

# Rat Detection Using Raspberry Pi

Rahul Bhowmik Shuvo, Arash Abdollahi Kakroodi, Mohammad Aftabudduza, Husain Ahmad Jahid  
*High Integrity Systems (MSc.)*  
Frankfurt University of Applied Sciences

## I. INTRODUCTION

Pests, particularly rats, seem to be a major problem in people's daily life. Rat detection using a Raspberry Pi plays a vital role in detecting the presence of rats using a small, low-cost computer. This paper presents a system for detecting rats using Raspberry Pi. This system proposes Raspberry Pi for model implementation, Cameras for capturing images, and YOLOV5 for testing and training of the objections detection model.

## II. BUILD A RASPBERRY PI KUBERNETES CLUSTER WITH K3S

The system is generated with 4 Raspberry Pi-3 and one Raspberry Pi-4. Here is one Raspberry used for the sensor node and the other three for master and worker nodes.

### A. Installation procedure of Raspberry Pi OS

First needs to download Raspberry Pi Imager and install on a local machine [1]. After installing, choose Raspberry Pi OS (64-bit) for the operating system then select microSD card as the following figure 1. Then it is configured with Raspberry Pi-04.

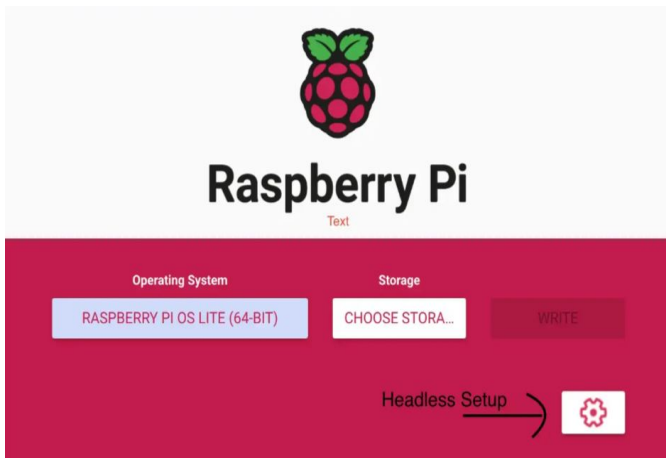


Fig. 1. Raspberry Pi Image:(64bit)

Choose the SD card and follow the write button to flash the card with the version of Raspberry Pi OS. After flashing the card, it needs to check the availability of the host network.

After being set up on Raspberry Pi-04, Raspberry Pi OS Lite (64-bit) is installed and configured on other Raspberry Pi-03 nodes as like before. The following figure 3 demonstrates

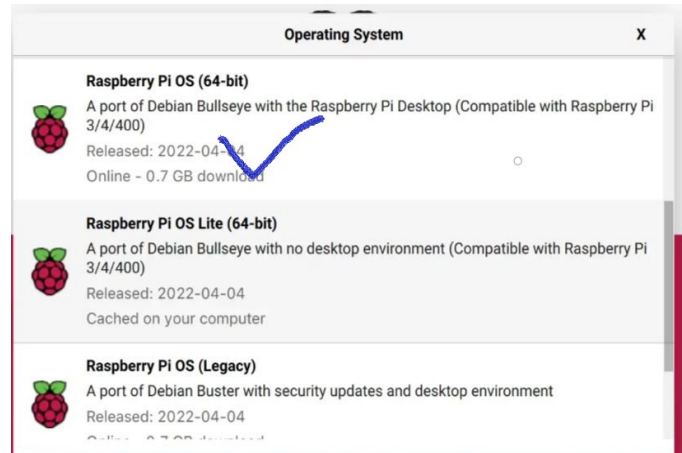


Fig. 2. Raspberry Pi OS(64bit)

the installation of Raspberry Pi OS Lite (64-bit) on these three Raspberry Pi-03.

