FRANKFURT UNIVERSITY OF APPLIED SCIENCES

Cloud Computing SS 2023

Group Project: Pet Detection

Group 6

Submitted by:	Khalid Butt (1276464),
	Ramses Ntsinda Nde (1416279),
	Sebastian Wagner (1414336),
	Bernard Wenigenrath (1414556),
	Saman Solimany (1270677)
Supervisor:	Prof. Dr. Christian Baun
Submission Date:	14.07.2023

Table of contents

List of figures

1. Introduction

The following project is part of the Cloud Computing module. The aim is to develop an edge computing solution for automatic pet detection. The project is based on single board computers (Raspberry Pi) and a camera module.

The following documentation describes the work packages carried out.

A total list of hardware components is provided:

- 1x Raspberry Pi4
- 4x Raspberry Pi3
- Raspberry Pi Camera Module 8MP v2
- 16 bolts for Raspberry Pi construction
- Anker PowerPort 10 (with 10 USB ports)
- 5 LAN cable
- 4 Micro USB cable
- 4 MicroSD cards
- 2 MicroSD standard SD card adapter

1.1.System Architecture

Based on the hardware, the architecture of the edge computing solution is indirectly given (see. Fig. 1). The Raspberry Pi4 works as a sensor node with the camera module attached to classify pets (cats, dogs and golden hamsters). When a pet gets detected by the camera module, the camera module takes a picture. After that the picture and the metadata get sent to the backend. For the backend four Raspberry Pi3 build a Kubernetes Cluster with k3s distribution. Within the Kubernetes Cluster, three Raspberry Pi3 are the worker nodes. The fourth Raspberry Pi3 controls the four worker nodes as a master node. The Kubernetes Cluster should be also able to store the picture and metadata to the object storage MinIO. For displaying the images, a web application is used which displays log information, event messages and an event map with evidence and timestamps as a Docker container.

The cluster is designed as a high availability cluster. Github is used as the collaboration tool. The corresponding Github repository can be found at the following link: https://github.com/ssolimany/cc-ss23-group5-pet-detector .

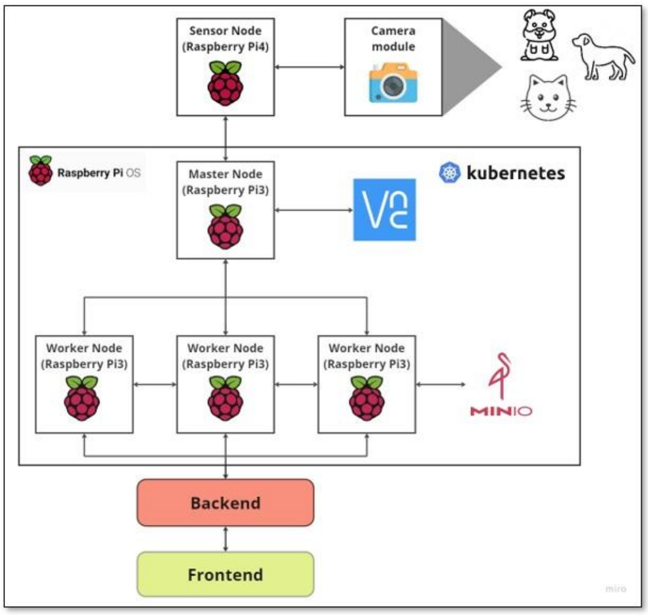

Sensor Node (Raspberry Pi4)
Camera module
Raspberry Pi OS
Master Node (Raspberry Pi3)
kubernetes
Worker Node (Raspberry Pi3)
Worker Node (Raspberry Pi3)
Worker Node (Raspberry Pi3)
MINIO
Backend
Frontend

Figure 1: Architecture of our project

1.2. Physical setup

The four Raspberry Pi`s are physically connected with metal bolts so that they are stacked on top of each other. In the next step all Raspberry Pis get connected to the power supply using a micro-USB cable. The camera module gets attached to the Raspberry Pi4. After those steps, the setup looks like the following (see. Fig. 2):

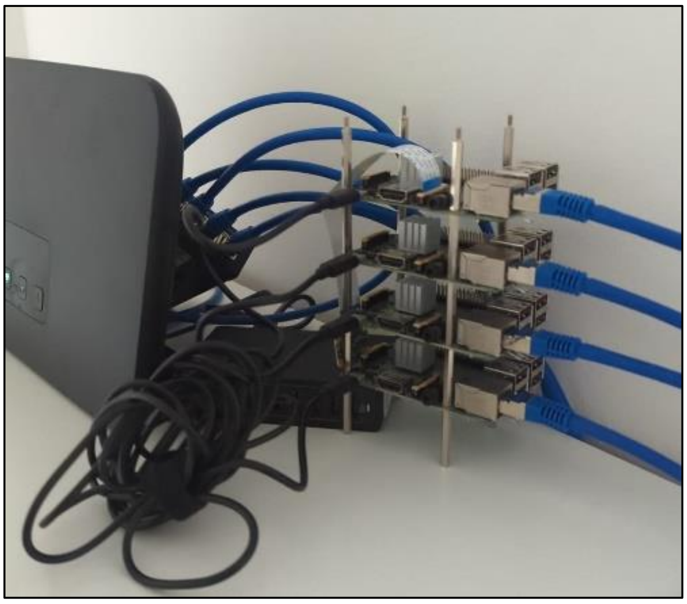
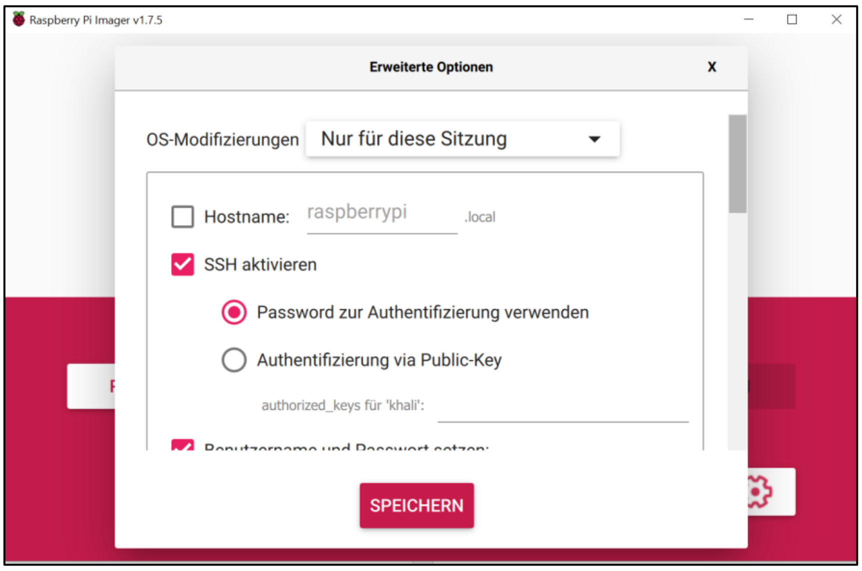


Figure 2: Setup of the Raspberry Pis

2. Setup Sensor Node

To setup the sensor node, Raspberry Pi OS (64-bit) is chosen as the operating system. This version enables desktop view. To install the Raspberry Pi OS on the Raspberry Pi4, the SD card gets flashed with the operating system using the Raspberry Pi Imager. After the SD card gets flashed with the operating system, the SD card gets connected to the Raspberry Pi4. The operating system gets fully installed on the Raspberry Pi4 with the first booting. During that the Raspberry Pi4 gets also connected to the local Wi-Fi.

