Cloud Computing SS2023

Project Report

Automatic Pet Detection using Edge Computing

Supervisor: Prof. Dr. Christian Baun

Submitted By: Vikrant Shah (1411054),
Sushil Koirala (1421387),
Shikha Singh (1421659),
Nega Venkata Sai Kumar Vadlani (1417605),
Zaghman Saghir (1411960)

Submission Date: 14th July 2023

Table of Contents

Team Members and Contributions

Task	Contributors
K3S cluster, Sensor Nodesetup	Sushil Koirala
ML Model Training	Shikha Singh
ML model Deployment on pi4	Vikrant Shah
API development	Vikrant Shah
User Interface development	Vikrant Shah
Object Storage (MinIO) , Docker image and all services	Vikrant Shah , Sushil, Zaghman
Notification and Alerts	Naga Venkata Sai Kumar Valdani
Sensor Node and Cluster Integration	Zaghman Saghir
Project Integration	Team
Documentation and Presentation	Team

Repository: https://github.com/vikrant0526/his-cloud-computing-project

1. Introduction

Our objective for the SS2023 semester project was create to an edge computing solution to detect pets at sensor node and subsequently store the results in the cloud.

Cloud computing involves sending data to cloud for processing and storage, while edge computing processes data at the edge node and only transmits a limited amount of data to the cloud for storage. Edge computing offers numerous advantages compared to traditional cloud computing approaches. It provides improved latency by processing data near the source i.e., edge node. Additionally, edge computing solutions require less bandwidth since data isprocessed at the edge node. These solutions are scalable, reliable, cost-effective and prioritize privacy and security.

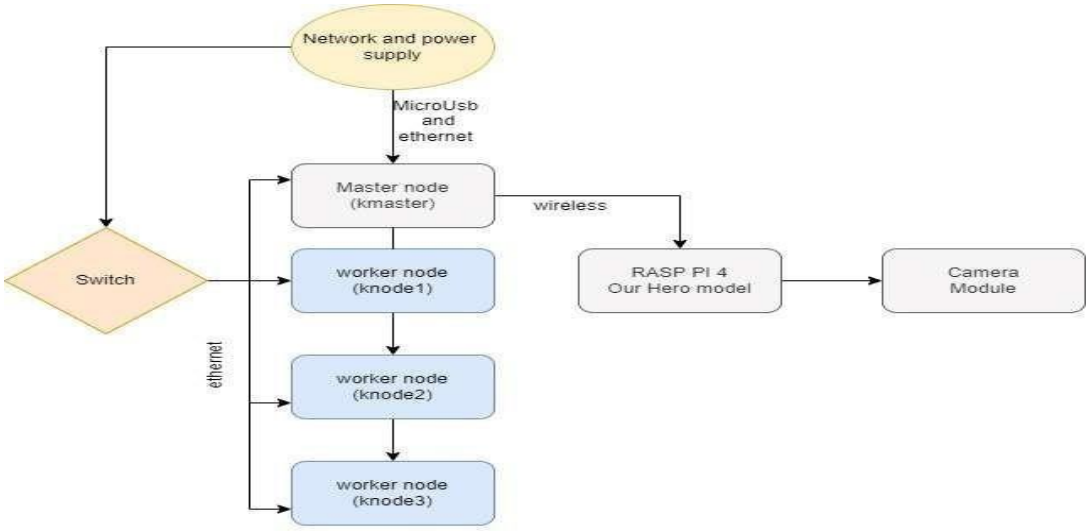
2. Architecture Diagram

In the given architecture diagram, the edge node performs the machine learning algorithm on its own device and calculates the expected result. Unlike cluster-managed setup, this configuration does not rely on multiple sensor nodes for data capture and processing. This is a fundamental concept used in Edge computing. In the event of sensor node failure, the other sensors nodes running in edge computing topology take over the task of capturing objects and performing necessary computation. The local processing of data on the edge node itself delivers faster and efficient results, eliminating the latency associated with transferring data back and forth to a cluster for processing.

The UI application has been developed using React, a framework that allows for rendering results. The displayed results include confidence intervals indicating the certainty of detecting pets.

By using the CLI is an effective method for assessing the overall health of cluster andmonitoring the status of various nodes, pods and services that are operating in the Cluster.

In the cluster there is one master node and three worker nodes. The master node manage active services, manage pod health and handle load balancing for incoming traffic. MinIO operates in a single pod using the Single node Single Drive deployment mode. Multiple pods are running for UI applications to distribute the incoming traffic.



3. Hardware and Software Configuration

3.1 Hardware Configuration in Edge Node

3.1.1 Raspberry Pi 4

The Raspberry Pi 4, developed by Raspberry foundation, is an affordable and high performance single-board computer with versatile applications in areas like home automation, computer vision IOT and AI. It supports various operating systems including Raspbian, Debian Ubuntu, windows and Linux and is compatible with popular programming languages such as Python, C and Java.

In the context of Pet detection, the Raspberry Pi 4 equipped with the Raspberry Pi 4 BULLSEYE64-bit Debian OS serves as an edge node.

To setup the sensor node, we installed Raspberry Pi OS 64 bit using the Raspberry Pi Imager tool.

The OS can be selected from the ***Choose OS*** option on the tool. The storage i.e., the SD card was set using the ***Choose Storage*** option of the tool. Once both these are set, we can write the OS on the SD card using the ***Write*** Option of the tool.

The SD card was then inserted back into the Raspberry Pi 4 SBC.

3.1.2 Raspberry Pi Camera module v2

The Raspberry Pi Camera module v2, an official module by Raspberry foundation captures high-quality video and images. It is connected to Raspberry and plays a crucial role in capturingframes for live pet detection.

3.2 Software Configuration in Edge Node

3.2.1 Python

Python is widely used high-level programming language for computer vision object detection, and it is supported by on Raspberry Pi with pre-installed OS support. Popularobject detection frameworks like OpenCV offer Python APIs for training and deployingmodels.

3.2.2 OpenCV

OpenCV is a popular and widely used open-source computer vision library, is highly compatible with Raspberry Pi4. It offers a large set of algorithms and functions for image and video processing including object detection.

3.2.3 YOLOv5

YOLOv5(You Only Look Once version 5) is a highly advanced real time object detection model that is widely used in computer vision applications, it requires high powerful GPU. It can be used on the Raspberry Pi 4 for object detection.

It requires an OpenCV library, in our project YOLOv5 is used for to train and detectsPets from the live camera feed.

