Prototyping Assignment – WildFogs

Problem Description

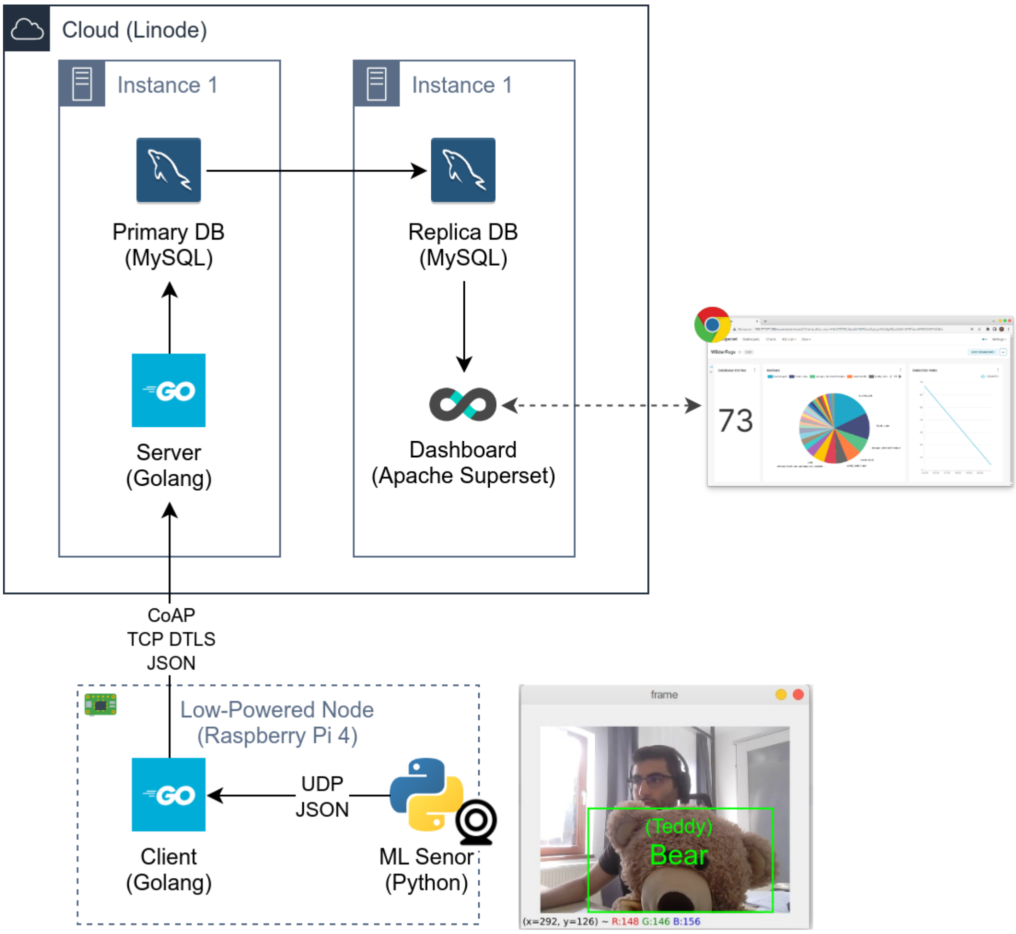
We want to track animals in national parks; That way we can e.g. measure animal populations and inform hikers of the presence of dangerous animals like wolves in an area.

Original map By Lencer - Own work, used:http://www.bfn.de/0308_nlp.htmlGeneric Mapping Tools, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=3665184
Added Legend, cameras, Fog Nodes, and Cloud servers

Our Solution

We designed WildFogs, a distributed network that can be used in the wilderness for machine-learning based animal detection. Our edge nodes are distributed over a forest and are tasked with recognizing objects in their environment and sending any detected tracked animals directly to the cloud. The cloud updates the edge nodes as to what animals it should be tracking.

Components and Interfaces



Edge Node

Each edge node hosts two applications: a sensor application and client application.

Sensor Application

We used a pre-trained object detection model with PyTorch using code from an official PyTorch tutorial[1] intended for a Raspberry Pi 4. The model can detect a variety of animals. Temperature sensor data is randomly generated.
The sensor application sends messages in an interval of one second to the client application. Because we assume that both the sensor and client application run on the same edge device we use UDP messages.

Client

Should the server go down we buffer the sensor data. We then attempt to reconnect to the server at a decreasing frequency.
In case too many messages are buffered the oldest messages are dropped first. The client also receives the list of animals it should track from the server and acts accordingly.

Cloud Instances

Server Application

Conversely the server application is responsible for receiving animal sightings from all edge nodes and adding them to the database. It is also responsible for sending the edge node configuration, which can be updated during runtime in a config file. It behaves like the client does in case of connection loss.

Database

The database contains a single table that holds the animal sighting metadata as shown in the image below. We replicate our database to complement reliable messaging. We use asynchronous primary copy replication distributed at two different data centers.

