Edge Computing Solution for Pet Detection

FRANKFURT UNIVERSITY OF APPLIED SCIENCES

CLOUD COMPUTING - SS 2023

UNDER THE GUIDANCE OF

PROF. DR. CHRISTIAN BAUN

BY

Syeed, Nur Uddin Muhammad(1420731)

Saddam Hossain (1440339)

Md Motaleb Hossain(1396506)

Omme Salma (1420582)

NASHRIF, A B M(1404537)

IMRUL KAYES TALUKDAR(1396425)

July 2023

0. Table Of Content

1. Introduction

Today, cluster computing has grown rapidly in the field of distributed and parallel processing. A cluster is a collection of computers that are connected together in a parallel manner using a high-speed local area network. These computers work together to process data in parallel and can be viewed as a single system.

Another Important term in cluster computing is Edge computing which is a distributed computing paradigm. It is the act of running workloads on edge devices that helps to Enhance efficiencies, minimized latency, and increase uptime.

This project aimed to develop an edge computing solution for the automatic detection of pets (Here cats and dogs) using Raspberry Pi as an edge/sensor node and multiple Raspberry Pis connected together and used as a single cluster. Nowadays we can get hardware that can do moderate computation very easily. we can distribute some computation on the edge sensor and make the cloud app more cost-friendly and efficient. A Raspberry Pi is a low-cost, single-board computer. For example, Raspberry Pi can be used as an edge node and can detect dogs or cats using a pre-trained Model and send only the detection data to the cloud (Kubernetes Cluster). In this way, the cloud will have to only take the responsibility of further data analysis.

1.1 Objective

The main objective of this project is to develop an edge computing solution for the automatic detection of pets (Cat, Dog) using Raspberry pi and k3s Cluster. This includes:

- Developing and setup up sensor nodes with Raspberry Pi 4 and camera modules
- Deployment of the operating system and object detection software
- Collecting a sufficient number of images for the training and testing of the object detection model
- Deploy the backend as a docker container on K3s to manage the sensor nodes and the collected data
- Deploy frontend as a docker container on k3s to present e.g. log information, event messages, and a map of events with proof like images and timestamps.
- Use REST API protocol to interact between the components (frontend, backend, Edge node, and other services)

To obtain this, we Set up an Edge node using Raspberry Pi 4 and Camera Module and deploy several micro-services on the Kubernetes Cluster (k3s) which are:

1. MySQL to store capture-related metadata for further analysis.
2. MinIO   Data storage to store images

3. API service for the data communication/exchange
4. Frontend Application to visualize the summary of the collected data

1.2 Required types of equipment

- 1 Raspberry Pi 4 Model B with 1 Samsung Pro Plus 32GB MicroSD memory card
- 4 Raspberry Pi 3 Model B with 4 Samsung Evo Plus 32GB MicroSD memory card
- 1 Raspberry Pi power adapter 3
- 22 bolts for Raspberry Pi construction
- 1 TP-Link Switch
- 1 Raspberry Pi camera module V2
- 5 LAN cable

1.3 System Architecture and project implementation detail

As we are given five Raspberry PIs and one of them is pi4 and the others are pi3, we have used the most powerful pi which is pi4 as the sensor/edge node to capture and detect images. This is because here capturing images and detecting them is the most heavy job. We have used other 4 pi3 to create the K3S cluster. We installed a headless light os on each node to avoid unnecessary loads.

Here Sensor node is responsible to capture and detecting of images. After detection, it only sends the detection data to the cloud if it can detect any cat/dog.

We have used the rest of the 4 raspberry pi3 to create the Kubernetes cluster on which our services reside which are MySql, MinIO, and WebApp. We have used the MySQL and MinIO images from the docker hub to deploy MySQL and MinIO the Kubernetes. The core responsibility which is receiving the data from SensorNodes and storing them and making them available to the front end is managed by WebApp which is a REST API-based API service. To deploy that image on K3S we dockerized the image and made it publicly available on dockerhub and used that image from docker-hub to deploy it on Kubernetes. Our Frontend is also integrated with the API service, So the front end also got deployed along with the API service. Once everything is deployed, the system is ready to receive sensor data and analyze them.

