FRANKFURT UNIVERSITY OF APPLIED SCIENCES

Cloud Computing - Dog Detection Software

Group 7

Jemish Moradiya
jemish.moradiya@stud.fra-uas.de
Matriculation Number: 1417346

Meet Kiritbhai Makadiya
meet.makadiya@stud.fra-uas.de
Matriculation Number: 1422302

Barun Chakroborty
barun.chakroborty@stud.fra-uas.de
Matriculation Number:

Md Shahinur Rahman
md.rahman4@stud.fra-uas.de
Matriculation Number: 1396467

Index

1. Introduction

Cloud computing offers a wide array of services, including storage, databases, networking, processing power, software, analytics, and intelligence, all delivered over the internet ("the cloud"). This technology enables faster innovation and flexible resource allocation.

By leveraging cloud computing, a Raspberry Pi-based object detection system can be implemented. Raspberry Pi, a low-cost single-board computer, can run machine learning algorithms and other software applications. When connected to sensors and cameras, Raspberry Pi can collect data and store it in the cloud.

In a dog detection system, a Raspberry Pi can be connected to a camera module to capture images or videos. These images can be processed using trained machine-learning models to detect and identify dogs. The results of the dog detection can then be stored in a MinIO server on the cloud, enabling further analysis, reporting, and display of the detected images on a user interface.

The data collected, combined with the images, provides a comprehensive understanding of the dog population. This data can be used to train machine learning models for more accurate dog detection and to study the behavior of rats in a given environment.

Leveraging cloud computing in a dog detection system brings several advantages, such as scalability, accessibility, and cost-effectiveness. Additionally, it eliminates the need for expensive hardware and storage solutions, making the dog detection system more affordable and accessible to a wider range of users.

2. Architecture Diagram

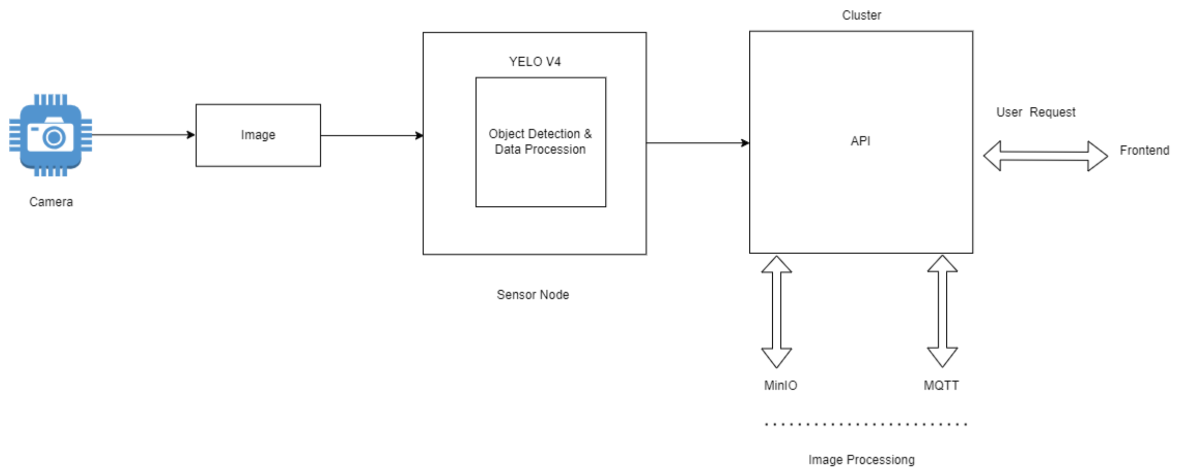


Figure 1: Architecture Diagram

The user interface through which users interact with the dog detection software. It could be a web-based interface, a mobile app, or any other form of user interface. We used it for our project MQTT. The MQTT is a very strong communication hub that acts as a central messaging hub, facilitating communication between the user interface and the Raspberry Pi devices. It receives messages published by the Raspberry Pi devices and routes them to the appropriate subscribers. The cloud computing infrastructure provides the computational power and scalability required for processing dog detection requests. It can include virtual machines, containers, serverless functions, or any other cloud resources. Cloud services, running on the cloud computing infrastructure, act as subscribers to the MQTT broker. They subscribe to specific MQTT topics to receive the dog image data published by the Raspberry Pi devices.

On the other hand, The MinIO cluster is responsible for storing the dog image datasets, training data, and other necessary files. It provides reliable and scalable object storage across multiple nodes, ensuring data redundancy and high availability. Raspberry Pi devices serve as edge devices in the architecture. They capture or receive dog images and publish them to the MQTT broker. These devices run the dog detection software locally for initial processing. The flow of the dog detection process involves the user interface capturing or uploading an image. The image is then published to the MQTT broker. The cloud services, acting as subscribers, receive the image data, process it using the dog detection software, and provide the detection results. The MinIO cluster stores the dog image datasets and other necessary files required for processing.

3. Project Configuration

3.1 Hardware Configurations

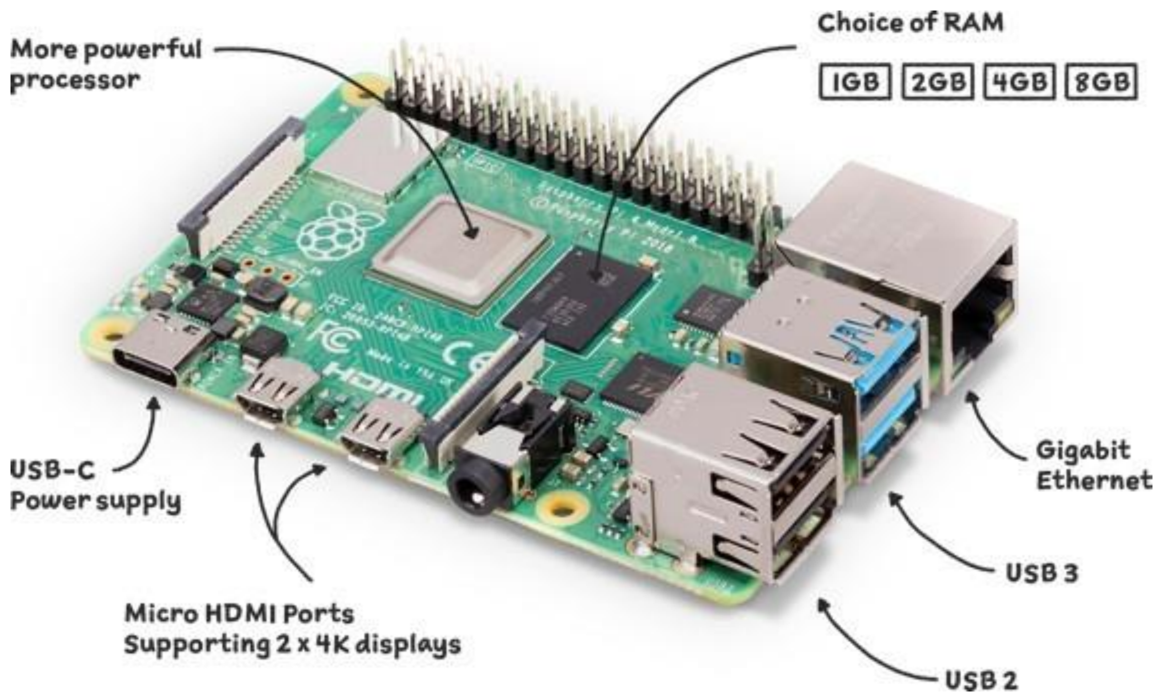


Figure 2: Raspberry Pi 4 (Sensor Node)

The Raspberry Pi 4 is a cost-effective and high-performance single-board computer developed by the Raspberry Pi Foundation. It finds applications in various fields like home automation, computer vision, IoT, and artificial intelligence. The Raspberry Pi 4 supports different operating systems such as Raspbian, Debian Ubuntu, Windows, and Linux, and it is compatible with popular programming languages like Python, C, C++, Java, and more.

