Cat-tracker

TFG de Javier Rubio Giménez

Resumen

Cat-Tracker es un proyecto que pretende facilitar la vida a los propietarios de mascota vía la monitorización de la actividad de su mascota, así como el control de su comedero y bebedero, utilizando tecnologías provenientes del mundo *maker*, siguiendo criterios de bajo presupuesto y libertad de software.

Se comenzara realizando un estudio de la situación actual de mercado, así como una selección de tecnologías *hardware* y software, que permitan alcanzar los objetivos propuestos en el proyecto, con especial énfasis en los criterios definidos en el proyecto.

Por ultimo, haciendo uso de las tecnologías seleccionadas, se desarrollaran diferentes sensores que se comunicaran con un servidor basado en Linux, el cual, utilizando InfluxDB y Grafana, almacenara y permitirá la consulta de los datos obtenidos por los sensores, lo que nos mostrara el día a día de nuestra mascota.

Palabras clave

Internet de las cosas, Sensores, BLE, InfluxDB, Grafana

Abstract

Cat-Tracker is a project which aims to make easier for pets owners to control different parameters of his pets, as is the activity levels and the level of food and water available to them, using technologies from the maker world, taking into account the cost of the materials used in the project and the “openness” of the technologies used.

To meet this end, in the first place we will do a study of the current market situation, and a selection of the technologies, software and hardware, that allow us to meet the objectives defined in the project following the established criteria.

With the technologies selected, the last part of the project will be to develop the sensor which will communicate with the Linux based server, which, using InfluxDB and Grafana will show the information gathered by the sensors, allowing to check the day to day life of our pet.

Keywords

Internet-of-things, Monitoring, BLE, InfluxDB, Grafana

Sumario

1. Introducción

En lo hogares actuales la convivencia con mascotas es habitual, conviviendo con perros grandes hasta animales pequeños como ratones o pájaros. Pero,¿Que hacen las mascotas cuando no las vemos? ¿Cuanto comen,juegan o duermen?

El objetivo de este proyecto es responder a estas preguntas, utilizando tecnologías propias del I.O.T. y el mundo *maker,* apoyándose en tecnologías abiertas y asequibles, de tal manera que el resultado pueda ser reproducible por otros entusiastas de la tecnología.

Para ello, buscaremos acoplar sensores al animal que nos permitan recoger información acerca de su comportamiento, tratando de realizarlo de la manera menos intrusiva posible para el animal.

Dicho datos se enviaran de forma inalámbrica a un servidor que los almacenara y permitirá consultarlos a través de un ordenador o un teléfono inteligente, permitiendo conocer los patrones de actividad del animal.

Adicionalmente, también monitorizaremos los patrones de alimentación y bebida del animal, registrando la hora y duración de las ingestas, así como la cantidad de alimento y agua restantes, permitiendo el sistema generar alertas cuando se requiera una recarga.

1.1 Descripción

Este proyecto consiste en desarrollar un sistema de seguimiento de actividad y monitorización para mascotas (centrado en mascotas caseras) que permita monitorizar varios parámetros acerca de la actividad, el estado y las necesidades del animal, incluyendo niveles de actividad diarios y nivel de la comida y bebida.

El proyecto constara de diferentes dispositivos que se interconectarán entre si:

- Un dispositivo de seguimiento de actividad del animal, que equipe acelerómetro y giroscopio y permita registrar la actividad del animal(cuando esta en reposo, cuando esta jugando...).

- Un servidor basado basado en Linux. Para este dispositivo se primara un bajo consumo energético así como la posibilidad de que funcione usando disipación pasiva.
- Adicionalmente, se valorará también el uso de algún tipo de sensores vía wifi o Bluetooth para controlar la cantidad de comida y agua disponible para el animal así como su consumo diario.

Para la elección de los diferentes dispositivos, se primaran los criterios de coste económico y licencias de software, de tal manera que el proyecto resultante sea lo mas reproducible posible a nivel domestico.

A nivel de software, la idea es utilizar el servidor Linux para almacenar los datos producidos por lo diferentes elementos *hardware*, para permitir tanto su consulta como generación de alarmas en base a los datos obtenidos, usando tecnologías web que permitan la conexión desde cualquier dispositivo de usuario.

Para la elección de este software, se primara el uso de tecnologías libres, tratando de reaprovechar tantos componentes existentes como sea posible, basándose idealmente en la integración y configuración de elementos existentes, contando con una base datos y una aplicación web que permita visualizarlos. El envío de alertas se haría en una primera fase vía e-mail.

El proyecto estará desarrollado específicamente para gatos, por disponibilidad de sujeto de pruebas(gato común europeo adulto), resultando extrapolable a otros animales de tamaño similar o superior.

1.2 El sujeto de pruebas

Para evaluar el funcionamiento del proyecto, nos apoyaremos en la ayuda de “Campanilla”:

Figura 1: Campanilla, sujeto de pruebas

Campanilla es una gata de tipo común europeo de tamaño medio(aprox 4 kg) que hará de sujeto de pruebas. Para ello monitorizaremos el tipo de datos que los sensores producción durante sus actividades normales, como dormir, explorar, jugar y comer.

Partiendo de dicha información, trataremos de catalogar aproximadamente que tipo de datos producen cada actividad, de tal manera que podamos clasificar aproximadamente los intervalos de tiempo que dedica a cada actividad.

2. Estudio de mercado

La idea de monitorizar el comportamiento de las mascotas no es nueva, siendo un campo que ha comenzado a explotarse en los últimos años, donde han empezado a surgir alternativas comerciales que podrían compararse al ámbito que tenemos planteado con este proyecto.

A continuación, vamos a explorar algunas y compararlas con el planteamiento del proyecto que queremos llevar a cabo.

2.1 PetPace

Figura 2: Collar y receptor PetPace

PetPace es una solución integrada compuesta por un collar, un receptor para la comunicación inalámbrica y un servidor corriendo en la nube, gestionado por el proveedor del dispositivo.

Su funcionamiento es similar a la propuesta de este proyecto. Consiste en recibir la información de los sensores adosados al animal, que son transmitidos al receptor. Dicho receptor cuenta con acceso a internet, donde retransmite los datos, que son almacenados y procesados en la nube.

Además, cuenta con aplicación de *smartphone* a juego, que se conecta a la nube para mostrar los diferentes parámetros del animal monitorizados, así como enviar alertas acerca que se puedan producir acerca de el estado del animal.

Esta propuesta es una alternativa comercial a la parte de monitorización que proponemos, contemplando mas parámetros que este proyecto de hecho, pero haciéndolo a un elevado coste, ya que además de ser costosa la adquisición, requiere una suscripción para acceder a los servicios en la nube, lo que le quita gran atractivo.

2.2 Tractive

Figura 3: Tractive

Tractive es un collar que equipa una batería, un modulo de GPS y un modulo de conexión móvil, permitiendo localizar al portador del collar en todo momento vía aplicación móvil.

Además, cuenta con la posibilidad de establecer alertas, realizar “geo-fencing”,esto es, delimitar zonas geográficas de las que el usuario sea alertado cuando el portador del collar entre o salga.

También cuenta con funcionalidad extra, como puede ser la generación de mapas de calor, para ver que zonas son las que mas frecuenta nuestra mascota, detectar patrones inusuales...

Este producto funciona mediante un pago inicial que conlleva la adquisición de los equipos, combinado con una suscripción mensual, por lo que no es económico de utilizar.

En comparación con el ámbito de nuestro proyecto, la funcionalidad de Tractive seria otro acercamiento a un objetivo similar, controlando la actividad del animal vía posición,

permitiendo conocer donde esta la mascota en todo momento, sin estar limitado geográficamente, pero a cambio de perder sensibilidad en la medición.

2.3 Petcare HD

Figura 4: Petcare HD

Petcare HD es un comedero automático equipado con conectividad móvil, lo que permite controlarlo a distancia vía aplicación móvil, programando entregas periódicas de cantidades predefinidas de comida.

Además, cuenta con una cámara con detección de movimiento por infrarrojos, lo que permite monitorizar los hábitos de alimentación de la mascota usuaria del comedero, a un coste de adquisición elevado.

En comparación con nuestro proyecto, Petcare ofrece una funcionalidad similar en cuanto a control de alimentación, si bien utiliza un principio mas avanzado, ya que en lugar de monitorizar las cantidades ingeridas, lo que hace es limitar la disponibilidad de alimento a la mascota usuaria.

3. Selección de tecnologías

En este capitulo describiremos las tecnologías seleccionadas para la realización de este proyecto, comenzando por las módulos *hardware* que utilizaremos, para a continuación hablar del software que usaremos.

3.1 Hardware

En primer lugar, definiremos que plataformas *hardware* vamos a utilizar para realizar el proyecto.

3.1.1 Monitorización animal

Para la monitorización de la actividad del animal, necesitamos una solución que cuente con sensores, capacidad de comunicación inalámbrica, y sea capaz de funcionar con batería. Adicionalmente, la solución deberá contar con una API que podamos utilizar, a ser posible basadas en tecnologías abiertas. En resumen, necesitamos lo siguiente:

1. Sensores: Acelerómetro y giroscopio
2. Comunicaciones: Wifi/Bluetooth, idealmente BLE
3. Alimentación: Batería
4. Tamaño: Pequeño y ligero.
5. Programación: API accesible
6. Bonus: Basado en tecnologías libres.

Teniendo esto en cuenta, las alternativas que hemos considerado para la realización del proyecto son las siguientes:

3.1.2 Alternativa 1 – Mi Band

Figura 5: Xiaomi Mi Band

La Xiaomi Mi Band es una pulsera de fitness tracking que funciona en sincronización con un teléfono inteligente y permite recoger distintos parámetros utilizando los sensores que lleva integrados(pasos, frecuencia cardíaca), que enviá al teléfono, a la vez que permite recibir notificaciones de aplicaciones de este.

Pros:

- *Hardware* comercial, gran disponibilidad, bajo coste
- Integra batería de larga duración
- Es posible utilizarla como collar sin modificaciones
- Se pueden reciclar unidades viejas para este propósito

Contras:

- No dispone de API oficial
- Los proyectos existentes que permiten comunicar con ella se basan en simular la aplicación móvil. No es posible obtener las lecturas directas de sus sensores.

Conclusión:

Si bien a nivel de *hardware* era mi opción preferida para realizar este proyecto, la falta de disponibilidad de software(no podemos acceder a los datos de los sensores, ni programar una aplicación personalizada que gestione las comunicaciones), hace que no sea posible utilizarlo en este proyecto, por lo que la opción quedo descartada.

3.1.3 Alternativa 2 – Arduino Lilypad

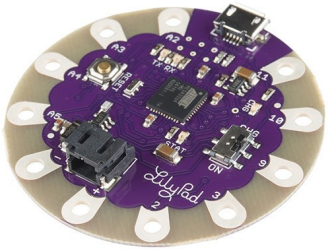

Figura 6: Arduino Lilypad

Arduino Lilypad es una linea de Arduino pensada para *e-wearables*(elementos para ser cosidos en prendas textiles), lo que permite fabricar ropa inteligente o, en nuestro caso, fabricar un arnés inteligente.

Esta plataforma esta basada en Arduino, por lo que cuenta con acceso a sus herramientas de software(Arduino IDE y las todas las librerías existentes), ademas de contar con muchos accesorios para extender su funcionalidad(soporte para pilas de botón, placas con sensores...).

Pros:

- Plataforma libre
- Multitud de recursos software existentes
- Muchos accesorios, altamente personalizable

Contras:

- No cuenta con un modulo de comunicación que sea integrable en un textil
- El montaje utilizando hilo conductor puede resultar poco resistente al estar adosado a un animal.

Conclusión:

Si bien utilizar una plataforma libre era una idea muy interesante, la falta de un modulo de comunicación integrable hace inviable su uso.

3.1.4 Alternativa 3 – ESP32 M5Stick C

Figura 7: M5Stick C(Basado en ESP32)

El M5Stick C es un kit de desarrollo de reloj basado en un controlador ESP32, integrando una pequeña pantalla, sensores y una pequeña batería, integrado en una carcasa con forma de reloj.

Se puede programar con *MicroPython* o las herramientas de Arduino, por lo que cuenta con gran cantidad de librerías libres compatibles con el controlador.

Pros:

- Integración de todos los elementos necesarios en un mismo sistema
- Alta disponibilidad de software
- Bajo coste

Contras:

- Kit de desarrollo, no un modelo final
- Batería de poca capacidad

Conclusión:

Esta alternativa cumple con los requisitos necesarios para la realización de este proyecto, si bien tiene algunos puntos mejorables, pero sera la opción que utilicemos para este proyecto.