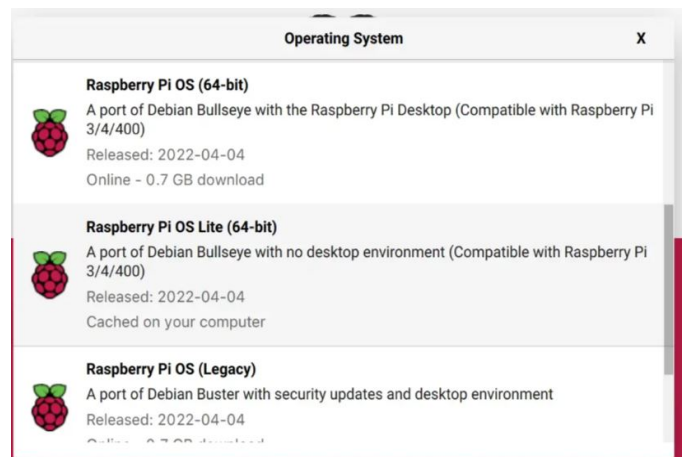


Fig. 3. Raspberry Pi OS lite(64bit)

Now unique host-name are written for the individual Raspberry Pi such as Kmaster for the master node and knode1 for the worker node1. After the naming, fix up the wifi configuration as well enable ssh along with password authentication. Then follow 4 the save button to the SD card. As like worker node1, need to write the same configuration for other nodes as well.

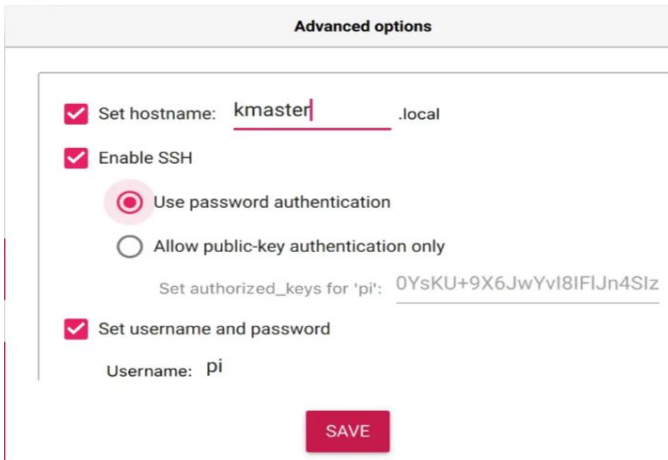


Fig. 4. Nodes labelling

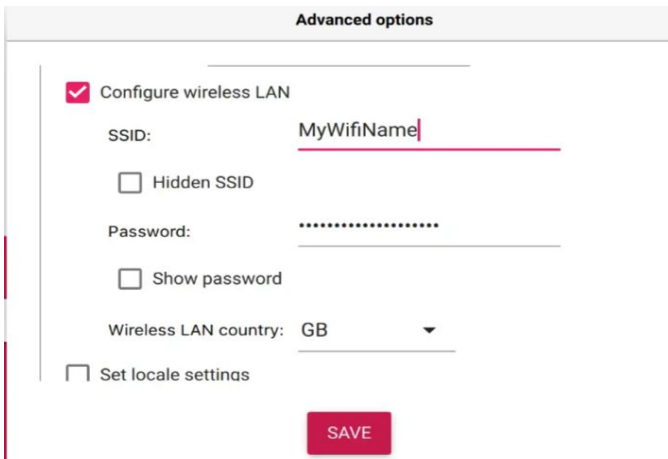


Fig. 5. Network set up

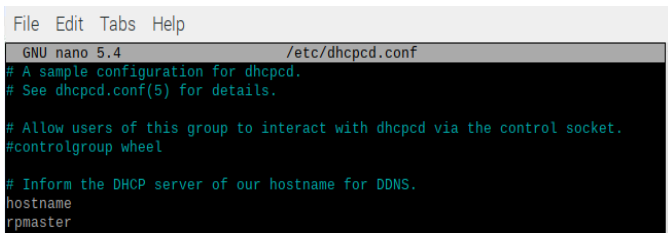


Fig. 6. Network set up

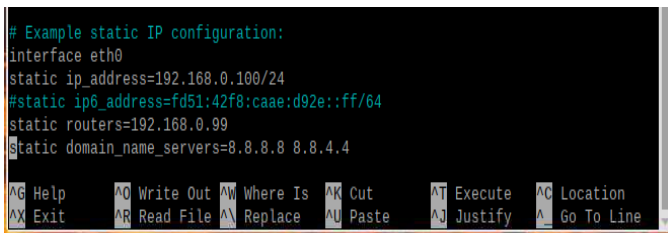


Fig. 7. Network set up

After installing OS, before removing the sd card from the computer it's needed to edit dhcpd.conf from the directory /etc for set IP address for sensor and master node6. So here is ip address 192.168.0.99 for sensor node and for 192.168.0.100 for all other raspberry pi-03 showed on figure.7

After configuration, power on all Raspberry Pi and check their availability on the current network. For set-up confirmation, run the ping command 8 according to their hostnames.

```
ping kmaster.local
ping knodel.local
```

Fig. 8. Ping command

### B. Final configurations

To upgrade the operating system and to enable the cgroup memories on raspberry pi's, an ssh connection can be done to remote access the boards looks like the following image 9.

