# Cloud Computing

Under the guidance of Prof. Dr. Christian Baun
Frankfurt University of Applied Sciences

# Project Report
# &
# Installation Manual of
AI/ML Based Pest Detection System

**Group Members**

Jatinkumar Nakrani (1386383)
Kaival Arvindbhai Akbari (1387256)
Sameer Soni (1392911)
Santosh Ganiger (1420663)
Usman Tariq (1384673)

*Report Prepared by Sameer Soni and Santosh Ganiger*

**INDEX**

## 1. Abstract

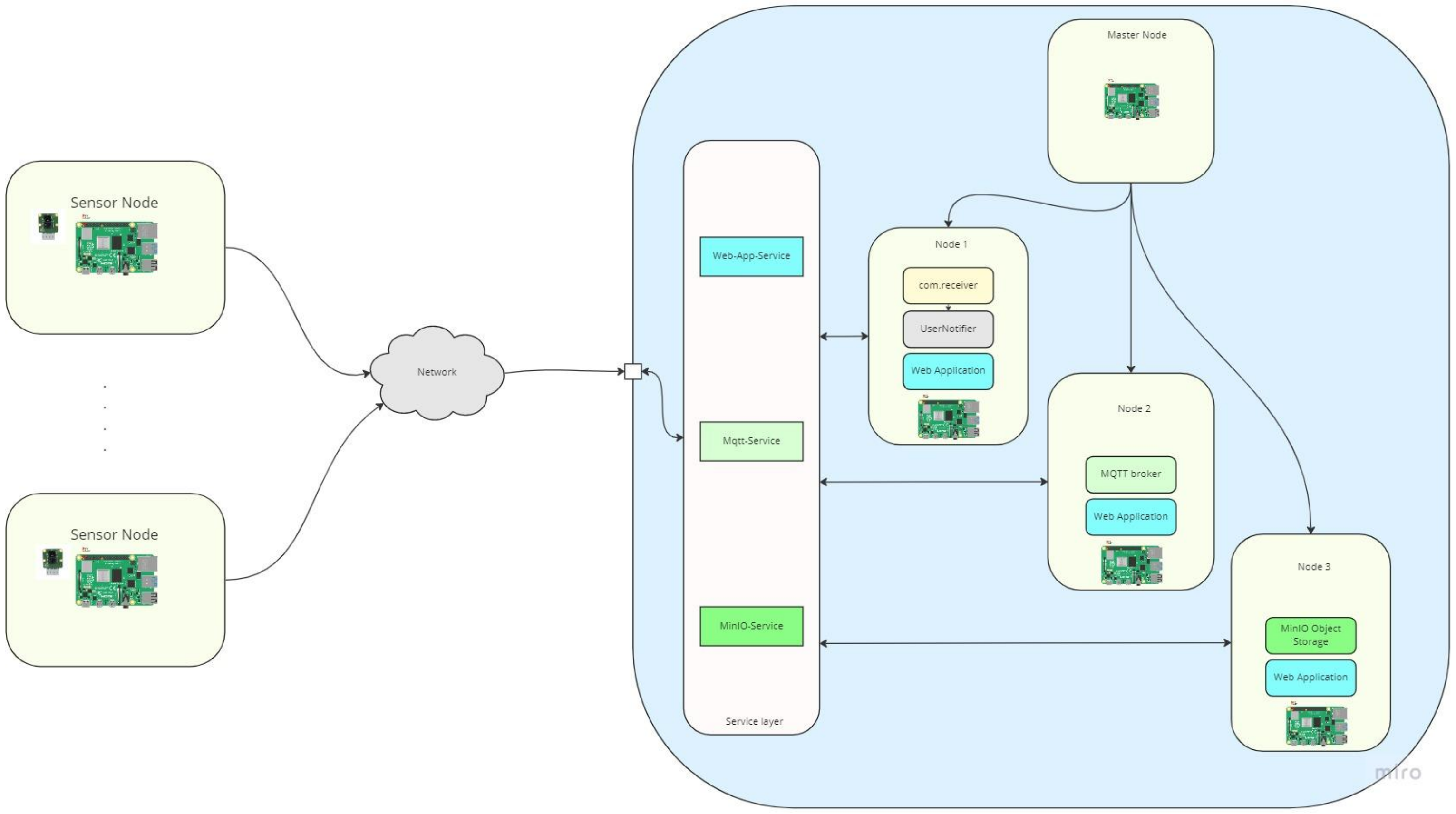
This document explains the architecture, operation, workflow and installation procedure of the solution built for the requirements provided by the user who is interested in detecting 'Rats' in the surroundings. Document also gives brief information about the method and technologies used to achieve the goal. Main purpose of this document is to give the user a step by step guide to install this solution in his own hardware.

## 2. Provided Requirements

1. Need a system to detect rat in the surroundings
2. System should should use ML/AI to detect rat
3. System should perform Edge computing and only send processed data
4. System should use Kubernetes cluster to store, access, view detections
5. System should notify user via telegram on rat detection