In the context of live dog detection, the Raspberry Pi 4 serves as an edge node, running the Raspberry Pi 4 BULLSEYE 64-bit Debian OS. By utilizing the Raspberry Pi 4 as an edge node, the system benefits from reduced data transmission to the cloud, lower latency, bandwidth savings, and faster data processing. This approach allows the dog detection software to perform real-time analysis on the Raspberry Pi 4 itself, without relying heavily on cloud-based processing, making it more efficient and responsive.

3.2 Raspberry Pi 3 for Cluster: Cloud computing refers to the delivery of computing resources, such as storage, processing power, and software applications, over the internet. In this context, dog detection software utilizing a cluster configuration of three Raspberry Pi 3 devices can leverage cloud computing to enhance its capabilities. The cluster configuration allows for distributed processing and increased computational power, enabling more efficient and accurate dog detection algorithms and analysis.

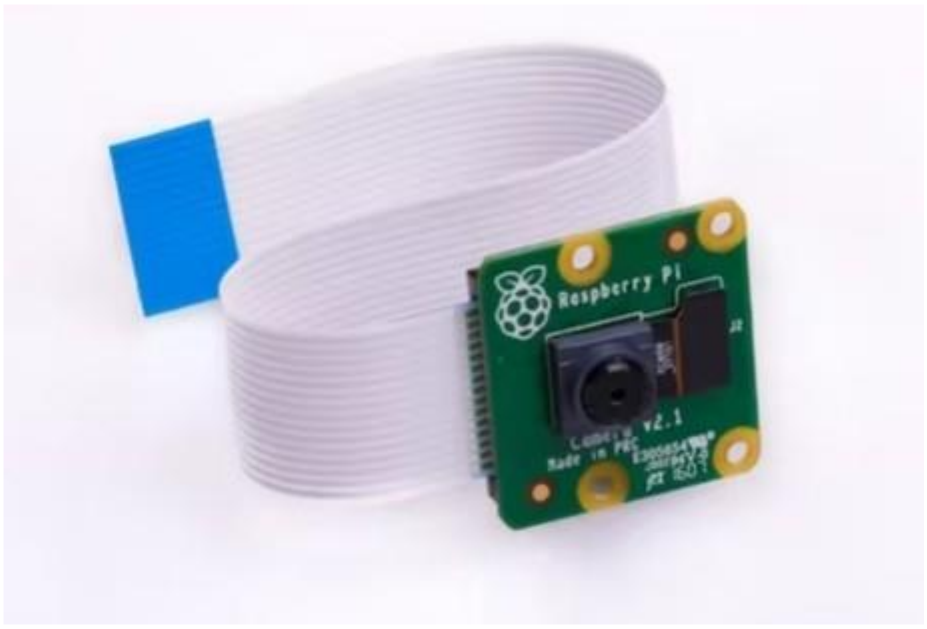


Figure 3: Raspberry Pi Camera Module v2

The Raspberry Pi Camera Module v2 is an official camera module provided by the Raspberry Pi Foundation. It is specifically designed for Raspberry Pi devices and offers excellent capabilities for capturing high-quality video and still images. The camera module features a Sony IMX219 8-megapixel sensor, which delivers high-resolution images and provides a wide field of view.

In the context of dog detection, the Camera Module v2 is utilized to capture frames from the live feed. It is connected to the Raspberry Pi using a ribbon cable, allowing the Raspberry Pi to access the camera's imaging capabilities. By leveraging the camera module, the system can capture real-time video frames, which can then be processed using the dog detection software to detect the presence of dogs.

3.3 Software Configurations

Python: Python is a widely used, high-level programming language that is well-suited for object detection tasks in computer vision. Raspberry Pi supports Python and comes with Python pre-installed in its operating system.

For object detection in computer vision, popular frameworks like TensorFlow, PyTorch, and OpenCV provide Python APIs that make it easy to train and deploy object detection models. These frameworks offer a range of tools and libraries for building and working with object detection models.

Python's popularity extends to the Raspberry Pi ecosystem, where many libraries and packages are specifically designed for Raspberry Pi development. These libraries provide functionalities for accessing the GPIO (General Purpose Input/Output) pins, controlling the Raspberry Pi Camera Module, working with sensors, and more. With Python on Raspberry Pi, developers have a rich ecosystem of tools and resources to leverage for object detection and other projects.

Python: OpenCV is indeed a widely-used open-source computer vision library, and it is highly suitable for running on the Raspberry Pi 4. By installing the OpenCV library on the Raspberry Pi 4, developers can leverage its extensive collection of algorithms and functions for image and video processing, including object detection capabilities.

One significant advantage of using OpenCV on the Raspberry Pi 4 is its improved performance. Compared to earlier models, the Raspberry Pi 4 is equipped with a more powerful CPU and GPU, making it better equipped to handle computationally demanding computer vision tasks. This increased processing power enables smoother and faster execution of object detection algorithms, enhancing the overall performance of the system.

By utilizing OpenCV on the Raspberry Pi 4, developers can tap into the library's robust features and optimize their object detection workflows for real-time applications. The combination of OpenCV and the Raspberry Pi 4's improved hardware capabilities allows for more advanced and efficient computer vision processing on this platform.

YOLO V5: YOLO V5 (You Only Look Once version 5) is indeed a state-of-the-art real-time object detection model that has gained significant popularity in computer vision applications. While YOLO V5 typically requires a powerful GPU for optimal performance, it is still feasible to use it on the Raspberry Pi 4 for object detection tasks.

To utilize YOLO V5 on the Raspberry Pi 4, an OpenCV library that includes the implementation of the YOLO V5 algorithm is required. This library provides the necessary tools and functions to train and detect objects, including rats, from a live camera feed.

In your project, you can leverage the YOLO V5 algorithm implemented in OpenCV to train a dog detection model and perform real-time object detection on the Raspberry Pi 4. By utilizing the computational capabilities of the Raspberry Pi 4, you can apply the YOLO V5 model to analyze the live camera feed and detect the presence of rats.

While the Raspberry Pi 4 may not offer the same level of performance as a powerful GPU, it can still handle object detection tasks with YOLO V5, albeit potentially at a slightly reduced speed. It is important to optimize the model and fine-tune the performance parameters to ensure efficient execution on the Raspberry Pi 4's hardware.

Minio: The Minio Client library is a software library designed to interact with Minio, an object storage server. It offers a set of straightforward APIs that enable developers to access Minio and perform various operations. These operations can include tasks such as listing buckets, creating new objects, deleting objects, and more.

