



UNIVERSIDAD DE MÁLAGA



Graduado en Ingeniería Informática

Desarrollo de un panel de control online para gemelos digitales de vehículos eléctricos

Development of a web-based dashboard for digital twins of electric vehicles

Realizado por
Roberto Navarro García

Tutorizado por
Javier Troya Castilla
Javier Cámaras Moreno

Departamento
Lenguajes y Ciencias de la Computación
UNIVERSIDAD DE MÁLAGA

MÁLAGA, Junio 2024



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADUADO EN INGENIERÍA INFORMÁTICA

**Desarrollo de un panel de control online para gemelos
digitales de vehículos eléctricos**

**Development of a web-based dashboard for digital twins
of electric vehicles**

Realizado por
Roberto Navarro García

Tutorizado por
Javier Troya Castilla

Cotutorizado por
Javier Cámera Moreno

Departamento
Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, JUNIO DE 2024

Fecha defensa: julio de 2024

Abstract

In the context of the creation of smart and sustainable cities, the use of non-polluting means of transport is of fundamental importance. This goal is fostering the purchase of electric vehicles, which brings with it the need to provide the cities with the necessary infrastructures for charging and maintenance. The most important type of infrastructure for the use of electric vehicles is charging stations. These charging stations are sufficiently complex to make the creation of digital twins of this infrastructure worthwhile. A digital twin has a wide range of functionality, encompassing, among other things, monitoring, simulation and prediction of behaviour.

This final degree project focuses on the development of a real-time monitoring platform for an electric vehicle charging station, prioritising the use of open source technologies and facilitating the incorporation of more stations to the system.

Keywords: Monitoring, Electric Vehicle, Digital Twin, Grafana, InfluxDB

Resumen

En el ámbito de la creación de las ciudades inteligentes y sostenibles, el uso de medios de transporte no contaminantes es fundamental. Este objetivo está en producir un auge en la compra de vehículos eléctricos, y esto trae consigo la necesidad de dotar a las ciudades con las infraestructuras necesarias para su carga y mantenimiento. El tipo de infraestructura más importante para el uso de vehículos eléctricos son las estaciones de carga. Estas estaciones de carga son lo suficientemente complejas como para que merezca la pena la creación de gemelos digitales de estas infraestructuras. Un gemelo digital cuenta con una funcionalidad muy amplia que abarca desde la monitorización, a la simulación y predicción de comportamientos.

Este trabajo de fin de grado se centra en el desarrollo de una plataforma de monitorización en tiempo real de una estación de carga de vehículos eléctricos, tratando de priorizar el uso de tecnologías de código abierto y facilitando la incorporación de más estaciones al sistema.

Palabras clave: Monitorización, Vehículo eléctrico, Gemelo Digital, Grafana, InfluxDB

Índice

1. Introducción	13
1.1. Motivación	13
1.2. Objetivos	14
1.3. Estructura del documento	15
2. Investigación Previa	17
2.1. Tecnologías estudiadas	17
2.1.1. Sistemas de bases de datos	17
2.1.2. Representación de las métricas	18
2.1.3. Lenguaje	19
2.2. Estado del Arte	19
2.2.1. OpenTwins	20
2.2.2. Snap4City	20
2.2.3. Renault	21
3. Metodología, Requisitos y Tecnologías Usadas	23
3.1. Metodología	23
3.2. Requisitos	25
3.2.1. Historias completas	26
3.2.2. Historias del Gestor	26
3.2.3. Historias del usuario general	26
3.3. Tecnologías usadas	28
3.3.1. Grafana	28
3.3.2. InfluxDB	29
3.3.3. Flux	31
3.3.4. Python	32
3.3.5. Docker	32

4. Arquitectura y Base de Datos	35
4.1. Arquitectura	35
4.2. Alojamiento	37
4.2.1. Servidor de terceros	37
4.2.2. Raspberry PI	38
4.2.3. Servidor Local	38
4.3. Simulación de los Datos	38
4.3.1. Función principal	39
4.3.2. Inicializar Vehículo	40
4.3.3. Generar datos para InfluxDB	41
4.3.4. Comprobar estado del vehículo	41
4.3.5. Calcular nuevo nivel de batería y factura	42
5. Panel de Control	45
5.1. Conexión con InfluxDB	45
5.2. Creación de la variable de Grafana para el filtrado	45
5.3. Vista de las matrículas	48
5.4. Vista del porcentaje de batería	50
5.5. Visualización del precio de la carga	51
5.6. Consumo energético por vehículo	53
5.7. Consumo total en las últimas 24 horas	54
5.8. Velocidad de carga de la estación	57
5.9. Precio de la electricidad	59
5.10. Cola de espera	60
5.11. Tiempo de espera medio hoy	62
5.12. Tiempo de espera medio últimos 5 días	65
5.13. Resultado final	69
6. Contenedorización de la aplicación	73
6.1. Estructura y contenido de los contenedores	73

7. Conclusiones y Líneas Futuras	79
7.1. Conclusiones	79
7.2. Líneas Futuras	80
7.2.1. Creación visualización modelo físico	81
7.2.2. Alojamiento servidor remoto	81
7.2.3. Conexión con fuentes de datos reales	81
7.2.4. Extensión a manejo de múltiples estaciones	82
7.2.5. Añadir varios tipos de cargadores	82
Apéndice A. Manual de Usuario	87
A.1. Instalación	87
A.1.1. Requisitos	87
A.1.2. Instalación y arranque del sistema	87
A.1.3. Alternativas	88
A.2. Manejo del software	88
A.2.1. Entrar al sistema	88
A.2.2. Visualizar un panel de control	89
A.2.3. Visualizar métricas	90
A.2.4. Filtrar los datos a mostrar	91
A.2.5. Cambiar temporización actualización	91
A.2.6. Visualizar todos los usuarios del sistema	91
A.2.7. Añadir usuarios	92
A.2.8. Editar usuarios	93
A.2.9. Añadir nuevas fuentes de datos	94
A.2.10. Modificar fuentes de datos	95
Apéndice B. Código Base de Datos	97

Índice de figuras

1.	Captura de la demo del gemelo digital de Florencia	21
2.	Historias de usuario completas	27
3.	Arquitectura actual del sistema	36
4.	Estructura de la base de datos	36
5.	Alternativas para la arquitectura	37
6.	Diagrama de flujo general de la simulación de los datos de entrada	39
7.	Diagrama de la inicialización de un vehículo	40
8.	Diagrama de la generación de datos para InfluxDB	41
9.	Diagrama de la comprobación del estado del vehículo	42
10.	Diagrama del cálculo del nivel de batería y factura	43
11.	Configuración de InfluxDB en Grafana	46
12.	Filtro de cargadores disponibles	46
13.	Configuración de la variable de filtrado en Grafana	47
14.	Vista de las matrículas de los vehículos cargando	48
15.	Configuración visualización matrículas	49
16.	Transformación consulta visualización matrículas	50
17.	Vista del nivel de batería actual de los vehículos cargando	50
18.	Configuración de la vista de los niveles de batería	52
19.	Visualización precio de la carga	53
20.	Configuración de la vista del coste de la carga energética	53
21.	Visualización consumo energético por vehículo	54
22.	Configuración de la vista del gasto energético de los vehículos actuales por cargador	55
23.	Visualización gasto energético por cargador	55
24.	Configuración de la vista del gasto total por cargador	57
25.	Visualización velocidad de carga	57
26.	Configuración de la vista de la velocidad de carga de la estación	59
27.	Visualización precio de la electricidad por kWh	60

28.	Configuración de la vista del precio de la electricidad por kWh	61
29.	Visualización cola de espera	61
30.	Configuración de la vista de la cola de espera	63
31.	Transformación datos cola de espera	63
32.	Visualización tiempo medio de espera hoy	63
33.	Configuración de la vista del tiempo medio de espera del último día	65
34.	Visualización media tiempo espera últimos 5 días	66
35.	Configuración de la vista del tiempo de espera medio de los últimos 5 días . . .	68
36.	Transformación de los datos para la media de espera de varios días	69
37.	Resultado final	70
38.	Resultado final usando filtro	71
39.	Pantalla de inicio de sesión	89
40.	Lista de tableros disponibles	89
41.	Acceso a la lista de tableros desde el menú desplegable	90
42.	Selección vista focalizada en métrica	90
43.	Selector desplegable para filtrar los cargadores	91
44.	Temporizador de actualización de las vistas	92
45.	Acceso a la lista de usuarios desde el menú desplegable	92
46.	Lista de usuarios del sistema	93
47.	Creación de un nuevo usuario	93
48.	Modificar un usuario	94
49.	Lista de plugins para fuentes de datos	94
50.	Acceso a la lista de conexiones de fuentes de datos	95
51.	Vista previa del plugin para PostgreSQL	95
52.	Listado de conexiones configuradas en Grafana	96

