

Bachelor's Degree in Computer Science and Engineering
2019-2020

Bachelor Thesis

“Combining Edge Computing and Data Processing with Kubernetes”

Jaime de Esteban Uranga

Tutor
David E. Singh
Colmenarejo, Madrid
July 2020



ABSTRACT

The objective of this thesis is to explore and expand on cutting edge concepts that have been introduced in the coursework during this bachelor's degree. In particular, we will be going into the world of distributed and "Cloud Computing", including the use of "Internet of Things" devices. We will also demonstrate our use case scenario implementing "Artificial Intelligence" concepts, including building and configuring a Neural Network.

In more detail, we explore how a computer cluster is built, scaling an application throughout an array of connected computers, building such applications on containers, and deploying them into the cluster. To build our use case scenario we will be developing an intelligent temperature control system that uses AI to determine the future temperature and using that prediction to reduce network congestion. This temperature control system and its other supporting applications will be run on the computer cluster and its performance will be evaluated.

Keywords: Cloud Computing, Edge Computing, Internet of Things, Machine Learning, Artificial Intelligence

DEDICATION

My dedication goes to all the professors that have shown passion for what they teach, their interest in transmitting their knowledge and the many hours preparing lectures. In particular I'd like to thank David Expósito Singh for being welcome to new ideas and additions to this very project and for being helpful in the entire year this project took place.

I'd also like to give a special thanks to the members of the international student's office who were always helpful and willing to offer support in my year of studies abroad.

TABLE OF CONTENTS

1	INTRODUCTION.....	1
1.1	MOTIVATION	1
1.2	OBJECTIVES.....	1
1.3	DOCUMENT STRUCTURE	2
2	STATE OF THE ART	2
2.1	CLOUD COMPUTING AND EDGE COMPUTING.....	2
2.2	MACHINE LEARNING AND NEURAL NETWORKS.....	5
2.3	INTERNET OF THINGS (IoT).....	6
3	DEVELOPMENT.....	8
3.1	ARCHITECTURE OVERVIEW.....	8
3.2	INTRODUCTION TO KUBERNETES.....	9
3.3	BUILDING THE SENSORS	17
3.4	INTRODUCING OUR TEMPERATURE SENSOR	18
3.5	INTRODUCING THE ESP PYTHON PROGRAM	19
3.6	MYSQL DATABASE	21
3.6.1	<i>CurrentTemp table</i>	22
3.6.2	<i>ProcTemp table</i>	23
3.7	APACHE WEB SERVER.....	24
3.7.1	<i>Proctemp.php</i>	25
3.7.2	<i>Proctemp2.php</i>	25
3.8	FEEDING THE DATABASE FROM EXTERNAL SOURCES.....	28
3.9	INTRODUCING OUR NEURAL NETWORK.....	32
3.9.1	<i>Neural Network architecture</i>	34
3.10	CLUSTER SETUP	40
3.10.1	<i>Server (master) setup</i>	41
3.10.2	<i>Worker setup</i>	42
3.10.3	<i>Deploying applications</i>	43
3.10.4	<i>ESP board setup</i>	44
3.10.5	<i>Temperature sensor setup</i>	45
3.10.6	<i>Building the complete sensor program in Python</i>	46
3.10.7	<i>Apache web server scripts</i>	49
3.10.8	<i>Building the weather prediction service</i>	53
3.10.9	<i>Building a Docker container</i>	56

3.10.10	<i>Deploying a Container to the Kubernetes Cluster</i>	58
3.11	BUILDING THE NEURAL NETWORK IN PYTHON	61
3.11.1	<i>Main thread</i>	62
3.11.2	<i>Secondary thread: TrainNN</i>	64
3.11.3	<i>Secondary thread: getPredictions</i>	70
4	PROPOSAL	72
4.1	METHODOLOGY AND PHASES	72
4.2	USE CASE ANALYSIS	73
4.3	REQUIREMENTS	75
4.3.1	<i>Functional requirements</i>	76
4.3.2	<i>Non-functional requirements</i>	79
4.3.3	<i>User requirements</i>	82
4.3.4	<i>Project planning</i>	83
4.4	SOCIO-ECONOMIC ENVIRONMENT.....	86
4.4.1	<i>Economic impact</i>	86
4.4.2	<i>Social and environmental impact</i>	86
4.5	REGULATORY FRAMEWORK.....	87
4.6	PROJECT BUDGET.....	89
4.6.1	<i>Market research and business model</i>	92
5	ANALYSIS	93
5.1	HARDWARE AND SOFTWARE PLATFORMS USED IN THIS PROJECT	93
5.2	FUNCTIONAL TESTS: CASE STUDIES TO EVALUATE THE SYSTEM.....	93
5.3	TRACEABILITY MATRIX	98
5.4	PERFORMANCE TESTS	99
5.4.1	<i>Neural Network analysis</i>	99
5.4.2	<i>Cluster analysis</i>	104
5.4.3	<i>Latency and data throughput in all connections</i>	105
6	CONCLUSIONS	108
7	BIBLIOGRAPHY	110
8	GLOSSARY	114
9	APPENDIX	

FIGURE INDEX

Fig. 2.1 Gartner Chart of Players in the Cloud Computing Space [44].	3
Fig. 2.2 Amazon (\$AMZN) Stock price in the last five years.	3
Fig. 2.3 Microsoft (\$MSFT) Stock price in the last five years.	4
Fig. 2.4 Google (\$GOOGL) Stock price in the last five years.	4
Fig. 3.1 Overview of the entire project including the Edge Kubernetes Cluster and the weather prediction server.	8
Fig. 3.2 Sensor placement inside the house.	9
Fig. 3.3 Kubernetes Pods and Containers [9].	11
Fig. 3.4 A Kubernetes Node showing four pods [9].	11
Fig. 3.5 A Kubernetes Cluster with three nodes [9].	12
Fig. 3.6 The Kubernetes Cluster used in the project and our application deployments.	13
Fig. 3.7 Kubernetes Deployments in our Cluster.	16
Fig. 3.8 Kubernetes Services in our Cluster.	16
Fig. 3.9 Pods Running in our Cluster.	16
Fig. 3.10 ESP32 Board [12].	17
Fig. 3.11 DHT11 Sensor [15].	19
Fig. 3.12 DHT22 Sensor [15].	19
Fig. 3.13 ESP32 Connection Diagram with the Cluster.	19
Fig. 3.14 ESP Python Program Flowchart.	20
Fig. 3.15 Connections Between Sensors and the Cluster.	20
Fig. 3.16 phpMyAdmin Main Page.	21
Fig. 3.17 'CurrentTemp' table in our database	22
Fig. 3.18 'CurrentTemp' table items.	23
Fig. 3.19 'procTemp' table items.	23
Fig. 3.20 'procTemp' table rows in our database.	24
Fig. 3.21 Scripts in the web server and their connections from outside the cluster.	25
Fig. 3.22 Dashboard in a laptop browser.	26
Fig. 3.23 Dashboard in a mobile browser.	27
Fig. 3.24 Connection between the weather prediction application, the API and the database.	28
Fig. 3.25 Weatherbit API pricing tiers.	29
Fig. 3.26 WeatherPred Program Flowchart.	31
Fig. 3.27 Complete 'ProcTemp' Database Filled by the weatherpred Application.	31
Fig. 3.28 First Two Neural Network Phases.	32
Fig. 3.29 Third Phase in Neural Network.	33
Fig. 3.30 Fourth Phase in Neural Network.	33
Fig. 3.31 Two sequences that the Neural Network will train from.	35
Fig. 3.32 LSTM Cell [18]	36

Fig. 3.33 Neural Network Layout and Architecture.	37
Fig. 3.34 Binary Step Activation Function [43].	38
Fig. 3.35 ReLU Activation Function [43].	38
Fig. 3.36 Inner Workings of the Last Layer in the Neural Network.	39
Fig. 3.36 Raspberry Pi 4B 4GB Version [21].	40
Fig. 3.37 Active Nodes in the Cluster.	42
Fig. 3.38 ESP32 and DHT22 Connection using GPIOs [28].	45
Fig. 3.39 ESP Python Program Flowchart.	46
Fig. 3.40 Scripts in the web server and their connections from outside the cluster.	49
Fig. 3.41 ProcTemp2.php Program Flowchart.	51
Fig. 3.42 Weather Predict Application Flowchart.	53
Fig. 3.43 Docker BuildX Compiler Process.	57
Fig. 3.44 WeatherPred container uploaded to the docker hub.	58
Fig. 3.45 All pods running in the Kubernetes Cluster.	60
Fig. 3.46 ‘trainNN’ Thread Flowchart.	64
Fig. 3.47 Database loaded into the Neural Network program using pandas.	65
Fig. 3.48 Future and Target Columns loaded in the dataframe.	65
Fig. 3.49 Dataframe before pre-processing of the data.	66
Fig. 3.50 Dataframe after pre-processing.	67
Fig. 3.51 Neural Network Layout and Architecture.	68
Fig. 3.52 Neural Network training.	69
Fig. 3.53 ‘getPredictions’ Thread Flowchart.	70
Fig. 4.1 Waterfall Methodology Phases [31].	72
Fig. 4.2 Gantt Chart detailing the entire development process of the project.	84
Fig. 5.1 Correlation between Neural Network predictions and actual temperature.	101
Fig. 5.2 Cloud Coverage Analysis compared to Neural Network confidence in prediction.	103
Fig. 5.3 Solar Radiation Analysis compared to Neural Network confidence in prediction.	104
Fig. 5.4 Entire Project overview detailing latencies between cluster, sensors, weather API and data warehouse.	105
Fig. 5.5 Wireshark analysis of the packets sent between the sensors and the cluster.	106
Fig. 5.6 Latencies to the cluster from different cities in the world.	106

TABLE INDEX

Table 3.1 Comparison Between Esp8266 And Esp32 Boards [13].	17
Table 3.2 Comparison Between Dht11 And Dht22 [14].	18
Table 4.1 Example Use Case.	73
Table 4.2 Use Case 1.	73
Table 4.3 Use Case 2.	74
Table 4.4 Use Case 3.	74
Table 4.5 Use Case 4.	75
Table 4.6 Example Requirement Table.	75
Table 4.7 Functional Requirement Fr1.	76
Table 4.8 Functional Requirement Fr2.	76
Table 4.9 Functional Requirement Fr3.	77
Table 4.10 Functional Requirement Fr4.	77
Table 4.11 Functional Requirement Fr5.	78
Table 4.12 Functional Requirement Fr6.	78
Table 4.13 Non-Functional Requirement Nfr1.	79
Table 4.14 Non-Functional Requirement Nfr2.	79
Table 4.15 Non-Functional Requirement Nfr3.	80
Table 4.16 Non-Functional Requirement Nfr4.	80
Table 4.17 Non-Functional Requirement Nfr5.	80
Table 4.18 Non-Functional Requirement Nfr6.	81
Table 4.19 Non-Functional Requirement Nfr7.	81
Table 4.20 User Requirement Ur1.	82
Table 4.21 User Requirement Ur2.	82
Table 4.22 Human Resource Costs.	89
Table 4.23 Direct Project Costs.	90
Table 4.24 Indirect Project Costs.	91
Table 4.25 Total Project Costs.	91
Table 5.1 Example Case Study.	93
Table 5.2 Case Study 1.	94
Table 5.3 Case Study 2.	94
Table 5.4 Case Study 3.	95
Table 5.5 Case Study 4.	95
Table 5.6 Case Study 5.	96
Table 5.7 Case Study 6.	96
Table 5.8 Case Study 7.	97
Table 5.9 Traceability Matrix.	98
Table 5.10 Neural Network Layout Tests.	100

Table 5.11 Neural Network Parameter Tests.	100
Table 5.12 Neural Network Observed Vs Predicted Test.	102
Table 5.13 Neural Network Overall Accuracy Test.	102
Table 5.14 Cluster Latencies And Data Packet Sizes.	105
Table 5.15 Kubernetes Cpu Load Test.	107
Table 5.16 Manual Deployment Cpu Load Test.	107

1 INTRODUCTION

1.1 Motivation

The motivation to embark on a project like this comes from the quest to learn new emerging concepts that are demanded in the industry. The current landscape in which computing as a whole is portrayed is a very fast-paced world in which old theorems and ideas are just beginning to be able to be exploited due to advancements in technology. As such, these ideas are being developed at an astonishing rate, and it's the job of Computer Scientists to research these new technologies and be knowledgeable of them.

1.2 Objectives

The set objectives laid out in this project are to be able to grasp a completely new branch in computing such as Cloud Computing and to be able to deploy and develop applications in that environment. To also be able to research and layout a case scenario that takes concepts seen in coursework such as Neural Networks and AI and mix them with completely new concepts like the use of containers or distributed computing.

As such, to label the project a success, the following must have occurred:

1. We have studied the foundations and the inner workings of Kubernetes and applied them to create an Edge Computing Kubernetes Cluster.
2. We built a successful project using IoT sensors and connected it to our Edge Cluster.
3. We have used AI techniques like a Neural Network to further improve our performance within the cluster.
4. There is an analysis on the data obtained from the project and conclusions are drawn from the performance obtained.

For the success of the project we need to demonstrate a valid use case scenario in which we could run a computer cluster and deploy many applications to the cluster to see how they would behave. We have chosen to deploy and develop a smart temperature control system that reads ambient temperature and humidity and combines them with weather predictions to be able to accurately determine if the temperature will remain constant or increase. This will be achieved using a Neural Network and the use of a deep learning network. We have chosen this approach based on the availability of documentation regarding multivariate predictions and the current market demand for high-quality machine learning applications, we also explore other AI and traditional statistical approaches to the problem, but ultimately decide to implement a Neural Network.

Lastly, there will be secondary objectives that will be laid out, mainly focusing on performance targets and cost metrics that will not be essential for the success of the project but that will be necessary to correctly assess the performance of the product as a whole.

1.3 Document structure

For ease of reading, we have included a small summary of each of the sections of the document and what is discussed in them.

In the second chapter, we will introduce all of the concepts that are used in this project, going in-depth and exploring the history of Cloud Computing, Machine Learning, and the Internet of Things.

Once those concepts have been introduced, we will go on to chapter three in which we will go in-depth into our project and carefully explain the motivation and deployment of each of the pieces of the project, how they interact with each other and how we can set everything up.

Later in the chapter, we will introduce the functionality and performance of our Neural Network, testing not only the feasibility of using such a tool but also analysing the performance that it brings us and its role in Edge Computing.

In the fourth chapter we will also go into detail describing the legal framework of the project, including all the laws that we would need to adhere to if the project was brought to market and all the technical standards that we need to follow in our code.

In the final chapter, we will explore the success of the project, how everything works with each other, and future expandability, including real-world use case scenarios and the scalability of our platform.

2 STATE OF THE ART

2.1 Cloud Computing and Edge Computing

To understand why this project and the underlying concepts that it is based on is important, we need to examine the landscape and context that it is being studied in.

As many early innovations in the computing world [1], cloud computing came out of the defense division of the United States, DARPA. Although the term “Cloud Computing” wasn’t coined until the late 1960s, there were many projects that relied heavily on cloud-like infrastructures and configurations, including project MAC [1] which had its premise as being able to use a computer by two people simultaneously. This is, in fact, an early premise to our current PaaS (Platform as a Service) systems in which multiple users can use a computer at the same time.

Another necessary precursor to the use of cloud was the development of the Internet, as individuals and companies needed a way to make use of these shared computers and the Internet was the perfect mode of transport for them.

One of the biggest players in Cloud Computing and the precursor to many companies that ended up adopting Cloud Computing services is Amazon. In 2006 they launched Amazon Web Services, which allowed people to run their applications on the cloud. After a while, they developed Cloud Storage, where you could pay a small fee and store your data on the cloud.



Fig. 2.1 Gartner Chart of Players in the Cloud Computing Space [44].

Today’s biggest AI and Cloud Computing players are currently those that are benefiting from the maximum amount of success, in both economical and user base growth. If we look at the Gartner Charts of Cloud Computing, we can see a clear correlation between the industry leaders and their growth as a business.

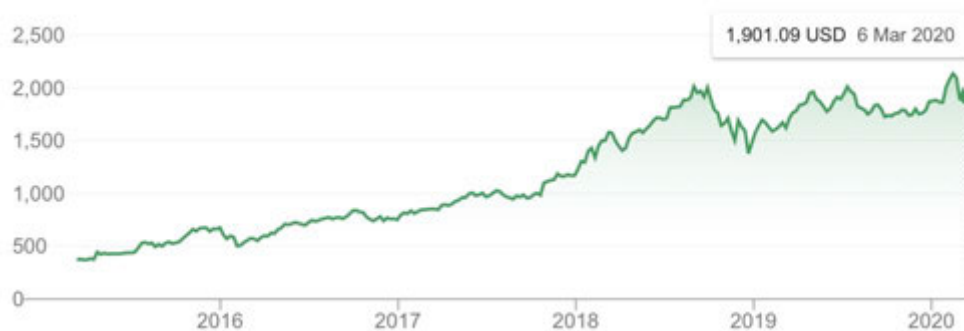


Fig. 2.2 Amazon (\$AMZN) Stock price in the last five years.

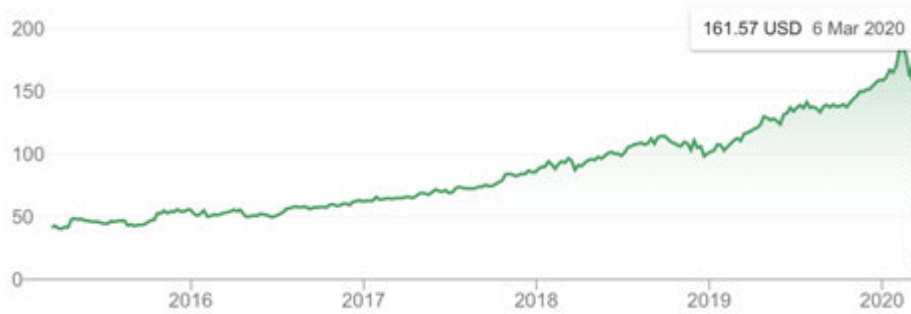


Fig. 2.3 Microsoft (\$MSFT) Stock price in the last five years.

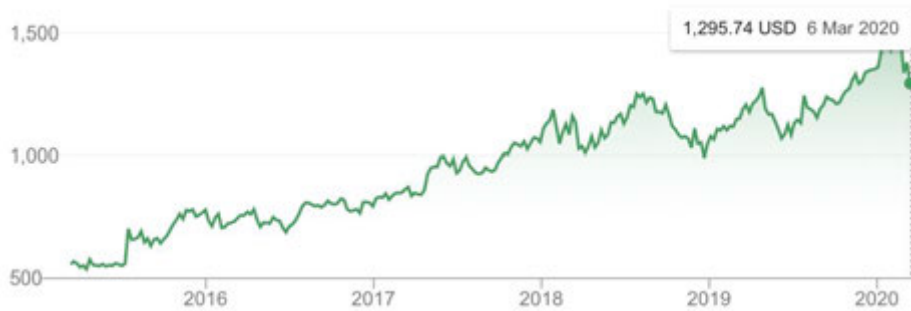


Fig. 2.4 Google (\$GOOGL) Stock price in the last five years.

If we look at the revenue of these companies in detail, particularly Amazon, we can see how its cloud services department (AWS) comprises 13% of the total 59.7B USD revenue [2] and that it alone has experienced a +41% increase in growth in a single year.

It's clear then that the cloud computing business is one that is highly in demand and with very fierce competition, and as such, it is a breeding ground for new technologies to explore and to maximize market share and revenue.

One of the key cloud computing breakthroughs in the past 5 years has been the use of containers. These containers are best explained as headless virtual machines that are comprised of an operating system kernel and the bare minimum so that a specific application can run on it.

The best way to understand a container is to imagine a very slimmed-down version of a virtual machine, running only the essential software. The sole reason a container exists is to run that specific application.

As such these containers allow for rapid deployment and development, as you can spin up a container for an individual, essentially creating a mini "PC" for them, saving their data and deleting the container when they're done using the service.

This premise has allowed the development of various revenue streams and offerings for these companies such as SaaS (Software as a Service) in which for example "Google Docs" is based. When you log in to create a document, a container is created just for you, once you finish your work that container is destroyed, using the same principle as a sandbox.

The idea of containers in itself isn't new and is based on the idea of virtual machines, but the main reason these are preferred to VM's are the overhead created by extraneous processes and applications that are run on VM's compared to a container. Apart from that, a container is inherently scalable, they can be added and destroyed with ease, the kernel that runs them is light and able to replicate a single container into a thousand instances.

The main player or platform in containers is Docker. This platform was built in 2013 [3] and it bases its business on offering the tools and the creation for container images, which can be stored on their website (Docker Hub). During this thesis, we will go in detail and explain how a container is created, uploaded, and hosted on docker hub and how to run a container from a specific computer.

The main problem that arises with the use of containers is that it lacks a content manager, there's no inherent automatic way in which a computer can tell when it's time to start a container, to deploy those containers and to connect the clients to them, that is where Kubernetes comes in.

Kubernetes is arguably one of the most exciting and talked about tools in the cloud computing branch. It allows us to extract the power of the container and smartly deploy it across several computers. These computers form a cluster network in which there are masters and workers. The workers run the containers and relay information back and forth with the clients and the master nodes run and manage the deployment of the containers.

Suddenly you're able to create a computer cluster running Kubernetes, a scalable and smart orchestration tool that allows containers to be spun up and be deleted at a moment's notice, resisting individual node failures such as a faulty individual computer in the cluster and you're able to service a vast amount of clients with your desired apps without having to think about scalability, availability, and security.

Another great thing about Kubernetes is that it not only allows you to create computer clusters over the world, but you can configure a local cluster with local machines that can act as an edge computing gateway, for local tasks such as IoT information gathering and operations that don't require a two-way communication from the outside web. This is a premise in which our case scenario will be built, and it is where we will explore the benefits of having an Edge Computing cluster instead of having all of our IoT sensors talk to the internet directly.

2.2 Machine Learning and Neural Networks

Machine learning has become one of the most important and profitable businesses in the tech industry as of late. This resurgence can be attributed back to the mid-2000s when google introduced its AI and Machine Learning libraries to do deep learning studies on pictures and different projects associated with the use of big data [4].

We also need to mention the usage of GPUs (Graphical Processing Units) as AI accelerators. These GPUs are specialized hardware's that can manipulate images and textures with lots of performance. In particular GPU manufacturer Nvidia has made great strides in developing GPUs specifically for AI use, like self-driving cars. Tensor cores are used in GPUs to accelerate Neural Network uses.

If we look even further back though, we can see that machine learning has been developing and improving itself since the early 1950s when IBM's Arthur Samuel developed a computer program for playing checkers. Eventually, he programmed his game as such that the program itself could decide what its next move was going to be, leading to the creation of the minimax algorithm.

Nowadays machine learning serves as the cornerstone for many enterprise solutions that we use every day, from facial recognition, image processing, speech recognition, and promising future projects such as autonomous driving [5], chatbots, or novel robot algorithms [6].

To easily explain how machine learning works is to think that a computer can learn and make decisions without being programmed to do so. This means that a computer can be able to take decisions like a human without any interaction from a person, in the same way, that a human produces an action from a stimulus, a computer using machine learning should be able to reproduce an action from a given stimulus.

To do this process of "learning" which is teaching the machine learning algorithm a way to react correctly to input, we must first compile our data. This data is crucial for the correct operation of the algorithm because the results we will get from the algorithm can only be attributed to the starting quality of the data it was given.

In the following chapters, we will discuss what constitutes "quality" data and how we can make our data even better for use in machine learning algorithms, using our use case scenario and collecting our data for this purpose.

2.3 Internet of Things (IoT)

Internet of Things is a fairly new concept that has only really been exploited for less than a decade. The term IoT was coined in 1999 to initially promote RFID technology, but it wasn't until the early 2010s that the market started exploring the possibility of these "Internet Connected Devices" that could be extremely useful for several given scenarios.

The formal definition of IoT according to McKinsey is *"Sensors and actuators embedded in physical objects are linked through wired and wireless networks, often using the same Internet Protocol (IP) that connects the Internet."* [7].

Maybe one of the biggest players in the IoT space, Google, kickstarted the IoT revolution when it was discovered that their Street View service was collecting Wi-Fi SSID names as it mapped the streets of the world. Another big push for IoT came from Nest [8], a smart thermostat that used machine learning and connected to the internet to smartly regulate the temperature in your house. As we will see in the next chapter, we base our use case scenario in a pseudo thermostat device. This nest smart thermostat is still currently being sold in stores for the price of USD 349 (As of 27/03/2020). The company that produces the product, Nest was sold to google for USD 3.2B in 2014.

In short, IoT could be any device that connects to the internet, but stricter rules dictate the usefulness of IoT, due to this IoT has been thrown around as a buzzword, integrating Wi-Fi connectivity to regular household items that didn't need it. An example of this is the

Juicero smart juicer, which failed after trying to promote a juice subscription service that used an always-connected juicing machine to read the packets.

Due to examples like these in the market, we need to stop and think about the true functionality of the product that is being designed, and to make the conscious decision to integrate IoT functionality only after it is clear that it brings a clear advantage or clear business sense.

Another problem with IoT that we will be tackling during this project is the network congestion caused by these millions of devices connected to the internet. We need to make sure we're only sending essential data, compressed as much as possible, and that we make ethical and responsible use of the internet channels available to us. To further drive this point, we only need to look at a domestic network and perform a scan. A couple of years ago it would've been possible to see between 5-10 connected devices, phones, laptops, and even TVs were the norm. Nowadays it is easy to exceed 50 devices, between smart lightbulbs, speakers, and smart home appliances. This not only poses a privacy risk, as countless bytes of data are being generated with every device, but it also congests the network.

To deal with this we will make sure we have a regular cadence of data, make sure we don't congest the network with our IoT devices, and we will explore the concept of Edge Computing, which will improve our latency between IoT devices and serve as a way to prevent internet leakage of data and internet congestion.

As such we can tie Cloud Computing, AI, Machine learning, and IoT into a cohesive project that demonstrates the power in today's tools and explain why they make sense as a commercial investment.

3 DEVELOPMENT

3.1 Architecture overview

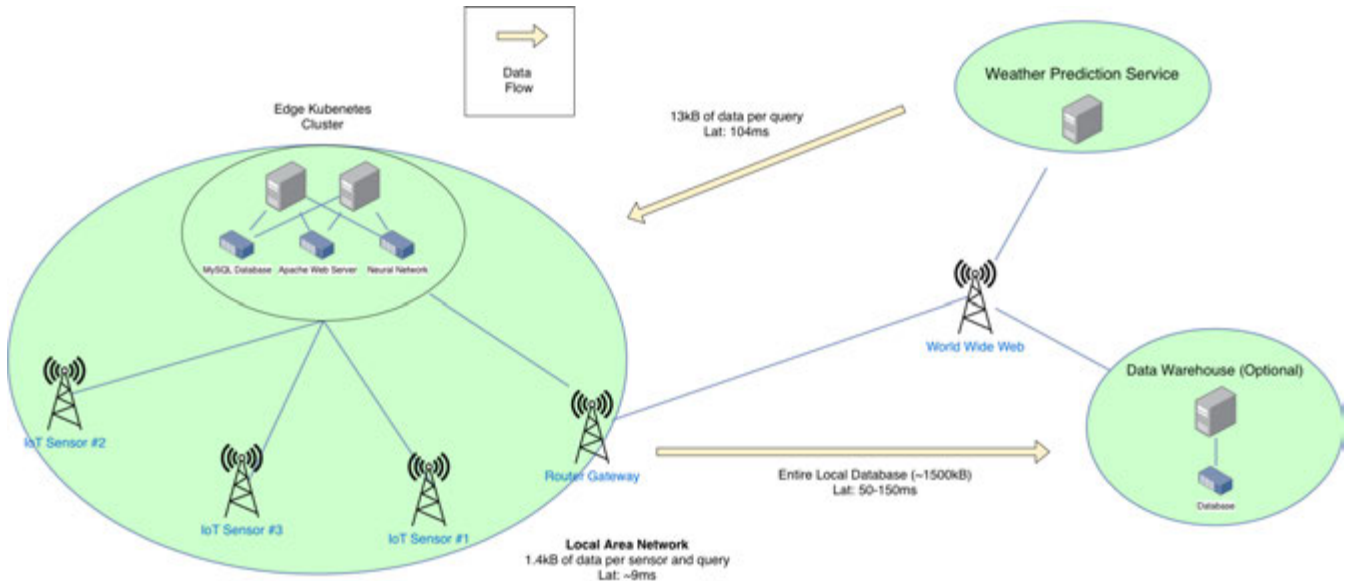


Fig. 3.1 Overview of the entire project including the Edge Kubernetes Cluster and the weather prediction server.

This is the complete architecture overview of our project. As we can see, it is quite busy at first glance, we have three main parts:

- **The Kubernetes Cluster:** It represents a stack of two Raspberry Pi's working in unison running a number of applications between each other. Kubernetes is the orchestrator that spins up and manages the applications running in the cluster, which are:
 - **MySQL Database:** To store the readings from our sensors.
 - **Apache Web Server:** To display web pages and to process our readings from the sensors.
 - **WeatherPred:** a custom-built Python application to get weather information from the internet.
 - **Neural Network:** A Neural Network built from scratch to predict temperature changes in the house and to enable Edge Computing features that will be explained in the next section.
- **The Sensors:** They are small ARM boards that can be programmed to do whatever task we need. We will be attaching a temperature sensor and using it as an IoT device to transmit data to our cluster.

- **The Data Warehouse:** This part is not implemented into the project but would be indicative of a real deployment by a business. The point of a data warehouse is to agglomerate all of the data collected by the cluster and many other local clusters. Instead of having the sensors send the data directly to the warehouse clogging the bandwidth and dealing with unnecessary amounts of information, we can process the data on-site with our Neural Network and only send information when the Neural Network cannot predict the temperature.