3.2.1 Monitorización comedero y bebedero.

Además de la monitorización directa del animal, también se plantea monitorizar la cantidad de materia disponible tanto en el comedero como en el bebedero, a través de la medición de parámetros de los recipientes correspondientes.

Para ello, necesitamos un sistema que sea capaz de soportar una celda de carga, recoger sus datos y enviarlos al servidor, por lo que necesitamos que cuente con conectividad inalámbrica.

Por otro lado, al ser instalación fija, no necesitamos que cuente con batería, ya que podemos alimentarlo directamente a partir de la red eléctrica, resultando ademas en una relajación de los requisitos de tamaño en comparación con el otro conjunto de sensores.

Además, es deseable que el sistema resultante sea silencioso, con consumo energético bajo, y de bajo coste, para poder ser integrado en el hogar de la mejor forma posible.

A nivel de programación, el dispositivo a utilizar deberá contar con un entorno de desarrollo que podamos utilizar, así como las necesarias APIs para poder acceder a las funciones necesarias.

En resumen, los requisitos son los siguientes:

1. Soportar I/O para controlar una célula de carga
2. Conectividad inalámbrica
3. Factor forma reducido
4. Consumo energético bajo
5. Coste reducido.

Teniendo en cuenta estos requisitos, las alternativas consideradas han sido las siguientes:

3.2.2 Alternativa 1 – Arduino Uno con placa Wifi/Bluetooth

Figura 8: Arduino Uno

Arduino Uno es una pequeña placa de desarrollo con multitud de conectores de entrada/salida combinado con un pequeño microcontrolador *ATMega* que puede ser programado mediante el usuario utilizando las herramientas proporcionadas por el fabricante.

Arduino es una plataforma extremadamente popular en el mundo aficionado, por lo que existen multitud de recursos a disposición del usuario que se puede utilizar,lo que facilita el trabajar con ella.

Además, en este caso habría que combinarlo con una placa *hat* que añada conectividad inalámbrica al conjunto, existiendo multitud en el mercado que se puede utilizar.

Pros:

- Bajo coste
- Dimensiones reducidas
- Gran comunidad detrás

Contras:

- Requiere de *hardware* adicional para ofrecer conectividad inalámbrica
- Capacidad de procesamiento extremadamente reducida, eso fácil agotar los recursos de los que dispone.

Conclusión:

Arduino es una opción valida para realizar nuestro proyecto, no obstante no es la plataforma que vamos a utilizar.

3.2.3 Alternativa 2 – T.I. MSP432 Launchpad con placa Wifi/Bluetooth

Figura 9: MSP 432

La MSP432 es una familia de placas de desarrollo proporcionada por *Texas Instruments* que combina diferentes puertos de entrada/salida con un microcontrolador programable para poder realizar aplicaciones personalizadas, siendo en este sentido similar a Arduino.

No incluye soporte a comunicaciones inalámbricas de serie, por lo que debe ser combinado con una placa adicional que proporcione la conectividad.

Cuenta con herramientas de programación similares a las de Arduino, y una comunidad detrás que proporciona recursos que se pueden utilizar para facilitar el desarrollo de las aplicaciones.

Pros:

- Coste reducido
- Factor forma reducido
- Consumo energético reducido
- Microcontrolador mas potente que Arduino

Contras:

- Comunidad mas reducida que Arduino
- La conectividad inalámbrica requiere de una placa adicional.

Conclusión:

Al igual que Arduino, esta placa cumple con todos los requisitos básicos. El motivo de escogerla es que ya dispongo de unidades que puedo utilizar.

3.2 Servidor

Una vez tenemos los sensores, necesitaremos almacenar la información que generan en algún sitio, que ademas sea capaz de interpretarlos y mostrarlos al usuario de una forma sencilla e intuitiva, osea, necesitamos algún dispositivo que haga de servidor.

Además de la capacidad de procesamiento, como requisitos extra, nos interesa que sea pequeño, silencioso, y con un consumo energético bajo, siempre enmarcado dentro de un presupuesto razonable.

Por otro lado, no necesitamos una alta potencia de procesamiento, ya que en principio el sistema no va a manejar grandes cantidades de datos, por lo que la potencia no es un factor importante.

En resumen:

1. Comunicaciones: Wifi y bluetooth/BLE
2. Consumo energético: Bajo
3. Factor forma: Pequeño
4. Precio: Bajo
5. Ruido: Silencioso

Teniendo esto requisitos definidos, las alternativas que se han considerado son las siguientes:

3.2.1 Alternativa 1 – Ordenador viejo

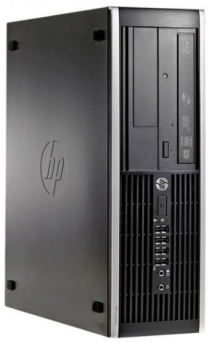

Figura 10: Ordenador genérico

La primera alternativa en consideración fue reutilizar cualquier ordenador antiguo sin uso al que tengamos acceso. Cualquier equipo relativamente antiguo(~10 años) cuenta con potencia suficiente para realizar las labores esperadas, con un conste muy contenido, variando entre gratuito a unos pocos euros si requiere algún adaptador como pueda ser una tarjeta Wifi.

Pros:

- Reutilizamos material existente en lugar de comprar material nuevo
- Mucha capacidad de calculo
- Variedad de plataformas software disponibles

Contras:

- Baja eficiencia energética
- Ruidoso
- Generalmente voluminoso en tamaño

Conclusión:

Si bien este tipo de ordenador es perfectamente valido para llevar adelante el proyecto, se ha decidido optar por otras alternativas que sean mas pequeñas, eficientes y silenciosas.

3.2.2 Alternativa 2 – Intel NUC

Figura 11: Intel NUC

Intel NUC es una familia de Intel de “mini” ordenadores, en un factor forma reducido, con una gran variedad de procesadores disponibles según el nivel de potencia deseado, utilizando disipación activa, y con un nivel de eficiencia energética razonable, contando con las mejoras que ha habido en los últimos años al respecto(descenso en el tamaño de los transistores).

Todas las familias NUC están basadas en la arquitectura de Intel x86-64, por lo que la variedad de software disponible es inmensa, pudiendo correr cualquier S.O. tradicional.

Pros:

- Factor forma reducido, ocupa poco espacio
- Gran variedad de software
- Alta capacidad de procesamiento
- Consumo energético

Contras

- Ruido, utilizan refrigeración activa
- Coste elevado

Conclusión:

Si bien es una alternativa perfectamente valida para realizar el proyecto descrito en esta memoria, se ha preferido optar por una alternativa mas económica y silenciosa.

3.2.3 Alternativa 3 – Raspeberry Pi

Figura 12: Raspberry Pi

La *raspberry Pi* es una pequeña placa que integra los componentes básicos de un ordenador en un factor muy reducido, basada en la arquitectura ARM, con un consumo energético extremadamente reducido en comparación a un ordenador tradicional, y con una importante comunidad detrás.

La *raspberry Pi* hace uso de sistemas operativos basados en Linux, con lo que dispone de gran variedad de software, si bien esta es reducida en comparación con las plataformas tradicionales basadas en x86-64.

Pros:

- Factor forma, tamaño extremadamente reducido
- Consumo energético reducido
- Su coste es reducido
- Puede funcionar con ventilación pasiva, haciéndola muy silenciosa

Contras:

- La capacidad de procesamiento es baja respecto a un ordenador tradicional
- La plataforma ARM dispone de una variedad reducida de software en comparación a una plataforma tradicional.

Conclusión

La *raspberry Pi* es una candidata perfecta para el proyecto planteado en su alcance actual, por lo que es la alternativa que utilizaremos.

4. Software

Con el *hardware* a utilizar en el proyecto ya definido, el siguiente paso es decidir que software vamos a utilizar para desarrollar las aplicaciones necesarias a ejecutar para alcanzar los requisitos descritos en el proyecto.

En este apartado, sera necesario distinguir entre las necesidades de los controladores de los sensores y la del servidor, ya que funcionaran en entornos bastante diferentes.

4.1 Sensores

Primero vamos a analizar las necesidades de los sensores, siguiendo un orden similar al que hemos seguido en el análisis del *hardware* que vamos a utilizar.

El primer apartado a tener en cuenta es que en estos apartados es que la escasez de recursos *hardware* hace inviable correr un sistema operativo tradicional(Windows/Linux), por lo que la variedad de software que podemos utilizar queda limitada por el software que soporten las placas, que normalmente son lenguajes de bajo nivel tipo C.

Considerando esto, lo que necesitamos del software es básicamente que nos permita acceder a la información de los sensores y enviarla al servidor vía comunicaciones inalámbricas.

En resumen, los requisitos son:

1. Acceder a la información proporcionada por los sensores
2. Procesarla para pasarla a un formato que el servidor pueda entender
3. Enviar la información al servidor utilizando comunicaciones inalámbricas.

Teniendo esto en cuenta, las alternativas a considerar son las siguientes:

4.1.1 Alternativa 1 – Sistema monotarea

La primera alternativa que consideramos para realizar esta función consiste en utilizar C para programar un bucle infinito que ejecute una tarea de forma repetida con infinitas repeticiones.

La tarea consistirá en la lectura de los valores de los sensores, su procesamiento y su posterior envío al sistema servidor, todo de forma secuencial, en un bucle que se

repetirá de forma infinita, contando con una espera al final de cada iteración que permita controlar el ritmo de envió de información al servidor.

Pros:

- Programación sencilla, no se requiere sincronización
- Código fácil de interpretar

Contras:

- Sistema monotarea
- Dificultad para responder a eventos
- Dificultad para mantener la sincronización con el servidor

Conclusión:

Si bien esta es la alternativa mas sencilla de utilizar, tiene grandes limitaciones en cuanto a como puede funcionar, ya que no permite ningún tipo de concurrencia, motivo por el que la descartamos.

4.1.2 Alternativa 2 – FreeRTOS

FreeRTOS es un sistema operativo de tiempo real diseñado para microcontroladores de código abierto que nos permite ejecutar diferentes tareas de forma concurrente, simulando ser un sistema multihilo.

Dentro de este sistema, podemos programar tareas que se encarguen de recibir la información de los sensores mientras utilizamos otras que nos permitan gestionar la comunicación con el servidor.

Pros:

- Sistema multitarea
- Permite respuesta en tiempo real
- Capaz de responder a eventos

Contras:

- Complejidad de programación
- Requiere sincronización entre las diferentes tareas

Conclusión:

FreeRTOS nos permite la flexibilidad necesaria para poder manejar los sensores y la comunicación con el servidor de forma asíncrona, por lo que es la opción que escogeremos para programar los sensores.

4.2 Servidor

Una vez decidido el software que vamos a utilizar en los sensores, nos falta decidir que software queremos utilizar para gestionar el servidor que tenemos que desarrollar.

Dicho software deberá manejar por un lado la comunicación con las placas de los sensores y almacenar la información que reciba de esta, y por otro lado encargarse de presentar la información recibida al usuario, de forma lo mas intuitiva posible, así como enviarle alertas.

Teniendo en cuenta los requisitos anteriores, se hace inmediato ver que necesitamos al menos dos módulos de software independientes, uno que gestione la comunicación con los sensores y almacene la información recibida, y otro que sea capaz de coger esa información y mostrarla al usuario de forma clara e intuitiva.

Por ultimo, tendremos también que considerar como vamos a almacenar la información, requiriendo por tanto tres módulos software independientes con sus propios requisitos:

1. Modulo sensores
    1. Gestión de la conectividad inalámbrica
    2. Gestión de la comunicación con los sensores
    3. Procesamiento de la información recibida
    4. Almacenamiento de la información en la base de datos.

2. Modulo interfaz
    1. Recopilación de información de la base de datos
    2. Interfaz de usuario que permita visualizar los datos
    3. Análisis de los datos para enviar alertas al usuario cuando sea necesario.
3. Modulo base de datos
    1. Permitir el almacenamiento de la información de los sensores junto con la información de tiempo
    2. Permitir la recopilación de los datos en series históricas para ser analizados y presentados al usuario.

Además, como características deseables, consideramos también que los sistemas a utilizar sean de código abierto, manteniendo el espíritu de este proyecto.

4.2.1 Modulo sensores - Python

Python es un lenguaje de alto nivel interpretado que ha ido ganando gran relevancia en los últimos años, hasta colocarse como uno de los lenguajes mas utilizados por los desarrolladores a nivel mundial.