1.3.1 System Diagram

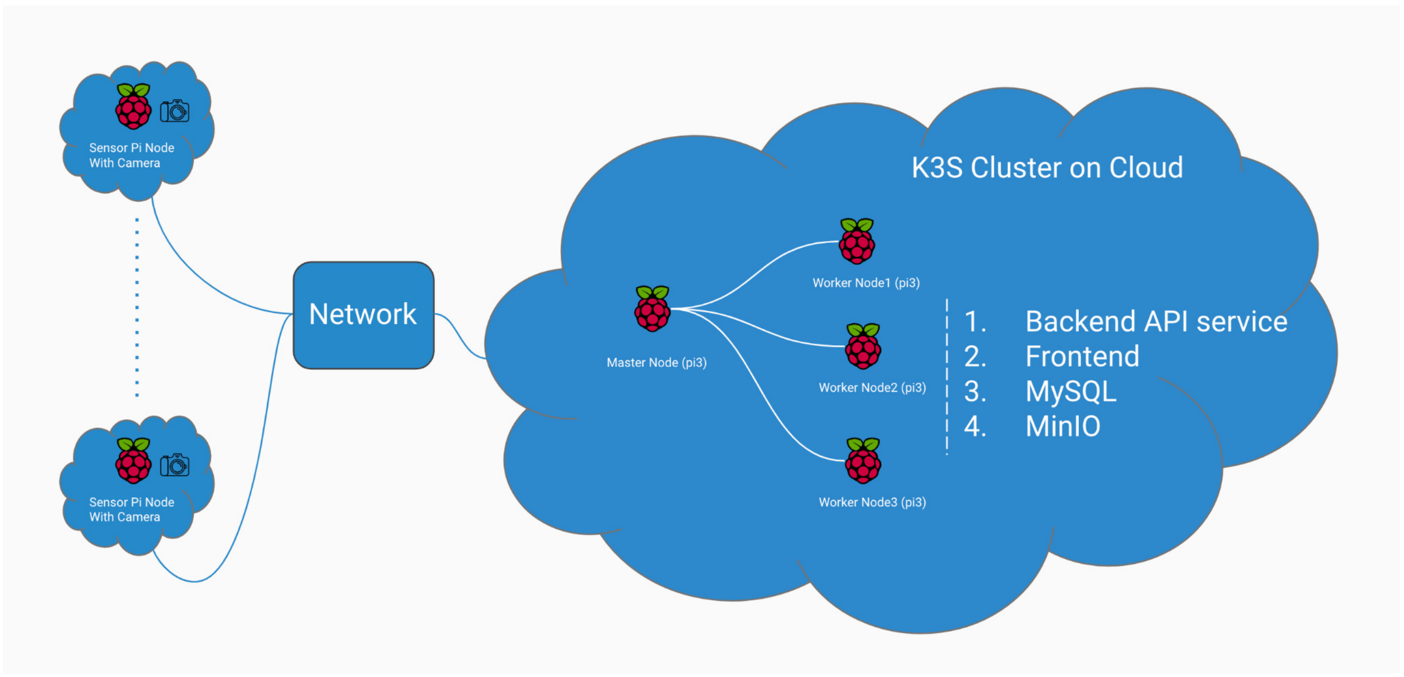


Figure 1: Overall System Setup

1.3.2 System Architecture

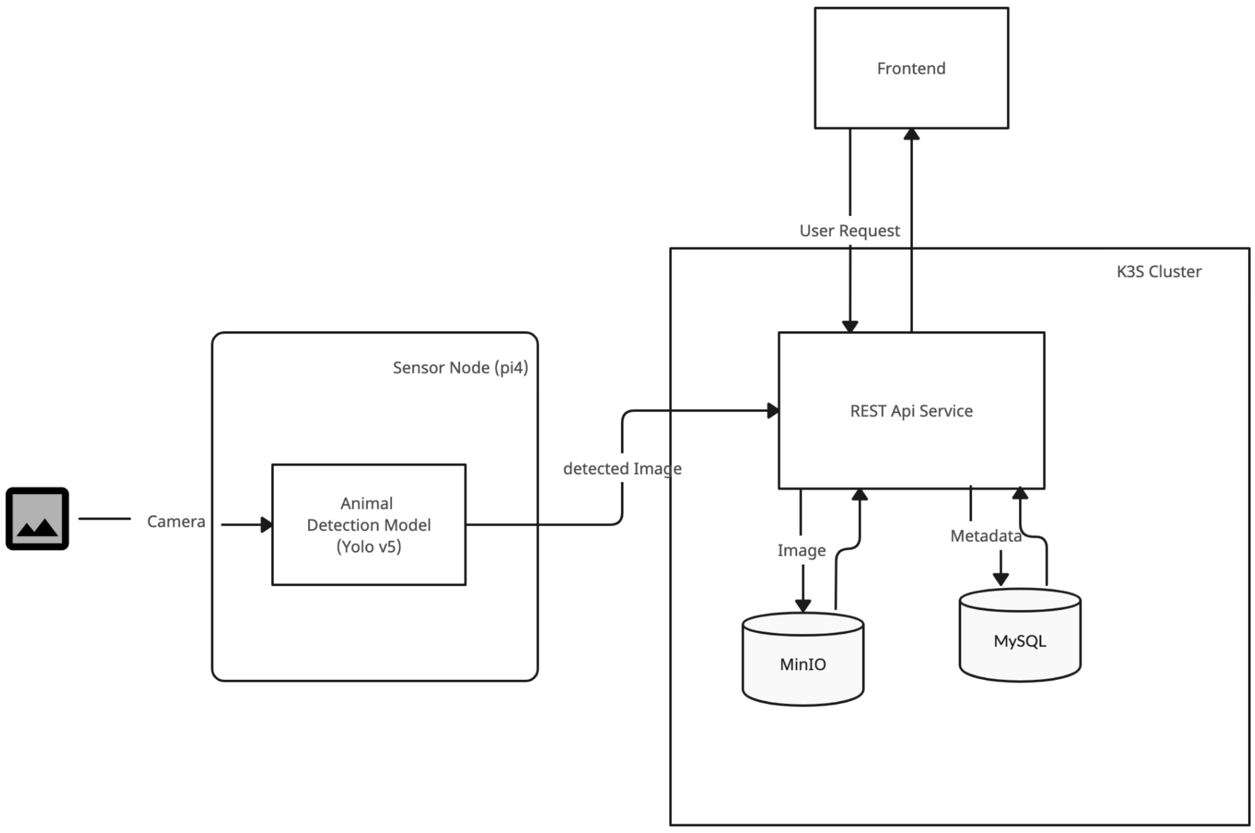


Figure 2: System Architecture

1.3.3 Pet Detection and Storage Process

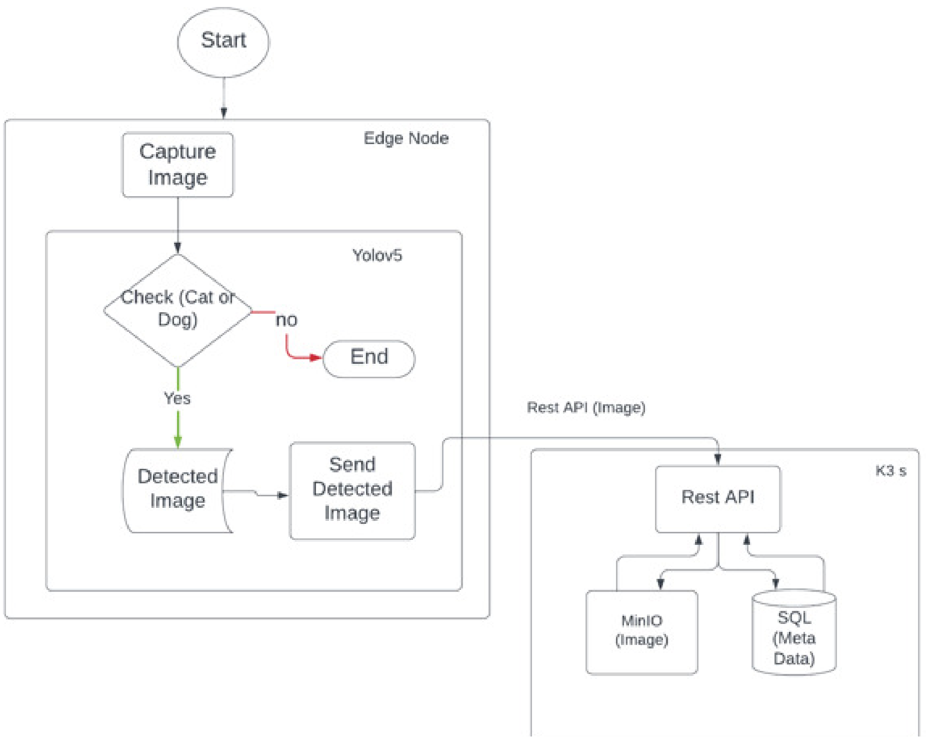


Figure 3: Pet Detection and Storage Process in K3s (Level 1)

1.3.4 User view of the System-Frontend

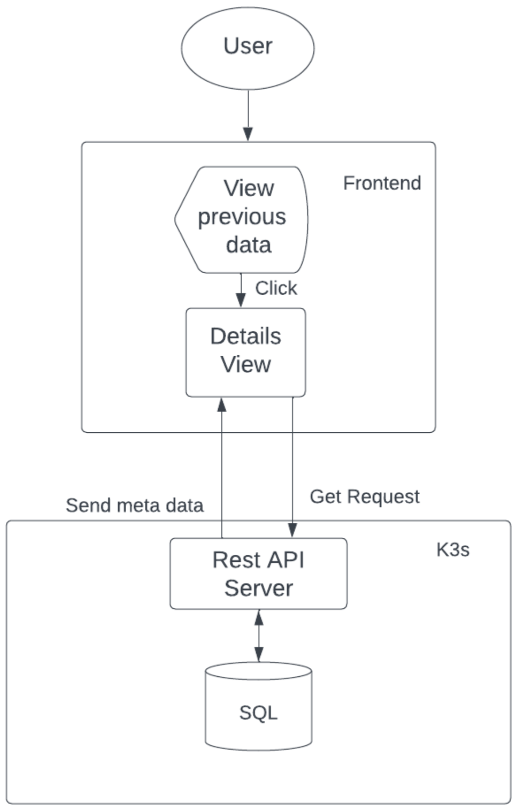


Figure 3: User view of the System-Frontend (Level 2)

2 Task Distribution:

Task distribution table among the team members.

Task	Member Name
Installing OS and K3S Cluster preparation	Omme Salma
YOLO Model Preparation	Saddam Hossain
Sensor preparations and Sensor Script	Nur Uddin Syeed
Backend REST API service Development	Nur Uddin Syeed and Motaleb Hossain
Front-End Development	Nashrif
Dockerization of the application	Imrul Kayes
Research and System Design on cluster setup / minIO / Persistent Storage	Motaleb Hossain
Development of K3S Scripts and apps deployment and networking on K3S Cluster	Nur Uddin Syeed and Nashrif

3 Operating System Installation

To operate, Raspberry Pi requires an operating system. Raspberry Pi Imager serves as the endorsed imaging software by Raspberry Pi and can be obtained from the official website at https://www.raspberrypi.org/. This software provides a convenient and efficient method to install Raspberry Pi OS and various other operating systems onto a microSD card, enabling seamless usage with Raspberry Pi. It is compatible with MacOS, Windows, Linux, and Ubuntu systems. To proceed, download and install Raspberry Pi Imager onto your computer, ensuring that you have an SD card reader. Insert the SD card intended for use with your Raspberry Pi into the reader, and launch Raspberry Pi Imager.

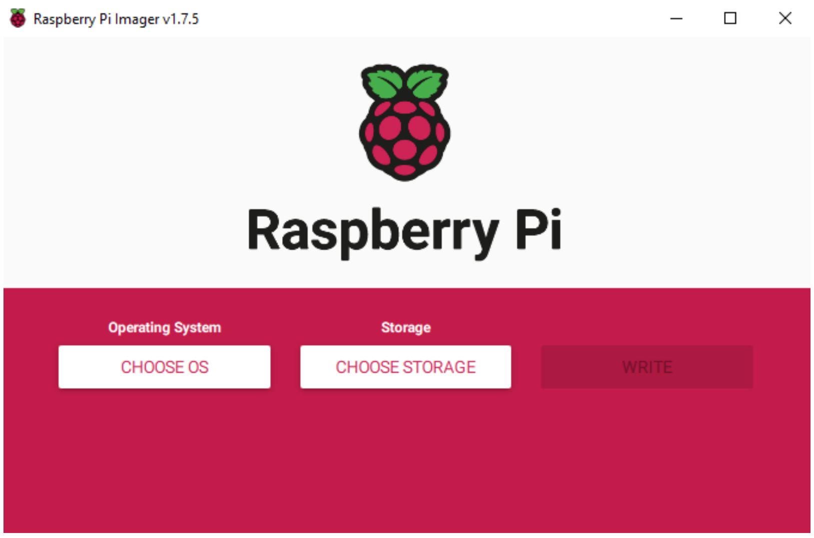
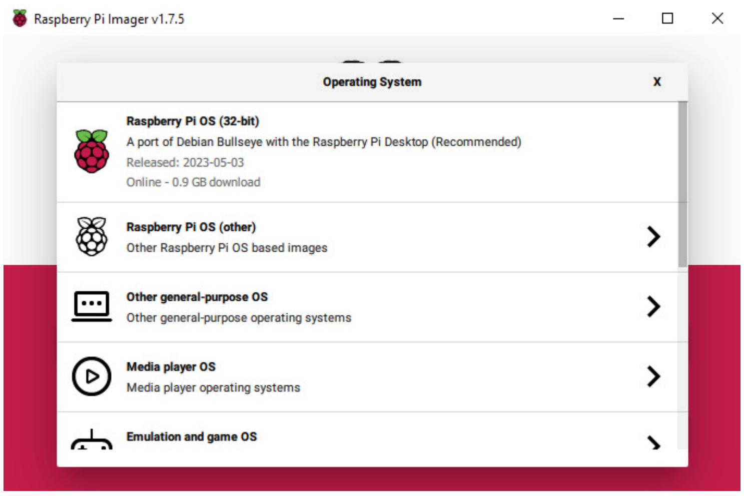


Figure 4: Raspberry Pi Imager

The Raspberry Pi Manager is a user-friendly application designed to facilitate the installation of various operating systems, including the recommended Raspbian Desktop, which is a Debian-Linux based operating system. Additionally, it offers alternative options such as the popular Raspberry Pi OS Lite, Ubuntu Mate, OSMC, and Retro Pie. It's worth noting that Raspbian Desktop serves as the official and primary operating system for Raspberry Pi.