Listings

1.	Ejemplos de medidas en formato ‘Line Protocol’	30
2.	Ejemplo de consulta en Flux	31
3.	Consulta de variable para filtrado	47
4.	Código para filtrar la variable en Flux	48
5.	Consulta de ID de vehículos cargando	48
6.	Consulta de porcentaje de batería de vehículos cargando	51
7.	Consulta de precio a pagar por la carga de los vehículos actuales	52
8.	Consulta del consumo energético por vehículo actual en cada cargador	54
9.	Consulta del consumo energético en las últimas 24 horas por cargador	55
10.	Consulta de la velocidad de carga de la estación	58
11.	Consulta del precio del kWh	60
12.	Consulta cola de espera	61
13.	Consulta y cálculo del tiempo medio de espera	64
14.	Consulta y cálculo tiempo medio de espera últimos 5 días	66
15.	Archivo docker-compose	74
16.	Archivo dashboard.yml	76
17.	Archivo datasource.yml	77
18.	Archivo Dockerfile	78
19.	Código en Python de la simulación	97

1

Introducción

En esta sección se da una primera aproximación al proyecto comentando la motivación del desarrollo del proyecto en este campo de estudio. También se van a comentar los objetivos y el alcance de este proyecto así como a explicar la estructura general del documento indicando qué información se va a encontrar en cada sección del mismo.

1.1. Motivación

La preocupación por la contaminación causada por los vehículos de combustión no ha dejado de crecer desde hace años. Esto ha llevado a buscar alternativas menos contaminantes como son los vehículos híbridos/eléctricos que, desde hace tiempo, han ido creciendo en popularidad.

Hoy en día, los vehículos eléctricos son una realidad más que cotidiana, y con los futuros cambios en las legislaciones respectivas a los vehículos de combustión interna, proliferarán aún más. Esto nos abre las puertas a una serie de problemas como son la autonomía del vehículo, su tiempo de carga y el precio de la electricidad.

Para alimentar estos vehículos eléctricos, son necesarias unas estaciones de carga que el propietario del vehículo pueda disponer en su vivienda. Sin embargo, ni todos los propietarios disponen de estas estaciones ni son siempre suficientes debido a la escasa autonomía del vehículo a la hora de realizar trayectos con más coste energético a causa de las características específicas del trayecto. Por este motivo, el uso de estaciones de carga compartidas se hace necesario para los usuarios de vehículos eléctricos.

La utilización de estas estaciones de carga conlleva un desafío para lograr el uso óptimo de las instalaciones, que pasa por la realización de un estudio de eficiencia energética teniendo en cuenta, entre otras cosas, la monitorización de los tiempos, patrones de uso o el precio de la electricidad [20].

Este tipo de sistemas son lo suficientemente complejos como para no permitir realizar un seguimiento y una optimización puramente manual ni puramente autónoma. Una herramienta muy útil e interesante para realizar este tipo de seguimientos y predicciones es la creación de un gemelo digital del sistema. Un gemelo digital es un modelo virtual de un objeto físico que utiliza datos en tiempo real para simular el comportamiento y supervisar en remoto las operaciones del objeto físico [1].

Un gemelo digital puede ser tan complejo como se requiera hacer, pudiendo crear un sistema capaz de simular y recrear cada detalle del objeto físico, así como de realizar predicciones y pruebas. Todo esto, sin embargo, no sirve de nada si no hay una parte del gemelo capaz de permitir una fácil visualización de todos los datos recibidos de los sensores y generados en las predicciones. Aquí entraría en juego el desarrollo de un panel de control que permita la monitorización del sistema de una forma lo más visual posible y amigable para el usuario.

Este Trabajo de Fin de Grado se centra en el desarrollo de un panel de control accesible desde web que supla la parte de monitorización en tiempo real de un gemelo digital de una estación de carga de vehículos eléctricos.

1.2. Objetivos

Este TFG se centra en desarrollar un panel de control desplegado en web que permita la monitorización y visualización del estado y los datos emitidos en tiempo real por una estación compartida de carga de vehículos eléctricos. El objetivo principal del trabajo es crear una herramienta que facilite la monitorización y el estudio del funcionamiento de una estación de carga de vehículos eléctricos por un técnico o gestor de la misma, sin renunciar a las funciones de automatización. El panel de control mostrará datos de interés sobre la estación de carga y los vehículos, tales como la velocidad de carga en cada momento, el coste en Euros de la carga de cada vehículo que está siendo cargado o el tiempo de espera medio de los vehículos en la estación.

El sistema se desarrollará utilizando la tecnología de código abierto ofrecida por Grafana. La instancia de Grafana irá conectada a una base de datos de series temporales InfluxDB en la que se almacenarán todos los datos del gemelo. El lenguaje utilizado para las consultas será Flux.

1.3. Estructura del documento

El documento se estructura en una serie de capítulos, las cuales tratan distintos temas. Estas secciones se resumen a continuación.

En el primer capítulo se expone una introducción al tema del proyecto que incluye su motivación y sus objetivos. En el segundo capítulo se desarrolla la investigación realizada previamente al comienzo del desarrollo de este proyecto. En esta se exponen las distintas tecnologías estudiadas como candidatas a ser utilizadas en el proyecto y que se dividen en sistemas de bases de datos, tecnologías para representar las métricas y lenguajes de programación. También se introducen algunos sistemas ya existentes relacionados con el tema de este proyecto.

En el tercer capítulo, se comienza el desarrollo del proyecto comentando la metodología utilizada, los requisitos especificados para el sistema y las tecnologías finalmente utilizadas. A continuación, en la sección cuatro se explica la arquitectura del sistema actual y posibles alternativas además de todo lo respectivo a la base de datos utilizada, incluido su alojamiento y la carga con datos simulados. La quinta sección está dedicada a todo el desarrollo y configuración del entorno de Grafana dando como resultado el panel de control final. Para concluir, en la sección seis se realiza un resumen de las conclusiones finales del proyecto y las posibles líneas futuras de trabajo.

Adicionalmente, se han añadido dos apéndices, el primero con el manual de instalación y usuario para facilitar la adaptación al sistema y un segundo apéndice que contiene el código empleado para crear el algoritmo de simulación del sistema físico para alimentar la base de datos.

2

Investigación Previa

En este apartado se habla de las tecnologías estudiadas y el estado del arte. El subapartado de tecnologías estudiadas se refiere a aquellas tecnologías que se han considerado y analizado previo al comienzo del desarrollo del proyecto como candidatas al mismo. En el Capítulo 3 (Metodología, Requisitos y Tecnologías Usadas) se detallan las seleccionadas finalmente. En el subapartado del estado del arte se exponen algunos sistemas o marcos de trabajo existentes orientados al mismo campo que el proyecto desarrollado. Algunos de estos sistemas son de código abierto pero lo más común es que se encuentren bajo licencia de software propietario.

2.1. Tecnologías estudiadas

Dentro de las tecnologías estudiadas, se van a diferenciar entre las tecnologías orientadas a los sistemas de bases de datos, la representación gráfica de las métricas y el lenguaje utilizado para el desarrollo del algoritmo de simulación.

2.1.1. Sistemas de bases de datos

Entre los sistemas de bases de datos estudiados, el primero de ellos y finalmente escogido fue InfluxDB [12], un sistema de base de datos optimizado para series temporales, con mucha comunidad y de código abierto. Este sistema, además de con un gran rendimiento, cuenta con dos lenguajes de consulta propios, uno de ellos (Flux) también orientado al análisis de los datos.