Python cuenta con una gran comunidad detrás, que ha desarrollado cientos de librerías y recursos que se pueden utilizar de forma gratuita, añadiendo gran cantidad de funcionalidad extra al lenguaje básico.

En nuestro caso concreto, utilizaremos Python para desarrollar la comunicación con los sensores aprovechando la librería Bluepy, que nos permite la conexión vía Bluetooth, y utilizando un adaptador para conectarnos la base de datos.

Pros:

- Lenguaje de alto nivel, amigable
- Lenguaje interpretado, es mas fácil de depurar y modificar
- Multitud de librerías
- Open source

Contras:

- Bajo rendimiento

Conclusión

Python es un lenguaje perfecto para realizar un prototipo rápido, con multitud de recursos disponibles, y en crecimiento.

4.2.2 Modulo base de datos – InfluxDB

InfluxDB es una base de datos orientada a series de datos temporales, con énfasis en el alto rendimiento, habitual en el ámbito del internet de las cosas(*I.O.T.*) y en el mundo *maker*.

Cuenta con un lenguaje similar al SQL, para facilitar su uso a los desarrolladores provenientes del mundo de las bases de datos relacionales clásicas, lo que facilita mucho su adaptación.

Pros:

- Orientada a series temporales
- Alto rendimiento
- Lenguaje parecido a SQL
- Software libre

Contras:

- Requiere aprendizaje

Conclusión

InfluxDB es una base de datos de código abierto que se acomoda bien a los requerimientos de nuestro proyecto, por lo que es la alternativa que usaremos.

4.2.3 Modulo interfaz – Grafana

Grafana es una plataforma apoyada en tecnologías web que sirve para mostrar y monitorizar información almacenada en una base de datos de algún tipo soportado, todo construido sobre tecnología open-source.

Por defecto cuenta con la funcionalidad básica de mostrar la información, permite la generación de alertas en base a limites que se definan en base al analisis de los datos contenidos en la base datos.

Además, cuenta con un sistema de plugins que permite extender su funcionalidad programando módulos, o usar alguno de los módulos ya desarrollados del mercado con el que cuenta.

Pros:

- Plataforma existente
- Código abierto
- Funcionalidad extensible mediante plugins
- Generación de alertas al usuario

Contras:

- Sistema especifico, requiere conocimiento acerca de la plataformas

Conclusión:

Grafana es una solución ya probada, que cuenta con una comunidad amplia y con multitud de recursos disponibles, adecuándose a nuestras necesidades, por lo que es la solución que vamos a utilizar.

4.2.4 Gestión de Software – Docker

Para facilitar el despliegue del software a utilizar en el servidor, nos vamos a apoyar en Docker y sus contenedores, lo que permitirá que la configuración desarrollada sea fácilmente portable entre distintas plataformas, permitiendo replicar la configuración en otros entornos de forma sencilla.

Para ello, Docker se basa en el uso de contenedores, que permiten correr procesos de forma aislada al resto del sistema, permitiendo utilizar imágenes autocontenidas que se ejecutaran de forma aislada al resto del sistema, conteniendo todo el software necesario para que el proceso se pueda ejecutar.

Este concepto, similar en objetivo al de maquina virtual, se consigue haciendo uso de diferentes tecnologías presentes en el kernel de Linux, como los *namespaces* y los *cgroups,* permitiendo ejecutar aplicaciones de forma aislada entre si.

Además, utilizaremos imágenes preconfiguradas ofrecidas directamente por la empresa responsable de Docker en el Dockerhub, lo que nos permitirá que cualquier usuario que quiera replicar el funcionamiento desarrollado en este proyecto, podrá coger directamente los diferentes contenedores y ejecutar sobre ellos el software que desarrollado para este proyecto.

Especificamente, para este proyecto, utilizaremos tres imagenes disponibles publicamente en el hub de docker:

- Python: Imagen disponible con la ultima versión estable de Python, mantenida por la comunidad de docker.
- InfluxDB: Imagen disponible con la ultima versión estable de InfluxDB, mantenida por la empresa responsable.
- Grafana: Imagen disponible mantenida por la empresa responsable de Grafana, actualizada a la ultima versión estable.

Además, junto con el proyecto se incluirá un *script* de Bash que permitirá descargar y configurar las imágenes con las configuraciones necesarias para crear contenedores listos para ejecutar el proyecto.

5. Sensores a utilizar

El objetivo de este proyecto es medir diferentes parámetros que afectan a la vida de nuestras mascotas, de forma que podamos cuantificarlos para obtener información acerca tanto de los hábitos de vida como de las necesidades de nuestras mascotas, para lo que necesitaremos sensores que permitan tomar valores del mundo físico.

En concreto, en este proyecto utilizaremos tres tipos de sensores diferentes:

- Acelerómetro y giroscopio para cuantificar la actividad de la mascota a lo largo del día, midiendo la posición del giroscopio y las aceleraciones de los diferentes ejes, lo que permitirá deducir su estado a partir de los cambios en las medidas con respecto a los valores anteriores.
- Sensor de ultrasonidos para cuantificar el ritmo de consumo de comida y poder notificar al usuario cuando se requiera relleno del comedero, midiendo la distancia desde un punto fijo donde se colocara el sensor al centro del comedero.
- Celda de carga para controlar el consumo de agua del animal, y notificar al usuario en caso de que el nivel de agua restante sea bajo, midiendo la fuerza que ejerce el comedero contra la celda de carga, los que nos permitirá controlar el peso.

A continuación, haremos una descripción mas precisa de cada uno de ellos.

5.1 Sensores del animal

Para medir la actividad de la mascota a lo largo del día, vamos a utilizar dos tipos de sensores, un acelerómetro de tres ejes y un giroscopio de tres ejes, que nos permitirán medir el movimiento del animal, así como sus cambios de posición a lo largo del tiempo.

Para ello, vamos a hacer uso de la MPU6886 integrada en el M5StickC, ya que es un acelerómetro/giroscopio que cumple con los requisitos de este proyecto, estando integrado en el dispositivo y siendo capaz de alimentarse de la batería que este integra.

Nuestro objetivo con este sensor es tomar medidas periódicas de todos los ejes disponibles, cuantificando el estado de actividad del animal según los niveles de aceleración, y los cambios de niveles del giroscopio, a intervalos de tiempo predefinidos.

El hecho de utilizar intervalos predefinidos nos resta precisión en la cuantificación, pero nos permite maximizar la vida de la batería, y nos sirve para tener una idea general de los indices de actividad, ya que este proyecto no requiere de grandes niveles de precisión para su funcionamiento.

Respecto a las medidas, recibiremos medidas de dos tipos diferentes: Posición de los ejes del giroscopio, y aceleración en los diferentes ejes, lo que nos permitirá estimar la posición del animal, así como si nivel de actividad en base a las diferencias de aceleración en el acelerómetro.

En el caso concreto de nuestro sujeto de pruebas, Campanilla, lo que esperamos ver con estos sensores son grandes franjas de inactividad(un gato puede dormir hasta 16 horas al día), combinado con unas horas de actividad ligera(gato explorando) y pequeñas ráfagas de alta actividad(gato jugando).

5.2 Sensor comedero

Para medir el nivel de comida, en nuestro caso vamos a optar por un sensor de ultrasonidos, cuya utilidad es medir la distancia libre que tienen en frente, utilizando una emisión de ultrasonidos y midiendo el retorno de la señal enviada.

Utilizando este principio, situaremos el sensor sobre el comedero, y mediremos la distancia que hay entre el sensor y el comedero, que sera menor cuanta mas comida contenga el recipiente, siendo además capaz de detectar cuando el animal esta usando el comedero por las disminuciones súbitas de distancia.

Para ello, necesitaremos realizar un proceso de calibración, tomando como referencias las distancias al comedero vacío, al comedero con poca comida, y al comedero lleno, y contando que distancias considerablemente menores a la posición de comedero lleno como presencia del animal.

Para esta tarea, vamos a utilizar un sensor HC-SR04, sensor de ultrasonidos de bajo coste con una precisión aproximada de 3mm, pudiendo medir distancia desde un rango

que va de los 2 centímetros a los 4 metros, precisión y rango mas que suficiente para los propósitos de este proyecto.

Dicho sensor ira conectado a una placa que se encargue de alimentarlo y recoger sus datos, que enviara periódicamente al servidor, para ser almacenado y procesado por las herramientas software del servidor.

En este caso, lo que esperamos ver son visitas cortas, pero frecuentes(varias veces al día), del animal al comedero, apreciando ligeras disminuciones de la cantidad de comida en cada visita, necesitando rellenarlo cada día aproximadamente para un comedero de tamaño pequeño.

5.3 Sensor bebedero

Para medir el nivel del bebedero, podríamos usar el mismo principio que para el comedero, no obstante, la mayoría de fuentes de agua que se venden para mascotas se basan en el uso de un compartimento cerrado lleno de agua, que una bomba saca por un chorro hacia arriba a través de una obertura.

Al no estar expuesto al aire el deposito del agua, el uso de un sensor de ultrasonidos se complica, ya que tendría que ser instalado dentro de la propia fuente, donde las condiciones son muy adversas, donde es muy fácil que la electrónica se moje.

Como alternativa, utilizaremos una celda de carga, que es un sensor que mide la fuerza que se ejerce contra el, lo que nos permite controlar el peso, y a partir del peso del bebedero, controlar la cantidad de agua disponible.

Adicionalmente, como el animal se apoya en el bebedero, también podremos controlar las visitas del animal al bebedero, ya que aumentos repentinos del peso que excedan el peso de la fuente llena se podrán considerar como indicativo de la presencial del animal.

Como celda de carga, utilizaremos una celda de carga de hasta 5kg conectado a un amplificador HX711, que es una unidad de bajo coste que se utiliza habitualmente para proyectos caseros, normalmente conectada a un Arduino o una Raspberry Pi, lo que la hace adecuada para el proyecto que queremos realizar.

Para su funcionamiento, deberemos realizar un proceso de calibración similar al proceso que seguimos para el comedero: Realizar medidas de la fuerza que ejerce el bebedero vacío, bebedero lleno y a partir de hay estimar el nivel de agua del bebedero.

Hay que tener en cuenta que el bebedero nunca estará realmente vacío, ya que la bomba que utiliza necesita un nivel mínimo para funcionar, por debajo del cual aunque la fuente siga conteniendo agua, la bomba no funcionara, por lo que el nivel no bajara.

En este caso, lo que esperamos ver son pocas visitas al día del animal, de 2 a 3, basándonos en los patrones observados en el animal durante la convivencia diaria con ella, ya que tiende a hacer pocas visitas, ingiriendo una cantidad considerable de agua en cada visita en relación a su tamaño.

6. Diseño software Sensores-Servidor

Con las tecnologías ya elegidas, es el momento de comenzar con el diseño del software de los diferentes elementos que vamos a utilizar, comenzando por los sensores para después tener en cuenta el software del servidor, que sera mas complejo.

6.1 Consideraciones generales

El objetivo de este modulo de software es la comunicación entre el servidor y los sensores, para permitir que el servidor almacene en la base de datos las series temporales con la información de los diferentes sensores, de cara a mantener hacer un histórico.

Para ello, haremos uso de la conectividad BLE(*Bluetooth Low Energy*) presente tanto en el dispositivo servidor como en los controladores de los sensores, permitiendo una comunicación de poca capacidad pero con poco consumo energético, lo que es adecuado para el tipo de dispositivos con los que estamos trabajando.

El funcionamiento de la comunicación se basara en el uso de dos tipos de nodos, nodo central y nodos periféricos, correspondiendo con sistema servidor y sistema sensores respectivamente, siendo los periféricos tantos como sensores tengamos.

El nodo central escaneara los dispositivos Bluetooth periódicamente hasta que encuentre periféricos que conozca, con los que pasa a establecer conexión. Una conectado, el nodo servidor esperara a recibir notificaciones de los nodos periféricos para pasar a leer información de las diferentes características presentes en el servicio BLE correspondiente.

Por su parte, los nodos sensores, al encender lo que harán es pasar a anunciar su existencia como periférico Bluetooth, mientras esperan recibir una solicitud de conexión por parte del servidor, pasando a enviar notificaciones cuando tengan datos disponibles una vez la conexión con el servidor se establezca.

Según el tipo de sensor que sea, las características de su servicio se ajustaran a la información que este tenga disponible, teniendo un tipo de servicio por cada tipo de sensor.