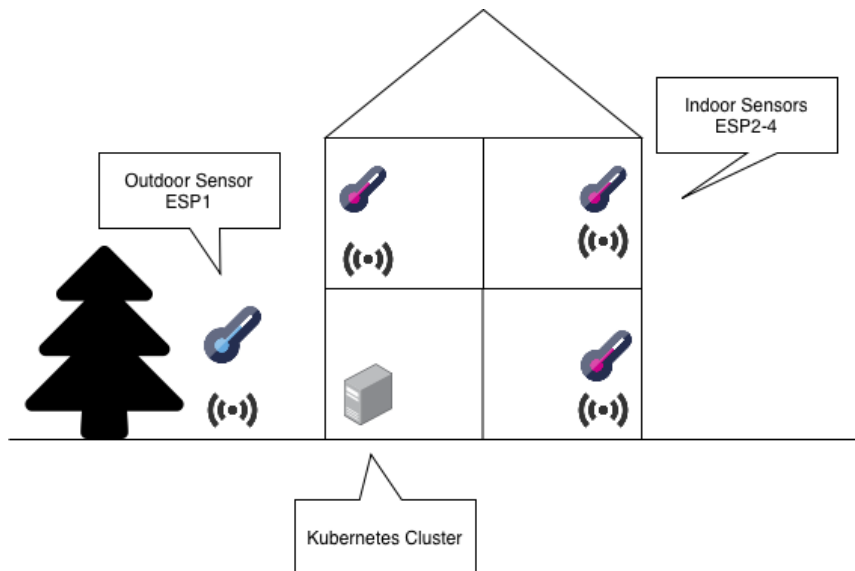


Fig. 3.2 Sensor placement inside the house.

3.2 Introduction to Kubernetes

During the duration of this project, we will be referring to different system architectures and configurations that we shall introduce. Kubernetes is the main orchestrator and makes the entire project feasible. As introduced before, it allows flexibility and scalability in the deployment of our applications and solutions and will automatically run our applications on any of the nodes of our computer cluster.

This removes a lot of the headaches associated with cluster computing and the parallelization of computer programs. To understand how Kubernetes manages all of this, we will give a brief description of the underlying components that allows Kubernetes to deploy applications.

To begin our Kubernetes description, we are going to imagine a two-node setup, exactly like the one we're going to use in our use case scenario and imagine that one of the nodes acts as the master and that the other one is the worker. Both master and worker can run programs, but it is the master that decides who runs what at every moment.

Truthfully, in a production setting, nodes aren't homogenous like that, some may be standard computers, or compute units, small boards like Raspberry Pi's or even huge servers, and that is the job of Kubernetes, allowing a non-homogenous cluster to work as one, deploying applications automatically to each node or nodes making sure all of the requirements are met.

For example, in our case, if we wish to deploy a database application like MySQL or MariaDB, we would want that to be tied to a node that has local storage if that's what we desire. As such we can let Kubernetes know that the MySQL deployment should only be deployed on nodes that have local storage. We can streamline our deployments and make sure that the right applications are deployed in the right nodes at the right time.

Another useful detail about Kubernetes is that it allows us to replicate deployments. Kubernetes deploys applications using Docker containers and these can be deployed individually from 1 to N containers. This allows a large number of people to have access to the resources at any given time. When demand is high, Kubernetes automatically deploys more containers, and when that demand is over, they automatically disappear to allow more efficient resource management. In our case this will prove useful in case we scale to the point where we need multiple deployments of the web server (in case we have a lot of incoming connections) or we need multiple deployments of the database (in case we have a lot of sensors).

Now that we have touched the subject of Containers, it is time to introduce how Kubernetes works and talks with the different components of this system. Kubernetes is an abstraction of all applications and runs on top of Linux (Although there is an increasing amount of Linux distributions that come with only Kubernetes) and it uses a series of encrypted API calls between all the nodes to coordinate the deployment and sharing of logs and data.

Kubernetes doesn't deploy directly on the worker nodes, instead, it creates pods, which can be referred to as small virtual machines with very little headroom that share containers, sometimes these pods only contain one container, but it varies between each deployment.

These pods contain the containers and a network interface that allows the containers to "natively" talk to the cluster and relay information back and forth. Kubernetes has a very strong networking background that we will get to in further detail in the next chapter.

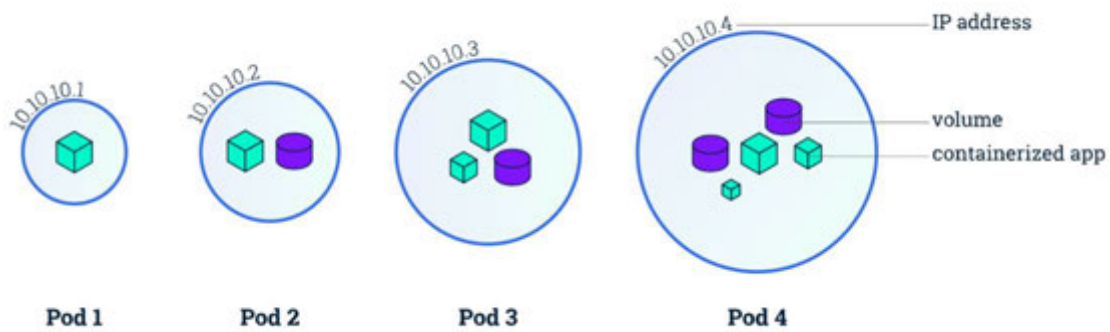


Fig. 3.3 Kubernetes Pods and Containers [9].

This network is comprised by a “local” cluster IP range, that allows the cluster to share information only between the cluster members, as such you can configure different networking settings in the deployment information to make sure a specific application can only communicate with the cluster IP or allow it to be reached from outside the cluster (like a web server).

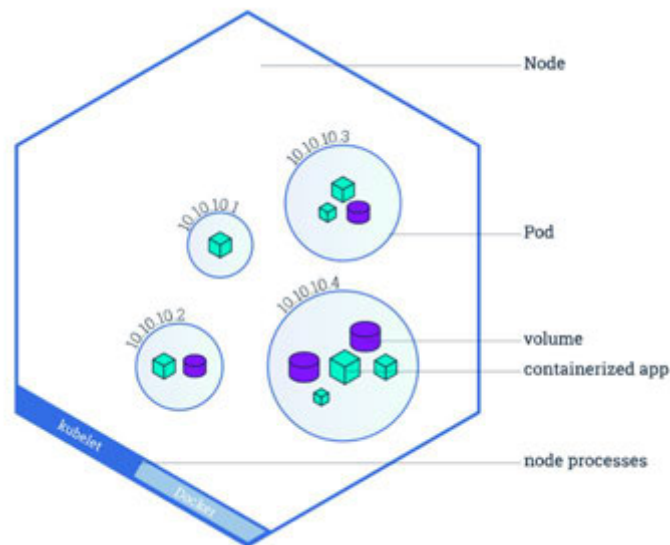


Fig. 3.4 A Kubernetes Node showing four pods [9].

Now that we have a basic understanding on how Kubernetes deploys its applications and before getting into deploying one, we need to quickly reference what a Service is, and how we connect to an application running on Kubernetes, as it has been quite possibly the biggest difficulty of this entire project. A service in our use case would be what connects the web server to the outside of the cluster. It acts as a proxy layer between the deployment and the outside.

Due to limitations in the software, we are not able to run a "complete" traditional install of Kubernetes, and we have opted to make our services instead of using an ingress controller like NGINX [10] or traefik [11] to direct the service endpoints.

In our case we only need to use the simplest services in Kubernetes, these are:

- ClusterIP: This service exposes the application only to the cluster, making it only reachable inside the cluster itself.
- NodePort: Exposes the service in a specific port inside the cluster, for example, we can set a specific NodePort in the cluster IP and making it accessible from outside the cluster if we use that port.
- LoadBalancer: This is the service we have experimented using with traefik and haven't had successful results. In more complete versions of Kubernetes, this ingress controller lets you automatically deploy to a port instead of having to set up a NodePort manually.

We also need to mention that a service can make not only multiple pods reachable but also multiple nodes, making for an interesting and flexible way of deploying and exposing your applications.

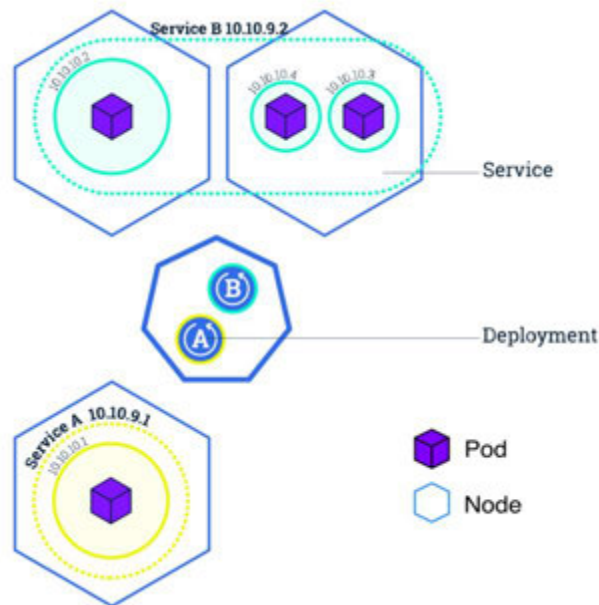


Fig. 3.5 A Kubernetes Cluster with three nodes [9].

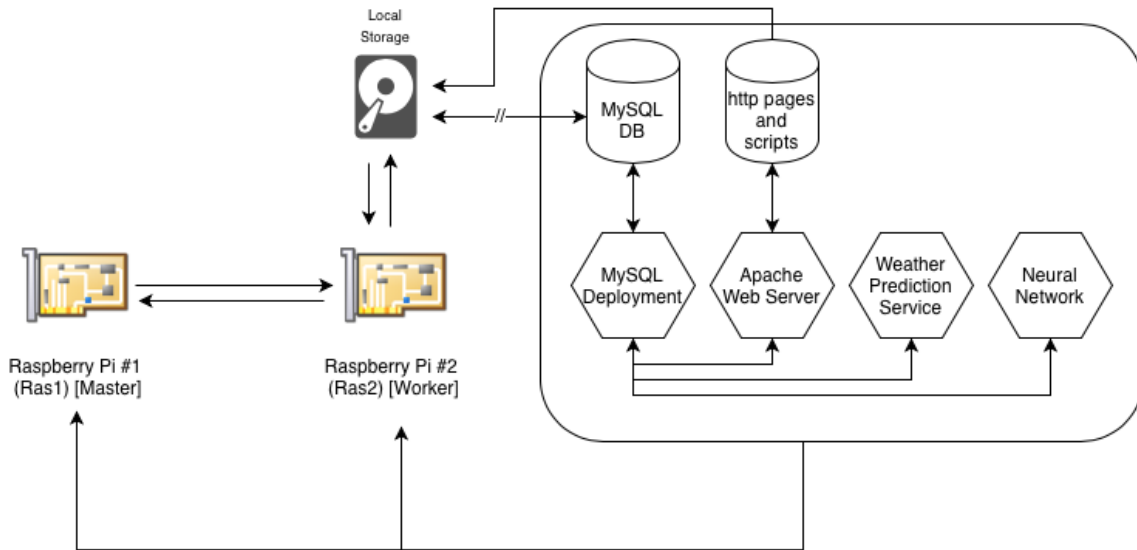


Fig. 3.6 The Kubernetes Cluster used in the project and our application deployments.

This is an overview of our use case scenario that we are going to implement using Kubernetes.

All of the Deployments are represented by hexagons and these are automatically deployed on either Raspberry Pi through Kubernetes. The only two applications that need local storage are the database and the webserver and these can only be allocated to the #2 Raspberry Pi (since it's the only one with local storage set-up for the cluster)

We will go further in detail in how these services are connected and how they can communicate with each other and with other elements outside the cluster (like the sensors)

Our main objective is to recreate a real use case scenario that a company would be interested in implementing in their business practice. For that, we need to think of a viable business strategy and have decided that deploying a project using Neural Networks, IoT, and Kubernetes was a perfect demonstration of a current-day business proposition.

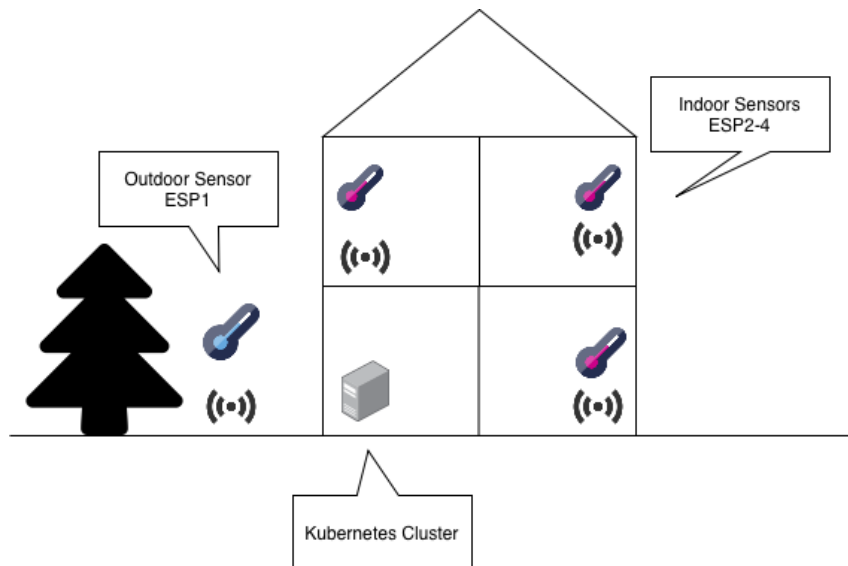


Fig. 3.2 Sensor Placement Inside the House.

In our use case scenario, we will have many sensors around the house, and we will build our applications with inputs from these sensors. For now, we will connect the sensors and the cluster and store the information coming from the sensors on the cluster.

As discussed, before we need some sort of storage solution that allows us to store and manage our data to be able to be used in our further programs. We have chosen to use a MySQL database since its widely used and there already is a container image of MySQL in the Docker Hub.

In broad terms, we just need to set up the name, location of the image (hyprriot/RPI-MySQL:latest) and then we need to set up specific environment variables for MySQL like the root password and the container port (to be able to communicate with the MySQL service which we will declare later).

We also need to declare where the database will be stored. Kubernetes allows an abstraction in terms of storage and as such, when we declare the storage path, we are not declaring “where” that path is, it’s just a relative path inside the pod that will be assigned to run.

We also need to instance a persistent volume claim (PVC), as its vital that with every restart of the pod, we don’t lose the data. Kubernetes allows pods to have temporary storage, and when the pod has finished, the data is deleted. We will be using that model in our posterior applications, but since we want the data to be persistent, we need a PVC.

We have also used a concept called ‘secrets’, it allows Kubernetes to handle sensitive data like passwords without being exposed to the cluster using a hash to store the passwords instead of leaving it in plain text.

Once we have applied the deployment, Kubernetes will deploy our application into the cluster. To check that the deployment is running we can ping the cluster for its running pods and deployments.

Now that we have our MySQL database deployed, we need to find a way to manage that database and a way to insert data in a secure and viable manner. The most sensible way is to use a Web Server that has MySQL access and that can perform SQL queries into the database. We have also explored a way to use MQTT, a lightweight messaging protocol built for IoT devices but unfortunately at the time of writing it is not yet operational with the sensors we will be using.

Since we have previous experience with PHP, we have decided it would be the easiest way to create scripts and web pages to interact with the cluster and the MySQL database. To use PHP, we need a Web Server that supports it and as such, we have decided to use an Apache 2 server and to install phpMyAdmin to manage the MySQL database.

To do this, just like with the database we only need to declare a .yaml file stating the docker image of apache2 and phpMyAdmin. For ease of installation, we will be using an image that already comes with phpMyAdmin pre-installed and the only thing we need to do is specify the MySQL database IP address (ClusterIP defined earlier) and the login details that we set up in the secret.

We then apply the changes and wait for Kubernetes to allocate us space (we need permanent storage here as well to store our web pages and scripts) and to launch the pods.

Now we need a way to communicate with the Apache server from outside the cluster. The applications we deploy in the cluster will be able to access the Apache instance since they share the same ClusterIP but if we want access to the database from outside the cluster, we need to set up a Kubernetes 'Service'.

A service can be thought of as a forwarding system, in which we access the cluster's external IP and a specific port, and our data is forwarded from the cluster into the specific deployment that that port is attached to.

That specific kind of service is called a NodePort, and to set one up we simply need to declare another .yaml file.

We can either manually create a specific NodePort by giving it a set number (31050 for example) or we can let Kubernetes assign an available one on deployment.

Once the service has been applied to the cluster, we can check on the status of the cluster by running these next commands.

```
> ./k3s kubectl get deployments
```

```
root@ras1:/home/pi/k3s# ./k3s kubectl get deployments
NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
phpmyadmin-deployment              1/1     1             1           37d
mysql-deployment                   1/1     1             1           37d
weatherpred-deployment             1/1     1             1           37d
```

Fig. 3.7 Kubernetes Deployments in our Cluster.

This command will retrieve our active deployments (weatherPred will be explained in the next chapter as it relates to our Neural Network).

```
> ./k3s kubectl get services
```

```
root@ras1:/home/pi/k3s# ./k3s kubectl get services
NAME            TYPE        CLUSTER-IP    EXTERNAL-IP    PORT(S)        AGE
kubernetes     ClusterIP   10.43.0.1     <none>         443/TCP        201d
phpmyadmin-service NodePort    10.43.231.54  <none>         80:32151/TCP   37d
```

Fig. 3.8 Kubernetes Services in our Cluster.

This command will retrieve the active services and their IP addresses and ports to access the deployments they're attached to.

As we can see we have a port associated with port 80 on NodePort, in our case port 32151 will be forwarded to port 80 and thus relay all the information to the phpMyAdmin deployment.

```
> ./k3s kubectl get pods
```

```
root@ras1:/home/pi/k3s# ./k3s kubectl get pods -o wide
NAME                                READY   STATUS    RESTARTS   AGE   IP            NODE
svclb-mysql-service-c7t4r          1/1    Running   1          37d   10.42.0.162   ras1
svclb-mysql-service-2rn5z          1/1    Running   0          35d   10.42.1.177   ras2
phpmyadmin-deployment-5ddb884-lmm6d 1/1    Running   0          37d   10.42.1.179   ras2
mysql-deployment-f65f88d54-cwrz5    1/1    Running   0          37d   10.42.1.178   ras2
weatherpred-deployment-7ff7b6ff5c-ttvn7 1/1    Running   0          35d   10.42.1.181   ras2
dnsutils                            1/1    Running   2290       96d   10.42.0.160   ras1
```

Fig. 3.9 Pods Running in our Cluster.

This command will be our complete view of what's running in the cluster, as we can see we get the status of all of our services and deployments, as well as restarts and their IP addresses. We also get to see what node they're deployed on.

3.3 Building the sensors

After completing the setup of the cluster and making it accessible to the network through our Kubernetes services we can start building our sensors that will provide the temperature readings for the subsequent calculations that our Neural Network will need to predict the temperature inside the house.

We have chosen sensors that made sense in the commercial sense, looking to what the industry leaders are building and decided that since we can't build custom made sensors, we would use cheap, readily available microcontrollers and temperature/humidity sensors from the market.

In particular we have decided to use the ESP32/ESP8266 boards. These allow us not only to run our applications and sensor needs reliably but also at a very limited cost.

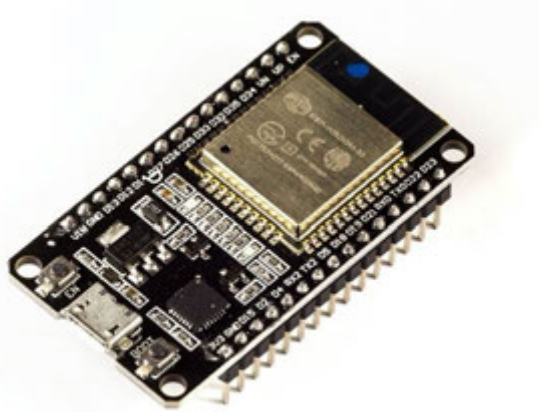


Fig. 3.10 ESP32 Board [12].

TABLE 3.1 COMPARISON BETWEEN ESP8266 AND ESP32 BOARDS [13].

	<i>ESP8266</i>	<i>ESP32</i>
<i>CPU</i>	Xtensa Single-core 32-bit L106	Xtensa Dual-Core 32-bit LX6 with 600 DMIPS
<i>Frequency</i>	80MHz	160MHz
<i>WIFI / Bluetooth</i>	Wi-Fi: HT20 802.11b/g/n Bluetooth: NO	Wi-Fi: HT40 802.11b/g/n Bluetooth: 4.2 and BLE
<i>GPIO Channels</i>	17	34
<i>Deep Sleep (Low Energy Consumption mode)</i>	NO	YES
<i>Price</i>	3-6 USD	6-12 USD

“Bare metal” ARM chips have become very popular in the recent years due to their cost proposition as well as their wide breath of use cases. Popular implementations of code that can run on these chips include: Assembly code, Arduino code and Python code.

Since we have learnt to use Python during our Machine Learning classes and the ease of use associated with it, we have decided to use a Python library to run on the ESP boards.

The Python implementation that runs on the ESP boards isn’t full Python, but a lightweight version called ‘MicroPython’ and as such presents itself with a few particularities that we need to take into account when writing code for the boards.

3.4 Introducing our temperature sensor

Since our board uses GPIO signals and outputs 3.3V to the pins, we have decided to use readily available external temperature and humidity sensors. In this category we have two types of sensors to use, the DHT11 and the DHT22 temperature/humidity sensors.

The differences between the two are:

TABLE 3.2 COMPARISON BETWEEN DHT11 AND DHT22 [14].

	<i>DHT11</i>	<i>DHT22</i>
<i>Temperature Range</i>	0-50C (+-2C)	-40 to 80C (+-0.5)
<i>Humidity Range</i>	20-90% (+-5%)	0-100% (+-2%)
<i>Resolution</i>	Humidity: 1% Temperature: 1C	Humidity: 0.1% Temperature: 0.1C
<i>Operating Voltage</i>	3-5V DC	3-6V DC
<i>Sampling Period</i>	1 second	2 seconds
<i>Price</i>	1-5 USD	4-10 USD

The use of the DHT11 would be permitted in our use case scenario except for its poor temperature resolution. In our development of the project, all our sensors equipped DHT11 sensors, but we found that the performance of the Neural Network suffered with such drastic temperature differences, as the temperature variations are much more nuanced indoors than outdoors. As such we upgraded all of our interior temperature sensors to DHT22 and kept the outside sensor as a DHT11.

Both sensors have identical setup and programming so a whole system upgrade is feasible if the project required so in the future.

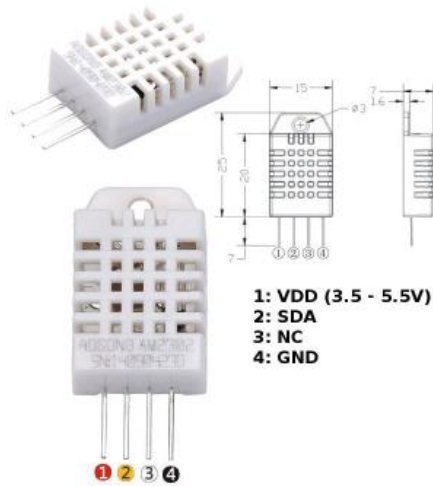


Fig. 3.12 DHT22 Sensor [15].

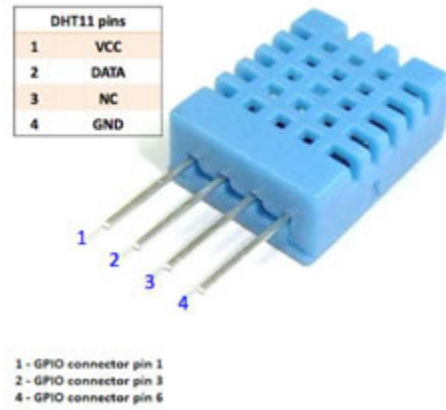


Fig. 3.11 DHT11 Sensor [15].

3.5 Introducing the ESP Python program

Since we have the capability to program any Python application into our board, we will first need to understand what kind of application we need the boards to run. Since the boards cannot communicate directly to the MySQL server, we will need a proxy that will allow us to communicate the temperature and humidity values from the board and into our database.

There is a number of different approaches that could allow us to achieve this. Since our board has Wi-Fi and Bluetooth capabilities, we could design a proprietary protocol that will allow the flow of information between Kubernetes cluster and the boards in a secure and lightweight manner. This protocol could run on TCP and use SSL sockets for encryption.

This use case scenario could be explored further, as with wider area networks, and large local area networks, the point of encryption becomes an important one. For smaller use case scenarios, where we are dealing with a LAN confined to a home or small enterprise use, we can use HTTP/TCP protocols with ease and take advantage of the many features that come with those protocols.

Mainly we're after the guaranteed delivery of packets that TCP offers and the readily available implementation of HTTP requests on MicroPython.

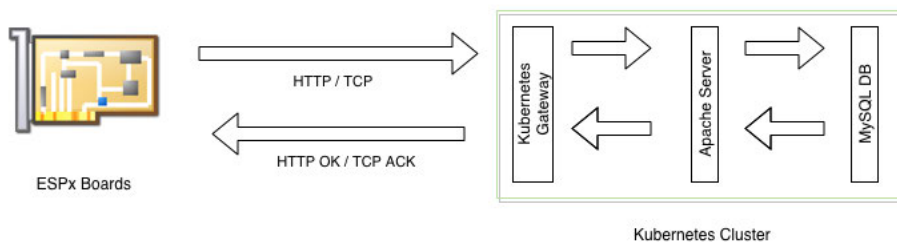


Fig. 3.13 ESP32 Connection Diagram with the Cluster.

Now that we're set in how the board and Kubernetes cluster will communicate, let's start with the Python application that will run on the ESPx sensor boards.

We will go in depth in how we can achieve this in the implementation chapter of this project but for now we will only discuss the program's flowchart and main objectives.

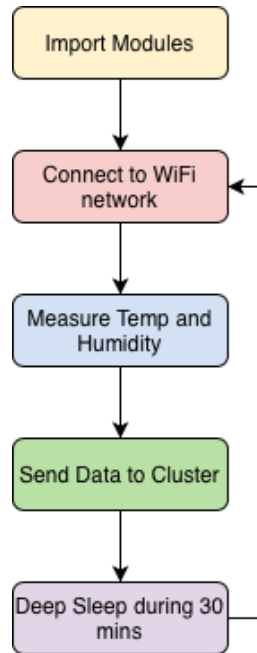


Fig. 3.14 ESP Python Program Flowchart.

The board will firstly import the necessary modules and connect to the network via Wi-Fi. We can then start measuring the temperature and humidity with the sensor and send the data to the cluster via our HTTP request.

We then take advantage of our ESP32 board and go into a deep sleep to make sure we aren't wasting any energy idling.

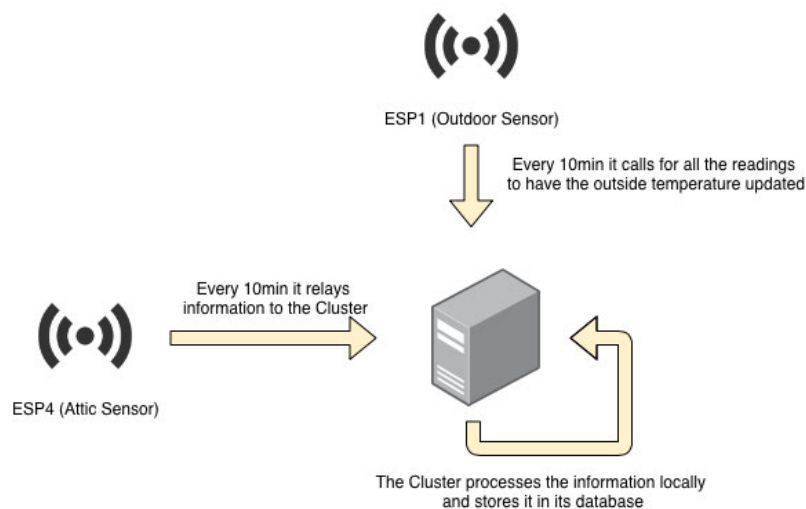


Fig. 3.15 Connections Between Sensors and the Cluster.

3.6 MySQL database

The MySQL database is arguably the most important piece of the deployments in the cluster. It is required in almost every operation inside the cluster since it stores the readings from the sensors and the weather prediction server used by all the other components in this project. For this reason, we require a reliable database that scales well with a growing number of connections.

The install of the MySQL database is simple, we have already deployed a container running a MySQL instance and we simply need to populate the database with tables and set their columns and access parameters.

We have two options for this, we can either manually access the MySQL terminal from the Kubernetes container and manually set new tables and their columns, or we can use phpMyAdmin on the Apache Server instance that we have deployed into the cluster.

Since we have set the MySQL credentials in the environment variables inside the apache deployment file, phpMyAdmin should run automatically accessing the following URL

```
> 192.168.1.111:32151/phpMyAdmin
```

Once we see the phpMyAdmin home page we will go through an initial setup and once that's completed, we can create our first database table. Since this process is visual throughout the GUI provided by phpMyAdmin no explanation is necessary, we will only discuss the two tables that are required for the entire project to function.

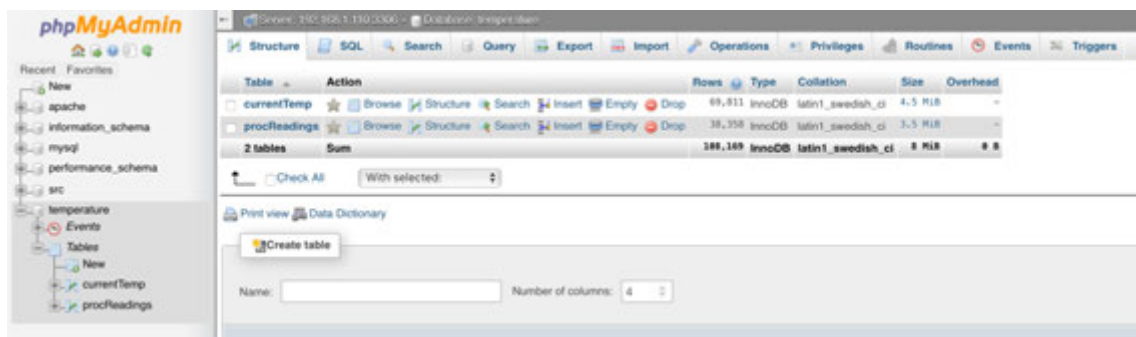


Fig. 3.16 phpMyAdmin Main Page.