Since the Raspberry Pi4 only has a mini HDMI port and the proper cable is not available, SSH gets activated on the sensor node. The SSH option can be activated during the installation process of the operation system in the advanced settings menu (see. Fig. 3). Through SSH, VNC gets enabled to start the remote desktop view.


Raspberry Pi Imager v1.7.5

Erweiterte Optionen

OS-Modifizierungen: Nur für diese Sitzung

☐ Hostname: raspberrypi.local

☑ SSH aktivieren

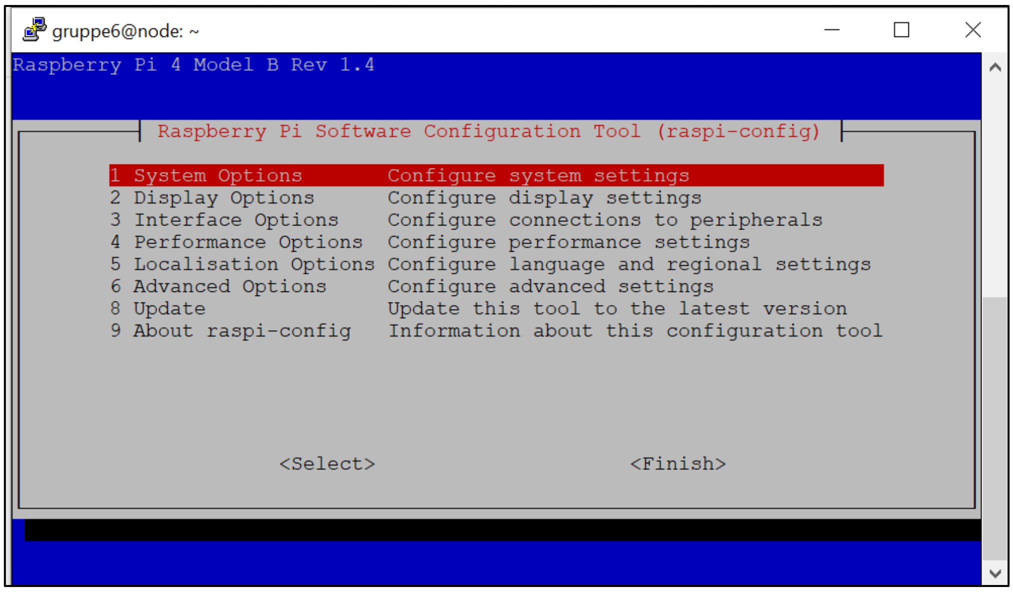
[✓ Password zur Authentifizierung verwenden] Authentifizierung via Public-Key

authorized_keys für 'khali':

☑ Benutzername und Passwort setzen:

Figure 3: Activating SSH on sensor node

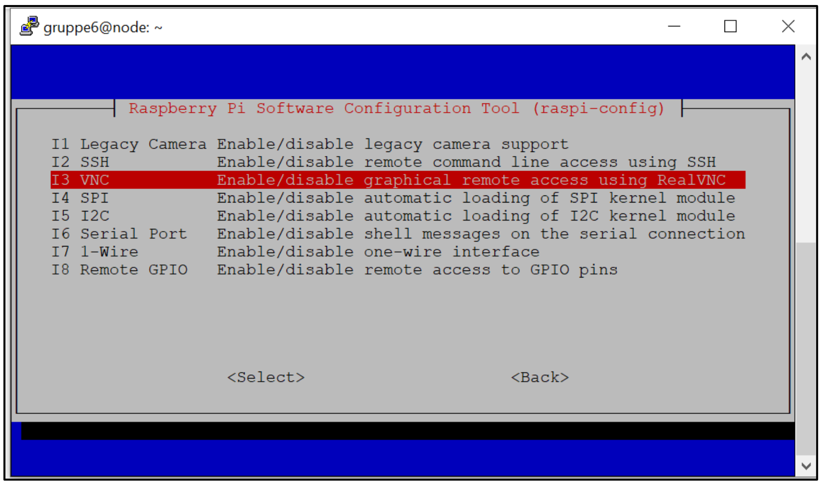
To make setting changes the command *raspi-config* gets executed on the Raspberry Pi4 and the third option *Interface Options* gets selected (see. Fig. 4).



```
gruppe6@node: ~
Raspberry Pi 4 Model B Rev 1.4

        Raspberry Pi Software Configuration Tool (raspi-config)

  1 System Options       Configure system settings
  2 Display Options      Configure display settings
  3 Interface Options    Configure connections to peripherals
  4 Performance Options  Configure performance settings
  5 Localisation Options Configure language and regional settings
  6 Advanced Options     Configure advanced settings
  8 Update               Update this tool to the latest version
  9 About raspi-config   Information about this configuration tool

               <Select>                     <Finish>
```

Figure 4: Raspberry Pi4 settings

The next step enables VNC (see. Fig. 5).



```
gruppe6@node: ~

        Raspberry Pi Software Configuration Tool (raspi-config)

   I1 Legacy Camera Enable/disable legacy camera support
   I2 SSH           Enable/disable remote command line access using SSH
   I3 VNC           Enable/disable graphical remote access using RealVNC
   I4 SPI           Enable/disable automatic loading of SPI kernel module
   I5 I2C           Enable/disable automatic loading of I2C kernel module
   I6 Serial Port   Enable/disable shell messages on the serial connection
   I7 1-Wire        Enable/disable one-wire interface
   I8 Remote GPIO   Enable/disable remote access to GPIO pins

               <Select>                     <Back>
```

Figure 5: Enabling VNC

Now the Raspberry Pi4 can get controlled remotely by another computer using the “VNC Viewer” software. When the connection is established, it looks like the following (see. Fig. 6):

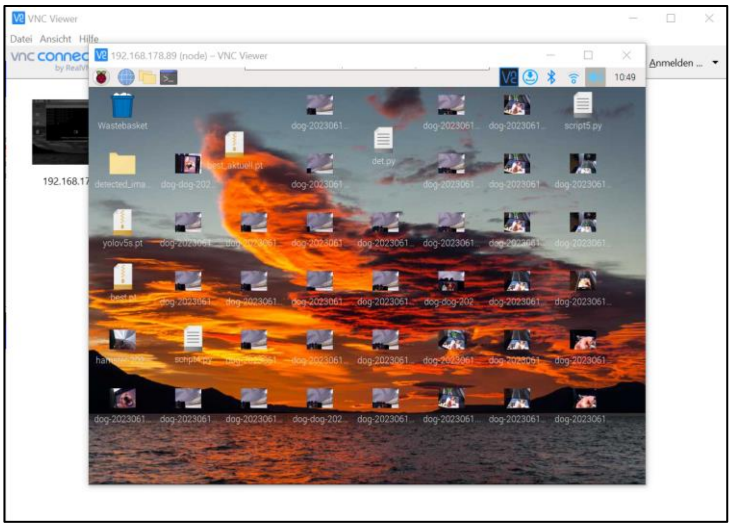


Figure 6: Raspberry Pi4 remote view

After the remote desktop view got set up, the camera module gets connected to the Raspberry Pi4 (see. Fig. 7 & 8).