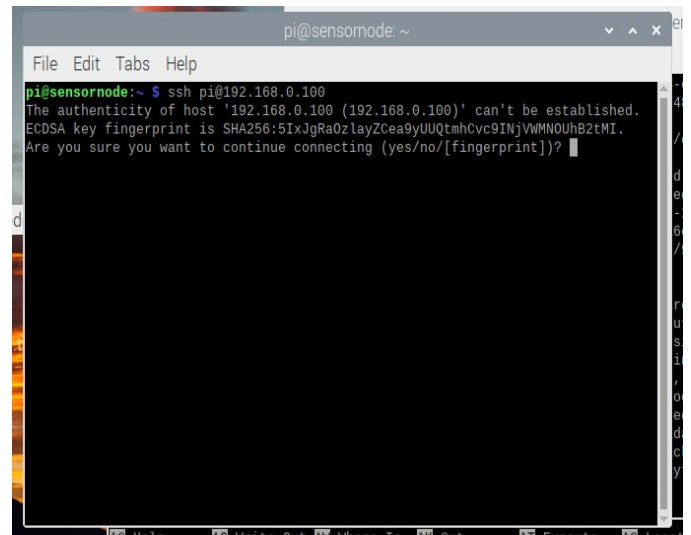


Fig. 9. SSH Connection for remote access

After this step, from the path /boot the following text must be added to the file cmdline.txt without creating a new line:

```
cgroup_memory=1 cgroup_enable=memory
```

The final step is to run the below command to upgrade the operating system:

```
sudo apt-get update && sudo apt-get upgrade
```

### C. Creating k3s Kubernetes Cluster

For the Raspberry pi cluster, a lightweight Kubernetes distribution k3s is used. K3S is specially designed for IOT solutions with restricted-resources devices such as raspberry pi boards.

