

Figure 16 – Server-side output from back-end automation scripts, not visible to the user

If we refresh the Prometheus page, we can see the new Endpoint “Raspberry-Pi-Test” is now a target (this is facilitated by the `add_prometheus.py` script):

The screenshot shows the Prometheus Targets page with two targets listed. The first target is 'Raspberry-Pi-Test' with endpoint 'http://192.168.0.50:9100/metrics', state 'UP', and labels 'instance="192.168.0.50:9100"' and 'job="Raspberry-Pi-Test"'. The second target is 'prometheus' with endpoint 'http://localhost:9090/metrics', state 'UP', and labels 'instance="localhost:9090"' and 'job="prometheus"'. Both targets have a 'Last Scrape' time and 'Scrape Duration'.

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://192.168.0.50:9100/metrics	UP	instance="192.168.0.50:9100" job="Raspberry-Pi-Test"	2.668s	99.028ms	
http://localhost:9090/metrics	UP	instance="localhost:9090" job="prometheus"	12.31s	7.452ms	

Figure 17 - Prometheus updated with the new endpoint (again, this is not user-facing, however, it helps to demonstrate the flow within the system)

The user will not need to view the Prometheus page, as they can see the new endpoint automatically added to Grafana by clicking the “View your dashboards” button, as seen in Figure 14.

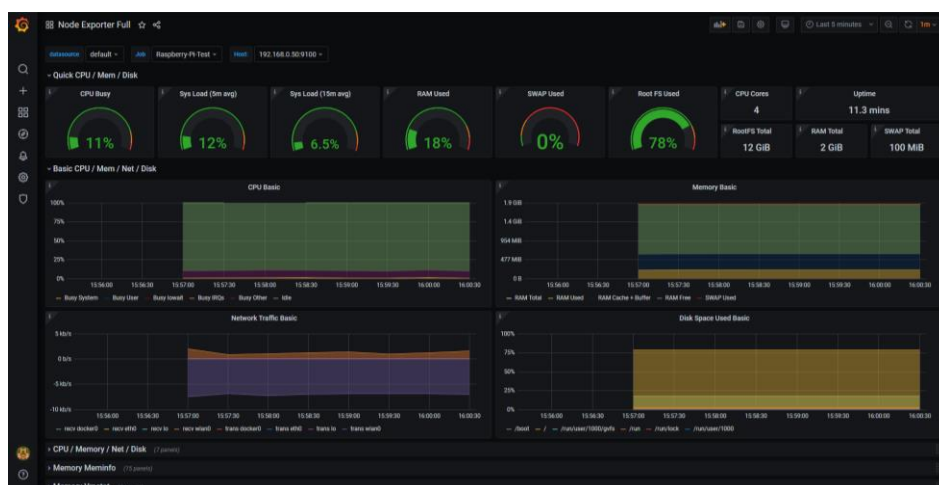


Figure 18 - Grafana updated with the new endpoint

In summary, adding an endpoint is extremely simple. The user simply has to fill in their Endpoint information into the Add Endpoint page and click submit. This saves a considerable amount of time compared to manually installing and configuring all of the components.

5.5.2 Deleting and endpoint

As seen in Figures 17 and 18, we know that the endpoint Raspberry-Pi-Test (192.168.0.50) is now added to the system. To remove it, we simply enter its IP Address into the Remove Endpoint page and click Remove.

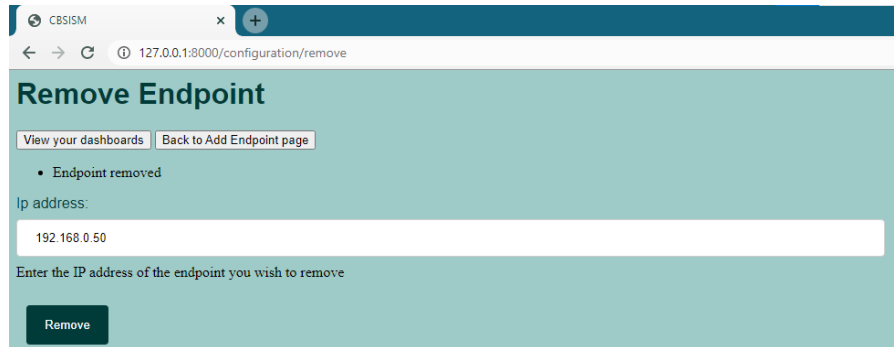


Figure 19 - Removing an endpoint

This runs the `remove_from_prometheus` method in `add_prometheus.py` to stop monitoring the endpoint. Now that the endpoint has been removed from the system, Grafana has stopped showing real-time data from 192.168.0.50. The historical data is still visible because we use a persistent volume for the Prometheus Docker container to enable the long-term storage of data.



