RAT DETECTION USING RASPBERRY PI

Group-4

Rahul Bhowmik Shuvo (1387395)
Arash Abdollahi Kakroodi (1359459)
Mohammad Aftabudduza(1393208)
Husain Ahmad Jahid (1293793)

Supervised By

Prof. Dr. Christian Baun,
Frankfurt University of Applied Sciences

FRANKFURT UNIVERSITY OF APPLIED SCIENCES

Content

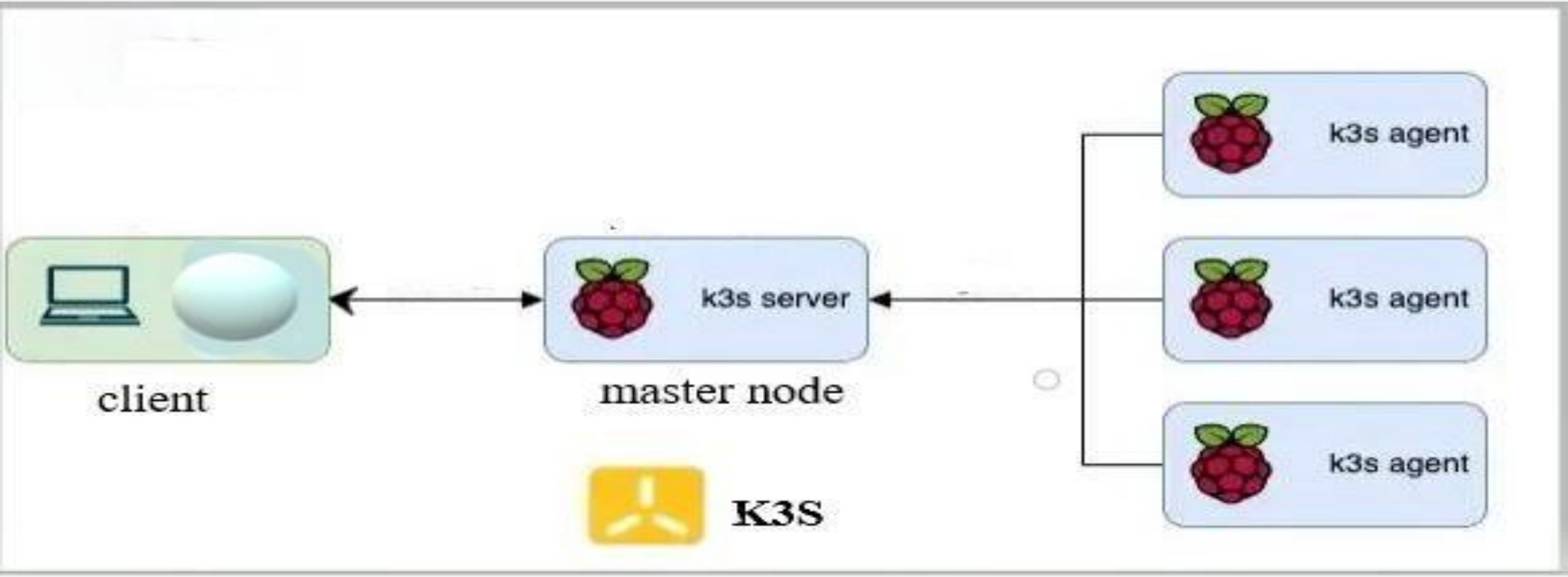
- Introduction
- System Architecture
- K3S Kubernetes Cluster
- Sensor node
- MQTT Broker and Logger
- Model training
- Backend and Frontend UI
- Demonstration

Introduction

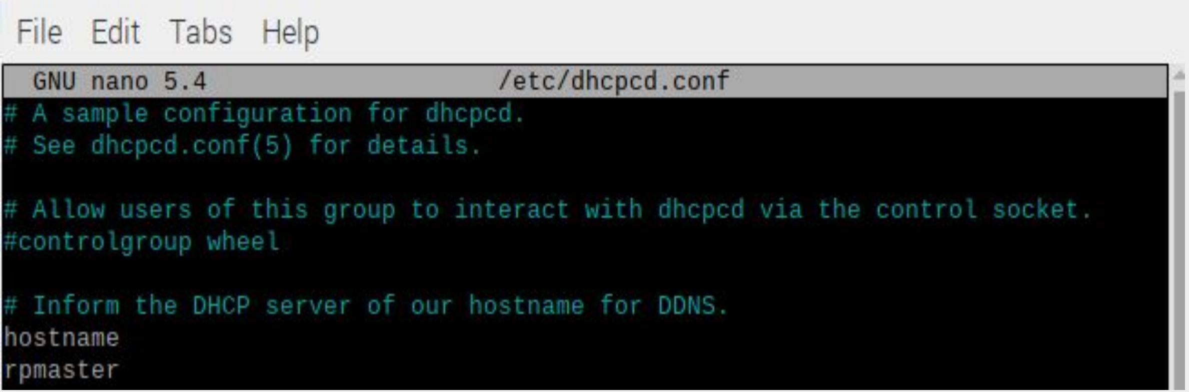
Pests, particularly rats, seem to be a major problem in people's daily life. This paper presents a system for detecting rats using Raspberry Pi.