Raspberry Pi OS (32-bit)
A port of Debian Bullseye with the Raspberry Pi Desktop (Recommended)
Released: 2023-05-03
Online - 0.9 GB download

Raspberry Pi OS (other)
Other Raspberry Pi OS based images

Other general-purpose OS
Other general-purpose operating systems

Media player OS
Media player operating systems

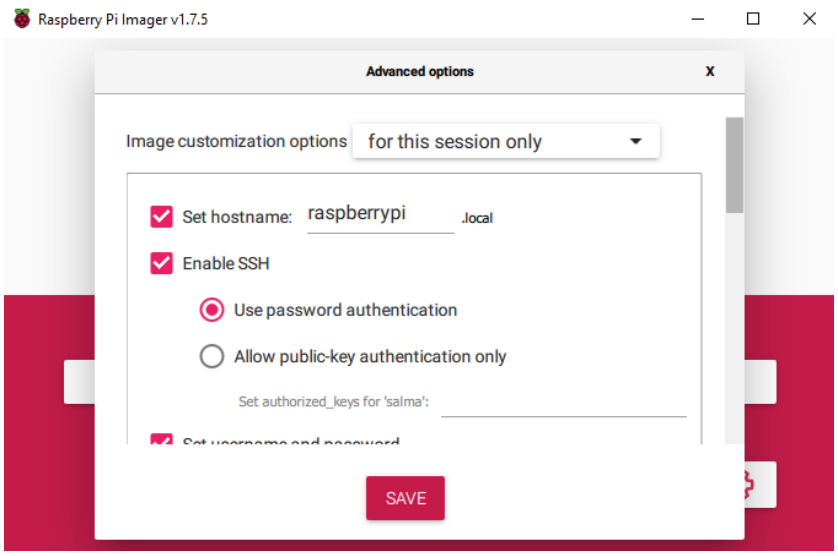
Emulation and game OS

Figure 5: Raspberry pi Imager (OS Selection)

1. Select an OS from the list of Raspberry Pi OS.
2. Click Choose SD card. Select from the list the SD card you want to write to.
3. To set up a raspberry pi in headless mode without the need for additional peripherals such as a monitor, keyboard, and mouse.
   I. Click on the advanced options of the Imager
   II. Enable SSH and set up a custom Hostname
   III. Configure Wi-Fi, by entering the SSID and

IV. the password of your Wi-Fi network
IV. Click on SAVE

4. Click Write to begin the image writing process.



Raspberry Pi Imager v1.7.5

Advanced options

Image customization options: for this session only

[✓] Set hostname: raspberrypi .local

[✓] Enable SSH

[● Use password authentication]

○ Allow public-key authentication only

Set authorized_keys for 'salma':

Figure 6: Raspberry pi SSH setup

After successfully installing the chosen operating system onto the SD card, we can proceed to insert the SD card into our Raspberry Pi in order to boot it. After Booting it will create a boot file on the SD card. Unplug the power to raspberry pi and SD card back into the usb device into the computer. Since we will install k3s Kubernetes on Raspberry, we have to update these two files to avoid errors during installation time.

1. In /boot/cmdline.txt file, The following text needs to be appended at the end of the file.

```
cgroup_memory=1cgroup_enable=memory ip=192.168.0.58 :: 192.168.1.1 :
255.255.255.0 : rpimaster: eth0 : off
```

2. In *the* config.txt file, the following line needs to be added

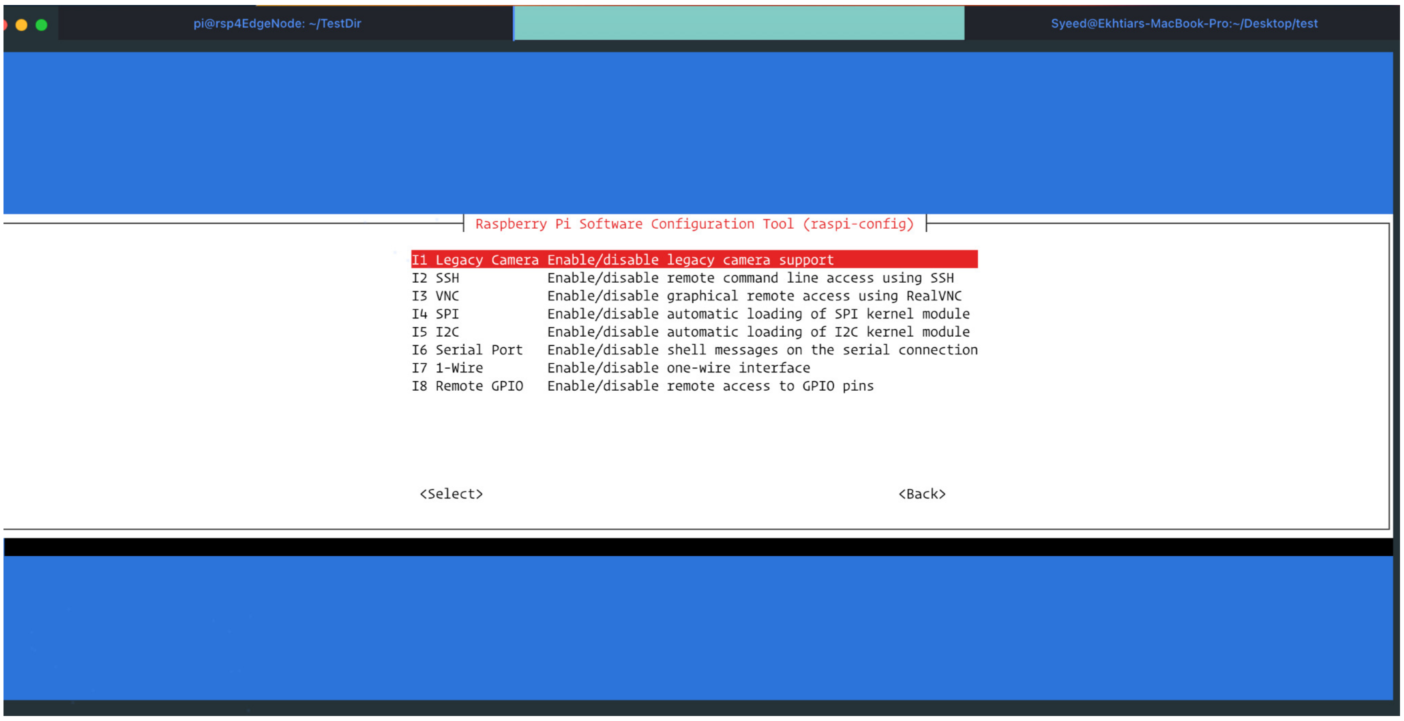
```
arm_64bit=1
```

After the save, the node needs to be rebooted. Similarly, install OS for on other raspberry pi worker nodes. To verify their availability, ping them using their device IP Address.

```
ping 192.168.0.58 -t
```

4 Sensor Node:

We have used Raspberry pi4 and pi camera v2 as the sensor node. Before start we also need to enable the camera of the sensor node.


pi@rsp4EdgeNode: ~/TestDir
Syeed@Ekhtiars-MacBook-Pro:~/Desktop/test

```
Raspberry Pi Software Configuration Tool (raspi-config)

I1 Legacy Camera Enable/disable legacy camera support
I2 SSH           Enable/disable remote command line access using SSH
I3 VNC           Enable/disable graphical remote access using RealVNC
I4 SPI           Enable/disable automatic loading of SPI kernel module
I5 I2C           Enable/disable automatic loading of I2C kernel module
I6 Serial Port   Enable/disable shell messages on the serial connection
I7 1-Wire        Enable/disable one-wire interface
I8 Remote GPIO   Enable/disable remote access to GPIO pins

<Select>                                          <Back>
```

4.1 Setup:

We have installed a headless Raspberry pi os on our sensor node. To be able to run our script on the sensor we had to install several packages on the Raspberry Pi, which include:

1. OpenCV and Numpy (Used to capture the image and process the image)
2. Pytorch (Used to load the pre-trained cat-dog detection model and detect pets.

4.2 Image detection on sensor node:

We have deployed a Python script to the sensor node to capture images and detect the pet. It just keeps capturing images in every second (In our case more than one second as the hardware takes time to run heavy computation for the image detection algorithm).
To start detection all one needs to do is connect to the sensor node via ssh and run the script by:

```
pip install -r requirements.txt
```

```
python run.py
```

This will start capturing the images. After capturing it starts processing the image and reduce the size, so that our detection model can process the image faster. If any cat or dog is detected it will do following tasks:

1. Draw the image
2. Convert the image to base64 string
3. Send the image data along with the detection data to the API server.