Figure 20 - Grafana with removed endpoint, showing only historical data

5.6 Long term data solutions

This system is designed for cloud deployment. With this design decision comes a wide array of options for storage solutions. Users may wish to store large amounts of historical monitoring data to use for purposes such as machine learning and pattern detection. To store long periods of time series data can take up considerable storage; therefore, using a service such as Amazon S3 [50] may be desirable. Using Amazon S3 will allow users to utilise the data within use cases such as data lakes, backups, archiving and big data analytics [50]. In addition to increasing scalability and durability of data.

Integration with Amazon S3 is simple for any user that is already an AWS user and customer. Although this project is not integrated with AWS or S3, there are sample scripts located within the ‘*s3-scripts*’ folder that provide examples of how a user could quickly adapt the system to integrate with Amazon S3. *backup-to-s3.sh* is a simple proof of concept bash script that creates an S3 bucket, creates a database dump (backup) and uploads the backup to S3. This shows how simple it is to integrate such procedures within the system. A script of this nature could be running as a cron job to back up data at regular intervals.

5.7 Deployment

It is straightforward to deploy this system. To do this, we run the *install-CBSISM.sh* bash script on the host server. This will install all required software and run the application. The install script is built for Ubuntu environments, as this is what was used for development. However, it is simple to adapt the script for other operating systems by changing the package manager and the Docker download URL.

Firstly, the script updates the operating system. Next, the packages are installed to allow the use of the Docker repository before adding the Docker GPG key. Once the repository is added, we install Docker. After installing Docker, we install Python 3 and pip3 (Python package manager), needed for the web application and back end scripts. Using pip, we install virtualenv, which we use so we can install all Python dependencies with the requirements.txt file. Finally, we create or pull Docker images for Grafana, Prometheus and TimescaleDB and run them before running the Django web application.

The *install-CBSISM.sh* script performs all of this for us, removing the need for tedious manual configuration associated with many other monitoring systems.

5.8 Containerisation

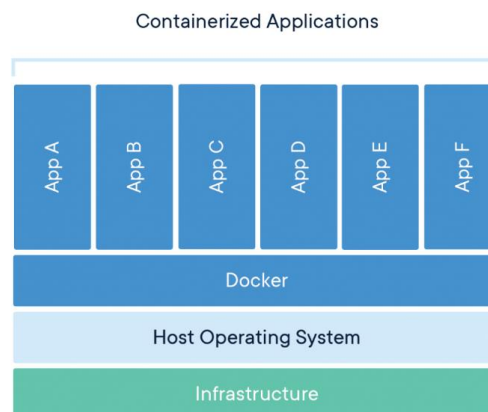


Figure 21 - Docker architecture [6]

“Containers are a standardised unit of software that allows developers to isolate their app from its environment, solving the “it works on my machine” headache” [52]. As seen in Figure 21, many Apps/Containers can run on top of a Docker deployment. Containers are created from Docker Images. Prometheus, Grafana and TimescaleDB have Docker images that are easy to use and deploy, this makes the deployment of all CBSISM components very simple.

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
f78a2a4b4410	prometheus	"/bin/prometheus --c..."	2 weeks ago	Up 14 seconds	0.0.0.0:9090->9090/tcp	2021-03-29-prom
99b95aff78c7	grafana/grafana	"/run.sh"	2 months ago	Up 13 seconds	0.0.0.0:3000->3000/tcp	romantic_babbage
96f6f27d414f	timescale/timescaledb:1.7.4-pg12	"docker-entrypoint.s..."	5 months ago	Up 12 seconds	0.0.0.0:5432->5432/tcp	timescaledb

Figure 22 - Docker processes

Figure 22 shows the containers used in the CBSISM. The Prometheus DockerFile is updated via python automation scripts whenever an endpoint is added to the system (hence the container name including the date of the latest revision), the image is then recreated, and the container is restarted, the overall

downtime of the container is minimal. The Grafana and TimescaleDB containers can stay the same and run without changing the entire deployment lifecycle.

By using Docker, the whole system can be deployed in seconds, using a simple script to create all necessary containers. Because Docker solves the “it works on my machine” issue, the system can be deployed on any Linux server (including cloud environments), reducing compatibility issues, and simplifying installation. It is guaranteed that every deployment of the system will be the same due to the nature of Docker containers being isolated from their environment. If Docker were not used for this project, the components would have to be installed on the host server as traditional applications, this can cause issues when there are differences in OS, as well as resulting in an overall less agile system.