Los otros dos sistemas de bases de datos principalmente estudiados son MongoDB y TimescaleDB. MongoDB es un sistema de bases de datos documental de propósito general diseñada para facilitar el desarrollo y escalado de aplicaciones [27]. Desde las últimas versiones se puede utilizar para almacenar series temporales adaptando la estructura de la base de datos. Gracias a su flexibilidad, es especialmente útil cuando los datos tienen una estructura cambiante en el tiempo. En este caso práctico, no resulta una ventaja.

TimescaleDB es un sistema de bases de datos de código abierto, diseñado a partir de PostgreSQL haciendo SQL escalable para datos de series temporales [23]. Aunque según los benchmarks proporcionados por la misma compañía, TimescaleDB tiene un mejor rendimiento que InfluxDB, esta mejoría de rendimiento solo se alcanza llegado a un nivel de carga que no se espera en este proyecto [24].

OpenTSDB es otro sistema de bases de datos de código abierto [10]. Desarrollado a partir de Apache HBase, está pensado para ser un sistema distribuido y escalable. Su última actualización fue en 2021. El último sistema que se ha tenido en cuenta ha sido Prometheus. Se trata de un sistema de código abierto de recogida y almacenamiento de series temporales [18]. Su principal uso está pensado para monitorización y creación de alertas en servidores. Comparado a InfluxDB resulta una opción muy similar [4] pero se ha terminado decantando la balanza por InfluxDB por su capacidad de clusterización, que puede resultar útil en caso de que el sistema tenga requisitos futuros relacionados con su escalabilidad.

2.1.2. Representación de las métricas

A la hora de realizar la representación de las métricas almacenadas se han tenido en cuenta distintas alternativas. Las primeras fueron distintas bibliotecas tanto de Javascript (Chart JS o C3 JS) como de Python (Matplotlib). Esta posibilidad venía de la idea inicial del proyecto de crear el método de visualización en web desde cero. Esta idea se acabó descartando gracias a que las siguientes alternativas resultaban cubrir los requisitos del sistema de forma más eficiente.

La opción finalmente elegida fue Grafana que es una plataforma de visualización y monitorización de código abierto [17]. Permite visualizar datos en forma de gráficos de diversas fuentes de datos en tiempo real. También cuenta con el servicio Grafana Cloud [11] que ofrece todas las funcionalidades de la versión autogestionada pero gestionada por Grafana Labs y cuenta con una más que generosa funcionalidad gratuita.

Finalmente, al haber elegido InfluxDB como sistema de base de datos, también se han considerado los métodos nativos de InfluxDB para la representación. Estos son Chronograf [5], únicamente para su versión 1 e InfluxDB UI [26] para su versión 2. Estos métodos se han descartado ya que no ofrecen tantas posibilidades como Grafana.

2.1.3. Lenguaje

En cuanto a los lenguajes utilizados, el primero en tenerse en cuenta ha sido Java por el previo conocimiento del mismo. Además se tuvo también en cuenta Python por su simplicidad y Javascript por su relación con la posibilidad de realizar las vistas de usuario con el mismo.

Teniendo en cuenta la documentación que InfluxDB aporta respecto a los lenguajes, los más detallados son Python y Javascript [8] pero también cuenta con soporte para otros lenguajes como Java, PHP, Arduino y Go.

Finalmente se ha elegido Python por su sencillez y la calidad de la documentación aportada.

2.2. Estado del Arte

Desde hace años se viene hablando de los gemelos digitales en ámbitos muy distintos como son cadenas de producción, mantenimiento y mejora de productos ya desarrollados o ciudades inteligentes y cada vez va tomando mayor importancia en las empresas.

Un gemelo digital es una réplica virtual de un producto o un proceso lo suficientemente complejo que recoge y monitoriza datos en tiempo real para realizar distintas funciones. Los gemelos digitales abarcan una funcionalidad muy compleja y amplia, yendo desde la monitorización hasta la simulación y predicción de comportamientos. Esto significa que pueden llegar a avisar de cualquier fallo que ocurra en el gemelo físico así como predecirlo antes de que ocurra. También pueden ser capaces de actuar de forma automática para solucionar ciertos problemas, por ejemplo, recalibrando ajustes de equipos en una cadena de montaje si se detecta una desalineación de los componentes. Por este motivo es necesaria la compleja tarea de unificar tecnologías de aprendizaje automático, IoT (Internet of Things), visualización 3D y monitorización, entre otras.

A continuación se presentan algunos ejemplos de proyectos existentes de gemelos digitales. Normalmente, al tratarse de sistemas ‘a medida’ diseñados para productos concretos, no se tratan de sistemas de código abierto, aunque sí que es posible encontrar algunos ejemplos como OpenTwins y Snap4City.

2.2.1. OpenTwins

OpenTwins es una plataforma que utiliza únicamente tecnologías de código abierto para el desarrollo de gemelos digitales. Desarrollado por investigadores de la Universidad de Málaga. Actualmente se encuentra aún en fase de desarrollo y no se recomienda su uso en producción pero está publicado y se puede consultar su arquitectura actual y hacer pruebas [9].

Al igual que este proyecto, integra Grafana para la visualización del gemelo incluyendo soporte de visualización 3D con un plugin para el uso de Unity Engine e InfluxDB como base de datos de series temporales. Además, actualmente, utiliza como núcleo Eclipse Ditto para la creación del gemelo. También aporta compatibilidad con funciones de aprendizaje automático con el uso de Kafka-ML.

2.2.2. Snap4City

Snap4City es una plataforma de gemelos digitales para ciudades inteligentes certificada por FIWARE (plataforma para el desarrollo y despliegue global de aplicaciones de internet impulsada por la Unión Europea) desarrollada por la Universidad de Florencia. Esta plataforma es de código abierto y proporciona soluciones para la creación de aplicaciones de ciudades inteligentes utilizando tecnologías de IoT, análisis de datos y big data.

Proporciona una vista de la ciudad general llamada Gemelo Digital Global y, dentro de esta, vistas de objetos de estudios más pequeños llamados Gemelo Digital Local. Un ejemplo de gemelo digital local sería una planta química con maquinarias, motores y silos, cada uno con representaciones diseñadas en 3D.

En la web de Snap4City [21] se puede probar una demo del gemelo digital de la ciudad de Florencia en la que se puede interactuar con la ciudad y observar, entre otras cosas, el estado del tráfico, paradas de autobuses, carriles bici y el estado de diversos sensores como los de la calidad del aire o el clima. En la figura 1 se puede ver el panel de control de esta demo.