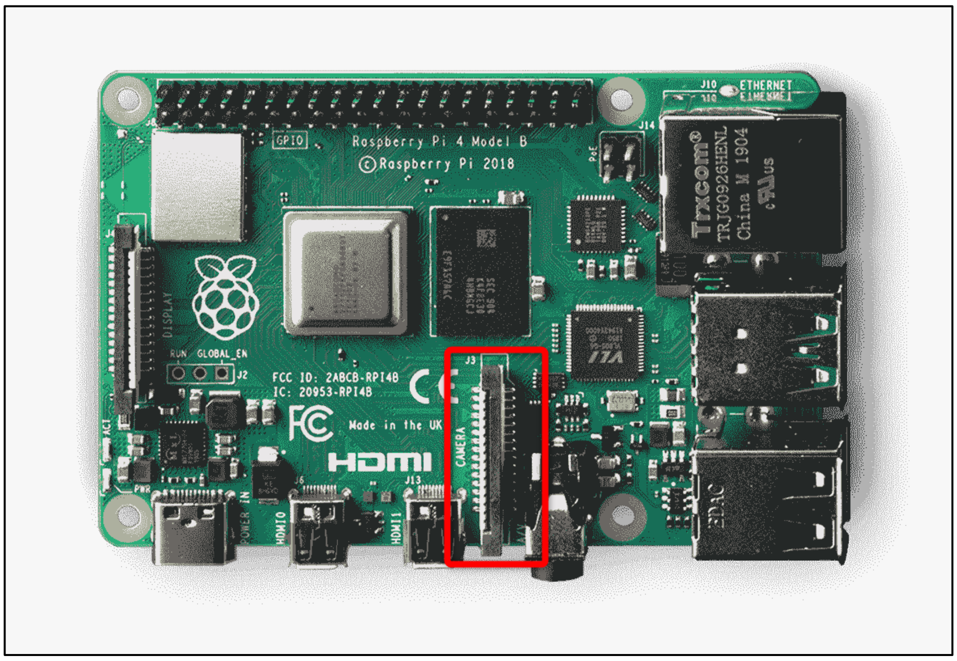


Figure 7: Raspberry Pi 4 camera slot

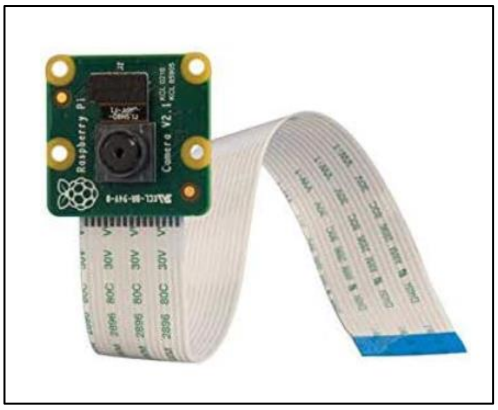


Figure 8: Raspberry Pi camera

The next step is to enable the camera in the Raspberry PI4's settings. As with enabling the VNC option, the camera is enabled in the *interface options* under the name *Legacy Camera* (see. Fig. 9).

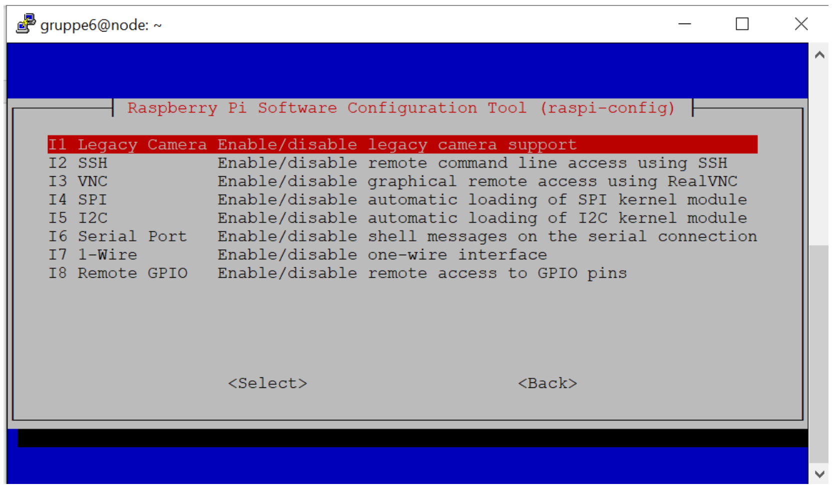
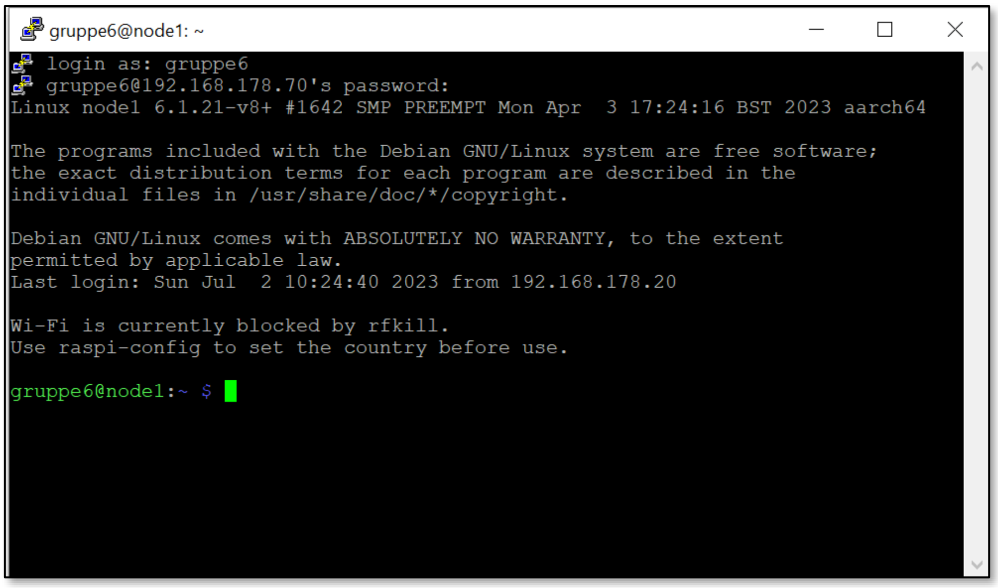

```
gruppe6@node: ~

         Raspberry Pi Software Configuration Tool (raspi-config)

   I1 Legacy Camera Enable/disable legacy camera support
   I2 SSH           Enable/disable remote command line access using SSH
   I3 VNC           Enable/disable graphical remote access using RealVNC
   I4 SPI           Enable/disable automatic loading of SPI kernel module
   I5 I2C           Enable/disable automatic loading of I2C kernel module
   I6 Serial Port   Enable/disable shell messages on the serial connection
   I7 1-Wire        Enable/disable one-wire interface
   I8 Remote GPIO   Enable/disable remote access to GPIO pins


                 <Select>                       <Back>
```

Figure 9: Enabling camera

Before setting up the Kubernetes Cluster on the four Raspberry Pi3´s the operating system needs to get installed. Those steps are very similar to the setup of the sensor node. The only difference is the used version of the Raspberry Pi OS. The four Raspberry Pi3´s are flashed with Raspberry Pi OS Lite (64-bit), the version without the desktop view, to increase performance.