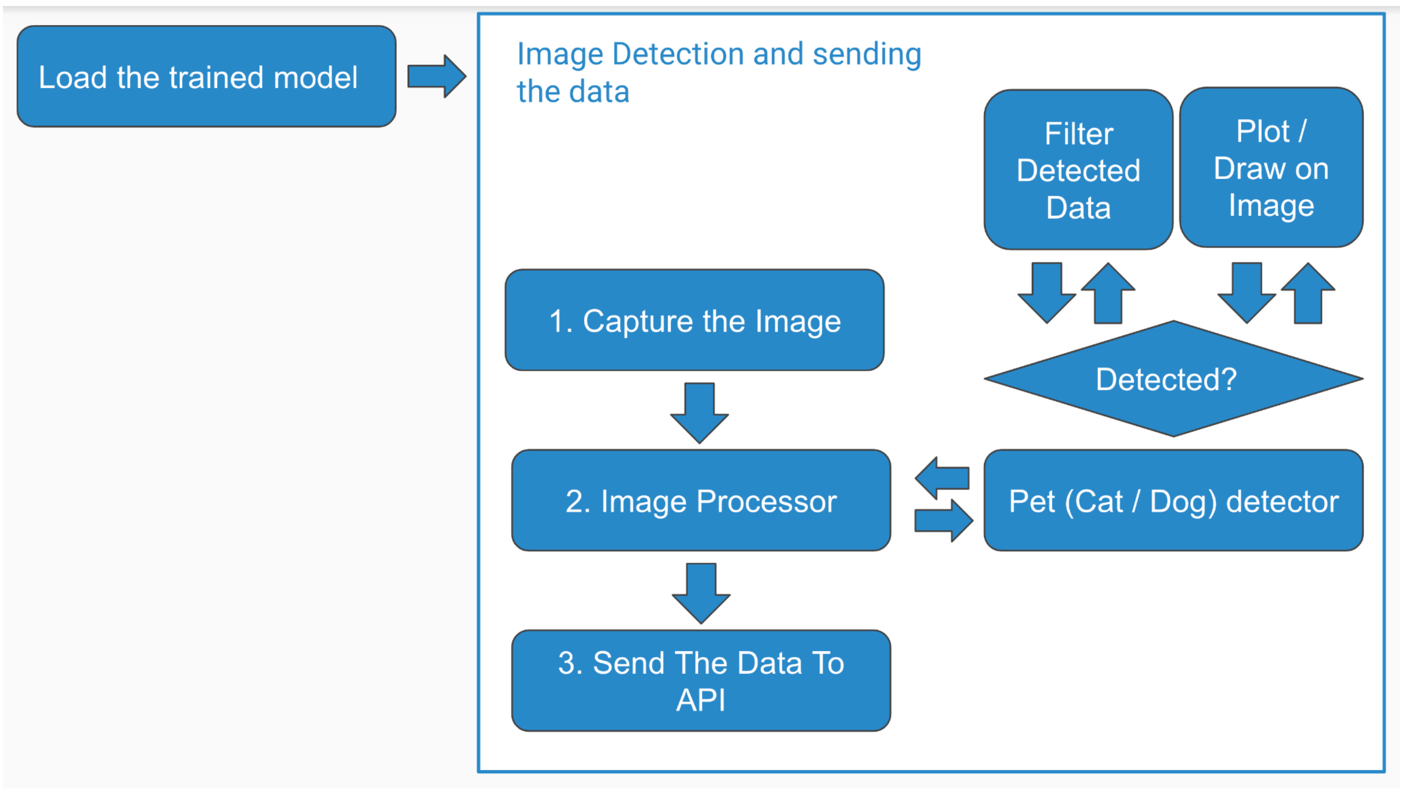


Figure: Sensor Node Workflow

4.3 Sequence Diagram of the Sensor Node

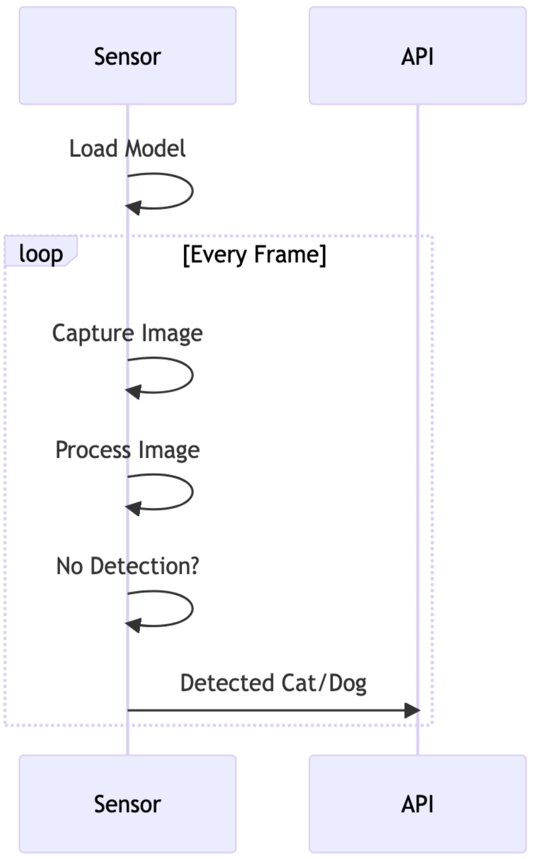


Figure: Image Capture and Detection on Edge Node

5. Pet Detection Model Preparation(YOLOv5)

YOLO (You Only Look Once) is a popular algorithm for object detection that uses a single neural network to predict bounding boxes and class probabilities directly from full images. YOLOv5, the fifth iteration of YOLO object detection model built on PyTorch and it is one of the fastest and most accurate object detection models available.

5.1 Data Collection

The first step in training a YOLOv5 model is to collect a dataset of images that we want the model to be able to detect objects in. For YOLOv5 model, we need labeled images with bounding boxes around the objects and assign a class label to each bounding box. We collect 3686 images of cat and dog's xml files with bounding box and class information from here
https://www.kaggle.com/datasets/andrewmvd/dog-and-cat-detection/code .

5.2 Data Preprocess and YOLO Format Conversion

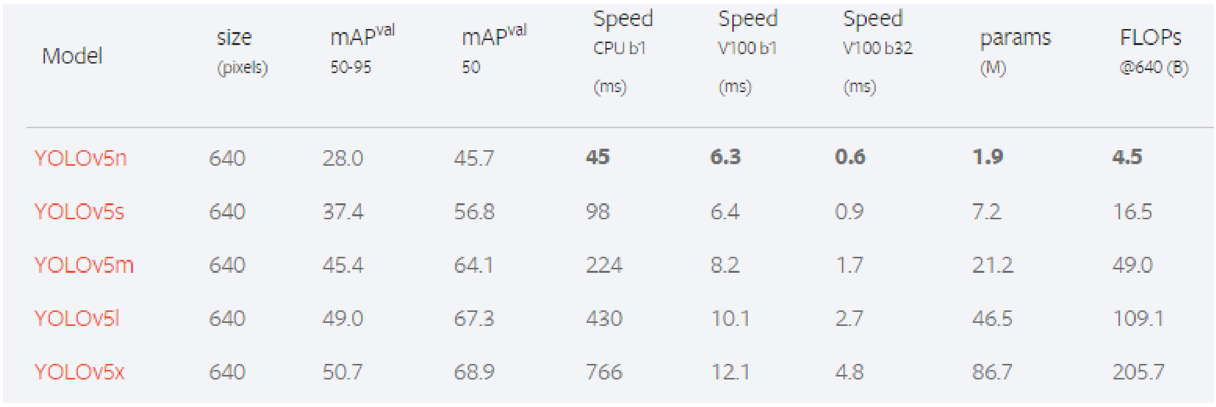
After Collecting the labeled dataset, we convert it Xml to .txt format as YOLO v5 expects annotations for each image in the form of .txt file where each line of the text file describes a bounding box. It requires the dataset to be in the YOLO format. Here's an outline of what it looks like:

- One text with labels file per image
- One row per object
- Each row contains: class_index bbox_x_center bbox_y_center bbox_width bbox_height
- Box coordinates must be normalized between 0 and 1

After converting the dataset into yolo format, we split the dataset into a training set and a test set for model training.

5.3 YOLOv5 Model Version Selection

YOLOv5 is available in four models, namely small (s), medium (m), large (l), and extra-large (x), each one of them offering different detection accuracy. Each variant also takes a different amount of time to train.