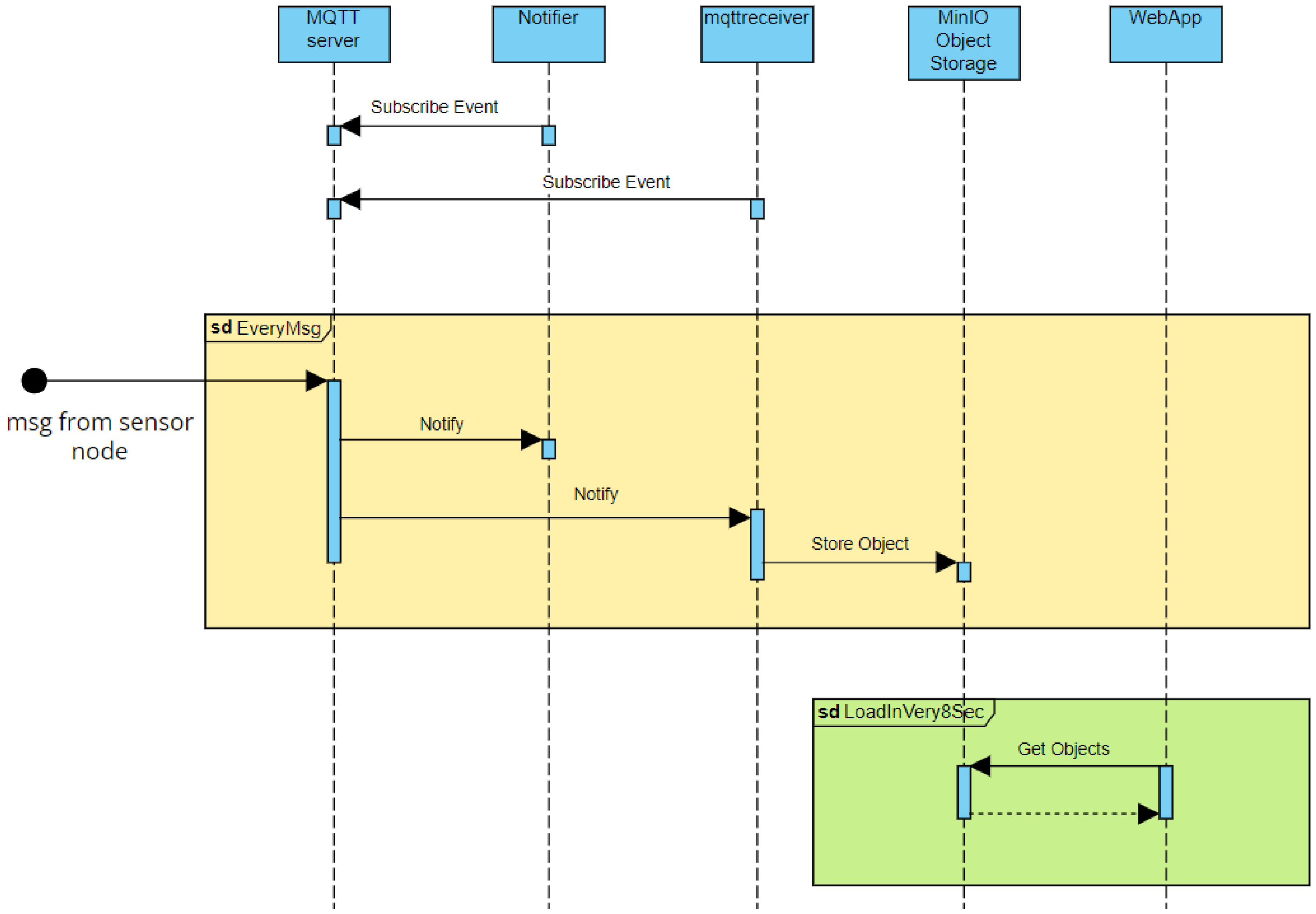
## 3. Solution Architecture

### 3.1. Overall Architecture

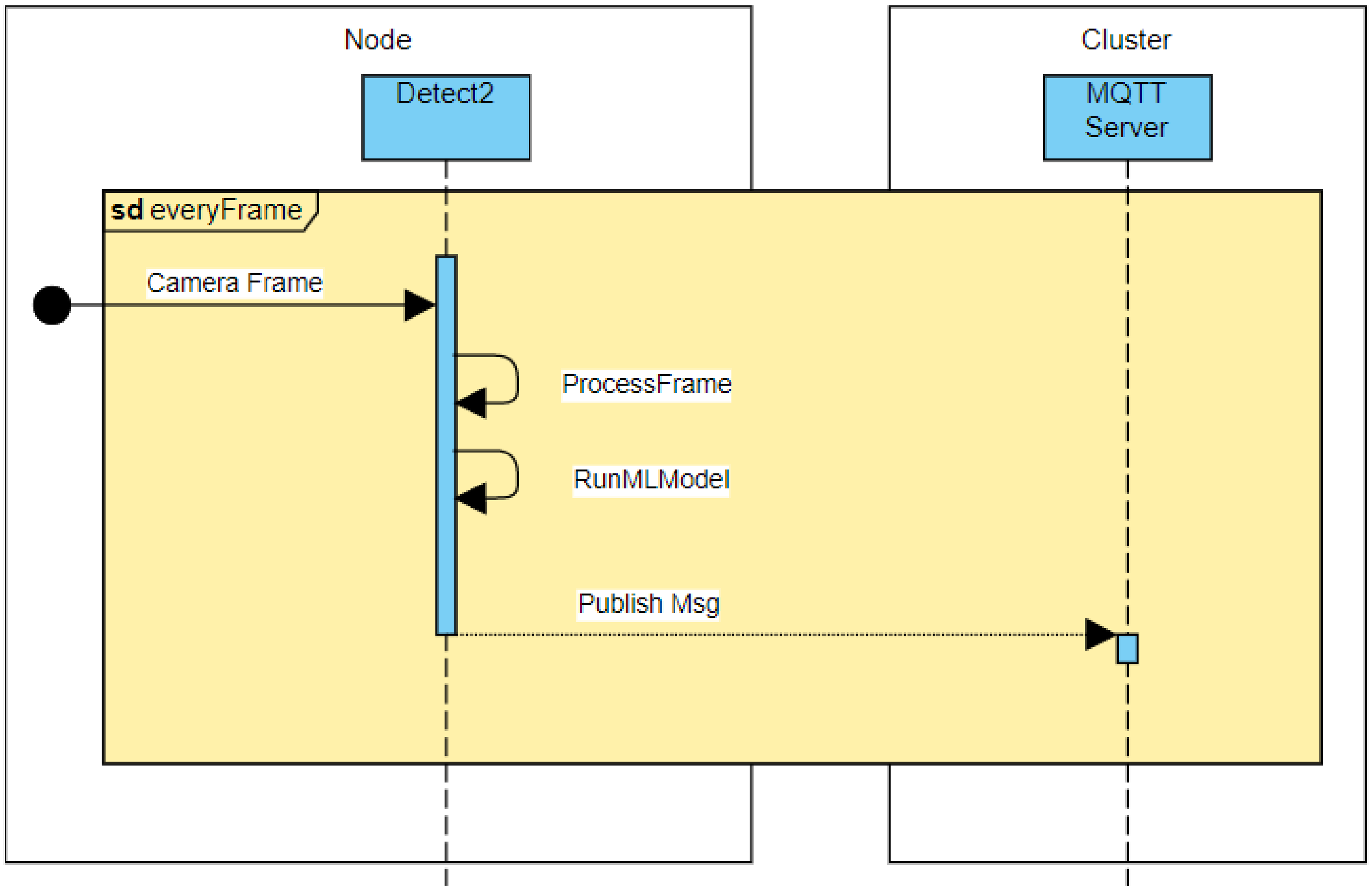


Overall Solution Architecture [Fig 1]

## 3.2. Sequence Diagrams



Cluster [Fig 2]



Node [Fig 3]

## 3.3. Communication API

```
{       "name": "rat",
        "today_datetime": "03-02-2023 00:01:48",
        "match_parent": "0.608",
        "img_content": "/9j/4AAQScndbsjjsdnodsnonsmpcndjksmpamp..........."
}
```

Communication API [Fig 4]

# 4. Tools and technologies

## 4.1. Node

YOLOv5
Python
PyTorch

## 4.2. Cluster

K3s Kubernetes
Docker
MinIO Object Storage
Mosquitto MQTT Server
Python

# 5. Applications

## 5.1. Sensor Node

Sensor node application ‘detect2.py’ is the main application which runs the machine learning model on captured camera frames and detects rats. This application loads already trained ML model weights and feeds it to the YOLOv5 model using PyTorch.

Application is available in the node directory of the project folder which can run an independent application or as a service. Running detect as a service is recommended.

Procedure of deployment is mentioned in the installation section.