5.9 Cloud deployment

Due to the containerised architecture of this project, it is suitable and compatible with cloud environments. Providing that the IIoT endpoints are reachable by the cloud server (firewall permissions and static public IP address for the devices), deployment of the system in the cloud is trivial.

Using AWS as an example cloud provider, the below diagram details a typical cloud deployment of the CBSISM:

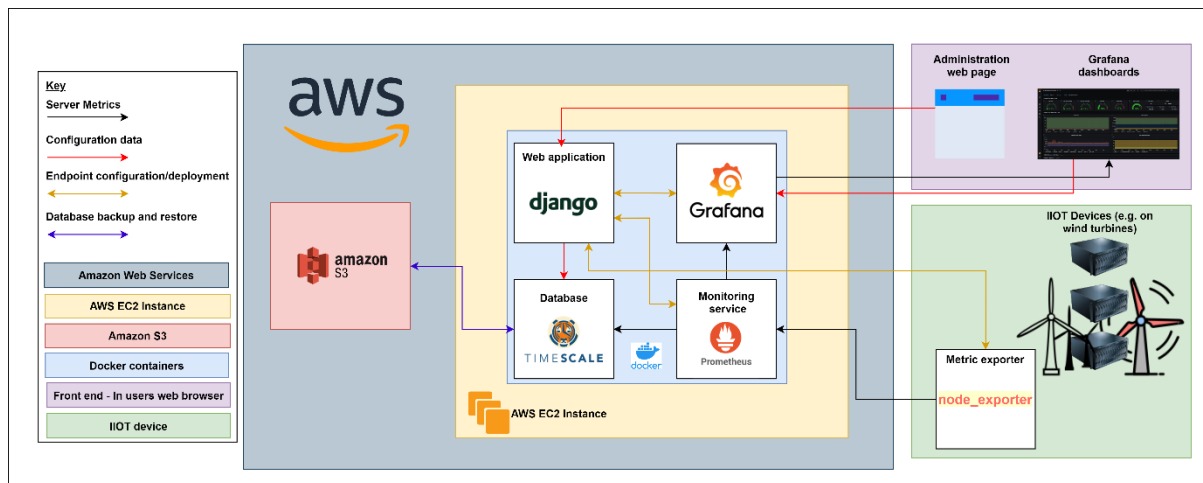


Figure 23 - CBSISM in AWS

Figure 23 shows a standard AWS EC2 instance [53] used for hosting the docker containers, with AWS S3 for database backup and restore.

Due to budget constraints (Public static IP for test devices, AWS EC2 and S3 costs), the project has not been deployed in the cloud; however, it is ready for deployment and is considered “cloud-ready” due to its containerised architecture providing portability with minimal compatibility issues.

5.10 System requirements

There are several prerequisite requirements to run the CBSISM. These can be broken down into two categories, host server requirements (Cloud) and endpoint requirements (IIoT devices).

5.10.1 Host requirements

The host server is where the back-end operations and web application hosting take place. This can be visualised as the yellow box in Figure 23. Ideally, this is a host located within a cloud provider such as Amazon Web services (AWS) or Microsoft Azure. However, for development and test purposes, this can be a simple Linux server. The server must have an internet connection, as well as access to a superuser with the ability to manage the firewall and permissions. The system has been tested and built using the Ubuntu OS; however, it is expected to work on any Debian or RHEL based distribution.

Some of the software used in this system is Docker-based, meaning they run in Docker containers on the Docker platform (see Figure 21). The host must have the latest Docker release installed and running.

The web application is built using Django, which must be installed along with the latest Python 3 release.

The installation script provides the installation of software and dependencies.

5.10.2 Endpoint requirements

The endpoint must be running one of the following supported operating systems:

- Ubuntu
- CentOS
- Fedora IOT
- Raspberry Pi 4 (Raspbian)

The reason that the operating system must be specified is that there are different versions of NodeExporter for different operating systems and architectures. By selecting the OS of the endpoint, the automation script knows which version of the metric exporter to install to the endpoint.

The endpoint must have an internet connection, with a running SSH server allowing connections. The SSH Host RSA Public key must be added to the known hosts of the host server, enabling the connection between the two systems, allowing the host server to verify the identity of the endpoint, improving security.

The credentials for a user with passwordless sudo must be used when adding the endpoint connection in the CBSISM.

5.11 Scalability

The scalability of this system is important due to the differences in the size of potential deployments. The overall scalability is determined by the ability of the individual components to scale in relation to the number of endpoints within the system. In this subsection, we will explore the different components and their limitations or benefits in regard to scalability.