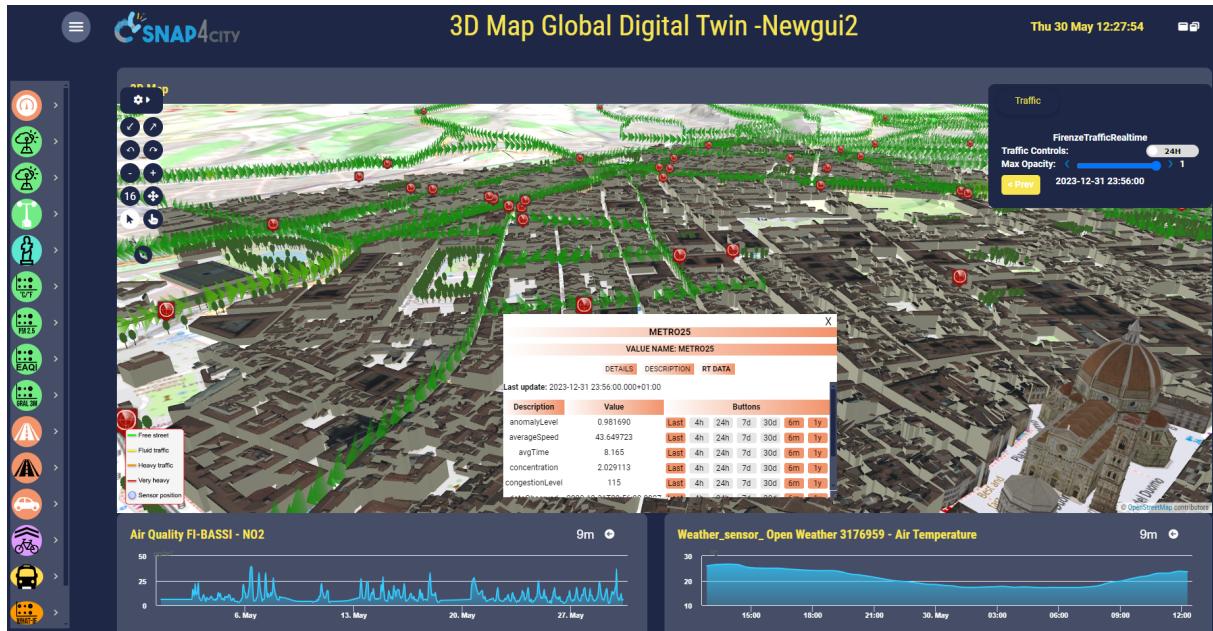


Figura 1: Captura de la demo del gemelo digital de Florencia

2.2.3. Renault

En Renault utilizan los gemelos digitales para su fase de diseño de los vehículos [19]. Crean un gemelo del nuevo vehículo que estén diseñando con todos los detalles para obtener un modelo exacto. El equipo de diseño se encarga de la parte estética del vehículo, creando un primer modelo del interior y exterior del mismo. Cuando esta parte finaliza, el equipo de ingeniería se encarga de crear el modelo técnico diseñando con todo detalle todas las piezas y sistemas del vehículo. Estas piezas se unen al gemelo digital del vehículo e incluso se puede añadir un pasajero virtual.

Para cada pieza se realizan los test necesarios para asegurarse de que todo sea correcto. Con el modelo completo se realizan pruebas de aerodinámicas y eficiencia del motor entre otras. La siguiente fase de test serían los test de seguridad, como son test de choques, escenarios virtuales de conducción en carretera o test de túneles de viento. Una vez que el gemelo digital ha pasado todas las pruebas y, por lo tanto, se considera un modelo consolidado, es el momento de crear el prototipo físico para que pase todas sus pruebas necesarias y se apruebe su producción.

De esta manera, son capaces de sustituir el clásico proceso de diseño y prototipado físico extremadamente caro por un proceso de diseño y prototipado digital, mucho más barato que

permite prototipar solo los modelos que ya han pasado todas las pruebas.

3

Metodología, Requisitos y Tecnologías Usadas

En esta sección se van a comentar tres temas distintos. El primero de ellos es la metodología seguida a lo largo del proyecto, el segundo los requisitos que se han especificado para el sistema y, por último, las tecnologías utilizadas para construir el sistema.

En el apartado de metodología se van a exponer los fundamentos de la metodología de desarrollo ágil, que es la que se ha seguido a lo largo del proyecto. En el apartado de los requisitos, se comentan los requisitos seleccionados en el sistema del proyecto en formato de historias de usuario desde la perspectiva propia de un administrador del sistema y cualquier otro usuario estándar del sistema (incluido el administrador). Por último, hablando de las tecnologías finalmente utilizadas para desarrollar cada parte del proyecto, Grafana se ha utilizado para construir la capa de visualización del usuario en la que podrá estudiar las métricas requeridas, InfluxDB ha sido el sistema de bases de datos seleccionado para almacenar las series temporales recogidas, Flux [7] es el lenguaje de consulta y análisis utilizado en la base de datos y Python el lenguaje utilizado para la simulación de los datos reales del sistema.

3.1. Metodología

La metodología de trabajo utilizada en este proyecto ha sido una metodología ágil. Las metodologías ágiles tienen como objetivo hacer más eficiente el proceso de desarrollo y entrega del producto aplicando una serie de técnicas en fases cortas de trabajo. La base de esta metodología es la colaboración y la entrega y mejora continua.

Toda esta metodología se basa en el Manifiesto Ágil escrito en 2001 [2]. Las bases de este

manifiesto son anteponer a los individuos e interacciones sobre procesos y herramientas, valorar más que el software entregado esté funcionando correctamente que la entrega de otro trabajo extra como una documentación exhaustiva, la colaboración con el cliente sobre la negociación contractual y la respuesta ante el cambio sobre seguir un plan a rajatabla. Este manifiesto implica que, trabajando con la metodología ágil, se debe seguir un método de trabajo que no interfiera en la fluidez del trabajo en equipo, se cuente con la colaboración activa del cliente mediante reuniones periódicas, se tenga una mentalidad pro-activa ante los cambios en los requisitos y centrar el trabajo en obtener un resultado funcional.

El Manifiesto Ágil también cuenta con los siguientes doce principios [3]:

1. **Entrega temprana y continua.** La prioridad es satisfacer al cliente realizando entregas continuas de software con valor.
2. **Cambios en los requisitos.** Se acepta que cambien los requisitos incluso en etapas tardías del desarrollo y se aprovechan los cambios para aportar una ventaja competitiva al cliente.
3. **Periodos de desarrollo cortos.** Se entregan versiones del software funcional de forma frecuente, en periodos de entre dos semanas y dos meses y preferiblemente optando por el periodo más corto.
4. **Trabajo en equipo.** Los responsables del negocio y los desarrolladores deben trabajar juntos durante todo el proyecto.
5. **Desarrollo entorno a individuos motivados.** Hay que dar a los desarrolladores el entorno y apoyo que necesiten, y confiarles la ejecución del trabajo.
6. **Conversación cara a cara.** El mejor método para comunicarse con el equipo es manteniendo una conversación cara a cara.
7. **Medida de progreso.** La medida principal de progreso es tener el software funcionando.
8. **Desarrollo sostenible.** Todos los implicados en el proyecto deben ser capaces de mantener un ritmo de trabajo constante y de forma indefinida.

9. **Excelencia técnica y buen diseño.** Se ha de prestar atención a la excelencia técnica y al buen diseño ya que mejoran la agilidad.
10. **Simplicidad.** Es esencial mantener la simplicidad.
11. **Equipos auto-organizados.** Los equipos auto-organizados crean las mejores arquitecturas, requisitos y diseños.
12. **Reflexiones regulares.** El equipo se debe reunir a intervalos regulares para reflexionar sobre cómo ser más efectivo y ajustar y perfeccionar su comportamiento.

A la hora de aplicar esta metodología, existen diversos marcos de gestión como son Kanban, Scrum, Extreme Programming (XP) o Lean. En el desarrollo de este proyecto se ha utilizado Scrum, un marco basado en ciclos de trabajo (sprints), en los que se genera una nueva versión del producto con un progreso incremental, además de la realización de reuniones periódicas para actualizar el estado del desarrollo y motivar al equipo. Scrum basa su metodología en que los miembros del equipo no tienen un conocimiento completo sobre el tema y evolucionará aprendiendo con la experiencia. Está estructurado para propiciar que los equipos se adapten a los cambios en los planes de forma natural.

Este proyecto se ha organizado en ciclos de trabajo de una o dos semanas dependiendo de la carga de trabajo. Tras cada ciclo, se ha realizado una reunión con los tutores que han actuado de clientes y propietarios del producto. En estas reuniones se ha discutido el progreso realizado hasta el momento y los siguientes pasos a seguir para lograr un avance óptimo y la entrega del producto requerido. Esta metodología ha sido especialmente útil para ir adaptando las prioridades del desarrollo del sistema de forma dinámica.

3.2. Requisitos

La ingeniería de requisitos es el conjunto de actividades y tareas del proceso de desarrollo de software que tienen como objetivo definir, con la mayor calidad posible, las características y el alcance del proyecto para que satisfaga las necesidades de negocio e integración del cliente y sus usuarios así como gestionar las líneas base y peticiones de cambios que se vayan produciendo en la especificación de los requisitos [14].

En la especificación de los requisitos en los marcos de trabajo ágiles, como el que se está siguiendo en este proyecto, se suele hacer uso de las historias de usuario. Una historia de usuario trata de una descripción general y en lenguaje informal de una funcionalidad del sistema. Estas historias se redactan desde la perspectiva de los usuarios finales del sistema.

3.2.1. Historias completas

Las historias para este proyecto se han descrito teniendo en cuenta dos tipos de usuarios, el gestor y el usuario general.

En la figura 2 se observan las historias de usuario desarrolladas a modo de requisitos y en las siguientes secciones se detallan según el tipo de usuario.