6.2 Software sensores

El software de todos los controladores de los sensores va a ser similar entre si, siguiendo siempre el mismo esquema:

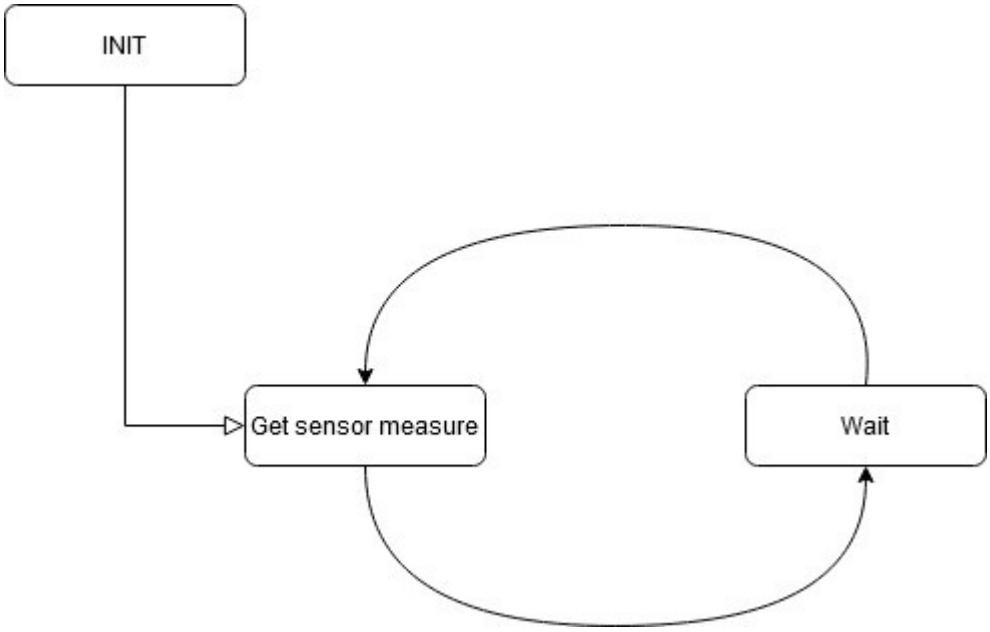

Figura 13: Diseño software sensores

A modo general, el diseño que vamos a utilizar para todos los sensores va a consistir en tres pasos:

- INIT: Estado inicial, el dispositivo deberá realizar las tareas de inicialización necesarias para que el *hardware* funcione, así como para configurar el modulo Bluetooth del dispositivo en cuestión de acuerdo a nuestras necesidades.
- GET MEASURE: Estado de medición, donde el dispositivo correctamente inicializado, pasara a tomar las diferentes medidas de los sensores, que serán almacenadas en la memoria RAM del controlador encargado de gestionar los sensores.

- WAIT: Estado de espera, donde el dispositivo deberá publicar las medidas realizadas en la diferentes características del servicio Bluetooth junto con un característica adicional que sea un contador del numero de paquete, y notificar al servidor de su disponibilidad vía Notificación, para a continuación esperar el tiempo predefinido y volver al estado anterior.

Este diseño es sencillo y cumple con los requisitos establecidos para el software de los sensores.

Adicionalmente, el modulo de la mascota podrá contar con una tarea que permita controlar el nivel de batería del sensor, pero no es imprescindible para el correcto funcionamiento del sistema.

A nivel de programación, este diseño se traslada a dos tareas en *FreeRTOS.* Una tarea que se encargue de recoger la información de los sensores y ponerla en memoria y otra tarea que se encargue de procesar esa información y hacerla visible para el servidor vía modulo Bluetooth.

Para la sincronización entre las dos tareas, haremos uso de un semáforo, que no permita que se procesen los datos de la memoria mientras estos estén siendo actualizados, de cara a evitar que el servidor pudiera leer datos cruzados de dos paquetes diferentes.

Por ultimo, una vez este disponible un paquete de datos, se le notificara al servidor, comenzando este el proceso de leer las diferentes características que tenga definidas el servicio en cuestión, asi como el contador del paquete para asegurar que los datos que se han recibido son frescos.

En el caso del modulo del animal, adicionalmente se podrá contar con la existencia de una tarea que permita consultar el estado de la batería del modulo M5StickC vía la pantalla LCD que incluye, utilizando para ellos los botones con los que cuenta el modulo.

6.3 Software servidor – Sensores

El software de los sensores seguirá el siguiente esquema:

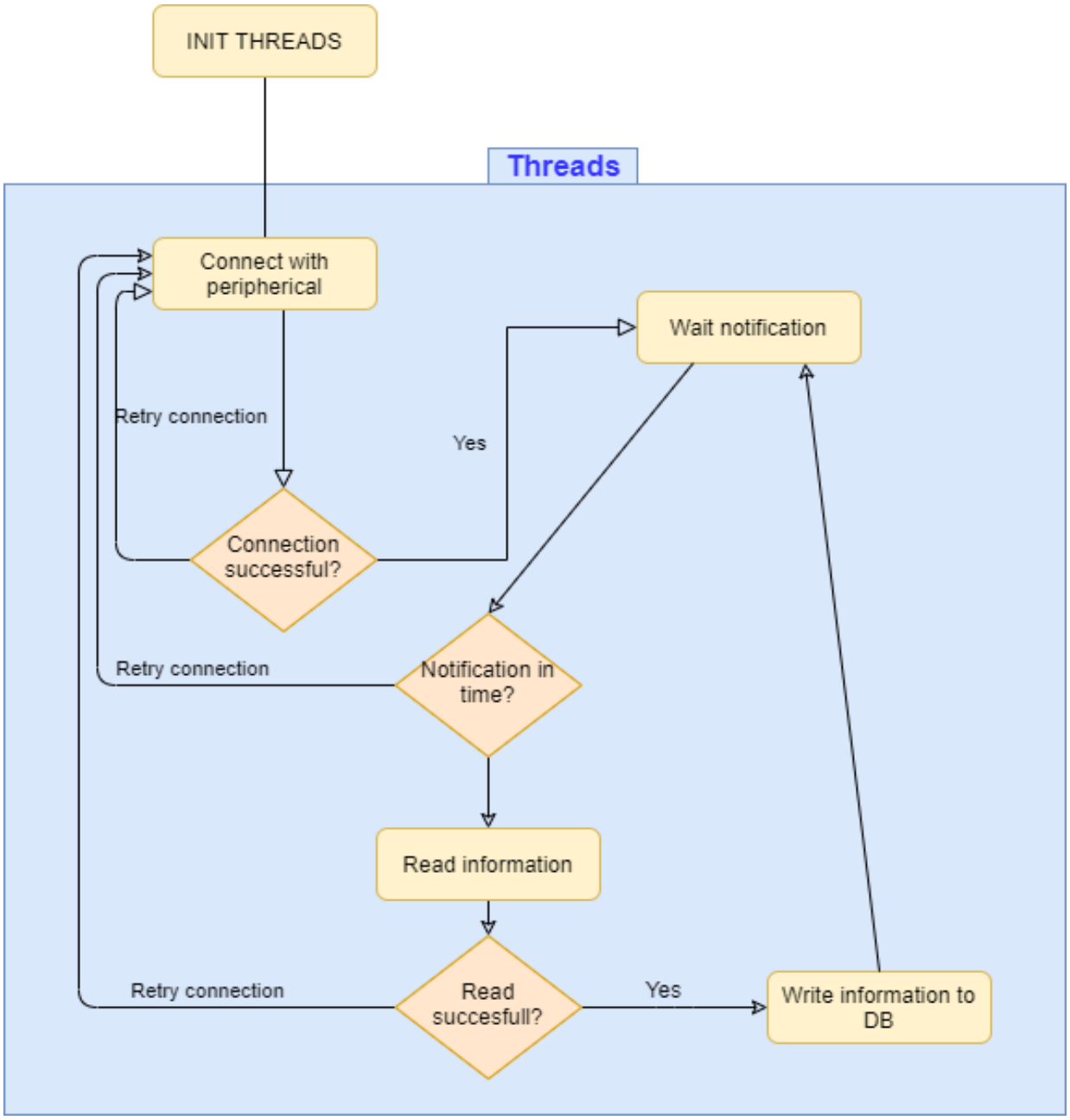

Figura 14: Diagrama software sensores servidor

De forma general, el sistema se compondrá de las siguientes etapas:

- INIT THREADS: Estado de inicialización, donde el sistema configurá el sistema para su correcto funcionamiento, arrancando el *hardware* necesario, aplicando las configuraciones necesarias para el funcionamiento del sistema y lanzando los *threads* correspondientes.
- CONNECT WITH PERIPHERICAL: Estado de conexión, donde cada hebra intentará conectar con el sensor que tenga configurado, re-intentando la conexión de forma periódica, dejando un intervalo de tiempo entre intentos, y pasando al siguiente estado en caso de éxito.
- WAIT NOTIFICATION: Una vez se haya establecido conexión con los dispositivos, el sistema pasara a un estado de espera, donde esperará recibir una notificación de los periféricos vía Bluetooth que le indiquen que tienen información disponible para enviar, en un tiempo preestablecido, volviendo a re-intentar la conexión con el dispositivo en caso de que la notificación no se reciba.
- READ INFORMATION: Estado de lectura, donde el sistema leerá la información de los diferentes sensores y, tras comprobar que el contador de paquete es superior al ultimo que recibió, procederá al siguiente paso.
- WRITE INFORMATION: Estado de escritura, donde cada paquete de información recibido será enviado a la base de datos para su almacenamiento y posterior tratamiento, volviendo al estado de esperar notificación tras su finalización.

Este diseño cumple con los requisitos establecidos en el proyecto, siendo sencillo y completo.

Es de destacar que este sistema no funcionara de forma secuencial, ya que, al contar el proyecto con diferentes conjuntos de sensores, el sistema deberá ser capaz de procesar simultáneamente las peticiones de los diferentes sensores, almacenando la información de cada uno de ellos en su correspondiente entrada en la base de datos.

A nivel de programación, este modulo se implementara en *Python*, haciendo uso de la librería *BluePy* para gestionar la comunicación BLE con los diferentes controladores de los sensores, de los que el sistema deberá recibir información.

Además, contara con una interfaz con *InfluxDB*, para permitir el envío de la información a la base de datos, para permitir su posterior gestión y procesado por el modulo software correspondiente.

La información relativa a los sensores se configurara mediante un archivo de configuración, para evitar que la información tenga que aparecer explícitamente en el código fuente del modulo.

Para la realización del archivo, haremos uso del formato *JSON*, que es un formato derivado de XML de uso generalizado en la web, lo que nos permitirá definir la información necesaria de forma que sea sencilla de entender al modulo software.

Dicho fichero, incluirá las direcciones *MAC* de los diferentes módulos de BLE, para poder identificarlos, y así saber a cuales se debe conectar, ademas de los servicios y características que estos tienen, de tal manera que cambiarlos sea tan sencillo como cambiar la configuración en el archivo, sin necesidad de modificar el código fuente del programa.

Además, también incluirá la información necesaria para establecer la conexión con la base de datos, que si bien en este proyecto la tendremos en la misma maquina, a través de la configuración se podría especificar un servidor que se encontrara en otra maquina.