3.2.4 MinIO Client library

MinIO is a object storage server and MinIO client library provides simple APIs to to assess MinIO and carry out operations such as bucket listing, object creation andlisting.

4 Build A Raspberry Pi Kubernetes cluster with K3S.

4.1 Installation of Raspberry Pi OS

At start we need to download Raspberry Pi Imager and install it on local machine.
Once installed, open the program and select the Raspberry Pi OS (64-bit) as the operating system.Next, insert the microSD card into your computer and choose it as the target for theinstallation.
After selecting the SD card, click on the "Write" button to flash the card with the Raspberry Pi OS version.Once the flashing process is complete, you need to ensure that the host network is available.

For Raspberry Pi-04, you will install and configure Raspberry Pi OS Lite (64-bit) in the same way as before. Each Raspberry Pi will have a unique hostname assigned to it, such as "Kmaster" for the master node and "knode1" for worker node 1. After assigning the hostnames, you should configure the Wi-Fi settings and enableSSH with password authentication.

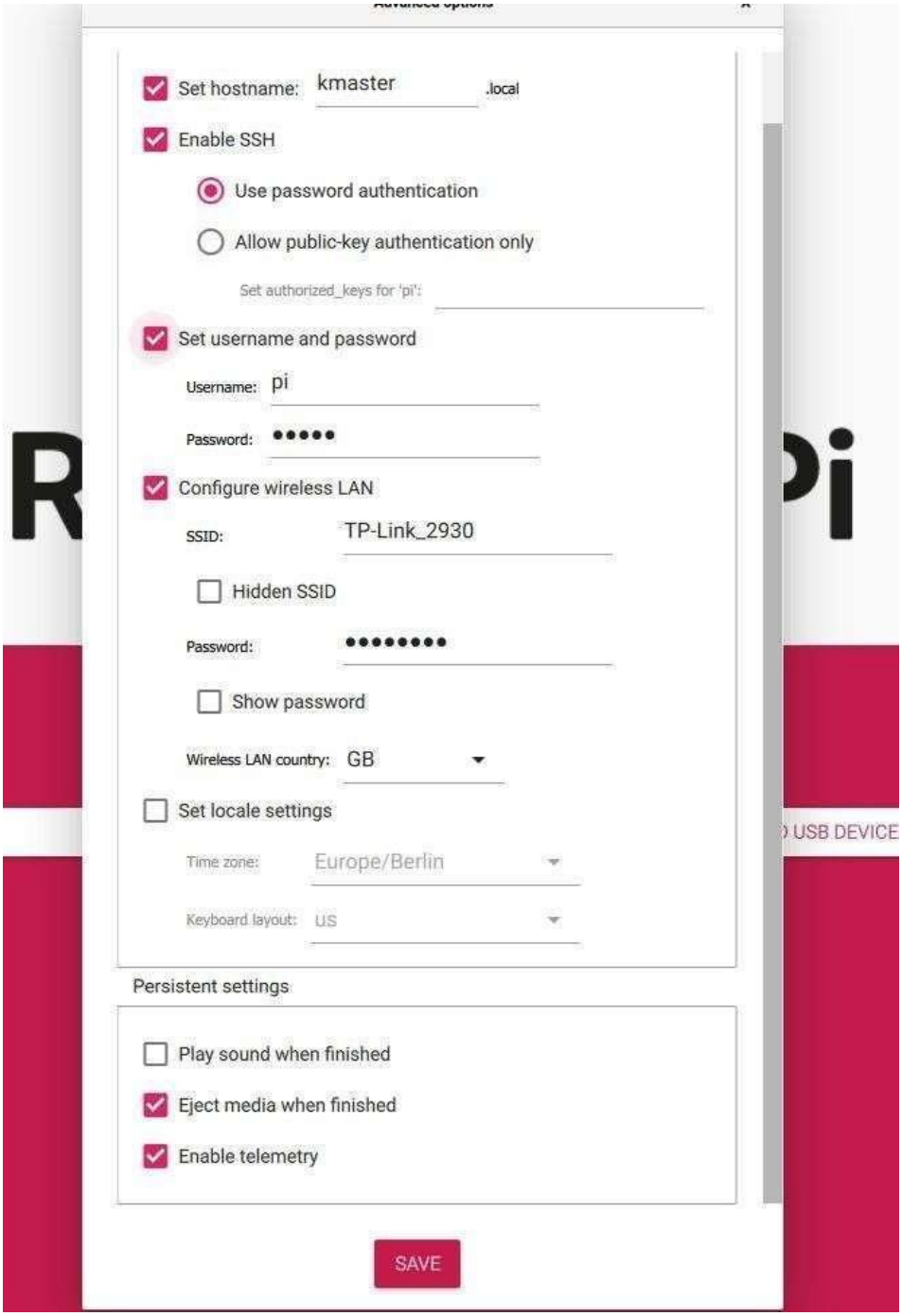


Fig. 2. Network set up.

After installing the operating system, it is important to modify the dhcpcd.conf file located in the "/etc" directory. This file allows you to configure the IP addresses for the sensor node and the master node6. For thesensor node, you should set the IP address as 192.168.0.99, while for all other Raspberry Pi-03 devices, the IPaddress should be set as 192.168.0.100.

Once you have made the necessary configurations in the dhcpcd.conf file, you can power on all the Raspberry Pi devices and verify if they have successfully connected to the current network. To confirm the setup, you can use the ping command. followed by the respective hostnames.

```
ping kmaster.local
ping knode1.local
```

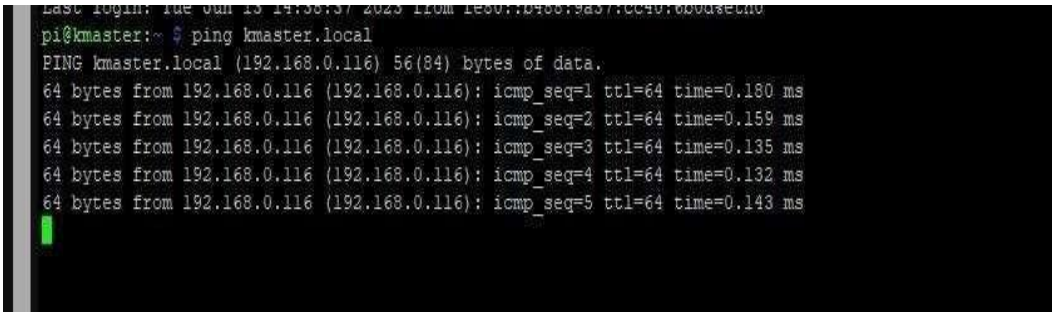


Fig 3. ping Command Execution

4.2 Creating K3s Kubernetes Cluster

The Raspberry Pi cluster utilizes a lightweight Kubernetes distribution called k3s. This distribution, k3s, is specifically designed to cater to IoT solutions that involve devices with limited resources, such as Raspberry Piboards.