3.2.2. Historias del Gestor

Gestión de usuarios. El gestor necesita poder acceder a un listado de todos los usuarios con sus permisos o roles así como editar todos los datos de los usuarios existentes o eliminarlos. También debe de ser capaz de crear nuevos usuarios con los permisos o roles específicos necesarios para poder entregar las credenciales al empleado.

Añadir fuentes de datos. El gestor tiene que poder acceder a las fuentes de datos actuales así como añadir nuevas fuentes de datos independientemente del tipo de sistema en el que estén basadas.

Modificar fuentes de datos. El gestor tiene que poder acceder a las fuentes de datos actuales así como modificar los detalles de su conexión.

3.2.3. Historias del usuario general

Entrar en el sistema. El usuario tiene que poder acceder a la web del sistema desde cualquier navegador. El Gestor necesita que cada usuario tenga unas credenciales propias que le otorguen los permisos necesarios y el usuario general necesita iniciar sesión en el sistema utilizando las credenciales proporcionadas por el gestor.

Visualizar panel de control. El usuario necesita acceder a un listado de paneles de control disponibles y poder seleccionar el deseado para ver toda su información en detalle.

Visualizar todas las métricas. El usuario debe poder visualizar todas las métricas del panel de control en un mismo lugar.

Usuario Gestor	
Actividades	Gestionar usuarios
Usuario	
Tareas	Añadir fuentes de datos
Usuario	Modificar fuentes de datos
Lanzamiento1	
Historias	
Usuario	
Lanzamiento1	
Historias	
Usuario	
Lanzamiento1	
Historias	
Usuario	
Lanzamiento1	
Usuario General	
Actividades	Añadir nuevas fuentes de datos
Usuario	Modificar fuentes de datos existentes
Tareas	
Usuario	
Lanzamiento1	
Entrar en el sistema	
Entrar en la web	
Iniciar sesión	
Entrar en la pestaña de control	
Visualizar un panel de control	
Visualizar todas las métricas	
Visualizar una métrica concreta	
Filtrar los datos a mostrar	
Seleccionar el panel deseado para entrar y ver la información	
Visualizar todas las métricas de forma adecuada en el mismo lugar	
Elegir una visualización de métrica para estudiarla en profundidad	
Seleccionar la información de un solo cargador o varios.	
Como usuario, quiero que cada usuario tenga unas credenciales que le otorgue los permisos necesarios	
Como usuario, quiero introducir mis credenciales de inicio de sesión	
Como usuario, quiero ver todos los paneles de control disponibles	
Como usuario, quiero poder entrar a ver el panel de control en detalle	
Como usuario, quiero ver todas las métricas a la vez en la pantalla del panel de control.	
Como usuario, quiero poder centrar la pantalla en una sola visualización de las métricas	
Como usuario, quiero poder filtrar la información en pantalla según el cargador que quiera seguir.	

Figura 2: Historias de usuario completas

Visualizar métrica concreta. El usuario debe poder seleccionar una de las visualizaciones del panel de control para priorizar su vista y estudiarla más detenidamente. Estas métricas son la batería de cada vehículo que se encuentre cargando, su matrícula, el precio de la factura de cada vehículo cargando, el precio del kWh, la energía consumida por cada vehículo que se encuentre cargando y la consumida en cada cargador en las últimas 24 horas, la velocidad de carga de la estación, la cola de espera de vehículos con su tiempo de espera, el tiempo de espera medio del día, y el tiempo de espera medio de los últimos cinco días.

Filtro de datos. El usuario debe poder filtrar la información visualizada en pantalla según el cargador o los cargadores que quiera estudiar en detenimiento.

3.3. Tecnologías usadas

3.3.1. Grafana

Grafana es un software libre soportado por Grafana Labs, publicado en 2014 y, actualmente, bajo la licencia AGPL-3.0 y desarrollado en Go y typescript. Su función principal es la visualización y monitorización de métricas, logs o trazas sin importar el tipo de fuente de datos en las que estén almacenadas. Permite la conexión con distintas bases de datos a través de los plugins de fuentes de datos. Estos plugins realizan la conexión con las fuentes a través de APIs (Interfaz de Programación de Aplicaciones con sus siglas en inglés) y procesan los datos en tiempo real.

La monitorización se realiza con la creación de paneles de control o tableros dinámicos y con múltiples opciones de gráficos para visualizar los datos de la forma que más se adapte a las necesidades del proyecto. El uso de los plugins de paneles ofrecen opciones casi infinitas de visualización ya que existe la posibilidad de crear paneles completamente personalizados. Cuenta con la posibilidad de crear variables de tablero que se pueden utilizar para filtrar la información mostrada en el tablero. También cuenta con la posibilidad de utilizar varias fuentes de datos en un mismo tablero. Cada visualización corresponde con una o varias consultas a la base de datos.

Los tableros y sus datos pueden ser compartidos a otros usuarios y equipos así como publicarlos de forma global. Igualmente, también proporciona herramientas de administración para mantener la configuración y seguridad de los datos y los permisos de usuario dentro de

cada organización.

Además, es capaz de crear y administrar alertas desde su propia interfaz. Al definir reglas para las métricas deseadas, Grafana las evalúa continuamente y, en caso de ser necesario, envía una notificación al sistema que lo requiera como Slack o PaperDuty. También puede generar automáticamente informes en PDF, según se haya programado, de cualquier tablero y enviarlos por email.

Grafana Labs ofrece el producto de Grafana de dos formas distintas. La primera es autogestionado y gratuito para usuarios y para empresas y la segunda es gestionado por Grafana Labs en su infraestructura de servidores mediante una suscripción al servicio. El servicio gestionado por la empresa cuenta con distintas opciones de suscripción entre las que se encuentran una capa gratuita de por vida y bastante generosa que debería ser suficiente para monitorizar proyectos pequeños y medianos de usuarios.

El stack tecnológico de Grafana cuenta también con otros productos de código abierto para complementar la monitorización y visualización con herramientas propias. Algunas de estas herramientas son Loki para almacenamiento y consulta de logs o Graphite para recopilación y almacenamiento de series temporales.

3.3.2. InfluxDB

InfluxDB es una plataforma de base de datos NO-SQL publicada en 2013 por InfluxData y desarrollado con el lenguaje de programación Rust. Actualmente se encuentra bajo la licencia Apache-2.0, MIT siendo de código abierto hasta su versión 2.x y estando su versión 3.x disponible solamente en su servicio gestionado en la nube.

InfluxDB está diseñado y optimizado concretamente para almacenar y manejar grandes volúmenes de datos en series temporales. Tiene la capacidad de manejar lectura y escritura de datos en tiempo real, lo que la hace perfecta para monitorización y análisis de infraestructuras e IoT. Tiene una estructura escalable y flexible pudiendo añadir mediciones a lo largo del tiempo y características de replicación y clustering.

Todos los datos se almacenan en Cubos (Buckets). Un Bucket se trata, básicamente, de una base de datos (comparable con una tabla de una base de datos del tipo SQL) con un periodo de retención de los datos. Dentro de un Bucket se encuentran las estructuras de datos, las series temporales.

Las series temporales cuentan con varios elementos. El primero a tener en cuenta es la Medida (Measurement). La medida es un valor de tipo cadena de caracteres que sirve para englobar un conjunto de Etiquetas (Tags), Campos (Fields) y las Marcas temporales (Timestamps).

Las etiquetas y los campos son conjuntos clave-valor. Las etiquetas sirven para filtrar la información, son opcionales y se indexan en la base de datos por lo que utilizarlos hace que mejore considerablemente el rendimiento. Los campos almacenan los datos (las métricas) recogidos, por ejemplo, por los sensores de IoT y susceptibles de analizar.

Por último, la marca temporal es el momento en el que se realiza la medición. Cada registro se compone de una medida, un conjunto de etiquetas, un conjunto de campos y una marca temporal. Cada linea de la base de datos se compone de la medida, el conjunto de etiquetas, un campo y la marca temporal.

En el siguiente ejemplo proporcionado por InfluxData en la documentación oficial [13], se muestran dos puntos de datos en formato protocolo línea ('Line Protocol') de dos medidas distintas. La primera medida, *airSensor*, tiene las etiquetas *sensorId* y *station* y los campos *humidity* y *temperature*. La segunda medida, *waterQualitySensor*, tiene las mismas etiquetas y los campos *pH* y *temperature*. Un punto de datos de cada una de estas dos medidas, en formato 'Line Protocol' se representa de la siguiente manera, siendo el último valor de cada conjunto la marca temporal.