Figura 15: Ejemplo JSON configuración

6.4 Software servidor – Base de datos

#	time	Accelerometer_X	Accelerometer_Y	Accelerometer_Z	Giroscope_X	Giroscope_Y	Giroscope_Z	Sensor
1	08/12/2020 11...	0,048828125	0,0126953125	1,079833984375	6,77490234375	-5,92041015625	56,6768684387...	gato
2	08/12/2020 11...	0,053466796875	0,010498046875	1,046875	6,28662109375	-6,4697265625	55,7037925720...	gato
3	08/12/2020 11...	0,050048828125	-0,004638671875	1,06298828125	6,4697265625	-6,77490234375	55,5416145324...	gato
4	08/12/2020 11...	0,375732421875	-0,70849609375	0,5087890625	37,7197265625	18,73779296875	55,8935127258...	gato
5	08/12/2020 11...	0,17431640625	-0,241455078125	-0,096923828125	148,8037109375	103,02734375	55,6242332458...	gato
6	08/12/2020 11...	-0,15576171875	-0,134521484375	0,439208984375	-60,546875	-239,80712890...	57,3133430480...	gato
7	08/12/2020 11...	-0,893798828125	0,469970703125	0,3544921875	-2,8076171875	10,25390625	55,8567924499...	gato
8	08/12/2020 11...	0,087890625	-0,0166015625	1,07666015625	6,4697265625	-5,79833984375	57,4602203369...	gato
9	08/12/2020 11...	0,0966796875	0,0029296875	1,07177734375	5,9814453125	-6,40869140625	55,8170127868...	gato
10	08/12/2020 11...	0,09619140625	-0,017578125	1,086669921875	6,04248046875	-6,2255859375	55,0122413635...	gato
11	08/12/2020 11...	0,088134765625	-0,00732421875	1,08642578125	5,7373046875	-6,34765625	57,4143218994...	gato
12	08/12/2020 11...	0,08837890625	-0,008544921875	1,06640625	7,32421875	-6,34765625	54,2839660644...	gato
13	08/12/2020 11...	0,0908203125	-0,000732421875	1,074462890625	6,591796875	-6,65283203125	55,9669532775...	gato
14	08/12/2020 11...	0,097412109375	-0,014892578125	1,082763671875	6,65283203125	-5,37109375	56,9063644409...	gato
15	08/12/2020 11...	0,09912109375	-0,01123046875	1,066162109375	6,40869140625	-5,615234375	56,4626693725...	gato
16	08/12/2020 11...	0,0947265625	-0,013916015625	1,06591796875	6,4697265625	-5,92041015625	56,6615676879...	gato
17	08/12/2020 11...	0,098876953125	-0,015625	1,075927734375	6,4697265625	-5,79833984375	54,5348854064...	gato
18	08/12/2020 11...	0,092529296875	0,005859375	1,05810546875	6,28662109375	-5,67626953125	55,9791908264...	gato
19	08/12/2020 11...	0,08740234375	0,002197265625	1,0693359375	6,65283203125	-5,67626953125	57,9712371826...	gato
20	08/12/2020 11...	0,091064453125	0,001953125	1,071533203125	7,38525390625	-6,16455078125	55,6089363098...	gato
21	08/12/2020 11...	0,094970703125	-0,013916015625	1,06494140625	6,103515625	-5,859375	55,6303558349...	gato

Figura 16: Captura de datos de InfluxDB via InfluxDBStudio

La información recibida por parte de los sensores será almacenada en una base de datos de tipo InfluxDB, que correrá en un contenedor Docker usando la imagen oficial de la base de datos proporcionada por la propia Influx.

Influx almacena la información utilizando tuplas identificadas por la marca de tiempo, conteniendo cada tupla una serie de *Tags* y de *Fields,* siguiendo ambos tipos de campos una estructura de pareja “llave-valor”.

No obstante, existe una diferencia entre ambos tipos de campo, consistente en la indexación, ya que los *Tags* se indexan automáticamente para permitir consultas sobre su valor, mientras que por el contrario los *Fields* no van indexados, siendo mas sencillos de tratar pero requiriendo una búsqueda secuencial que recorra todos los valores almacenados en la base de datos en caso de buscar un valor en concreto.

En nuestro caso en particular, el esquema de almacenamiento que vamos a seguir será bastante sencillo consistiendo en el uso de un *Tag* y de uno o varios *Fields* por sensor:

- *TAG*: Utilizaremos un *Tag* para identificar que sensor esta registrando la medida, que en el caso de este proyecto podrá ser gato, comedero o bebedero.
- *FIELDS:* Utilizaremos de uno a N *Fields* por cada medida, correspondiendo cada uno a una de las medidas que realice el sensor(6 en el caso de los sensores del gato, 1 en el resto).

Este diseño es sencillo y permitirá consultar los datos agrupándolos por sensor de forma sencilla y eficiente, resultando sencillo representarlo gráficamente utilizando Grafana como capa de interfaz para los datos recogidos.

6.5 Software servidor – Visualización de datos

Figura 17: Grafana dashboard

Una vez tenemos recogida y almacenada la información de los sensores, el ultimo paso restante es mostrarla a los usuarios del sistema. Con esa finalidad, utilizaremos el software Grafana, corriendo sobre Docker gracias a la imagen proporcionada por la empresa desarrolladora.

Grafana es un software de código abierto diseñado para mostrar, sacar métricas y analizar datos almacenados en base de datos, basándose en tecnología cliente-servidor utilizando como core un servidor web, permitiendo la interacción de los usuarios vía navegador web.

En nuestro caso, vamos a usar Grafana con dos finalidades distintas:

- Mostrar al usuario los datos generados por la mascota de forma grafica, permitiéndole acotar los resultados según intervalo de tiempo definidos
- Generar alertas que avisen al usuario de la necesidad de reponer comida o agua, cuando estas alcance un nivel critico

Para mostrar los datos, realizaremos tres *Dashboards* distintos. Un *Dashboard* es una pantalla de presentación en Grafana, donde se organizan diferentes paneles que muestran la información seleccionada al usuario. Los *Dashboards* que utilizaremos son:

- Actividad, donde se mostraran los valores de los sensores del acelerómetro y del giroscopio.
- Comedero, donde se mostrara el nivel de comida del comedero según la distancia al centro del comedero, asi como los intervalos en que el animal este presente en el comedero.
- Agua, donde se mostrara la cantidad de agua disponible en la fuente, así como los momento en que el animal este interactuando con el.

Además, también se contarán con dos alarmas:

- Nivel de comida, donde se emitirá una alerta cuando la distancia superar un limite predeterminado
- Nivel de agua, donde se emitirá una alerta cuando el peso quede por debajo de un limite.

Dichas alertas se pondrán consultar en Grafana, así como ser recibidas por el usuario vía email auto generado, evitando así que el animal pueda pasar hambre o sed por falta de suministro.

Toda esta funcionalidad forma parte del nucleo de Grafana, por lo que su implementación es sencilla de realizar, pudiendo ser realizada sin coste gracias a la orientación de código abierto del proyecto, lo que permite que podamos ejecutarlo en nuestra maquina sin coste alguno, cumpliendo en todo momento con los terminos de la licencia.

7. Diseño hardware de los sensores

Además de diseñar el software necesario para el funcionamiento de este proyecto, la naturaleza de este proyecto también hace necesario contar con un hardware especifico donde ejecutar el software.

7.1 Consideraciones generales

Para el desarrollo de la parte hardware de este proyecto, nos basamos en la selección de dispositivos que tuvieran bajo coste de adquisición y fueran amigables para el desarrollo y para el uso del usuario, utilizando además como criterio la disponibilidad de material antiguo del que ya dispusiera, al reducir el coste de la implementación del proyecto.

Para su implementación en el ámbito domestico he tratado de utilizar elementos disponibles en cualquier hogar, para poder poner a prueba los conceptos planteados en este proyecto, sin incurrir en el uso de material muy especifico o de alto coste.

La única excepción a esta regla ha sido la utilización de una estación de soldadura amateur, que si bien no es de alto coste(~30€), entiendo que no es un elemento que este presente por lo general en el ámbito domestico, pero su utilización ha sido imprescindible para el montaje de unos de los sensores, al ser enviado por el vendedor en forma de elementos sueltos, cuya conexión ha requerido de la soldadura de varios elementos.

El resto de elementos utilizados(herramientas genéricas, cinta adhesiva, *Tuppers*...) si están disponible de forma normal en los hogares, por lo que desde el proyecto interpretamos que pueden ser reutilizados para el proyecto.

Es importante recalcar que la implementación hardware realizada responde a la idea de una prueba de concepto o prototipo, por lo que su implementación esta pensada para ser sencilla y permitir la prueba de los diferentes elementos, mas que para ser una implementación firme, fiable y resistente en el tiempo.

7.2 Sensor animal

El sensor que ira acoplado al animal se basa en el uso de un M5StickC, que integra en su interior un ESP32, con forma de pulsera para poder ser utilizado como kit *smartwatch*, gracias a la correa que incorpora, así como a la integración de una pantalla LCD y de una pequeña batería.

En este caso, la preparación del sensor para su uso con el animal ha sido muy sencillo, consistiendo simplemente en abrir un agujero extra de tal manera que la correa tenga diámetro suficiente para poder ser atada alrededor del cuello del animal sin que le haga daño.

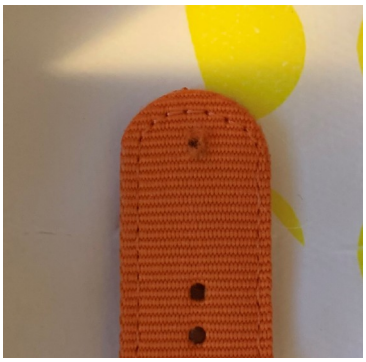

Figura 18: Agujero extra

Tras esta pequeña modificación, el dispositivo quedo listo para ser utilizado con el animal:

Figura 19: Sensor montado

A pesar de lo voluminoso que parece el montaje, nuestro sujeto de pruebas fue colaborador y acepto llevar el dispositivo de buen grado, sin que parecería molestarle en exceso, llevando una nivel de actividad normal.

7.3 Sensor comedero

El siguiente sensor a montar, fue el sensor del comedero, cuya función consiste en medir la distancia entre su punto fijo y el comedero, considerando que a mayor distancia menor cantidad de comida queda presente en el comedero.

El montaje del *hardware* fue sencillo, consistiendo en conectar utilizando los cables que vinieron con el sensor, con la placa utilizada, en este caso la Launchpad MSP432, de Texas Instruments, quedando el montaje de la siguiente manera:

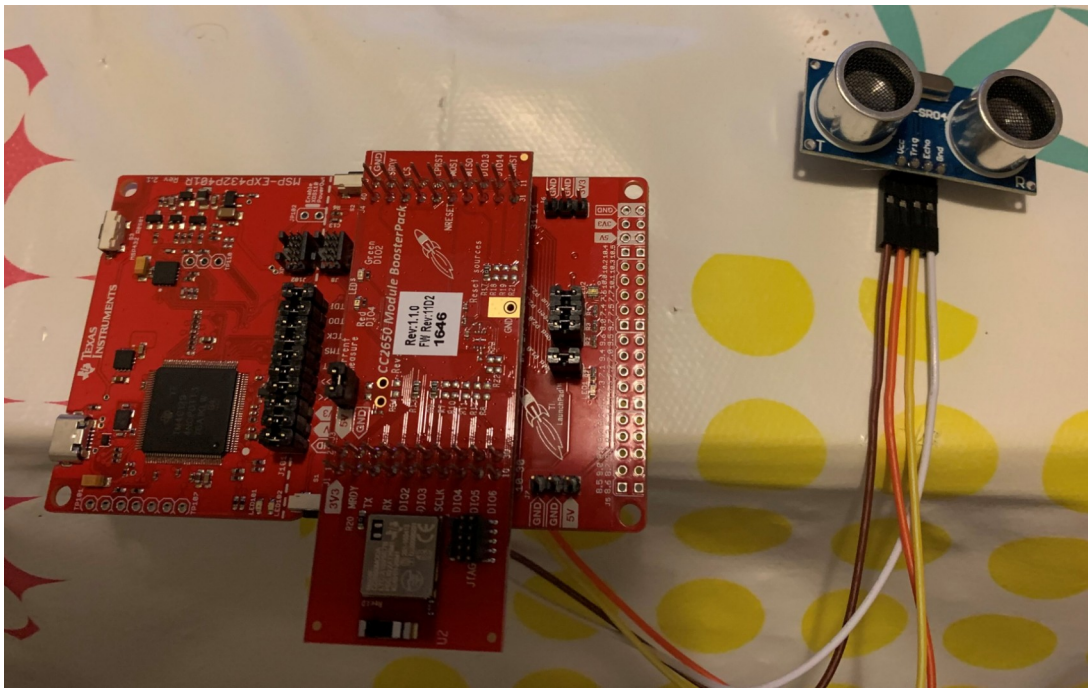

Figura 20: Sensor comedero

Para el montaje del sensor, aproveche el hueco que deja un mueble donde coloco el comedero, permitiéndome montar el sensor con facilidad utilizando cinta adhesiva, lo que me permitió obtener un anclaje firme de forma sencilla.

Además, utilice un poco mas de cinta para sujetar el comedero al suelo, lo que lo mantiene centrado con respecto al sensor, de manera que el sensor siempre apunta al centro del comedero.