To install K3s on the master node, you can execute a given command. This command fetches the K3s installationscript using the `curl` command and executes it as a shell script using `sh`. This process installs the K3s server onthe master node. On the other hand, the worker nodes, which are Raspberry Pis, will have

K3s worker installed on them. Once the installation is complete, all the agent nodes, including the worker nodes, will be registered and connected to the master node. After this step, the token from the master node can be read from the file, With thistoken, k3s can be Installed on worker nodes by the following command:

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

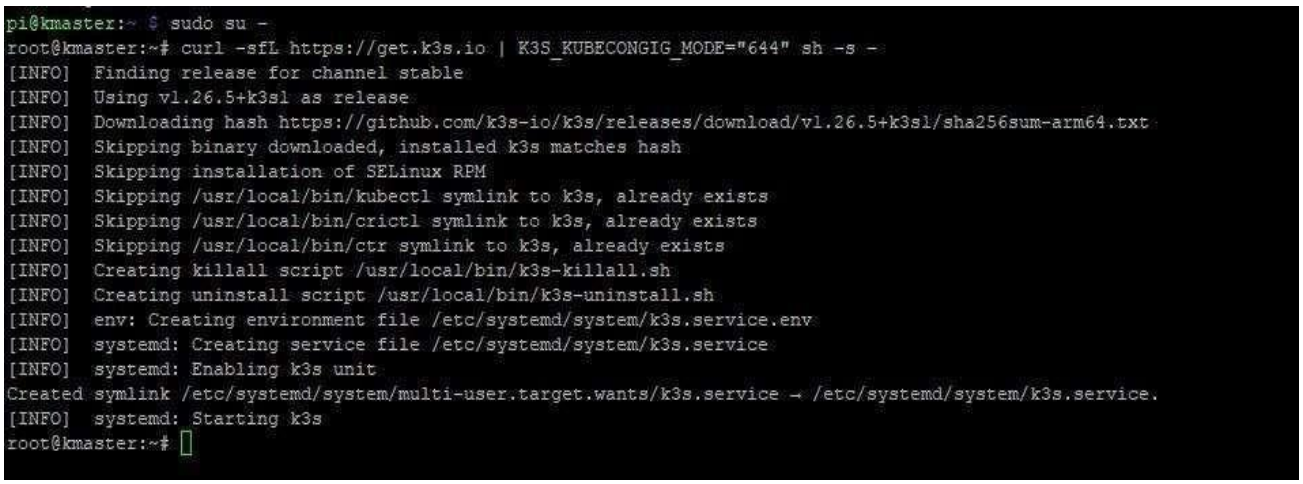
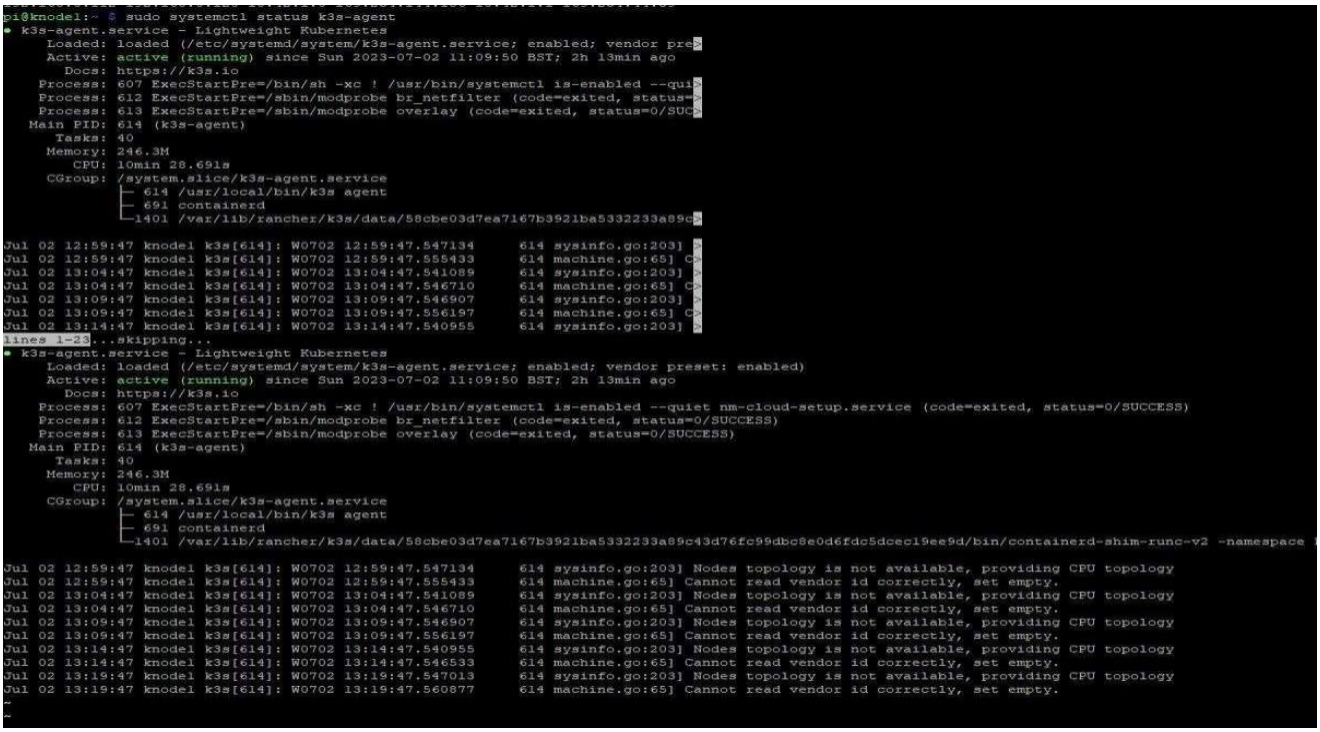


Fig: Command Execution



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 on worker nodes by the following command:

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

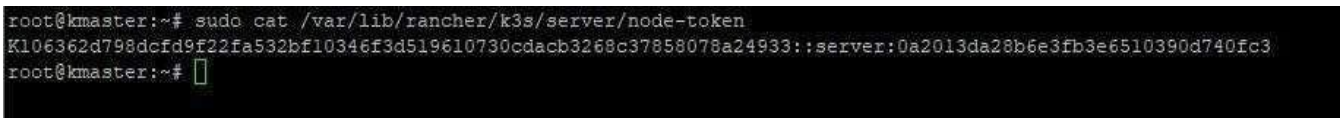


Fig: Token Command Execution

5 Implementation

5.1 Edge Node and Camera Module Set Up for Object Detection

To set up Raspberry Pi 4 as an Edge Node with the Pi Camera Module v2, the following Steps are followed.