This system proposes **Raspberry Pi** for model implementation, **Cameras** for capturing images, and **YOLO** v5s for testing and training of the objections detection model.

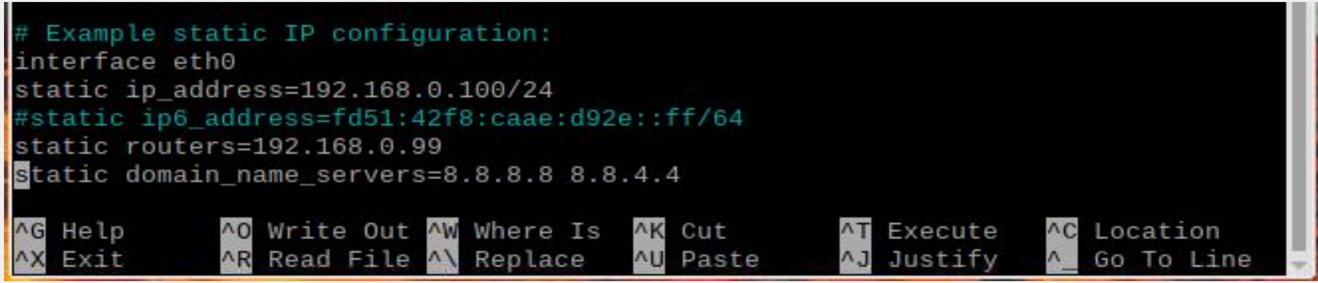
System Architecture



System Architecture



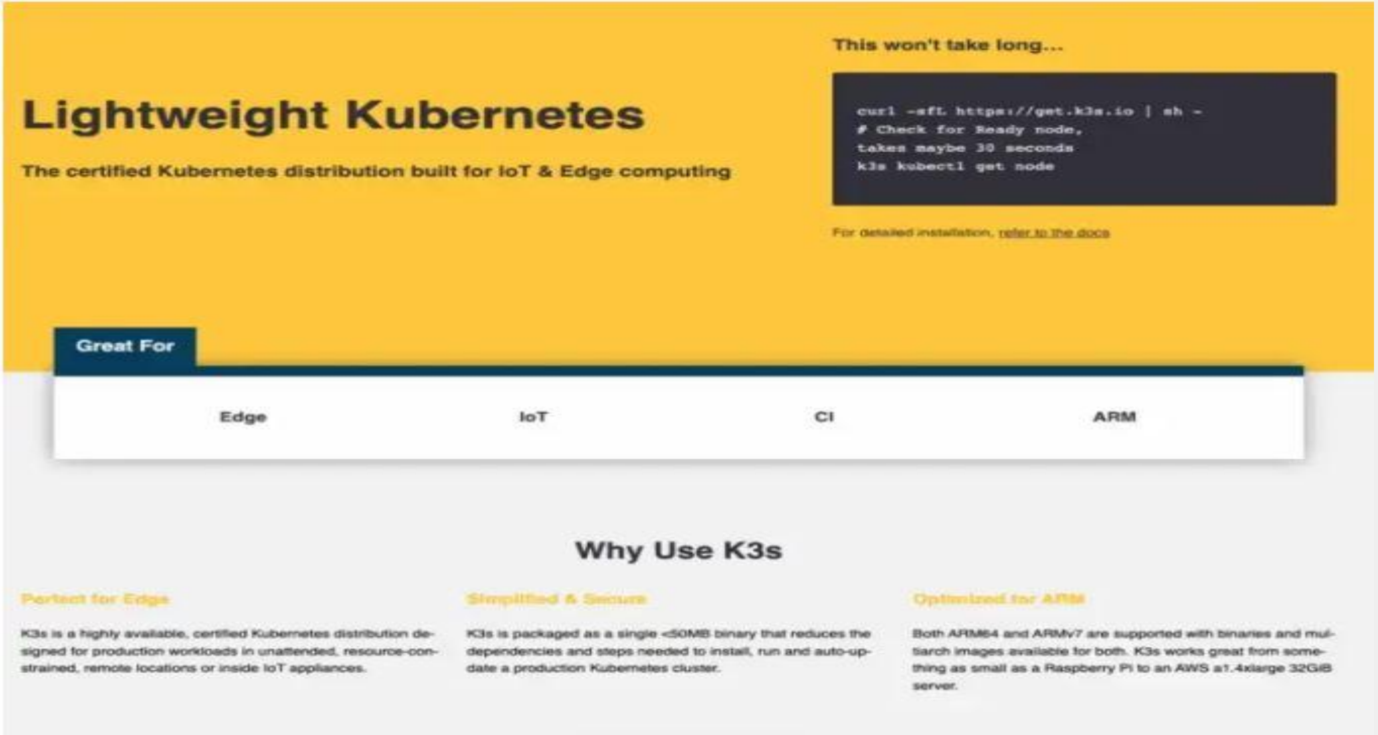
```
  GNU nano 5.4                   /etc/dhcpcd.conf
# A sample configuration for dhcpcd.
# See dhcpcd.conf(5) for details.

# Allow users of this group to interact with dhcpcd via the control socket.
#controlgroup wheel

# Inform the DHCP server of our hostname for DDNS.
hostname
rpmaster
```



```
# Example static IP configuration:
interface eth0
static ip_address=192.168.0.100/24
#static ip6_address=fd51:42f8:caae:d92e::ff/64
static routers=192.168.0.99
static domain_name_servers=8.8.8.8 8.8.4.4
```