As we can see in the phpMyAdmin overview, we have two tables with almost one hundred thousand rows of data, collected from approximately 6 months of use. If we extrapolate this, without any sort of automatic cleaning of the database, keeping years and years of records would only occupy a small amount of memory, which is one of the main advantages of the use of a MySQL database.

We will now go into each table and explain how they were created and why they have their respective columns.

3.6.1 CurrentTemp table

This is the main ‘dump’ table that collects every single reading from the sensors, this table gets fed from all the different sensors, interior and exterior, but no external (Internet) sources.

```
> SELECT * FROM `currentTemp`
```

id	temp	hum	station	date	room_id	proc
9624	20	37	ESP4	2019-11-22 03:58:05	4	1
9625	8	91	ESP1	2019-11-22 04:03:29	2	1
9626	20	50	ESP2	2019-11-22 04:06:12	3	1
9627	21	41	ESP5	2019-11-22 04:07:10	1	1
9628	20	36	ESP4	2019-11-22 04:09:15	4	1
9629	8	90	ESP1	2019-11-22 04:13:45	2	1
9630	21	41	ESP5	2019-11-22 04:17:22	1	1
9631	20	50	ESP2	2019-11-22 04:17:27	3	1
9632	20	37	ESP4	2019-11-22 04:20:26	4	1
9633	8	90	ESP1	2019-11-22 04:23:59	2	1
9634	21	41	ESP5	2019-11-22 04:27:34	1	1
9635	20	50	ESP2	2019-11-22 04:28:38	3	1
9636	20	36	ESP4	2019-11-22 04:31:40	4	1
9637	8	90	ESP1	2019-11-22 04:34:15	2	1
9638	21	41	ESP5	2019-11-22 04:37:47	1	1
9639	20	50	ESP2	2019-11-22 04:39:48	3	1
9640	20	36	ESP4	2019-11-22 04:42:50	4	1
9641	8	90	ESP1	2019-11-22 04:44:29	2	1

Fig. 3.17 ‘CurrentTemp’ table in our database

As we can see in the database, once we query every record in the database, we get an overview of the entire table. Breaking down every column in the database we can see:

- ID: A numeric non repeatable id to identify every row in the database
- Temp and Hum: Values extracted from the DHT sensor in the ESP boards.
- Station: Individual ESP boards
- Date: a Datetime object in YYYY-MM-DD HH:MM:SS format
- Room_id: individual room (2: outside, rest of numbers: inside rooms)
- Proc: This value will be used to check if the row has been processed by the PHP script which will move it into a processed table (procTemp) [0: Not Processed, 1: Processed]

#	Name	Type	Collation	Attributes	Null	Default	Extra
1	id	int(11)			No	None	AUTO_INCREMENT
2	temp	double			No	None	
3	hum	float			No	None	
4	station	varchar(20)	latin1_swedish_ci		No	None	
5	date	datetime			No	None	
6	room_id	int(11)			No	None	
7	proc	int(11)			No	0	

Fig. 3.18 'CurrentTemp' table items.

3.6.2 ProcTemp table

This table exists because we need to find a way to process the raw data from the sensors into a processed table that has a tuple between the inside temperature and the outside temperature. The reasoning behind this is because our Neural Network can't use data from different rows, it needs to be organized in a normalized row with all its attributes.

The script responsible to populate this table resides in the Apache Web Server, specifically procTemp2.php which is triggered by the outside ESP boards. When this script is triggered, the PHP script grabs every sensor reading from the inside sensors in the past hour, attaches the current exterior temperature and humidity to it and inserts it into the new processed 'procTemp' table.

#	Name	Type	Collation	Attributes	Null	Default	Extra
1	id	int(11)			No	None	AUTO_INCREMENT
2	temp	float			No	None	
3	hum	float			No	None	
4	out_temp	float			No	None	
5	out_hum	float			No	None	
6	clouds	int(11)			No	0	
7	solar_rad	double			No	0	
8	uv	double			No	0	
9	date	datetime			No	None	
10	room_id	int(11)			No	None	

Fig. 3.19 'procTemp' table items.

This table differs from 'currentTemp' because it also receives information from external sources such as the internet. In the next section we will discuss why we need external weather information and how we're able to get that data. In the meantime, let's discuss all the columns in the table.

- ID: Unique ID to identify the rows
- Temp & Hum: Interior readings extracted from 'currentTemp'
- Out_Temp & Out_Hum: Exterior readings extracted and added from 'currentTemp'
- Clouds, Solar_Rad, UV: Weather data sourced from the internet.

- Date: Same date from the original sensor measurement in ‘currentTemp’
- Room_id: Extracted from the room ‘temp’ and ‘hum’ was measured in ‘currentTemp’

If we query the table in SQL,

```
> SELECT * FROM `procReadings`
```

id	temp	hum	out_temp	out_hum	clouds	solar_rad	uv	date	room_id
1									
38389	24	33	20	65	0	0	0	2020-05-23 06:02:49	4
38388	25	32	20	65	0	0	0	2020-05-23 05:52:44	4
38387	25	32	20	64	0	0	0	2020-05-23 05:42:40	4
38386	25	32	20	63	0	0	0	2020-05-23 05:22:32	4
38385	25	32	20	63	0	0	0	2020-05-23 05:32:36	4
38384	25	32	20	63	0	0	0	2020-05-23 05:12:29	4

Fig. 3.20 ‘procTemp’ table rows in our database.

As we can see, the columns that contain the internet weather data (clouds, radiation and uv) are empty, this is because the data will be added after an hour has passed, to eliminate working with a prediction (in the next chapter we will discuss why we have chosen these values to monitor)

The script that will update those values will run as a Kubernetes application called ‘weatherPred’ and in the next section we will discuss how to build such a Kubernetes application and deploying it to the cluster.

As described, we now have two tables and have prepared our MySQL database to receive the data coming from the sensors. Next, we need a way to carry the data from the ESP boards to the MySQL database.

3.7 Apache web server

To handle the incoming connections, we need to explore the world of web development and PHP programming. Ordinarily, we would’ve used a Python or C socket to receive the incoming information, but looking at the wide market we see that IoT devices usually come with a dashboard or a control panel where an end user can see how the sensors are doing, see the records stored in the database and make changes to the system.

To comply with this, we’ve not only built an Apache Web Server to receive the incoming data from the server, but we have also built a dashboard to display the temperatures and humidity to manage the entire array of sensors.

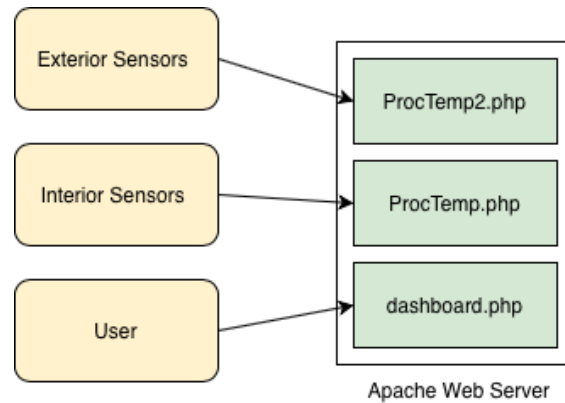


Fig. 3.21 Scripts in the web server and their connections from outside the cluster.

Our web server will contain 3 pages / scripts. The only page intended for the user to visit is the dashboard, and as such we will restrict the usage of any other page to any user that isn't a sensor. We can do this because the HTTP headers contain information about the browser and the device that is requesting that web page.

This of course isn't a secure way to restrict access but in the context this server is going to be used (LAN) it is enough to prevent accidental access to these restricted scripts and potentially introducing erroneous information into the database.

3.7.1 Proctemp.php

This script will process the incoming information from the ESPx sensor boards. Firstly, we will grab the parameters in the URL, prepare the information and make a SQL query to store that information into the 'currentTemp' table inside the database.

3.7.2 Proctemp2.php

The main difference between proctemp2 and procTemp is that this version is called by the external sensors. This is important because as discussed in the previous sections, the exterior sensors are the ones that trigger the database to update and combine the exterior and interior temperatures. This is done to avoid having a program running to combine the two and simply using a different PHP script during the sensor measurement, necessitating less processing power.

The way this script works is similar to procTemp but using a while statement. In pseudocode, the script's behaviour is:

1. Get last 10 readings from 'currentTemp'
2. Iterate through the last 10 readings
3. If the reading is NOT older than 30 minutes
 - a. Update Record to 'PROC' = 1
 - b. Save Temp / Hum / Date / Room_ID in variables.
 - c. Insert Exterior Readings and the values in variables to 'procReadings'
4. Insert current exterior temperature to 'currentTemp'

This is how we're able to connect the entire pipeline from getting the reading on the sensor, sending it through the network, getting into Kubernetes NodePort, into the apache web server, running the PHP script and finally writing into the SQL database.

3.7.3 Dashboard

This is perhaps the only part of the entire project that doesn't affect the functionality of the cluster or its applications, its only purpose is to inform and describe the current situation inside the cluster and the sensors. This would be the only part of the project that would be accessible to an end user and as such it is no less important than other functional requirements in the project.

We need to offer a clean, efficient and informative dashboard for the users to monitor what their cluster is doing and how the sensors are working throughout the house or enterprise.

We opted for a clean, in house implementation of a responsive web page. Able to be viewed on tablets, phones and PC's without any loss of usability.

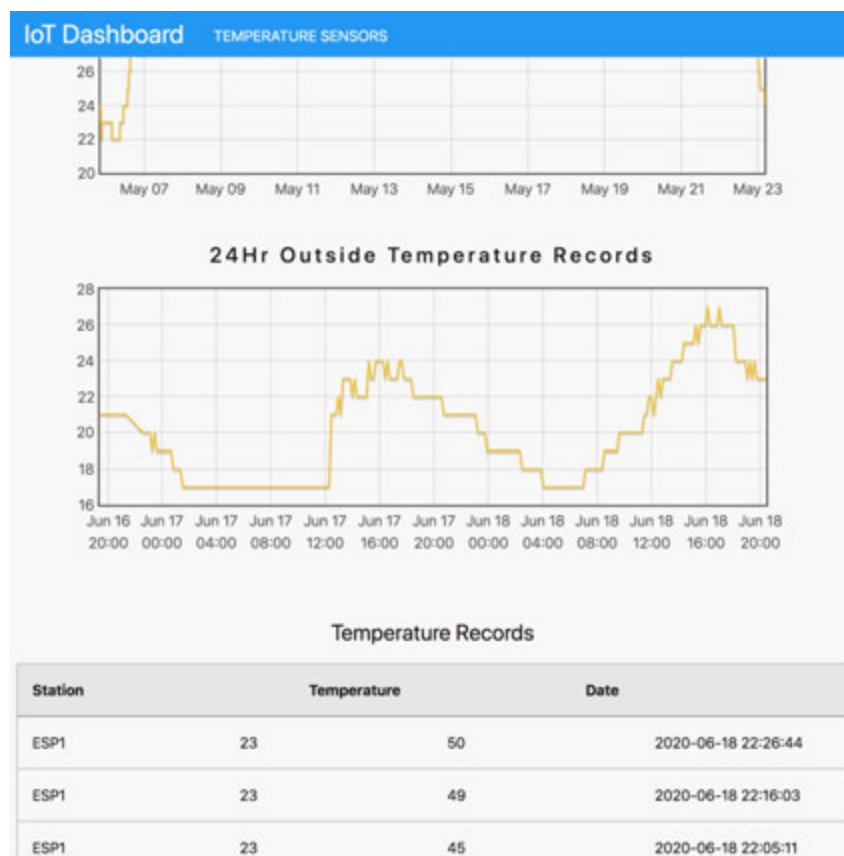


Fig. 3.22 Dashboard in a laptop browser.

As we can see, the dashboard offers a clean, minimalist design and offers two prominent interactive graphs to visualize the 24Hr temperatures inside and outside. We have also built a 'sticky' navigation bar complying with Nielsen's usability principles and allow the user to always return to their 'home' page.

As we scroll further down the page, we can go into more in depth readings of the database, as we get a 'raw' dump of the data contained in the 'currentTemp' table. These records have the temperature, humidity, date and station where the measurement was taken.

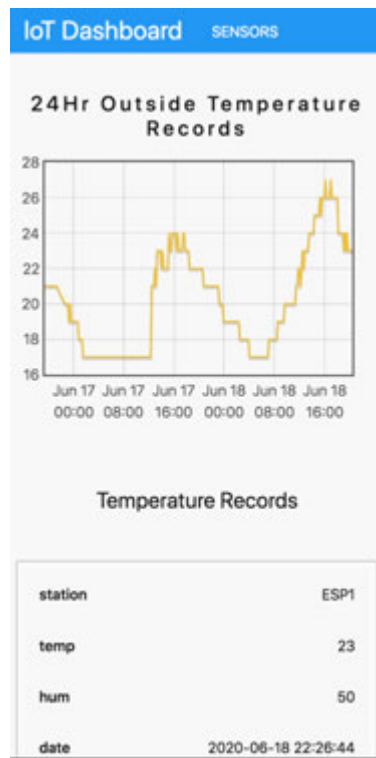


Fig. 3.23 Dashboard in a mobile browser.

We can also see that we have made the whole website responsive, able to adapt to any device accessing the website.

To be able to display the graphs we use Plotly, built on JavaScript and jQuery. To get the dataset required to display the graphs we simply query the SQL database and use a few commands to get everything working.

We have encountered a lot of bugs with Plotly and our database because the timezones and the timestamps wouldn't show up correctly on the graph. To fix this we had to adjust the scale in the timestamp and use UNIX style timestamps (in milliseconds)

The result is being able to view interactive plots in any screen size, orientation and position, the page adapts to the screen size of the user in real time.

3.8 Feeding the database from external sources

Originally, we were building a database from entirely on-site sensors, but due to original Neural Network performance we researched ways to improve our scores. We concluded that adding information about the exterior weather increased our chances for our Neural Network to be able to correctly identify cases in which the external temperature and humidity weren't indicative enough.

Another reason why we included external sources is to showcase interaction between the internet and a deployment running on Kubernetes. This might seem trivial, but due to the complex way Kubernetes lays its containers on pods and those pods are wired with specific ClusterIP's we must follow a specific set of instructions that we've curated to be able to communicate from a deployment running on Kubernetes to the wide internet.

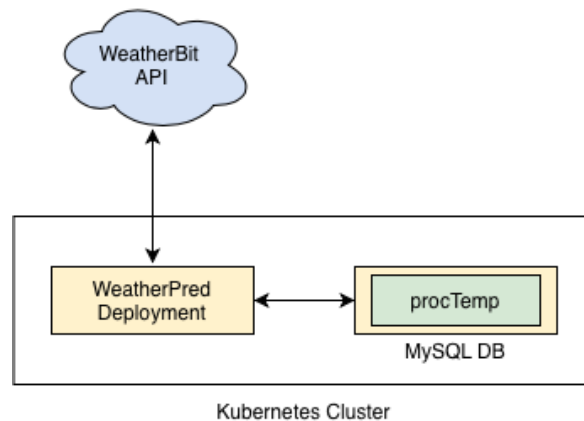


Fig. 3.24 Connection between the weather prediction application, the API and the database.

During our research we found various alternatives that could offer a weather API service. This aspect of the project needed special consideration because it could introduce a cost variable, as well as being tied to the services of another company.

We settled on Weatherbit [16], this API service offers a lot of data as well as historic data in case we need to retroactively fill incomplete data. There are various tiers of pricing starting at 0 USD for a non-commercial limited use and a 35 USD tier that allows commercial use. Since this project is educational, we are going to use the free tier, but will take the 35 USD tier into account in the price breakdown of the entire project in case this project was brought to market.

Free	Starter	Developer	Advanced
\$0/mo	\$35/mo	\$160/mo	\$470/mo
Includes: Daily forecasts Current weather data Non-Commercial Use	Includes: Daily forecasts Current weather data Hourly forecasts Commercial Use License	Includes: Daily forecasts Current weather data Hourly forecasts Historical data (1 year) Commercial Use License	Includes: Daily forecasts Current weather data Hourly forecasts Historical data (10 years) Air Quality / Energy / Climate / Agweather API access Commercial Use License
Sign Up	Sign Up	Sign Up	Sign Up

Fig. 3.25 Weatherbit API pricing tiers.

The only feature we're going to use is the current weather data, that way we avoid dealing with predictions or forecasts and we get an accurate 'current' reading of the weather.

Let's take a deeper look into what an example API request is like and what the response is from the server. We will need to know what to expect and how to parse the reply from the server to obtain the information.

An example API Request could be:

```
> https://api.weatherbit.io/v2.0/current?city=Madrid&country=ES&key=K
```

Or, for a more precise measurement we can use latitude and longitude coordinates

```
> https://api.weatherbit.io/v2.0/current? key=API_KEY&lat=38.123&lon=-78.543
```

Our response from the server will be formatted in a JSON dictionary format, this will allow us to use a very simple JSON parser/wrapper in Python to extract the data.

We will quickly look over all the available information that the JSON file contains and explain which of the values we're going to use in our program.

- lat: Latitude (Degrees).
- lon: Longitude (Degrees).
- sunrise: Sunrise time (HH:MM).
- sunset: Sunset time (HH:MM).
- timezone: Local IANA Timezone.
- station: Source station ID.
- ob_time: Last observation time (YYYY-MM-DD HH:MM).
- datetime: Current cycle hour (YYYY-MM-DD:HH).
- ts: Last observation time (Unix timestamp).

- city_name: City name.
- country_code: Country abbreviation.
- state_code: State abbreviation/code.
- pres: Pressure (mb).
- slp: Sea level pressure (mb).
- wind_spd: Wind speed (Default m/s).
- wind_dir: Wind direction (degrees).
- wind_cdir: Abbreviated wind direction.
- wind_cdir_full: Verbal wind direction.
- temp: Temperature (default Celcius).
- app_temp: Apparent/"Feels Like" temperature (default Celcius).
- rh: Relative humidity (%).
- dewpt: Dew point (default Celcius).
- clouds: Cloud coverage (%).
- pod: Part of the day (d = day / n = night).
- vis: Visibility (default KM).
- precip: Liquid equivalent precipitation rate (default mm/hr).
- snow: Snowfall (default mm/hr).
- uv: UV Index (0-11+).
- aqi: Air Quality Index [US - EPA standard 0 - +500]
- dhi: Diffuse horizontal solar irradiance (W/m²) [Clear Sky]
- dni: Direct normal solar irradiance (W/m²) [Clear Sky]
- ghi: Global horizontal solar irradiance (W/m²) [Clear Sky]
- solar_rad: Estimated Solar Radiation (W/m²).
- elev_angle: Solar elevation angle (degrees).
- h_angle: Solar hour angle (degrees).

In our application, since we're dealing with temperature variation (the Neural Network will try to predict interior temperature variances) we need to look at those values in the JSON file that could influence temperature and that we can't measure ourselves with a sensor.

The values we have decided to measure are:

- UV Index: Since it is a measurement on how much UV light is shining (a key component in heating up a house)
- Solar Radiation: An even more precise measurement on how much energy is transmitted by the sun.
- Cloud Coverage Percentage: This will tell us how much of the sky is covered by clouds in % affecting directly how much sun is available to heat up the house.

Now that we have selected our attributes, we need to start building our application to process these API calls and update the 'procTemp' table in our MySQL server. We will be using Python as our language of choice for this program, as we are also using Python for our Neural Network and we can simply add this functionality to the Neural Network program in the future.

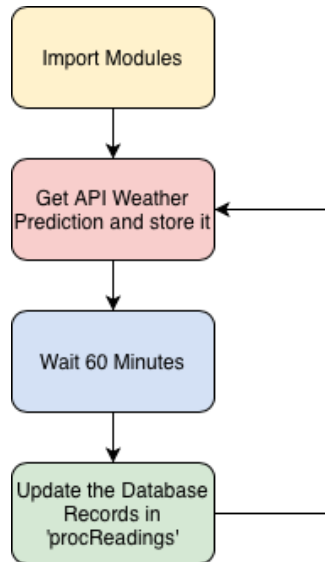


Fig. 3.26 WeatherPred Program Flowchart.

We will go into detail on how to build this program and then how to build it into a docker image and finally deploying it into the Kubernetes cluster in the implementation chapter.

Since this is a timed application and we're going to be running it for a long time in a mostly sleeping phase, we need to take into account that the program is not going to be consuming any resources while it sleeps. To do this we can use a scheduler using the sched module, set a function to run and set a timer to repeat every x seconds.

Once programmed and deployed into the cluster we should see our database getting filled out correctly.

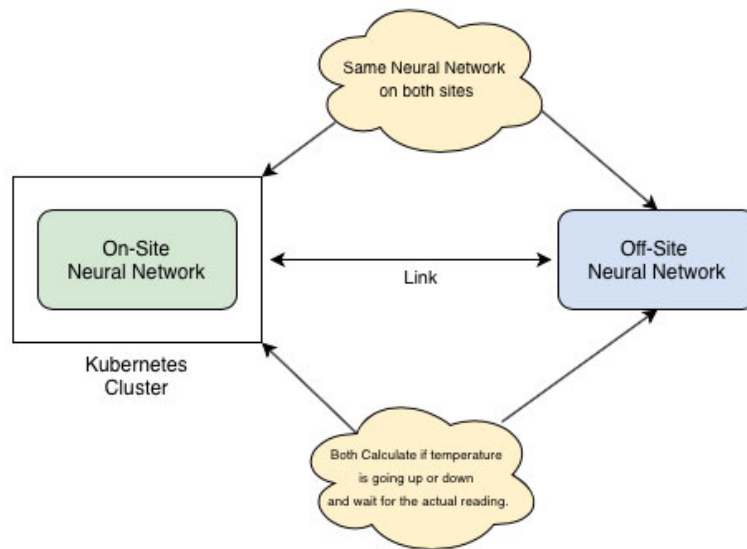
id	temp	hum	out_temp	out_hum	clouds	solar_rad ▲ 1	uv	date	room_id
34524	21	33	13	68	26	848.474	6.15338	2020-04-04 14:04:00	4
34525	21	33	13	65	26	848.474	6.15338	2020-04-04 14:14:05	4
34526	21	33	14	62	26	848.474	6.15338	2020-04-04 14:24:10	4
34527	22	33	13	66	26	848.474	6.15338	2020-04-04 14:34:14	4
34528	22	33	13	65	26	848.474	6.15338	2020-04-04 14:44:19	4
34529	22	33	14	62	26	848.474	6.15338	2020-04-04 14:54:23	4
34530	22	33	14	62	26	848.474	6.15338	2020-04-04 15:04:27	4

Fig. 3.27 Complete 'ProcTemp' Database Filled by the weatherpred Application.

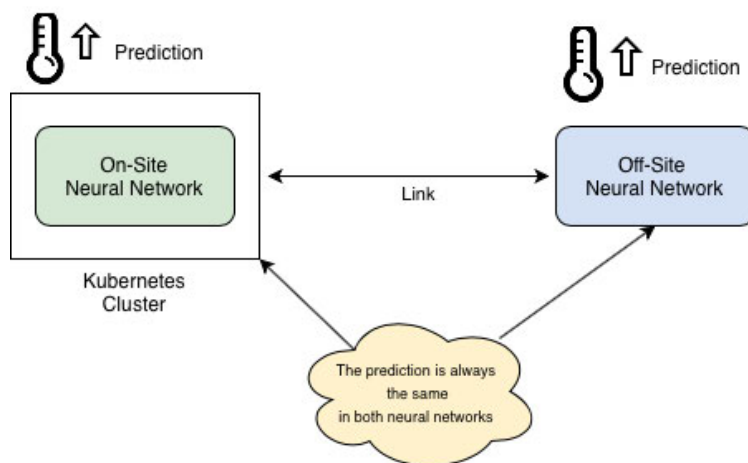
3.9 Introducing our Neural Network

The reasoning behind using AI features, in particular, a Neural Network is to fully drive home the point of Edge Computing and showcasing how AI features in traditional programs can benefit non-traditional uses.

Since we're building a computing cluster on-site, inside a home network, we need to have a link between the home network and an off-site backup/ agglomerate that can process the information. In an effort to optimize this, we can use the Neural Network to 'predict' when there is going to be a change in temperature, so we can anticipate it and not need to send that information through the link, further optimizing the link and reducing the amount of data that is sent.



(a)



(b)

Fig. 3.28 First Two Neural Network Phases.

If the prediction is ‘correct’ [Fig 3.29] the home cluster won’t send the information as the Neural Network running off site will already know that the information won’t change from previous readings. If the prediction is wrong [Fig 3.30], then the on-site cluster sends the information to the off-site location so that the Neural Network can be trained to hopefully prevent further incorrect predictions.

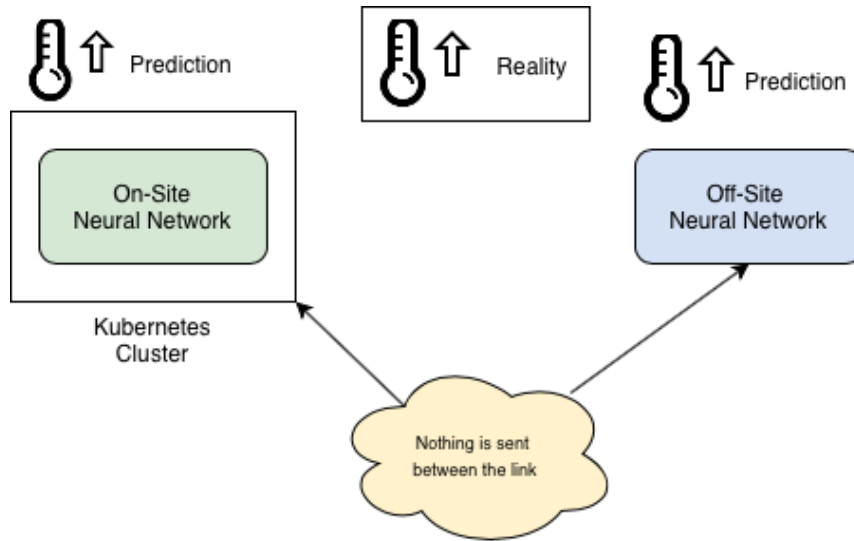


Fig. 3.29 Third Phase in Neural Network.

As seen, if the prediction matches the reality nothing is sent between the link joining the on-site cluster and the off-site network.

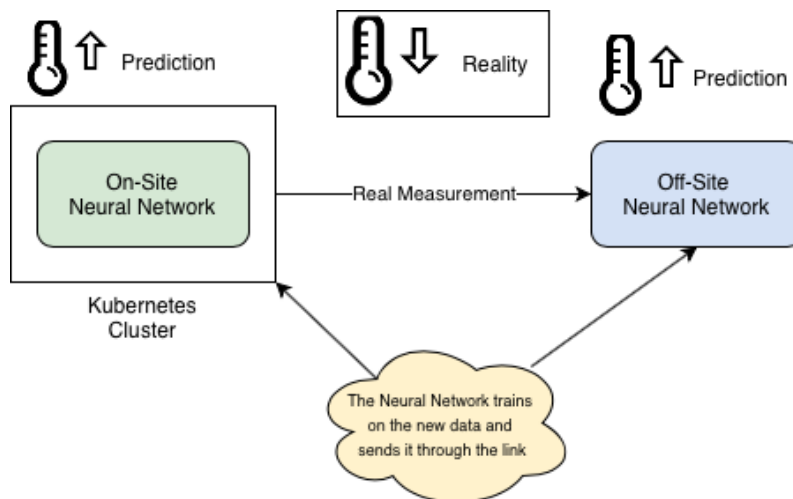


Fig. 3.30 Fourth Phase in Neural Network.

If the real measurement doesn't match the prediction, the following occurs:

1. The onsite Neural Network will train again using the new data to limit the possibilities of the same error occurring again
2. The Neural Network will send its parameters and the real measurement through the link to the offsite Neural Network.
3. The off-site Neural Network overwrites its parameters with the ones received from the Kubernetes cluster.

Ordinarily, the second step will put more strain on the link than just sending the measurement and having no network, but we need to remember that the sending only occurs when there's an error in prediction, as such if we aim for a >60% accuracy in the Neural Network, we can start to see a benefit to this arrangement

3.9.1 Neural Network architecture

Firstly, we need to generalize the problem in question. Since we're dealing with a short number of rows of data, we can't expect incredible performance out of the Neural Network. This is simply because weather is an extremely nuanced task, with many variables that all contribute to a different outcome.

As such, we have decided to try to predict a change in temperature, instead of trying to predict the temperature itself. This is because our original problem would be a non-linear regression which is historically a very difficult task. Instead we have transformed the problem into a non-linear classification problem with two classes:

- Temperature stays or goes low
- Temperature increases

This allows us to still do our Edge Computing features as described beforehand and work with a much higher accuracy rate which will improve our performance even more.

The issue with Neural Networks is that it takes an incredible amount of time to fine-tune all of the different parameters associated with one. There are many variables like network architecture, epoch length, optimizers, models, activation functions, etc.

We will start first with arguably the most important part of a Neural Network, the data. There is a vast amount of pre-processing that needs to be done before we can start to feed the data into the network.

The pre-processing goes as follows:

- Creation of a target function, we need to tell the Neural Network what the 'correct' results is so it can compare when it produces the wrong prediction and that it can learn from those errors.
- Removal of unnecessary columns, once we have loaded our database, information like room_id or date aren't relevant (date is implied in the time series).