El resultado del comedero es el siguiente:

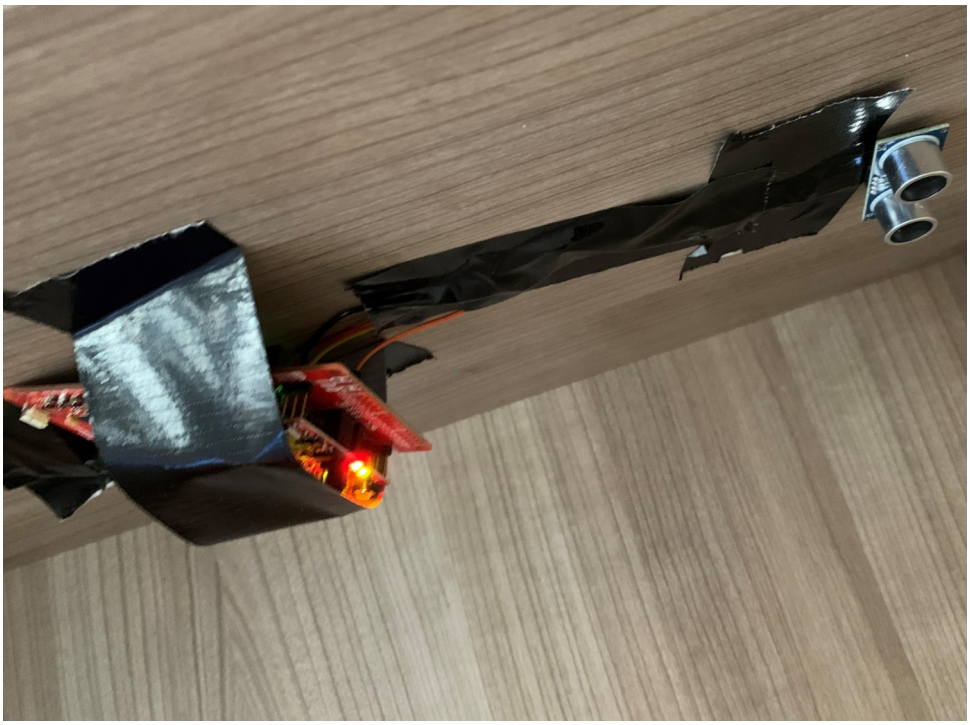

Figura 21: Sensor comedero montado

Figura 22: Sujeción comedero

El resultado del montaje fue aceptado de buen grado por el sujeto de pruebas, quien tras olisquear el montaje decidió aceptarlo y utilizarlo.

7.4 Sensor bebedero

El ultimo sensor que utilizaremos para este proyecto es el sensor del bebedero, consistente en una celda de carga, que medirá el peso del bebedero, los que nos permitirá estimar la cantidad de agua restante, así como la interacción del animal con el bebedero.

Este sensor es el mas complicado de montar, ya que va a funcionar en un ambiente con posible presencia de agua, por lo que tenemos que proteger la electrónica de entrar en contacto con el agua en medida de lo posible, ya que de lo contrario sera muy fácil que se dañe y deje de funcionar correctamente.

Ante este problema, la solución por la que hemos optado ha sido por montar la electrónica en un recipiente de plástico, utilizando para ello un *tupper,* resultando una forma económica de acceder a un recipiente razonablemente cerrado.

Además, dado que la electrónica que estamos usando es de baja potencia, no tenemos problemas de temperatura, a pesar de funcionar en un ambiente sin ningún tipo de refrigeración, lo que inicialmente me parecía un posible problema.

El resultado del montaje es el siguiente:

Figura 23: Sensor bebedero

Quedando fuera del recipiente la célula de carga:

Figura 24: Célula de carga

Con los elementos preparados, el ultimo paso es montar el sensor bajo el bebedero, de tal manera que podamos medir su peso, para controlar la cantidad de agua disponible en todo momento.

Para ello, utilizamos nuevamente cinta adhesiva, fijando los diferentes elementos a una alfombrilla de plástico, dado que el bebedero lo tenemos ubicado en la bañera, para minimizar la cantidad de agua a recoger en caso de que el gato se ponga a jugar con el agua.

El resultado del montaje es el siguiente:

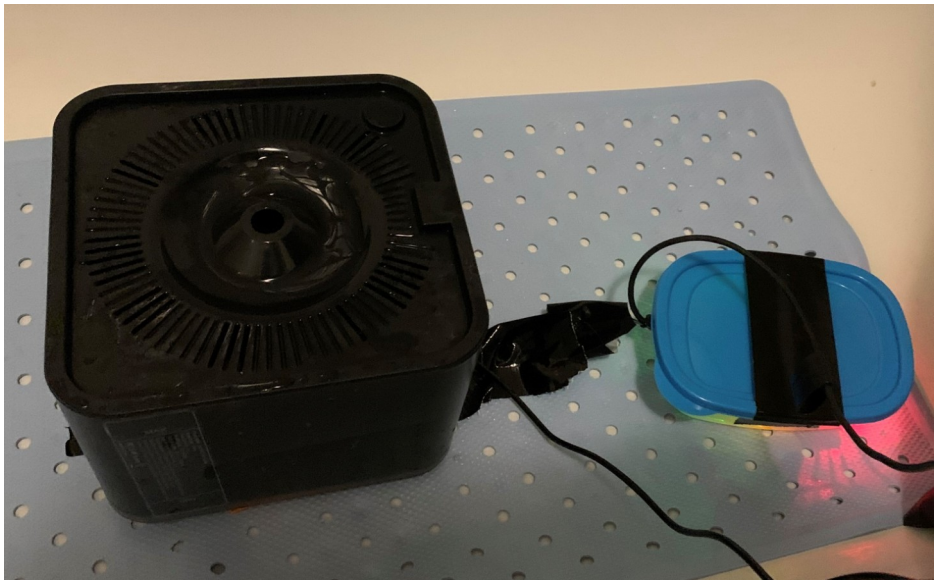

Figura 25: Montaje sensor bebedero

El montaje, aun no siendo completamente estanco, es resistente a pequeñas cantidades de agua, resultando su montaje sencillo al estar basado en cinta adhesiva, resultando suficiente para probar el concepto que consideramos en este proyecto.

En este caso, tras la curiosidad inicial y un poco de jugueteo, el animal termino por aceptar el montaje:

Figura 26: Gato con sensor de bebero

8. Resultados

Con los sensores montados y funcionando, vamos a dedicar un apartado a comentar los resultados obtenidos tras la realización de este proyecto, considerando para cada apartado su viabilidad técnica, su precisión y sus principales carencias.

En general los resultados obtenidos han sido satisfactorios, si bien no todos los puntos han funcionado tan bien como se esperaba, debido a las limitaciones del *hardware* utilizado, que no siempre es todo lo fiable que se esperaba inicialmente.

8.1 Resultado del sensor animal

El sensor animal ha sido capaz de recoger los datos adecuadamente, resultando sencillo distinguir diferentes periodos de actividad del animal monitorizado, como se aprecia en la siguiente captura de pantalla:

Figura 27: Visualización del sensor animal con Grafana

En la captura podemos apreciar un periodo de actividad intensa, con grandes cambios en los valores obtenidos, aproximadamente entre las 15.10 y las 15.25, durante el cual el sujeto de pruebas estuvo jugando con un ratón en caña.

Después del rato del juego, el sujeto de pruebas procedió a dormir, por lo que los valores que se aprecian en la captura se ven estables, dado que el animal dejo de moverse.

En la captura también se aprecia un hueco sin información, correspondiente a falta de batería por parte del sensor, cuya pequeña batería ofrece poca autonomía, pero siendo rápida de cargar.

En general, el resultado de este sensor es satisfactorio, dado que nos permitir distinguir el nivel de activación del animal en base a los cambios en los valores que registran, si bien los valores no se pueden interpretar en valor absoluto.

Es falta de precisión en los valores deriva del sistema de sujeción utilizado, que si bien es lo bastante laxo para que el animal no este excesivamente incomodo con el, permite que el sensor pueda rotar alrededor de su cuello, haciendo inviable tomar la información de los giroscopio como indicador de la posición del animal.

No obstante, esta falta de precisión se compensa con la indicación de los acelerómetro, que nos permiten saber si el gato esta en movimiento o en reposo según los cambios en aceleración, resultando este apartado un éxito, si bien se podría definir mejor utilizando una frecuencia de sensado mayor, a costa de utilizar mas espacio de almacenamiento en la base de datos.

Mención especial merece la autonomía del sensor utilizado, en este un M5StickC, cuyo resultado ha sido bastante inferior al esperado, haciendo necesaria su recarga cada hora aproximadamente, lo que hace impráctico su uso mas allá de la prueba de concepto.

De cara al desarrollo de un sensor en un formato mas comercial, seria necesario aumentar la batería y reducir el volumen del encapsulado, posiblemente eliminando la pantalla LCD, que en este caso no nos es útil al no aportarnos ninguna información.

8.2 Resultados sensor comedero

El sensor del comedero ha funcionado adecuadamente, siendo capaz de registrar el nivel de comida del comedero en base a la distancia desde su ubicación al centro del comedero, como se puede apreciar a continuación:

Figura 28: Resultado sensor comedero, valores invertidos en la grafica

En la figura se puede apreciar como el nivel se observa estable dentro de un margen de error(el sensor no tiene una precisión del 100%), tras lo cual hay un salto que se corresponde con la detección animal que se acerca a comer, lo que provoca una disminución del nivel del comedero.

Este es el sensor que mejor ha funcionado del proyecto, demostrando la viabilidad del concepto mas allá de toda duda, siendo sencillo de programar y resultando toda la electrónica necesaria de bajo coste económico.

En caso de querer realizar una implementación con un objetivo mas comercial de dicho sensor, lo único que se requeriría seria una mayor integración de la electrónica utilizada, para eliminar el cableado intermedio, que puede reducir la durabilidad del conjunto, sin ser necesaria ninguna modificación adicional.

El único punto de conflicto que puede presentar este sistema es el centrado del sensor con respecto al centro del comedero, ya que es fácil que se desvíe en caso de no estar bien sujeto, requiriendo en su caso de una revisión del centraje.

El otro punto de preocupación que me causaba este sensor era que pudiera causar molestias al animal, dado que funciona con ultrasonidos, y hay animales con mayor

espectro de audición que los humanos, pero en este caso el sensor no parece causar ningún efecto en el sujeto de pruebas, por lo que este punto de preocupación queda resuelto.

8.3 Resultados sensor bebedero

El sensor del bebedero ha resultado el mas problemático de los tres implementados en este proyecto, al resultar el sensor extremadamente sensible al posicionado exacto de los elementos sobre la célula de carga.

Además, hay que contar que este sensor, por su forma de funcionamiento, no puede ser “enchufar y usar”, dado que al arranque realiza una calibración a cero, durante la que toma la fuerza que este midiendo en ese momento como cero, por lo que se debe arrancar sin peso encima.

En la siguiente figura, se muestran los resultados obtenidos:

Figura 29: Resultados sensor bebedero

Al principio de la figura, se ven valor extraños con mucha variación, correspondientes con el proceso de centrado del bebedero con respecto al sensor, tras el cual se ve un cambio de peso provocado por la interacción del animal con el bebedero.

Por ultimo, se ve que hay otro cambio tras el cual el sensor empieza a dar valores erróneos. Nuevamente, este cambio esta producido por movimiento del bebedero sobre el sensor causados por el animal.

Tras analizar las evidencias, la conclusión es que la configuración elegida no es adecuada para monitorizar de forma precisa el peso de la fuente, requiriendo de un centrado extremadamente preciso para funcionar, lo hace poco practica su utilización en un entorno real.

Esto se debe a que el sensor elegido no ha cumplido con las expectativas, teniendo una fiabilidad baja y siendo muy sensible a cualquier cambio. Este problema podría deberse a algún defecto concreto de la unidad empleada, pero al no disponer de ningún recambio, no puedo más que descartar el sensor como una alternativa aceptable.

Si bien la teoría detrás del funcionamiento sigue siendo valida, para una aplicación comercial seria necesario cambiar de dispositivo por otro que permita medir el peso con mas fiabilidad.

9. Conclusiones

A lo largo de este proyecto se ha realizado el desarrollo de un sistema que permita monitorizar diferentes parámetros relativos a la vida y necesidades de nuestras mascotas, comenzando con un estudio de mercado y una selección de tecnologías, contando ademas con la implementación necesaria para hacer funcionar los diferentes dispositivos, así como el servidor.

La implementación de los dispositivos ha resultado mas compleja y difícil de lo que estimé inicialmente, ralentizando el desarrollo del proyecto, pero resultando didáctica a nivel personal.