5.11.1 Prometheus

It is expected that a single Prometheus server is able to handle a thousand targets [54], this is more than required for the scope of this project, as our primary use case is UK offshore wind farms with an average of 59 turbines (Appendices A). There are many options to improve the scalability of Prometheus if a very large number of endpoints is required, such as splitting by use (running multiple Prometheus servers) and horizontal sharding [54]. However, this was not within the scope of this project. Additionally, it is possible to adjust the sample rate within Prometheus to reduce the frequency of data scraping, this can reduce the computing power needed for monitoring, however results in less precise data.

5.11.2 Grafana

Grafana is considered lightweight in the use of memory and CPU, with the minimum requirements being 255MB RAM and 1 CPU [55], it is expected that there are no issues concerning the scalability of Grafana.

5.11.3 Database

The user should ensure that necessary provisions for storage requirements are in place within this system. By default, this systems database solution will be suitable for our use case of 59 devices. If long term data storage is required, the user should consider options such as Amazon S3 (section 5.6). Time-series databases such as TimescaleDB are suitable for scaling time-series data, this is ideal if it is expected that a large number of endpoints (100+) will be used within the system, however for this project, it was

only necessary to use the default Prometheus database mounted to a Docker persistent volume. As previously discussed, switching the time-series data storage from Prometheus to TimescaleDB within this system is trivial and can be achieved with the Prometheus PostgreSQL Adapter [48] or the new tool Promscale [49].

6 Testing

During the development of this project, we used two test devices:

- Linux raspberrypi-4B 5.4.83-v7l+ running Raspbian GNU/Linux (pi@192.168.0.50)
- Asus Zenbook UX301LA running Ubuntu 18.04.2 LTS (test@192.168.0.90)

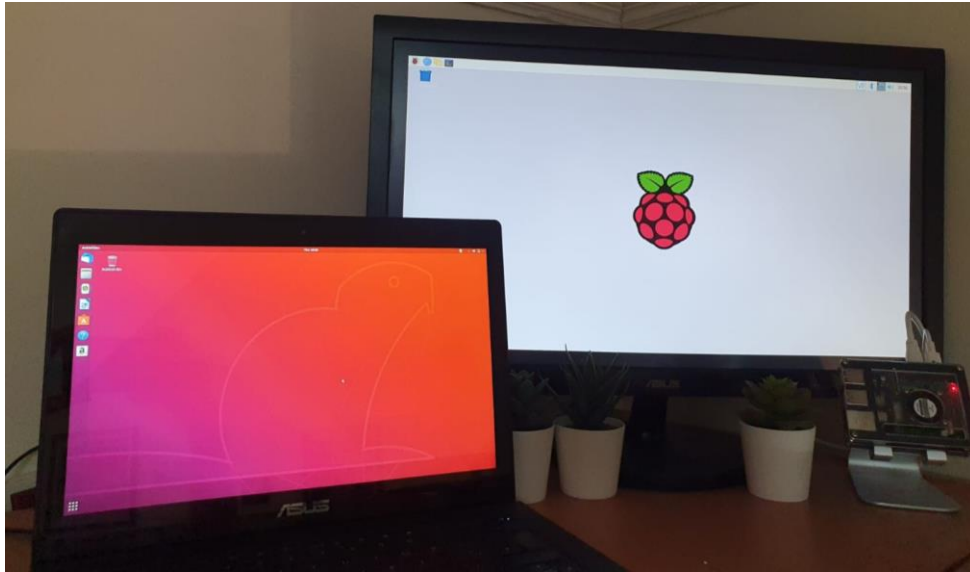


Figure 24 - Test devices

These devices acted as test endpoints for the system, they are Linux based, which makes them comparable to typical IIoT devices. The devices enabled testing to take place throughout development, using them to test the back-end code and connection of endpoints to the system. As shown with the example workflows in Section 5.5, all use cases of the system were tested with these two devices. When code was developed or changed, the process of adding the device, viewing the dashboards, and removing the device was taken place to ensure functionality is present and accurate.

Additionally, the *tests.py* script was used throughout development as a way to test functionality. This utilises Django's testing framework and enables the automation of invoking methods and functions without interacting with the user interface. Using the Django testing framework during development, it is assured that all functionality is in working order.

The system is fully functional when tested with the two testing devices. However, one area of this project that did not come to fruition is using and testing the final solution on a large number of endpoints. Due to budget and time constraints, it was not possible to run the solution with a large number of test devices in the way that the system is designed for. Because of this, the scalability of the system was researched and discussed in Section 5.11. However, it would have been beneficial to the projects testing process to evaluate the system with a larger number of real IIoT devices.