- Normalization of data, due to the way our activation functions operate, we need to reduce all values of the database two values in the range [0-1] so that there aren't any extremes that would introduce a bias towards a specific value.
- Removal of incomplete rows: this is important because we cannot feed a Neural Network incomplete data, or else it could learn from those empty fields and generalize incorrectly for a wider data set.
- Creation of sequences, this isn't required for every Neural Network but given we are using a Recurrent Neural Network (RNN) we need to create a sequence that contains all the data to be learnt from.

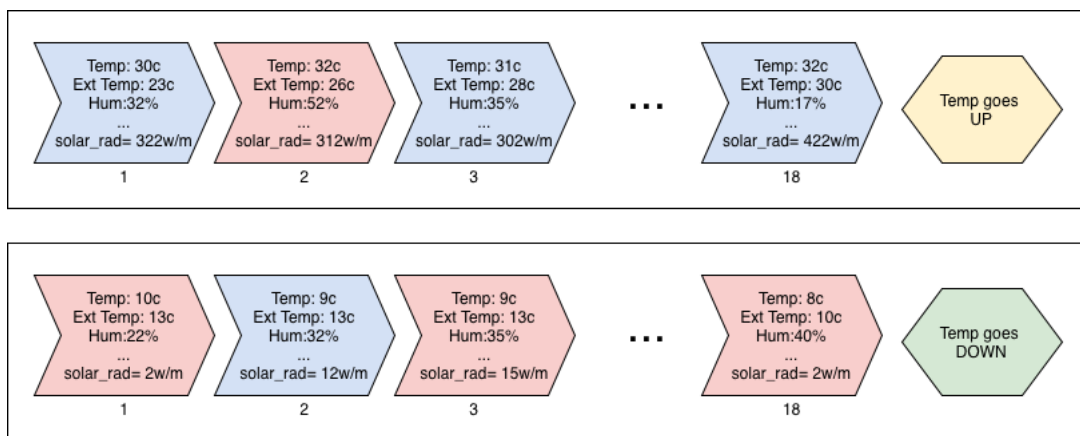


Fig. 3.31 Two sequences that the Neural Network will train from.

As we can see, a sequence contains a number of ‘rows’ of the database and a target (temp goes up, temp goes down / stays). The number of rows in a sequence is arbitrary and must be determined by trial and error. We have established 18 as a good balance between looking back into the past and enough performance so we have enough sequences to learn from.

A high number of rows in sequences limits the amount of learning as eventually you miss out on the details and try to generalize more. A low number of rows in a sequence doesn't provide enough history to gather data from. These sequences are what the Neural Network learns from. Once trained, we must feed an 18-row sequence to the network to get a result (temp goes high or low).

Continuing with pre-processing the data, we still need to:

- Balance the data, this is an extremely important and self-explanatory part of the pre-processing, the sequences that we have obtained must all be balanced in terms of how many highs and lows there are in total.

For example, if we have 200 sequences that have ‘temperature goes up’ as their target and only 50 that have ‘temperature goes down’ this will create a bias towards the ‘temperature goes up’ category. To solve this, we simply reduce the amount of sequences to match each other, meaning, we would have to discard 150 ‘temperature goes up sequences’ to have 50 of both types.

- Randomize data, fairly self-explanatory, we need to have a completely random dataset in terms of sequence placement. We only randomize the placement of each sequence but not the sequences themselves.
- Split the dataset, we need to set training data and evaluation data, this is because any Neural Network could learn from training data, what we need to make sure is that the network performs well with data it has never seen before. For this we split 95-5 into training and evaluation sets.

Once our pre-processing has been completed, we need to start looking at network architecture. A Recurrent Neural Network is specifically built to handle time series, it differs from a normal Neural Network in that the connections between nodes form a directed graph and use their internal memory to process variable length inputs.

In particular, we are going to use LSTM networks. Long short-term memory (LSTM) networks were invented by Hochreiter and Schmidhuber in 1997 and set accuracy records in multiple applications domains [17].

LSTMs are characterized for having feedback connections, it can not only process a single data point, but it can also feed entire sequences of data (precisely our use case).

A common LSTM unit is comprised of the following:

- A cell that houses the layers and pointwise operations
- An input gate that brings in the data
- An output gate that outputs the data
- A forget gate that discards data

Looking at the architecture more closely, we can see each of the layers and pointwise operations.

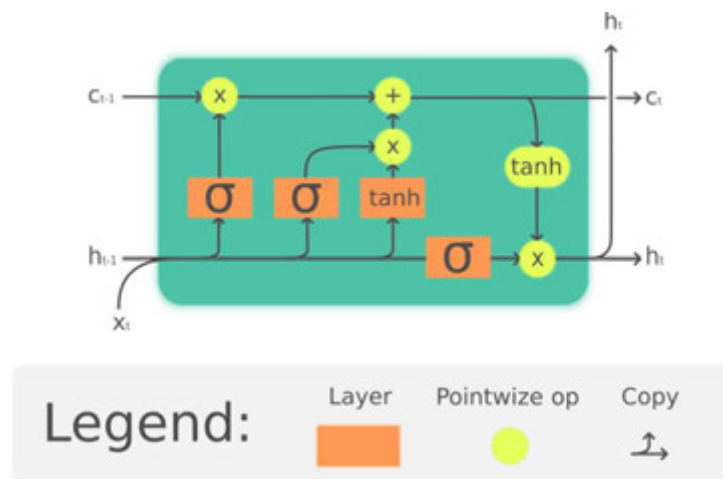


Fig. 3.32 LSTM Cell [18]

This type of LSTM cell gives us much more individual control over the data than using standard cells, it allows us to have a much finer control over the individual datapoints and sequences. For example, it can discard a single row or data point in a sequence if it thinks that the value is not significant to the overall result.

We then need to formulate a valid network architecture that allows us to properly adapt a RNN into our own problem. The best way to test an architecture is to simply run a battery of tests to analyse performance changing the layers and the number of cells.

The chosen layout for our RNN comprises of 3 convolutional layers that will process the data with 128 LSTM cells, these layers will be connected with each other and have a dropout (we drop 20% of the worst performing cells) until we arrive in the fourth layer.

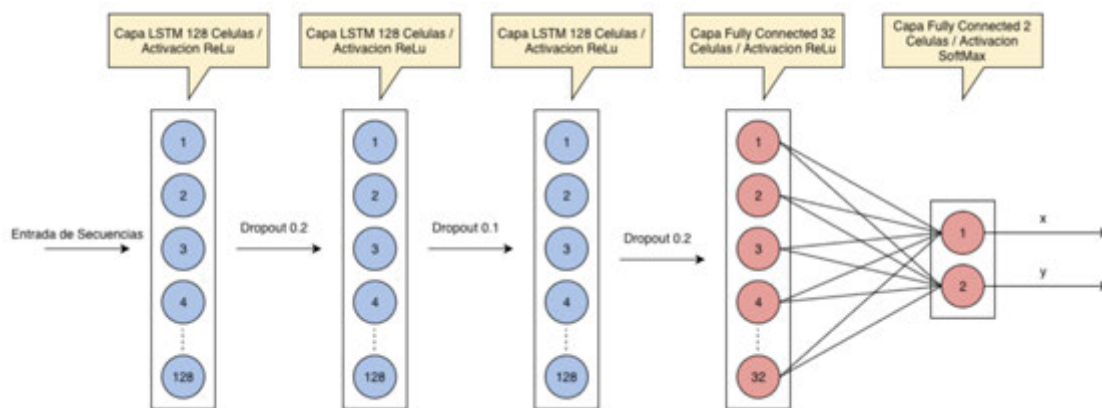


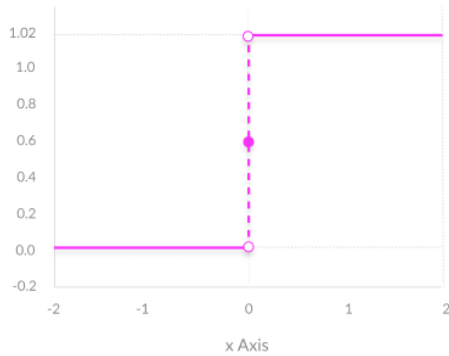
Fig. 3.33 Neural Network Layout and Architecture.

The fourth layer gets 32 cells instead of 128 simply because we change the type of cell that is being used. We change from LSTM cells to fully connected cells, or perceptron cells. The difference with these cells is that the cells have links with every other cell in the next layer, meaning, there is a value (weight) for each link meaning that they are ‘fully connected’

It becomes apparent that we suffer a big problem with space as soon as we have lots of fully connected cells, this is why we only run fully connected cells at the end, after running 3 dropouts and only with 2 cells in the final layer.

The final layer will always need to bear resemblance with how the problem is formulated. Since our problem is a non-linear classification with two classes, we need to set our final layer to the output of the problem, leaving us with two cells in our last layer.

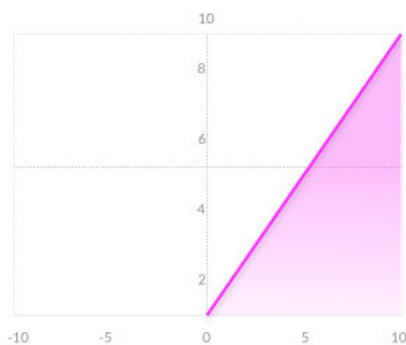
We will use two types of activation function. The activation function defines the output of a given cell. It is the equivalent of how the output behaves to a given input. There are linear and non-linear activation functions. The most basic activation function is the equivalent to an ‘on/off’ switch.



$$f(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases}$$

Fig. 3.34 Binary Step Activation Function [43].

X being the input weight that varies between problems. Since we are working with a non-linear problem though, we can’t use these types of activation functions. We need to use a non-linear activation function, for this, the most optimal one is a ReLU (Rectified Linear Unit).



$$f(x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases}$$

Fig. 3.35 ReLU Activation Function [43].

The advantages of ReLU over other non-linear activation functions is that it is very cheap to compute since the function is simple. It has a fast convergence; the linearity means that the slope never hits a plateau and that it doesn’t have problems where it gets ‘saturated’ when ‘x’ gets too large.

Finally, for our last 2 output cells we cannot use a ReLU simply because our activation cannot be 0, and we need to use a specific activation function for non-linear classification.

Introducing the SoftMax activation function, it allows to turn the numeric output of the last fully connected layer into a probability. The way it can do this is with the following

equation:

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}} \text{ for } i = 1, \dots, K \text{ and } \mathbf{z} = (z_1, \dots, z_K) \in \mathbb{R}^K \quad (1)$$

In words, we have to get the exponential function to each element Z_i of the Z input vector and normalize all of these values by dividing it by the sum of every exponential in every element Z_i . This normalization guarantees that the sum of the elements of Z_i will add up to 1.

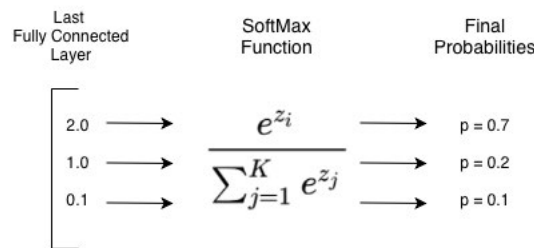


Fig. 3.36 Inner Workings of the Last Layer in the Neural Network.

Now that we have established a valid and efficient network architecture, we need to find a way to measure if our network is performing correctly. We cannot simply use the % of accuracy since that wouldn't paint a picture on how close or how far the guess was off. Since our SoftMax layers are returning probabilities, i.e. a chance of the temperature going up vs the chance of temperature staying or going low, we need to quantify how far off the Neural Network was from a correct guess.

To solve this, we are going to use a special kind of metric called Sparse Categorical Cross Entropy (SCCE) [19]. We use SCCE in problems where only one result is correct such as our use case. SCCE will compare the distribution of the predictions, meaning, the activations in the output layer that will be compared against the 'true class' coming from the results (the sequences we generated before).

$$J(w) = -\frac{1}{N} \sum_1^N [y_i \cdot \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)] \quad (2)$$

$J(w)$ represents the SCCE value where:

- w represents the model parameters (weights of the RNN)
- y_i represents the 'true' label / class
- \hat{y}_i represents the 'predicted' label coming from the RNN

As such we can reliably get a reading of how accurate our network is being and if we're making correct improvements to its performance.

IMPLEMENTATION

3.10 Cluster setup

One of the main objectives of this project is to not only demonstrate how powerful and useful a Kubernetes cluster is but also to make it more accessible for other users to be able to build a cluster themselves.

As this is a very enterprise-level solution for many, tutorials and information on how to set everything up and deploying your applications is scarce, and we wanted to make a part of this project open and accessible for many to follow.

As such, during the duration of this chapter, we will be discussing setup and debugging, as well as troubleshooting some common problems that we have had to fix and building our chosen use case.

Our use case scenario will encompass how a smart home, or a small business could use edge computing paired with Kubernetes to deploy its applications and store its data locally without needing offsite hosting and expensive broadband internet connections to make use of their services. This use case scenario has seen business use, more recently in fast food restaurants where they run all of their applications on-site (Kiosks and information banners, PoS systems and databases) and use Kubernetes to deploy the applications to a local cluster. [20]

In our use case scenario, we will be using two nodes in our cluster. These will be two Raspberry Pi 4B 4GB Ram boards, loaded with a light version of Linux to help performance.



Fig. 3.37 Raspberry Pi 4B 4GB Version [21].

The main reasoning for the use of Raspberry Pi is its price. At the time of writing, you can get the 2GB RAM version for approximately EUR 32 and the 4GB RAM version for EUR 64.

The use of these small boards will demonstrate the advantages of Kubernetes with a small budget and will allow us to see the scalability of the platform.

Firstly, we need to flash the SD cards of both Raspberry Pi's with the latest version of Raspbian lite [22], this step is widely documented so we don't see the need to go into detail. We will then boot up both boards and begin the setup process.

We soon stumble into one of our first problems with the setup, we need to make sure our boards have static IP addresses, this is because when we are using our applications and our IoT boards, we need to point them towards a static IP address that is always reachable. At the current time of writing, the IoT boards and software we use can't make use of DNS or DHCP, so we need to take this step into account.

To fix this, we just need to change the file located at `/etc/dhcpd.conf` and make sure our interface has a static IP. For the duration of this project we will have the following network settings:

- Ras1 (master) = 192.168.1.110
- Ras2 (worker) = 192.168.1.111
- IoT Sensors = 192.168.1.120-123

Now we need to enable some container features in the kernel, editing some key-value pairs inside the `cmdline.txt` registry in `/boot`.

The only thing that is required is to add `"cgroup_enable=cpuset cgroup_memory=1 cgroup_enable=memory"` to the end of the file, save and reboot the Raspberry.

Since we are using an ARMv7 architecture running on the raspberry pi, we do not have a full Kubernetes installation like we would have on an x86 / ARM64 architecture. We have to use a special distribution or 'flavour' of Kubernetes that runs on ARMv7. This is 'k3s'¹, built specifically as a lightweight Kubernetes distribution that runs on low-power devices.

Now we're going to set up k3s. First, we download the latest version available from the release page of k3s on GitHub. As this is an ARMv7 board, we'll need to select the `armhf` version of k3s, but it is fully compatible, and in further sections, we will discuss the integration possibilities of the cluster on other architectures. Using `wget` and the GitHub link we download k3s to a directory on the Raspberry Pi.

Once downloaded, we need to make the file an executable using `'chmod +x'` on the file we just downloaded.

We can rename the file to k3s using `'mv k3s-armhf k3s'`

3.10.1 Server (master) setup

Once we're ready, we can run `'sudo ./k3s server'` to begin the initial start-up of Kubernetes. The initial start-up will take around a minute because the private keys need to initialize. Once it's done setting up, we can stop the process and send it to the background using `^Z` and `bg`

We can get our first glimpse of the cluster using `"sudo ./k3s kubectl get nodes"` furthermore we can take a look at the whole single node cluster by running `"sudo ./k3s kubectl --all-namespaces=true get all"` to view the entire cluster including pods, services, deployments, and the replicasets.

¹ 'k3s' Lightweight Kubernetes: <https://k3s.io>

3.10.2 Worker setup

To set up the different nodes (regardless of their architecture) we need to get the node token or cluster secret to add nodes to our cluster. In our master machine we simply run:

```
> cat /var/lib/rancher/k3s/server/node-token
```

and save the output for our nodes. Then we need to find the IP address of the master and the cluster port. Since we manually set up our IP addresses before we only need to find the cluster port. For that, we run:

```
> cat /etc/rancher/k3s/k3s.yaml | grep kubeconfig
```

and find the port behind the 127.0.0.1, usually, it's port 6443.

Then we go to our k3s folder and run the command

```
> sudo ./k3s agent -s https://192.168.1.110:6443 -t NODE_TOKEN
```

replacing the IP for your IP and the node token for the output we found on the master node.

Immediately after running that command, we will see that the second Raspberry has joined our cluster successfully and that if we run:

```
> sudo ./k3s kubectl get nodes
```

We can see a second “worker” node ready to be used in our cluster.

```
pi@ras1:~/k3s $ sudo ./k3s kubectl get nodes
NAME      STATUS    ROLES    AGE   VERSION
ras2     Ready    worker   27s   v1.15.4-k3s.1
ras1     Ready    master   16h   v1.15.4-k3s.1
pi@ras1:~/k3s $
```

Fig. 3.38 Active Nodes in the Cluster.

3.10.3 Deploying applications

Yaml files are the way that Kubernetes applies and configures settings. We need to follow a certain pattern and apply it to the cluster. In our case, our deployment yaml file for the MySQL database will be:

```
apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2
kind: Deployment
metadata:
  name: mysql-deployment
spec:
  template:
    metadata:
      labels:
        app: mysql
        tier: mysql
    spec:
      containers:
      - image: hypriot/rpi-mysql:latest
        name: mysql
        env:
        - name: MYSQL_ROOT_PASSWORD
          valueFrom:
            secretKeyRef:
              name: mysql-pass
              key: password
        ports:
        - containerPort: 3306
          name: mysql
        volumeMounts:
        - name: mysql-persistent-storage
          mountPath: /var/lib/mysql
      volumes:
      - name: mysql-persistent-storage
        persistentVolumeClaim:
          claimName: my-claim
```

As we can see, we have a specific pattern we need to follow, the most important parts being:

- Declaring the kind: This will tell Kubernetes what kind of effect or service we require from the .yaml file (Deployment, Pod, Node, Service, Configuration, ...).
- Declaring the spec: A spec is what the deployment is going to contain, since we are deploying a container we need to specify one and the following information.
 - Image: What the container image is (the application itself) in this case, a MySQL database.
 - Env variables: The variables the container needs to operate (these are different depending on the container).
 - Ports: what ports the container needs to operate (3306 for MySQL).
 - Volume Mounts: where the data (files) are going to be stored for the container.
- Declaring persistent storage: If we require persistent storage (storage doesn't get deleted after each container restart) we need to declare a PVC (Persistent Volume Claim).

3.10.4 ESP board setup

Firstly, there's no underlying operating system on the board, and the filesystem must be managed manually. There is also a lack of PIP, the package installer for Python and if we need to use an external library, we must download it as a .py file and upload it into the board using the boards filesystem.

To use Python on an ESP32 board we have based our work on diverse sources that support the MicroPython project (Python for ESP boards) and other bug fixes found on these forums. [23] [24]

We firstly need a serial port driver to communicate with the board, using UART which is the protocol that serial devices need to communicate with each other. The drivers change between architectures and as such will need to be investigated for each computer that is connected to the board. In our case, we're using an x86 architecture on MacOS and have found success using the CP2102 USB-to-Serial Bridge Driver [25].

Once we can communicate with the board via Serial / USB we can start to flash the firmware that the ARM chip will run. Since we have decided to use MicroPython we simply need to download its latest firmware, and flash it using ESPTool.py [26] a Python written tool that allows us to flash ESP firmware to ESP boards through the terminal.

Firstly, we need to erase the board:

```
esptool.py --chip esp32 --port /dev/cu.SLAB_USBtoUART erase_flash
```

where we specify the board `--chip esp32 / esp8266` and then the driver.

And then we load our firmware into the ESP board with the following command

```
esptool.py --chip esp32 --port /dev/cu.SLAB_USBtoUART write_flash -z 0x1000 esp32-idf3-20191101-v1.11-549-gf2ecfe8b8.bin
```

declaring the board, and where to write the firmware (position 0x1000 is the boot portion of the flash memory onboard an ESP32 board [27])

After this we will have successfully flashed an ESP32 board with MicroPython and we're ready to start writing Python code to run on the chip.

3.10.5 Temperature sensor setup

Using a DHT22 sensor, let's connect it to our board and write our first lines of code to make sure it functions correctly.

We then connect the wires to an available GPIO port, in our case GPIO14, attaching it with a 10k Ohm pull up resistor to smooth the digital frequency our GPIO port receives.

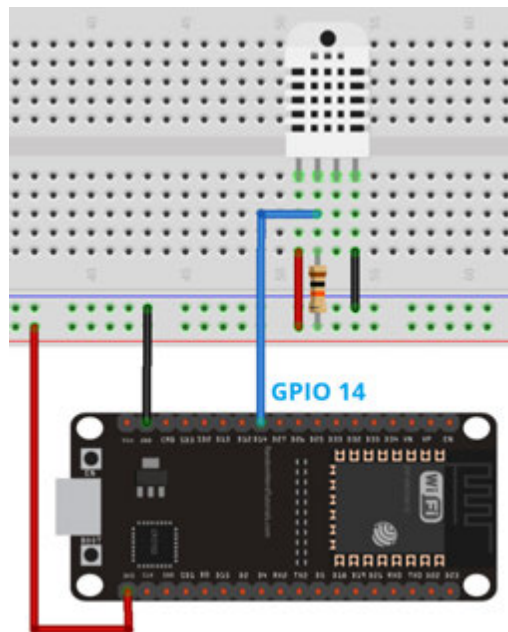


Fig. 3.39 ESP32 and DHT22 Connection using GPIOs [28].

Now we just need to run a few lines of code to get the current temperature and humidity.

```
from machine import Pin
import dht

sensor = dht.DHT22(Pin(14))
#sensor = dht.DHT11(Pin(14))
```

DHT is a library that comes with MicroPython to facilitate the use of the DHT family of sensors, we simply declare the sensor as `dht.sensor` and the `pin` its connected to (In our case GPIO14).

```
sensor.measure()
temp = sensor.temperature()
hum = sensor.humidity()
```

With those simple commands, we can get the temperature and humidity from our sensor into the board.

3.10.6 Building the complete sensor program in Python

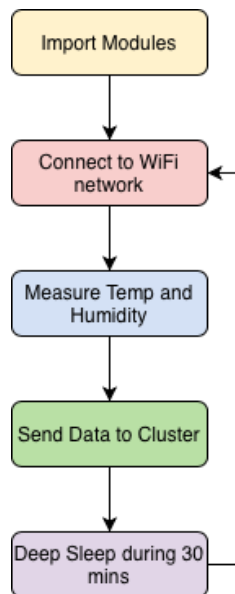


Fig. 3.40 ESP Python Program Flowchart.

Briefly, the complete program will boot up the board, connect to the network, take a measurement using the DHT22 sensor and send the data to the cluster. This will run in a loop forever.

Import modules

For this application we need some important modules, we only need one external module which is 'urequests' the MicroPython version of the popular HTTP requests package for Python.

To use this module, we first need to download it as a '.py' file and flash it to the onboard memory in the ESP board. We can do this using esptool as we did in the previous section. The entire list of modules we are going to use are:

- Usocket (for external communications with HTTP)
- Network (for Wi-Fi connectivity)
- Esp (has board specific commands to allow us communication with GPIO)
- Gc (Garbage collector)
- Time (to timestamp readings of temperature)
- Machine.pin (for GPIO communication)
- Machine.deepsleep
- Urequests (micro version of python package requests, to allow HTTP requests)

Connect to Network

This is a simple function we've created that takes advantage of the native WLAN support on the ESPx boards.

```
def connectToNet():
    global station
    ssid = 'SSID'
    password = 'SSID_PASS'

    station.active(True)
    station.connect(ssid, password)
    while station.isconnected() == False:
        pass
    print('Connection successful')
    print(station.ifconfig())
```

If we desire, we can modify the network parameters before establishing a connection, if for example, we want to set a static IP for the board. We simply use `station.ifconfig` before the connect command like so,

```
station.ifconfig(('board_ip', 'netmask', 'gateway', 'dns_server'))
```

Measure Temp and Humidity

We first need to set up the sensor location and pins like we did in the previous sections. Once that's completed, we can focus on managing the results from the sensor. To avoid extremes and since our polling rate is every 30 mins, we have decided to employ a running average of temperatures to obtain a more stable temperature reading out of the sensor.

Our sensor reads the current temperature and humidity every 4 seconds and creates an average of those two readings over a minute. Those results are then saved and ready to be transmitted to the cluster.

Sending the data to the cluster

To send the data back to the cluster, we have concluded that we will use HTTP and TCP to manage the connections, to allow for this we need to assume that the recipient allows for incoming HTTP connections. Since we have an apache server instance running in our cluster, we can simply direct our connections to that server and let it handle incoming requests. We will go into more detail on how to set up the Apache Server and the PHP scripts that allow us to communicate with the MySQL database in the next sections.

Firstly, we need to bind port 80 using sockets to our connection,

```
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind(("", 80))
```

Once this is done, we can build our URL which will contain our information. Since our values aren't sensitive, we can use HTTP GET instead of POST, which will make it easier for our PHP scripts to get the information from the URL.

```
addr='http://192.168.1.111:32151/procTemp.php?temp=%d&hum=%d&station=%s&room_id=%d' % (temp,hum,STATION_NAME,ROOM_ID)
res = urequests.get(addr)
print ("Sent Req")
res.close()
```

Basically, we're pointing the URL towards the Kubernetes cluster (192.168.1.111) then using the NodePort assigned to the Apache server (Proper Kubernetes installations wouldn't need the NodePort, LoadBalancer would take over, but we're dealing with the limitations of k3s and the Raspberry Pi)

Then we need to direct it to the PHP script that processes temperature (/proctemp.php) and then using the GET parameters to send the temperature, humidity, station number and the room id.

Then we simply send that HTTP request, the Apache Server will process the request, make sure it's coming from a legitimate ESPx board, and return an 'HTTP 200 OK' message to the board, allowing it to continue working.

After this, the board will go into a deepsleep, which is a feature only found on the ESP32, our ESP8266 will do a normal sleep. The advantages to a deepsleep is that the power draw becomes incredibly small. It is recorded at around 10 μ A of power consumption [29] and will enable the RTC to take over once the timer is depleted.

And with that, our Python program is complete. To upload the program, we just use ESPTool like before and the program will automatically run every time the RST button is pressed, or power is connected to the board.

Since we have 3 sensors, we flash this same program 3 times, changing the global variables STATION_NAME to ESP1, ESP2, ... depending on the specific board.

We also need to change the ROOM_ID in the program, depending on where the sensor is located since, we have the possibility of having multiple exterior sensors or multiple sensors per room.

In the case of the exterior sensors, we also need to change the PHP script responsible of processing the data. This is done because the exterior temperature is the trigger to update the database records into a complete row of information (more about this in the next section).

With that our ESP boards will start sending the temperature, humidity and other information to the cluster every 10 minutes and sleeping in between to maximize efficiency. We now need a way to process that information that's incoming and store it into the MySQL database inside of the Kubernetes Cluster.

3.10.7 Apache web server scripts

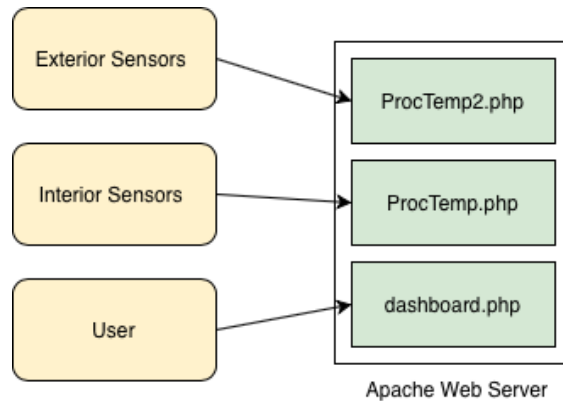


Fig. 3.41 Scripts in the web server and their connections from outside the cluster.

Proctemp.php

Both proctemp.php and proctemp2.php are purely PHP scripts since they contain no HTML or CSS code to display to a browser. The main reason they are needed is to act as a proxy between the incoming HTTP data and the MySQL database in the cluster.

Firstly, we need to authenticate into our MySQL database in the Apache server. To do this we can use the following script

```
<?php
$servername = "192.168.1.111";
$username = "root";
$password = "root";
$dbname = "db_name";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);

// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
?>
```

To find the credentials to the MySQL database, we simply look into our Kubernetes configuration, like we did in the previous chapter.

Since we set up MySQL as a ClusterIP (Only accessible within the cluster) we don't need to use a specific port like we did with the Apache Web Server. We simply direct the server name to 192.168.1.111 and use the credentials that were set on the ENV_VARIABLES portion of the .yaml deployment in the MySQL deployment.

After we have established connection with our MySQL database, we then fetch the data from the HTTP headers and URL coming from the sensor. As discussed before, we will be using HTTP GET for this, but the use of POST is recommended and encouraged with more sensitive data.


```
//getting values from form into local variables
$temp= addslashes($_GET["temp"]);
$station= addslashes($_GET["station"]);
$room_id=addslashes($_GET["room_id"]);
$hum=addslashes($_GET["hum"]);
```

The reason we use addslashes() is to protect against SQL injection attacks as adding slashes in the variables that have spaces or special characters will make it so that when they are inserted into the database, there isn't any possibility of an attack through that vector.