Install and boot the Raspbian OS: Raspberry Pi runs bullseye, a 64-bit Debian OSOn an SD Card. It boots the Raspbian Desktop and can be accessed remotely via SSH(PuTTY) or VNC Viewer.

1. **Connect the Pi Camera module v2:** To enable camera module on Raspberry Pi 4 connect it to dedicated camera port and update the configuration using respi-config.
2. **Install OpenCV and Python Packages:** Python is preinstalled in Raspberry Pi 4. Python dependencies like NumPy an OpenCV is installed for object detection.

   To upload the detected image to MinIO, The MinIO Client library for Python is installed. This will allow interact with the MinIO server from the Python code.

   For NumPy installation following commands are used

(a) Install pip package manage.

sudo apt-get install python3-pip

(b) Use pip to install Python.# pip3 install Python

6 Creating the Pets Detector Model

Data Collection and Labeling: Collect images of pets and annotate them to create a training dataset. The dataset should consist of diverse Pets positions, lighting variations and backgrounds.

2) **Dataset Preparation:** The dataset is divided into two sections: a training set and a validation set. The training set is employed to train the model, while the validation set is employed to assess its performance.

3) **Hardware and Software Setup:** A suitable hardware platform, such as a powerful desktop computer or a GPU instance in the cloud, is selected for model training. In this scenario, Google Colab is utilized as a cloud-based computing resource. The essential software components and libraries, including a deep learning framework like PyTorch or Tensor-Flow, along with the YOLOv5 implementation
are installed.

4) **Model Initialization:** The architecture of YOLOv5 and pre-trained weights from a largeimage classification dataset is obtained by downloading them.

5) **Configuration:** The YOLOv5 configuration file is adjusted to match the attributes of the dataset,such as the number of classes, input image size, and the count of anchor boxes.
Hyperparameters for the training process, like the learning rate, batch size, and number of epochs, areconfigured.

6) **Training:** The training process begins by providing the model with the training set images and their respective labels. The progress of the training process is monitored, which involves tracking
the loss function and the accuracy of the validation set. Model weights are saved periodically or whenthe validation set accuracy reaches a specified threshold.

7) **Model Evaluation and Fine-Tuning:** The trained YOLOv5 model is utilized to conduct object detection on a test set of images. The model's performance is evaluated by measuring metrics such as precision, recall, and F1-score. If needed, the model undergoes fine-tuning, and the evaluation process is repeated to enhance its performance. By adhering to these steps, a YOLOv5 model can be trained using a dataset, and its performance can be assessed for the specific task of pet detection. It's a pre trained model that has beenfurther trained using 2576 images, 20 epochs.

7 Using This Model in Raspberry Pi V4 With Raspberry Pi Camera V2

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

2) **Setting up the Sensor Node:** To function as the sensor node, a Raspberry Pi 4 board has beenselected and configured with a 64-bit version of the Raspbian operating system.

3) **Connecting the Camera:** The Raspberry Pi 4 board is connected with a Raspberry PiCamera v2to capture the images of environment.

4) **Transferring the Model and Libraries:** The YOLOv5 model, along with all the essential librariesrequired for its execution, are transferred to the Raspberry Pi 4 board.

5) **Configuring the Raspberry Pi 4:** The setup of the Raspberry Pi 4 board involves configuring it to execute the YOLOv5 model, utilizing the Raspberry Pi Camera v2 as the input source .

6) **Running the Model:** On the Raspberry Pi 4 board, the YOLOv5 model is executed to conduct object detection on the images acquired through the Raspberry Pi Camera v2.

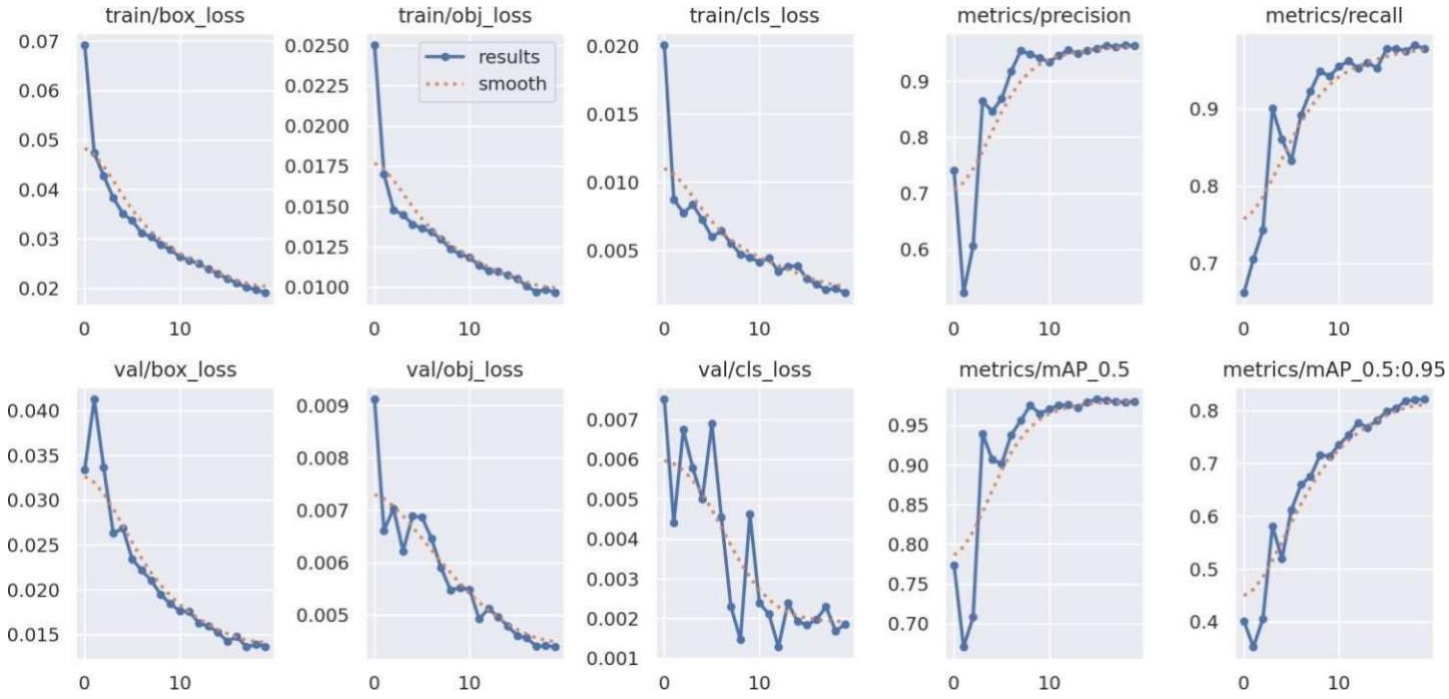
7) **Processing and Communication of Results:** After processing the detection outcomes, the resultsare transmitted to the master node for additional analysis and evaluation. By following these steps, the trained YOLOv5 model can be used on a Raspberry Pi 4 board with a Raspberry Pi Camerav2 for the task of rat detection.

```
train: Scanning /content/drive/MyDrive/yolov5/dataset3/train/labels.cache... 2576 images, 84 backgrounds,
train: Caching images (2.2GB ram): 100% 2660/2660 [00:12<00:00, 205.59it/s]
```

```
      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
      18/19      5.13G    0.01976   0.009863   0.002198         12        640: 100% 167/167 [00:38<00:00,  4.37it/s]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 24/24 [00:07<00:00,
                   all        746        738      0.964      0.983      0.979       0.82

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
      19/19      5.13G    0.01906   0.009646   0.001863          9        640: 100% 167/167 [00:38<00:00,  4.37it/s]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 24/24 [00:07<00:00,
                   all        746        738      0.963      0.978       0.98      0.821

20 epochs completed in 0.256 hours.
Optimizer stripped from runs/train/exp12/weights/last.pt, 14.4MB
Optimizer stripped from runs/train/exp12/weights/best.pt, 14.4MB

Validating runs/train/exp12/weights/best.pt...
Fusing layers...
Model summary: 157 layers, 7015519 parameters, 0 gradients, 15.8 GFLOPs
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 24/24 [00:10<00:00,
                   all        746        738      0.963      0.978       0.98      0.821
                   cat        746        251      0.962       0.98      0.981       0.85
                   dog        746        487      0.963      0.975      0.978      0.792
Results saved to runs/train/exp12
```