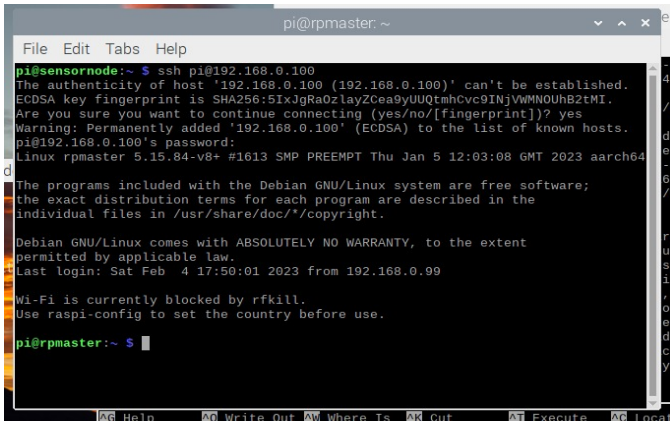


Fig. 10. SSH Connection for remote access

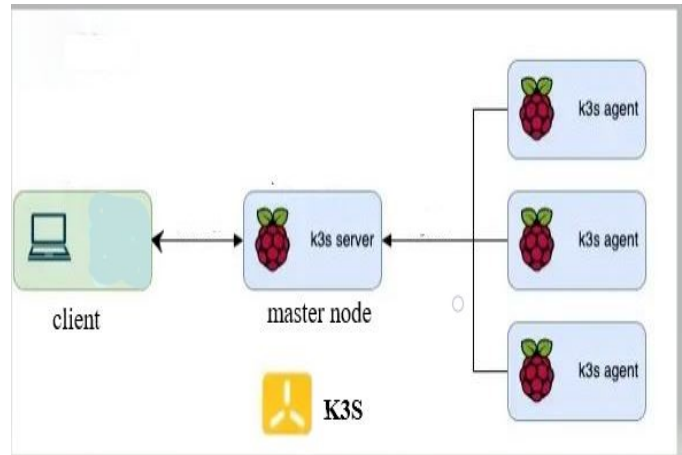


Fig. 12. System Architecture

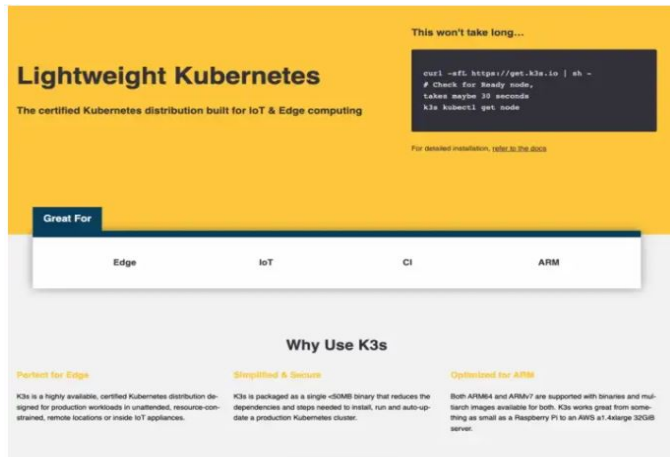


Fig. 11. K3s downloading

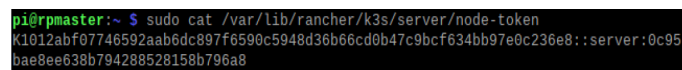


Fig. 13. Server-token ID

on figure 14, are created on this device. One of this images which belongs to the program on sensor node will be run locally on raspberry pi 4 and the other two which are for the logger program on the back-end and the front end website will be pushed to docker hub to be used for deployment on cluster.

```
pi@sensornode:~$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
arashkakroodi/sensor	v1	e2b396d43413	2 hours ago	2.5GB
sensor	v1	f9ec430a81f7	29 hours ago	2.5GB
arashkakroodi/logger	v1	976c102ae993	2 days ago	954MB
python	3.9-bullseye	4d4ab8de1554	11 days ago	867MB
python	3.9.16-bullseye	4d4ab8de1554	11 days ago	867MB
python	3.10.9-bullseye	a443975c5ddd	11 days ago	872MB
hello-world	latest	46331d942d63	10 months ago	9.14kB

Fig. 14. Docker Images

1) *System Architecture*: K3S server installed on the master node whereas worker nodes Raspberry Pi possesses K3S worker as like the following figure 12. And all of the agent nodes are registered to the master node:

To install K3s [2] the following command must be executed on the master node:

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

After this step, the token from the master node can be read from the file shown on figure 13.

With this token, k3s can be installed[3] on worker nodes by the following command:

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

### III. DOCKER IMAGES

For running the program on cluster, it is essential to create docker images. The challenge is to create docker images for raspberry pi CPU architecture. To achieve this goal, docker is installed on raspberry pi-4 and the docker images, shown

### IV. DEPLOYMENT OF DOCKER IMAGES ON CLUSTER

To deploy the docker images of the applications on cluster, a deployment configuration file must be created to specify a name for the service, Name of the docker image, replica set and the ports which are needed for communication. This file will be saved with a meaningful arbitrary name with the extension of yaml. The following image is illustrated this 15.

The following command can be used to deploy the docker image using the yaml file:

```
kubectl apply -f <filename>.yaml
```

At this point, it is seen the pod running on the cluster, However to access the container within the pod a service must be defined with another yaml file which contains the type of the services(LoadBalancer, NodePort, ...), and the necessary port definitions.