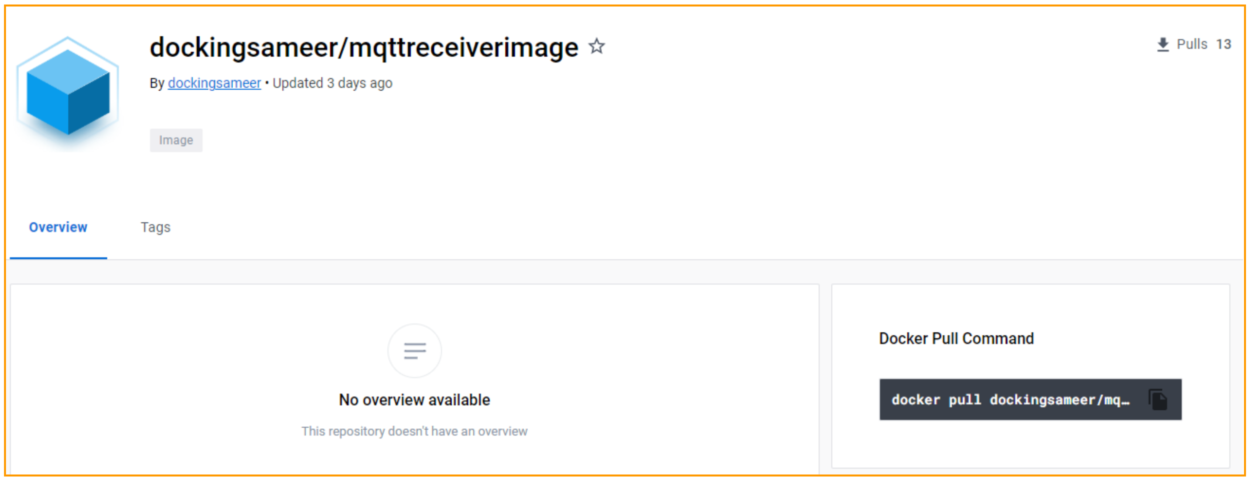
## 5.2. Kubernetes Cluster Applications

### 5.2.1. MQTTReceiver

MQTTReceiver is a custom application built to achieve specific goals for the system. MQTTReceiver app connects to MQTT broker server and subscribes to the topic mentioned below. Whenever this message appears on the MQTT broker server forwards the message to all the subscribers.

Topic : **detection/rat**

Docker image of application can be pulled from the https://hub.docker.com/r/dockingsameer/mqttreceiverimage and can also be built locally by executing script ***buildImage.sh*** in the respective application directory.


dockingsameer/mqttreceiverimage ☆

By dockingsameer • Updated 3 days ago

Image

Pulls 13

No overview available

This repository doesn't have an overview

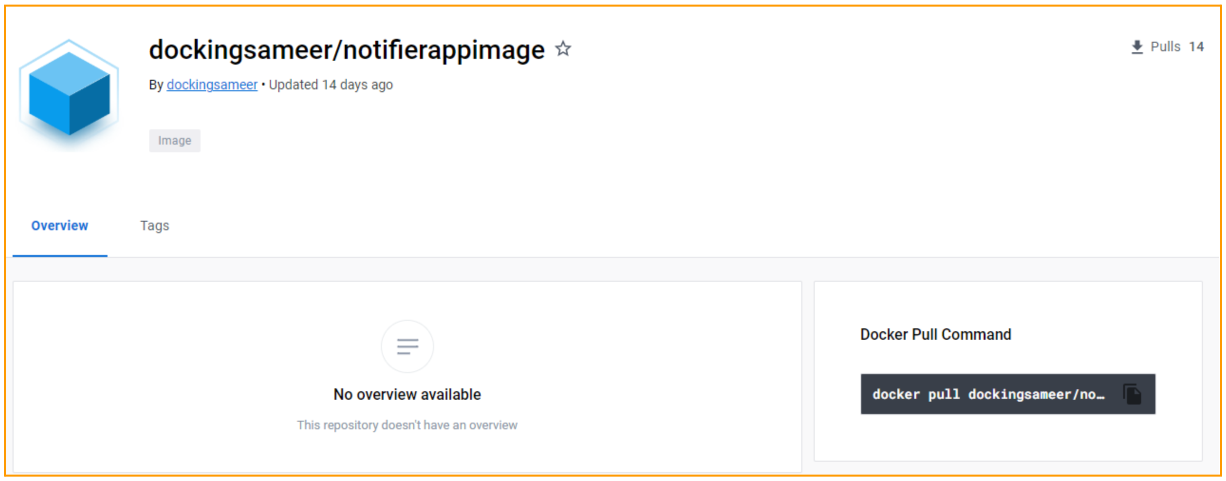
Docker Pull Command

```
docker pull dockingsameer/mq…
```

### 5.2.2. Notifier

Notifier application is a custom application which sends detection pictures directly to Telegram bot. This application connects to MQTT broker server and subscribes to the same topic as MQTTReceiver and whenever a message is received from it sends to the bot.

Docker image of application can be pulled from the https://hub.docker.com/r/dockingsameer/notifierappimage and can also be built locally by executing script ***buildImage.sh*** in the respective application directory.


dockingsameer/notifierappimage ☆

By dockingsameer • Updated 14 days ago

Image

Pulls 14

No overview available

This repository doesn't have an overview

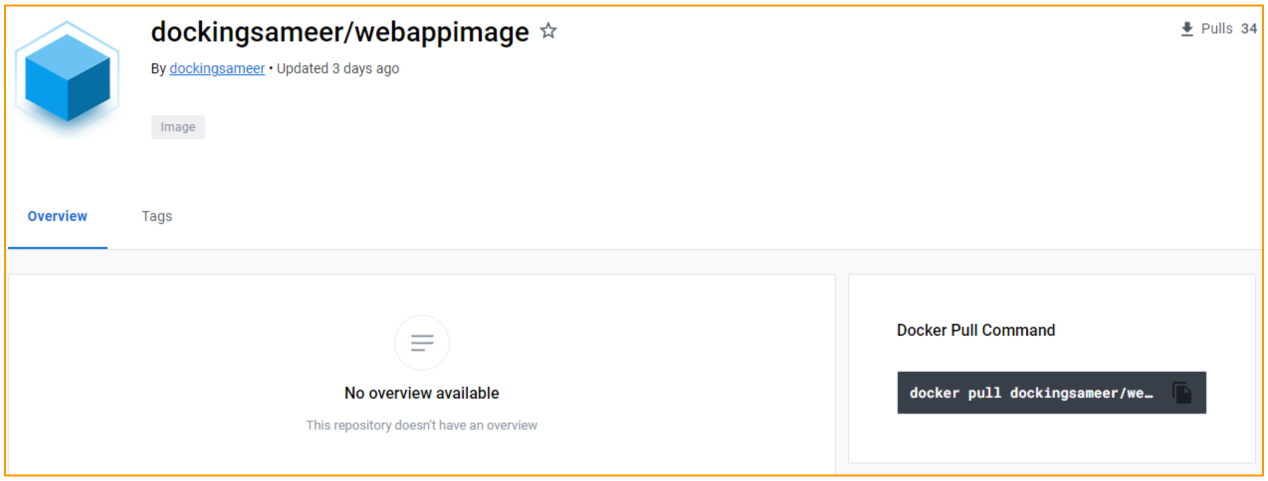
Docker Pull Command

```
docker pull dockingsameer/no…
```

### 5.2.3. WebApp

WebApp is the main custom user interface which shows all the pests (Rat) detected by the system over time. It reads all the objects stored in the database and shows them in the Web browser.

Docker image of application can be pulled from the https://hub.docker.com/r/dockingsameer/webappimage and can also be built locally by executing script ***buildImage.sh*** in the respective application directory.