2.1.Setup Kubernetes cluster

After the installation of the operating system on all Raspberry Pi3's, the installation of the Kubernetes cluster follows. The first step is the enabling of SSH for convenience purposes using the software Putty (see. Fig. 10). One Raspberry Pi3 is used as the master node, while the three Raspberry Pi3 have the role as worker nodes.



```
login as: gruppe6
gruppe6@192.168.178.70's password:
Linux node1 6.1.21-v8+ #1642 SMP PREEMPT Mon Apr  3 17:24:16 BST 2023 aarch64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sun Jul  2 10:24:40 2023 from 192.168.178.20

Wi-Fi is currently blocked by rfkill.
Use raspi-config to set the country before use.

gruppe6@node1:~ $
```

Figure 10: Remote access with Putty

The following commands are executed to set up the K3S cluster:

sudo apt update && sudo apt upgrade -y

This command gets executed on all Raspberry Pi´s to update and upgrade the operating system. The following command runs on the master node:

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

Even if the installation triggers an error, it will not stop. Then execute:

sudo nano / boot / cmdline . txt

This command opens a text document 'cmdline.txt' in the text editor. This document contains one line and must be extended by the following command without leaving the first line:

cgroup_memory =1 cgroup_enable = memory

The node token must be received before the master node is rebooted. This string is important for the installation of the worker node. Therefore execute:

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

The Master node must now be rebooted.

sudo reboot

After this reboot, the master node is fully installed. The next steps will install the worker nodes. The K3S installation of the worker nodes is very similar to the installation on the master node. The K3S cluster is installed on the worker nodes using the following command. Make sure that the IP address of the master node and the token are entered correctly in the marked fields.

*curl - sfL https :// get . k3s . io | K3S_URL = https :// < masternode_IP_Address >:**6443** K3S_TOKEN = < **masternode_token** > sh –*

During the installation, an error will occur again, but like the last time, it will not interrupt the installation process.

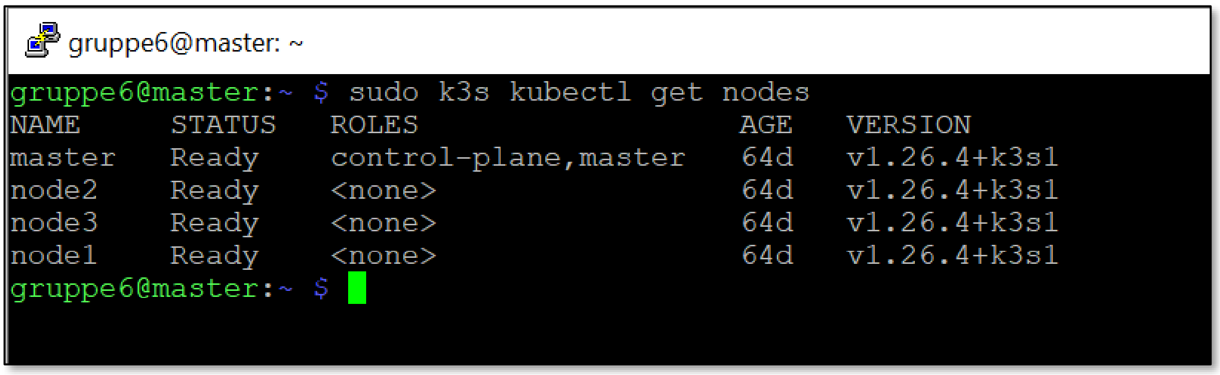
The worker node is then rebooted:

sudo reboot

The installation process for the worker node is repeated for the two outstanding worker nodes. Once the K3S cluster is installed on the master node and the three worker nodes, the installation status can be checked.

sudo k3s kubectl get nodes

The installation is successful if all nodes are in *Ready* state (see. Fig. 11).



```
gruppe6@master:~ $ sudo k3s kubectl get nodes
NAME      STATUS   ROLES                  AGE   VERSION
master    Ready    control-plane,master   64d   v1.26.4+k3s1
node2     Ready    <none>                 64d   v1.26.4+k3s1
node3     Ready    <none>                 64d   v1.26.4+k3s1
node1     Ready    <none>                 64d   v1.26.4+k3s1
gruppe6@master:~ $
```

Figure 11: Overview about the Kubernetes cluster

2.2.Integration MinIO

For the next step, MinIO, an object storage solution, gets installed on Kubernetes cluster.

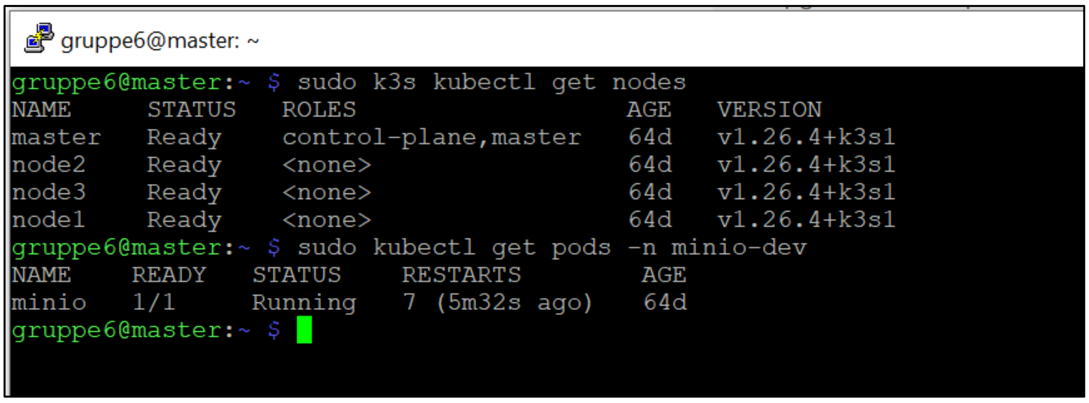
For the installation the official documentation is used: https://min.io/docs/minio/kubernetes/up-stream/index.html. The installation of MinIO has some requirements. The Kubernetes cluster must have at least one worker node that has a locally attached drive. Furthermore, a local kubectl installation must be configured to create and access resources on the target Kubernetes deployment. For the installation of MinIO a YAML-file is required on the Kubernetes Cluster. To download the YAML-file to the host machine the following code gets executed:

curl https://raw.githubusercontent.com/minio/docs/master/source/extra/examples/minio-dev.yaml -O