Model	size (pixels)	mAP^{val} 50-95	mAP^{val} 50	Speed CPU b1 (ms)	Speed V100 b1 (ms)	Speed V100 b32 (ms)	params (M)	FLOPs @640 (B)
YOLOv5n	640	28.0	45.7	**45**	**6.3**	**0.6**	**1.9**	**4.5**
YOLOv5s	640	37.4	56.8	98	6.4	0.9	7.2	16.5
YOLOv5m	640	45.4	64.1	224	8.2	1.7	21.2	49.0
YOLOv5l	640	49.0	67.3	430	10.1	2.7	46.5	109.1
YOLOv5x	640	50.7	68.9	766	12.1	4.8	86.7	205.7

Figure :YOLOv5 model version

We select a model to start training from. Considering computational power for the training process, we selected YOLOv5m, the third smallest and fastest model available.

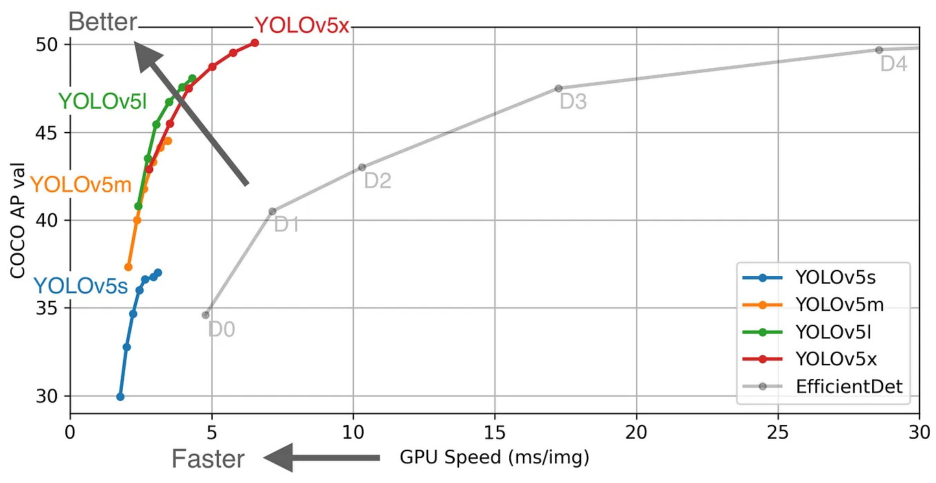


Figure : YOLOv5 Model performance

5.4 Model Training and Validation

To train the YOLOv5 model, navigate to the YOLOv5 folder and execute the following command:

```
python train.py --data ../data.yaml --weights yolov5m.pt \
 --img 352 --epochs {EPOCHS} --batch-size 16 --name {RESULT_DIR}
```

Here,

- –img: quadratic image size in pixels
- –batch: training data used per training epoch
- --epoch: maximum number of training steps
- –data: data.yaml, that comes with the downloaded data set
- –weights: trained weights from Yolo which is yolov5m.pt.

Depending on the machine used, model training may take a long time. We set epoch=20 and training is done on our local machine, and it takes 7.750 hours to train. Then model evaluation is done on the validation set. After the model is trained and evaluated, test it with new cat and dog images. Here is the github link of our model https://github.com/Sabuj-CSE11/AnimalDetection (Only for the model preparation).

6. Installation and Setup of K3s (Cluster Setup)

3.1 Install k3s on Master Node

1. First launch terminal
2. Go to ssh into the first Raspberry Pi where we want to install the master node using the hostname or IP address.

`ssh pi@192.168.0.58` and Authenticated via password.

3. Go to root using the command sudo su -
4. Install k3s server with:

```
curl -sfL https://get.k3s.io | K3S_KUBECONFIG_MODE="644" sh -s -
```

5. After installing k3s on master node, check if Kubernetes works and the master node is available with this command

```
kubectl get nodes
```

6. In order to install the k3s worker node on the other Raspberry Pi we need to know the access token from the master node. To extract the token run the following commands.

“cat /var/lib/rancher/k3s/server/node-token”

This command generates a token that is used to register the worker node to the cluster. E.g.

```
K101568b95ffbf1ddc0dfdsad1d87eb702eb04fce3376204be44d5eded02831a36f::server:8
3684b530e6562f86b84d5d5bf4a2eab
```

3.1 Install k3s on Worker Node

1. First launch terminal
2. Go to ssh into the first Raspberry Pi where we want to install the master node using the hostname or IP address.

```
ssh pi @192.168.0.60 and Authenticated via password.
```

3. Go to root using the command

```
sudo su -
```

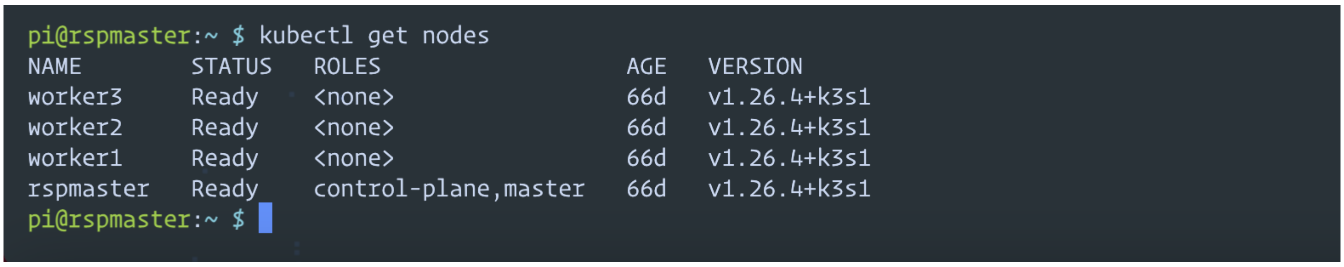
4. Install k3s worker with

```
curl -sfL https://get.k3s.io |
K3S_TOKEN=K1032197fb8d62f0e99f92997ab627427f8f5dea03c583c313bc4a9f5b44ad97cc6
::server:4e333637efa34d5a4e352791a3911b93 K3S_URL="https://192.168.0.59:6443"
K3S_NODE_NAME="worker1" sh -
```

5. Repeat this step for all other 3 worker nodes

Now We can check the information about the master and worker nodes by running the following command in the terminal:

```
Kubectl get nodes
```



```
pi@rspmaster:~ $ kubectl get nodes
NAME         STATUS   ROLES                  AGE   VERSION
worker3      Ready    <none>                 66d   v1.26.4+k3s1
worker2      Ready    <none>                 66d   v1.26.4+k3s1
worker1      Ready    <none>                 66d   v1.26.4+k3s1
rspmaster    Ready    control-plane,master   66d   v1.26.4+k3s1
pi@rspmaster:~ $
```

7. Frontend:

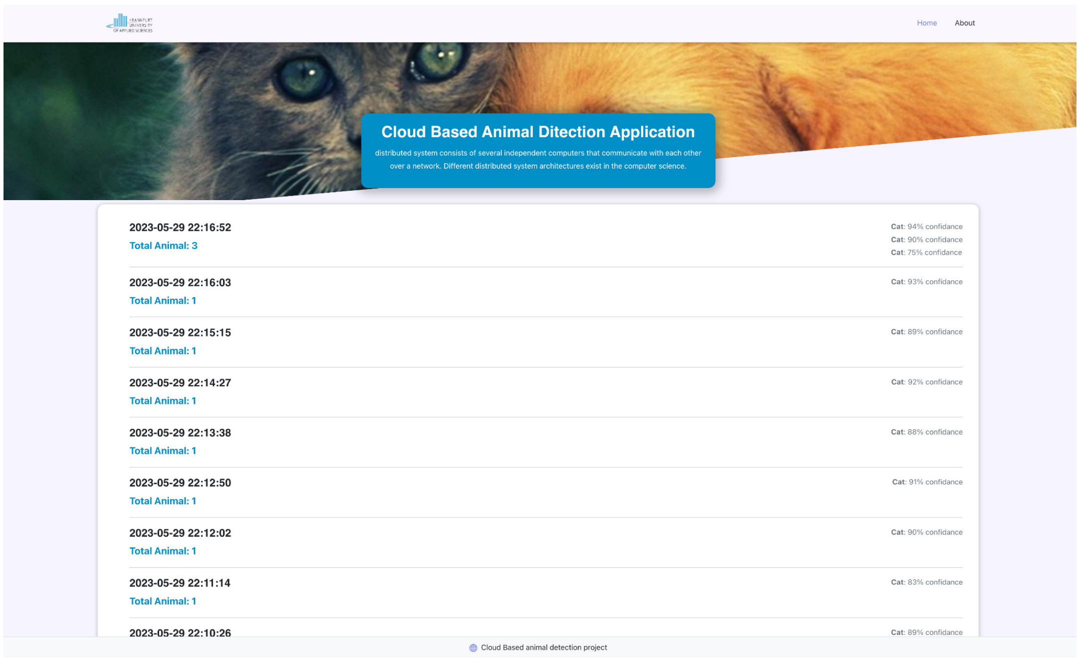
We have integrated the frontend along with the Backend service and used jinja2 to generate HTML pages. The main task of the front is to enable the user to analyze the detection data detected by all sensor nodes (In our case only one sensor node).

The way to run the front end is explained in the Backend section. Once the frontend is running user can access the frontend using any ip of the cluster and on port 30080.

```
http://192.168.0.60:30080/
http://192.168.0.61:30080/
http://192.168.0.62:30080/
http://192.168.0.63:30080/
```

In the home page the latest detection data from the sensor node is shown but it is possible to fetch specific data efficiently as we have used MySQL to store the data. Depending on the use case anyone can add a specific fetch API endpoint and show the specific date to the user.