Listing 1: Ejemplos de medidas en formato 'Line Protocol'

```
airSensor , sensorId=A0100  humidity =35.06 , temperature =21.6  16367295430
waterQualitySensor , sensorId=W0101  pH=6.1 , temperature =16.1  14725152000
```

Listing 1: Ejemplos de medidas en formato 'Line Protocol'

InfluxDB proporciona dos lenguajes: InfluxQL, un lenguaje de consulta parecido a SQL y Flux, una alternativa a InfluxQL y otros lenguajes de tipo SQL que utiliza patrones de programación funcional que mejora muchas de las limitaciones de InfluxQL. InfluxDB cuenta también con la posibilidad de utilizar integraciones de forma sencilla con otras aplicaciones y servicios a través de su API.

La plataforma se oferta de distintas formas, entre las que se encuentran las opciones autogestionadas (InfluxDB Clustered, Enterprise y OSS) y las gestionadas por InfluxData en sus

propios servidores (InfluxDB Cloud, Cloud Serverless y Dedicated).

El stack tecnológico de InfluxDB (TICK) está formado por Telegraf (recolección de métricas), InfluxDB (almacenamiento de métricas), Chronograf (visualización y monitorización) y Kapacitor (procesamiento de datos en tiempo real).

3.3.3. Flux

FLux es un lenguaje de programación funcional de código abierto diseñado por InfluxData para realizar consultas, analizar y actuar en los datos [7]. Está desarrollado en Go y Rust. Actualmente está en periodo de mantenimiento por parte de InfluxData pero hay una bifurcación del proyecto en la que la comunidad continúa añadiendo funciones. Flux acepta múltiples fuentes de datos que son InfluxDB, Prometheus, archivos .csv y distintas bases de datos SQL.

En resumen, el funcionamiento de Flux se divide en cuatro pasos. Primero recupera los datos de una fuente, después los filtra según unos parámetros de tiempo o valores, procesa los datos y, por último, devuelve el resultado al usuario.

La estructura básica de una consulta escrita en Flux está formada por fuentes, filtros, formateos (shape) y procesos de los datos. En la fase de fuentes, se importan los datos. Todas las fuentes de datos devuelven los datos como un flujo de tablas. En la fase de filtrado se filtran los datos recibidos por tiempo o por valores de las columnas. Las funciones principales de filtrado son *range()* para tiempo y *filter()* para valores. En la fase de formateos se da forma y se ordenan los datos devueltos por los filtros en la fase anterior con funciones como *group()* y, finalmente, en la última fase, se realizan operaciones sobre los datos de tipo agregación, selección de puntos de datos específicos, reescribir filas con *map()* o envío de notificaciones. En el código representado en el listing 2 se encuentra un ejemplo de consulta básica en Flux, en la que realiza operaciones de filtrado, formateo y procesado.

Listing 2: Ejemplo de consulta en Flux

```
from(bucket: "example-bucket") // Fuentes
|> range(start: -1d)          // Filtrado en tiempo
|> filter(fn: (r) => r._field == "foo") // Filtrado en valores de
                                         // columnas
|> group(columns: ["sensorID"]) // Formateo
```

```
| > mean() // Procesado
```

Listing 2: Ejemplo de consulta en Flux

3.3.4. Python

Python es un lenguaje de programación de código abierto publicado con su primera versión en 1991 por Guido van Rossum. Es un lenguaje de alto nivel, interpretado y multiparadigma ya que soporta la programación orientada a objetos, programación imperativa y la programación funcional.

Python está administrado por Python Software Fundation y cuenta con una gran comunidad detrás que lo hace uno de los lenguajes más populares actualmente. Esta comunidad permite que el lenguaje esté en continuo desarrollo y ofrece una gran cantidad de recursos y apoyo tanto para aprender el lenguaje como para solucionar problemas.

Este lenguaje cuenta con una sintaxis que trata de ser muy legible y sencilla de aprender. Se trata de un lenguaje ampliamente utilizado en desarrollo web, aprendizaje automático (Machine Learning o ML), ciencia de datos, tareas de automatización y desarrollo de software en general.

Una de las características que más a simple vista llaman la atención del lenguaje es que no utiliza llaves para delimitar los bloques de código, sino sangría. Además, se trata de un lenguaje tipado dinámicamente, lo que significa que, al declarar las variables, no es necesario especificar el tipo de datos que van a contener (aunque se puede hacer) ya que lo hará Python en tiempo de ejecución.

Python cuenta con una biblioteca estándar muy amplia que permite agilizar el desarrollo reutilizando código.

3.3.5. Docker

Docker es una plataforma de código abierto lanzada en 2013 y diseñada para ayudar a los desarrolladores a construir, compartir y ejecutar aplicaciones contenedorizadas. Docker empaqueta el software en unidades llamadas ‘Contenedores’ que contienen todo el sistema incluyendo el código, sus dependencias y el tiempo de ejecución. Esto proporciona un entorno independiente del sistema operativo en el que se ejecuta la aplicación empaquetada evitando

errores de compatibilidad al ejecutar el sistema en una máquina distinta a la que se ha utilizado en el desarrollo.

El funcionamiento de Docker es similar al de una máquina virtual en el sentido de la abstracción del sistema operativo anfitrión y virtualización del hardware, pero con la diferencia de que añade otra capa por encima del sistema operativo. Esta capa es la responsable de que una aplicación empaquetada en un contenedor Docker se ejecute sin problema en cualquier sistema operativo y en cualquier máquina solamente teniendo instalada la plataforma de Docker.

Un contenedor se define en un archivo llamado *Dockerfile* en el que se incluye la versión del sistema que va a utilizar (imagen) y una serie de comandos que debe realizar para que el servicio funcione. En el caso de necesitar definir varios contenedores, se utilizaría docker-compose. Docker-compose es una herramienta para definir y ejecutar aplicaciones multi-contenedores. Se utiliza definiendo el sistema en un archivo *compose.yml*. En este archivo se definen las imágenes de los subsistemas que se van a ejecutar en los contenedores y todo lo necesario para su aprovisionamiento. Las imágenes utilizadas por los contenedores se encuentran en la web de Docker Hub [6] y están clasificadas en imágenes oficiales publicadas por Docker, editor verificado y esponsorizada de software libre.

4

Arquitectura y Base de Datos

En este capítulo, primero se habla sobre la arquitectura actual del sistema y algunas alternativas que se podrían tener en cuenta en un futuro. Actualmente el sistema, al no poder contar con una base de datos que almacene métricas reales de una estación de carga en funcionamiento, cuenta con un programa escrito en Python que se encarga de realizar la simulación del funcionamiento del sistema real y el envío de las métricas a la base de datos. Esto se ve reflejado en el apartado de la arquitectura y en el apartado de la base de datos.

En los apartados de la base de datos se exponen las distintas opciones estudiadas para el alojamiento de la base de datos, teniendo en cuenta las posibilidades de utilizar un servidor de terceros, una Raspberry PI o un servidor local, y el funcionamiento en detalle del programa de simulación de los datos.

4.1. Arquitectura

El sistema actualmente, al utilizar solo datos de prueba, tiene una arquitectura muy simple como se puede ver en la figura 3. Consta de tres partes, un script escrito en Python encargado de la simulación de los datos y el envío a la base de datos, la base de datos de series temporales InfluxDB y el sistema de visualización para las métricas de Grafana.

La conexión entre el script de Python e InfluxDB se realiza a través de la API proporcionada por Influx y la conexión entre InfluxDB y Grafana se realiza a través del plugin nativo de Grafana para la conexión con InfluxDB y su uso como fuente de datos para las métricas.