With the Minio Client library, developers can easily integrate Minio into their applications and leverage its object storage capabilities. The library abstracts away the complexity of interacting with the Minio server and provides a simplified interface for accessing and managing objects stored in Minio.

By using the Minio Client library, developers can efficiently interact with Minio and perform necessary operations for their applications, such as managing data stored in Minio buckets or creating new objects within the storage server. This simplifies the development process and allows for seamless integration with Minio in a variety of use cases.

3.4 Cluster Configurations

Raspberry Pi 4 model: The cluster hardware consists of four Raspberry Pi 3 devices. Among these devices, one is designated as the master node, and the remaining three are configured as worker nodes. Each Raspberry Pi 3 device is equipped with a 32 GB microSD card, on which the appropriate operating system will be installed.
The master node acts as the central control unit for the cluster, coordinating and managing tasks across the worker nodes. The worker nodes, on the other hand, perform the assigned computing tasks and communicate with the master node.
By setting up a cluster with Raspberry Pi 3 devices, you can distribute the workload and take advantage of parallel processing capabilities. This configuration allows for efficient and scalable computation, making it suitable for various applications, including data processing, distributed computing, and other computationally intensive tasks.

TP-LINK Switch: In addition to the Raspberry Pi devices, the hardware setup includes a TP-Link Switch. The TP-Link Switch is used to facilitate connectivity among the Raspberry Pi devices in the cluster. Each Raspberry Pi device is connected to the switch via its Ethernet ports.

By utilizing the TP-Link Switch, a centralized network connection is established, allowing for efficient communication and data transfer between the Raspberry Pi devices. The switch acts as a splitter, enabling multiple devices to connect to a single network infrastructure.

With the Ethernet connections, the Raspberry Pi devices in the cluster can seamlessly exchange data, synchronize tasks, and collaborate effectively. This setup ensures stable and reliable network connectivity, enabling efficient distributed computing within the cluster.

Fritz-Box: The hardware setup also includes a Fritz Box, which serves as a router for the cluster. The Fritz Box is responsible for providing network services, including IP address configuration and DHCP (Dynamic Host Configuration Protocol) functionality.

With the Fritz Box, the IP addresses of the Raspberry Pi devices in the cluster can be configured. The router assigns unique IP addresses to each device, allowing them to communicate and connect within the network. The DHCP functionality automates the process of IP address assignment, simplifying the setup and management of the network.

By utilizing the Fritz Box router, the cluster hardware setup benefits from a well-configured network infrastructure. The router ensures that the Raspberry Pi devices can establish reliable connections, communicate seamlessly, and access the necessary network resources.

3.5 To install the software components on the cluster

Before setting up the cluster, the following software deployment and configurations are done on the devices:

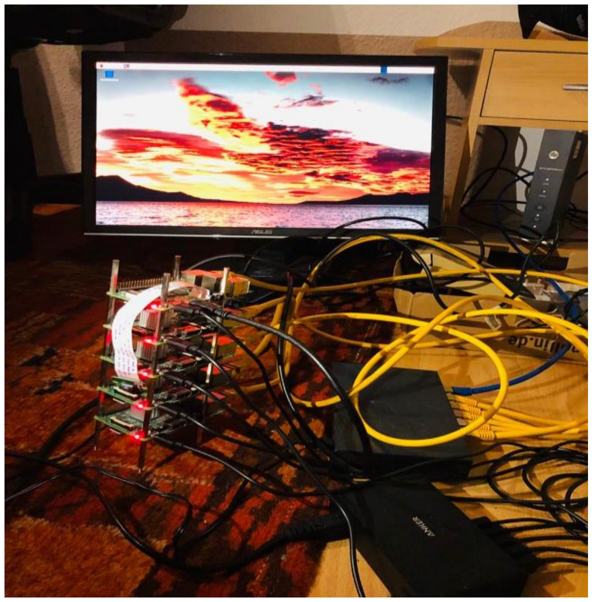
OS Installation: The Raspberry Pi imager tool is used to install the operating system (OS) on the SD cards. The chosen OS is the headless (lite) version of the 64-bit Debian Bullseye without a desktop environment. The decision to use the headless version is made to avoid overloading the devices and ensure optimal performance. During the OS image setup, SSH connection details can be added for later access to the Raspberry Pi devices via SSH protocol.

SD Card Setup: After installing the OS on the SD cards, they are plugged into the Raspberry Pi devices. Once the devices are ready, they are connected to the network through the router, which helps in identifying them on the network. Successful login via SSH allows for further configuration.

Package Update: After SSH access is established, the first step is to run the package update and upgrade commands to ensure that the devices have the latest packages and updates available.

Configuration Update: The /boot/cmdline.txt file needs to be modified by appending the following text at the end: cgroup_memory=1 cgroup_enable=memory. After saving the changes, the Raspberry Pi devices need to be restarted.

Static IP Setup: Setting up static IP addresses for all the nodes in the cluster is essential for maintaining stable configurations. The /etc/dhcpcd.conf file is updated on each available node (master and worker nodes) to define the static IP configurations. This ensures that each Raspberry Pi device retains the same IP address whenever it boots up and connects to the network.



4 Implementation

4.1 Camera Module and Edge Node Set Up for Object Detection

To set up a Raspberry Pi 4 as an edge node with the Pi Camera Module v2, the following steps are typically followed:

Raspberry Pi 4 BULLSEYE 64-bit Debian OS: To install and boot the Raspbian OS (Raspberry Pi 4 BULLSEYE 64-bit Debian OS) on a Raspberry Pi 4, follow these steps:

1. **Download Raspbian OS:** Visit the official Raspberry Pi website and download the Raspbian OS image suitable for Raspberry Pi 4.
2. **Flash the SD Card:** Use the Raspberry Pi imager tool or equivalent software to flash the Raspbian OS image onto the SD card. Insert the SD card into the Raspberry Pi 4.
3. **Power On the Raspberry Pi 4:** Connect the necessary peripherals (keyboard, mouse, and display) to the Raspberry Pi 4. Power it on by plugging it into a power source.
4. **Boot into Desktop Environment:** After the Raspberry Pi 4 is powered on, it should start booting into the Raspbian OS. It will take a few moments to initialize. Eventually, the Raspberry Pi 4 should reach the desktop environment.
5. **Access Desktop Environment via SSH (PuTTY):** If we want to access the Raspberry Pi 4's desktop environment remotely, we can use SSH. Ensure that the Raspberry Pi 4 is connected to the same network as your computer. Use an SSH client like PuTTY to establish an SSH connection to the Raspberry Pi's IP address. Login with your username and password, and we can access the Raspberry Pi 4's desktop remotely.
6. **Access Desktop Environment via VNC Viewer:** Another way to access the Raspberry Pi 4's desktop remotely is by using VNC Viewer. Make sure VNC is enabled on the Raspberry Pi 4 by going to the Raspberry Pi Configuration settings. Then, use a VNC viewer application on your computer to connect to the Raspberry Pi's IP address. Enter the appropriate credentials, and we can remotely access the Raspberry Pi 4's desktop environment.