Parte de esta dificultad se ha debido a mi desconocimiento sobre el funcionamiento del BLE(*Bluetooth Low Energy*), que ha resultado ser mucho mas compleja de lo que estime inicialmente, requiriendo tanto estudio, como pruebas para conseguir el funcionamiento necesario por parte de los sensores.

El otro punto de dificultad que me he encontrado durante el desarrollo ha sido el montaje de los sensores, que ha requerido del uso de un soldador para su montaje, con el cual ha sido mi primera vez, por lo que conseguir un resultado funcional me ha resultado complicado, requiriendo de múltiples intentos.

Por otra parte, la implementación del servidor me ha sorprendido gratamente, topando con un entorno estable y una programación sencilla pero funcional, como es Python, lo que ha permitido compensar en parte las dificultades encontradas en el desarrollo de los sensores.

Parte de esta sorpresa deriva de Docker, en el cual he encontrado un aliado sencillo de utilizar y fiable. No es esta mi primera experiencia con el sistema, pues lo utilice hace unos años, siendo entonces mucho mas inestable y problemático. Desde entonces ha mejorado mucho y para bien.

La otra parte que me ha sorprendido gratamente ha sido la *Raspberry Pi*, en su cuarta iteración, contando con le versión con 4gb de RAM. Esta versión es capaz de ofrecer un rendimiento cercano a un ordenador en un entorno de escritorio, siendo capaz de ejecutar una versión completa de Ubuntu, y permitiendo la programación tanto del

servidor como los sensores. Teniendo en cuenta su bajo coste, me hace pensar en el futuro que tienen los procesadores ARM en entornos de escritorio.

La otra sorpresa en este proyecto ha venido por la comunidad existente alrededor del ESP32. Nuevamente, se trata de una familia de dispositivos de la que había oído hablar, pero que ha resultado ser mucho mas fiable y contar con muchos mas recursos de los que pensaba que existian.

Respecto a los resultados, solo se pueden considerar un éxito parcial, pues uno de los sensores planificados inicialmente ha resultado no ser tan fiable como se esperaba, haciendo que sus resultados carezcan de la fiabilidad necesaria para ser considerados utilizables.

Los otros dos sensores planteados han sido un éxito, demostrando la viabilidad del concepto, resultando viable su implementación y ofreciendo resultados validos y razonablemente fiables, por lo que estoy satisfecho con su desempeño, desde la perspectiva de la demostración de la validez del concepto.

Estos sensores son el sensor de distancia por ultrasonidos y el acelerómetro, que han demostrado ser alternativas viables de muy bajo coste de adquisición y programación sencilla, contando sus entornos de desarrollo con suficientes ejemplos y ayudas para que sea sencillo utilizarlos.

En el otro lado, tenemos las tecnologías empleadas en el servidor, siendo estas InfluxDB y Grafana. No contaba con experiencia previa en ninguna de las dos tecnologías, y ambas me han sorprendido gratamente.

InfluxDB es una base de datos orientada a series temporales, que cuenta con documentación oficial de buena calidad y un diseño bastante lógico, resultando sencilla de configurar y de utilizar vía API, por lo que trabajar con ella me ha resultado una experiencia grata y constructiva.

Caso similar es Grafana, cuya interfaz de usuario es clara e intuitiva, además de contar con recursos en linea de buena calidad, por lo que su uso ha sido muy sencillo, produciendo resultados claros con poco esfuerzo, y resultando muy sencilla de instalar gracias al contenedor de Docker que la propia empresa desarrolladora ofrece de forma gratuita.

En general, este proyecto ha sido muy instructivo, permitiendo familiarizarme con una serie de tecnologías sobre las que había oído hablar, pero con las que no tenia experiencia propia, lo que contribuyo a que mi planificación inicial y estimación de esfuerzos no contaran con la exactitud que yo esperaba, habiéndome obligado a reconsiderar mi planificación para ser capaz de llegar a los resultados obtenidos.

Figuras

Figura 29	Resultados sensor bebedero

10. Bibliografía

- PetPace: https://petpace.com/smart-sensing-collar/
- Tractive: https://tractive.com/es/pd/gps-tracker-cat?shopCountry=AT&gclid=Cj0KCQiAy579BRCPARIsAB6QoIaq9qHMooN-5DtBLubtPAheZdN6LZOy7WVCRcqAUQ6Ri1Eg3i4hzWgaAs6WEALw_wcB
- Petcare HD: https://www.ikohs.com/es/comprar-mascotas/64299-petcare-hd-comedero-dispensador-automatico-perros-y-gatos.html?id_c=126171&gclid=Cj0KCQiAy579BRCPARIsAB6QoIZ6EOBQerIfmi05v1IUNwXktY-Wb8hj0HzNMsu0TnQEUmOJA5aijW4aAmiBEALw_wcB
- Mi Band 5: https://www.mi.com/global/mi-smart-band-5/specs/
- Arduino Lilypad: https://www.mouser.com/catalog/specsheets/lilypad.pdf
- M5Stick C: https://m5stack.com/products/stick-c
- Arduino Uno: https://store.arduino.cc/arduino-uno-rev3
- MSP 432: https://www.ti.com/store/ti/en/p/product/?p=MSP-EXP432P401R
- Intel NUC: https://www.intel.es/content/www/es/es/products/boards-kits/nuc.html
- Raspberry Pi: https://www.raspberrypi.org/products/raspberry-pi-4-model-b/specifications/?resellerType=home
- Free RTOS: https://www.freertos.org/about-RTOS.html
- Python: https://www.python.org/doc/essays/blurb/
- Influx DB: https://www.influxdata.com/time-series-platform/
- Grafana: https://grafana.com/grafana/
- Docker: https://codeahoy.com/2019/04/12/what-are-containers-a-simple-guide-to-containerization-and-how-docker-works/
- MPU6886: https://docs.m5stack.com/#/en/unit/imu?id=application
- Sueño gatos: https://petcentral.chewy.com/how-much-should-your-adult-cat-sleep/
- HC-SR04: https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf
- HX-711: https://cdn.sparkfun.com/assets/b/f/5/a/e/hx711F_EN.pdf

Anexo I – Código fuente

food_sensor

```
#ifndef __CC3200R1M1RGC__
// Do not include SPI for CC3200 LaunchPad
#include <SPI.h>
#endif

#include <BLE.h>

#include <stdio.h>

// Core library
#include "Energia.h"
#include "Task.h"

float distancia;
long tiempo;

int pin_trigger = A4;
int pin_echo = A6;

char valueR = 0;

Task taskManager;

/* Declare Sensor BLE Service Characteristics */
BLE_Char DistanciaChar =
{
 {0xF1, 0xFF},
 BLE_READABLE,
```

```
 "Distancia"
};

BLE_Char sensorNotificationChar =
{
 {0x01, 0xFF},
 BLE_NOTIFIABLE,
 "Distancia notificacion"
};

/* BLE LED Service is made up of BLE Chars */
BLE_Char *sensorServiceChars[] = {&DistanciaChar,&sensorNotificationChar};

/* LED Service Declaration */
BLE_Service sensorService =
{
 {0xF0, 0xFF},
 2, sensorServiceChars
};

void sensorTask(){
 while(true){
  digitalWrite(pin_trigger, HIGH);
  delayMicroseconds(10);
  digitalWrite(pin_trigger, LOW);
  tiempo=pulseIn(pin_echo, HIGH);
  // 0.017 taken from example
  distancia = 0.017*tiempo;
  Serial.print(distancia);
  Serial.println(" cm");

  ble.writeValue(&DistanciaChar, distancia);
  ble.writeValue(&sensorNotificationChar,1);
```

```
  delay(10000);
 }
}

void heartBeatTask(){
 while(true){
  valueR = 1 - valueR;
  digitalWrite(BLUE_LED, valueR);
  delay(1000);
 }
}

void setup() {
 Serial.begin(115200);

 ble.begin();

 /* Add and intialize LED service */
 ble.addService(&sensorService);
 ble.writeValue(&DistanciaChar, 0);
 ble.writeValue(&sensorNotificationChar,0);

 Serial.println("Start");
 pinMode(pin_trigger, OUTPUT);
 pinMode(pin_echo, INPUT);
 digitalWrite(pin_trigger, LOW);

 pinMode(BLUE_LED, OUTPUT);

 /* Start Advertising */
 ble.setAdvertName("SensorComida");
 ble.startAdvert();
```

```
 taskManager.begin(sensorTask,1);
 taskManager.begin(heartBeatTask,2);
}

void loop() {

}
```

water_sensor

```
#ifndef __CC3200R1M1RGC__
// Do not include SPI for CC3200 LaunchPad
#include <SPI.h>
#endif

#include <BLE.h>

#include <stdio.h>

#include "HX711.h"

// Core library
#include "Energia.h"
#include "Task.h"

#define DOUT 8
#define CLK 9

char valueR = 0;

Task taskManager;

/* Declare Sensor BLE Service Characteristics */
BLE_Char PesoChar =
{
 {0xE1, 0xFF},
 BLE_READABLE,
 "Peso"
};
```

```
BLE_Char sensorNotificationChar =
{
 {0xE1, 0xFF},
 BLE_NOTIFIABLE,
 "Peso notificacion"
};

/* BLE LED Service is made up of BLE Chars */
BLE_Char *sensorServiceChars[] = {&PesoChar,&sensorNotificationChar};

/* LED Service Declaration */
BLE_Service sensorService =
{
 {0xF0, 0xFF},
 2, sensorServiceChars
};

HX711 scale;
float peso;
float calibration_factor = 82940; //for 5 kg weighing scale, taken after calibration procedure

void sensorTask(){
 while(true){
  Serial.print("Reading: ");
  peso=scale.get_units();
  Serial.print(peso);
  Serial.println(" Kg");

  ble.writeValue(&PesoChar, peso);
  ble.writeValue(&sensorNotificationChar,1);

  delay(10000);
 }
```

```
}

void heartBeatTask(){
 while(true){
  valueR = 1 - valueR;
  digitalWrite(BLUE_LED, valueR);
  delay(1000);
 }
}

void setup() {
 Serial.begin(115200);

 ble.begin();

 /* Add and intialize LED service */
 ble.addService(&sensorService);
 ble.writeValue(&PesoChar, 0);
 ble.writeValue(&sensorNotificationChar,0);

 // initialize the digital pin as an output.
 pinMode(BLUE_LED, OUTPUT);

 scale = HX711();
 scale.begin(DOUT,CLK);
 scale.set_scale();
 scale.tare(); //Reset the scale to 0
 scale.set_scale(calibration_factor); //Adjust to this calibration factor

 /* Start Advertising */
 ble.setAdvertName("SensorAgua");
 ble.startAdvert();
```

```
 taskManager.begin(sensorTask,1);
 taskManager.begin(heartBeatTask,2);
}

void loop() {
}
```