Using this method, we're able to get the corresponding parameters from the URL using GET, storing them as PHP variables and being ready to insert these values into our database.

```
date_default_timezone_set('Europe/Madrid');
$dt = new DateTime();
$dt = $dt->format('Y-m-d H:i:s');
```

We need to use the datetime library to get the current date and time to insert into the SQL database. Since we're going to use Python to read this data, we use a known date time format.

```
//formulating sql insert into table
$sql4="INSERT INTO `temperature`.`currentTemp` (`id`,
`temp`,`hum`,`station`,`date`,`room_id`) VALUES (NULL, '$temp', '$hum',
'$station', '$dt', '$room_id');";
$result = $conn->query($sql4) or die(mysqli_error($conn));
exit;
```

Now we simply formulate the insert command in SQL language, using the variables we had extracted from the URL previously. We then wait for confirmation that the query was successful and exit the script.

Proctemp2.php

To get the last 10 readings, we simply query the SQL database with:

```
> SELECT * FROM currentTemp WHERE room_id=1 AND proc=0 ORDER BY id DESC  
LIMIT 10
```

This will return the last 10 records ordered in descending order (newest first)

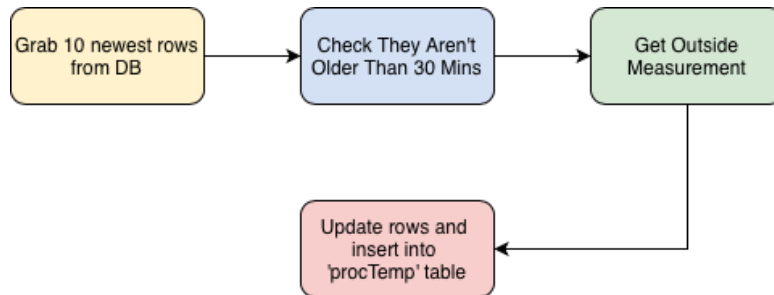


Fig. 3.42 ProcTemp2.php Program Flowchart.

We firstly iterate through the records with the while loop, saving in variables those values from the record. Then we simply compare the current time with the datestamp saved in the record. If this time is over 1800 seconds (30 mins) we discard the record since the exterior temperature isn't accurate with the time the interior temperature was taken.

If the record is less than 30 minutes old, we then insert the temperature stored in the variables.

This is how we're able to connect the entire pipeline from getting the reading on the sensor, sending it through the network, getting into Kubernetes NodePort, into the apache web server, running the PHP script and finally writing into the SQL database.

Dashboard

The way we're able to get interactive, feature rich plots into our webpage is using Plotly, a tool built on JavaScript and jQuery. What we do is query the SQL database for the specific room and time frame and encode those databases into a JSON file.

```
<script type="text/JavaScript">
$(function () {
  var d1 = <?php echo json_encode($dataset1); ?>;
  var d2 = <?php echo json_encode($dataset2); ?>;

  $.plot("#placeholder", [d1], {
    xaxis: {
      autoScale: "none",
      mode: "time",
      tickangle: 45,
      minTickSize: [1, "hour"],
      twelveHourClock: false,
      timeBase: "milliseconds"
    }
  }, {responsive: true});

  $.plot("#placeholder2", [d2], {
    xaxis: {
      autoScale: "none",
      mode: "time",
      tickangle: 45,
      minTickSize: [2, "hour"],
      twelveHourClock: false,
      timeBase: "milliseconds"
    }
  }, {responsive: true});
});
</script>
```

The result is being able to view interactive plots in any screen size, orientation and position, the page adapts to the screen size of the user in real time.

We also custom built a CSS stylesheet which will be included in the appendix.

3.10.8 Building the weather prediction service

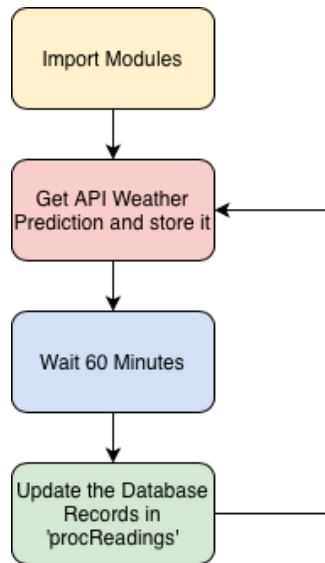


Fig. 3.43 Weather Predict Application Flowchart.

We only need to import a few Python libraries for this program:

- Time, datetime, sys: for time management and system commands
- Weatherbit.api: A Python wrapper to parse the JSON file received by the API reply
- MySQL.Connector: a very useful connector for MySQL databases written in C. Will allow us to run SQL queries on our database.
- Sched: Scheduler module for Python

Since this is a timed application and we're going to be running it for a long time in a mostly sleeping phase, we need to take into account that the program is not going to be consuming any resources while it sleeps. To do this we can use a scheduler using the sched module, set a function to run and set a timer to repeat every x seconds.

```
> s = sched.scheduler(time.time, time.sleep)
```

Will allow us to set a scheduler and then adding the target function to run in the desired interval.

To fetch the weather prediction using the Python Weatherbit wrapper we simply need to specify our options.

```
api_key = "xxxxxxxxx"  
lat = 40.432873  
long = -3.810316  
api = Api(api_key)  
api.set_granularity('hourly')  
forecast = api.get_forecast(lat=lat, lon=long).json.get('data')[0]
```

We receive the JSON dictionary with the `api.get_forecast()` call and we simply select it using the slicer “[0]”.

Then using the dictionary keys for the JSON file we have declared before we can extract the information relevant to us.

```
clouds = forecast.get('clouds')
solar_rad = forecast.get('solar_rad')
uv = forecast.get('uv')
```

We can then print traces using those variables, taking into account we need to flush the standard output since Kubernetes doesn’t register a trace with a normal print command.

Anything outputted to the standard output or error output will show up into the Kubernetes trace command.

```
print "[" + str(datetime.datetime.now()) + "] Fetched Prediction (" +
str(clouds) + "," + str(solar_rad) + "," + str(uv) + ")")
sys.stdout.flush()
```

Continuing with the flowchart, we then sleep for the required amount. Once the timer is up (set by our scheduler) we can go into updating the MySQL database. Since the weather provider (Weatherbit) provides hourly readings, it doesn’t make sense to run this program at a frequency higher than an hour.

The MySQL connector we’re using is comprised of two main components, the connection handler ‘cnx’ and the cursor ‘mycursor’. The connection is given by a MySQL connection, which must be filled exactly with the same credentials and IP address as the Apache Server, since they are both going to be running inside the Kubernetes cluster.

```
cnx = mysql.connector.connect(user='root', password=root,
                             host='192.168.1.111', port='3306',
                             database='temperature')
```

The database only updates once the connexion makes a commit and the cursor is deleted.

To optimize performance in this application, instead of updating every record in the last hour (we would need to comb through all the records) so we establish which rooms are ‘alive’ so we can eliminate a large number of rows that don’t need to be examined.

```
sqlStart = 'SELECT DISTINCT `room_id` FROM `procReadings`'
mycursor.execute(sqlStart)
record = mycursor.fetchall()
aliveRooms = []

for row in record:
    aliveRooms.append(row[0])
```

Then we simply iterate through those rooms and query the database ordered by the last 3 rows in descendent order. We only choose three as that’s the maximum amount of records that could be submitted in an hour (30~ minute intervals).

```
sql = 'SELECT * FROM `procReadings` WHERE `room_id` = ' + str(room) +  
' ORDER BY id DESC LIMIT 3'  
mycursor.execute(sql)  
record = mycursor.fetchall()
```

We must then check again that the delta between the current time and the time the reading was submitted isn't over an hour as that would mean the weather could've changed in a meaningful way and the entire reading would be invalid.

Once we have correctly made sure that our record needs to be updated with the external weather information, we update the database using this SQL command.

```
sql2 = 'UPDATE `temperature`.`procReadings` SET `clouds` = %s,  
`solar_rad` = %s, `uv` = %s WHERE `procReadings`.`id` = ' + str(  
rowID) + ';' 
```

Where the %s values are given in and committed using

```
val = (str(clouds), str(solar_rad), str(uv))  
mycursor.execute(sql2, val)  
cnx.commit()
```

We verify that everything is working correctly and run the Python program locally. We should be able to get the weather prediction, but we should get an error connecting to the MySQL database.

This is because the MySQL is not accessible outside the cluster. For security and by design, only the deployments inside the cluster can reach the MySQL database at that port. As discussed, if we need to make this connection available outside the cluster, for debug or testing purposes we can assign a NodePort to the MySQL deployment.

Now that we have our Python application running locally, we need to find a way to upload it to the cluster and let Kubernetes automatically deploy it. To do this we need to make a docker container that contains all necessary components and the Python program needed to make the program work.

3.10.9 Building a Docker container

Instead of running the program locally on one of the Raspberry Pi's we can take advantage of the Kubernetes cluster (availability, efficiency, security and more) and deploy our Python application to the cluster.

As we have established before, Kubernetes runs on docker containers and we need to package our application into one. To do this we must create a Dockerfile, a document that contains all the 'instructions' for the docker compiler, such as the base image, any dependencies or requirements, and any commands that need to be run when the container is started.

Since our application runs on Python, we need to include a base Python image. We can find a list of base images on docker hub, a website built to host docker images / containers that will be used to store or own image.

We can find the official Python image in the docker hub² and take a look at all the different versions that are uploaded in the site. We can see that there is a special version of Python, called 'alpine' which is a very lightweight version of Python with the essentials to run Python applications.

Since we aren't using any specialized Python libraries like in the Neural Network, we can choose this version using its tag. For a specific version of an image we can pull it using its tag or using 'latest' to choose its latest version.

In our case we will be using alpine version 3.11 so we simply state the following in our Dockerfile

```
> FROM Python:alpine3.11
```

We then add the .py file that contains our weatherPred application

```
> ADD weatherPred.py /
```

And tell the compiler to run the following commands to install the required dependencies

```
> RUN pip3 install pyweatherbit  
> RUN pip3 install mysql-connector-python
```

That will complete our Python installation, now we simply need to tell the docker compiler to run our Python application when the container is started. To do this we simply state the compiler to run a terminal command 'Python3 ./weatherPred.py' like we would do ourselves.

```
> CMD [ "Python3", "./weatherPred.py" ]
```

With that done, our Dockerfile is complete and we're ready to compile our Python application into a container.

² https://hub.docker.com/_/Python

Ordinarily, this next process would be quite simple. We would run one terminal command to compile, upload and process our Dockerfile and its contents into a container, but we are compiling for different platforms. Things get complicated when we do cross platform compiling on docker, since it's still a beta feature. If we had done a Kubernetes cluster with its computers running on x86, this step wouldn't be needed.

Since the Raspberry Pi's run on ARMv7, we need to find a way to compile an application in that specific platform, enter 'BuildX'

BuildX is currently a beta docker 'builder' which is essentially a compiler that deals with multi and cross platform architectures. To enable this functionality, we need to enable the experimental docker desktop builds [30].

Once enabled we can list our current builders running the following command:

```
> docker buildx ls
```

And then creating a new BuildX builder using:

```
> docker buildx create --name myBuild
```

And then using 'docker BuildX inspect --bootstrap' to make the builder active. The output should look like this,

```
bash-3.2# docker buildx inspect
Name: myBuild
Driver: docker-container

Nodes:
Name: mybuild0
Endpoint: unix:///var/run/docker.sock
Status: inactive
Platforms:
bash-3.2# docker buildx inspect --bootstrap
unknown flag: --bootstrap
See 'docker buildx inspect --help'.
bash-3.2# docker buildx inspect --bootstrap
[+] Building 4.6s (1/1) FINISHED
=> [internal] booting buildkit 4.6s
=> => pulling image moby/buildkit:buildx-stable-1 4.0s
=> => creating container buildx_buildkit_mybuild0 0.0s
Name: myBuild
Driver: docker-container

Nodes:
Name: mybuild0
Endpoint: unix:///var/run/docker.sock
Status: running
Platforms: linux/amd64, linux/arm64, linux/ppc64le, linux/s390x, linux/386, linux/arm/v7, linux/arm/v6
bash-3.2#
```

Fig. 3.44 Docker BuildX Compiler Process.

Once that is done, we can see all the supported platforms that this compiler can work with, including ARM64, ARMv7 and x86.

Now we simply tell docker to compile and push our image / container into its docker hub.

```
> build ./ --platform linux/amd64,linux/arm/v7 -t user/scheduletest:latest -push
```

Docker will run the compilation, install dependencies and when finished, we will have our image stored in the docker hub, ready to use in any computer running docker.

IMAGE	DIGEST	OS/ARCH	COMPRESSED SIZE
latest			
	Last updated 31 minutes ago		
	cdb52dd5111c	linux/amd64	32.15 MB
	ef6ecc587dc4	linux/arm/v7	28.74 MB

Fig. 3.45 WeatherPred container uploaded to the docker hub.

As we can see, we have uploaded two images to the docker hub, both in amd64 and ARMv7 platforms. We can also see the low size that these images command, merely 28Mb for the ARMv7 version which will be ran on the cluster (for reference, the image contains an entire OS, Python install, dependencies and the actual program).

We can check our docker image runs correctly by running the following command on any docker compatible PC.

```
> docker pull user/scheduletest:latest
```

And then if we run that container, we will see that in the same way our program ran before, it will start running in a container without the need to use any of our system configs or installations. It will run in its own 'bubble' or small VM where it will execute perfectly.

3.10.10 Deploying a Container to the Kubernetes Cluster

It is here where we start to realize the potential of docker and containers. For much larger and more complex projects, we would've had to go through a lot of installation, debugging, dependencies and setting up to get an application running. With docker and a container, we simply pull the image from the internet, and it will come with everything it needs to run.

The next step is to simply deploy the application into the cluster like we've done with our previous deployments. In this case we will need to specify the image location and since we don't need any persistent storage, once we deploy it, the Kubernetes orchestrator will decide which node is best to deploy into.

Our specific deployment .yaml file for our weatherPred application is

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: weatherpred-deployment
  labels:
    app: weatherpred
spec:
  selector:
    matchLabels:
      app: weatherpred
  template:
    metadata:
      labels:
        app: weatherpred
    spec:
      containers:
        - name: scheduletest
          image: user/scheduletest:latest
          imagePullPolicy: Always
          ports:
            - containerPort: 80
```

A few things worth mentioning about this deployment is that we have set ‘imagePullPolicy’ as Always, this will make it so that Kubernetes refreshes the image every time the container is started. This will make it so the application being ran is always up to date and has the latest version.

We also needed to set ‘containerPort’ to 80 as the Python application is making calls to the API service through HTTP and it uses port 80 as its destination.

Another consideration we have to make that we had mentioned in the past, is that usually connections between the internet and pods deployed in the cluster is not permitted. This is because the Kubernetes cluster runs its own networking and as such, it has its own DNS service, that allows pods and services inside the cluster to find and reach any of the intra-cluster destinations.

The problem comes when a query coming from inside a pod in a cluster is looking for a destination that is outside the cluster. For example, our weatherPred service will try to find the DNS name for weatherbit.io but when passed to the DNS query service in Kubernetes, it will return a ‘not found’ error, as the domain name for Weatherbit is outside the scope of the Kubernetes DNS service.

To solve this, we have come up with a solution not found on the internet, and we have also submitted this ‘fix’ to the official k3s Kubernetes GitHub repository and hope it is included in further revisions of this Kubernetes distribution.

The fix comprises of a change to the ‘dnstools’ pod that runs as part of the backend of Kubernetes. Usually normal users of Kubernetes wouldn’t see these ‘pods’ running on their cluster as they are in a different namespace, but we can make Kubernetes show them to us by using the following command.

```
> ./k3s kubectl get pods --all-namespaces
```

```
root@ras1:/home/pi/k3s# ./k3s kubectl get pods --all-namespaces
```

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE
kube-system	helm-install-traefik-78nkg	0/1	Completed	0	254d
kube-system	coredns-5b595cf97c-4nhzj	1/1	Running	1	27h
kube-system	local-path-provisioner-5b8648d6f6-dgpp5	1/1	Running	9	233d
kube-system	svclb-traefik-mqcn9	3/3	Running	78	254d
kube-system	svclb-traefik-j7mwq	3/3	Running	66	253d
kube-system	traefik-d869575c8-9vbgx	1/1	Running	26	254d
default	svclb-mysql-service-hgkfk	1/1	Running	0	27h
default	svclb-mysql-service-r9bkg	1/1	Running	0	27h
default	phpmyadmin-deployment-5ddbfb884-5f2mb	1/1	Running	0	27h
default	mysql-deployment-f65f88d54-66p2w	1/1	Running	0	27h
default	weatherpred-deployment-648d9c8ff6-crnl8	1/1	Running	0	27h
default	dnsutils	1/1	Running	3289	149d

Fig. 3.46 All pods running in the Kubernetes Cluster.

As we can see, we have dnsutils running (ordinarily, it would run in the kube-system namespace, but the screenshot is post fix)

Since it is just a pod, we can make changes to it via a .yaml file and apply it using kubectl. The changes we need to make are simple, we need to override the 'Corefile' file that contains all the destinations for the DNS queries inside the cluster and add another entry into the file.

The premise is that we need to forward all those DNS queries that are unanswered by the internal DNS system to an external DNS server. We have chosen 1.1.1.1 as it is arguably the best WAN DNS provider. Hosted by Cloudflare, it is reliable, fast and region agnostic.

The entire fix is written into a yaml file and applied into the cluster.

```
> cat dns.yaml
```

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: coredns-custom
  namespace: kube-system
data:
  Corefile.override: |
    forward . 1.1.1.1
```

```
> ./k3s kubectl apply -f ./dns.yaml
```

With that complete, we have successfully built a Python application, compiled it into a docker container, deployed it to Kubernetes and made it communicate with the Internet.

3.11 Building the Neural Network in Python



Fig. 3.46 Neural Network Components.

Python is the biggest player in AI programming and Neural Network usage. This is in part due to the vast number of libraries and packages that make it easy to build a really complex model such as a Neural Network.

One of the most relevant AI modules for Python is TensorFlow. Owned by Google, it is an open source software library that allows users to use widely available algorithms and techniques to build AI solutions. We are going to use this module to build our RNN and to test it with our database.

Since this program is quite long and complex, we are going to once again present the program workflow and then go into each of the subparts that constitute the entire program.

One of the main disadvantages of running a Neural Network is the amount of time, energy and processing power it requires to maintain and train. That is why its paramount to try and squeeze every last bit of efficiency out of our program. For this reason, we have decided to use advanced techniques such as thread programming in Python to allow the program to automatically allocate tasks and start and end threads on demand.

We need to remember that while the secondary threads work the main thread sleeps and stays waiting to be 'awaken' by any exception that is raised.

The secondary threads are called in different time intervals, ranging from 20 minutes to get a prediction, to various days to train the Neural Network (might be trained earlier if there is an error in prediction).

3.11.1 Main thread

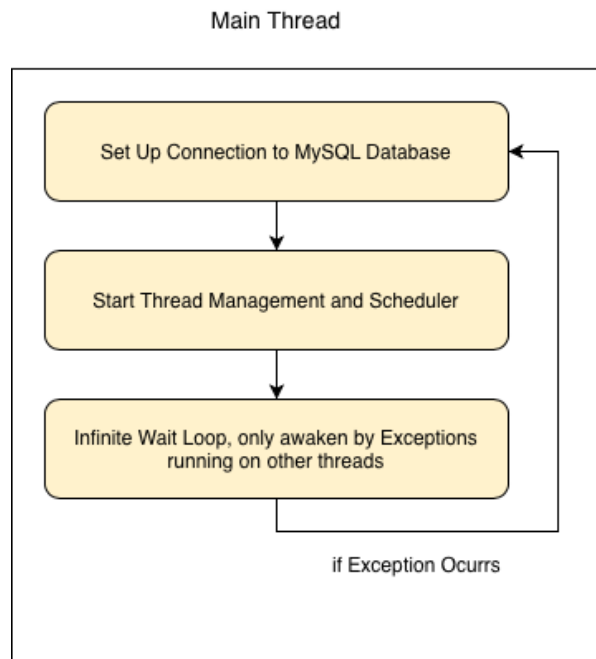


Fig. 3.46 Main Thread Flowchart.

Implementing the functionality described in the main thread is quite simple, since it doesn't really do any function but import the necessary modules and launch the secondary threads.

The modules we are going to use in this program are quite extensive, ranked in terms of importance they are:

- TensorFlow, the main AI suite, it allows us to import lots of specific tools for building Neural Networks. Specifically, we are going to import a few different tools from the TensorFlow suite
 - Backend, this is needed for a bug fix we encountered, we had an unmanaged memory leak due to the way TensorFlow and Python allocated memory and using developer backend tools we managed to stop it.
 - Sequential, this allows us to build a sequential model (needed to declare the different layers in the Neural Network)
 - Dense, LSTM, Dropout, BatchNormalization, all specific tools to build Neural Networks.
- TimeLoop, it allows us to easily declare threads, monitor exceptions and integrate it with a simple datetime scheduler to launch threads every x number of seconds.
- SkLearn, another AI suite similar to TensorFlow, we are going to use its pre-processing tools to make the sequences
- NumPy, the most important math suite in Python. It allows us to use NumPy arrays which are the de-facto way to introduce data into TensorFlow.

- Pandas, to make changes to the data using dataframes and loading the MySQL database onto a variable.
- Collections.deque a special type of list that has a maximum set length that automatically deletes from the front when the max length has been reached (useful for building the sequences)
- Random, datetime, io, sys. Various modules used for Python operations.

When the program starts, it directs itself to the main part of the program. We only need to declare the threads using TimeLoop and then go into an infinite loop that sleeps.

```
> @t1.job(interval=datetime.timedelta(seconds=3600))
def trainNN():
```

That is how we can declare a new thread and give it a timer on how often it should run. Coming back to the main, once we have declared all of our threads using the declaration above, we simply start them

```
> t1.start(block=True)
```

The flag block=True will allow us to run the threads asynchronously, meaning that one or many threads may run at the same time.

We also need the main thread to declare a couple of global constants, these will be:

- FUTURE_PERIOD_PREDICT: How many units of time we will be predicting. Since our unit of time is 10 minutes (we get a reading from the sensors every 10 minutes) we will be trying to predict 6 future periods meaning the Neural Network will predict what the temperature will do an hour from the moment it is queried.
- SEQ_LENGTH: This constant will represent how many rows of data we have in each sequence. As said before, we are setting this value to 18 rows of data (the last 3hr of weather data to predict the next hour)
- BATCH_SIZE: This unit will represent how many sequences of data will be fed at a time into the Neural Network. This value is optimized with trial and error and we have found 64 to be a good compromise between performance and time required to train the network.

3.11.2 Secondary thread: TrainNN

'TrainNN' is not indicative of the whole process that this thread accomplishes as it must also get the data from the MySQL database, pre-process all of the sequences, build the model, train it and store it on the disk.

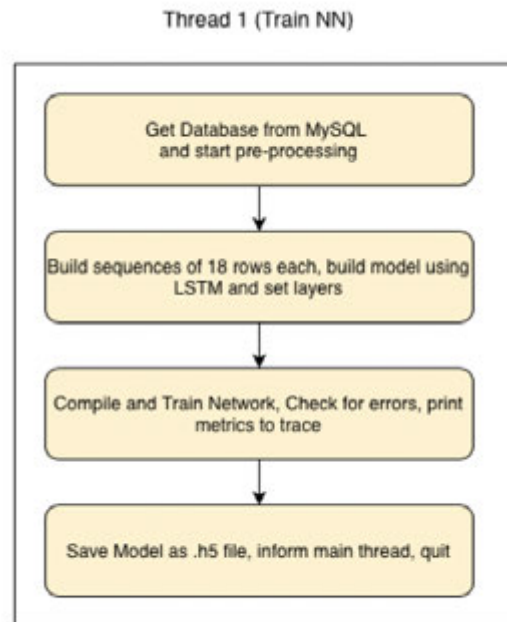


Fig. 3.47 'trainNN' Thread Flowchart.

Starting with the first item, getting the database is a simple process, we've done the same procedure with the 'weatherPred', so we simply need to query and load the database into a pandas dataframe.

```
global df
cnx = connectToDB()
mycursor = cnx.cursor()
sqlStart = 'SELECT * FROM `procReadings`'
mycursor.execute(sqlStart)
record = mycursor.fetchall()
df = pd.DataFrame(record, columns=mycursor.column_names)
cnx.close()
```

Now we will have our database stored in the df variable globally in the program. If we query the df variable in pandas we can see the following:

```

>>> roomDF.head()
      id  temp  hum  ...      uv      date  room_id
17904 17925  21.0  44.0  ...  0.975965 2019-12-22 16:21:27      4
17907 17928  21.0  44.0  ...  0.975965 2019-12-22 16:32:39      4
17909 17930  21.0  44.0  ...  0.930396 2019-12-22 16:43:50      4
17911 17932  21.0  44.0  ...  0.930396 2019-12-22 16:55:02      4
17913 17934  21.0  44.0  ...  0.930396 2019-12-22 17:06:14      4

[5 rows x 10 columns]

```

Fig. 3.48 Database loaded into the Neural Network program using pandas.

If we go back to our pre-processing explanation, we need to start to prepare the raw data starting with finding our target column. Our target will be if the temperature goes up or down in the next 18 readings. To attach this to the row we will be shifting the whole database 18 places and attaching the 'future' temperature to the row.

```
> roomDF['future'] = roomDF['temp'].shift(-FUTURE_PERIOD_PREDICT)
```

This will shift the temperature 18 places and attach it to the row

```
> roomDF = roomDF.assign(target=list(map(classify, roomDF['temp'],
roomDF['future'])))
```

That command will assign a target function, if future is higher than current temperature target will be '1' otherwise the value is '0'

```

>>> roomDF.head()
      id  temp  hum  out_temp  ...      date  room_id  future  target
17904 17925  21.0  44.0      11.0  ... 2019-12-22 16:21:27      4    21.0      0
17907 17928  21.0  44.0      11.0  ... 2019-12-22 16:32:39      4    21.0      0
17909 17930  21.0  44.0      11.0  ... 2019-12-22 16:43:50      4    21.0      0
17911 17932  21.0  44.0      11.0  ... 2019-12-22 16:55:02      4    21.0      0
17913 17934  21.0  44.0      11.0  ... 2019-12-22 17:06:14      4    21.0      0

[5 rows x 12 columns]

```

Fig. 3.49 Future and Target Columns loaded in the dataframe.

As we can see, we have created a future row and a target column. In this case the temperature remains the same in the future and the target column is set to '0'

The next step is to divide the database into validation and training data, as discussed before this is to test how effective the Neural Network is with values it has never seen before. We will use a 95-5 split to divide our sets so that the network trains with 95% of the database and tests with 5%.

```

times = sorted(roomDF.id.values)
last_5pct = times[-int(0.1 * len(times))]
validation_room_df = roomDF[(roomDF.id >= last_5pct)]
roomDF = roomDF[(roomDF.id < last_5pct)]

```


Now we need to build the actual sequences that our network will learn from. We have built an auxiliary function to take care of the pre-processing. It takes a dataframe as a parameter and returns two NumPy arrays, x and y, 'x' being the sequences without the labels (solutions) and 'y' being those labels.

We pass both our training dataframe and our validation dataframe into the pre-processing function. The first thing we do is strip the dataframe from those columns that don't give any information. These are 'future' (since we've already created target from it), 'date', 'room_id' and 'id'

We also delete any rows that have negative or 0 values.

```
>>> dfa.head()
   temp  hum  out_temp  out_hum  clouds  solar_rad      uv  target
17904  21.0  44.0     11.0    88.0     88   108.8570  0.975965     0
17907  21.0  44.0     11.0    89.0     88   108.8570  0.975965     0
17909  21.0  44.0     11.0    90.0     85    47.7171  0.930396     0
17911  21.0  44.0     11.0    89.0     85    47.7171  0.930396     0
17913  21.0  44.0     11.0    90.0     85    47.7171  0.930396     0
```

Fig. 3.50 Dataframe before pre-processing of the data.

Now we do a very interesting set of operations. The first thing we do is column operations in a loop, starting with temperature, then humidity, ...

The point of these operations is to normalize and make it easier for the Neural Network to spot changes or important information and help it learn. Our first operation is going to be transforming the values into percentage change. This means that instead of having 21.0C degrees as temperature we will have a percentage which will indicate how much it has changed from the previous reading. That means that if row 1 has 10C as temperature and row 2 has 11C the value of the second row will be 10% as it's how much it has changed. If two values are the same, then the value is 0.

Then we will go through a process of scaling. Any value that can't be represented as percent change will need to be scaled from 0 to 1 to prevent any sort of extremity bias as described before.

```
> for col in dfa.columns:
    dfa[col] = dfa[col].pct_change()
    dfa.dropna(inplace=True)
    dfa[col] = preprocessing.scale(dfa[col].values)

> dfa.dropna(inplace=True)
```

Once that is complete, we will have the following database, ready to be split into sequences, where no value is over 1 and we have pre-processed the data correctly.

```
>>> dfa.head()
      temp      hum  out_temp  ...  solar_rad      uv  target
17913 -0.008811 -0.006377 -0.024966  ...  -0.239352  0.183333      0
17915 -0.008811 -0.006377 -0.024966  ...  -0.239352  0.183333      0
17917 -0.008811 -0.006377 -0.024966  ...  -0.239352  0.183333      0
17919 -0.008811 -0.006377 -0.024966  ...  -0.239352  0.183333      0
17921 -0.008811 -0.006377 -0.024966  ...  -0.239352  0.183333      0

[5 rows x 8 columns]
```

Fig. 3.51 Dataframe after pre-processing.

To build the actual sequences we need to use the deque list we had described before. The length of this queue will be the sequence length we have set before (18)

```
sequential_data = []
prev_days = deque(maxlen=SEQ_LEN)
```

We then use a loop to go through all the values in the database where we append all values to the list until we reach 18 rows and then grab the `i[-1]` which is the last column ('target') and add it to our labels list. We then divide the sequences into two categories. Those that end with the temperature going up and those that end with the temperature staying or going low.

```
for i in dfa.values:
    prev_days.append([n for n in i[:-1]])
    if len(prev_days) == SEQ_LEN:
        sequential_data.append([np.array(prev_days), i[-1]])
random.shuffle(sequential_data)

hot = []
cold = []

for seq, target in sequential_data:
    if target == 0:
        cold.append([seq, target])
    elif target == 1:
        hot.append([seq, target])
```

We need to do this split because as discussed before we need to balance the amount of sequences that end in hot and cold to avoid any sort of bias the Neural Network might develop. We then get the lowest amount of sequences (usually always 'temperature goes high') and cut the amount of 'temperature goes low' sequences to that.

```

lower = min(len(hot), len(cold))
hot = hot[:lower]
cold = cold[:lower]

sequential_data = hot + cold
random.shuffle(sequential_data)

X = []
y = []

for seq, target in sequential_data:
    X.append(seq)
    y.append(target)

```

With that, our pre-processing is complete, and we shuffle the sequences to make sure that the learning is as efficient as possible. We then return both X and Y as NumPy arrays and go back to our secondary thread.