The subsequent steps are described in detail on the website mentioned above. To check if MinIO got installed correctly on the Kubernetes cluster, the following code gets executed:

sudo kubectl get pods -n minio-dev

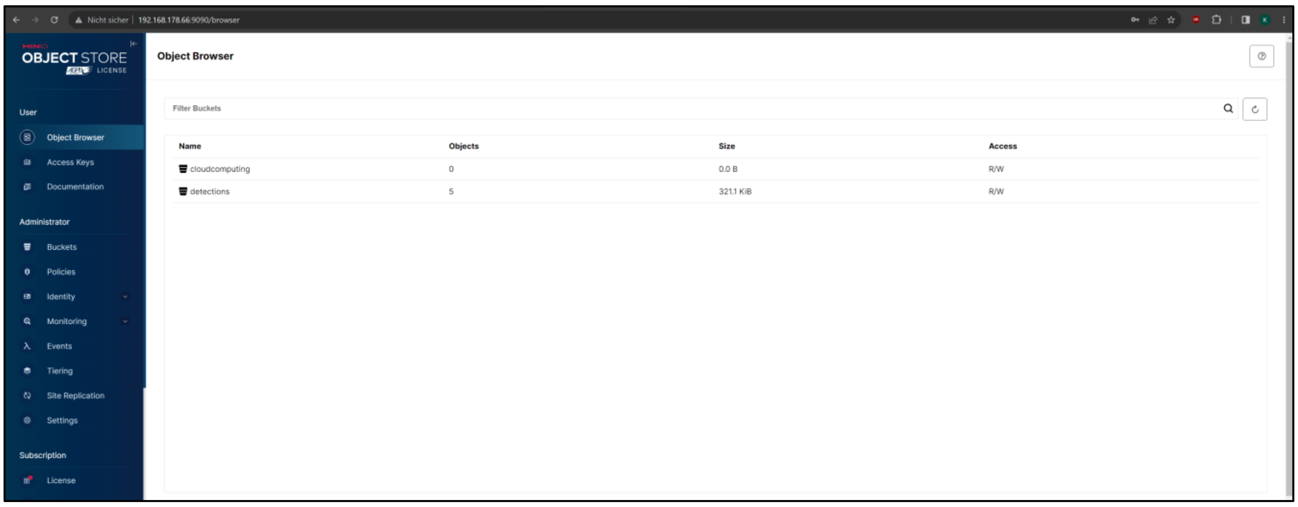
It should look like the following (see. Fig. 12):



```
gruppe6@master: ~
gruppe6@master:~ $ sudo k3s kubectl get nodes
NAME      STATUS   ROLES                  AGE   VERSION
master    Ready    control-plane,master   64d   v1.26.4+k3s1
node2     Ready    <none>                 64d   v1.26.4+k3s1
node3     Ready    <none>                 64d   v1.26.4+k3s1
node1     Ready    <none>                 64d   v1.26.4+k3s1
gruppe6@master:~ $ sudo kubectl get pods -n minio-dev
NAME     READY   STATUS    RESTARTS        AGE
minio    1/1     Running   7 (5m32s ago)   64d
gruppe6@master:~ $
```

Figure 12: MinIO status

MinIO can be reached via the following IP address: http://192.168.178.66:9090/ and looks like the following (see. fig. 13):

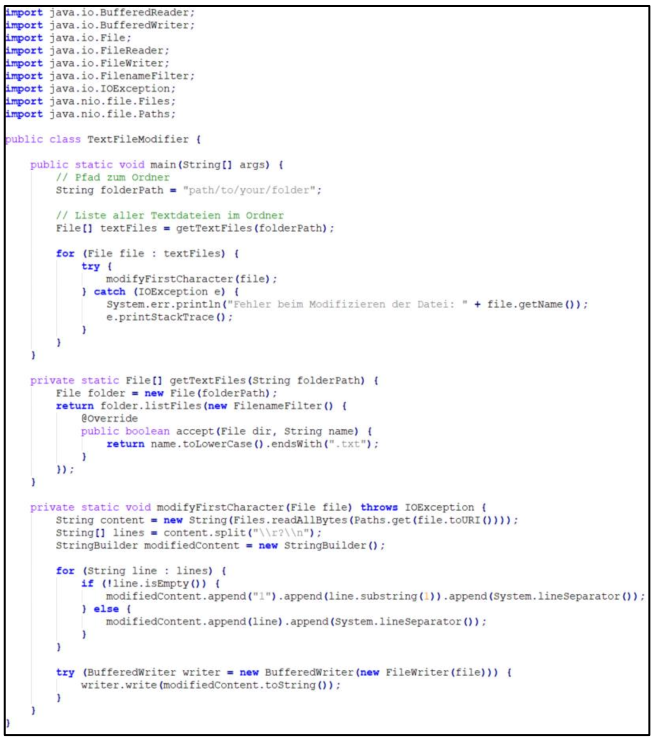


Name	Objects	Size	Access
cloudcomputing	0	0.0 B	R/W
detections	5	321.1 KiB	R/W

Figure 13: MinIO User Interface

3. Model Training

For the pet detection, a machine learning model gets developed. For this purpose, individual datasets with annotated images of dogs, cats and hamsters from the Roboflow platform are used and stored on the OneDrive cloud. The data set contains around 1000 hamster, 400 cat and 150 dog pictures. The first step is to merge the individual datasets to obtain a training set. It turns out, that the labels of the different datasets are inconsistent. The dog images as well as the cat images have the label zero. Using a script, the labels of the cat images are transformed from zero to one and the hamster images from zero to two (see figure 14). The script is also available in the GitHub Repository.



```java
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.FilenameFilter;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;

public class TextFileModifier {

    public static void main(String[] args) {
        // Pfad zum Ordner
        String folderPath = "path/to/your/folder";

        // Liste aller Textdateien im Ordner
        File[] textFiles = getTextFiles(folderPath);

        for (File file : textFiles) {
            try {
                modifyFirstCharacter(file);
            } catch (IOException e) {
                System.err.println("Fehler beim Modifizieren der Datei: " + file.getName());
                e.printStackTrace();
            }
        }
    }

    private static File[] getTextFiles(String folderPath) {
        File folder = new File(folderPath);
        return folder.listFiles(new FilenameFilter() {
            @Override
            public boolean accept(File dir, String name) {
                return name.toLowerCase().endsWith(".txt");
            }
        });
    }

    private static void modifyFirstCharacter(File file) throws IOException {
        String content = new String(Files.readAllBytes(Paths.get(file.toURI())));
        String[] lines = content.split("\\r?\\n");
        StringBuilder modifiedContent = new StringBuilder();

        for (String line : lines) {
            if (!line.isEmpty()) {
                modifiedContent.append("1").append(line.substring(1)).append(System.lineSeparator());
            } else {
                modifiedContent.append(line).append(System.lineSeparator());
            }
        }

        try (BufferedWriter writer = new BufferedWriter(new FileWriter(file))) {
            writer.write(modifiedContent.toString());
        }
    }
}
```

Figure 14: Data pre-processing of the labels

In the following, the backend development will be described. With the transformed dataset, three labels are obtained. For training, the computer vision algorithm YOLOv5 is utilized and trained using Google Collab. The outcome is the model, which is deployed on the cluster. Furthermore, the model metrics are provided below (see fig. 15):

The trained model achieves an accuracy of 91% for hamsters, 58% for cats and 72%.