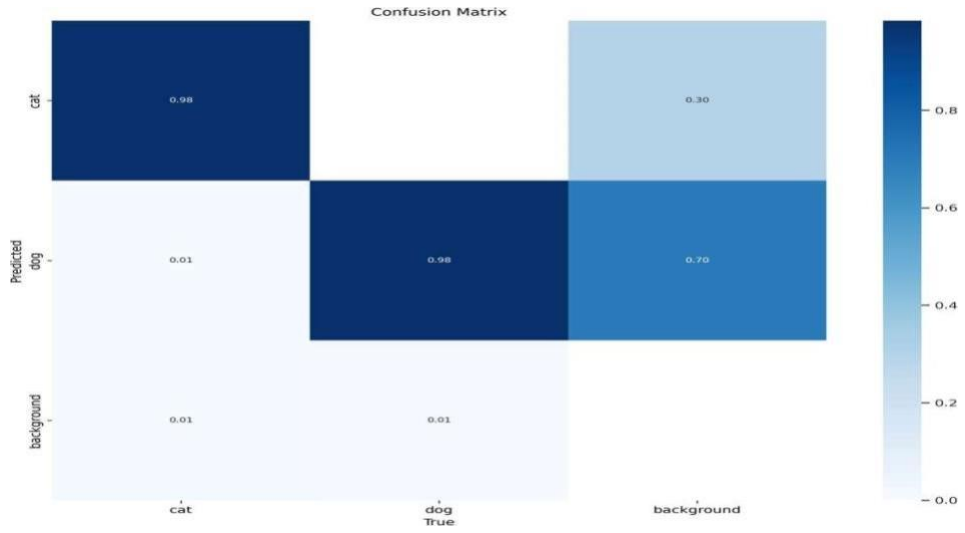
```
tensor([[ 71.51021,  87.13828, 225.61143, 214.06235,   0.78494,   1.00000]])
0: 480x640 1 dog, 1939.4ms
tensor([[ 72.02774, 111.48162, 226.61969, 239.83640,   0.83531,   1.00000]])
0: 480x640 1 dog, 1941.8ms
tensor([[114.10225, 124.24533, 217.07570, 228.89560,   0.78873,   0.00000],
        [224.51062,  94.90694, 361.40222, 208.75429,   0.52080,   1.00000]])
0: 480x640 1 cat, 1 dog, 2148.4ms
tensor([[306.36926,  40.06342, 445.10657, 147.89743,   0.80572,   1.00000],
        [201.75195,  61.43716, 297.75153, 157.73923,   0.74263,   0.00000]])
0: 480x640 1 cat, 1 dog, 1897.8ms
tensor([[280.08588,   1.71737, 418.63910, 117.02032,   0.78367,   1.00000],
        [170.12122,  21.48952, 271.56097, 123.18181,   0.65133,   0.00000]])
0: 480x640 1 cat, 1 dog, 1875.3ms
tensor([[1.63480e+02, 9.57100e-01, 2.90347e+02, 1.07180e+02, 4.03237e-01, 0.00000e+00],
        [2.74918e+02, 6.03977e-01, 4.26312e+02, 9.28575e+01, 3.00563e-01, 1.00000e+00]])
0: 480x640 1 cat, 1 dog, 1857.7ms
tensor([[179.18550,   1.07813, 282.13837, 102.85790,   0.63096,   0.00000]])
0: 480x640 1 cat, 1878.8ms
tensor([[ 2.82992e+02, -3.72150e-01,  4.28774e+02,  7.60848e+01,  5.76285e-01,  1.00000e+00]])
0: 480x640 1 dog, 2091.6ms
tensor([[ 2.83460e+02, -2.01641e-01,  4.30869e+02,  7.68182e+01,  6.93186e-01,  1.00000e+00]])
0: 480x640 1 dog, 1869.3ms
tensor([[ 2.91990e+02, -2.87811e-01,  4.38346e+02,  7.88094e+01,  7.19789e-01,  1.00000e+00]])
0: 480x640 1 dog, 1915.8ms
```

Fig : Image Detection



Confusion Matrix: The confusion matrix is a commonly used tool to assess the effectiveness of an object detection model. It consists of two dimensions: actual classes and predicted classes. The actual classes refer to thetrue labels of the objects in the test dataset, while the predicted classes are the labels assigned by the model during its inference process.