After pre-processing the data, we need to build our Neural Network using a sequential model. If we look at how our Neural Network architecture that we set on previously,

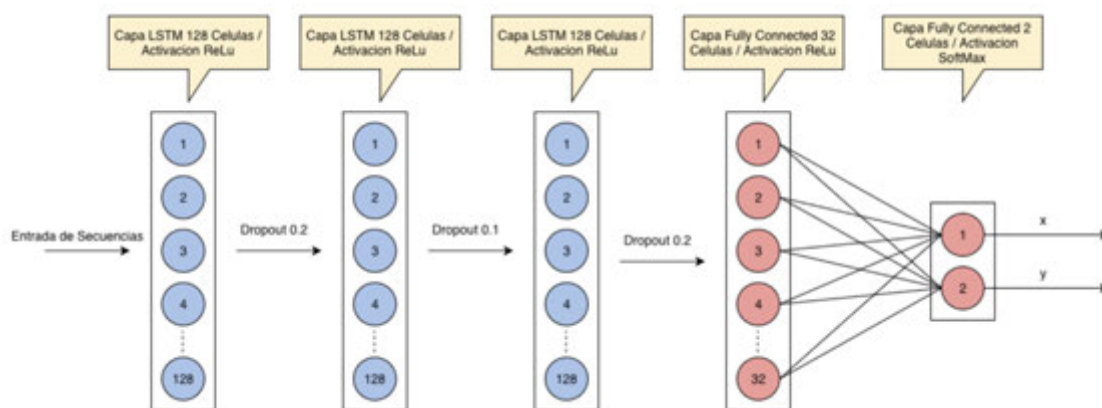


Fig. 3.52 Neural Network Layout and Architecture

We can start coding this architecture into the model. Let's start declaring the model and the first layer.

```

model = Sequential()
model.add(LSTM(128, input_shape=(train_x.shape[1:]),
activation='relu', return_sequences=True))
model.add(Dropout(0.2))
model.add(BatchNormalization())

```

As we can see, we set the model as sequential, and then declare a Neural Network layer with 128 LSTM cells, set the input shape and set our activation layer as ReLU. We then add a Dropout layer of 20% and a Batch Normalization layer.

We then model the rest of the Neural Network using the same type of commands and changing the cells to Fully Connected layers (Dense in TensorFlow nomenclature) and changing the activation function for the output layer.

```

model.add(LSTM(128, activation='relu', return_sequences=True))
model.add(Dropout(0.1))
model.add(BatchNormalization())

model.add(LSTM(128, activation='relu'))
model.add(Dropout(0.2))
model.add(BatchNormalization())

model.add(Dense(32, activation='relu'))
model.add(Dropout(0.2))

model.add(Dense(2, activation='softmax'))

```

After declaring the Neural Network completely, preparing the data and getting everything read we can finally compile the model.

```

model.compile(
    loss='sparse_categorical_crossentropy',
    optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3, ),
    metrics=['accuracy']
)

```

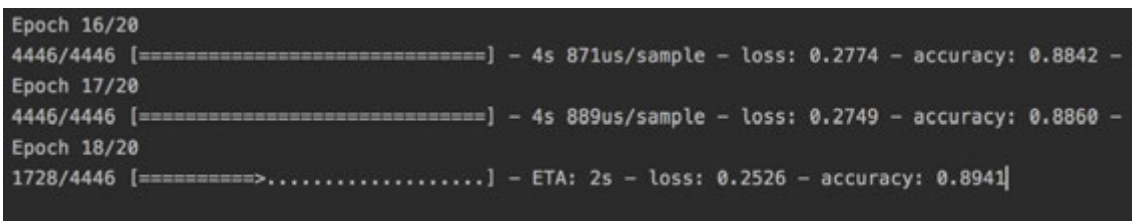
As discussed, before we are going to use Sparse Categorical Cross Entropy as our metric top optimize, we are going to display accuracy in the trace, and we are going to use the Adam optimizer at a learning rate of 0.001

```

# Train model
history = model.fit(
    train_x, train_y,
    batch_size=BATCH_SIZE,
    epochs=EPOCHS,
    validation_freq=1,
    validation_data=(validation_x, validation_y)
)

```

We then train the model and wait. Depending on our computers performance this will take about 5 minutes. At the time of writing we have 4500 sequences to train on and 1000 sequences to validate on.



```

Epoch 16/20
4446/4446 [=====] - 4s 871us/sample - loss: 0.2774 - accuracy: 0.8842 -
Epoch 17/20
4446/4446 [=====] - 4s 889us/sample - loss: 0.2749 - accuracy: 0.8860 -
Epoch 18/20
1728/4446 [=====>.....] - ETA: 2s - loss: 0.2526 - accuracy: 0.8941]

```

Fig. 3.53 Neural Network training

We can see that the network is learning correctly, and our initial accuracy and loss numbers are looking good.

To finish this thread, we need to save our model to the disk and clear the session to fix any memory leaks. We also run the garbage collector to make sure there aren't any variables that are kept alive when the thread quits.

```
model.save('./my_model.h5')
K.clear_session()
del model
gc.collect()
```

3.11.3 Secondary thread: getPredictions

The getPredictions thread is actually a quite simple set of instructions despite the flowchart being complicated. The most important aspect of this thread is gathering the current and past temperature data from the MySQL server and doing the same pre-processing as the previous thread did. Then running the single sequence through the trained Neural Network and getting the prediction.

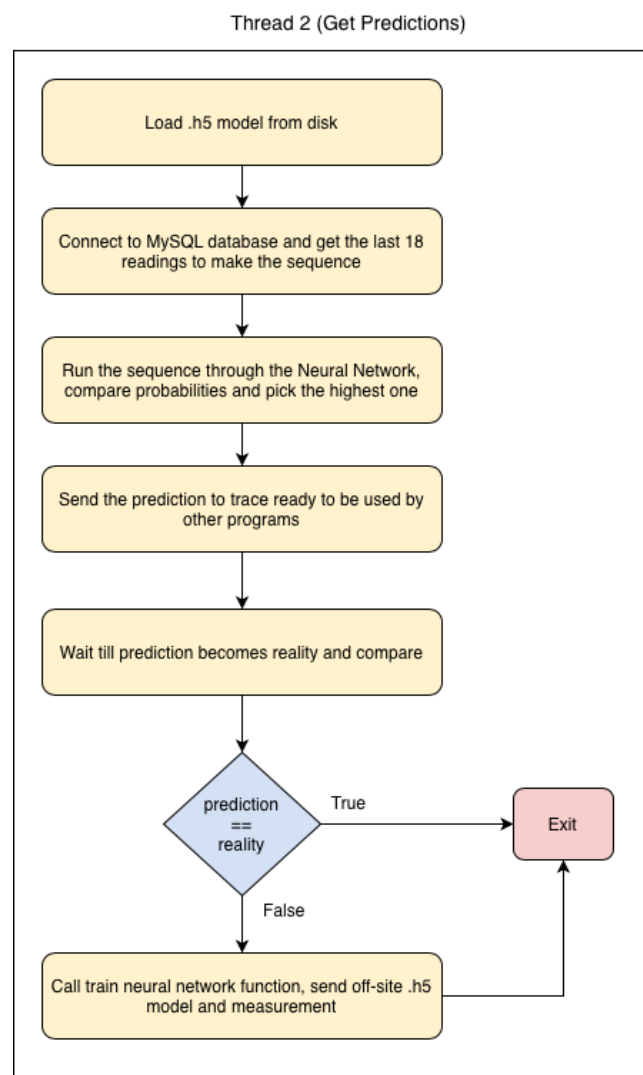


Fig. 3.54 'getPredictions' Thread Flowchart.

First, we need to load the .h5 model from the disk. Since we saved it to a specific path in the other thread, we first need to make sure that the file isn't being overwritten as its loaded. For this we use a TensorFlow function that takes concurrency into account and loads the model into a variable.

```
model = tf.keras.models.load_model("./my_model.h5")  
model._make_predict_function()
```

We then connect to our database and query the last 18 rows of our sensor measurements

```
> SELECT * FROM `procReadings` ORDER BY id DESC LIMIT ' + str(SEQ_LEN)
```

Since we declared a pre-processing function before, we can simply access it and pass this dataframe we collected from the MySQL database and get the sequence.

```
try:  
    predictions = model.predict(try_x)  
    if predictions[0][0] > predictions[0][1]:  
        print "[" + str(datetime.datetime.now()) + "] Temperature is  
staying or going low")  
  
    elif predictions[0][0] <= predictions[0][1]:  
        print "[" + str(datetime.datetime.now()) + "] Temperature is  
going up in room")  
except:  
    print "[" + str(datetime.datetime.now()) + "] ERROR: Failed  
Getting Predictions")
```

We can finally get a prediction out of our Neural Network and use it for any sort of subsequent tasks. Since we don't have a real off-site server that agglomerates our data, we can't implement the function to send the Neural Network. But if needed, the way to do so would be to implement a socket and sending the .h5 file that we created earlier through TCP.

With that set, we have successfully built a fully working Recurrent Neural Network that interacts with our Kubernetes cluster and also uses our own created data from our own sensors.

To deploy the Neural Network into the cluster we simply follow the same procedure as when we deployed our weatherPred service but changing the Dockerfile to include our modules.

4 PROPOSAL

4.1 Methodology and phases

During this section we will introduce the work methodology we are using, the user requirements we have set, the user stories that we will be basing those requirements on and the entire development process in terms of time and resources used.

During this entire process we have been using a mixture of waterfall process and mixing some other tools from other methodologies such as lean or agile. The waterfall process is a sequential methodology where a phase needs to be finished before the next one is started.

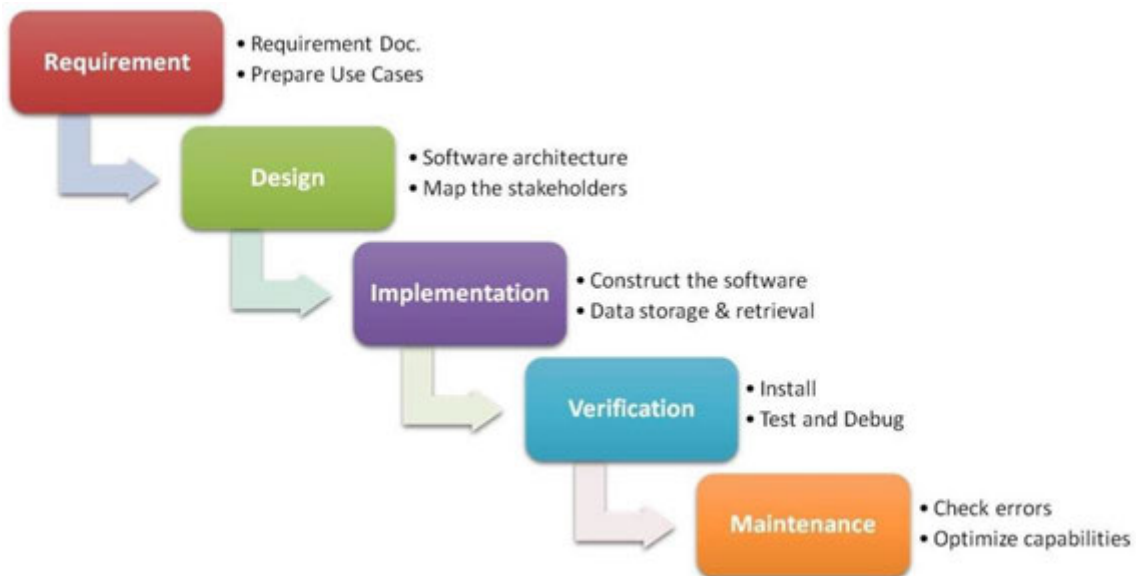


Fig. 4.1 Waterfall Methodology Phases [31].

Describing the phases in detail:

- **Requirements:** This phase is the start of the project; we first prepare the requirements (both functional and non-functional) and the use cases to try and extract information to build more requirements to include in the document.
- **Design:** During this phase we start thinking about what tools, technologies and methodologies we're going to use during the project. We also think about what the target audiences and the project stakeholders.
- **Implementation:** Usually the longest phase. We start by implementing those requirements we have set in previous phases using those technologies described in the implementation.
- **Verification:** During this phase we test all implemented in the previous phase. We can either use unit tests or manual testing. We have to take into account that some of the requirements have specific criteria that need to be met in this phase in order to be considered 'verified'.

- Maintenance: Especially important in our case, since were deploying this software for a 24/7 runtime which introduces special cases to debug and maintain.

4.2 Use case analysis

In this section we will first look at an example use case chart and then discuss all the possible use cases that we have found in our project. This is important because we will base our requirements from these use cases.

TABLE 4.1 EXAMPLE USE CASE.

Use Case

<i>ID</i>	A unique Identifier (UCxxx)
<i>Name</i>	A descriptive name
<i>Actors</i>	Entities involved in this UC (User, System, ...)
<i>Description</i>	Text description of the UC
<i>Preconditions</i>	Conditions that need to be met before the UC is executed
<i>Effects</i>	What happens after the UC
<i>Scenario</i>	Brief description of what happens in the UC

Now we can start modelling our use cases.

TABLE 4.2 USE CASE 1.

Use Case

<i>ID</i>	UC1
<i>Name</i>	Checking Data
<i>Actors</i>	User
<i>Description</i>	The user can go into the dashboard to check the data
<i>Preconditions</i>	System is up, web server loaded, database is active
<i>Effects</i>	System loads webpage and it is shown on user's screen
<i>Scenario</i>	-User types in dashboard URL -Webpage queries sql database -Webpage processes sql data and shows it in screen

TABLE 4.3 USE CASE 2.

Use Case

<i>ID</i>	UC2
<i>Name</i>	Sensors reading data
<i>Actors</i>	Sensors (System)
<i>Description</i>	The sensors can read the DHT module and send that information to the cluster via http
<i>Preconditions</i>	ESP32 is working, connections are set and there is an active connection to cluster
<i>Effects</i>	Temperature is sent to the database
<i>Scenario</i>	<ul style="list-style-type: none"> -ESP32 queries the DHT sensor -Saves information in a variable -Sends HTTP command to server -Receives OK from server

TABLE 4.4 USE CASE 3.

Use Case

<i>ID</i>	UC3
<i>Name</i>	Kubernetes System
<i>Actors</i>	System
<i>Description</i>	The Kubernetes system correctly is allocating all of the deployed applications and managing all the incoming services and connections.
<i>Preconditions</i>	Kubernetes is set up, applications are deployed, services are set
<i>Effects</i>	All applications are accessible via the internal cluster network or outside the cluster if they have been given a NodePort service.
<i>Scenario</i>	The Kubernetes system allocates all pods into their corresponding nodes.

TABLE 4.5 USE CASE 4.

Use Case

<i>ID</i>	UC4
<i>Name</i>	Neural Network
<i>Actors</i>	System
<i>Description</i>	The Neural Network inside the cluster is able to correctly predict temperature changes.
<i>Preconditions</i>	Kubernetes is up and running, database is ok, sensors and transmitting data to the cluster, Neural Network is trained.
<i>Effects</i>	The Neural Network produces predictions on temperature changes correctly.
<i>Scenario</i>	-Neural Network grabs database -Neural Network trains -Neural Network produces predictions

4.3 Requirements

During the course of this section we will go into what requirements were set for the optimal completion of this project. We will also look at the requirements depending on if they are functional or non-functional to the whole system.

A sample requirement can be defined like the following one:

TABLE 4.6 EXAMPLE REQUIREMENT TABLE.

User or System Requirement

<i>ID</i>	URxx , FRxx or NFRxx
<i>Name</i>	A descriptive name
<i>Source</i>	Where the requirement is coming from.
<i>Description</i>	Description of the requirement
<i>Significance</i>	Determines the importance of the requirement (Essential, Desirable, Optional)
<i>Priority</i>	It determines the importance in the development process (High, Medium, Low)
<i>Verifiability</i>	Determines to what degree it is possible to verify that the requirement is working as intended (High, Medium, Low)

We will now go into describing the actual requirements in our project.

4.3.1 Functional requirements

In this section we will present those requirements that define what the system should accomplish.

TABLE 4.7 FUNCTIONAL REQUIREMENT FR1.

<i>System Requirement</i>	
<i>ID</i>	FR1
<i>Name</i>	Sensors must send the temperature and humidity to the cluster
<i>Source</i>	UC2
<i>Description</i>	The ESPx boards must collect the temperature and humidity from the DHTx sensor and send it to the cluster via HTTP
<i>Significance</i>	Essential
<i>Priority</i>	High
<i>Verifiability</i>	High (We can check if the data is in the database)

TABLE 4.8 FUNCTIONAL REQUIREMENT FR2.

<i>System Requirement</i>	
<i>ID</i>	FR2
<i>Name</i>	The database must be accessible in the cluster
<i>Source</i>	UC2, UC3, UC4
<i>Description</i>	The MySQL database must be accessible in the cluster to other applications like the Neural Network, the Weather Predict service and the Apache Web Server
<i>Significance</i>	Essential
<i>Priority</i>	High
<i>Verifiability</i>	High (We can trace its accessibility)

TABLE 4.9 FUNCTIONAL REQUIREMENT FR3.

System Requirement

<i>ID</i>	FR3
<i>Name</i>	The Neural Network must make predictions based on the database
<i>Source</i>	UC4
<i>Description</i>	The Neural Network we built must take the sensor data stored in the database, train itself with it and then make predictions over the current temperature and weather data
<i>Significance</i>	Desirable
<i>Priority</i>	Medium
<i>Verifiability</i>	High

TABLE 4.10 FUNCTIONAL REQUIREMENT FR4.

System Requirement

<i>ID</i>	FR4
<i>Name</i>	The weather prediction service must fill the missing database entries
<i>Source</i>	UC2, UC3, UC4
<i>Description</i>	The WeatherPred program needs to pull weather data from an API, update the records with extra weather information and save it in the database.
<i>Significance</i>	Desirable
<i>Priority</i>	Medium
<i>Verifiability</i>	High (We can see it in the database)

TABLE 4.11 FUNCTIONAL REQUIREMENT FR5.

System Requirement

<i>ID</i>	FR5
<i>Name</i>	The Neural Network must send its configuration to the off-site agglomerate
<i>Source</i>	System
<i>Description</i>	To perform our Edge Computing feature, we must have two replicas of the same Neural Network running on and off-site. When the on-site one trains, it must send its configuration to the off-site one
<i>Significance</i>	Optional
<i>Priority</i>	Medium
<i>Verifiability</i>	High (We can see if it has sent)

TABLE 4.12 FUNCTIONAL REQUIREMENT FR6.

System Requirement

<i>ID</i>	FR6
<i>Name</i>	The apache web server must process the ESP readings
<i>Source</i>	UC2
<i>Description</i>	The apache web server must grab the incoming HTTP connections from the ESP boards, get the data and insert it into the database in its corresponding table.
<i>Significance</i>	Essential
<i>Priority</i>	High
<i>Verifiability</i>	High (We can trace its logs and, in the database,)

4.3.2 Non-functional requirements

Non-functional requirements represent those requirements that although they do not represent functionality, they present how that functionality should occur.

TABLE 4.13 NON-FUNCTIONAL REQUIREMENT NFR1.

<i>System Requirement</i>	
<i>ID</i>	NFR1
<i>Name</i>	Use of Python language
<i>Source</i>	Client
<i>Description</i>	In order to make things accessible we will be programming the ESP boards, the weather prediction service and the Neural Network in python
<i>Significance</i>	Essential
<i>Priority</i>	High
<i>Verifiability</i>	High

TABLE 4.14 NON-FUNCTIONAL REQUIREMENT NFR2.

<i>System Requirement</i>	
<i>ID</i>	NFR2
<i>Name</i>	Use of PHP language for Web Server
<i>Source</i>	Client
<i>Description</i>	The PHP language and scripts will be used to do server-client functions and to parse the information coming from the server
<i>Significance</i>	Essential
<i>Priority</i>	High
<i>Verifiability</i>	High

TABLE 4.15 NON-FUNCTIONAL REQUIREMENT NFR3.

<i>System Requirement</i>	
<i>ID</i>	NFR3
<i>Name</i>	Usability Standards
<i>Source</i>	Client
<i>Description</i>	We must comply with Nielsen's usability guidelines to make sure our dashboard is a good user experience
<i>Significance</i>	Desirable
<i>Priority</i>	Low
<i>Verifiability</i>	High

TABLE 4.16 NON-FUNCTIONAL REQUIREMENT NFR4.

<i>System Requirement</i>	
<i>ID</i>	NFR4
<i>Name</i>	Use of OS level programming
<i>Source</i>	Client
<i>Description</i>	In order to increase efficiency in computing power and energy, we must do OS level programming like threading and scheduling of tasks in our python programs.
<i>Significance</i>	Desirable
<i>Priority</i>	Medium
<i>Verifiability</i>	High

TABLE 4.17 NON-FUNCTIONAL REQUIREMENT NFR5.

<i>System Requirement</i>	
<i>ID</i>	NFR5
<i>Name</i>	Use of local storage in Kubernetes
<i>Source</i>	Client
<i>Description</i>	In order to store our database and our web pages/ scripts once the cluster reboots, we need to set a local storage class in Kubernetes
<i>Significance</i>	Essential
<i>Priority</i>	Medium
<i>Verifiability</i>	High

TABLE 4.18 NON-FUNCTIONAL REQUIREMENT NFR6.

System Requirement

<i>ID</i>	NFR6
<i>Name</i>	Use of established modules in the Neural Network
<i>Source</i>	Client
<i>Description</i>	In order to make a competitive Neural Network in python we must use modules like TensorFlow and SkLearn to take advantage of pre-made optimizers and solutions to get good performance out of our network.
<i>Significance</i>	Essential
<i>Priority</i>	High
<i>Verifiability</i>	High

TABLE 4.19 NON-FUNCTIONAL REQUIREMENT NFR7.

System Requirement

<i>ID</i>	NFR7
<i>Name</i>	Latency Limit
<i>Source</i>	Client
<i>Description</i>	In order to keep the performance of the cluster at the maximum level we have to keep the latency between the sensors and the cluster and the weather API server and the cluster at less than 200ms
<i>Significance</i>	Desirable
<i>Priority</i>	Low
<i>Verifiability</i>	High (We can use the ping tool/ Wireshark)

4.3.3 User requirements

We will now go into what user requirements we must follow when building our application. Since we only have one part of the project that the end user (consumer) is supposed to see, we just have two user requirements.

TABLE 4.20 USER REQUIREMENT UR1.

<i>User Requirement</i>	
<i>ID</i>	UR1
<i>Name</i>	Current Data Analysis
<i>Source</i>	Client
<i>Description</i>	A user must be able to see what the current data (local and coming from the weather API) is through a dashboard built into the web server.
<i>Significance</i>	Essential
<i>Priority</i>	Medium
<i>Verifiability</i>	High

TABLE 4.21 USER REQUIREMENT UR2.

<i>User Requirement</i>	
<i>ID</i>	UR2
<i>Name</i>	Past Data Analysis
<i>Source</i>	Client
<i>Description</i>	A user must be able to see what the past data stored in the database was, including a graph to make analysis of past data easier.
<i>Significance</i>	Desirable
<i>Priority</i>	Medium
<i>Verifiability</i>	High

4.3.4 Project planning

We strived to finish this project in under 6 months. Initial planning and setting up of the cluster took place around October 2019. We started off with a lot of issues since this is a very experimental software but quickly got the first deployments running in Kubernetes.

The planification of this project was divided in these phases in chronological order:

1. Construction of the Kubernetes Cluster.
 - a. Setting up of the Raspberry Boards.
 - b. Initial Kubernetes Setup.
2. Construction of the ESPx Sensors.
 - a. Attaching the DHT22 Temp/Hum Sensors.
 - b. Flashing the MicroPython software.
 - c. Flashing our Python application into the board.
3. Deploying Applications.
 - a. Deploying the MySQL database into the cluster.
 - b. Deploying the Apache Web Server into the cluster.
4. Building Custom Applications.
 - a. Building the 'weatherPred' application.
 - i. Finding the API.
 - ii. Connecting to the database inside the Kubernetes cluster.
 - iii. Building the application as a docker container.
 - iv. Deploying the container into the Kubernetes cluster.
 - b. Building the Neural Network.
 - i. Researching the Layout / Optimizers / Parameters.
 - ii. Building the Network using TensorFlow, Keras, ...
 - iii. Pre-Processing data for the network.
 - iv. Training and Predicting correctly.
 - v. Deploying as container and into cluster.
5. System Evaluation.
 - a. Requirements testing (Case Studies).
 - b. Performance testing.
 - i. Neural Network Performance.
 - ii. Cluster Performance.
 - iii. Network / Sensor Performance.
6. Project Report.

If we plot this progression on a Gantt chart:

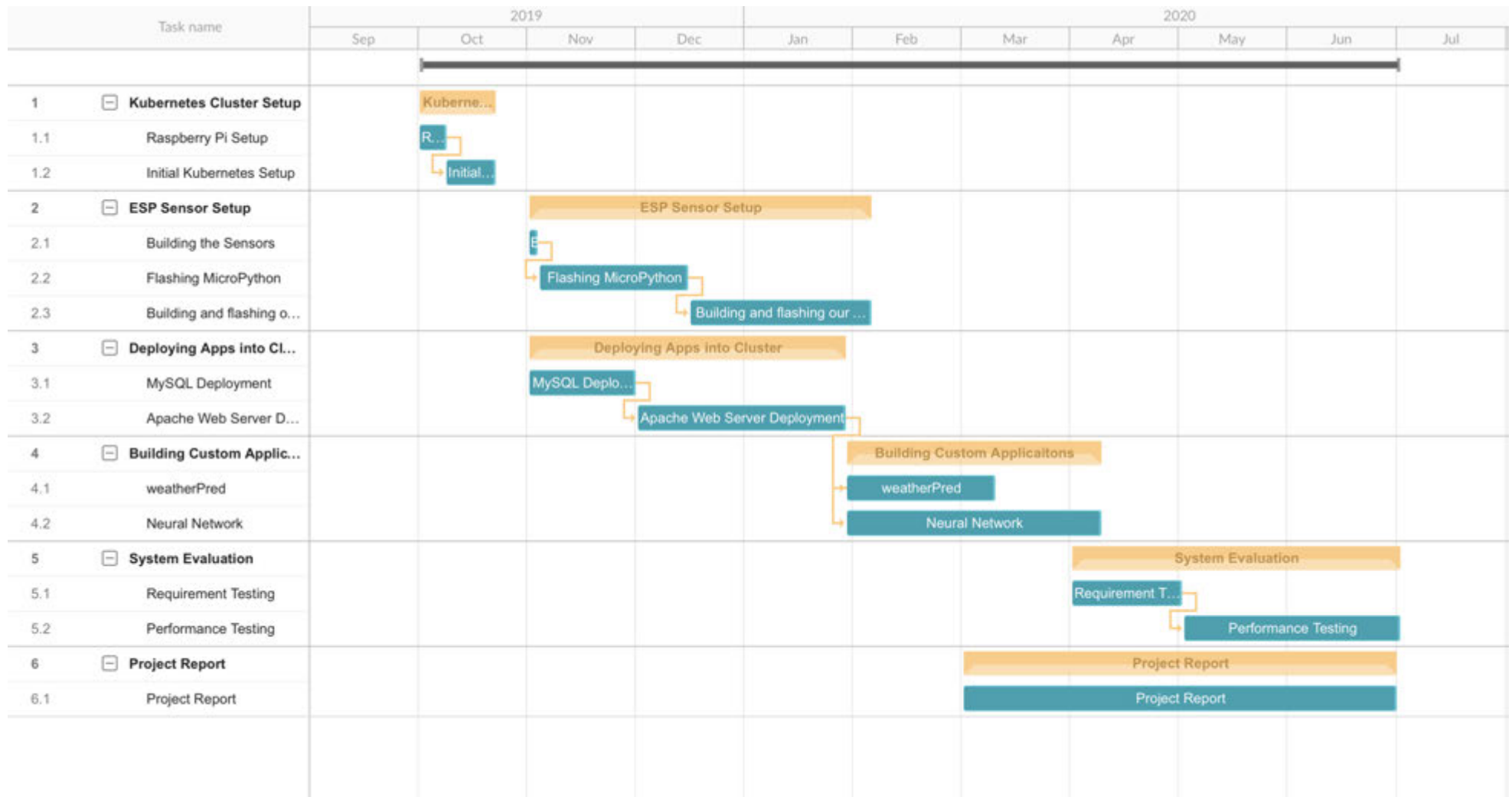


Fig. 4.2 Gantt Chart detailing the entire development process of the project.

As we can see the development process of this project took a long time. This is mainly because we needed to allow a long period of time for data to be recorded into the database before we could launch any type of Neural Network.

The actual deployment of the Kubernetes system was over in the span of two months, even with some debugging and troubleshooting that we needed to address. The sensors took a long time to get perfect, (over two months) because we would have long time stability issues, where a sensor would crash after a week or so running, and we couldn't troubleshoot it since there's no way to debug an ESP board. We finally concluded it was a memory leak and implemented a garbage collection system onboard.

Operationally, the system has been fully working since May and we have had a flawless two months May and June with no bugs or problems in the entire system (Kubernetes, sensors, Neural Network)

4.4 Socio-economic environment

4.4.1 Economic impact

In order to understand what kind of economic impact our project may have on society we simply need to look at what the growth the companies that are using similar technologies and tools are having. Tech companies that have invested in the use of cluster technology such as Amazon or Google are experiencing incredible growth in both economic and company growth.