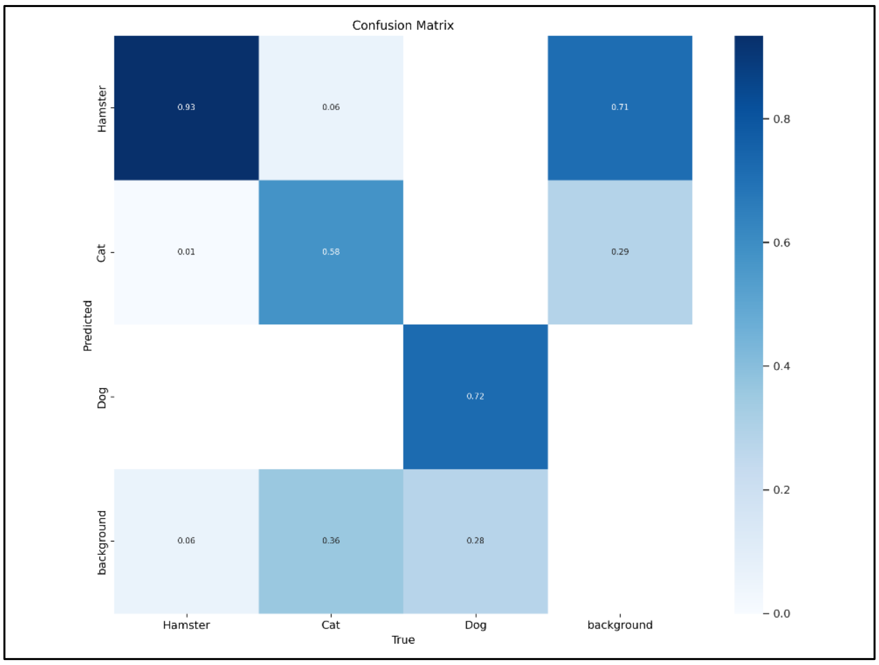


Figure 15: Confusion Matrix

The next step is to load the model on the sensor node (see Fig. 16). A python script is developed to capture the frames of the camera sensor and to load the YOLOv5 model. The camera passes the images to the model and if a pet is detected it uploads the image as a JPG with metadata to MinIO.

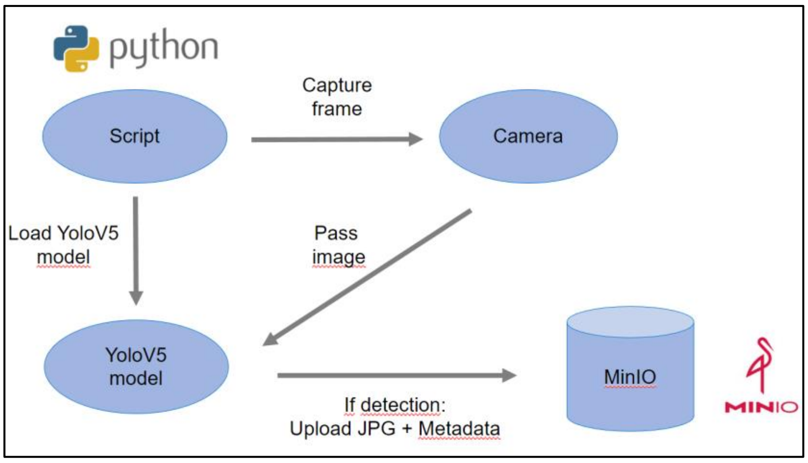


Figure 16: Sensor node architecture

4. Backend Development: WebApp

In the following, the backend development will be described. Python is used as the programming language and Flask serves as the web framework. Additionally, the object storage Min-IO is utilized as the database in our system architecture, as previously mentioned. The configuration data of MinIO can be found in lines 16-19 of *cluster-nodes/web-app/back-end/app.py*. To ensure the functionality, the data related to the endpoint, access key, and secret key must be adjusted according to the respective setup. The backend is responsible for receiving the base64 encoded image from the sensor node and forwarding it to the Telegram bot, as well as uploading the images along with all the metadata (such as detection amount, classes, confidence values, etc.). Moreover, it retrieves the nine most recent detections from MinIO and sends them to the frontend. To facilitate communication between the backend and frontend, two REST APIs are employed. Within the *app.py* file, specifically in lines 30-36, the backend checks if a bucket with the predefined name mentioned in line 19 exists during runtime. If the bucket does not exist, it will be created.

Using the POST method (*http://<backend address>:5001/upload/jpeg*), the backend expects the following JSON data from the sensor node:

- image_data: Base64 encoded image data
- detection_amount: number of detections in the image
- detection_classes: detected classes i.e., cat, dog or hamster
- detection_confidence_values: the confidence values for every detection

In addition to that a timestamp will also be created by the backend as additional metadata for the image. The image and the metadata will be uploaded to MinIO and sent to the Telegram bot, which posts the image and the data in a Telegram channel.

Using the GET method (*http://<backend address>:5001/*), the backend retrieves the nine most recent objects from MinIO, sorted by the 'last_modified' attribute in descending order and sends the following JSON data to the frontend:

- image_data: base64 encoded image data
- detection_amount: number of detections in the image
- detection_classes: detected classes i.e., cat, dog or hamster
- detection_confidence_values: the confidence values for every detection
- detection_timestamp: a timestamp of the detection in the format YYYYMMDDHHMMSS

For testing purposes, two Python scripts are created to test the functionality of two endpoints. These scripts can be found in our GitHub repository at *cluster-nodes/web-app/back-end/test/send-test-detections-to-backend.py*. The first script sends 3 dummy images from the *cluster-nodes/web-app/back-end/test/example-images* directory, along with some mock metadata, to the backend's POST endpoint. To perform this test, the backend URL needs to be adjusted accordingly. The second Python script, located at cluster-nodes/web-app/back-end/test/get-latest-detections-from-backend.py, sends a request to the backend GET endpoint to retrieve the nine most recent detections and displays them in the console. Once again, the backend URL on line 7 should be modified accordingly.

Finally, a docker image of the backend is created and deployed to the kubernetes cluster using a *yaml* file. After that, the backend IP address was exposed so that the backend could be accessed. The last step is to verify the services on the service list of the kubernetes cluster.

In the following section, we provide instructions for a local installation.

Start the MinIO server, use minioadmin as username and password

macOS (using Terminal):

- *# MINIO_ROOT_USER=minioadmin MINIO_ROOT_PASSWORD=minioadmin minio server ~/data --console-address ":9001"*

Windows (using PowerShell):

- *PS> setx MINIO_ROOT_USER minioadmin*
- *PS> setx MINIO_ROOT_PASSWORD minioadmin*
- *PS> C:\minio.exe server F:\Data --console-address ":9001"*

1. Setup and start the Flask backend

Navigate to *cluster-nodes/web-app/back-end/app.py*

- Change the MinIO server configuration in line 16-19 accordingly
- Change the Telegram Bot API Token, URL and Chat ID in line 11-13 accordingly

Create and activate a Python virtual environment:

Windows (using PowerShell):

- *PS> python -m venv .venv*
- *PS> .venv\Scripts\activate*

macOS (using Terminal):

- *# python3 -m venv .venv*
- *# source .venv/bin/activate*

Install the required pip packages

Windows:

- *PS> pip install -r requirements.txt*

macOS:

- *# pip3 install -r requirements.txt*

Start the Flask backend

- Windows:
  - *PS> python app.py*
- macOS:
  - *# python3 app.py*

2. Testing (optionally)

Navigate into cluster-nodes/web-app/back-end/test

- Change the backend URL in line 8 accordingly
- Run *send-test-detections-to-backend.py*

Windows:

- *PS> python send-test-detections-to-backend.py*

macOS:

- *# python3 send-test-detections-to-backend.py*

Receive the latest detections from the backend using the test script *get-latest-detections-from-backend.py*