K3S Kubernetes Cluster

For the Raspberry pi cluster, a lightweight Kubernetes distribution k3S is used.



Lightweight Kubernetes

The certified Kubernetes distribution built for IoT & Edge computing

This won't take long...

```
curl -sfL https://get.k3s.io | sh -
# Check for Ready node,
takes maybe 30 seconds
k3s kubectl get node
```

For detailed installation, refer to the docs

Great For

Edge IoT CI ARM

Why Use K3s

Perfect for Edge

K3s is a highly available, certified Kubernetes distribution designed for production workloads in unattended, resource-constrained, remote locations or inside IoT appliances.

Simplified & Secure

K3s is packaged as a single <50MB binary that reduces the dependencies and steps needed to install, run and auto-update a production Kubernetes cluster.

Optimized for ARM

Both ARM64 and ARMv7 are supported with binaries and multiarch images available for both. K3s works great from something as small as a Raspberry Pi to an AWS a1.4xlarge 32GiB server.

K3S Kubernetes Cluster

To install K3S the following command must be executed on the master node:

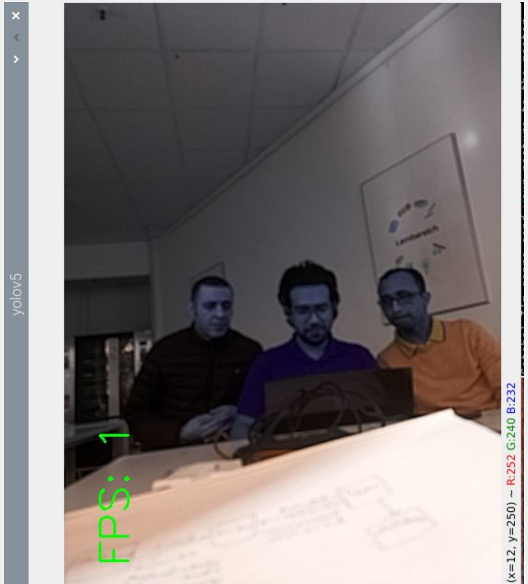
```
curl sfL https://get.k3s.io | K3S_KUBECONFIG_MODE="644" sh -s -
```

With this token, k3S can be installed on worker nodes by the following command:

```
curl sfL https://get.k3s.io |K3S_TOKEN="<TOKEN>"
K3S_URL="https://<master_node_ip>:6443" sh -
```

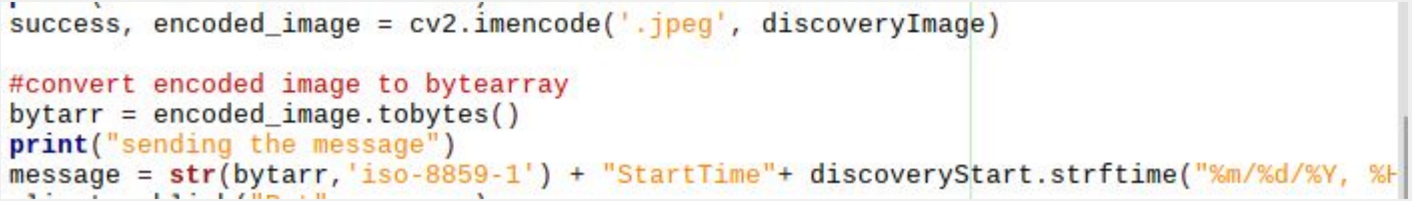
Sensor Node

- **Opencv library to work with the camera**
    - Picamera is not supported for 64 bits os
    - Picamera v2 needs libcamera: Very slow to build the docker image
    - The problem with video0 file : *bcm2835-v4l2*
    - For camera quality