Amazon has gone from a resale first company to having a large chunk of the ‘Cloud Computing’ market by investing and offering the same tools we have used in this project. Being able to advertise that our project not only runs those technologies but that we have built other novel features into them (Neural Network integration, use of IoT devices) only increases the economic impact that this project will have.

In fact, over the past six months that this project has been in development, we’ve had numerous high-level executives show interest in the project, saying that the timing of the project is right in line to showcase AI and Cloud Computing features.

Since the initial investment is low, we only need to set up a hardware cluster which can be as cheap as our Raspberry Pi’s our project becomes profitable extremely quickly, with the AI features and the IoT sensors costing no money in upkeep.

4.4.2 Social and environmental impact

The social impact might not be apparent at first, but these technologies ultimately allow the users to not only regain ownership of their data (since the data is stored locally not in a server where it might be vulnerable) but it also allows users to take advantage of whatever these technologies might bring.

In the immediate future and relating to this project we could further implement the temperature prediction to control a smart thermostat and control the heating in the user’s house in a more efficient and more economical manner that would also be beneficial to the environment.

We could argue against the environmental problems that running and training a Neural Network could have as it is definitely a source of power consumption and heat exhaustion. We currently believe that the benefits outweigh the problems with these AI features, but we recognize that strides need to be made to further make running a Neural Network a more environmentally friendly task.

4.5 Regulatory framework

We will now go into detailing what laws and regulations would apply in this project if it were brought to market. The first and most obvious problem we would face is the collection of personal data in our database. Since we are in the European Union and the project would be deployed into this space, we need to adhere to the EU's GDPR laws (General Data Protection Regulation).

“The General Data Protection Regulation (GDPR) is the toughest privacy and security law in the world. Though it was drafted and passed by the European Union (EU), it imposes obligations onto organizations anywhere, so long as they target or collect data related to people in the EU. The regulation was put into effect on May 25, 2018. The GDPR will levy harsh fines against those who violate its privacy and security standards, with penalties reaching into the tens of millions of euros.” [32]

According to [32] the GDPR principles are:

1. **Lawfulness, fairness and transparency** — *Processing must be lawful, fair, and transparent to the data subject.*
2. **Purpose limitation** — *You must process data for the legitimate purposes specified explicitly to the data subject when you collected it.*
3. **Data minimization** — *You should collect and process only as much data as absolutely necessary for the purposes specified.*
4. **Accuracy** — *You must keep personal data accurate and up to date.*
5. **Storage limitation** — *You may only store personally identifying data for as long as necessary for the specified purpose.*
6. **Integrity and confidentiality** — *Processing must be done in such a way as to ensure appropriate security, integrity, and confidentiality (e.g. by using encryption).*
7. **Accountability** — *The data controller is responsible for being able to demonstrate GDPR compliance with all of these principles.*

We will now go into each of the principles and explain if our project would be compliant or not with each of them:

1. **Lawfulness, fairness and transparency** — We do not store anything that the user is not aware of thus complying with this point.
2. **Purpose limitation** — The only data we collect from the user is the temperature and humidity from their homes/ business and it is only used for the Neural Network and to be displayed in their own private Dashboard. We comply with this point.
3. **Data minimization** — Again, we only collect the essential and we comply with this point.
4. **Accuracy** — There is no personal data that identifies any specific user, like a user ID or username we only collect raw weather data.
5. **Storage limitation** — We currently do not comply with this point as we keep data for an indefinite amount of time, but this could easily be solved by auto-deleting the database every x amount of months/years.

6. **Integrity and confidentiality** — We use encryption everywhere we have a chance of data leaking and we employ security measures such as the use of secrets in Kubernetes and HTTPS when the cluster connects to the internet. We comply with this point.
7. **Accountability** — We comply with this point as we are accountable of our security decisions.

With the GDPR we also need to take into account people's privacy rights, these are: [32]

1. *The right to be informed*
2. *The right of access*
3. *The right to rectification*
4. *The right to erasure*
5. *The right to restrict processing*
6. *The right to data portability*
7. *The right to object*
8. *Rights in relation to automated decision making and profiling.*

We currently do not comply with any of these as we don't have any sort of information to the user to what data is being stored.

We also don't employ a way for the user to delete their data, but all of these could be changed before business deployment as they are simple to implement in our database with SQL queries and automated modules available in our web server.

Apart from the GDPR we don't need to comply with any other laws per se, but we need to conform to the PEP8 Python Programming Standard [33] which we are fully compliant of.

The PEP8 standard [33] stipulates over the following aspects of Python programming:

- *Formatting styles and guidelines*
- *Comments*
- *Naming conventions (variables, functions, constants)*

4.6 Project budget

During this section we will discuss all of the costs associated with building the project both directly and indirectly.

We firstly need to remember the long development process and the length of time it took to develop a well performing Neural Network. The costs associated with the project were researched using average salaries obtained from the BOE (Boletín Oficial del Estado) [34] and GlassDoor [35]

The following table represents the cost associated with human resources and development pricing.

TABLE 4.22 HUMAN RESOURCE COSTS.

	<i>Gross Monthly Salary</i>	<i>Number of Months</i>	<i>Social Security (28%)</i>	<i>Final Cost</i>
<i>Web Developer</i>	1835.50	1.0	385.50	2220.90
<i>Cloud Developer</i>	5083.30	3.5	4981.34	22772.85
<i>Tester</i>	2592.50	2.0	1451.80	6636.80
<i>Web Designer</i>	2021.30	0.3	169.70	776.09
<i>Project Manager</i>	3916.60	6.0	6580.00	30079.60
<i>Systems Engineer</i>	2625.50	1.8	1323.20	6049.10
<i>TOTAL</i>				68535.34

We have concluded that the entire cost in human resources during this project has been of 68535.34 EUR. If we include tax which is a 21% increment that would leave us at **82927.76 EUR**

We will now detail all the costs related to the equipment used. We can further divide the costs into direct (those costs that are directly needed by the development of the project) and indirect costs (those that are needed for the project but aren't directly a cost associated with the project)

TABLE 4.23 DIRECT PROJECT COSTS.

<i>Item</i>	<i>Base Price (Including Tax)</i>	<i>Number of Units</i>	<i>Total Cost</i>
<i>Raspberry Pi 4 4GB Version</i>	61.99	2	123.98
<i>32GB MicroSD Card</i>	5.94	2	11.88
<i>USB Type C Charger</i>	8.99	2	17.98
<i>5 Port 1Gbps Network Switch</i>	16.95	1	16.95
<i>Cat6 0.5m Ethernet Cable</i>	1.78	2	3.57
<i>USB Micro-B Charger</i>	7.99	4	31.96
<i>ESP32 Board</i>	9.99	3	29.97
<i>ESP8266 Board</i>	7.79	1	7.79
<i>DHT22 Sensor + DuPont Cables</i>	8.79	4	39.16
<i>Development Laptop (MacBook Pro 15")</i>	2699.00	1	2699.00
<i>TOTAL</i>			2982.24

The total hardware cost comes out to 2982.24 EUR including all necessary components to make the cluster and sensors work. Since we can recuperate the money invested into our hardware with our product we can assume a 10% amortization on all hardware components leaving our total cost of hardware being **2684.24 EUR**

We will now detail all the indirect costs related to the project. The energy cost is calculated by taking the total energy consumed by the Cluster (10W/h) and the Sensors (~1W/h)

TABLE 4.24 INDIRECT PROJECT COSTS.

<i>Item</i>	<i>Base Price (Including Tax)</i>	<i>Number of Units</i>	<i>Total Cost</i>
<i>Energy Consumption (Per KW/h)</i>	0.08591	30240 W/h (For 6 Months)	2598.92
<i>On-site Meetings with Development Team</i>	23.54	6	141.24
<i>Internet Connection (Per Month)</i>	50	6	300.00
<i>TOTAL</i>			3040.16

The total indirect costs relating to the indirect hardware costs come to **3040.16 EUR**. If we sum all the costs associated with the project, we can visualize the entire costs in the following table:

TABLE 4.25 TOTAL PROJECT COSTS.

<i>Item</i>	<i>Total Cost (EUR)</i>
<i>Human Resource Costs</i>	82927.76
<i>Direct Costs</i>	2684.24
<i>Indirect Costs</i>	3040.16
<i>Weather Prediction API</i>	31.32/month
<i>TOTAL (without Weather API)</i>	88652.16

As we can see, the total cost of our project is 88652.16 EUR. We see that we needed to account for the Weather Predict API (six months of use) in case this project was brought to market raising the total to **88840.08 EUR**

4.6.1 Market research and business model

In order to create a comprehensive report about our budget and our revenue from this project we needed to elaborate on the market solutions that are available.

As discussed before we don't exactly have any other product in the market that brings the features and functionality that our project does but there are a few that need to be taken into account:

- Nest Learning Thermostat [8] : It employs a Neural Network service to make predictions on temperature like our project, but it lacks the edge server with Kubernetes or external IoT temperature sensors. Regardless, its price is 249 USD with no subscription.
- Iberdrola Smart Thermostat [36] : This service is offered by a gas and electricity provider but it comes with a Nenatmo Smart Thermostat [37] and comes with an app for mobile phones and computers, similar to the dashboard we implemented in our project. This option doesn't have any upfront costs, but it is subject to a 30-month contract for a 10.83 EUR charge per month.
- Ecobee SmartThermostat with Voice Control [38] : This is similar to the Nest Learning Thermostat, but it includes IoT sensors like our project and comes at a lower price starting at 219 USD

Having researched the market, we have decided to implement a subscription model, as it is a relatively niche solution for our market, and it prevents the user from making a large purchase price for the hardware (283 EUR)

We have set our purchase price at a one-time payment of 79 EUR and a subsequent 14.99 EUR monthly fee for the server upkeep (minimal since it's an Edge Server) and the upkeep of the central server / data warehouse.

This business plan will make sure our project is profitable long term and is able to be supported solely by our project. This set price represents a 30% of the hardware cost and the remaining cost (203 EUR) can be recouped in only 14 months ($14.99 \text{ EUR/month} * 14 \text{ months} = 209.8 \text{ EUR}$)

After the 14 months are over we need to recuperate our other costs which include human resources and other indirect costs. We can't give an exact reading for this as we don't know what number of units we will sell but looking at similar products and the market viability of our project we can safely establish an estimate of 1000 units per year. This will bring a 14.99 EUR benefit per unit after the initial 14 months, meaning a 14990 EUR benefit per month, taking a total of 6 months to cover our total budget remaining (human resources and indirect costs which total 85965 EUR).

Ultimately this plan allows us to be profitable in 20 months (most long-term business loans range from 3-10 years [39]) meaning that we will be undercutting our loan and becoming profitable quicker.

5 ANALYSIS

5.1 Hardware and software platforms used in this project

For the entirety of this project, we have kept a business and financial sense in which we think about the economic implications and scalability that a project like this entails. As such, we will be using fairly basic hardware with potent software that can scale into professional-level hardware with the same setup and procedures.

- The main computer cluster is built on Raspberry Pi's, in particular, we're using the Raspberry Pi 4B 4GB RAM Version as it gives us more flexibility and power in deployment. We are currently using 2 of these but it is scalable from 2 to N.
- We are running Linux Debian on the Raspberry Pi's, the lightweight version with no GUI, which at this current time of writing is Raspbian Buster Lite 4.19 [22]
- Our Kubernetes distribution will be a modified one to be able to run on ARM v7 that the Raspberry Pi uses. We have chosen k3s as it is the most complete solution in the market, and it is open source. We will be modifying it throughout the project to tailor the software to our needs. [40]
- The IoT Devices programmed and deployed are a combination of ESP32 chips and ESP8266 boards running bare metal Python scripts to communicate with our cluster.

5.2 Functional tests: case studies to evaluate the system

In this section we will present all of the case studies we have created to fully evaluate and make sure that the system is correctly working as intended, cross referencing these tests with the requirements we have set previously.

Each case will be represented as a table like in the following test use case:

TABLE 5.1 EXAMPLE CASE STUDY.

Case Study

<i>ID</i>	A unique Identifier (CSxxx)
<i>Objective</i>	What the test will cover
<i>Sequence of Events</i>	What the test will be comprise of, step by step
<i>Result</i>	If we are able to verify the requirement(s) or not
<i>Traceability</i>	What requirements are being verified

With the explanation underway, we will now start presenting our real case studies:

TABLE 5.2 CASE STUDY 1.

Case Study

<i>ID</i>	CS1
<i>Objective</i>	Verify the correct functioning of the sensors
<i>Sequence of Events</i>	<ul style="list-style-type: none"> -Verify the correct functioning of the python program running on the board by checking the status LEDs on the board -Verify the Apache Web Server is getting the measurements from the boards -Using Wireshark, verify the HTTP requests are being made correctly.
<i>Result</i>	Verified
<i>Traceability</i>	FR1,FR6,NFR7,NFR1

TABLE 5.3 CASE STUDY 2.

Case Study

<i>ID</i>	CS2
<i>Objective</i>	Verify the correct handling of data in the web server
<i>Sequence of Events</i>	<ul style="list-style-type: none"> -Verify the HTTP sequences are arriving -Verify the script is working as intended -Verify in the database that new rows are being created
<i>Result</i>	Verified
<i>Traceability</i>	NFR2, FR6, FR1

TABLE 5.4 CASE STUDY 3.

Case Study

<i>ID</i>	CS3
<i>Objective</i>	Verify that the dashboard is operational
<i>Sequence of Events</i>	-Type In the correct URL -Observe that the dashboard is loaded correctly -Test functionality of interactive elements like graphs
<i>Result</i>	Verified
<i>Traceability</i>	FR2,NFR2,NFR3,NFR5,UR1,UR2

TABLE 5.5 CASE STUDY 4.

Case Study

<i>ID</i>	CS4
<i>Objective</i>	Verify the correct functionality of the weather predict service
<i>Sequence of Events</i>	-Verify that the program works in our local python compiler -Verify that the program has a valid API key and can get the weather data from the API provider as a JSON file -Verify that the program updates the missing database entries in the 'procTemp' table in the MySQL database.
<i>Result</i>	Verified
<i>Traceability</i>	FR2,FR4,NFR1,NFR4,NFR5,NFR7

TABLE 5.6 CASE STUDY 5.

Case Study

<i>ID</i>	CS5
<i>Objective</i>	Verify that the Neural Network can train
<i>Sequence of Events</i>	<ul style="list-style-type: none"> -Verify that there is a connection between the database and the Neural Network -Verify the data is pre-processed correctly -Verify that the network is training correctly on that data -Verify that the trained model is being saved to disk
<i>Result</i>	Verified
<i>Traceability</i>	FR2,FR3,NFR1,NFR4,NFR5,NFR6

TABLE 5.7 CASE STUDY 6.

Case Study

<i>ID</i>	CS6
<i>Objective</i>	Verify that Kubernetes has deployed all of our applications
<i>Sequence of Events</i>	<ul style="list-style-type: none"> -Run the command to check our deployments -Run the command to check active pods -Verify our deployments are up by verifying their logs
<i>Result</i>	Verified
<i>Traceability</i>	FR2,FR3,FR4,FR5,FR6

TABLE 5.8 CASE STUDY 7.

Case Study

<i>ID</i>	CS7
<i>Objective</i>	Verify that the Neural Network is producing predictions and processing them
<i>Sequence of Events</i>	<ul style="list-style-type: none"> -Verify our model is trained -Loading the model -Loading the database -Pre-Processing data from the database -Getting Predictions -Processing Predictions
<i>Result</i>	Verified
<i>Traceability</i>	FR2,FR3,FR5,NFR1,NFR4,NFR6

5.3 Traceability matrix

We will now go into analysing our traceability matrix. This table will represent how our case studies and tests cover our original requirements (both functional and non-functional) to see if we're correctly testing all of our requirements set in previous sections.

TABLE 5.9 TRACEABILITY MATRIX.

	<i>CS1</i>	<i>CS2</i>	<i>CS3</i>	<i>CS4</i>	<i>CS5</i>	<i>CS6</i>	<i>CS7</i>
<i>FR1</i>	X	X					
<i>FR2</i>			X	X	X	X	X
<i>FR3</i>					X	X	X
<i>FR4</i>				X		X	
<i>FR5</i>						X	X
<i>FR6</i>	X	X				X	
<i>NFR1</i>	X			X	X		X
<i>NFR2</i>		X	X				
<i>NFR3</i>			X				
<i>NFR4</i>				X	X		X
<i>NFR5</i>			X	X	X		
<i>NFR6</i>					X		X
<i>NFR7</i>	X			X			
<i>UR1</i>			X				
<i>UR2</i>			X				

5.4 Performance tests

5.4.1 Neural Network analysis

In order to reach the configuration and layout of the architecture we have implemented before we needed to go through an extensive amount of trial and error. Below we have some of the many tests we did to try and find the best performance out of our Neural Network.

Firstly, we tried to make changes to the actual layout of the network, including more LSTM cells, less, different number of layers and varying the fully connected cells as well. Below we have our 4 best layouts for our Neural Network:

- Network 1:
 - LSTM Layer, 128 Cells, ReLU activation function
 - Dropout 0.2
 - LSTM Layer, 128 Cells, ReLU activation function
 - Dropout 0.1
 - LSTM Layer, 128 Cells, ReLU activation function
 - Dropout 0.2
 - Fully Connected Layer, 32 Cells, ReLU activation function
 - Fully Connected Layer, 2 Cells, SoftMax Activation
- Network 2:
 - LSTM Layer, 64 Cells, ReLU activation function
 - Dropout 0.2
 - LSTM Layer, 64 Cells, ReLU activation function
 - Dropout 0.2
 - Fully Connected Layer, 64 Cells, ReLU activation function
 - Fully Connected Layer, 2 Cells, SoftMax Activation
- Network 3:
 - LSTM Layer, 128 Cells, TanH activation function
 - Dropout 0.2
 - LSTM Layer, 128 Cells, TanH activation function
 - Dropout 0.2
 - LSTM Layer, 128 Cells, TanH activation function
 - Dropout 0.2
 - Fully Connected Layer, 64 Cells, TanH activation function
 - Fully Connected Layer, 2 Cells, SoftMax Activation
- Network 4:
 - LSTM Layer, 128 Cells, ReLU activation function
 - Dropout 0.4
 - Fully Connected Layer, 64 Cells, ReLU activation function
 - Fully Connected Layer, 2 Cells, MaxOut Activation

And the results we obtained testing these different Network layouts:

TABLE 5.10 NEURAL NETWORK LAYOUT TESTS.

	<i>Validation Score (%)</i>	<i>Loss (%)</i>	<i>Time to Train (s)</i>
<i>Network 1</i>	80.6%	28.9%	190
<i>Network 2</i>	67.9%	28.7%	102
<i>Network 3</i>	75.2%	53.7%	186
<i>Network 4</i>	55.0%	72.36%	87

As expected, the network we chose to build won but it is also worth noting that it is the network that took the longest to train and that is a disadvantage to the other networks that achieved ~10% less score but took more than a minute less to train.

Another factor that we need to test is all the constants we declared when we implemented the Neural Network. The fact is that we chose 18 as our sequence size because we ran a battery of tests to understand what sequence size and what future predict interval we needed to choose. Below is a summary of that battery of tests.

TABLE 5.11 NEURAL NETWORK PARAMETER TESTS.

	<i>Sequence Length</i>	<i>Future Predict Intervals</i>	<i>Epochs</i>	<i>Validation Score (%)</i>
	12	3	10	70.6%
	15	3	15	72.4%
	18	6	20	81.2%
	21	6	20	65.0%
	27	12	30	57.4%

As we can see we tested to see if we could predict half an hour in the future and 2 hours in the future, but it seemed impractical for our use and we didn't get the performance we were aiming for.

We also tried to lengthen the sequence size, but we got diminishing returns with higher sequence length. We found a very steep drop starting with 20 or so rows, maybe owing to the fact that the temperature in four hours isn't as representative as a shorter time frame.

We also tested different optimizers and their effects on the score but found that they either slowed the training process a lot or their effect on the metrics were negligible

We still decided to use one as it improved around 5% of our score and the Adam optimizer doesn't take a performance hit like the other ones we tried. (RMSProp and Adagrad) [41]

Even if we do all of these 'validation' tests we really won't know how well our network performs until we save the prediction guesses that it makes and then compare it with what really happened to the temperature.

To test this, we waited for a day with a lot of sunlight (cloudy days would offer bad network performance) and saved what the Neural Network was predicting compared to what really ended up happening. The result is the following:

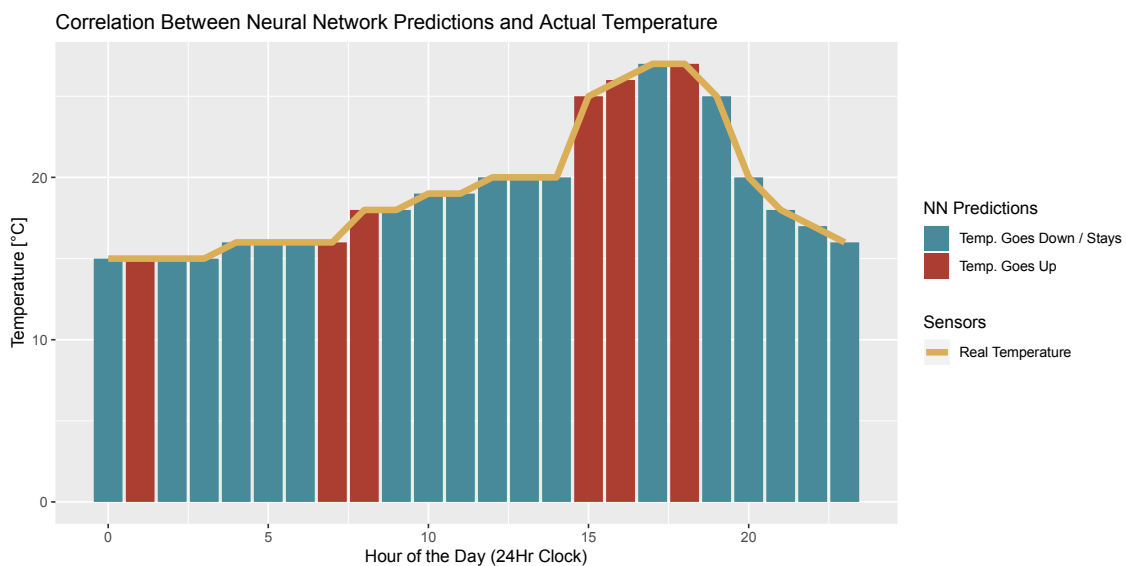


Fig. 5.1 Correlation between Neural Network predictions and actual temperature.

Basically, we can observe that whenever the Neural Network guessed that the temperature was going to go up, we can observe a jump in temperature just like predicted. We have a few outliers like the 'going up' prediction at 1 in the morning but in general we see that the Neural Network is performing correctly and guessing temperature changes correctly.

To make sure that the results weren't a fluke, we compiled more days into this test and took note of the times that the network correctly guessed the temperature was going up or going down. The results of that test were as follows:

TABLE 5.12 NEURAL NETWORK OBSERVED VS PREDICTED TEST.

	<i>Correct Guesses</i>	<i>Correct Guesses (%)</i>
<i>Day 1</i>	18/24	75.0
<i>Day 2</i>	22/24	91.6
<i>Day 3</i>	16/24	66.67
<i>Day 4</i>	21/24	87.5
<i>Day 5</i>	21/24	87.5
<i>Day 6</i>	20/24	83.3
<i>Day 7</i>	16/24	66.67
<i>Week Average</i>	19.14/24	79.75

If we compare this value to the ‘theoretical’ values, we got from TensorFlow on network accuracy and performance:

TABLE 5.13 NEURAL NETWORK OVERALL ACCURACY TEST.

<i>TensorFlow Accuracy (%)</i>	<i>Real Observed Accuracy (%)</i>	<i>Delta (%)</i>
80.6-81.2	79.75	0.85-1.45

As we can see, our network performance is quite accurate with what is observed in the real measurements pointing to the fact that we have achieved data maturity, that is, that our data that we have curated in our database is representative of our problem and the network is correctly generalizing for new cases.

Overall, we have achieved pretty respectable scores with our network. We believe that if we wanted to achieve better performance, the addition of more sensors and the inclusion of a light meter that could read lux readings inside the house would add even more accuracy to the network.

For now, the fact that we are getting around 75-80% accuracy on the network means that we only have to send our measurements 20% of the time meaning a direct reduction of the 80% of the bandwidth congestion in the link between the on-site cluster and the off-site data aggregator.

We also need to do a quick analysis on the weather data that we collected using our API weather service. Since it is a paid addition to the project, we need to make sure it is actually benefiting our Neural Network performance.

To test this, we will first train the network as usual with our standard database, but instead making real predictions, we will make ‘fake’ predictions where one of the attributes is controlled. This will show us if a specific attribute has a very high weight inside the network (meaning it is more important than another attribute)

We will first test cloud coverage in percentage.

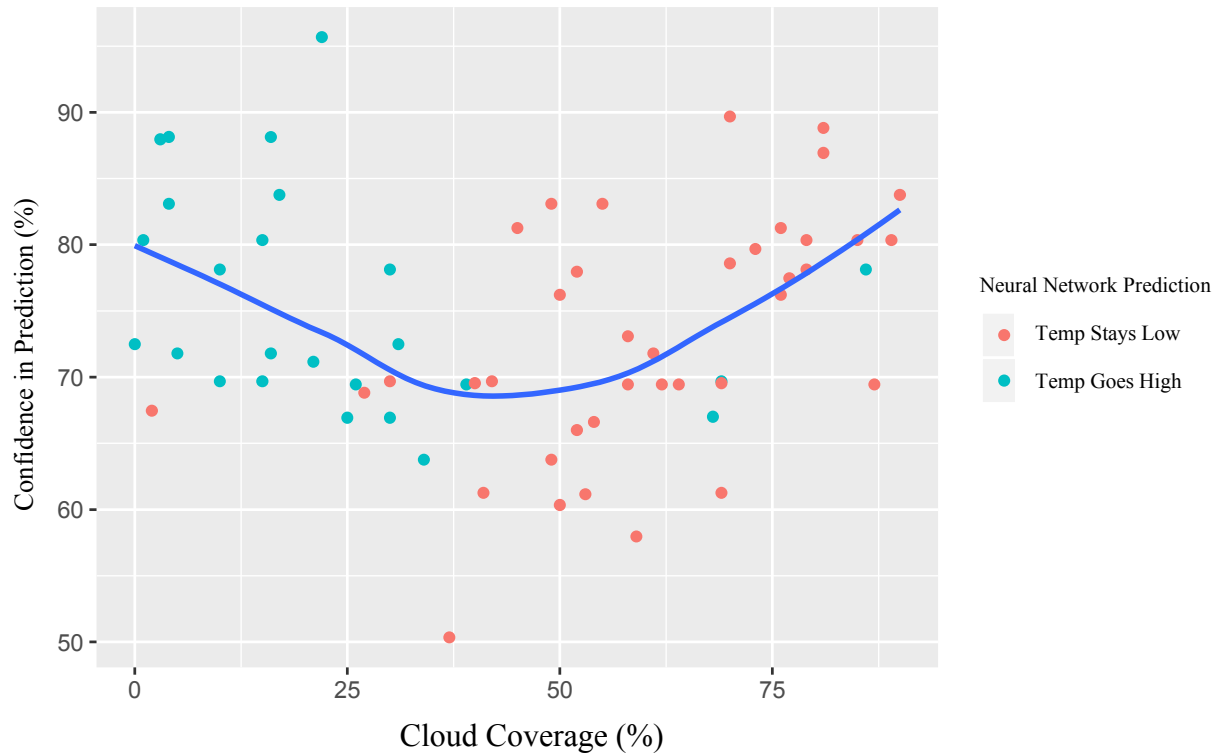


Fig. 5.2 Cloud Coverage Analysis compared to Neural Network confidence in prediction.

As we can see, the dots indicate individual guesses. Getting a ‘V’ regression is a good sign as it indicates that the more extreme the value is the more ‘confident’ the network is in its decision. We can also see that the predictions are in line with common sense, the more clouds in the sky the less the temperature is going to go up.

If we repeat this same experiment with solar radiation, we get the following:

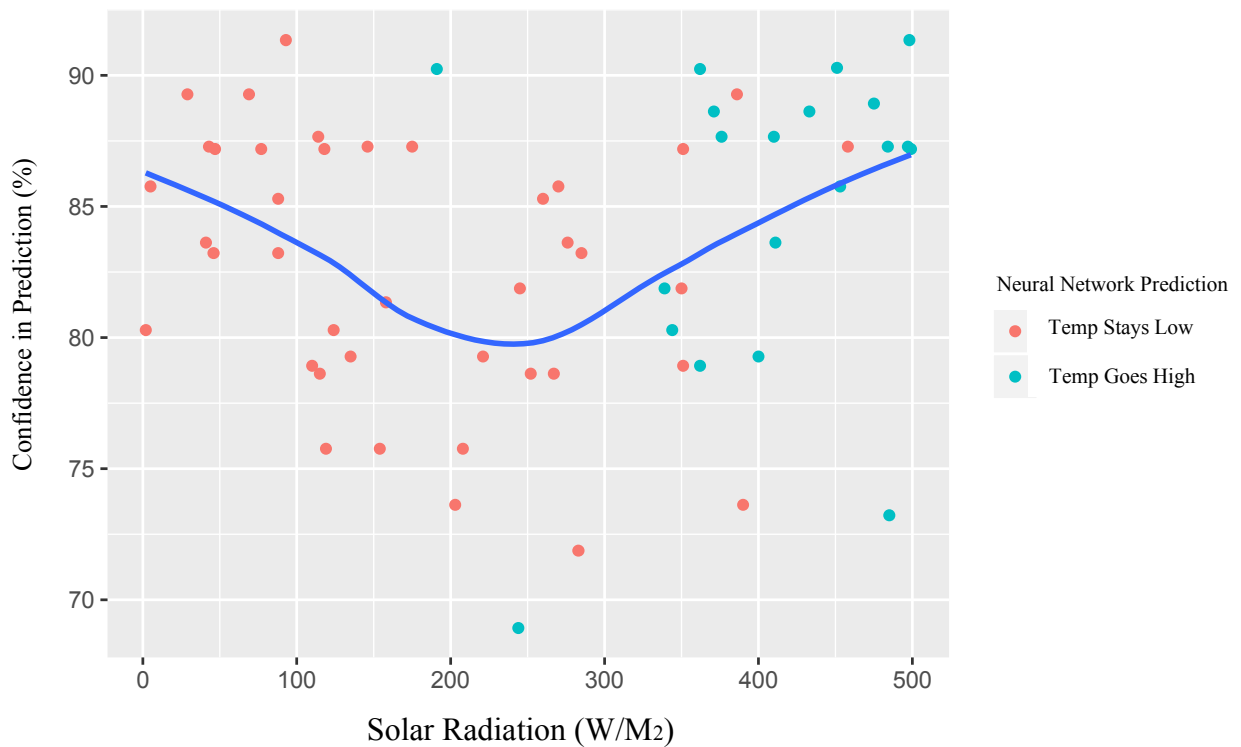


Fig. 5.3 Solar Radiation Analysis compared to Neural Network confidence in prediction.