7 Conclusion

7.1 Future developments

Future developments are areas within the project that were considered out of scope but would benefit the solution if implemented in the future, given additional resources.

7.1.1 Kubernetes

Kubernetes is an open-source platform for managing containerised environments and services [56]. In this project, we use Docker for containerisation. However, this can be extended to Kubernetes to provide benefits such as storage orchestration, automated rollbacks, and automatic bin packing (management of nodes to make the best use of resources). Another feature of Kubernetes is self-healing. Self-healing is the process of restarting, replacing, and killing failed containers [56], this ensures that a containerised system is fully stable and functional. A future development of this system is to use Kubernetes to orchestrate containerisation, providing numerous benefits such as stability and security in a large-scale cloud deployment.

7.1.2 Machine learning

Machine learning (ML) can utilise the large amount of time-series data gathered by monitoring the IIoT devices to generate predictive models used to predict failures and issues with the IIoT devices. This is beyond the scope of this project due to the specialist Machine Learning knowledge required, however it would be a beneficial future development to include ML assisted prediction models with the CBSISM to facilitate business decisions to reduce maintenance costs and time.

7.1.3 Auto determination of endpoint OS

A development of this project that would improve usability for the user is the ability to auto determine the operating system of an Endpoint, removing the need for a user to select their operating system when adding the Endpoint to the system. If the user selects the wrong OS when adding the endpoint into the system, the wrong Node Exporter binary will be installed to the device, breaking the installation of Node Exporter. A future development would be to remove the option to select the operating system manually and introduce code to enable the detection of the operating system through reading operating system version information on the device before choosing which Node Exporter to install.

7.1.4 Exclusion of metrics

Initially, the ability to allow users to select which specific metrics to gather was considered, however, it did not seem necessary within the scope of this project due to the ability to use Grafana to customise which metrics are shown in dashboards. A future development is to enable this functionality, to allow users to fine-tune their systems only to collect essential metrics.

7.1.5 Deploying the system in the cloud

As discussed in section 5.9, this system is suitable and ready for cloud deployment. Unfortunately, due to the budget constraints of this project, it was considered out of the scope of this project to deploy the system in a cloud environment practically. It is a future development to deploy this project in the cloud to simulate a real-world use case of the system, combined with a large number of endpoints for testing.

7.2 Critical Evaluation

The majority of planned requirements in this project were met, however several requirements were not complete to their full extent. Below evaluates and reflects upon these incomplete requirements.

Requirement F8 is concerning the ability to predict possible future issues within the system based upon gathered data. Although this was not achieved within this project, it was considered a future development by introducing machine learning features.

Requirement NF3 discusses the need for scalability. Although this was not addressed in the project in the form of a technical solution, scalability is discussed in the implementation section in the form of research and evaluation.

NF7 is a requirement for self-healing, this was a desirable requirement that did not come to fruition, however, it is put forward as a future development through the use of Kubernetes.

NF9 is a requirement of this project that was not complete to the desired extent. This requirement planned the use of Transport Layer Security (TLS) [57]. Throughout this project, we use HTTP for communication between Prometheus and Node Exporter, as this is the default protocol for those tools. HTTP is not the most secure protocol. The Prometheus security documentation states that Transport Layer Security (TLS) will be included in future versions of Prometheus, which is needed to enable HTTPS. Near the end of development, it was discovered that it is currently possible to create self-signed RSA key certificates to use between Prometheus and Node Exporter to enable TLS [58]. However, this was not achieved in this project due to difficulty incorporating this functionality with the Prometheus Docker image.

All requirements for this project have either been complete or otherwise addressed critically.

7.3 Conclusion

In conclusion, the success of this project can be determined by the solutions ability to fulfil its original aims and objectives. The aims and objectives of this project have been achieved. As previously mentioned, the overall aim of this project was to use a curated selection of technologies to design and develop a deployable cloud-based solution to IIoT server monitoring, allowing IIoT servers/devices to be monitored easily from a cloud-based central server. This project achieves this aim by providing a way for a business to quickly deploy a monitoring system that can be interacted with entirely through a user-friendly web interface. In addition to the aims, the objectives within this project have been fully met. The solution put forward in this project is designed to save businesses valuable time and money. The process of deploying this system and adding IIoT devices is considerably simpler, faster, and more accessible than competing solutions and methodologies for IIoT server monitoring.