Sensor Node

- **On Detection : publishes the data**
  - encode the image as jpg file format
  - Convert the binary to byte array
  - Encode the byte array with iso_8859_1 standard



```
success, encoded_image = cv2.imencode('.jpeg', discoveryImage)

#convert encoded image to bytearray
bytarr = encoded_image.tobytes()
print("sending the message")
message = str(bytarr,'iso-8859-1') + "StartTime"+ discoveryStart.strftime("%m/%d/%Y, %H
```

Sensor Node

Issues in docker Image:

- Problems with picamera and picamera2 and libcamera
- Extra libraries for opencv :
  - libsm6
  - libxext6
  - Libxrender

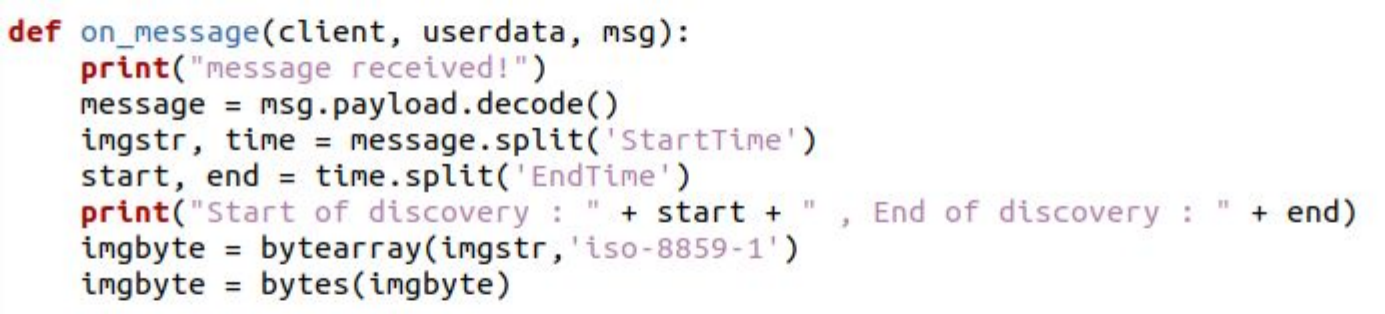
Docker installed on Raspberry pi4:

- Running the sensor node image
- Creating docker images for the cluster

Logger

Tasks:

- Subscribes to MQTT broker
- Handles the new message with on_message method
- Inserts the new discovery into database



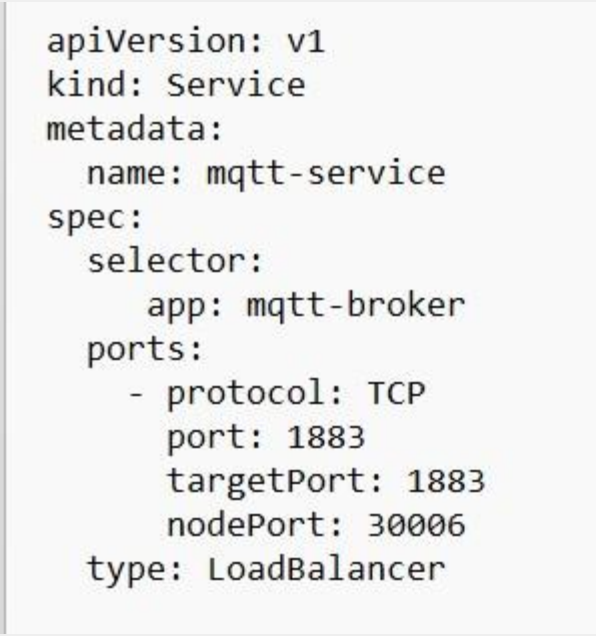
```
def on_message(client, userdata, msg):
    print("message received!")
    message = msg.payload.decode()
    imgstr, time = message.split('StartTime')
    start, end = time.split('EndTime')
    print("Start of discovery : " + start + " , End of discovery : " + end)
    imgbyte = bytearray(imgstr,'iso-8859-1')
    imgbyte = bytes(imgbyte)
```