We can see that the cloud of points is much more disperse here, which would indicate that the variable doesn't provide as much 'confidence' to the network as our cloud coverage percentage. Another interesting finding is that the network almost always says the temperature will decrease except when the solar radiation is at extremes (>400W/m2)

This further drives the point that this variable isn't as indicative as our other variables. Overall the use of an external weather provider is very useful but since it is a paid option, we need to further analyse those variables that we are introducing into the network if we were to maximize network performance.

5.4.2 Cluster analysis

The sensors began reading data and saving it to the MySQL database around December 2019 and we quickly started seeing success in the platform. Around January we had to re-deploy our cluster due to some long-term bugs and a memory leak on Kubernetes that would cause our cluster to fail after a week of uptime. We successfully fixed the leak and got back to operating that same month.

We waited around 4 months to gather a sufficient amount of data in the database before building the Neural Network. Once we ran some initial tests on the network, we found that we weren't getting the performance we were hoping for (60-70% of accuracy) and we looked into ways of improving that score. We researched how to get more information about the weather and finally ended up getting the weatherPred service we built using the API.

After successfully testing the Neural Network and its performance we are going to analyse the general function of the cluster and its parts, going over its bandwidth requirements, latencies and the messages that are emitted between each other.

We firstly need to refresh what the overall cluster architecture looks like; in the following figure we can see the details of the data sent and the latency between each node.

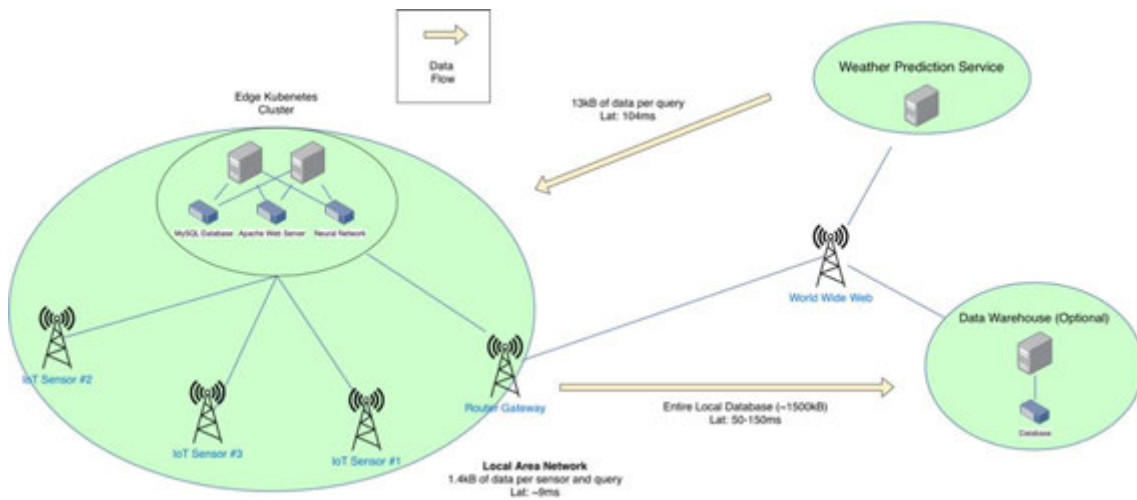


Fig. 5.4 Entire Project overview detailing latencies between cluster, sensors, weather API and data warehouse.

5.4.3 Latency and data throughput in all connections

TABLE 5.14 CLUSTER LATENCIES AND DATA PACKET SIZES.

	Latency (ms)	Total Data (kB)
<i>Weather API Server- Cluster</i>	~104	13
<i>Sensors - Cluster</i>	~9	1.4
<i>Data Aggregator/Warehouse - Cluster</i>	~50-150	~1500

To find these numbers we have done a network packet analysis using Wireshark. We will first detail the messages that go into the connection between the sensors and the cluster.

The following figure details the network packet analysis between the sensors and the cluster.

1.52	192.168.1.111	TCP	78	49700 → 32151	[SYN]	Seq=0	Win=65535	Len=0	MSS=1460	WS=64	TSval=234358795	TSecr=0	SACK_PERM=1
1.111	192.168.1.52	TCP	74	32151 → 49700	[SYN, ACK]	Seq=0	Ack=1	Win=27968	Len=0	MSS=1410	SACK_PERM=1	TSval=2631772907	TSecr=0
1.52	192.168.1.111	TCP	66	49700 → 32151	[ACK]	Seq=1	Ack=1	Win=131392	Len=0	TSval=234358804	TSecr=2631772907		
1.52	192.168.1.111	HTTP	477	GET /procTemp.php?temp=28&hum=60&station=ESPX&room_id=3 HTTP/1.1									
1.111	192.168.1.52	TCP	66	32151 → 49700	[ACK]	Seq=1	Ack=412	Win=29056	Len=0	TSval=2631772911	TSecr=234358804		
1.111	192.168.1.52	HTTP	277	HTTP/1.1 200 OK (text/html)									
1.52	192.168.1.111	TCP	66	49700 → 32151	[ACK]	Seq=412	Ack=212	Win=131200	Len=0	TSval=234358817	TSecr=2631772921		
1.111	192.168.1.52	TCP	66	32151 → 49700	[FIN, ACK]	Seq=212	Ack=412	Win=29056	Len=0	TSval=2631777927	TSecr=234358817		
1.52	192.168.1.111	TCP	66	49700 → 32151	[ACK]	Seq=412	Ack=213	Win=131200	Len=0	TSval=234363889	TSecr=2631777927		
1.52	192.168.1.111	TCP	66	49700 → 32151	[FIN, ACK]	Seq=412	Ack=213	Win=131200	Len=0	TSval=234363889	TSecr=2631777927		
1.111	192.168.1.52	TCP	66	32151 → 49700	[ACK]	Seq=213	Ack=413	Win=29056	Len=0	TSval=2631777998	TSecr=234363889		

Fig. 5.5 Wireshark analysis of the packets sent between the sensors and the cluster.

As we can see, the overall communication between the sensors and the cluster is short. This is beneficial to the overall efficiency of the system while still having important features such as TCP connection and the use of the HTTP protocol. The overall communication protocol is as follows:

- Initial TCP Connection (SYN, SYN ACK, ACK)
- HTTP Command containing the sensor measurements (temp,hum,room_id,station)
- Cluster ACK and HTTP 200 Code OK
- End of TCP Connection (ACK, FIN ACK, ACK)

Using this simple communication scheme, we're able to securely and reliably get information from our ESP sensors into the Apache Web Server inside the Kubernetes Cluster.

We also did an analysis of the latency that would occur between the data aggregator and the Kubernetes Cluster. For this we analyse the latency between different cities in the world and our external Cluster IP. We have obtained the following readings:







Location	Min. RTT	Avg. RTT	Max. RTT	Result
 Dallas, Texas, United States	125.101	125.174	125.313	Done
 Frankfurt, Hesse, Germany	37.143	37.469	38.004	Done
 London, England, United Kingdom	28.003	28.111	28.277	Done
 Madrid, Spain	10.401	10.718	11.176	Done
 Milan, Lombardia, Italy	35.954	36.116	36.313	Done
 Paris, Île-de-France, France	47.051	47.142	47.218	Done

Fig. 5.6 Latencies to the cluster from different cities in the world.

These latencies would be quite beneficial in case this project was brought to market and we needed to decide a location that would have good latency to the on-site cluster and offered affordable server hosting. We recommend any location in Europe as the reading from the United States of (~125ms) could be too large for the operations we would be intending to do.

We also need to analyse the energy usage and thermal efficiency of the Kubernetes cluster.

We need to take into account these values as they are indirect costs to the project, in our research, we have found that Kubernetes takes up very little performance and in general it offers good CPU usage. We have used ‘htop’ in Linux to measure the CPU usage of Kubernetes in both Raspberry Pi’s.

TABLE 5.15 KUBERNETES CPU LOAD TEST.

	<i>CPU Load (%)</i>	<i>Memory Usage (%)</i>
<i>Raspberry Pi #1 (Master) w/Kube</i>	26.6	12.4
<i>Raspberry Pi #2 (Worker) w/Kube</i>	6.5	2.4

In order to get significant comparison data, we will now run the same applications without Kubernetes to see what the difference between both really is.

TABLE 5.16 MANUAL DEPLOYMENT CPU LOAD TEST.

	<i>CPU Load (%)</i>	<i>Memory Usage (%)</i>
<i>Raspberry Pi #1</i>	18.9	9.7
<i>Raspberry Pi #2</i>	10.5	2.1

As we can see, rather interestingly, the second board now runs slightly higher load numbers, but the old master board runs with much less load. This is because all of the Kubernetes management that runs in the background is now eliminated. Even after this, we can safely recommend the use of Kubernetes as the load difference isn’t significant and we have to remember we are running very low power CPU’s and that the difference would be even smaller if we were using enterprise grade computers to run the cluster.

6 CONCLUSIONS

Overall during the length of this project, we have seen the many benefits of the new cloud technologies and how they could revolutionize the way we interact with applications and how we deploy new programs into the public.

The use of containers is already established in the industry, as it is apparent the many benefits it brings to developers and users. Faster times to deploy new applications as well as multi-architecture support allow for new ways to interact with software.

Kubernetes and other cluster orchestrators that work with containers are newer technologies that aim to revolutionize how we think of computing clusters. Instead of having server rooms full of huge enterprise servers (which Kubernetes also supports) it is much more flexible in its compatibility, allowing small computers such as Raspberry Pi's to become powerful cluster where we can implement the benefits of Edge Computing.

If we then add the proven benefits of Artificial Intelligence such as the use of Neural Networks to further optimize performance of our applications and we integrate technologies such as the use of Internet of Things devices like our ESP sensors, we have a very appealing complete package. In our very own project, we could observe directly how implementing our Neural Network could improve our edge computing implementation and reduce network traffic. We went from sending 100% of the readings from the sensors to only sending when the network produces the wrong prediction, meaning a ~80% reduction in network traffic compared to using no Neural Network.

This very project could be expanded with the aforementioned data aggregator and implementing a smart thermostat to use the Neural Network predictions.

In terms of completing our set objectives at the start of the project we achieved the following:

- We completed objective 1 relating to the study of Kubernetes and creating an edge computing cluster.
- We successfully achieved objectives 2 and 3 by building a whole system that integrates a Neural Network relating to those “AI Techniques” mentioned in the objective and we also implemented IoT devices in the form of ESP32 boards.
- We also completed our fourth objective by documenting and making all of our findings public as well as made a report on the performance found in the cluster and in the Neural Network

As such, we have completed all of our set objectives for this project.

To conclude, I'd like to make a point about how the beauty of this project is that the foundations of the cluster are already configured, and that we could deploy a completely new use case in a matter of minutes if we desired to thanks to the power of containers and Kubernetes.

7 BIBLIOGRAPHY

- [1] J. Pyfer, «Encyclopædia Britannica,» Encyclopædia Britannica, inc., 26 September 2018. [Online]. Available: <https://www.britannica.com/topic/Project-Mac>. [Last Access: June 2020].
- [2] AlphaStreet, «AlphaStreet,» 25 April 2019. [Online]. Available: <https://news.alphastreet.com/amazon-earnings-q1-2019>. [Last Access: June 2020].
- [3] Docker Enterprise Team, «Mirantis,» Mirantis Co., 17 January 2020. [Online]. Available: <https://blog.docker.com/2014/06/its-here-docker-1-0/>. [Last Access: June 2020].
- [4] K. D. Foote, «Dataversity,» Dataversity, 26 March 2019. [Online]. Available: <https://www.dataversity.net/a-brief-history-of-machine-learning/#>. [Last Access: June 2020].
- [5] M. B. e. al, «End to End Learning for Self-Driving Cars,» Nvidia Corporation, 2016.
- [6] K. & A. I. H. & T. I. & A. S. P.-T. Al Smadi, «Artificial Intelligence for Speech Recognition Based on Neural Networks,» Journal of Signal and Information Processing, 2015.
- [7] K. L. Lueth, «IoT Analytics,» 19 December 2014. [Online]. Available: <https://iot-analytics.com/internet-of-things-definition/>. [Last Access: June 2020].
- [8] Nest Inc., «Google Store,» Google, June 2020. [Online]. Available: https://store.google.com/us/product/nest_learning_thermostat_3rd_gen?hl=en-US. [Last Access: June 2020].
- [9] Kubernetes Team, «Kubernetes.io,» 2020. [Online]. Available: <https://kubernetes.io/docs/tutorials/kubernetes-basics/explore/explore-intro/>. [Last Access: June 2020].
- [10] NGINX Inc., «NGNINX,» 2020. [Online]. Available: <https://www.nginx.com/products/nginx-controller>. [Last Access: June 2020].
- [11] Traefik, «Containous,» Countainous, 2020. [Online]. Available: <https://containo.us/traefik/>. [Last Access: June 2020].
- [12] Ubahnverleih, «ESP32 Espressif ESP-WROOM-32 Dev Board,» Wikimedia, 2018.
- [13] S. Santos, «Maker Advisor,» 18 May 2020. [Online]. Available: <https://makeradvisor.com/esp32-vs-esp8266/>. [Last Access: June 2020].

- [14] iainandrew, «wiaCommunity,» October 2018. [Online]. Available: <https://community.wia.io/d/54-dht11-vs-dht22-what-s-the-difference>. [Last Access: June 2020].
- [15] mchobby.de, «DHT22/DHT11 + Extra Temperature & Humidity Sensor,» 2020.
- [16] Weatherbit Team, «Weatherbit.io,» 2020. [Online]. Available: <https://www.weatherbit.io>. [Last Access: June 2020].
- [17] S. Hochreiter y J. Schmidhuner, «Long Short-Term Memory,» *Neural Computation*, 1997.
- [18] G. Chevalier, Artist, *LSTM Cell*. [Art].
- [19] Peltarion, «Peltarion,» [Online]. Available: <https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/loss-functions/categorical-crossentropy>. [Last Access: June 2020].
- [20] C. H. A. C. Brian Chambers, «Medium,» Chick-fil-A Tech Blog, 26 June 2018. [Online]. Available: <https://medium.com/@cfatechblog/bare-metal-k8s-clustering-at-chick-fil-a-scale-7b0607bd3541>. [Last Access: June 2020].
- [21] M. Henzler, «Wikimedia,» *Laserlicht*, 3 July 2019. [Online]. Available: https://es.m.wikipedia.org/wiki/Archivo:Raspberry_Pi_4_Model_B_-_Side.jpg. [Last Access: June 2020].
- [22] RaspberryPi Org., «Raspberry Pi,» 27 May 2020. [Online]. Available: <https://www.raspberrypi.org/downloads/raspberry-pi-os/>. [Last Access: June 2020].
- [23] MicroPython Forum, «MicroPython Forum,» 22 June 2020. [Online]. Available: <https://forum.micropython.org>. [Last Access: June 2020].
- [24] MicroPython.org, «MicroPython.org,» September 2018. [Online]. Available: <https://docs.micropython.org/en/latest/esp32/tutorial/intro.html>. [Last Access: June 2020].
- [25] Pololu, «Pololu,» January 2019. [Online]. Available: <https://www.pololu.com/docs/0J7/all>. [Last Access: June 2020].
- [26] A. G. Frederik Ahlberg, «github.com,» 19 June 2020. [Online]. Available: <https://github.com/espressif/esptool>. [Last Access: June 2020].
- [27] Espressif Systems, «Espressif,» 2019. [Online]. Available: https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf. [Last Access: June 2020].

- [28] Randomnerdtutorials, «Randomnerdtutorials.com,» September 2018. [Online]. Available: <https://randomnerdtutorials.com/esp32-esp8266-dht11-dht22-micropython-temperature-humidity-sensor/>. [Last Access: June 2020].
- [29] LastMinuteEngineers, «LastMinuteEngineers,» December 2019. [Online]. Available: <https://lastminuteengineers.com/esp32-sleep-modes-power-consumption/>. [Last Access: June 2020].
- [30] J. Wallen, «TheNewStack,» 30 January 2020. [Online]. Available: <https://thenewstack.io/how-to-enable-docker-experimental-features-and-encrypt-your-login-credentials/>. [Last Access: June 2020].
- [31] UK Essays, «UK Essays,» November 2018. [Online]. Available: <https://www.ukessays.com/essays/computer-science/waterfall-methodology-in-software-development.php>. [Last Access: June 2020].
- [32] GDPR.EU, «GDPR.EU,» [Online]. Available: <https://gdpr.eu/what-is-gdpr/>. [Last Access: June 2020].
- [33] G. v. Rossum, «python.org,» 5 July 2001. [Online]. Available: <https://www.python.org/dev/peps/pep-0008/>. [Last Access: June 2020].
- [34] Agencia Estatal Boletín Oficial del Estado, «Agencia Estatal Boletín Oficial del Estado,» 11 March 2020. [Online]. Available: https://www.boe.es/diario_boe/txt.php?id=BOE-A-2020-3485. [Last Access: June 2020].
- [35] GlassDoor, «GlassDoor,» 2020. [Online]. Available: <https://www.glassdoor.es/index.htm>. [Last Access: June 2020].
- [36] Iberdrola, «Iberdrola,» Iberdrola, June 2020. [Online]. Available: <https://www.iberdrola.es/en/services/equipment/smart-thermostat>. [Last Access: June 2020].
- [37] Netatmo, «Netatmo,» June 2020. [Online]. Available: <https://www.netatmo.com/en-eu/energy/thermostat>. [Last Access: June 2020].
- [38] ecobee, «ecobee,» June 2020. [Online]. Available: <https://www.merchantmaverick.com/long-term-business-loans/>. [Last Access: June 2020].
- [39] Merchant Maverick, «Merchant Maverick,» June 2020. [Online]. Available: <https://www.merchantmaverick.com/long-term-business-loans/>. [Last Access: June 2020].

- [40] Rancher Labs, «Github,» 6 June 2020. [Online]. Available: <https://github.com/rancher/k3s/blob/master/README.md>. [Last Access: June 2020].
- [41] S. Ruder, «ruder.io,» 19 January 2016. [Online]. Available: <https://ruder.io/optimizing-gradient-descent/>. [Last Access: June 2020].
- [42] K. D. Foote, “Dataversity.net,” 22 June 2017. [Online]. Available: <https://www.dataversity.net/brief-history-cloud-computing/>. [Accessed March 2020].
- [43] missinglink.ai, «missinglink.ai,» [Online]. Available: <https://missinglink.ai/guides/neural-network-concepts/7-types-neural-network-activation-functions-right/>. [Last Access: June 2020].
- [44] L. Dignan, «ZDNet.com,» Between the Lines, 18 July 2019. [Online]. Available: <https://www.zdnet.com/article/google-cloud-gains-in-gartners-2019-cloud-infrastructure-magic-quadrant/>. [Last Access: June 2020].

8 GLOSSARY

AI – Artificial Intelligence

VM – Virtual Machine

GPU – Graphical Processing Unit

IoT – Internet of Things

GPIO – General Input Output

HTTP – Hypertext Transfer Protocol

TCP – Transmission Control Protocol

SSL – Secure Sockets Layer

LAN – Local Area Network

URL – Uniform Resource Locator

DNS – Domain Name System

API – Application Program Interface

LSTM – Long-Short Term Memory

ReLU – Rectified Linear Unit

9 APPENDIX

Raw Data From ‘procTemp’ table in the MySQL database

Below we can find our data collected from the sensors and processed by our web server, further updated by our weather predict service. This is the raw data used by our Neural Network to train itself and generate our predictions.

Table structure for table ‘procTemp’

Column	Type	Null	Default
<i>id</i>	int(11)	No	
temp	float	No	
hum	float	No	
out_temp	float	No	
out_hum	float	No	
clouds	int(11)	No	0
solar_rad	double	No	0
uv	double	No	0
date	datetime	No	
room_id	int(11)	No	

Dumping data for table procReadings

Id/temp/hum/out_temp/out_hum/clouds/solar_rad/uv/date/room_id

34314	22	35	11	83	100	211.78	2.24985	2020-04-02 14:24:00	4
34313	22	35	10	80	100	211.78	2.24985	2020-04-02 14:13:55	4
34312	21	35	10	82	100	211.78	2.24985	2020-04-02 14:03:51	4
34311	21	35	10	81	30	797.576	5.3155	2020-04-02 13:53:46	4
34310	21	35	10	80	30	797.576	5.3155	2020-04-02 13:43:41	4
34309	21	35	10	84	30	797.576	5.3155	2020-04-02 13:33:37	4
34308	21	35	10	84	30	797.576	5.3155	2020-04-02 13:23:32	4
34307	21	35	9	85	30	797.576	5.3155	2020-04-02 13:13:27	4
34306	21	35	9	86	30	797.576	5.3155	2020-04-02 13:03:24	4
34305	21	35	9	86	30	797.576	5.3155	2020-04-02 12:53:19	4
34304	21	35	9	87	30	797.576	5.3155	2020-04-02 12:43:15	4
34303	21	35	9	88	30	797.576	5.3155	2020-04-02 12:33:11	4
34302	22	36	9	90	69	549.393	2.78087	2020-04-02 12:23:08	4
34301	22	36	8	91	69	549.393	2.78087	2020-04-02 12:13:03	4
34300	22	36	8	92	69	549.393	2.78087	2020-04-02 12:02:59	4
34299	22	36	8	92	69	549.393	2.78087	2020-04-02 11:52:56	4
34298	22	36	8	93	69	549.393	2.78087	2020-04-02 11:42:51	4
34297	22	36	8	93	69	549.393	2.78087	2020-04-02 11:32:47	4
34296	22	36	8	93	69	549.393	2.78087	2020-04-02 11:22:43	4
34295	22	36	8	94	69	549.393	2.78087	2020-04-02 11:12:39	4
34294	22	36	8	94	69	549.393	2.78087	2020-04-02 11:02:35	4

34293	21	36	7	94	0	347.9	2.60452	2020-04-02 10:52:31	4
34292	21	36	7	94	0	347.9	2.60452	2020-04-02 10:42:26	4
34291	21	36	7	93	0	347.9	2.60452	2020-04-02 10:32:22	4
34290	21	36	7	93	0	347.9	2.60452	2020-04-02 10:22:18	4
34289	21	36	7	93	0	347.9	2.60452	2020-04-02 10:12:13	4
34288	21	36	7	93	0	347.9	2.60452	2020-04-02 10:02:08	4
34287	21	36	6	92	0	347.9	2.60452	2020-04-02 09:52:04	4
34286	21	36	6	92	0	347.9	2.60452	2020-04-02 09:42:00	4
34285	21	36	6	92	0	347.9	2.60452	2020-04-02 09:31:55	4
34284	21	36	6	92	1	148.21	2.04176	2020-04-02 09:21:51	4
34283	21	36	6	91	1	148.21	2.04176	2020-04-02 09:11:46	4
34282	21	36	6	91	1	148.21	2.04176	2020-04-02 09:01:42	4
34281	21	36	6	90	1	148.21	2.04176	2020-04-02 08:51:38	4
34280	21	36	5	91	1	148.21	2.04176	2020-04-02 08:41:33	4
34279	21	36	5	90	1	148.21	2.04176	2020-04-02 08:31:29	4
34278	21	36	5	91	1	148.21	2.04176	2020-04-02 08:21:24	4
34277	21	36	6	91	1	148.21	2.04176	2020-04-02 08:11:20	4
34276	21	36	6	91	1	148.21	2.04176	2020-04-02 08:01:17	4
34275	21	36	5	91	2	0	0	2020-04-02 07:51:14	4
34274	21	36	6	91	2	0	0	2020-04-02 07:41:10	4
34273	21	36	6	90	2	0	0	2020-04-02 07:31:06	4
34272	21	36	6	90	2	0	0	2020-04-02 07:21:01	4
34271	21	36	6	90	2	0	0	2020-04-02 07:10:56	4
34270	21	36	6	90	2	0	0	2020-04-02 07:00:52	4
34269	21	36	6	90	2	0	0	2020-04-02 06:50:47	4
34268	21	36	6	90	2	0	0	2020-04-02 06:40:43	4
34267	21	36	6	90	2	0	0	2020-04-02 06:30:38	4
34266	21	36	6	90	0	0	0	2020-04-02 06:20:34	4
34265	21	36	6	89	0	0	0	2020-04-02 06:10:30	4

CSS Stylesheet from Dashboard

As discussed during the project, this is the stylesheet that allows our dashboard to display our responsive layout and our interactive graphs.

```
.centerAligned tr th, .centerAligned tr td {
  text-align: center
}

.centerAlign {
  width: auto
}

.splitBox2 {
  float: left;
  width: 100%
}

* {
  box-sizing: border-box;
}

.rowA {
  display: -ms-flexbox; /* IE10 */
  display: flex;
  -ms-flex-wrap: wrap; /* IE10 */
  flex-wrap: wrap;
  margin: 0 -16px;
}

.col-25 {
  -ms-flex: 25%; /* IE10 */
  flex: 25%;
  padding: 30px;
}

.column50 {
  -ms-flex: 50%; /* IE10 */
  flex: 50%;
  padding: 30px;
}

.col-75 {
  -ms-flex: 75%; /* IE10 */
  flex: 75%;
  padding: 30px;
}

.col-25,
.col-50,
.col-75 {
  padding: 0 16px;
}

.select-css {
  display: block;
  font-size: 16px;
  font-family: sans-serif;
  font-weight: 700;
  color: #444;
  line-height: 1.3;
  padding: .6em 1.4em .5em .8em;
  width: 100%;
  max-width: 100%;
  box-sizing: border-box;
  margin: 0;
  border: 1px solid #aaa;
}
```

```

    box-shadow: 0 1px 0 1px rgba(0,0,0,.04);
    border-radius: .5em;
    -moz-appearance: none;
    -webkit-appearance: none;
    appearance: none;
    background-color: #fff;
    background-image: url()
      linear-gradient(to bottom, #ffffff 0%,#e5e5e5 100%);
    background-repeat: no-repeat, repeat;
    background-position: right .7em top 50%, 0 0;
    background-size: .65em auto, 100%;
  }
  .select-css::-ms-expand {
    display: none;
  }
  .select-css:hover {
    border-color: #888;
  }
  .select-css:focus {
    border-color: #aaa;
    box-shadow: 0 0 1px 3px rgba(59, 153, 252, .7);
    box-shadow: 0 0 0 3px -moz-mac-focusring;
    color: #222;
    outline: none;
  }
  .select-css option {
    font-weight:normal;
  }

  .container {
    padding: 5px 20px 15px 20px;
  }

  input[type=text] {
    width: 100%;
    margin-bottom: 20px;
    padding: 12px;
    border: 1px solid #ccc;
    border-radius: 3px;
  }

  label {
    margin-bottom: 10px;
    display: block;
  }

  .icon-container {
    margin-bottom: 20px;
    padding: 7px 0;
    font-size: 24px;
  }

  .btn {
    background-color: #4CAF50;
    color: white;
    padding: 12px;
    margin: 10px 0;
    border: none;
    width: 100%;
    border-radius: 3px;
    cursor: pointer;
    font-size: 17px;
  }

  .btn:hover {
    background-color: #45a049;
  }

  * {
    box-sizing: border-box;
  }

  a {
    color: #2196F3;
  }

```

```

hr {
  border: 1px solid lightgrey;
}

span.price {
  float: right;
  color: grey;
}

/* Responsive layout - when the screen is less than 800px wide, make the two
columns stack on top of each other instead of next to each other (also change
the direction - make the "cart" column go on top) */
@media (max-width: 800px) {
  .rowA {
    flex-direction: column;
  }
  .col-25 {
    margin-bottom: 20px;
  }
}

.modal {
  display: none; /* Hidden by default */
  position: fixed; /* Stay in place */
  z-index: 1; /* Sit on top */
  padding-top: 100px; /* Location of the box */
  left: 0;
  top: 0;
  width: 100%; /* Full width */
  height: 100%; /* Full height */
  overflow: auto; /* Enable scroll if needed */
  background-color: rgb(0,0,0); /* Fallback color */
  background-color: rgba(0,0,0,0.4); /* Black w/ opacity */
}

/* Modal Content */
.modal-content {
  position: relative;
  background-color: #fefefe;
  margin: auto;
  padding: 0;
  border: 1px solid #888;
  width: 80%;
  max-width: 650px;
  box-shadow: 0 4px 8px 0 rgba(0,0,0,0.2),0 6px 20px 0 rgba(0,0,0,0.19);
  -webkit-animation-name: animatetop;
  -webkit-animation-duration: 0.4s;
  animation-name: animatetop;
  animation-duration: 0.4s
}

/* Add Animation */
@-webkit-keyframes animatetop {
  from {top:-300px; opacity:0}
  to {top:0; opacity:1}
}

@keyframes animatetop {
  from {top:-300px; opacity:0}
  to {top:0; opacity:1}
}

```