8 Bibliography

- [1] IBM, “What is the IOT,” 17 November 2016. [Online]. Available: <https://www.ibm.com/blogs/internet-of-things/what-is-the-iot/>. [Accessed April 2021].
- [2] S. S. K. R. J. a. P. G. A. Dorri, “Blockchain for IoT security and privacy: The case study of a smart home,” *2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pp. 618-623, 2017.
- [3] MarketWatch, “Industrial Internet of Things (IIoT) Market 2020 Size,Share Global Growth Analysis, Gross Margin Analysis, Industry Leading Players Update, Development History, Business Prospect and Industry Research Report 2023,” 24 August 2020. [Online]. Available: <https://www.marketwatch.com/press-release/industrial-internet-of-things-iiot-market-2020-sizeshare-global-growth-analysis-gross-margin-analysis-industry-leading-players-update-development-history-business-prospect-and-industry-research-report-2023-2020-08>. [Accessed 5 November 2020].
- [4] Gartner, “Gartner Forecasts Worldwide Public Cloud Revenue to Grow 17.5 Percent in 2019,” Gartner, 2 April 2019. [Online]. Available: <https://www.gartner.com/en/newsroom/press-releases/2019-04-02-gartner-forecasts-worldwide-public-cloud-revenue-to-g>. [Accessed 5 November 2020].
- [5] A. Balalaie, A. Heydarnoori and P. Jamshidi, “Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture,” *IEEE Software*, vol. 33, no. 3, pp. 42-52, 2016.

- [6] Docker, "What is a container?," 2020. [Online]. Available: <https://www.docker.com/resources/what-container>. [Accessed 10 November 2020].
- [7] Glassdoor, "Senior Systems Administrator Salaries," [Online]. Available: https://www.glassdoor.co.uk/Salaries/senior-systems-administrator-salary-SRCH_KO0,28.htm. [Accessed December 2020].
- [8] BCS, "BCS Code of Conduct," 2020. [Online]. Available: <https://www.bcs.org/membership/become-a-member/bcs-code-of-conduct/>. [Accessed 5 November 2020].
- [9] H. Boyes, B. Hallaq, J. Cunningham and T. Watson, "The industrial internet of things (IIoT): An analysis framework," *Computers in Industry*, vol. 101, pp. 1-12, 2018.
- [10] Y. Lu, P. Witherell and A. Jones, "Standard connections for IIoT empowered smart manufacturing," in *Manufacturing Letters Volume 26*, United States, National Institute of Standards and Technology, 2020, pp. 17-20.
- [11] G. F. Savari, V. Krishnasamy, J. Sathik, Z. M. Ali and S. H. Abdel Aleem, "Internet of Things based real-time electric vehicle load forecasting and charging station recommendation," in *ISA Transactions Volume 97*, 2019, pp. 431-447.
- [12] S. Badugu, K. Srikanth and I. Narayana, "IoT for Healthcare," *International Journal of Science and Research (IJSR)*, vol. 5, pp. 2319-7064, 2016.
- [13] M. S. Hossain and G. Muhammad, "Cloud-assisted Industrial Internet of Things (IIoT) – Enabled framework for health monitoring," *Computer Networks*, vol. 101, pp. 192-202, 2016.
- [14] K. L. In Lee, "The Internet of Things (IoT): Applications, investments, and challenges for enterprises," in *Business Horizons*, 2015, pp. 431-440.
- [15] F. R. Y. Y. T. V. C. M. L. S. M. Liu, "Performance Optimization for Blockchain-Enabled Industrial Internet of Things (IIoT) Systems: A Deep Reinforcement Learning Approach," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 6, pp. 3559-3570, 2019.
- [16] T. FUKATSU and M. HIRAFUJI, "Field Monitoring Using Sensor-Nodes with a Web Server," *Journal of Robotics and Mechatronics*, vol. 17, no. 2, pp. 164-172, 2005.
- [17] J. S. a. Y. Pingle, "IOT in agriculture," in *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, 2016, pp. 1456-1458.
- [18] C. Roblee, V. Berk and G. Cybenko, "Implementing Large-Scale Autonomic Server Monitoring Using Process Query Systems," in *Second International Conference on Autonomic Computing (ICAC'05)*, Seattle, WA, USA, 2005.
- [19] S. P. V. R. Gore, "Big Data challenges in smart Grid IoT (WAMS) deployment," in *2016 8th International Conference on Communication Systems and Networks (COMSNETS)*, Bangalore, India, 2016, pp. 1-6.
- [20] H. Geng, "SCADA FUNDAMENTALS AND APPLICATIONS IN THE IoT," in *Internet of Things and Data Analytics Handbook*, Wiley, 2017, pp. 283-293.