```

1 apiVersion: v1
2 kind: Deployment
3 metadata:
4   name: logger
5   labels:
6     name: logger
7 spec:
8   replicas: 1
9   selector:
10    matchLabels:
11      name: logger
12   template:
13     metadata:
14       labels:
15         name: logger
16     spec:
17       containers:
18         - name: logger
19           image: logger:v1
20           ports:
21             - name:
22               containerPort: 10234
23               protocol: TCP
24             - name:
25               containerPort: 3306
26               protocol: TCP

```

Fig. 15. Deployment code

## V. DEPLOYMENT OF MOSQUITTO MQTT BROKER ON CLUSTER

In this project to communicate between different applications, one eclipse Mosquitto version 1.6.15 has been used. The reason for this version choice is that by default versions above 2 only serve on localhost and cannot be reached by other computers in the network. To achieve that, a configuration file must be created and a persistent volume must be defined in the deployment's yaml file which unfortunately was time constrained for such practice. The chosen version by default serves all the nodes in the network.

After the Deployment of the broker on k3s a service is needed to provide access to the MQTT broker on the cluster. Hence the IP address of the server is required for clients, this service is defined from the type LoadBalancer. MQTT uses default port 1883.

To program the publisher and the subscriber (the publisher is the docker image running on the sensor node and the current subscriber is the logger application on the cluster), The sample code on the documentation was tried which was functional for small payloads. However, for heavy payloads such as images, it was inconsistent and unreliable and the reason was the disability of the publisher to successfully send the message. To solve this issue on the publisher side, two methods are provided by the API [4] which are `loop_start()`, `loop_forever` and `loop_stop()` methods. By these methods, the `publish()` method which is not a block, is handled successfully.

Another challenge is to encode the binary image file as a string because the payload of the MQTT message must be a string. To achieve the goal of the project, we used the same encoding standard that 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.

## VI. CREATING THE RAT DETECTOR MODEL

1) *Data Collection and Labeling*: Gather images of rats and label them to create the training dataset. The dataset should include a variety of different rat poses, lighting conditions, and backgrounds. In this case, a data set from roboflow is used.

[app.roboflow.com/frauas/rat\\_detection/3](https://app.roboflow.com/frauas/rat_detection/3)

2) *Dataset Preparation*: The dataset is then split into two parts: a training set and a validation set. The training set is used to train the model, while the validation set is used to evaluate its performance.

3) *Hardware and Software Setup*: A suitable hardware platform for training the model is chosen, such as a high-end desktop computer or a cloud-based GPU instance. In this case, google colab16 is used as a cloud-based computer. The necessary software and libraries, including a deep learning framework such as PyTorch or TensorFlow, and the YOLOv5 implementation, are installed.

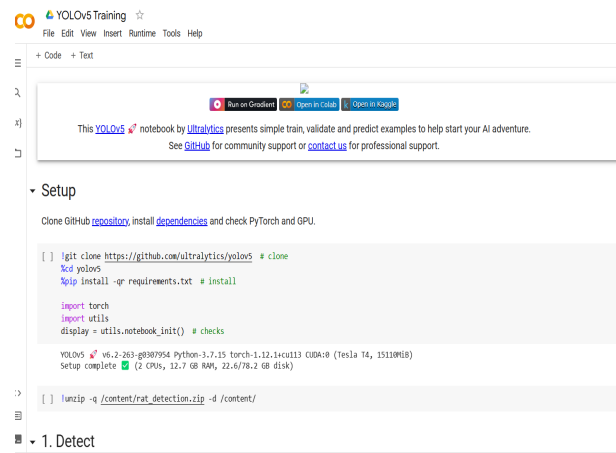


Fig. 16. Google colab platform

4) *Model Initialization* : The YOLOv5 architecture and weights pre-trained on a large image classification dataset are downloaded.

5) *Configuration*: The YOLOv5 configuration file is modified to reflect the characteristics of the rat dataset, including the number of classes, the input image size, and the number of anchor boxes. The hyperparameters for the training process, such as the learning rate, batch size, and a number of epochs, are set.

6) *Training*: The training process is started, with the model being fed the training set images and their corresponding labels. The training process is monitored, including the loss function and the accuracy of the validation set. The model weights are saved at regular intervals or when the accuracy



of the validation set reaches a desired threshold.

7) *Model Evaluation and Fine-Tuning:* The trained YOLOv5 model is used to perform object detection on a test set of images. The performance of the model is evaluated, including the precision, recall, and F1-score. The model is fine-tuned and the evaluation process is repeated if necessary.

By following these steps, a YOLOv5 model can be trained on a rat dataset collected from roboflow and its performance can be evaluated for the task of rat detection.