Connect the Pi Camera Module v2:

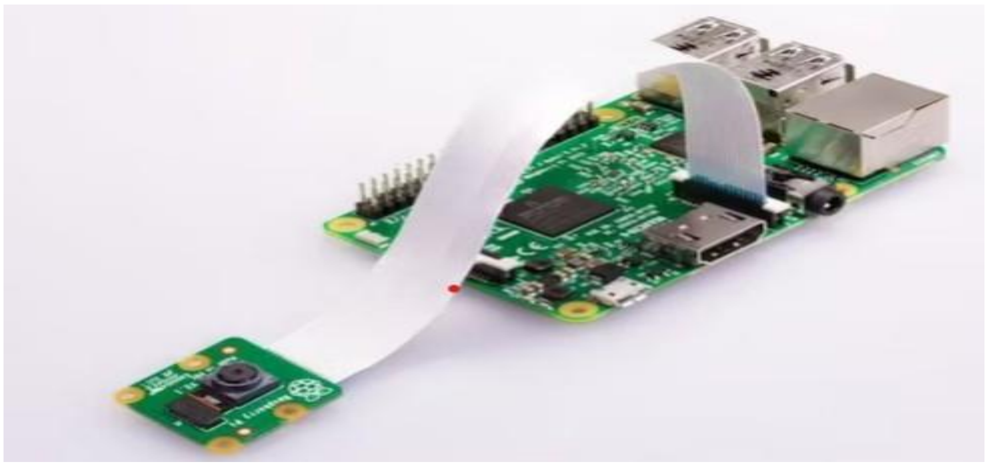


Figure 4: Connect the Raspberry Pi Camera Module v2

To connect the camera module to the Raspberry Pi 4 and enable the camera interface, follow these steps:

1. **Connect the Camera Module:** Make sure the Raspberry Pi 4 is powered off and not connected to any power source. Connect the camera module ribbon cable to the camera port on the Raspberry Pi 4 board. Ensure it is securely inserted in the correct orientation.
2. **Power On the Raspberry Pi 4:** After connecting the camera module, power on the Raspberry Pi 4 by plugging it into a power source.
3. **Enable Camera Interface:** Access the Raspberry Pi Configuration settings by running the command sudo raspi-config in the terminal. This will open the Raspberry Pi Software Configuration Tool.
4. **Navigate to Camera Settings:** Inside the Raspberry Pi Configuration Tool, use the arrow keys to navigate to "Interfacing Options" and press Enter.
5. **Enable Camera:** In the Interfacing Options menu, scroll down to "Camera" and press Enter. A prompt will appear asking if we want to enable the camera interface. Select "Yes" and press Enter to enable it.
6. **Update raspi-config:** Once the camera interface is enabled, update the raspi-config settings by running the command sudo raspi-config in the terminal again.
7. **Update Start X Setting:** Inside the raspi-config menu, navigate to "System Options" and press Enter. Then, scroll down to "Boot / Auto Login" and press Enter. In the next screen, select "Desktop Autologin" and press Enter. This will update the raspi-config with the setting to start the X desktop environment on boot.
8. **Reboot the Raspberry Pi 4:** After making these changes, exit the raspi-config menu and reboot the Raspberry Pi 4 for the changes to take effect. we can do this by running the command sudo reboot in the terminal.

OpenCV and Python Packages Setup: Python is included as part of the preinstalled Desktop version of Raspberry Pi OS on the Raspberry Pi 4. To enable object detection capabilities, dependencies such as NumPy and OpenCV 4.7.0 are installed. The installation of these dependencies allows for performing object detection tasks.
In order to upload the detected image to MinIO, the Minio Client library for Python is installed. This library enables interaction with the Minio server directly from Python code. To install NumPy, we can use the following commands:

NumPy installation:
(-) Install the pip package manage
# sudo apt-get install python3-pip
(-) Use pip to install Numpy
# pip3 install numpy
OpenCV Installation following commands are executed [5]
(-) Install minimal prerequisites
# sudo apt update   sudo apt install -y cmake g++ wget unzip
(-) Download and unpack sources
# wget -O opencv.zip https://github.com/opencv/opencv/archive/4.x.zip unzip opencv.zip

(-) Create build directory
# mkdir -p build cd build
(-) Configure
# cmake ../opencv-4.x
(-) Build
# cmake --build .
MinIO python library installation, the following command is used.
(-) Use pip to install Minio Client Library
# pip install minio

4.2 Cluster Configuration

Docker Container Setup for master node: To set up a Raspberry Pi cluster for this project, using Docker as the runtime container is preferred due to its widespread adoption and standardization for container images and runtime environments. By default, k3s uses containers as the container runtime. However, it can be changed to Docker by specifying the Docker runtime option during the k3s setup on each node.

To make Docker available for use, we can use one of Rancher's scripts to install Docker by executing the following command:

- *curl https://releases.rancher.com/install-docker/20.10.sh | sh*

This command fetches the script from the specified URL and executes it using the sh command.

After Docker is successfully installed, we can proceed with setting up k3s on the master node using the following command:

- curl -sfL https://get.k3s.io | sh -s - --docker

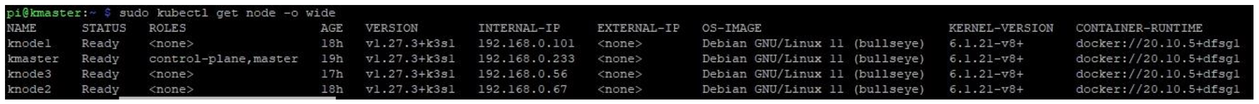
This command downloads the k3s installation script and executes it with the *--docker* flag to specify the Docker runtime.

Once the k3s service is up and running on the master node, the next step is to extract the token from the master node. This token will be used while setting up the worker nodes. The token can be found at the location */var/lib/rancher/k3s/server/node-token*. we can retrieve the token using various methods, such as reading the file directly or using commands like *dog or scp* to transfer the token to another node.

Remember to substitute the necessary values and adapt the commands according to your specific setup and requirements.
Docker Container Setup for worker node: To set up a worker node for the k3s service in the Raspberry Pi cluster, we can use the following command. Please note that the Docker runtime should already be available on the worker node. If not, refer to the previous section for instructions on downloading the Docker runtime.

- *curl -sfL https://get.k3s.io | K3S_URL=https://<ip-address>:6443 K3S_TOKEN=<node-token> sh -s - --docker*

Kubectl Utility Configuration:



```
pi@kmaster:~ $ sudo kubectl get node -o wide
NAME      STATUS   ROLES                  AGE   VERSION        INTERNAL-IP     EXTERNAL-IP   OS-IMAGE                         KERNEL-VERSION   CONTAINER-RUNTIME
knode1    Ready    <none>                 18h   v1.27.3+k3s1   192.168.0.101   <none>        Debian GNU/Linux 11 (bullseye)   6.1.21-v8+       docker://20.10.5+dfsg1
kmaster   Ready    control-plane,master   19h   v1.27.3+k3s1   192.168.0.233   <none>        Debian GNU/Linux 11 (bullseye)   6.1.21-v8+       docker://20.10.5+dfsg1
knode3    Ready    <none>                 17h   v1.27.3+k3s1   192.168.0.56    <none>        Debian GNU/Linux 11 (bullseye)   6.1.21-v8+       docker://20.10.5+dfsg1
knode2    Ready    <none>                 18h   v1.27.3+k3s1   192.168.0.67    <none>        Debian GNU/Linux 11 (bullseye)   6.1.21-v8+       docker://20.10.5+dfsg1
```

command:

- *Sudo kubectl get pods - wide*

To use the kubectl utility without physically logging into the Raspberry Pi, we can copy the contents of the file /etc/rancher/k3s/k3s.yaml from the master node to your local kubectl config file.