cat sensor

```
#include <M5StickC.h>
#include <M5Display.h>

#include <BLEDevice.h>
#include <BLEUtils.h>
#include <BLEServer.h>



#if CONFIG_FREERTOS_UNICORE
#define ARDUINO_RUNNING_CORE 0
#else
#define ARDUINO_RUNNING_CORE 1
#endif

#define RED_LED 10

#define SERVICE_UUID "bbde8b56-8f58-4f63-97fe-1c5dbbdee9ce"

#define MPU6886_ACCX_UUID "964570ef-8b93-4053-8908-43ca205b4360"
#define MPU6886_ACCY_UUID "964570ef-8b93-4053-8908-43ca205b4361"
#define MPU6886_ACCZ_UUID "964570ef-8b93-4053-8908-43ca205b4362"

#define MPU6886_GYROX_UUID "964570ef-8b93-4053-8908-43ca205b4370"
#define MPU6886_GYROY_UUID "964570ef-8b93-4053-8908-43ca205b4371"
#define MPU6886_GYROZ_UUID "964570ef-8b93-4053-8908-43ca205b4372"

#define MPU6886_TEMP_UUID "964570ef-8b93-4053-8908-43ca205b4382"

#define MPU6886_NOTI_UUID "964570ef-8b93-4053-8908-43ca205b4392"
```

```
float accX = 0;
float accY = 0;
float accZ = 0;

float gyroX = 0;
float gyroY = 0;
float gyroZ = 0;

float temp = 0;

BLEServer *pServer = NULL;
BLEService *pService = NULL;

BLECharacteristic * MPU6886_accX = NULL;
BLECharacteristic * MPU6886_accY = NULL;
BLECharacteristic * MPU6886_accZ = NULL;

BLECharacteristic * MPU6886_gyroX = NULL;
BLECharacteristic * MPU6886_gyroY = NULL;
BLECharacteristic * MPU6886_gyroZ = NULL;

BLECharacteristic * MPU6886_Temp = NULL;
BLECharacteristic * MPU6886_Noti = NULL;

char valueR = 0;

// define two tasks for Blink & AnalogRead
void TaskMPU6886_GetData( void *pvParameters );
void TaskBLE_HeartBeat( void *pvParameters );

void InitBLEServer()
{
 BLEDevice::init("Cat-Sensor");
```

```
pServer = BLEDevice::createServer();
pService = pServer->createService(SERVICE_UUID);

//Accelerometer characteristics
MPU6886_accX = pService->createCharacteristic(
        MPU6886_ACCX_UUID,
        BLECharacteristic::PROPERTY_READ
       );
MPU6886_accY = pService->createCharacteristic(
        MPU6886_ACCY_UUID,
        BLECharacteristic::PROPERTY_READ
       );
MPU6886_accZ = pService->createCharacteristic(
        MPU6886_ACCZ_UUID,
        BLECharacteristic::PROPERTY_READ
       );

//Gyroscope characteristics
MPU6886_gyroX = pService->createCharacteristic(
        MPU6886_GYROX_UUID,
        BLECharacteristic::PROPERTY_READ
       );
MPU6886_gyroY = pService->createCharacteristic(
        MPU6886_GYROY_UUID,
        BLECharacteristic::PROPERTY_READ
       );
MPU6886_gyroZ = pService->createCharacteristic(
        MPU6886_GYROZ_UUID,
        BLECharacteristic::PROPERTY_READ
       );

//Temperature characteristic
```

```
 MPU6886_Temp = pService->createCharacteristic(
                MPU6886_TEMP_UUID,
                BLECharacteristic::PROPERTY_READ
               );

 MPU6886_Noti = pService->createCharacteristic(
                MPU6886_NOTI_UUID,
                BLECharacteristic::PROPERTY_NOTIFY
               );

 pService->start();
 pServer->getAdvertising()->start();
}

// the setup function runs once when you press reset or power the board
void setup() {
 //Start M5Stick
 M5.begin();

 //Init gyroscope/accelorometer
 M5.MPU6886.Init();

 pinMode(RED_LED, OUTPUT);

 InitBLEServer();
 // Now set up two tasks to run independently.
 xTaskCreatePinnedToCore(
  TaskMPU6886_GetData
  , "TaskMPU6886_GetData" // A name just for humans
  , 2048 // This stack size can be checked & adjusted by reading the Stack Highwater
  , NULL
  , 2 // Priority, with 3 (configMAX_PRIORITIES - 1) being the highest, and 0 being the lowest.
  , NULL
```

```
  , ARDUINO_RUNNING_CORE);


  xTaskCreatePinnedToCore(
   TaskBLE_HeartBeat
   , "TaskBLE_UpdateData"
   , 2048 // Stack size
   , NULL
   , 1 // Priority
   , NULL
   , ARDUINO_RUNNING_CORE);


 // Now the task scheduler, which takes over control of scheduling individual tasks, is automatically
started.
}


void loop()
{
 // Empty. Things are done in Tasks.
}


/*--------------------------------------------------*/
/*---------------------- Tasks ---------------------*/
/*--------------------------------------------------*/


void TaskMPU6886_GetData(void *pvParameters)
{
 (void) pvParameters;


 for (;;) // A Task shall never return or exit.
 {
  M5.MPU6886.getGyroData(&gyroX, &gyroY, &gyroZ);
  M5.MPU6886.getAccelData(&accX, &accY, &accZ);
  M5.MPU6886.getTempData(&temp);
```

```
  MPU6886_gyroX->setValue(gyroX);
  MPU6886_gyroY->setValue(gyroY);
  MPU6886_gyroZ->setValue(gyroZ);

  MPU6886_accX->setValue(accX);
  MPU6886_accY->setValue(accY);
  MPU6886_accZ->setValue(accZ);

  MPU6886_Temp->setValue(temp);

  MPU6886_Noti->notify();

  vTaskDelay(10000);
 }
}

void TaskBLE_HeartBeat(void *pvParameters) // This is a task.
{
 (void) pvParameters;

 for (;;)
 {
  valueR = 1 - valueR;
  digitalWrite(RED_LED, valueR);
  delay(60000);
 }
}
```

sensor.py

```
from bluepy import btle

import struct


class SensorNotificationHandle(btle.DefaultDelegate):

  __ble_service = None

  __sensed_values = None


  def __init__(self, ble_service,sensed_values):

    btle.DefaultDelegate.__init__(self)

    self.__ble_service = ble_service

    self.__sensed_values = sensed_values


  def handleNotification(self, cHandle, data):

    sensed_values = dict()

    try:

      sensor_characteristics = self.__ble_service.getCharacteristics()

      for char in sensor_characteristics:

        sensed_values[self.characteristics_uuids[str(char.uuid)]] = struct.unpack('f', char.read())[0]


    except Exception as e:

      print(str(e))



class Sensor:


  name = None

  address = None

  service_uuid = None

  characteristics_uuids = None
```

```
  __ble_sensor = None
  __sensed_values = None

  def __init__(self, name, address, service_uuid, characteristics_uuids):
    self.name = name
    self.address = address
    self.service_uuid = service_uuid
    self.characteristics_uuids = dict()
    for char_dict in characteristics_uuids:
      for char_uuid, char_value in char_dict.items():
        self.characteristics_uuids[char_uuid] = char_value
    self.__sensed_values = dict()

  def try_connect_sensor(self):
    try:
      self.__ble_sensor = btle.Peripheral()
      self.__ble_sensor.connect(self.address)
      self.__ble_sensor.setDelegate(
SensorNotificationHandle(self.__ble_sensor.getServiceByUUID(self.service_uuid),self.__sensed_values) )
      return True

    except Exception as e:
      print(str(e))
      return False

  def sensing(self):
    if self.__ble_sensor.waitForNotifications(20.0):
      return self.__sensed_values
    else:
      return None
```

db_manager.py

```
from influxdb import InfluxDBClient

import datetime

import threading


class DbManager:


  db_address = None

  db_port = None

  db_name = None

  db_username = None

  db_password = None


  __influxdb_client = None

  __lock = None


  def __init__(self, db_address, db_port, db_name, db_username, db_password):

    self.db_address = db_address

    self.db_port = db_port

    self.db_name = db_name

    self.db_username = db_username

    self.db_password = db_password


    self.__lock = threading.Lock()


  def connect_database(self):

    try:

      self.__influxdb_client = InfluxDBClient(host=self.db_address, port=self.db_port,
username=self.db_username,

                        password=self.db_password, database=self.db_name)

    except Exception as e:
```

```
    print(str(e))


def write_point(self, meas_name, tags, fields):
  # point = "test_point,tag1='valor1',tag2='valor2' valor=10"
  json_body = {"measurement": meas_name, "tags": {}, "time": str(datetime.datetime.now()), "fields": {}}
  for tag_name, tag_value in tags.items():
    json_body["tags"][tag_name] = tag_value


  for field_name, field_value in fields.items():
    json_body["fields"][field_name] = field_value


  with self.__lock:
    self.__influxdb_client.write_points([json_body], protocol="json")


def __del__(self):
  self.__influxdb_client.close()
```

config_manager.py

```
import json

from sensor import Sensor

from db_manager import DbManager



class ConfigManager:


  config_path = None


  __json_data = None

  __json_loaded = None


  def __init__(self, config_path):

    self.config_path = config_path

    self.__json_loaded = False


  def load(self):

    try:

      with open(self.config_path, 'r') as json_file:

        self.__json_data = json.load(json_file)

        self.__json_loaded = True

    except Exception as e:

      print(str(e))


  def get_sensors(self):

    sensors_array = []

    if not self.__json_loaded:

      raise Exception("Error, json not loaded")

    for sensor in self.__json_data["sensors"]:

      sensors_array.append(Sensor(sensor["name"], sensor["address"], sensor["service_uuid"],
```

```
                sensor["characteristics_uuid"]))

    return sensors_array


  def get_db_connection(self):

    if not self.__json_loaded:

      raise Exception("Error, json not loaded")

    db_data = self.__json_data["db_connection"]

    return DbManager(db_data["db_address"], db_data["db_port"], db_data["db_name"],
db_data["db_username"],

            db_data["db_password"])
```

server_manager.py

```
from config_manager import ConfigManager

import time

import threading



class ServerManager:


  sensors = None

  db_conn = None

  config_manager = None

  __threads_reference = None


  def __init__(self):

    self.__threads_reference = []


  def load_config(self, path):

    self.config_manager = ConfigManager(path)


    try:

      self.config_manager.load()

      self.sensors = self.config_manager.get_sensors()

      self.db_conn = self.config_manager.get_db_connection()

    except Exception as e:

      print(str(e))


  def start_db_conn(self):

    try:

      self.db_conn.connect_database()

    except Exception as e:
```

```
    print(str(e))


def sensor_task(self, sensor):
  while True:
    try:
      connected = sensor.try_connect_sensor()
      if not connected:
        print("Not connected")
        time.sleep(10)
      else:
        print("Connected")
      while connected:
        meas = sensor.sensing()
        if meas is None:
          break
        self.db_conn.write_point("Cat-Tracker", {"Sensor": sensor.name}, meas)
        time.sleep(10)
    except Exception as e:
      print(str(e))


def start_server_ble(self):
  for sensor in self.sensors:
    thread_ref = threading.Thread(target=self.sensor_task, args=(sensor,))
    thread_ref.start()
    self.__threads_reference.append(thread_ref)
```

main.py

```
from server_manager import ServerManager

import time


if __name__ == '__main__':


  server = ServerManager()

  server.load_config("/home/jvr/Cat-tracker/Pi-Server/server_conf.json")

  server.start_db_conn()

  server.start_server_ble()
```

server_conf.json


```
 {
"sensors":[
  {
    "address":"50:02:91:8d:37:52",
    "name":"gato",
    "service_uuid":"bbde8b56-8f58-4f63-97fe-1c5dbbdee9ce",
    "characteristics_uuid":[
      {"964570ef-8b93-4053-8908-43ca205b4360":"Accelerometer_X"},
      {"964570ef-8b93-4053-8908-43ca205b4361":"Accelerometer_Y"},
      {"964570ef-8b93-4053-8908-43ca205b4362":"Accelerometer_Z"},
      {"964570ef-8b93-4053-8908-43ca205b4370":"Giroscope_X"},
      {"964570ef-8b93-4053-8908-43ca205b4371":"Giroscope_Y"},
      {"964570ef-8b93-4053-8908-43ca205b4372":"Giroscope_Z"},
      {"964570ef-8b93-4053-8908-43ca205b4380":"Temperature"}
    ]
  },
  {
    "address":"a0:e6:f8:c5:3f:01",
    "name":"comida",
    "service_uuid":"0000fff0-0000-1000-8000-00805f9b34fb",
    "characteristics_uuid":[
      {"0000fff1-0000-1000-8000-00805f9b34fb":"Distance"}
    ]
  },
  {
    "address":"78:07:27:E9:6A:86",
    "name":"agua",
    "service_uuid":"0000eff0-0000-1000-8000-00805f9b34fb",
    "characteristics_uuid":[
      {"0000fff1-0000-1000-8000-00805f9b34fb":"Weight"}
    ]
```

```
    }
  ],
  "db_connection":{
    "db_address":"127.0.0.1",
    "db_port":8086,
    "db_name":"CatTracker",
    "db_username":"root",
    "db_password":"root"
  }
}
```

influxdb.sh

```
#!/bin/sh


if [ "$1" = "new" ]; then
  `docker run -p 8086:8086 -v influxdb:/var/lib/influxdb --name=influxdb_tfg influxdb >/dev/null &`
elif [ "$1" = "start" ]; then
  `docker restart influxdb_tfg >/dev/null &`
elif [ "$1" = "stop" ]; then
  `docker stop influxdb_tfg`
else
  echo "Unknown  command"
fi
```

grafana.sh

```
#!/bin/sh


if [ "$1" = "new" ]; then
  `docker run -d --name=grafana -p 3000:3000 --name=grafana_tfg grafana/grafana >/dev/null &`
elif [ "$1" = "start" ]; then
  `docker restart grafana_tfg >/dev/null &`
elif [ "$1" = "stop" ]; then
  `docker stop grafana_tfg`
else
  echo "Unknown  command"
fi
```