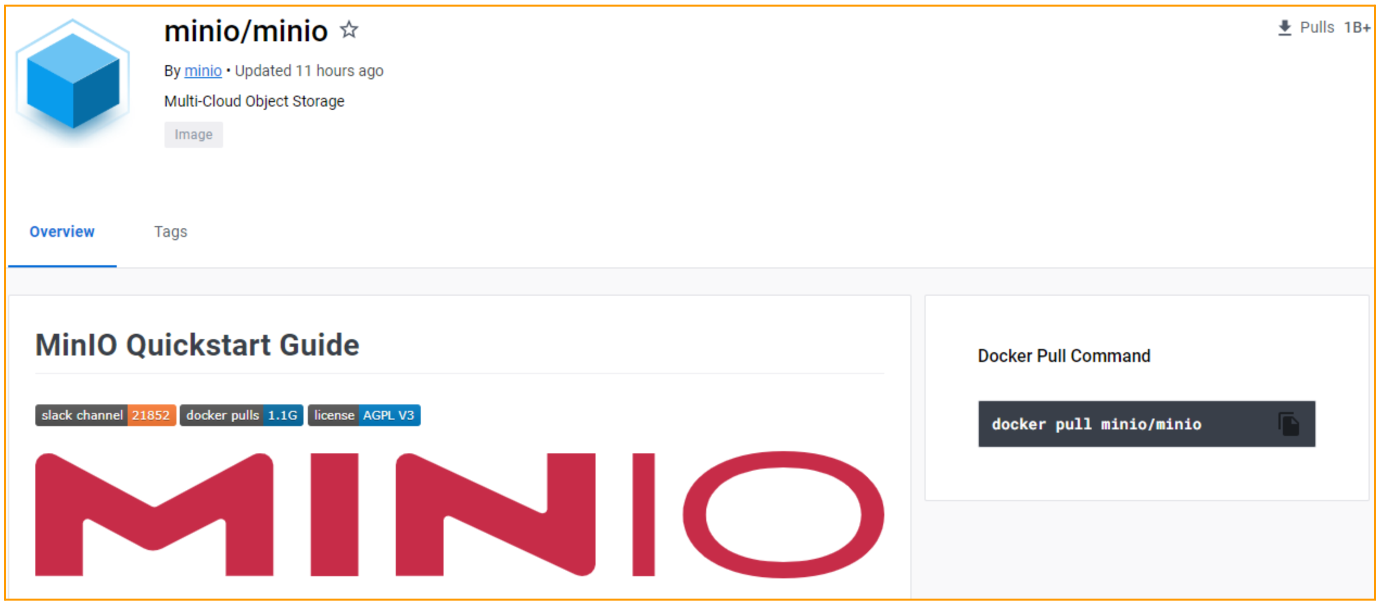

dockingsameer/webappimage ☆
By dockingsameer • Updated 3 days ago
Image
Pulls 34
Overview
Tags
No overview available
This repository doesn't have an overview
Docker Pull Command

```
docker pull dockingsameer/we…
```

### 5.2.4. MinIO Object Storage

This is a third party open source application which is used in the system to store and access objects received from sensors. MinIO is a high performance object storage solution that provides an Amazon Web Services S3-compatible API and supports all core S3 features.[1]

System uses MinIO docker image available on https://hub.docker.com/r/minio/minio and deploys on the kubernetes cluster.



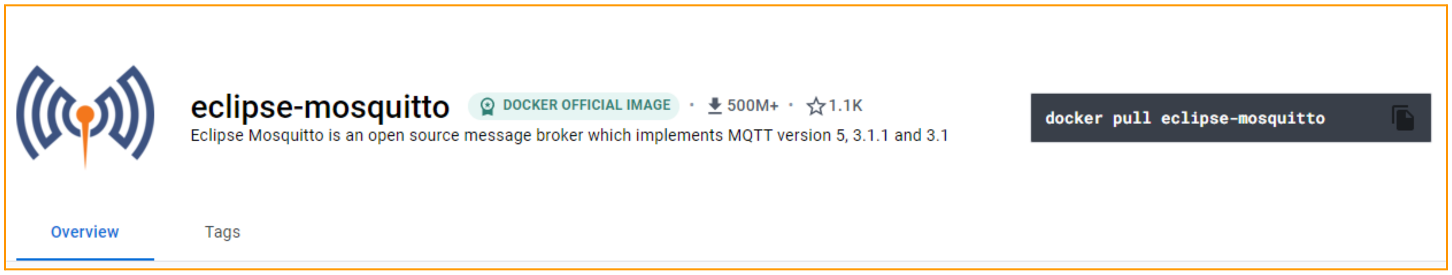
```
docker pull minio/minio
```

### 5.2.5. Mosquitto MQTT Broker

System uses Eclipse Mosquitto message broker which implements MQTT protocol. It provides a lightweight method of carrying out messaging using a publish/subscribe model. [2]

Sensors connect on the MQTT server as publishers and a custom application mqttreceiver also connects to the server as a subscriber.

System uses MinIO docker image available on [https://hub.docker.com/_/eclipse-mosquitto](https://hub.docker.com/_/eclipse-mosquitto) and deploys on the kubernetes cluster.


eclipse-mosquitto

Eclipse Mosquitto is an open source message broker which implements MQTT version 5, 3.1.1 and 3.1

```
docker pull eclipse-mosquitto
```

## 6. Installation Procedure

### 6.1. Hardware Requirements

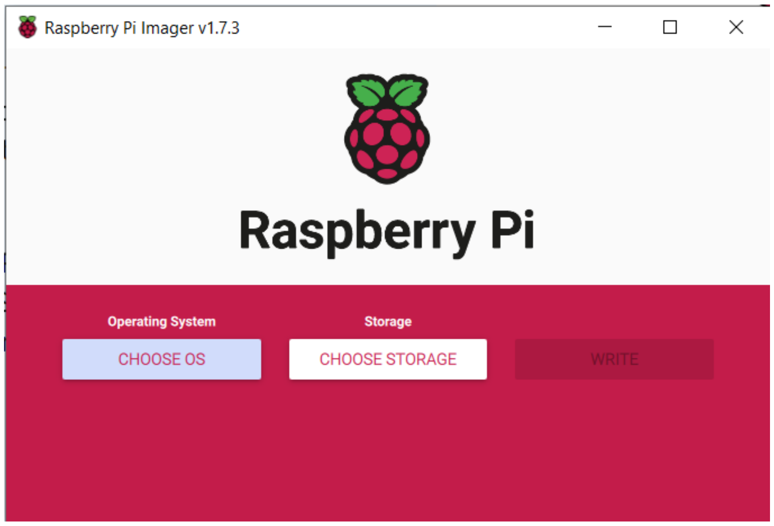
a. At Least 4 units Raspberry Pi v3 or above

b. Pi camera for sensor node

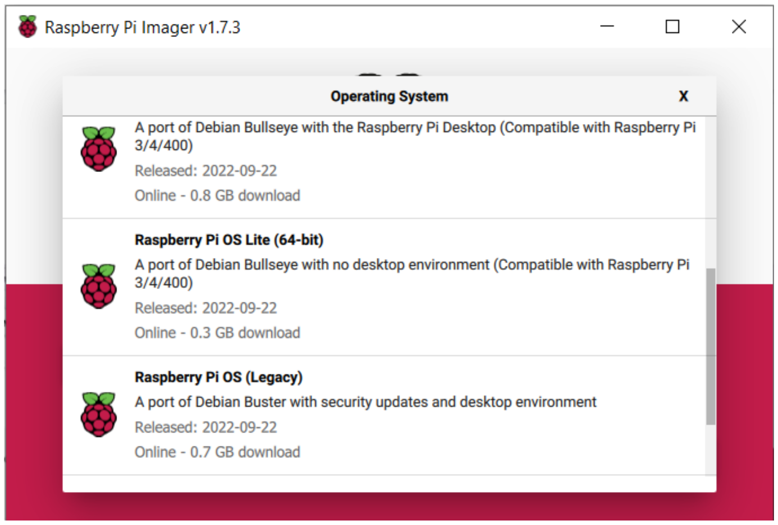
### 6.2. Install operating system

#### 6.2.1. Setup Raspberry Pi headless:

1. Install an operating system on the Raspberry Pi SD Card using Raspberry Pi imager.
2. Select OS "Raspberry Pi OS Lite (64-bit)" in Raspberry Pi imager and select the SD card you want to flash.