Follow these steps:

1. Copy the contents of the file */etc/rancher/k3s/k3s.yaml* from the master node.
2. Open your local *kubectl* config file. The default location for the config file is *~/.kube/config*.
3. Replace the value of the *server* field in the config file. Change https://127.0.0.1:6443 to the actual IP address of the Raspberry Pi running the master node. This ensures that *kubectl* communicates with the correct Kubernetes API server.

After making these changes, we should be able to use the *kubectl* utility from your local machine to interact with the Kubernetes cluster.

Deploying of MinIO in Cluster: MinIO was chosen as the database for this image detection software due to its specific design for handling the storage of objects, particularly images. Several factors were considered before selecting MinIO, including scalability, resilience, and the availability of a RESTful API specifically tailored for use in a Kubernetes cluster.

The deployment of MinIO is done using a deployment artefact, and a service is created to expose access to the command-line interface. MinIO offers multiple modes for setup in a cloud network. In this project, the Single Node Single Drive mode is utilized. This mode requires fewer hardware resources compared to the Multi-Node Multi-Drive mode, which involves synchronizing multiple instances of the database running on the cluster. Although the Multi-Node Multi-Drive mode was experimented with, due to the challenges faced and the time constraints of the project, the decision was made to implement the Single Node Single Drive mode.

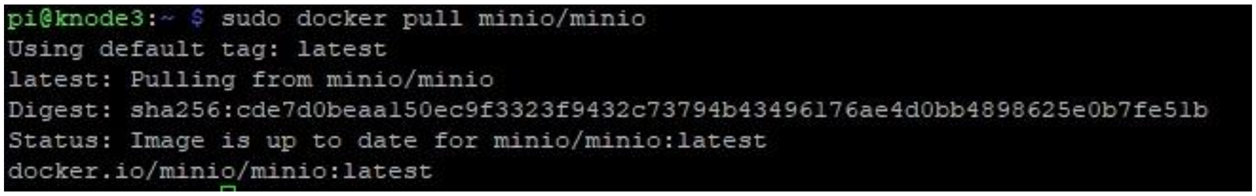
For detailed information and configuration settings, we can refer to the YAML file available in the GitHub repository at the following location: https://github.com/minio/minio

Minio over Kubernetes Service Configuration: At our project, MinIO is made available through a Kubernetes service, which is beneficial for testing and integration purposes. The service allows access to the MinIO Command Line UI from outside the cluster network, enabling easier management and interaction with the MinIO database.

The RESTful API provided by MinIO is utilized in this project to establish a connection between the Edge Node and the MinIO database. This connection enables seamless uploading of images from the Edge Node directly to the MinIO database, leveraging the capabilities of the MinIO API for efficient storage and retrieval of objects.

By utilizing the MinIO RESTful API and the Kubernetes service, the project achieves a streamlined workflow for uploading images to the MinIO database and leveraging its features for object storage and retrieval.

MinIO with persistent volumes Activation: Using persistent volumes is highly recommended to ensure data availability in scenarios where the POD running the MinIO server crashes for any reason. By utilizing persistent volumes, data loss can be prevented in cloud-based infrastructures.



```
pi@knode3:~ $ sudo docker pull minio/minio
Using default tag: latest
latest: Pulling from minio/minio
Digest: sha256:cde7d0beaa150ec9f3323f9432c73794b43496176ae4d0bb4898625e0b7fe51b
Status: Image is up to date for minio/minio:latest
docker.io/minio/minio:latest
```

Once the MinIO service is up and running, we can access it using the following command:

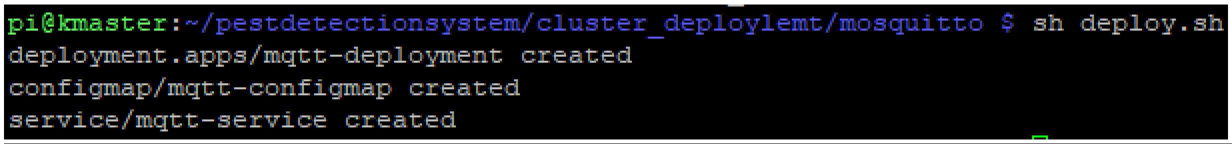
- *Sudo docker pull minio/minio*
- *- kubectl get services*

This command retrieves information about the services running in the Kubernetes cluster. It will display a list of services, including the MinIO service, along with details such as the service name, type, cluster IP, and external IP (if applicable).

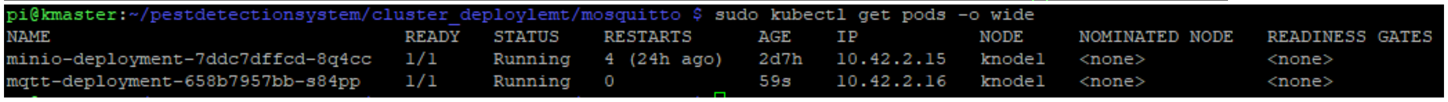
By running this command, we can obtain the necessary information to connect to and interact with the MinIO service in the Kubernetes cluster.

Mqtt:

MQTT is a lightweight messaging protocol for efficient communication between devices. It follows a publish-subscribe architecture, supports multiple Quality of Service levels, and operates asynchronously. MQTT is designed to have a small footprint and is widely adopted in IoT and other industries. It offers scalability, persistent sessions, security features, and is known for its simplicity and efficiency.



```
pi@kmaster:~/pestdetectionsystem/cluster_deploylemt/mosquitto $ sh deploy.sh
deployment.apps/mqtt-deployment created
configmap/mqtt-configmap created
service/mqtt-service created
```



```
pi@kmaster:~/pestdetectionsystem/cluster_deploylemt/mosquitto $ sudo kubectl get pods -o wide
NAME                              READY   STATUS    RESTARTS      AGE    IP           NODE     NOMINATED NODE   READINESS GATES
minio-deployment-7ddc7dffcd-8q4cc  1/1     Running   4 (24h ago)   2d7h   10.42.2.15   knode1   <none>           <none>
mqtt-deployment-658b7957bb-s84pp   1/1     Running   0             59s    10.42.2.16   knode1   <none>           <none>
```

Model Training: For object detection, the YOLO V5 model is commonly employed. YOLO, an acronym for We Only Look Once, is a real-time object recognition system capable of identifying multiple objects within a single frame. In comparison to other recognition systems, YOLO provides faster and more accurate object recognition.

- One crucial aspect of object detection is having high-quality labeled images for training data. we can find labeled datasets on websites like https://universe.roboflow.com/, which may meet your requirements. Alternatively, we can create your own labeled dataset using available tools. One recommended tool for labelling is Yolo Label, which can be found at https://github.com/developer0hye/Yolo-Label.