La base de datos de InfluxDB tiene una arquitectura formada por una Organización que contiene un cubo ('Bucket'). Este cubo es el que contiene las Medidas o Métricas "stationA" y "stationAQueue" y estas contienen los Campos y las Etiquetas de las mediciones. En la figura



Figura 3: Arquitectura actual del sistema

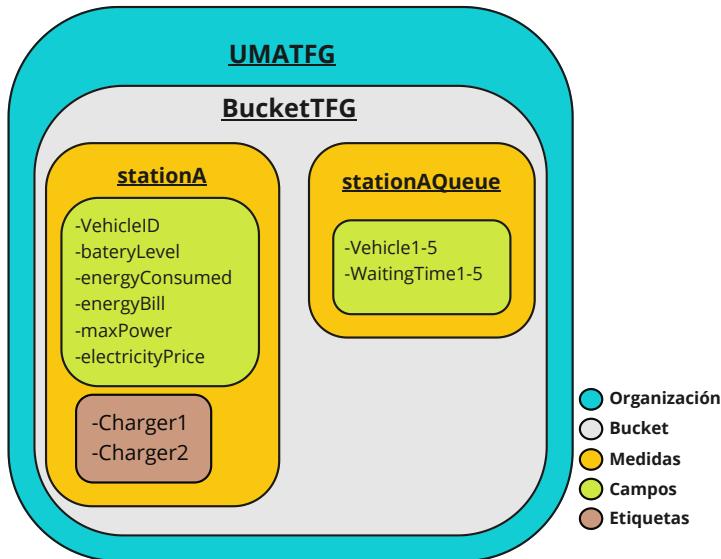


Figura 4: Estructura de la base de datos

4 se muestran gráficamente con los nombres utilizados para que pueda entenderse con más claridad.

En el momento en que se requiera trabajar con datos reales, InfluxDB cuenta con distintas formas de recolección de datos. El modo más similar al actual sería el uso de la API que proporciona Influx para 12 lenguajes entre los que se incluyen, a parte de Python, Javascript, Arduino y Java. Otra herramienta de recolección de datos muy potente y también proporcionada por Influx sería Telegraf, un agente basado en servidor para recolectar y enviar todas las métricas y eventos de bases de datos, sistemas y sensores IoT (Internet of Things) [22].

Adicionalmente, si no se deseara utilizar InfluxDB, también existirían otras alternativas válidas como son TimescaleDB o OpenTSDB que son alternativas de bases de datos de series temporales o incluso MongoDB adaptando su esquema a los requerimientos. La conexión de estas bases de datos alternativas con Grafana, al igual que con InfluxDB, se realizaría por medio de plugins propios de Grafana para la conexión de estas bases de datos.

En la figura 5 se observa gráficamente algunas alternativas a la arquitectura actual en línea discontinua y con líneas continuas la arquitectura desarrollada.

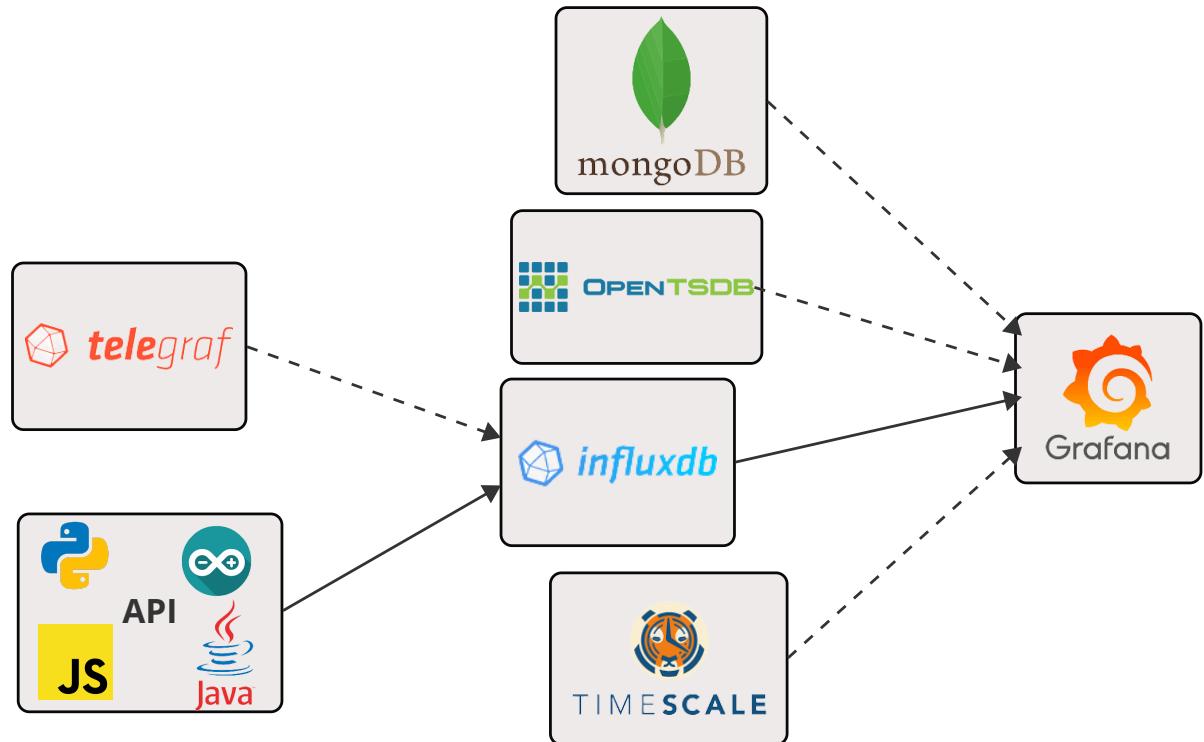


Figura 5: Alternativas para la arquitectura

4.2. Alojamiento

Con el fin de poder hacer las pruebas y demostraciones necesarias en este proyecto, se ha creado una base de datos con información simulada de cómo funcionaría el sistema real. La simulación de los datos se realiza mediante un programa que se detalla más adelante.

Esta base de datos necesita un servidor donde estar alojada. Para ello se han planteado tres opciones, de las cuales se ha terminado por escoger la última de ellas.

4.2.1. Servidor de terceros

El uso de un servidor de terceros otorga las ventajas de poder acceder a él desde cualquier lugar de trabajo y la liberación del equipo local de la carga de recursos que pueda suponer la ejecución y almacenamiento de la base de datos. Este último punto también lleva a tener en cuenta que un fallo del equipo local resultaría un obstáculo menor. Sin embargo, trae también

algunos inconvenientes. Muchos servidores de terceros cuentan con una capa de uso gratuita que funciona hasta cierta cantidad de recursos. Utilizar esta capa sería lo ideal, sin embargo, siempre se corre el riesgo de alcanzar ese límite y comenzar a realizar pagos de forma imprevista.

Otro punto a tener en cuenta es que, al ser un proyecto desarrollado por una única persona y por el tamaño del mismo, el tener que estar manejando un servidor externo resultaría más en una pérdida de agilidad que en una ventaja real.

4.2.2. Raspberry PI

Las ventajas del uso de una Raspberry PI son la posibilidad de liberar los recursos del equipo local sin tener que lidiar con empresas externas. Las desventajas son la necesidad de realizar toda la configuración necesaria de forma manual para poder acceder a la base de datos desde otros dispositivos y la imposibilidad de acceder desde cualquier localización.

Un punto decisivo para este método es que, al estudiar los requisitos de InfluxDB para instalarse en una Raspberry PI, se observa que es necesaria el uso de un modelo PI 4+ o 400 [15] y el modelo de Raspberry disponible para esta tarea sería un PI 3B.

4.2.3. Servidor Local

Este ha sido el método de alojamiento elegido finalmente. Montar la base de datos en el equipo local otorga simplicidad, control total de la misma y agilidad. Esto permite evitar preocupaciones en cuanto al alojamiento como tal y mantener la concentración en el desarrollo real del proyecto.

Los inconvenientes mencionados anteriormente como la ocupación de los recursos y un posible fallo del equipo se pueden mitigar limitando el funcionamiento y la longevidad de los datos de la base de datos y guardando el avance del código de simulación en plataformas como GitHub.

4.3. Simulación de los Datos

Para la simulación y almacenamiento de los datos, se ha desarrollado un programa en Python. Este programa emula el funcionamiento de una estación de carga de vehículos eléctricos.

tricos con dos puestos de carga y una cola de espera de 5 puestos y almacena una serie de métricas en tiempo real. Al tratarse únicamente de datos de prueba, se ha diseñado para que, una vez que entra un vehículo a cargar, solo se desconecta cuando su nivel de carga ha llegado al 100 % .