Logger

- Docker image creation
- Deployment on the cluster with two container ports:
  - 3306 to connect to Mariadb
  - 1883 to connect to MQTT broker
- Services:
  - NodePort on 1883 for MQTT broker
  - NodePort on 3306 for Mariadb

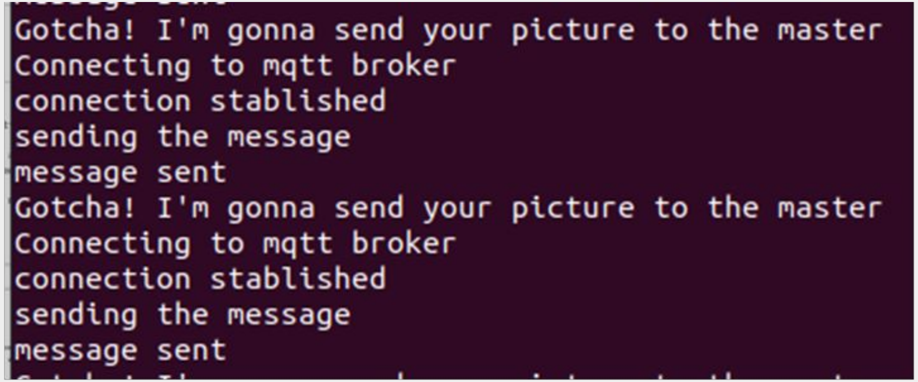
Deployment of MQTT Broker on k3s

- eclipse Mosquitto version 1.6.15
- it serves all the nodes in the network
- MQTT service is defined from the type LoadBalancer.
- MQTT uses default port 1883.

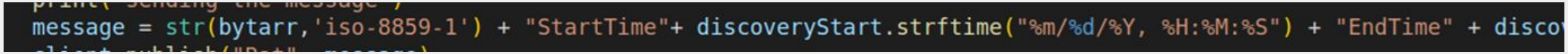


```
apiVersion: v1
kind: Service
metadata:
  name: mqtt-service
spec:
  selector:
     app: mqtt-broker
  ports:
    - protocol: TCP
      port: 1883
      targetPort: 1883
      nodePort: 30006
  type: LoadBalancer
```

MQTT Broker



```
Gotcha! I'm gonna send your picture to the master
Connecting to mqtt broker
connection stablished
sending the message
message sent
Gotcha! I'm gonna send your picture to the master
Connecting to mqtt broker
connection stablished
sending the message
message sent
```

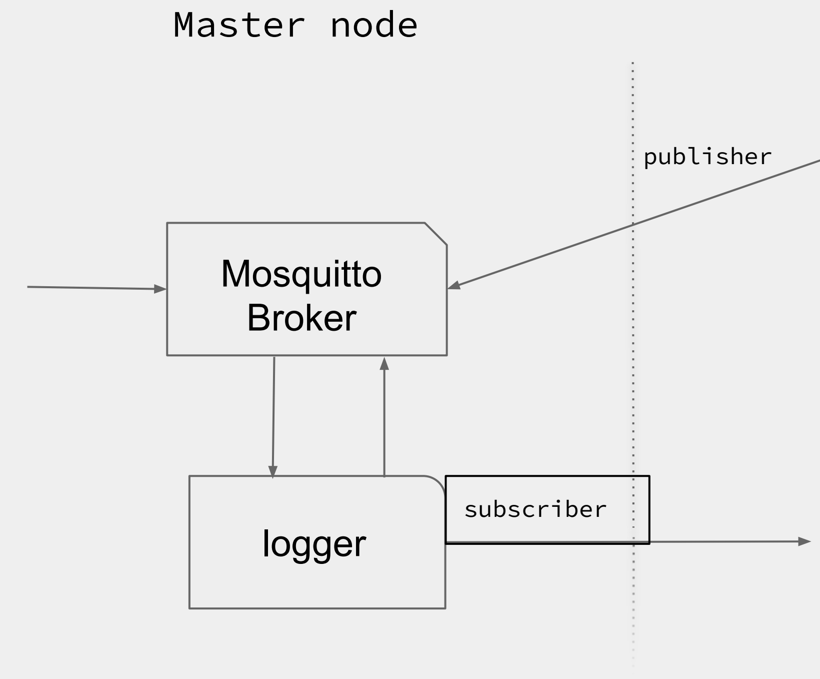


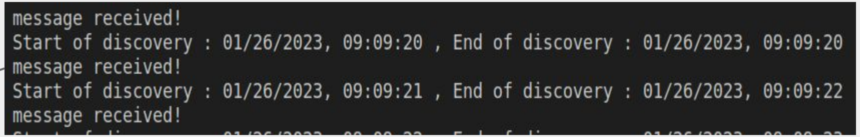
```
message = str(bytarr,'iso-8859-1') + "StartTime"+ discoveryStart.strftime("%m/%d/%Y, %H:%M:%S") + "EndTime" + disco
```