Above [Fig 5]; Below [Fig 6]

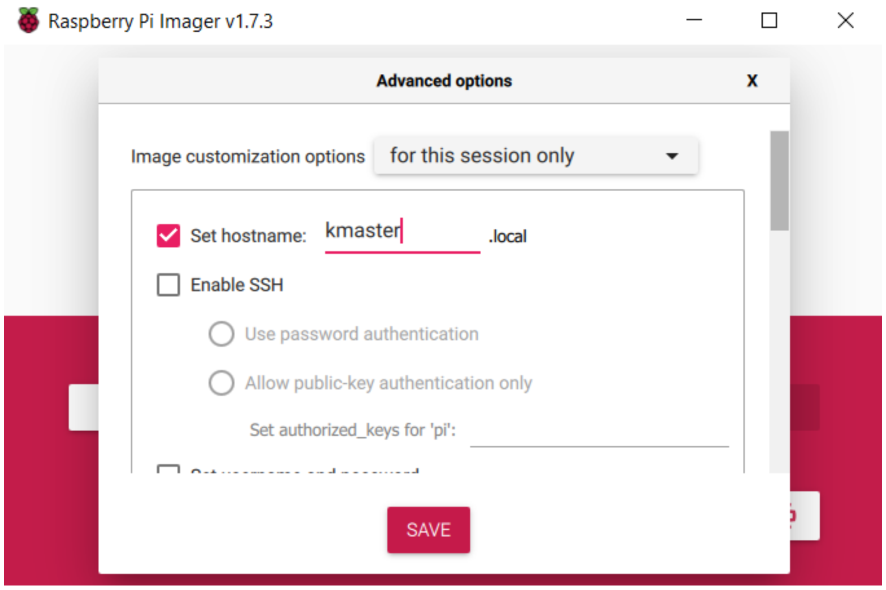


A port of Debian Bullseye with the Raspberry Pi Desktop (Compatible with Raspberry Pi 3/4/400)
Released: 2022-09-22
Online - 0.8 GB download

**Raspberry Pi OS Lite (64-bit)**
A port of Debian Bullseye with no desktop environment (Compatible with Raspberry Pi 3/4/400)
Released: 2022-09-22
Online - 0.3 GB download

**Raspberry Pi OS (Legacy)**
A port of Debian Buster with security updates and desktop environment
Released: 2022-09-22
Online - 0.7 GB download

3. After successful installation, assign a unique hostname for each Raspberry Pi, for example, "kmaster" for the master node and "knode1" for the first worker node. Save and Write in the SD Card.


Raspberry Pi Imager v1.7.3
Advanced options
Image customization options
for this session only
Set hostname:
kmaster
.local
Enable SSH
Use password authentication
Allow public-key authentication only
Set authorized_keys for 'pi':
SAVE

[Fig 7]

4. Similarly, install OS on two more raspberry pi worker nodes knode2 and knode3.

5. To verify their availability, ping them using their local hostname or check through our router login interface.

## 6.3. Installing Kubernetes [3][4]

### 6.3.1. Install Docker on Master Node

```
sudo apt install docker              #Install docker
sudo systemctl start docker          #start docker service
sudo systemctl enable docker         #set enable service
sudo systemctl status docker         #get status of service
```

### 6.3.2. Install k3s on Master Node

```
curl -sfL https://get.k3s.io | sh -s - --docker
sudo systemctl status k3s
sudo kubectl get nodes -o wide
```

Extract the token:

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

### 6.3.3. Install k3s on Worker Nodes

a. Follow steps mentioned in step 6.3.1 to install docker on all the nodes.
b. Replace <master_IP> with **master node ip** address and run below mentioned command -
***curl -sfL http://get.k3s.io | K3S_URL=http://<master_IP>:6443 K3S_TOKEN=<join_token> sh -s - --docker***
c. Run : ***sudo systemctl status k3s-agent*** to check status of the worker agent
d. Do the step *a* to *c* for all the worker nodes.

### 6.3.4. Validate k3s cluster nodes

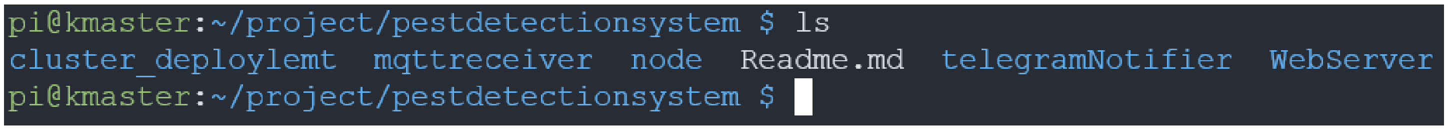
Run : ***sudo kubectl get node -o wide***

## 6.4. Deploying Pods and Services on Worker Nodes

### 6.4.1. Downloading / Copying deployment files

Copy or clone project on the master node and copy 'node' sensor node

*git clone [https://github.com/CloudStation1/pestdetectionsystem.git](https://github.com/CloudStation1/pestdetectionsystem.git)*
*After downloading, directory content will look like this -*



```
pi@kmaster:~/project/pestdetectionsystem $ ls
cluster_deploylemt  mqttreceiver  node  Readme.md  telegramNotifier  WebServer
pi@kmaster:~/project/pestdetectionsystem $
```

### 6.4.2. MinIO Object Storage Deployment

Go to directory : pestdetectionsystem/cluster_deploylemt/minio
Execute:  sh deploy.sh
Alternatively, you can execute below mentioned commands

```
sudo kubectl apply -f minio-storage.yaml        # creates persistent storage volume
sudo kubectl apply -f minio.yaml                # creates minio pod
sudo kubectl  apply -f minio-service.yaml       # creates a service for minio
```