- [21] H. A. K. S. A. Sajid, "Cloud-Assisted IoT-Based SCADA Systems Security: A Review of the State of the Art and Future Challenges," *IEEE Access*, vol. 4, pp. 1375-1384, 2016.
- [22] Y. K. A. E. A. Shahzad, "Secure IoT Platform for Industrial Control Systems," *2017 International Conference on Platform Technology and Service (PlatCon)*, pp. 1-6, 2017.
- [23] F. A.-T. a. M. Abujubbeh, "IoT-enabled smart grid via SM: An overview," in *Future Generation Computer Systems*, vol. 96, Turkey, 2019, pp. 579-590.
- [24] T. Tarui, T. Tanaka, K. Mizuno and K. Naono, "Performance monitoring system, bottleneck detection method and management server for virtual machine system". US Patent US20100268816A1, 2009.
- [25] A. Nasuto, G. Cassone and D. Gotta, "Method and system for monitoring performance of a client-server architecture". AT US CN WO EP Patent US7933988B2, 2004.
- [26] W. Zeng and Y. Wang, "Design and Implementation of Server Monitoring System Based on SNMP," *International Joint Conference on Artificial Intelligence*, vol. 2009, pp. 680-682, 2009.
- [27] AppDynamics, "IoT Monitoring: Connected Devices and Internet of Things Applications," [Online]. Available: <https://www.appdynamics.com/product/end-user-monitoring/internet-of-things>.
- [28] AppDynamics, "Cognition Engine," Cisco, [Online]. Available: <https://www.appdynamics.com/product/application-performance-monitoring/cognition-engine>. [Accessed April 2021].
- [29] Baseapp, "Server Monitoring System with SwarmSense IoT Platform," [Online]. Available: <https://www.baseapp.com/swarmsense/server-monitoring-system-swarmsense-iot-platform/>. [Accessed 5 November 2020].
- [30] PandoraFMS, [Online]. Available: <https://pandorafms.com/iot-monitoring/>. [Accessed November 2020].
- [31] PandoraFMS, "Home," [Online]. Available: <https://pandorafms.com/>. [Accessed December 2021].
- [32] Nagios, "Nagios XI - Easy Network, Server Monitoring and Alerting," Nagios, 2020. [Online]. Available: <https://www.nagios.com/products/nagios-xi/#pricing>. [Accessed 5 November 2020].
- [33] M. Glinz, "On Non-Functional Requirements," *15th IEEE International Requirements Engineering Conference*, pp. 21-26, 2007.
- [34] I. Sommerville, "Domain requirements," St Andrews, 2008. [Online]. Available: <https://ifs.host.cs.st-andrews.ac.uk/Books/SE9/Web/Requirements/DomainReq.html>. [Accessed March 2021].
- [35] "What is Open Source," <https://opensource.com/>, [Online]. Available: <https://opensource.com/resources/what-open-source>. [Accessed April 2021].
- [36] Grafana, "Grafana," [Online]. Available: <https://grafana.com/oss/grafana/>. [Accessed November 2020].

- [37] rfrail, “Node Exporter Full,” Grafana Labs, [Online]. Available: <https://grafana.com/grafana/dashboards/1860>. [Accessed December 2020].
- [38] Prometheus, “Node Exporter,” [Online]. Available: https://github.com/prometheus/node_exporter.
- [39] collectd, “collectd,” [Online]. Available: <https://collectd.org/>. [Accessed November 2020].
- [40] Prometheus, “Comparison,” [Online]. Available: <https://prometheus.io/docs/introduction/comparison/>. [Accessed 8 November 2020].
- [41] Prometheus, “Prometheus storage,” [Online]. Available: <https://prometheus.io/docs/prometheus/latest/storage/>. [Accessed December 2020].
- [42] Python, “Python,” [Online]. Available: <https://www.python.org/>. [Accessed November 2020].
- [43] Paramiko, “Paramiko,” [Online]. Available: <http://www.paramiko.org/>. [Accessed March 2021].
- [44] <https://github.com/jbardin>, “SCP,” [Online]. Available: <https://github.com/jbardin/scp.py>. [Accessed October 2020].
- [45] Pallets, “Flask,” [Online]. Available: <https://palletsprojects.com/p/flask/>. [Accessed November 2020].
- [46] Django, “The Web framework for perfectionists with deadlines | Django,” 2020. [Online]. Available: <https://www.djangoproject.com/>. [Accessed 5 November 2020].
- [47] TimescaleDB, [Online]. Available: <https://www.timescale.com/>. [Accessed November 2020].
- [48] Timescale, “Getting Started with Prometheus and TimescaleDB,” Timescale, [Online]. Available: <https://docs.timescale.com/latest/tutorials/prometheus-adapter>. [Accessed December 2020].
- [49] Timescale, “Promscale,” [Online]. Available: <https://github.com/timescale/promscale>. [Accessed April 2021].
- [50] Amazon AWS, “Amazon S3,” [Online]. Available: <https://aws.amazon.com/s3/>. [Accessed April 2021].
- [51] Django, “Models,” Django, [Online]. Available: <https://docs.djangoproject.com/en/3.1/topics/db/models/>. [Accessed April 2021].
- [52] Docker, “Why docker,” Docker, [Online]. Available: <https://www.docker.com/why-docker>. [Accessed 4 2021].
- [53] Amazon, “AWS EC2,” [Online]. Available: <https://aws.amazon.com/ec2/?ec2-whats-new.sort-by=item.additionalFields.postDateTime&ec2-whats-new.sort-order=desc>. [Accessed Feb 2021].
- [54] Robust Perception, “Scaling and federating prometheus,” [Online]. Available: <https://www.robustperception.io/scaling-and-federating-prometheus>. [Accessed April 2021].
- [55] Grafana, “Requirements,” [Online]. Available: <https://grafana.com/docs/grafana/latest/installation/requirements/>. [Accessed April 2021].