Users can view all the data as a list and view the detail with the image of a single detection by clicking over the list of the data.



Cloud Based Animal Ditection Application

distributed system consists of several independent computers that communicate with each other over a network. Different distributed system architectures exist in the computer science.

2023-05-29 22:16:52
Total Animal: 3
Cat: 94% confidance
Cat: 90% confidance
Cat: 75% confidance

2023-05-29 22:16:03
Total Animal: 1
Cat: 93% confidance

2023-05-29 22:15:15
Total Animal: 1
Cat: 89% confidance

2023-05-29 22:14:27
Total Animal: 1
Cat: 92% confidance

2023-05-29 22:13:38
Total Animal: 1
Cat: 88% confidance

2023-05-29 22:12:50
Total Animal: 1
Cat: 91% confidance

2023-05-29 22:12:02
Total Animal: 1
Cat: 90% confidance

2023-05-29 22:11:14
Total Animal: 1
Cat: 83% confidance

2023-05-29 22:10:26
Cat: 89% confidance

Cloud Based animal detection project

8. Backend API

8.1 Backend Framework (FastAPI)

The Backend web app is developed using one of the modern Python web frameworks named FastAPI. FastAPI is a fast, and highly efficient Python web framework for building APIs. It was created to provide developers with a high-performance alternative to existing web frameworks, such as Flask and Django, while maintaining ease of use and simplicity. Overall, FastAPI combines the best features of modern Python and asynchronous programming to deliver high-performance web APIs with a developer-friendly experience. Its simplicity, speed, and scalability make it an excellent choice for building robust and efficient API services.

8.2 Running the application:

Running the application is super simple. All one needs to do is follow the following 5 steps.

1. Clone the project

```
git@github.com:nuruddinsayeed/CloudApp101.git
```

2. Goto WebApp Directory
3. Create a virtual environment

```
Python -m venv venv
```

4. Activate virtual environment

```
source venv/Scripts/activate (Linux and Mac)
venv/Scripts/activate (Windows)
```

5. Install all requirements

```
pip install -r requirements.txt
```

6. Run the app

```
python run.py
```

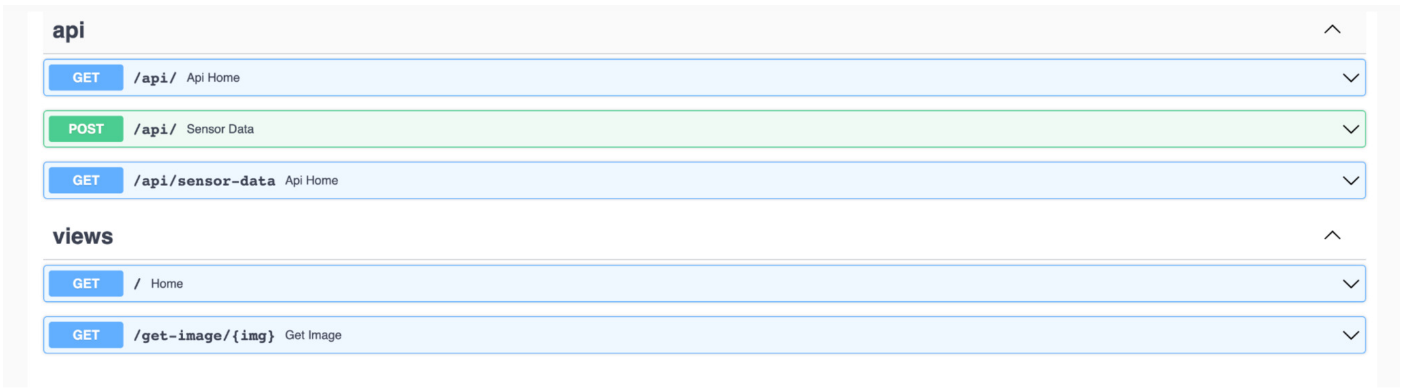
8.3 API Endpoints:

Our main requirement is to receive the data from the sensor and store it in a database and enable it to perform operations on the data. For that, initially, we have created three API endpoints:

1. Data receive endpoint (to receive data from sensors)
2. Api-Home to send the latest detection data to the frontend

3. Api-Home-Detail for viewing the detail of specific detection

Because using cluster our web app can handle a good amount of requests and process them, although we have used only on sensor node in our case. But if we use multiple sensor nodes and send data at a time the backend service will not complain as it is able to handle a good amount of sensor data at a time because of load balancing functionality in our cluster


api

GET /api/ Api Home

POST /api/ Sensor Data

GET /api/sensor-data Api Home

views

GET / Home

GET /get-image/{img} Get Image

Figure: API endpoints

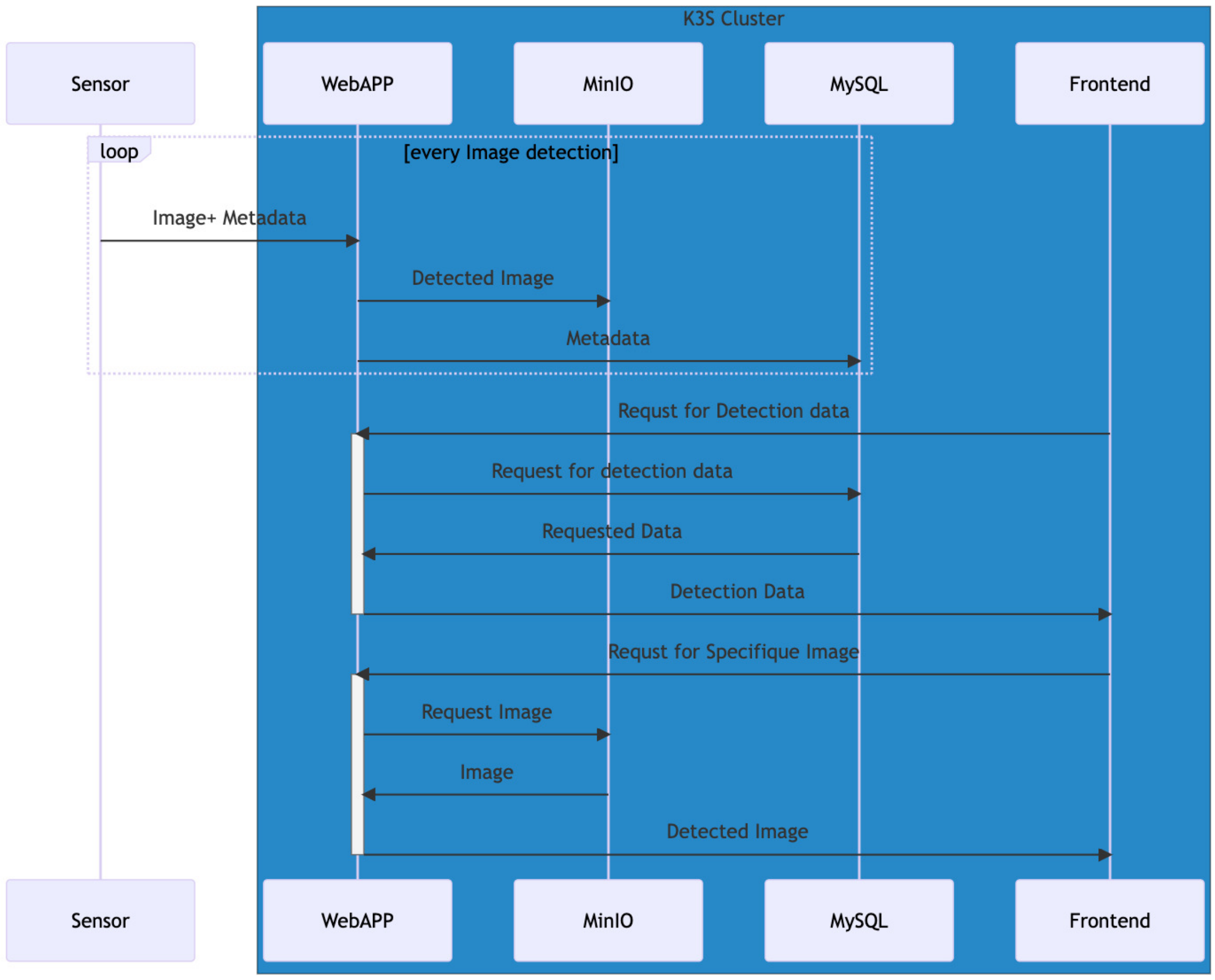
The API endpoint expects the data in the following format as a post request:

```
//Image
{"image": "base64............"

// Data
{
   "detected_at": "detection time",
   "confidences": [
      "('Animal1', 99)",
      "('Animal 2', 90)",


      "('Animal N', 89)",
   ]
}
```

8.4 Sequence Diagram of the BackendAPP:



8.5 Data storage:

To make the data analysis easier and fetch operation faster we have used two different database systems. One is MinIO and the other one is MySQL. MinIO is used only to store the detected image and detection data is stored in MySQL. Also having three workers means we can deploy and use two database images on the container without adding any extra delay for the computation. Besides having different data storage for **structured data** and **images** allowed us to perform fetch operations much faster.