By examining the confusion matrix, we can observe that the model has made accurate predictions in most cases, with only a few minor errors. These errors indicate that the model's predicted bounding boxes slightly deviate from the ground truth objects. The concept of "loss" comes into play here, as it quantifies how well the predictedbounding box areas cover the actual objects.

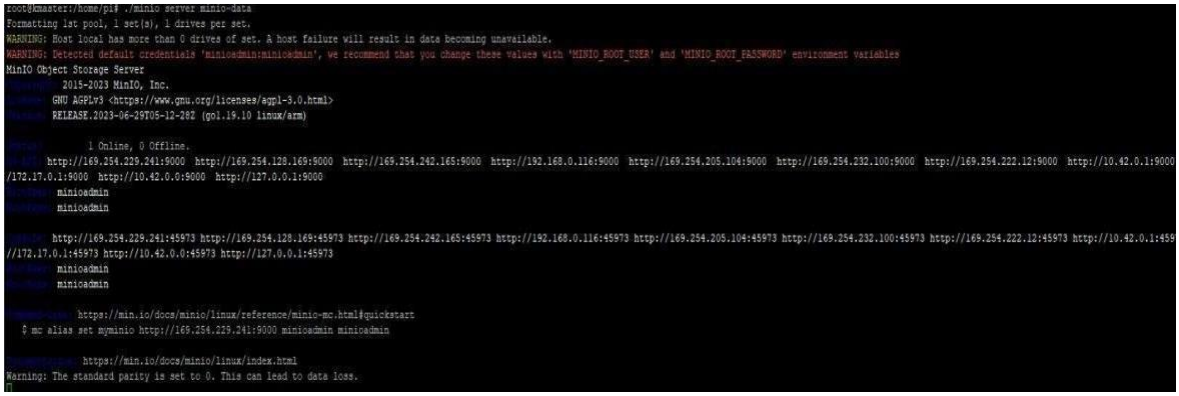


8 Docker Images

To run the program on a cluster, it is necessary to generate docker images. The difficulty lies in creating docker images specifically tailored for the raspberry pi CPU architecture. To address this challenge, Docker is installed on a Raspberry Pi 4, and the docker images are generated on this device. One of these images, associated with the program running on the sensor node, will be executed locally on the Raspberry Pi 4. The other two images, intended for the logger program on the back end and the front-end website, will be uploaded to Docker Hub for deployment on the cluster.

9 MinIO Object Storage Deployment

MinIO was selected as the database for the image detection software because it is specifically designed to handle the storage of objects, such as images, which is a requirement for this software. Several factors weretakeninto consideration before choosing MinIO. Its scalability, resilience, and the availability of a RESTfulAPI customized for use in Kubernetes clusters are some of the beneficial features. To deploy MinIO, a deployment artifact is utilized, and a service is created to provide access to the command line interface.
MinIO can be set up in various modes within the cloud network. In this project, the Single Node SingleDrive mode was employed.
The Multi-node multi-drive mode requires more powerful hardware to ensure synchronization betweenmultipleinstances of the database running on the cluster.



10. Backend and Frontend Technology :

In order to develop both the backend and frontend components of the application, we opted for Next.js, a meta framework for React.js that offers full-stack capabilities. Next.js allows us to build the frontend usingReact.js and provides an abstraction for creating REST endpoints. One notable advantage of Next.js is its file-based routing system, which reduces the need for extensive configuration compared to other frameworks.

Another significant benefit of choosing Next.js is the simplified deployment process. By using Next.js, we can manage the deployment of a single application hosted on the same server instance and URL. In our case, the application is accessible through http://localhost:3000, while all REST endpoints are accommodated under http://localhost:3000/api.

i. Backend and Frontend Functionality :

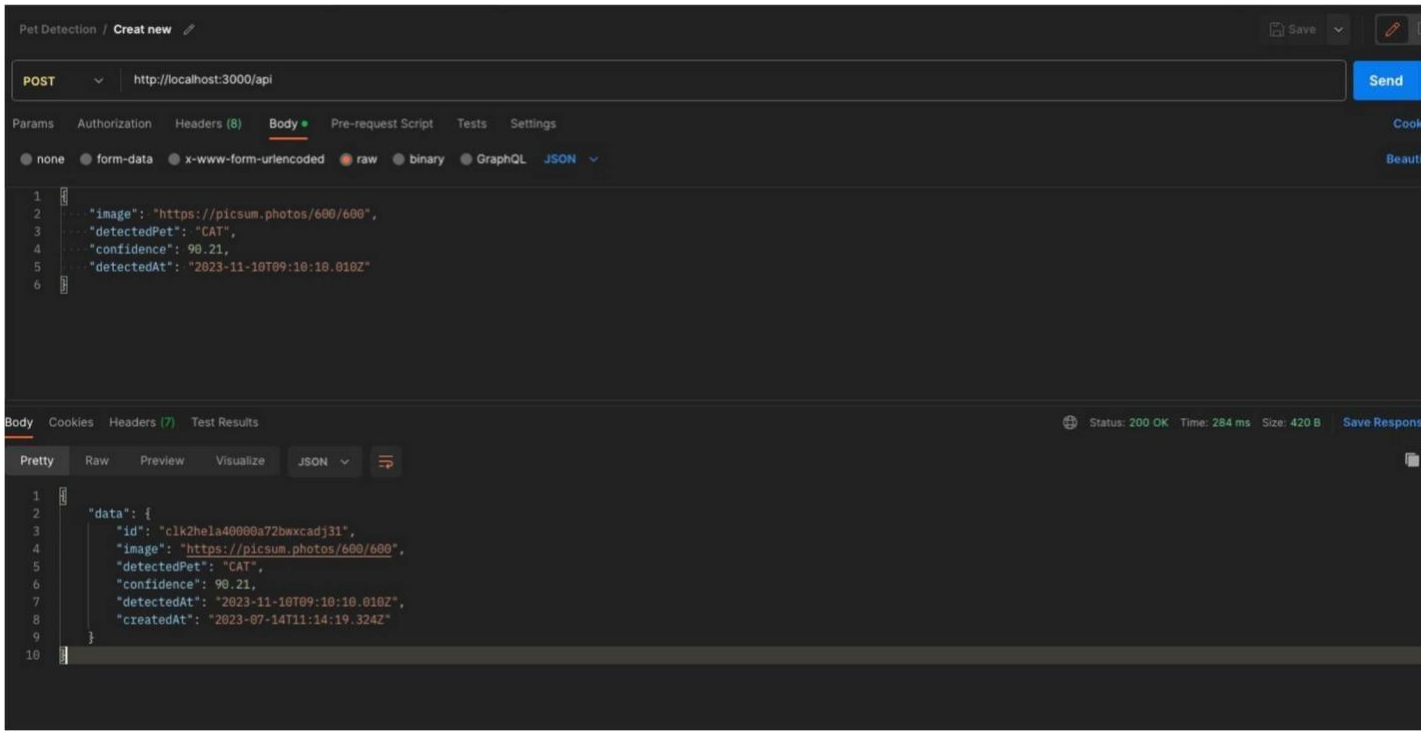
The frontend component consists of a single route that displays a list of all the detected pets. On the other hand, the backend component provides three endpoints:

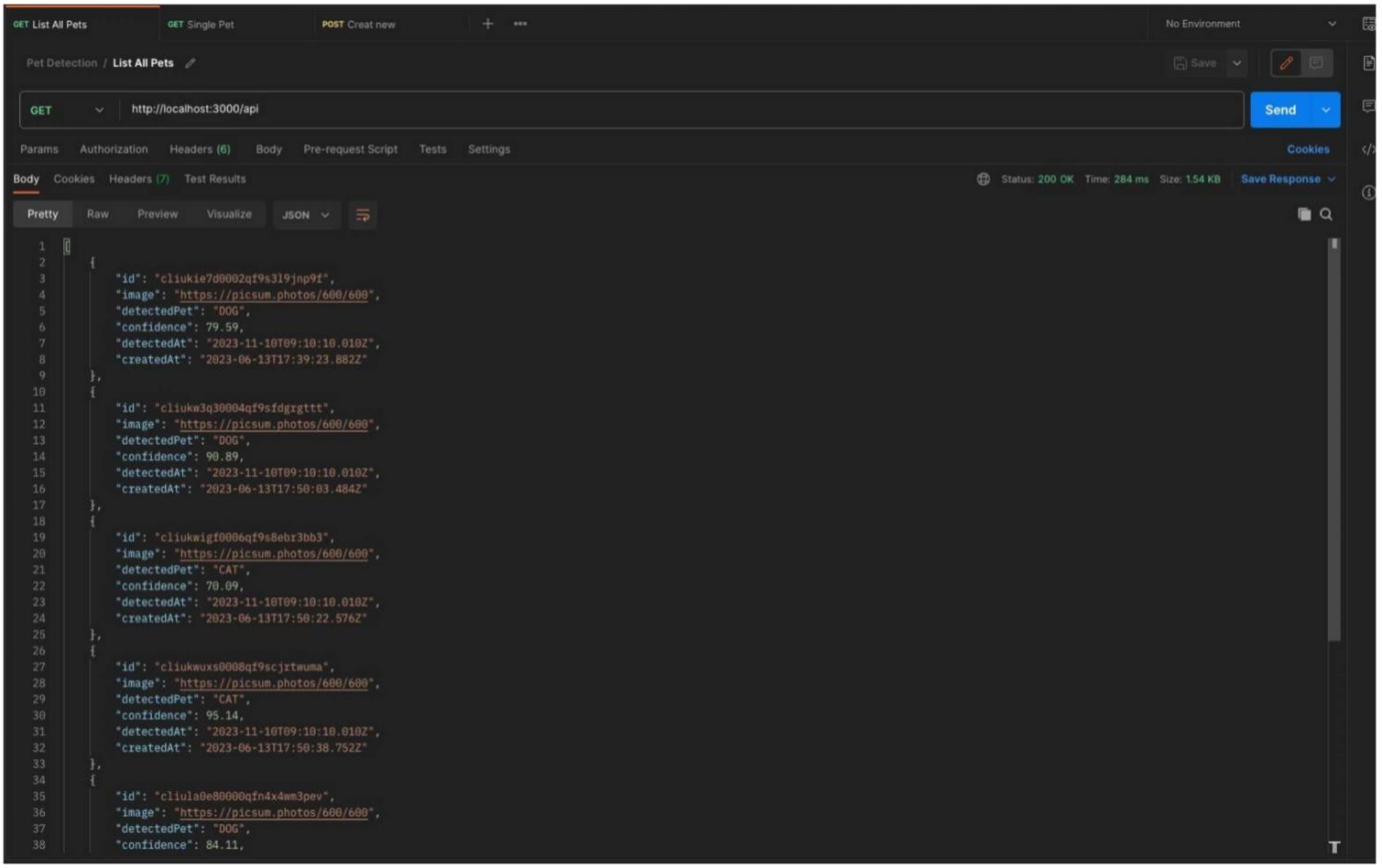
EndPoints	Descriptions
GET /api	Returns a list of all detected pets along with their corresponding MinIO URLs.
Post /api	Stores an image and related data in MinIO. Saves the MinIO URL and associated data in thedatabase.
GET /api[id]	Retrieves information for a specific instance of a detected pet from the database. Identifiedby "id" parameter.

ii. Database Technology :

For the database, we employed a Cloud Database service called PlanetScale, which utilizes MYSQL and Vitess for hosting and managing database deployments. PlanetScale also supports horizontal scaling, which was particularly advantageous given the limited performance capabilities of the Raspberry Pi 3. By offloading our database to PlanetScale, we were able to free up resources on the Raspberry Pi 3 for other services.