- After labelling the objects in your dataset, we should have a text file that contains the bounding box coordinates and labels for each object. The format of the label file can vary depending on the version of YOLO we are using. In YOLO V5, the label file typically follows the Darknet format, where each line represents an annotated object in an image. The format is as follows: *<object-class> <x_center> <y_center> <width> <height>*

  Here, <object-class> refers to the label or class of the object, <x_center> and <y_center> represent the normalized coordinates of the object's center relative to the image width and height, and <width> and <height> represent the normalized width and height of the bounding box, also relative to the image dimensions.
  For example, a line in the label file for an object labeled as "dog" with a bounding box centered at (0.5, 0.6) and dimensions of (0.3, 0.2) would look like this: *dog 0.5 0.6 0.3 0.2*
  It's important to note that the specific format of the label file may vary depending on the tools or scripts we use for labelling and the specific implementation of YOLO we are working with. Therefore, it's recommended to refer to the documentation or guidelines specific to your chosen labelling tool and YOLO version.

Model Training: The parameter "subdivisions" in the yolov5-custom.cfg file refers to the number of mini-batches that a complete batch is divided into during training. Let's consider an example where the batch size is set to 52. This means that for each training iteration, 52 images will be loaded.

If the subdivision value is set to 4, it means that the batch of 52 images will be split into 4 mini-batches. Each mini-batch will contain 52/4 = 13 images. These 13 images will be processed together as a mini-batch, and this process will be repeated until the entire batch is completed.

The size of the subdivision can be adjusted based on the available GPU memory. If the GPU has limited memory, a higher subdivision value can be set to process fewer images per mini-batch, reducing the memory requirement. On the other hand, if the GPU has ample memory, a lower subdivision value can be used, allowing for more images to be processed in each mini-batch.

When modifying the yolov5-custom.cfg file, we can make changes to the subdivision parameter based on your GPU memory capacity and requirements. It's important to find the right balance to ensure efficient training without running out of memory. Here is a paraphrased explanation of the parameters in the yolov5-custom.cfg file we provided:

- classes: This parameter defines the number of classes or types of objects that the model can detect. In this case, there is 1 class.
- max_batches: The training process will run for a total of 3000 iterations or batches.
- batch: Each training batch will contain 64 samples or images.
- subdivisions: The batch is divided into 16 blocks or mini-batches, which are processed in parallel on the GPU.
- width: The network size for input images will be 416 pixels in width.
- height: The network size for input images will be 416 pixels in height.
- channels: The input images will be converted to 3 channels during training and detection (typically Red, Green, and Blue channels).
- momentum: This parameter determines the accumulation of movement or how much the history affects the further weight changes during optimization. A value of 0.949 is used here.
- decay: The decay parameter diminishes the weights to avoid having excessively large values. It is set to 0.0005.
- learning_rate: The initial learning rate for the training process is 0.001.
- steps: These are checkpoints or specific iterations at which the learning rate will be adjusted. In this case, scales will be applied at iterations 640 and 720.
- scales: The scales parameter determines the coefficients at which the learning rate will be multiplied at the specified checkpoints. It controls how the learning rate changes during increasing iterations during training. Here, a scale of 0.1 is used for both checkpoints.
These parameters collectively define the configuration and settings for the YOLO V5 model during training.

Edge Node for Dog Detection:

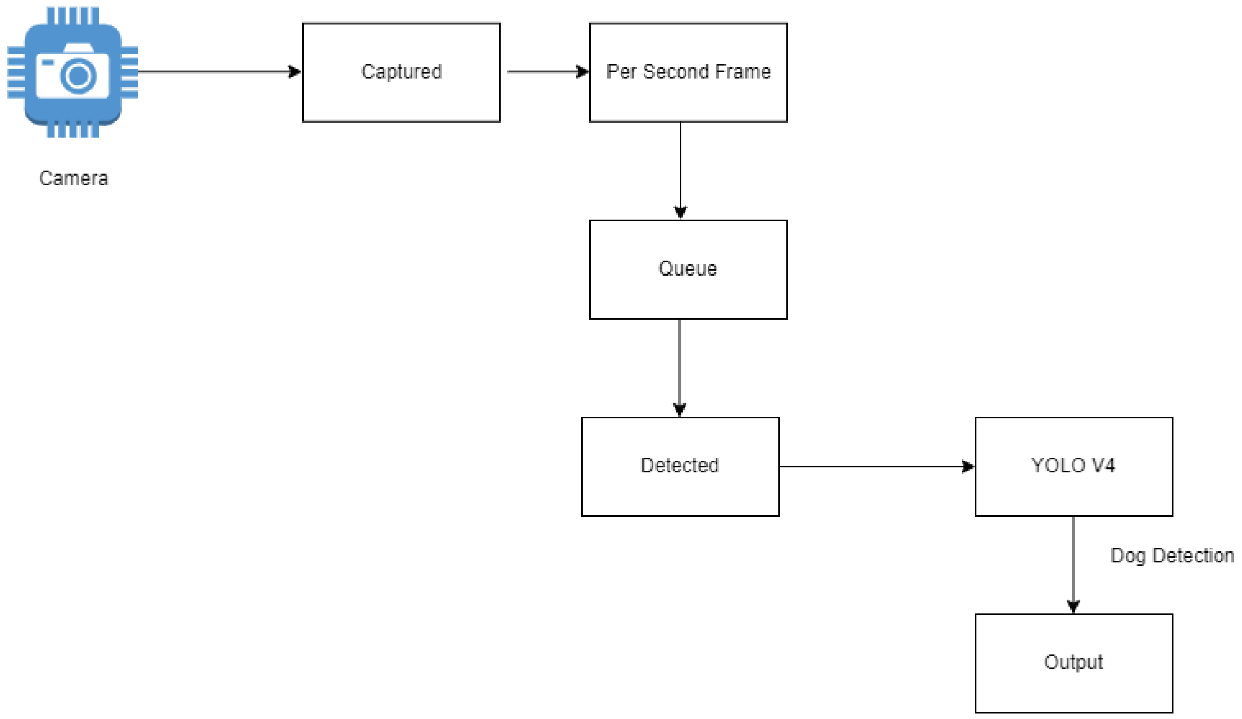
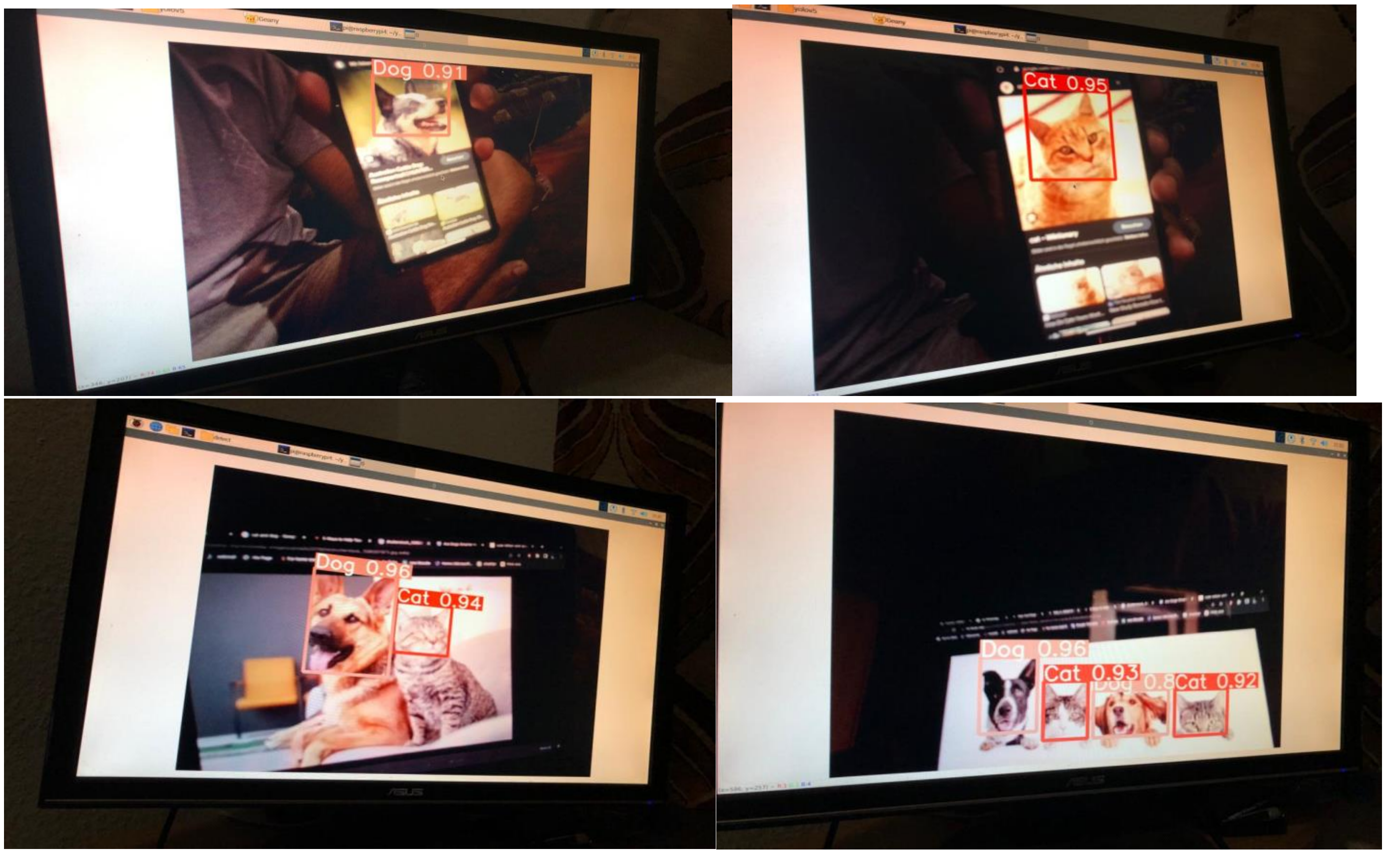