8.5.1 MinIO:

We found MinIO as a perfect choice for us that can fulfill our requirements. MinIO is one of the popular high-performance object storage servers. It is designed to be simple, lightweight, and compatible with the Kubernetes cluster, making it an excellent alternative to proprietary cloud storage services for us. It is compatible with K3S and it has a docker image which is runnable in our Raspberry Pi (ARM processor).

8.5.1 MySQL:

MySQL is an open-source relational database management system (RDBMS) that is commonly used for storing and managing structured data. It is one of the most popular and widely adopted databases in the world, known for its reliability, performance, and ease of use.

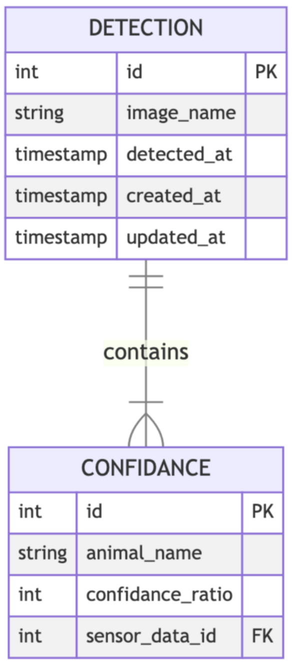


Diagram: Structure of the detected on MySQL

9. Containerization

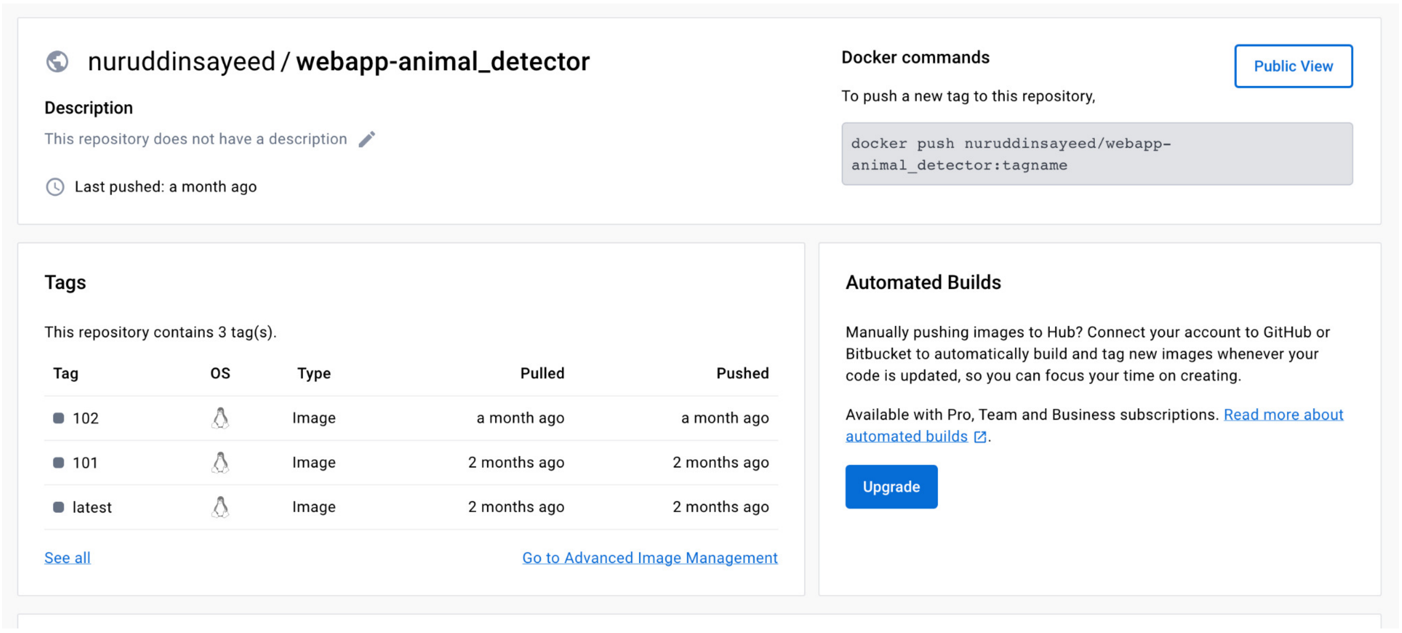
We have used Docker to containerize our apps. To be specific we just had to containerize our WebApp and used already containerized images from Docker-Hub.

9.1 Creating WebAPP docker Image:

As we are going to deploy our app on ARM-based hardware we had to create an image that is compatible with an ARM-based processor. Then we uploaded our docker image to the docker-hub as we will need it to be easy to deploy containerized to K3S from the docker-hub.
We have multiple versions of the image and it is recommended to use only the latest version. (In our case we have used version 102)

To run this image on your local directory just run the following command: (it is expected that you have MySQL with the required database name and MinIO running on your system)

```
docker pull nuruddinsayeed/webapp-animal_detector
```


nuruddinsayeed / **webapp-animal_detector**

Description

This repository does not have a description

Last pushed: a month ago

Docker commands

To push a new tag to this repository,

```
docker push nuruddinsayeed/webapp-
animal_detector:tagname
```

Tags

This repository contains 3 tag(s).

Tag	OS	Type	Pulled	Pushed
102		Image	a month ago	a month ago
101		Image	2 months ago	2 months ago
latest		Image	2 months ago	2 months ago

See all

Go to Advanced Image Management

Automated Builds

Manually pushing images to Hub? Connect your account to GitHub or Bitbucket to automatically build and tag new images whenever your code is updated, so you can focus your time on creating.

Available with Pro, Team and Business subscriptions. Read more about automated builds.

Figure: Docker image

10. Deployment on K3S:

To deploy all the apps on our K3S we have written individual scripts in YML format. We have followed the following sequence for the deployment:

10.1 Create Secrets:

Our every app uses some secrets like passwords or keys and we have defined it on the k3s cluster by setting the secrets from kubectl-secrert.yml file.

```
kubectl apply -f kubectl-secrert.yml
```

10.2 Database deployment:

We had to do something special to deploy database images on k3s. For the use case, we needed something that allows users to request and use storage resources from a storage provider in a portable and platform-independent manner. That's where PVC (PersistentVolumeClaims) and PV helped.

PersistentVolumeClaims and PersistentVolumes play a crucial role in managing and accessing persistent storage in Kubernetes. They provide a portable way for applications to request and use storage resources without being tightly coupled to specific storage providers or infrastructure details. By abstracting storage management, PVCs and PV contribute to the flexibility, portability, and scalability of applications deployed in Kubernetes clusters.

So we have used PVC for MinIO and PV for the MySQL. So We have deployed MinIO and MySQL by following:

```
kubectl apply -f minio-config.yml
kubectl apply -f minio-storage.yml
kubectl apply -f minio.yml

kubectl apply -f mysql-config.yml
kubectl apply -f mysql-storage.yml
kubectl apply -f mysql.yml
```

After that we also hade to go inside of the MySQL image to create a database named “detection_data” as our webapp requires a database with that name:

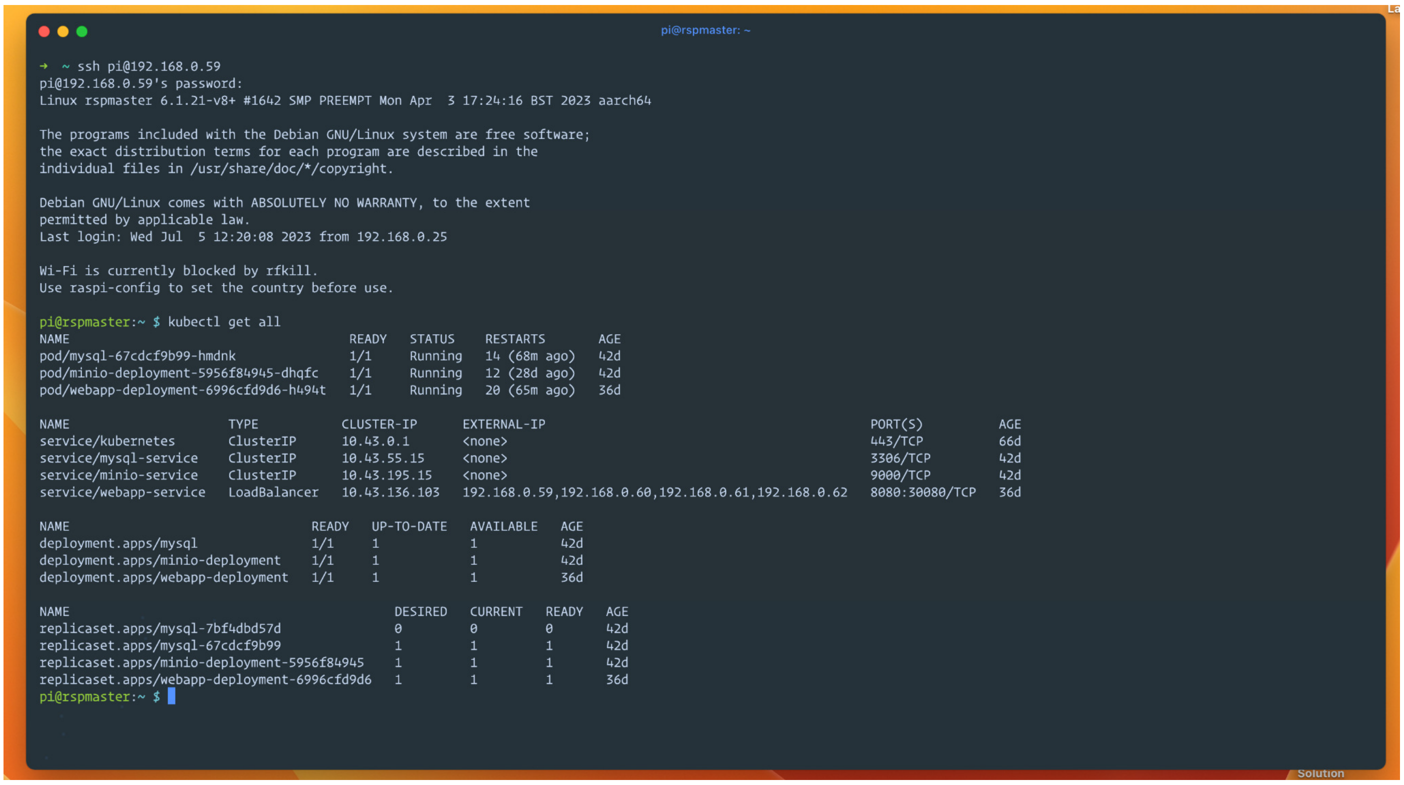
To access Mysql On K3S

```
kubectl exec --stdin --tty mysql-67cdcf9b99-hmdnk -- /bin/bash
```