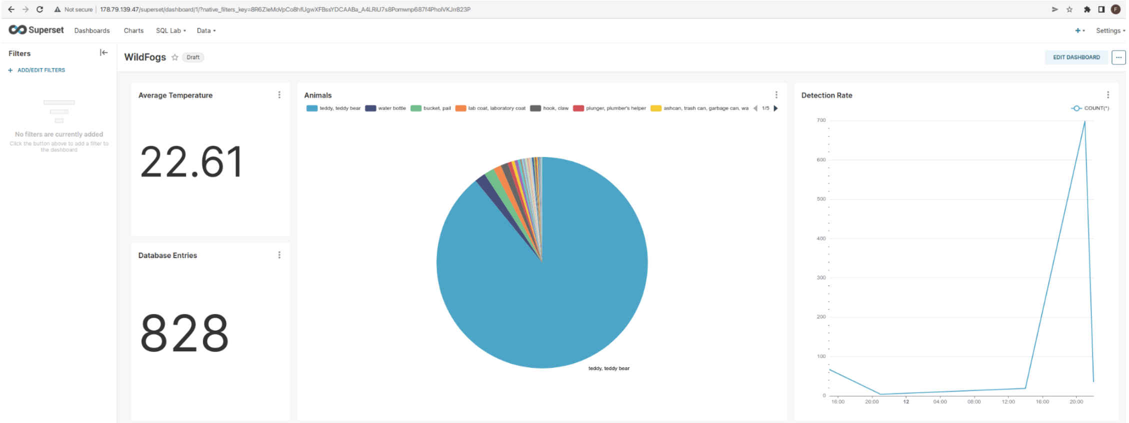
```go
type Animal struct {
    DetectionID    int     `json:"detection_id"`    // Identifier set by cloud database
    DeviceUuid     int     `json:"device_uuid"`     // Device UUID set by camera device (client.go)
    DetectionTime  string  `json:"detection_time"`  // Set by Detection component main.py
    DetectedAnimal string  `json:"detected_object"` // Set by Detection component main.py
    Temperature    float64 `json:"temperature"`     // Set by Detection component main.py
}
```

Label	Status	Plan	IP Address	Region	Last Backup	
ubuntu-eu-central	Running	Linode 2 GB	172.104.142.115	Frankfurt, DE	Never	•••
ubuntu-london	Running	Linode 2 GB	178.79.139.47	London, UK	Never	•••

[1] Real Time Inference on Raspberry Pi 4 (30 fps!) — PyTorch Tutorials 1.12.0+cu102 documentation https://pytorch.org/tutorials/intermediate/realtime_rpi.html

<u>Dashboard</u>

In our dashboard we have four simple charts: the amount of database entries, pie chart of detected animals, number of detected animals over time, average temperature.



Interface between Cloud and Edge

For communication between client and server we chose the Constrained Application Protocol (CoAP) because it has less overhead than HTTP, all communication between the client and the server takes place over TCP it allows for reliable, ordered and corruption-free data streams.

```
cameraNodeTarget := os.Getenv("CLIENT_HOG_CAMERA_IP")                    // Default: "localhost:3333"
fogNodeTarget := os.Getenv("CLIENT_HOG_SERVER_IP")                       // Default: "localhost:3444"
deviceUUIDString := os.Getenv("CLIENT_HOG_DEVICE_UUID")                  // Default: "352" (Doesn't matter)
initialTrackedAnimalsString := os.Getenv("CLIENT_HOG_TRACKED_ANIMALS")   // Default: "[\"Bear\",\"Racoon\",\"Gazelle\"]"
clientHogConfigRxPort := os.Getenv("CLIENT_HOG_LOCAL_CONFIG_RECEIVER_PORT") // Default: ":3555"

fogNodePort := os.Getenv("SERVER_HOG_FOG_NODE_PORT")               // Default: ":3444"
dataSourceName := os.Getenv("SERVER_HOG_DATA_SOURCE_NAME")         // Default: "root:my_fog_password@(172.104.142.115:3306)/my_database"
configFile := os.Getenv("SERVER_HOG_CONFIG_FILE")                  // Default: "config.txt"
clientsConfigurationIPString := os.Getenv("SERVER_HOG_CLIENT_LIST") // Default: Default: "[\"localhost:3555\"]"
```

Environment Variables of the Client and Server Applications

Techstack

- **Go:** client and server applications are written in Go. Go is applicable to concurrent and memory efficient applications, which works well for devices with low resources.
- **MySQL:** We chose MySQL because replication is simple to set up. Bitnamis docker container allows us to set the necessary information for replication as environment variables at startup.
- **Apache Superset:** Apache Superset is a data visualization tool that can use databases as sources to generate good looking charts of our data.
- **Python & PyTorch:** For object detection using a lightweight pretrained model
- **Docker:** As an alternative to traditional installation/deployment

Hardware Specifications

The edge nodes are designed to run on a Raspberry Pi 4 with a camera attached to it. Whereas the server app and dashboard are running on the cloud.