Navigate into *cluster-nodes/web-app/back-end/test*

- *Change the Backend URL in line 7 accordingly*
- *Run get-latest-detections-from-backend.py*

Windows:

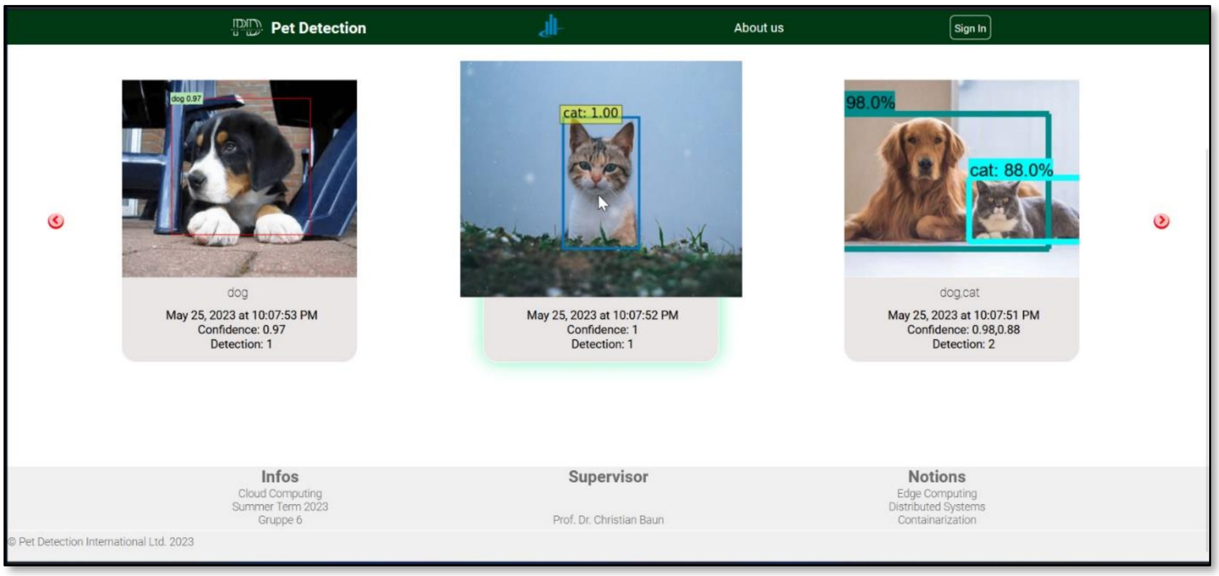
- *PS> python get-latest-detections-from-backend.py*

macOS:

- *# python3 get-latest-detections-from-backend.py*

5. Frontend Development: Web App

The following text presents the frontend development for an edge computing solution (see fig. 17.). The frontend is responsible for presenting log information, event messages, and a map of events with proof like images and timestamps. It is built using React, a modern JavaScript library for building user interfaces. This documentation provides an explanation of the main components, libraries, and technologies used in the frontend development and their benefits for the application.



dog
May 25, 2023 at 10:07:53 PM
Confidence: 0.97
Detection: 1

May 25, 2023 at 10:07:52 PM
Confidence: 1
Detection: 1

dog,cat
May 25, 2023 at 10:07:51 PM
Confidence: 0.98,0.88
Detection: 2

Infos
Cloud Computing
Summer Term 2023
Gruppe 6

Supervisor
Prof. Dr. Christian Baun

Notions
Edge Computing
Distributed Systems
Containarization

© Pet Detection International Ltd. 2023

Figure 17: Pet Detection WebApp

To set up the frontend development environment, the following steps were followed:

First, the project repository can cloned from either the Docker Hub or the version control system, enabling a local copy of the project on the machine. Next, it is important to ensure that Node.js is installed on the system. Node.js serves as a requirement for running JavaScript applications and provides the necessary runtime environment. After confirming the presence of Node.js, the project directory can be navigated to using the command line or terminal. This step holds significance in terms of accessing and effectively managing the project files. Once inside the project directory, the "npm install" command is executed to install all the necessary dependencies. This command automatically fetches and installs the required packages and libraries needed for the project to run smoothly. After installing the dependencies, the "npm run dev" command is used to run the application in development mode. This command triggers the necessary scripts and configurations to start the application and allows work to begin on it. To access the application, one can open a web browser and enter the specified port number. By default, the application is set to run on port 5173, but this value can vary depending on the project's configuration. By following these steps, the frontend development environment is successfully installed and set up, enabling efficient work on the project and access to the application through the specified port for testing and further development. Following are the most important file of the project repository:

- *src/components*: Contains reusable React components used in the application.
- *src/data.js*: Provides sample data for cards to be displayed in the Slider component.
- *src/index.js*: Entry point of the application.
- *src/Slide.scss*: Styling file for the Slide component.
- *package.json*: Defines the project dependencies and scripts.
- *Dockerfile*: Specifies the Docker configuration for containerization.

The frontend project relies on the following dependencies:

- *react*: A JavaScript library for building user interfaces.
- *react-slick*: A React carousel component used for the Slider.
- *slick-carousel*: Provides the required CSS files for the Slider component.
- *react-router-dom*: Enables client-side routing for the application.
- *sass*: Allows the usage of Sass (Syntactically Awesome Style Sheets) in the project.
- *vite*: A build tool and development server for modern web projects.

The Card component (*components/card/Card.js*) represents an individual card displayed within the Slider component. It receives a card item as a prop and renders the necessary information.

The Slide component (*components/slide/Slide.jsx*) is the main component responsible for displaying a carousel of cards. It utilizes the react-slick library to create an interactive and visually appealing slider. The component fetches data from a specified endpoint and renders the received data as cards within the slider. If the data fetching fails, it falls back to using sample data from data.js.

Finally, a docker image of the web application is created and deployed to the kubernetes cluster using a *yaml* file. After that, the web applications IP address was exposed so that the backend could be accessed. The last step is to verify the services on the service list of the kubernetes cluster.

The benefits of using React and Vite include:

- Efficient rendering: React's virtual DOM efficiently updates and renders only the necessary components, resulting in optimal performance.
- Component reusability: React's component-based architecture promotes code reusability and maintainability.

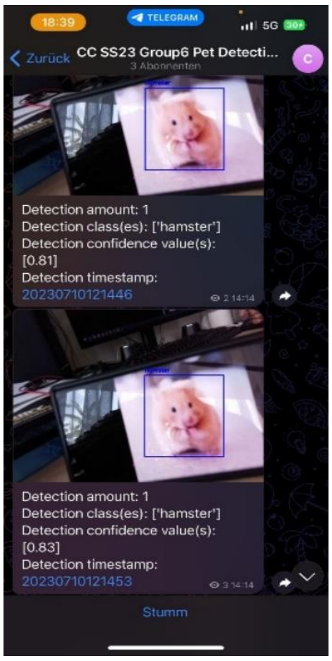
- Fast development and hot module replacement: Vite provides a development server with fast startup times and hot module replacement, allowing for a smooth development experience.
- Rapid refresh: Vite's development server updates the changes in the browser without full page reloads, enabling faster feedback loops during development.
- Raspberry Pi compatibility: React and Vite are lightweight frameworks that can run efficiently on Raspberry Pi, ensuring.