Figure 5: Dog Detection at Edge Node

The concept of dog detection at the edge node, based on the reference "Object detection using deep learning with OpenCV and Python", has been implemented using multithreading in Python and OpenCV. The architecture consists of two threads: Thread 1 captures the live feed and processes one frame per second, adding it to a queue. Thread 2 fetches a frame from the queue and applies the YOLO V4 detection model to it.

In the YOLO V4 detection model, the weights, configuration, and classes are provided, and the network is loaded into the OpenCV DNN module. The DNN module utilizes the pre-trained deep neural network model to calculate the output layers for object detection tasks. By forwarding an input image through the network, the output layers generate predictions that describe the locations of objects, along with their class probabilities.

The model identifies dogs by returning bounding boxes around the detected dogs, labeling them as "dog," and assigning confidence scores. An example of a detected dog at the edge node is depicted in the figure provided, with the class label "dog" and a confidence level of 72%.
For the implementation of object detection, you can refer to the codebase available at the following GitHub repository: https://github.com/shahinator/YOLOv5-Dog-Cat-Detection



Upload Frame to Minio:

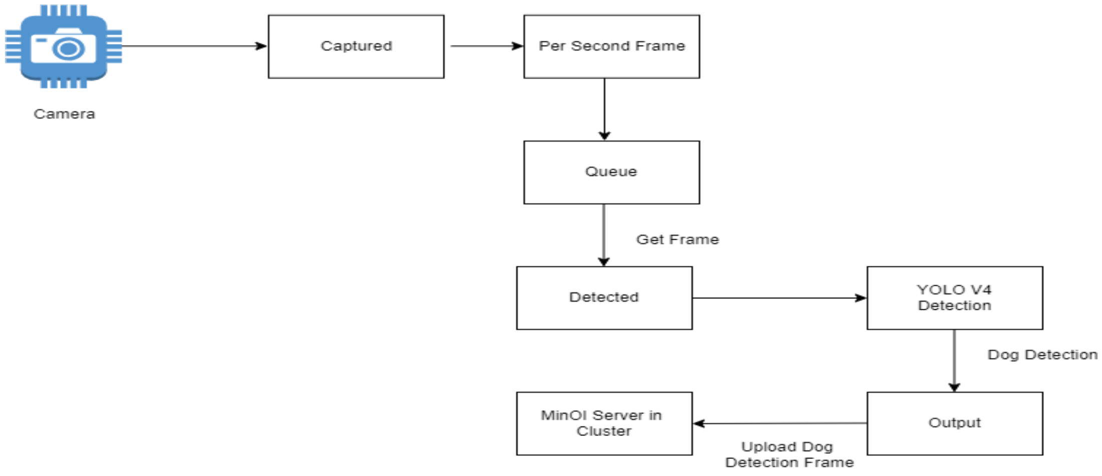


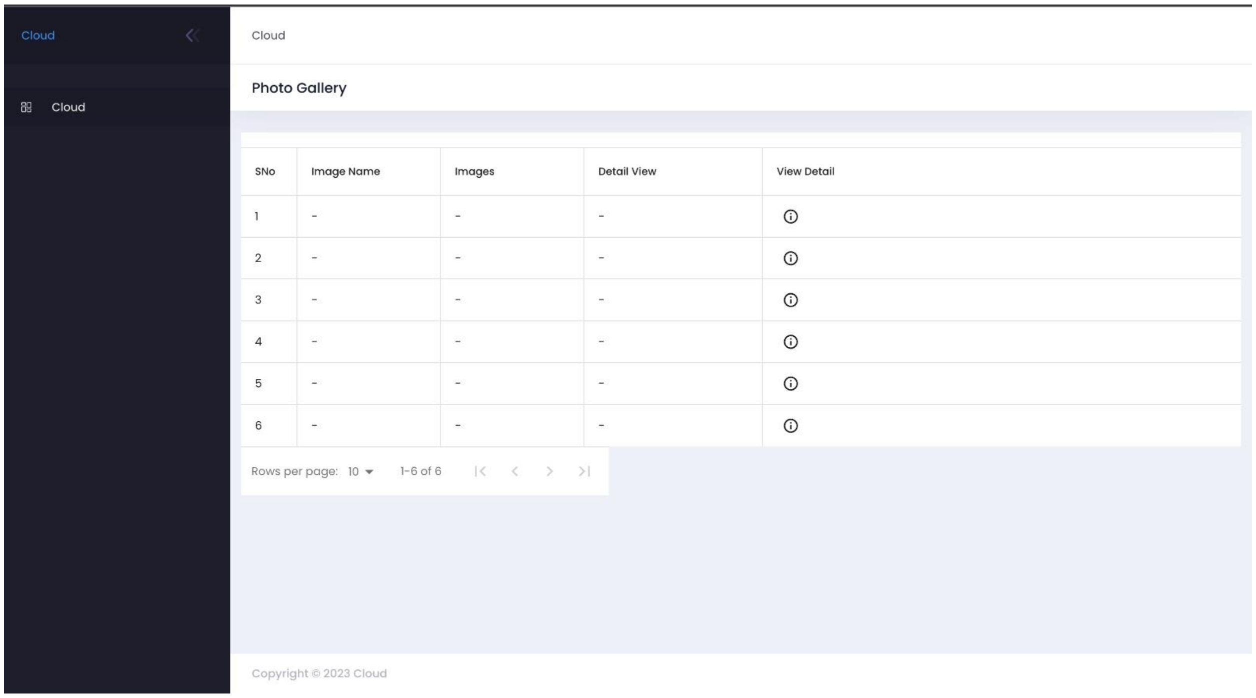
Figure 6: Dog Edge Frame Upload to MinIO