Publisher and Subscriber

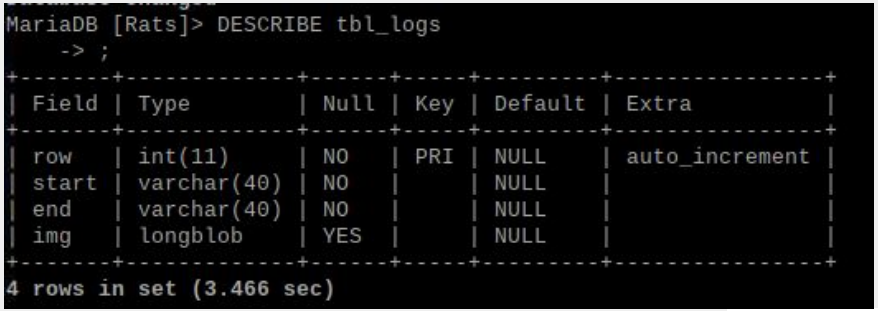
- publisher - the docker image running on the sensor node
- subscriber - the logger application on the cluster
- publisher side: loop_start(), loop_forever and loop_stop() methods are used.
- publish() method which is not a block, is handled successfully
- Mosquitto uses to encode messages which is iso-8859-1. This standard handles the special characters in a binary file that are not allowed to be in a string variable

MQTT





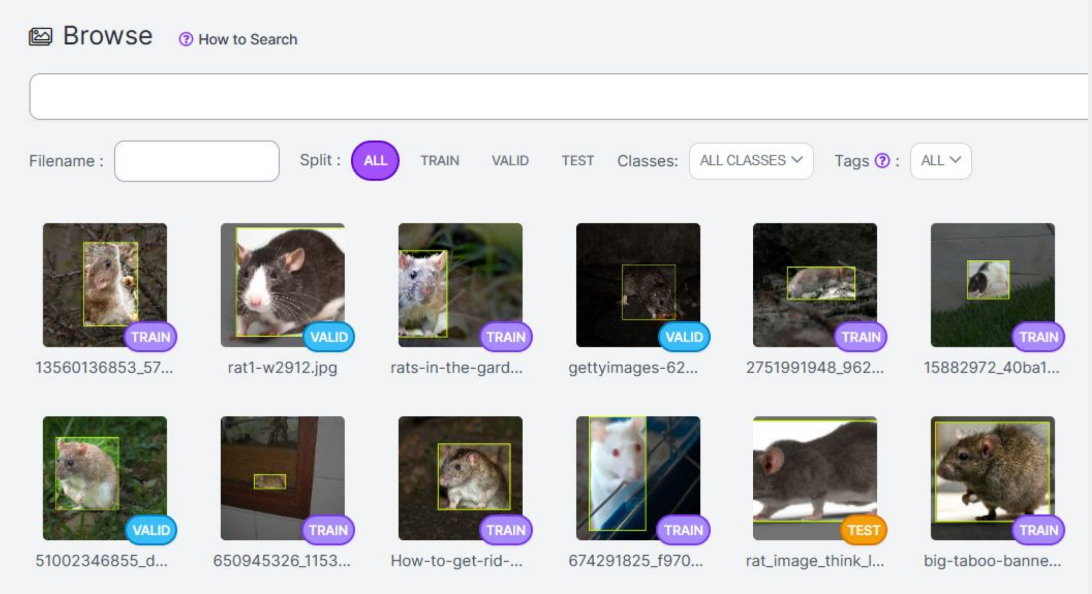
```
message received!
Start of discovery : 01/26/2023, 09:09:20 , End of discovery : 01/26/2023, 09:09:20
message received!
Start of discovery : 01/26/2023, 09:09:21 , End of discovery : 01/26/2023, 09:09:22
message received!
```



```
MariaDB [Rats]> DESCRIBE tbl_logs
    -> ;
+-------+-------------+------+-----+---------+----------------+
| Field | Type        | Null | Key | Default | Extra          |
+-------+-------------+------+-----+---------+----------------+
| row   | int(11)     | NO   | PRI | NULL    | auto_increment |
| start | varchar(40) | NO   |     | NULL    |                |
| end   | varchar(40) | NO   |     | NULL    |                |
| img   | longblob    | YES  |     | NULL    |                |
+-------+-------------+------+-----+---------+----------------+
4 rows in set (3.466 sec)
```

Model training

- Using data set https://app.roboflow.com/frauas/rat_detection/3 from group 2
- Train the dataset using YOLO v5s model in google colab
- Getting the best trained model for our detection project



Split : ALL

Classes: ALL CLASSES

Tags : ALL

Model training

Backend

Flask:

- Flask is a micro web framework written in Python that makes it easy to build web applications.
- Flask provides support for integrating HTML, CSS, and JavaScript into your web application and also provides built-in support for serving static files like CSS and JavaScript.