- [56] Kubernetes, “What is kubernetes?,” [Online]. Available: <https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/>. [Accessed May 2021].
- [57] E. R. T. Dierks, “The Transport Layer Security (TLS) Protocol,” [Online]. Available: <https://tools.ietf.org/html/rfc5246>. [Accessed April 2021].
- [58] Prometheus, “TLS Encryption,” [Online]. Available: <https://prometheus.io/docs/guides/tls-encryption/>. [Accessed 30 April 2021].

9 Appendix

A. Operational offshore UK windfarms research

The table below contains all operational offshore windfarms in the UK, this data was used to build use cases for this project and to calculate possible deployment sizes. The data was gathered using <https://www.renewableuk.com/page/UKWEDSearch> with search conditions “Operational” and “Offshore”.

NAME	TURBINES	PROJECT (MW)	TURBINE (MW)	YEAR	REGION
Barrow	30	90	3	2006	North West
Beatrice	84	588	7	2019	Scotland
Blyth Offshore Wind Demonstration Project (Phase 1)	5	41.5	8.3	2018	North East
Burbo Bank	25	90	3.6	2007	North West
Burbo Bank Extension	32	257.92	8.06	2017	North West
Dudgeon	67	402	6	2017	East of England
East Anglia ONE	102	714	7	2020	East of England
European Offshore Wind Deployment Centre (EOWDC, Aberdeen Bay)	11	93.2	8.4	2018	Scotland
Galloper	56	353	6.3	2018	East of England
Greater Gabbard	140	504	3.6	2012	East of England
Gunfleet Sands 1 & 2	48	172.8	3.6	2010	East of England
Gunfleet Sands Demonstration Project	2	12	6	2013	East of England
Gwynt Y Mor	160	576	3.6	2015	North Wales

Hornsea Project One	174	1218	7	2020	Yorkshire & Humber
Humber Gateway	73	219	3	2015	Yorkshire & Humber
Hywind Scotland Pilot Park	5	30	6	2017	Scotland
Kentish Flats	30	90	3	2005	South East
Kentish Flats Extension	15	49.5	3.3	2015	South East
Kincardine Phase 1	1	2	2	2018	Scotland
Levenmouth Demonstration Turbine (Energy Park Fife)	1	7	7	2015	Scotland
Lincs	75	270	3.6	2013	East Midlands
London Array Phase One	175	630	3.6	2013	South East
Lynn and Inner Dowsing	54	194.4	3.6	2009	East Midlands
North Hoyle	30	60	2	2004	North Wales
Ormonde	30	150	5	2012	North West
Race Bank	91	573	6	2018	East of England
Rampion	116	400.2	3.45	2018	South East
Rhyl Flats	25	90	3.6	2009	North Wales
Robin Rigg	58	174	3	2010	Scotland
Scroby Sands	30	60	2	2004	East of England
Sheringham Shoal	88	316.8	3.6	2012	East of England
Teesside	27	62.1	2.3	2013	North East
Thanet	100	300	3	2010	South East
Walney 1	51	183.6	3.6	2011	North West
Walney 2	51	183.6	3.6	2012	North West
Walney Extension East	47	329	7	2018	North West
Walney Extension West	40	330	8.25	2018	North West
West of Duddon Sands	108	388.8	3.6	2014	North West
Westermest Rough	35	210	6	2015	Yorkshire & Humber
TOTAL	2292				
Average turbines per farm	58.76923077				