6. Telegram Bot

Additionally, a telegram bot has been integrated into the pet detector, enabling the display of a preview of the captured image, the detection amount, the detection classes, the detection confidence values, and the detection timestamp (see figure 18). The integration of the Telegram bot was accomplished by following the subsequent steps:

1. Create and obtain your Bot token.
2. Find a telegram bot named @Botfather.
1. Print “/help” and you will see all possible commands that the botfather can operate.
2. To create a new bot type “/newbot” or click on it and follow the given instructions. After that a bot token will be created.

After obtaining the bot token, a Telegram channel was created to be utilized by the telegram bot. Additionally, the channel ID needs to be acquired, which is the username of the channel preceded by an "*@*" symbol. The bot token is added to line 11, and the chat ID is added to line 13 of *cluster-nodes/web-app/back-end/app.py*. As previously mentioned, the backend POST Endpoint (*http://<backend address>:5001/upload/jpeg*) sends the image and metadata to the Telegram bot on line 143 of cluster-nodes/web-app/back-end/app.py via another POST request, utilizing the Telegram Bot API Token and URL specified in lines 11 and 12 respectively. The content will be posted in the Telegram channel but modifying the Telegram Chat ID in line 13 of *cluster-nodes/web-app/back-end/app.py* allows the use of any other channel.



Detection amount: 1
Detection class(es): ['hamster']
Detection confidence value(s):
[0.81]
Detection timestamp:
20230710121446
2 14:14

Detection amount: 1
Detection class(es): ['hamster']
Detection confidence value(s):
[0.83]
Detection timestamp:
20230710121453
3 14:14

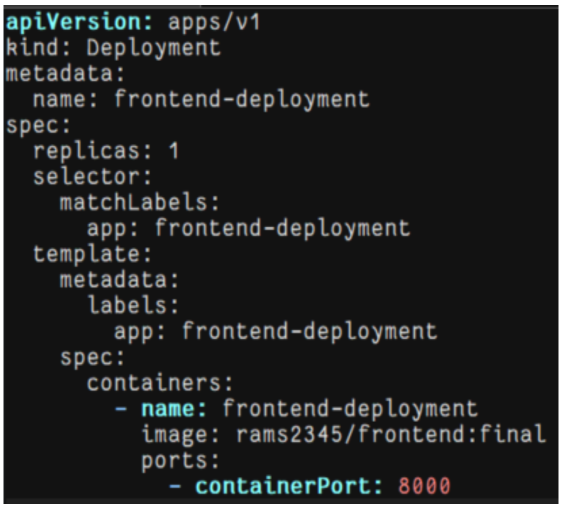
Figure 18: Telegram channel

7. Cluster Deployment

This section depics the deployment of the frontend and backend on the cluster. The development of both the frontend and backend was initially done on local machines. Dockerfiles were then created to build Docker images for both components, which were subsequently pushed to a public repository. YAML files were created to specify the deployment configuration for each component. The Kubernetes command-line tool, kubectl, was used to deploy the images onto the cluster:

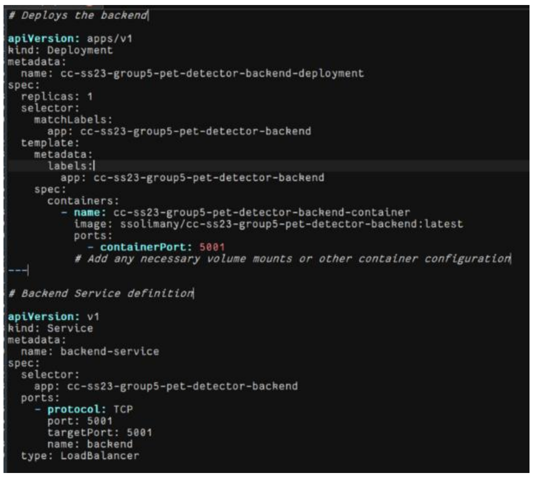
kubectl apply -f frontend-deployment.yaml for the frontend.
kubectl apply -f backend-deployment.yaml for the backend.

Figure 19 and figure 20 depics both YAML files. The successful deployment of the containers is verified using the *kubectl get deployments* command.



```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: frontend-deployment
  template:
    metadata:
      labels:
        app: frontend-deployment
    spec:
      containers:
        - name: frontend-deployment
          image: rams2345/frontend:final
          ports:
            - containerPort: 8000
```

Figure 19: Frontend Yaml file



```yaml
# Deploys the backend

apiVersion: apps/v1
kind: Deployment
metadata:
  name: cc-ss23-group5-pet-detector-backend-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: cc-ss23-group5-pet-detector-backend
  template:
    metadata:
      labels:
        app: cc-ss23-group5-pet-detector-backend
    spec:
      containers:
        - name: cc-ss23-group5-pet-detector-backend-container
          image: ssolimany/cc-ss23-group5-pet-detector-backend:latest
          ports:
            - containerPort: 5001
          # Add any necessary volume mounts or other container configuration
---

# Backend Service definition

apiVersion: v1
kind: Service
metadata:
  name: backend-service
spec:
  selector:
    app: cc-ss23-group5-pet-detector-backend
  ports:
    - protocol: TCP
      port: 5001
      targetPort: 5001
      name: backend
  type: LoadBalancer
```

Figure 20: Backend Yaml file

8. Lessons Learned

During the course of this project, several valuable lessons have been learned. These lessons shed light on important aspects related to hardware setup, resource allocation, choosing the right solution and data quality for model training.

The process of setting up the hardware was something that none of the team members had done before. The project documentation from the previous groups was a great help, especially at the beginning of the project, for a quick learning curve. Furthermore, ChatGPT was a great help in solving code bugs. However, it was quickly realized that ChatGPT still had severe limitations and could be unsuitable for the installation process of a tool, for example.

Another limitation was the singular ownership of hardware. Since only one person could have the hardware at his place, meet ups in person were necessary until the remote access was fully working.

It was also quickly realized that the hardware provided, consisting of Raspberry Pis, had serious performance problems, so for example the lite version on the Raspberry Pi3's was used as the operating system.

The quality of data used for model training is of paramount importance. It is crucial to ensure that the data is accurate, diverse, and representative of the problem domain. In this case, finding the correct data for the labelled pet pictures was a real challenge.

9. Sources

Baun, C. (n.d.). Cloud Computing Course (SS2023). Cloud Computing (SS2023). Retrieved May 13, 2023, from https://www.christianbaun.de/CGC23/index.html

GitHub: Let’s build from here. (n.d.). GitHub. Retrieved July 13, 2023, from https://github.com/

Introducing Chatgpt. (n.d.). ChatGPT. Retrieved June 1, 2023, from https://openai.com/blog/chatgpt

K3s. (n.d.). K3ss. Retrieved May 15, 2023, from https://k3s.io/

Minio Object Storage for kubernetes — Minio Object Storage for kubernetes. (n.d.). MinIO. Retrieved June 4, 2023, from https://min.io/docs/minio/kubernetes/upstream/index.html

Roboflow: Give your software the power to see objects in images and video. (n.d.). https://roboflow.com/

Solawetz, J. (2023). What is YOLOv5? A Guide for Beginners. Roboflow Blog. https://blog.roboflow.com/yolov5-improvements-and-evaluation/