The process of detecting a dog using the object detection script involves uploading the resulting image to a MinIO server hosted in a K3s cluster. The figure provided illustrates the end-to-end process, starting from the camera capturing the image, detecting the rat, and finally uploading the image to the MinIO server.

To facilitate the upload, a MinIO client object is created, representing the MinIO server hosted in the K3s cluster. The MinIO client object is configured with the MinIO server's endpoint, access key, secret key, and security level. In order to store the image in the MinIO server, buckets need to be created. The MinIO client can be used to create the required bucket, and the image can be uploaded using the put object method.

The picture displayed demonstrates the rat-detected frame from the edge node, which has been successfully uploaded to the MinIO server hosted in the K3s cluster.

Web Application Development:



SNo	Image Name	Images	Detail View	View Detail
1	-	-	-	ⓘ
2	-	-	-	ⓘ
3	-	-	-	ⓘ
4	-	-	-	ⓘ
5	-	-	-	ⓘ
6	-	-	-	ⓘ

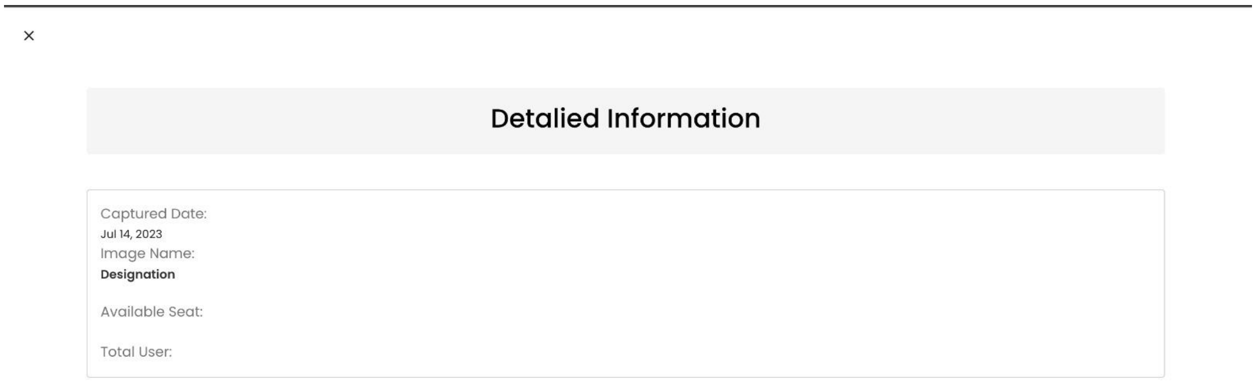
Rows per page: 10 1-6 of 6

Copyright © 2023 Cloud

Figure 7: Upload Dog Image Frame to MinIO

To showcase the dog images captured through the camera and stored in the MinIO database, a web application has been developed using a React JS framework. React JS is a lightweight web framework in JavaScript that offers convenient tools and features, simplifying the process of building web applications.

By leveraging React, a user interface has been constructed, allowing users to interact with the application. The React application serves as a platform to display the dog images retrieved from the MinIO database, providing a seamless and user-friendly experience.



Detalied Information

Captured Date:
Jul 14, 2023
Image Name:
Designation

Available Seat:

Total User:

Figure 8: Frontend View

5. Summary

The implementation of object detection using a K3s cluster cloud infrastructure on Raspberry Pi offers several advantages compared to traditional on-premise solutions. Cloud infrastructure allows for scalability, improved performance, faster object detection, and easy accessibility. The utilization of a K3s cluster facilitates efficient deployment and management of containerized object detection applications, while also enabling seamless integration with cloud services like storage and databases. However, there are challenges associated with implementing object detection in cloud infrastructure, including containerization, expertise in cloud infrastructure and object detection, and the need to address data privacy, security, and regulatory compliance concerns. Despite these challenges, the combination of these technologies provides a flexible and powerful solution for object detection that can be applied across various use cases.

6. Challenges

While implementing dog detection on Raspberry Pi with edge and cloud infrastructure, several challenges and learning experiences were encountered:

1. **Edge Node Configuration**: Configuration issues arose when accessing the camera's live feed with OpenCV on the Edge node. To resolve this issue, a complete reinstallation of OpenCV was required, as the error messages were not clear.
2. **Performance of YOLO V4 Model**: The Raspberry Pi 4's limited processing power posed performance issues when using the full YOLO V4 object detection model. Although using the YOLO V4 tiny model improved performance, it significantly reduced accuracy. To overcome this, multithreading was implemented to enhance the efficiency of the YOLO V4 model.
3. **Infrastructure Configuration and Sharing**: Sharing the infrastructure led to disarray in configurations, especially with pre-existing static IP addresses. As a result, there were instances where the infrastructure had to be set up from scratch. The risk of wear and tear during the exchange of devices or components also affected usability.
4. **Resource Constraints**: The effective utilization of the Raspberry Pi devices posed a learning curve due to their resource-constrained nature. Optimizing the code and managing resources efficiently became crucial for successful implementation.
5. **Limitations of MinIO in Multi-Node Multi-Drive Mode**: It was discovered that utilizing MinIO in Multi-Node Multi-Drive mode was not feasible due to the resource capacity required for synchronization processes.
6. **DHCP Server Configuration**: Configuring the infrastructure using a DHCP server proved to be a time-consuming process. Ultimately, a Fritz-Box was used to overcome this challenge.

These challenges and learnings contributed to a better undFritz Boxer standing of the limitations and considerations involved in implementing dog detection on Raspberry Pi with edge and cloud infrastructure.

7. References

[1] Raspberry Pi 4 Model B. Retrieved from: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/

[2] Raspberry Pi Camera Module V2. Retrieved from: https://www.raspberrypi.com/products/camera-module-v2/

[3] AlexeyAB/darknet GitHub Repository. Retrieved from: https://github.com/AlexeyAB/darknet/

[4] Object Detection with OpenCV GitHub Repository. Retrieved from: https://github.com/arunponnusamy/object-detection-opencv

[5] OpenCV Linux Installation Tutorial. Retrieved from: https://docs.opencv.org/4.x/d7/d9f/tutorial_linux_install.html

[6] Uploading Files in MinIO Using Python. Retrieved from: https://medium.com/featurepreneur/upload-files-in-minio-using-python-4f987f902076

[7] MinIO Linux Installation Documentation. Retrieved from: https://min.io/docs/minio/linux/operations/installation.html