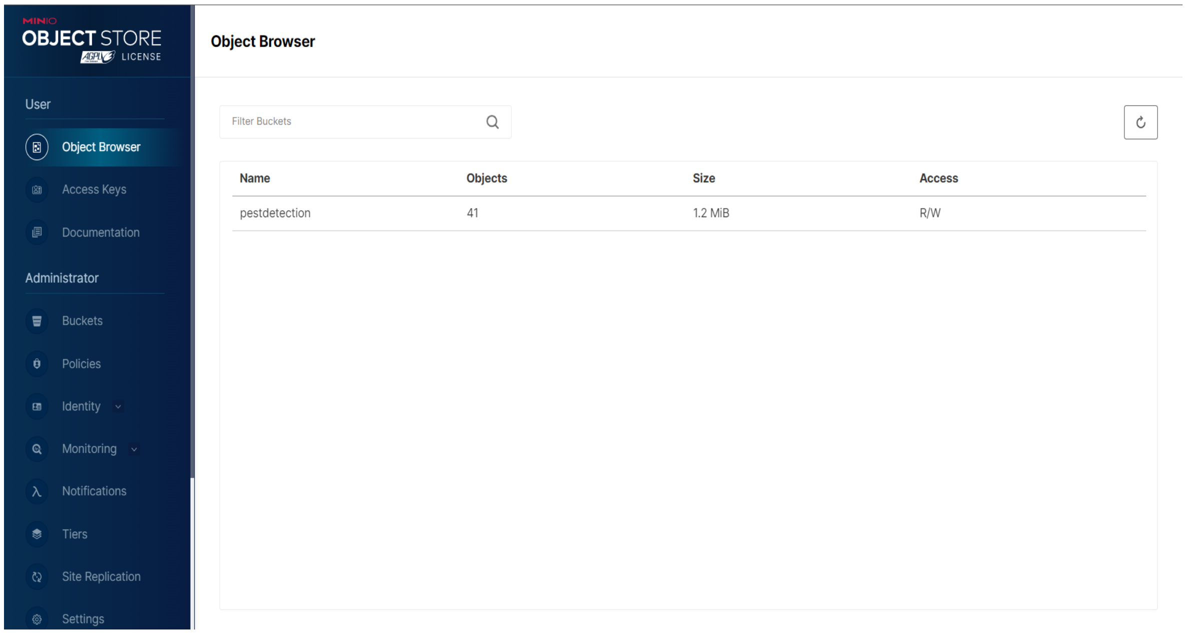
After successful deployment, running ***'sudo kubectl get pods -o wide'*** should give output similar to below.

```
NAME                              READY   STATUS    RESTARTS   AGE    IP
minio-deployment-7f7d8cd6b4-78bct 1/1     Running   0          169m   10.42.2.46
```

***sudo kubectl get services -o wide***

```
NAME          TYPE          CLUSTER-IP     EXTERNAL-IP                      PORT(S)
kubernetes    ClusterIP     10.43.0.1      <none>                           443/TCP
minio-service LoadBalancer  10.43.252.52   192.168.2.10,192.168.2.12        9000:31598/TCP,41589:32549/TCP
```

How to access MinIO Console? <Any Node IP>:9000



| Name | Objects | Size | Access |
|---|---|---|---|
| pestdetection | 41 | 1.2 MiB | R/W |

### 6.4.3. MQTT server Deployment

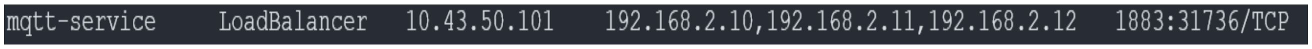
Go to directory : pestdetectionsystem/cluster_deploylemt/mosquitto
Execute: sh deploy.sh
Alternatively, you can execute below mentioned commands

```
sudo kubectl apply -f mosquitto.yaml              # creates mosquitto mqtt server pod
sudo kubectl apply -f mosquitto-service.yaml      # creates mosquitto service
```

After successful deployment, running ***'sudo kubectl get pods -o wide'*** should give output similar to below.

```
mqtt-deployment-f975dcf66-d5n65     1/1    Running   0          17m    10.42.2.91    knode2
```

***sudo kubectl get services -o wide***



```
mqtt-service     LoadBalancer   10.43.50.101    192.168.2.10,192.168.2.11,192.168.2.12   1883:31736/TCP
```

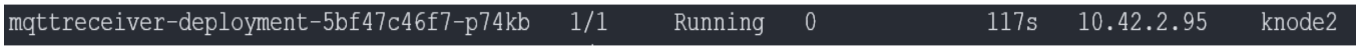
How to access MQTT service? MQTT server can be accessed by using any node ip and its port number 1883.

### 6.4.4. MQTTReceiver Application Deployment

Go to directory : pestdetectionsystem/cluster_deploylemt/mosquitto
Execute: sh deploy.sh
Alternatively, you can execute below mentioned commands

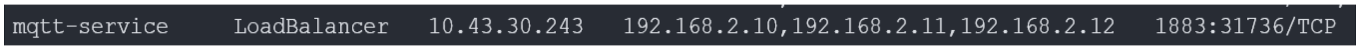
```
sudo kubectl apply -f mqttreceiver.yaml                # creates mqttreceiver pod
```

After successful deployment, running ***‘sudo kubectl get pods -o wide’*** should give output similar to below.



```
mqttreceiver-deployment-5bf47c46f7-p74kb   1/1     Running   0            117s   10.42.2.95    knode2
```

***‘sudo kubectl get services -o wide’***



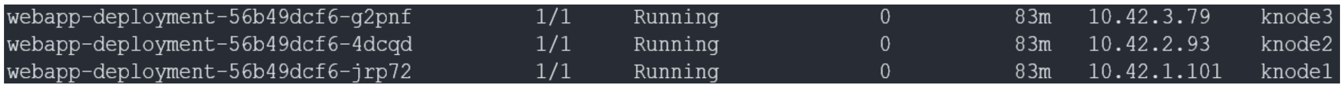
```
mqtt-service     LoadBalancer   10.43.30.243   192.168.2.10,192.168.2.11,192.168.2.12   1883:31736/TCP
```

### 6.4.5. WebApp Deployment

Go to directory : *cd pestdetectionsystem/cluster_deploylemt/WebApp*
Execute: ***sh deploy.sh***
Alternatively, you can execute below mentioned commands

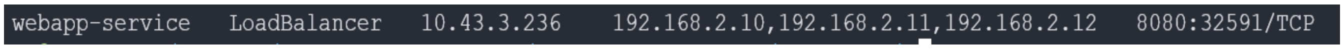
```
sudo kubectl apply -f webApp.yaml                # creates webApp pod
sudo kubectl apply -f webApp-service.yaml        # creates webApp service
```

After successful deployment, running ***‘sudo kubectl get pods -o wide’*** should give output similar to below.



```
webapp-deployment-56b49dcf6-g2pnf          1/1     Running           0          83m   10.42.3.79    knode3
webapp-deployment-56b49dcf6-4dcqd          1/1     Running           0          83m   10.42.2.93    knode2
webapp-deployment-56b49dcf6-jrp72          1/1     Running           0          83m   10.42.1.101   knode1
```

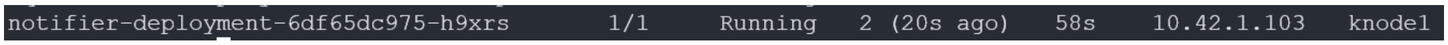
***‘sudo kubectl get services -o wide’***


```
webapp-service   LoadBalancer   10.43.3.236    192.168.2.10,192.168.2.11,192.168.2.12   8080:32591/TCP
```