10.3 WebApp deployment

WebApp is deployed by running the following command:

```
kubectl apply -f webapp.yml
```



```
pi@rspmaster: ~

➜  ~ ssh pi@192.168.0.59
pi@192.168.0.59's password:
Linux rspmaster 6.1.21-v8+ #1642 SMP PREEMPT Mon Apr  3 17:24:16 BST 2023 aarch64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Wed Jul  5 12:20:08 2023 from 192.168.0.25

Wi-Fi is currently blocked by rfkill.
Use raspi-config to set the country before use.

pi@rspmaster:~ $ kubectl get all
NAME                                     READY   STATUS    RESTARTS      AGE
pod/mysql-67cdcf9b99-hmdnk               1/1     Running   14 (68m ago)   42d
pod/minio-deployment-5956f84945-dhqfc    1/1     Running   12 (28d ago)   42d
pod/webapp-deployment-6996cfd9d6-h494t   1/1     Running   20 (65m ago)   36d

NAME                     TYPE           CLUSTER-IP      EXTERNAL-IP                                                   PORT(S)          AGE
service/kubernetes       ClusterIP      10.43.0.1       <none>                                                        443/TCP          66d
service/mysql-service    ClusterIP      10.43.55.15     <none>                                                        3306/TCP         42d
service/minio-service    ClusterIP      10.43.195.15    <none>                                                        9000/TCP         42d
service/webapp-service   LoadBalancer   10.43.136.103   192.168.0.59,192.168.0.60,192.168.0.61,192.168.0.62           8080:30080/TCP   36d

NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/mysql               1/1     1            1           42d
deployment.apps/minio-deployment    1/1     1            1           42d
deployment.apps/webapp-deployment   1/1     1            1           36d

NAME                                           DESIRED   CURRENT   READY   AGE
replicaset.apps/mysql-7bf4dbd57d               0         0         0       42d
replicaset.apps/mysql-67cdcf9b99               1         1         1       42d
replicaset.apps/minio-deployment-5956f84945    1         1         1       42d
replicaset.apps/webapp-deployment-6996cfd9d6   1         1         1       36d
pi@rspmaster:~ $
```

Figure: All deployed apps on K3S Cluster

11. Project links:

GitHub: https://github.com/nuruddinsayeed/CloudApp101
Docker: https://hub.docker.com/r/nuruddinsayeed/webapp-animal_detector

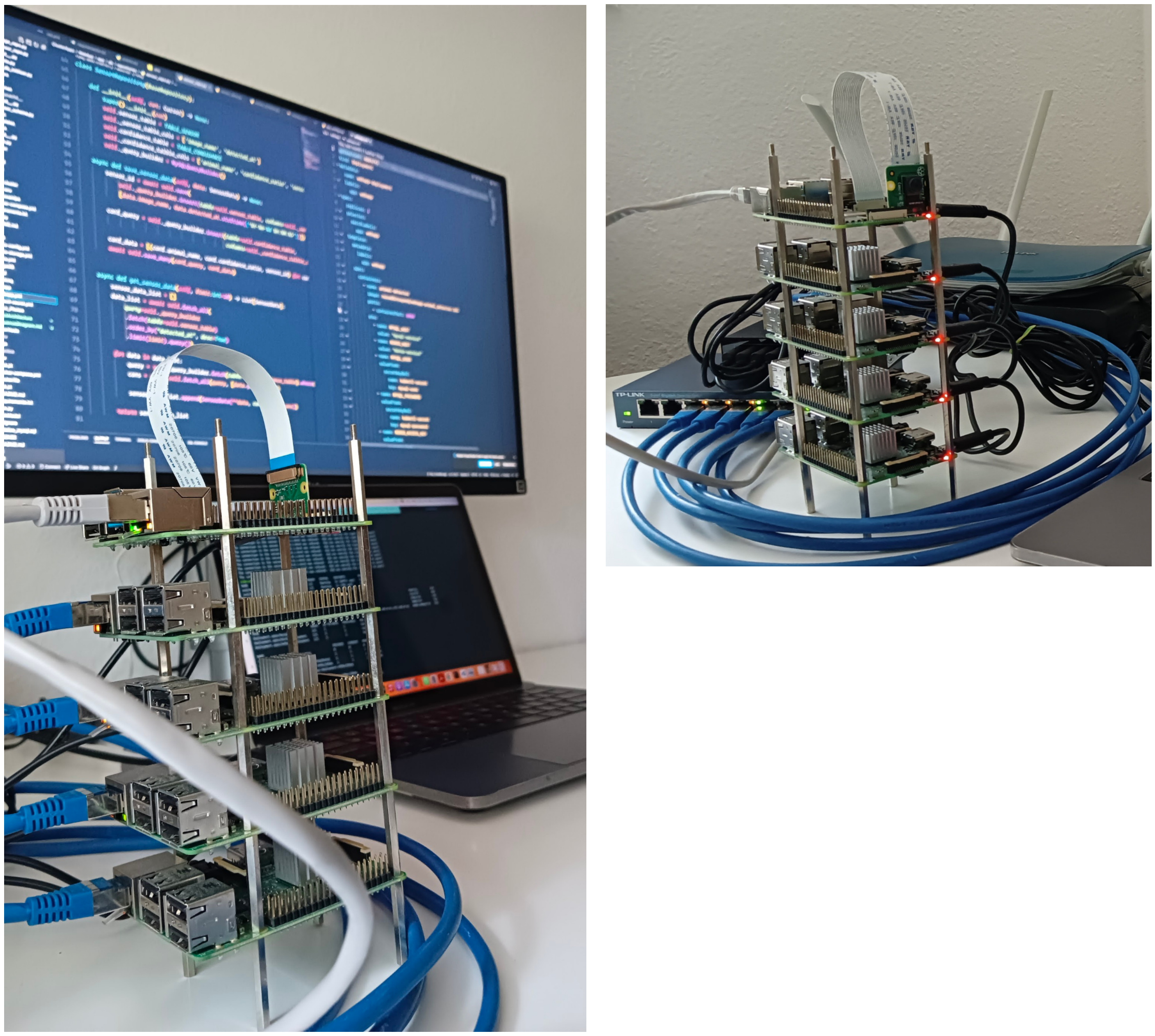


Figure: Project Hardware Setup

12. Bibliography

1. Christian Baun. *Cloud Computing (SS2023)*. URL:https://www.christianbaun.de/CGC23/index.html [Access: 09. 06. 2023]
2. *Raspberry Pi 4*: URL: https://www.raspberrypi.com/documentation/computers/os.html [Access: 25. 05. 2023].
3. *K3s - Lightweight Kubernetes*. URL: https://docs.k3s.io/installation [Access: 28. 05. 2023].
4. *Raspberry Pi OS.* URL: https://www.raspberrypi.com/software/ [Access: 25. 05. 2023].
5. Comprehensive Guide to Ultralytics YOLOv5. URL: https://docs.ultralytics.com/yolov5/ [Access: 09. 06. 2023].
6. *YOLOV5.* URL: https://pytorch.org/hub/ultralytics_yolov5/ [Access: 09. 06. 2023].
7. MinIO Object Storage for Kubernetes.URL:https://min.io/docs/minio/kubernetes/upstream/index.html [Access: 12. 06. 2023].
8. Docker.URL: https://docs.docker.com/ [Access: 25. 05. 2023].