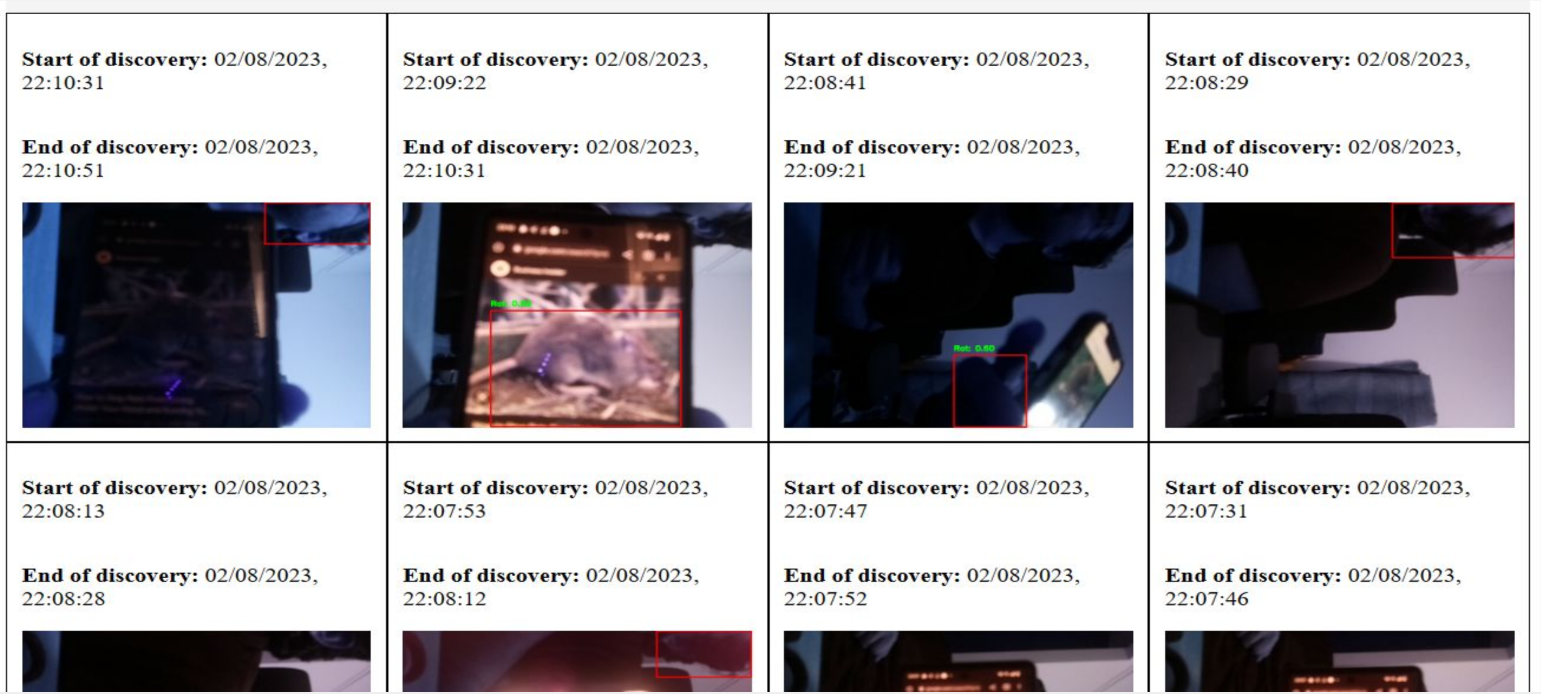
FrontEnd

HTML, CSS, and JavaScript:

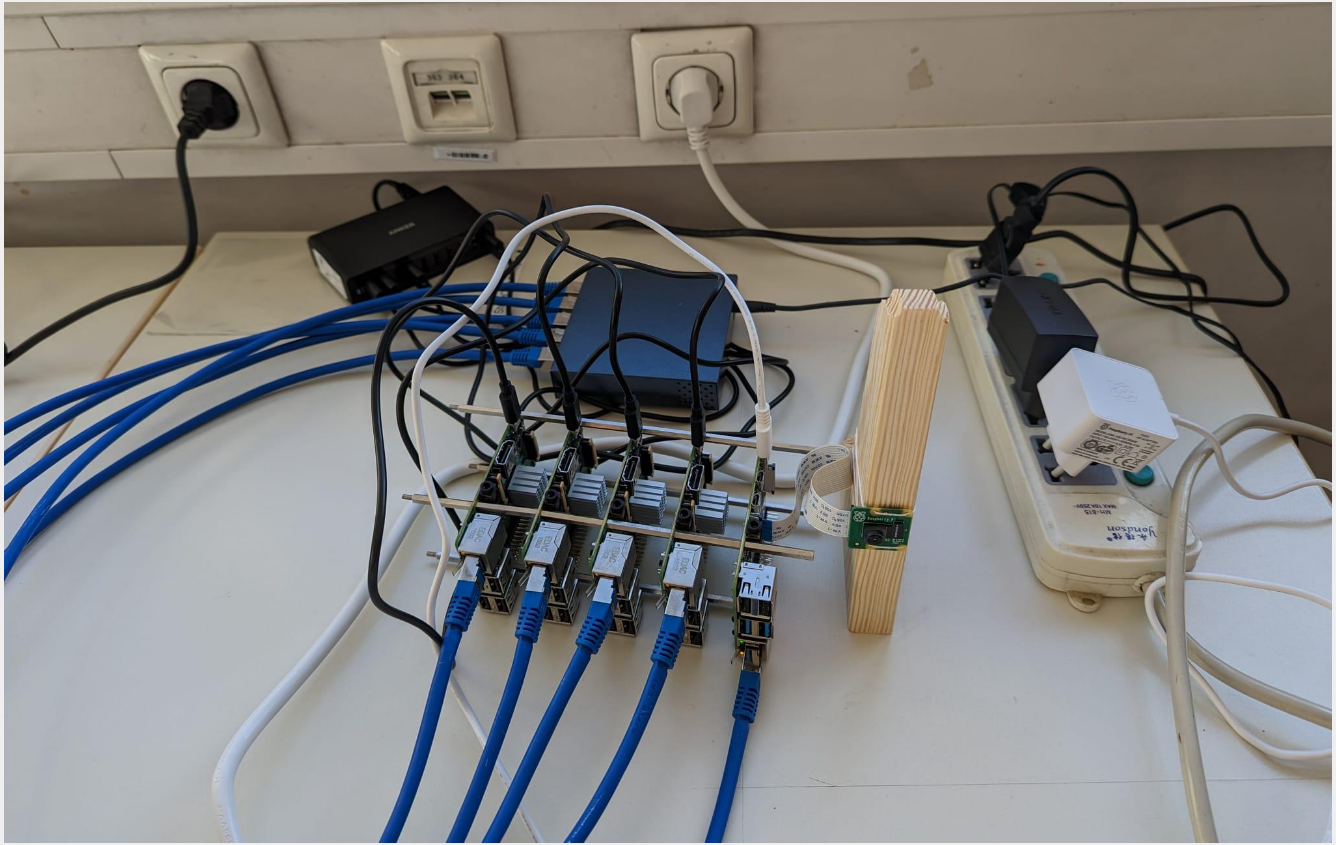
- HTML (Hypertext Markup Language) is used to structure the content of web pages.
- CSS (Cascading Style Sheets) is used to define the appearance and layout of web pages.
- JavaScript is a client-side scripting language that can be used to add interactivity and dynamic behavior to web pages.

FRANKFURT UNIVERSITY OF APPLIED SCIENCES

FrontEnd



Start of discovery: 02/08/2023, 22:10:31 **End of discovery:** 02/08/2023, 22:10:51	**Start of discovery:** 02/08/2023, 22:09:22 **End of discovery:** 02/08/2023, 22:10:31	**Start of discovery:** 02/08/2023, 22:08:41 **End of discovery:** 02/08/2023, 22:09:21	**Start of discovery:** 02/08/2023, 22:08:29 **End of discovery:** 02/08/2023, 22:08:40
Start of discovery: 02/08/2023, 22:08:13 **End of discovery:** 02/08/2023, 22:08:28	**Start of discovery:** 02/08/2023, 22:07:53 **End of discovery:** 02/08/2023, 22:08:12	**Start of discovery:** 02/08/2023, 22:07:47 **End of discovery:** 02/08/2023, 22:07:52	**Start of discovery:** 02/08/2023, 22:07:31 **End of discovery:** 02/08/2023, 22:07:46



https://github.com/rahulshuvo/rat-detector

Demonstration

Thank you!