### 6.4.6. Telegram Notifier Deployment

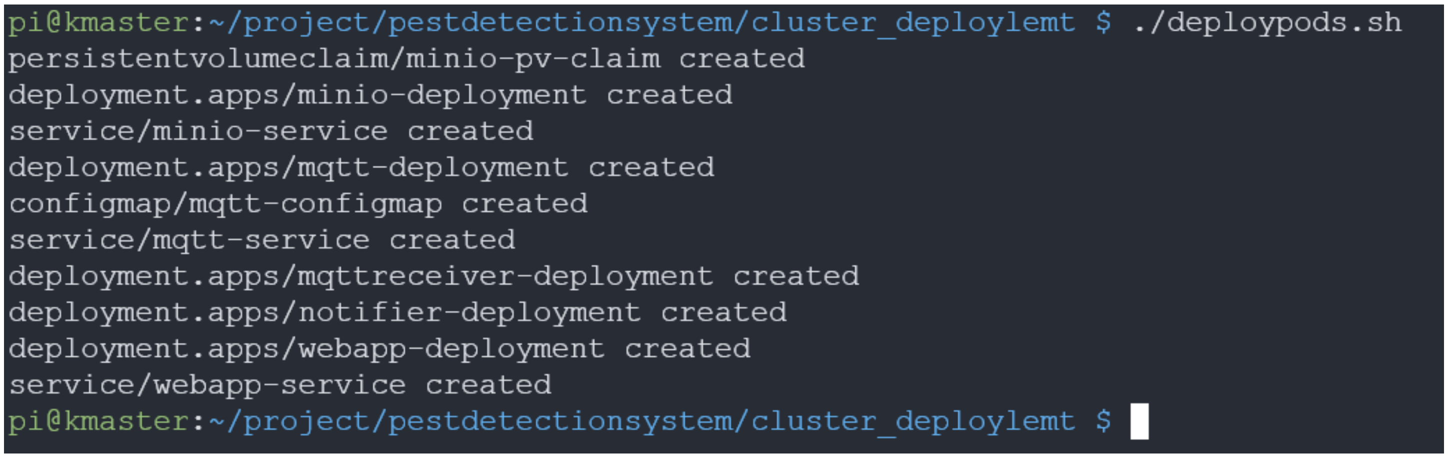
Go to directory : *cd pestdetectionsystem/cluster_deploylemt/notifier*
Execute: ***sh deploy.sh***
Alternatively, you can execute below mentioned commands
*sudo kubectl apply -f notifer.yaml* # creates mqttreceiver pod

After successful deployment, running ***'sudo kubectl get pods -o wide'*** should give output similar to below.


```
notifier-deployment-6df65dc975-h9xrs      1/1     Running   2 (20s ago)    58s    10.42.1.103   knode1
```

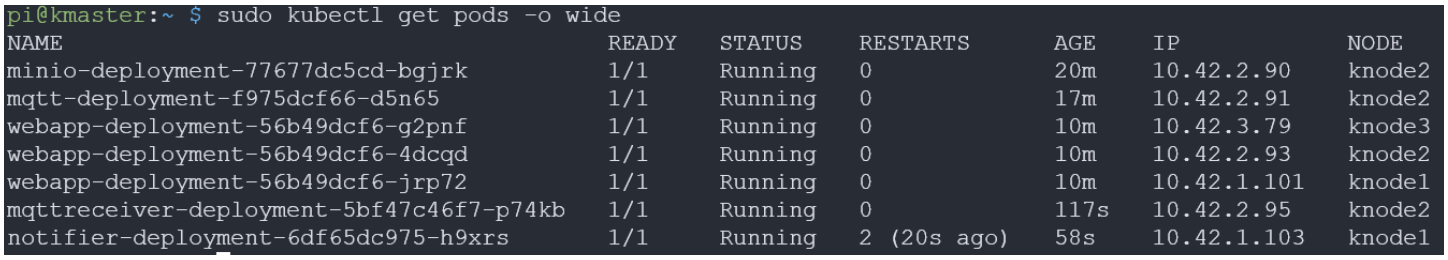
### 6.4.7. Alternate Procedure of deployment

Alternative to the above process of deploying applications one by one, users can also run ***deploypods.sh*** which is available in the ***cluster_deployment*** directory. This script will take care of all of the deployment.



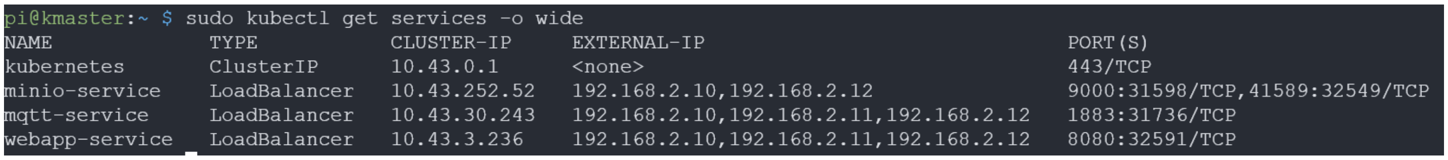
```
pi@kmaster:~/project/pestdetectionsystem/cluster_deploylemt $ ./deploypods.sh
persistentvolumeclaim/minio-pv-claim created
deployment.apps/minio-deployment created
service/minio-service created
deployment.apps/mqtt-deployment created
configmap/mqtt-configmap created
service/mqtt-service created
deployment.apps/mqttreceiver-deployment created
deployment.apps/notifier-deployment created
deployment.apps/webapp-deployment created
service/webapp-service created
pi@kmaster:~/project/pestdetectionsystem/cluster_deploylemt $
```

After successful deployment status of all pods will look similar to below:



```
pi@kmaster:~ $ sudo kubectl get pods -o wide
NAME                                      READY   STATUS    RESTARTS      AGE    IP            NODE
minio-deployment-77677dc5cd-bgjrk         1/1     Running   0             20m    10.42.2.90    knode2
mqtt-deployment-f975dcf66-d5n65           1/1     Running   0             17m    10.42.2.91    knode2
webapp-deployment-56b49dcf6-g2pnf         1/1     Running   0             10m    10.42.3.79    knode3
webapp-deployment-56b49dcf6-4dcqd         1/1     Running   0             10m    10.42.2.93    knode2
webapp-deployment-56b49dcf6-jrp72         1/1     Running   0             10m    10.42.1.101   knode1
mqttreceiver-deployment-5bf47c46f7-p74kb  1/1     Running   0             117s   10.42.2.95    knode2
notifier-deployment-6df65dc975-h9xrs      1/1     Running   2 (20s ago)   58s    10.42.1.103   knode1
```

Status of services will similar to below:



```
pi@kmaster:~ $ sudo kubectl get services -o wide
NAME             TYPE           CLUSTER-IP      EXTERNAL-IP                                PORT(S)
kubernetes       ClusterIP      10.43.0.1       <none>                                     443/TCP
minio-service    LoadBalancer   10.43.252.52    192.168.2.10,192.168.2.12                  9000:31598/TCP,41589:32549/TCP
mqtt-service     LoadBalancer   10.43.30.243    192.168.2.10,192.168.2.11,192.168.2.12     1883:31736/TCP
webapp-service   LoadBalancer   10.43.3.236     192.168.2.10,192.168.2.11,192.168.2.12     8080:32591/TCP
```

### 6.5. Edge Node Configuration

Copy node directory in the sensor node or clone the github repository as mentioned in the above section.

Go to the directory: ***cd ./pestdetection/node***
Run following commands:

```
pip install -r requirements
sudo cp detect2.service /etc/systemd/system/
sudo systemctl enable detect2.service
sudo systemctl start detect2.service
```

To see logs: ***tail -f detect2.log***

## 7. Testing Rat Detection System

### 7.1. Accessing the WebApp

After system setup, check if all pods are running and services are active as described in all above sections. After verifying, go to <any node ip>:8080. WebApp should look similar to fig 1.