## VII. USING THIS MODEL IN RASPBERRY PI 4 WITH RASPBERRY PI CAMERA V2

1) *Selection of Trained Model:* A YOLOv5 model that has been trained on a rat dataset is selected for use.

2) *Setting up the Sensor Node:* A Raspberry Pi 4 board with Raspbian OS 64 bits installed is chosen to serve as the sensor node.

3) *Connecting the Camera:* A Raspberry Pi Camera v2 is connected to the Raspberry Pi 4 board to capture images of the environment.

4) *Transferring the Model and Libraries:* The trained YOLOv5 model and the necessary libraries required for running the model are transferred to the Raspberry Pi 4 board.

5) *Configuring the Raspberry Pi 4:* The Raspberry Pi 4 board is configured to run the YOLOv5 model using the Raspberry Pi Camera v2 as the input source.

6) *Running the Model:* The YOLOv5 model is run on the Raspberry Pi 4 board to perform object detection on the images captured by the Raspberry Pi Camera v2.

7) *Processing and Communication of Results:* The detection results are processed and communicated to the master node for further analysis.

By following these steps, the trained YOLOv5 model can be used on a Raspberry Pi 4 board with a Raspberry Pi Camera v2 for the task of rat detection.

## VIII. THE SYSTEM VISUALISATION

The web component of the rat detection system has been designed to seamlessly connect the database and the frontend. This is achieved by using the Flask web framework. Flask is a lightweight and flexible framework that provides the necessary tools and functionalities to handle the communication between the database and the frontend.

The frontend of the system has been developed using HTML, CSS, and JavaScript. HTML provides the structure and layout of web pages. CSS is used to style and enhance the visual appearance of the frontend. JavaScript is used to provide dynamic functionalities and interactivity to the frontend. These technologies combined create an interface that allows users to easily access and view the detection results stored in the database shown on figure 18.



Fig. 17. Web page

Thank you for visiting and we look forward to sharing our progress with you.

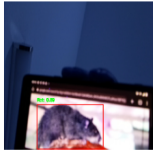
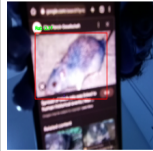
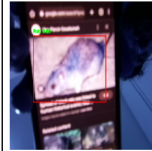
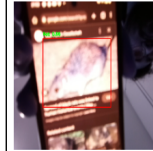




Start of discovery: 02/04/2023, 23:14:19	Start of discovery: 02/04/2023, 20:52:29	Start of discovery: 02/04/2023, 20:52:20	Start of discovery: 02/04/2023, 20:52:07
End of discovery: 02/04/2023, 23:14:54	End of discovery: 02/04/2023, 20:52:31	End of discovery: 02/04/2023, 20:52:28	End of discovery: 02/04/2023, 20:52:19
			
Start of discovery: 02/04/2023, 20:51:32	Start of discovery: 02/04/2023, 20:50:37	Start of discovery: 02/04/2023, 20:50:35	Start of discovery: 02/04/2023, 20:49:43
End of discovery: 02/04/2023, 20:52:06	End of discovery: 02/04/2023, 20:51:31	End of discovery: 02/04/2023, 20:50:36	End of discovery: 02/04/2023, 20:50:34
			

Fig. 18. Data page

## IX. CONCLUSION

In summary, the "Rat Detector Using Raspberry Pi" project has described the implementation of a system that uses one Raspberry Pi 4 board as the sensor node, four Raspberry Pi 3 boards as worker nodes, and a Raspberry Pi Camera v2 as the input source. The system was developed using Raspbian OS 64 bits for the Raspberry Pi 4 and Raspbian

OS lite 64 bits for the Raspberry Pi 3 boards. The object detection model was trained using YOLOv5s on a rat dataset. The trained model was used on the Raspberry Pi 4 board to perform object detection, and the results were communicated to the master node for further analysis. The documentation has described the steps involved in setting up the system and using the trained model for the task of rat detection.

#### REFERENCES

- [1] [Online]. Available: ["https://medium.com/thinkport/how-to-build-a-raspberry-pi-kubernetes-cluster-with-k3s-76224788576c"](https://medium.com/thinkport/how-to-build-a-raspberry-pi-kubernetes-cluster-with-k3s-76224788576c)
- [2] [Online]. Available: ["https://docs.k3s.io/advanced#raspberry-pi"](https://docs.k3s.io/advanced#raspberry-pi)
- [3] [Online]. Available: ["https://saintcoder.wordpress.com/2017/08/14/sharing-internet-connection-to-raspberry-pis-wired-to-local-network/"](https://saintcoder.wordpress.com/2017/08/14/sharing-internet-connection-to-raspberry-pis-wired-to-local-network/)
- [4] [Online]. Available: ["https://www.eclipse.org/paho/index.php?page=clients/python/index.php/"](https://www.eclipse.org/paho/index.php?page=clients/python/index.php/)