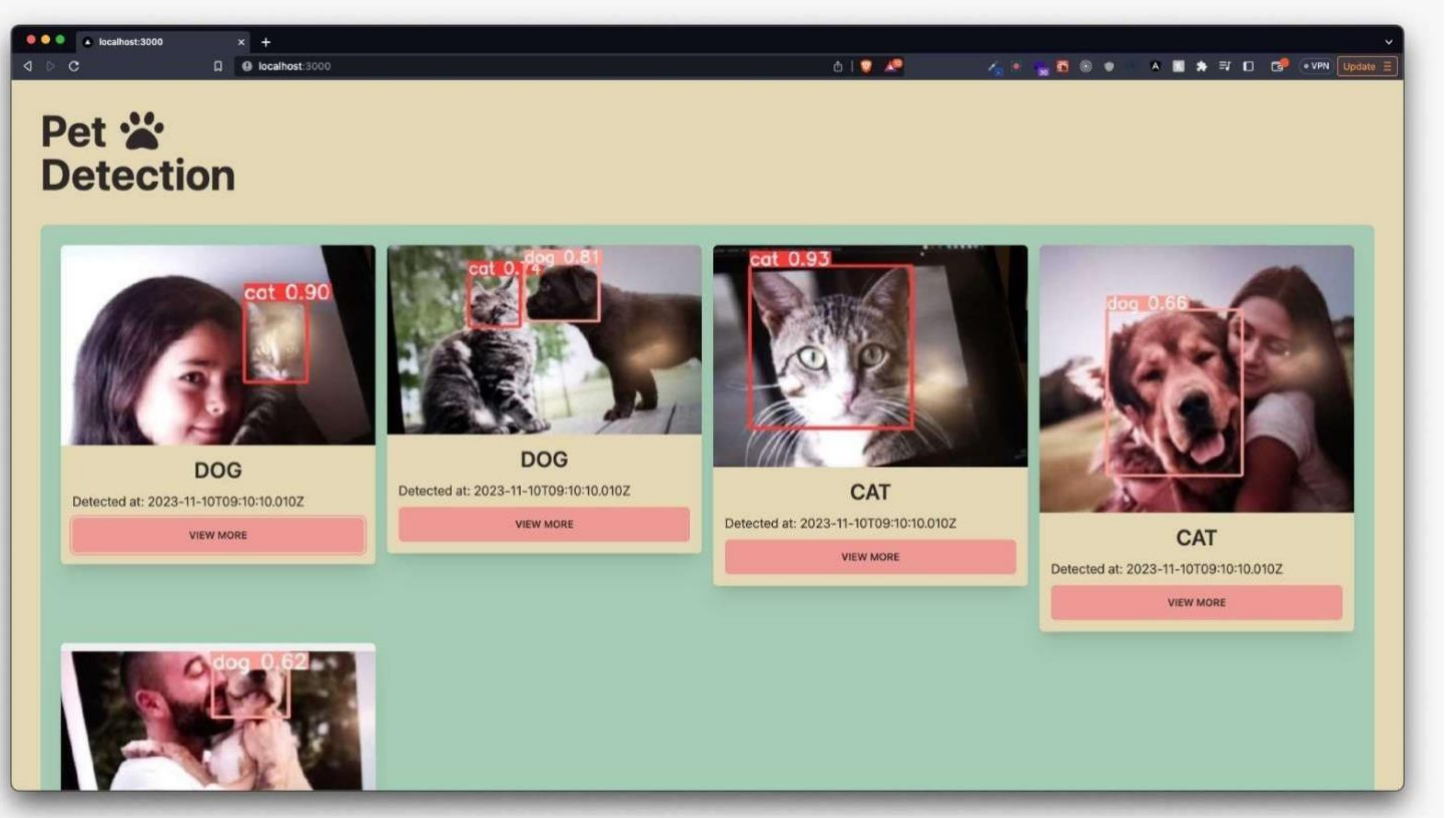
Additionally, we leveraged Prisma, a type-safe ORM for TypeScript that supports various SQL databases. Prisma eliminates the need to write manual queries and mitigates the risk of SQL attacks, such as query injection. This integration with our backend was seamless, resulting in a backend comprised of only two files and approximately 40 lines of code, including the necessary boilerplate code.




GET List All Pets
GET Single Pet
POST Creat new
No Environment
Pet Detection / List All Pets
Save
GET
http://localhost:3000/api
Send
Params
Authorization
Headers (6)
Body
Pre-request Script
Tests
Settings
Cookies
Body
Cookies
Headers (7)
Test Results
Status: 200 OK
Time: 284 ms
Size: 1.54 KB
Save Response
Pretty
Raw
Preview
Visualize
JSON
[
{
"id": "cliukie7d0002qf9s3l9jnp9f",
"image": "https://picsum.photos/600/600",
"detectedPet": "DOG",
"confidence": 79.59,
"detectedAt": "2023-11-10T09:10:10.010Z",
"createdAt": "2023-06-13T17:39:23.882Z"
},
{
"id": "cliukw3q30004qf9sfdgrgttt",
"image": "https://picsum.photos/600/600",
"detectedPet": "DOG",
"confidence": 90.89,
"detectedAt": "2023-11-10T09:10:10.010Z",
"createdAt": "2023-06-13T17:50:03.484Z"
},
{
"id": "cliukwigf0006qf9s8ebr3bb3",
"image": "https://picsum.photos/600/600",
"detectedPet": "CAT",
"confidence": 70.09,
"detectedAt": "2023-11-10T09:10:10.010Z",
"createdAt": "2023-06-13T17:50:22.576Z"
},
{
"id": "cliukwuxs0008qf9scjrtwuma",
"image": "https://picsum.photos/600/600",
"detectedPet": "CAT",
"confidence": 95.14,
"detectedAt": "2023-11-10T09:10:10.010Z",
"createdAt": "2023-06-13T17:50:38.752Z"
},
{
"id": "cliula0e80000qfn4x4wm3pev",
"image": "https://picsum.photos/600/600",
"detectedPet": "DOG",
"confidence": 84.11,


GET List All Pets
GET Single Pet
POST Creat new
No Environment
Pet Detection / Single Pet
Save
GET
http://localhost:3000/api/cliukwuxs0008qf9scjrtwuma
Send
Params
Authorization
Headers (6)
Body
Pre-request Script
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit

Results:


localhost:3000
Pet
Detection
cat 0.90
DOG
Detected at: 2023-11-10T09:10:10.010Z
VIEW MORE
cat 0.74
dog 0.81
DOG
Detected at: 2023-11-10T09:10:10.010Z
VIEW MORE
cat 0.93
CAT
Detected at: 2023-11-10T09:10:10.010Z
VIEW MORE
dog 0.66
CAT
Detected at: 2023-11-10T09:10:10.010Z
VIEW MORE
dog 0.62

REFERENCES:

[1] [Online]. Available: ”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
[3] [Online]. Available: ”https://saintcoder.wordpress.com/2017/08/14/ sharing-internet-connection-to-raspberry-pis-wired-to-local-network/
[4] [Online]. Available: https://medium.com/datadriveninvestor/how-do-you-use-yolo-to-detect-objects-8bb634ec7d2c
[5] [Online]. Available: https://jooskorstanje.com/super_simple_yolo_notebook.html
[6] [Online]. Available: https://github.com/ultralytics/yolov5
[7] [Online]. Available: https://pytorch.org/hub/ultralytics_yolov5/
[8] [Online]. Available: ”https://www.eclipse.org/paho/index.php?page=clients/python/index.php
[9] [Online] https://min.io/docs/minio/kubernetes/upstream/