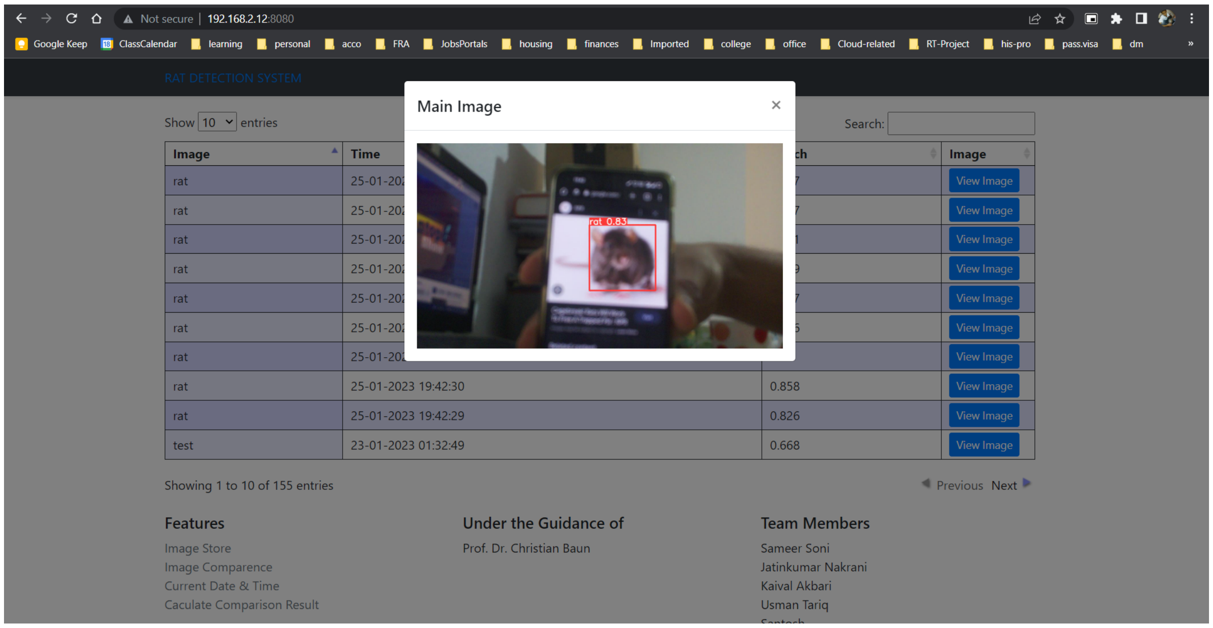
### 7.2. Subscribing to Telegram Bot

1. Go to Telegrams App on your mobile
2. Go to search "***cc-pest-ws-22-bot***"
3. Join bot

After subscribing to this bot, the user will start receiving detection images every time the system detects a rat.

### 7.3. Checking Results

a. Show a rat in front of camera
b. Go to WebApp by typing <node ip>:8080, webApp should show something similar to Fig.1
c. Check the telegram bot for a picture similar to Fig 2.


RAT DETECTION SYSTEM

Show 10 entries

Search:

| Image | Time | ch | Image |
|---|---|---|---|
| rat | 25-01-202 | 7 | View Image |
| rat | 25-01-202 | 7 | View Image |
| rat | 25-01-202 | 1 | View Image |
| rat | 25-01-202 | 9 | View Image |
| rat | 25-01-202 | 7 | View Image |
| rat | 25-01-202 | 5 | View Image |
| rat | 25-01-202 | | View Image |
| rat | 25-01-2023 19:42:30 | 0.858 | View Image |
| rat | 25-01-2023 19:42:29 | 0.826 | View Image |
| test | 23-01-2023 01:32:49 | 0.668 | View Image |

Main Image

Showing 1 to 10 of 155 entries

Previous Next

Features

Image Store
Image Comparence
Current Date & Time
Caculate Comparison Result

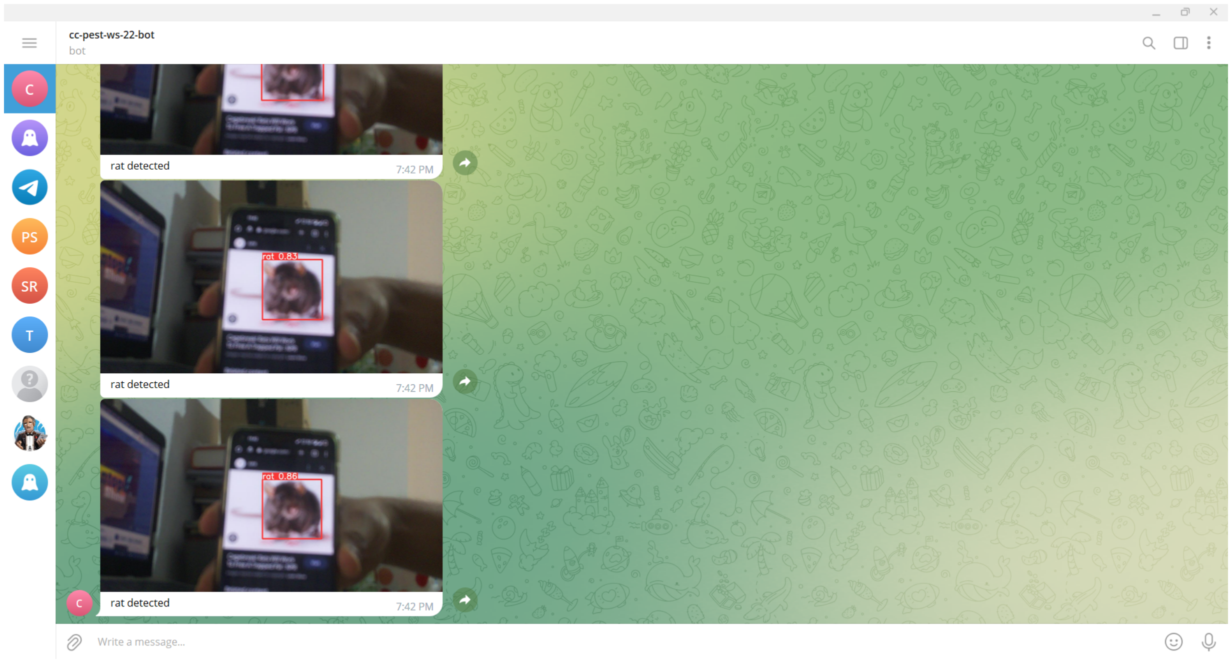
Under the Guidance of

Prof. Dr. Christian Baun

Team Members

Sameer Soni
Jatinkumar Nakrani
Kaival Akbari
Usman Tariq

WebApp [Fig 8]


cc-pest-ws-22-bot
bot

rat detected 7:42 PM

rat detected 7:42 PM

rat detected 7:42 PM

Telegram Bot [Fig 9]

## 8. Open Points

- WebApp currently refreshes every 8 seconds which is not a good idea, WebApp should only load when new objects are added.
- MinIO Object storage is deployed on Single Node Single Drive architecture which works but does not provide reliable storage. MinIO should be deployed as Single Node Multi Drive Architecture
- Sensor node takes camera frames and processes which can be improved by feeding camera output directly to the model.
- PiCamera quality is bad, better camera integration is suggested.

## 9. References

[1] https://min.io/docs/minio/kubernetes/upstream/

[2] https://mosquitto.org/download/

[3] k3s Installation

[4] https://docs.k3s.io/advanced