Se ha elegido el lenguaje de programación Python por la simplicidad de su sintaxis y la buena documentación que proporciona Influx [8] para realizar la conexión y tratamiento de datos con su API de este lenguaje.

Este programa crea y envía a la base de datos, respecto a los cargadores, las mediciones del nombre/ID del cargador, localización/ID de la estación, matrícula/ID del vehículo, energía consumida por vehículo, nivel de batería del vehículo, velocidad de carga, precio de la electricidad y marca temporal de la medición. Respecto a la cola de espera crea y envía las mediciones de las matrículas de los vehículos en la cola y el tiempo que llevan esperando.

El código completo de este programa se puede consultar en el Apéndice B de este documento pero aquí se va a explicar el funcionamiento en detalle del mismo.

4.3.1. Función principal

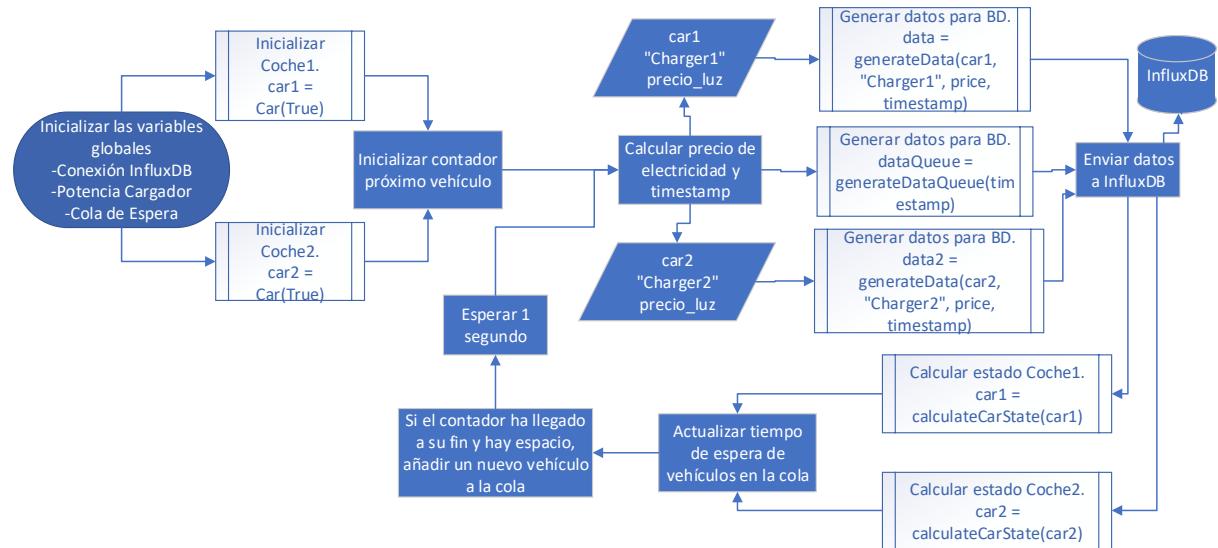


Figura 6: Diagrama de flujo general de la simulación de los datos de entrada

Como se puede ver en la figura 6, el primer paso que realiza el programa es inicializar las variables globales. Estas variables forman tanto la configuración de la conexión de cliente de InfluxDB como la velocidad de carga de la estación y la cola de espera.

A continuación, el programa entra en la función principal *main()*. Aquí, lo primero es inicializar los dos vehículos con parámetro igual a ‘True’. De este modo, se crean dos objetos de tipo *Car* con la información necesaria para comenzar a cargar. Antes de comenzar el bucle principal, se inicializa la variable *nextCar* (entero aleatorio que representa el tiempo que va tardar el siguiente vehículo en llegar a la cola de espera).

Al entrar en el bucle principal, calcula aleatoriamente el precio de la electricidad entre los valores de 0.15-0.75€/kWh, crea la marca temporal en nanosegundos (*timestamp*) que guarda el momento de la medición y genera los datos que se almacenarán en el siguiente paso en la base de datos con la función auxiliar o subproceso *generateData(car, “nombreCargador”, price, timestamp)* y *generateDataQueue(timestamp)*.

Tras enviar los datos a InfluxDB haciendo uso de su API, se recalcula el estado de los vehículos con la función auxiliar *calculateCarState(car, price)* y se actualiza la cola de espera en dos pasos. Primero, si hay vehículos en la cola, se actualiza el tiempo que llevan esperando; y segundo, en caso de que el contador haya llegado a su fin y la cola no esté llena, se añade otro vehículo (*Car(True)*) al final de la cola.

Finalmente, se espera un segundo para realizar la siguiente iteración.

4.3.2. Inicializar Vehículo

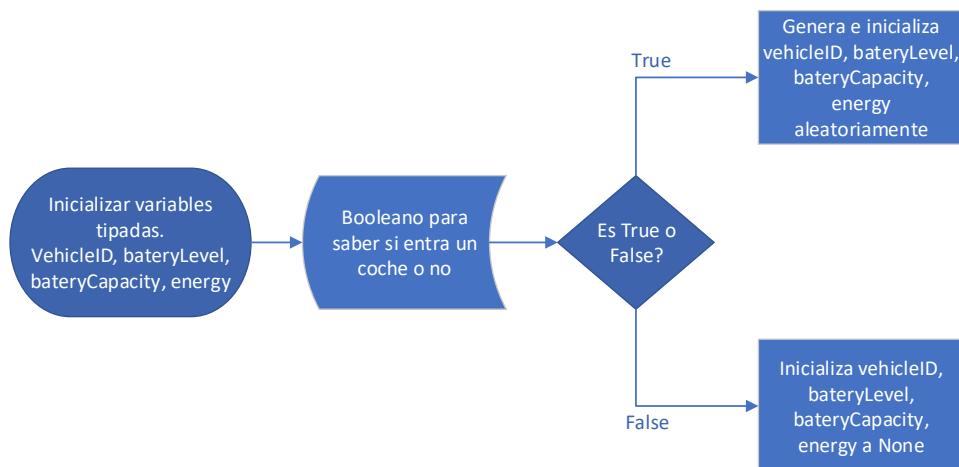


Figura 7: Diagrama de la inicialización de un vehículo

En este subproceso representado en la figura 7 se trabaja con un objeto de la clase *Car*. A este objeto le entra por parámetro un booleano *enter* que determina si se trata de un vehículo

con datos vacíos o no. El booleano sería ‘True’ cuando hay un vehículo cargando en uno de los puntos de carga.

En primer lugar se declaran las variables del vehículo marcando sus tipos de datos. Estas variables son *vehicleID* (matrícula), *batteryLevel*, *batteryCapacity*, *energyConsumed* y *energyBill*. A continuación se comprueba si *enter* es verdadero o falso. En caso de ser falso se inicializan las variables a *None* y en caso de ser verdadero se inicializan con valores aleatorios. La matrícula se genera con el subproceso *generateVehicleID()*, el porcentaje de batería entre los valores 0-99 %, la capacidad de batería entre 40-100 kWh y la energía consumida y factura a 0.

4.3.3. Generar datos para InfluxDB

En la figura 8 se representa la función *generateData()*. Se generan los datos enviados a InfluxDB creando y devolviendo, como estructura de datos, un objeto de clase de datos *Charger* o *dataQueue* que almacena todo lo necesario.

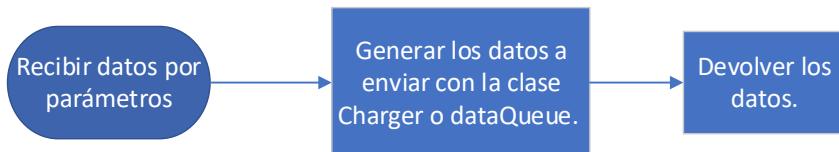


Figura 8: Diagrama de la generación de datos para InfluxDB

4.3.4. Comprobar estado del vehículo

Al llegar a la función *calculateCarState(car, price)*, representada en la figura 9, el primer paso es comprobar la matrícula del vehículo.

En caso de que la matrícula del vehículo sea igual a *None*, se genera un booleano aleatorio que determina si entra a cargar un nuevo vehículo o no (se comprueba si hay vehículos esperando en la cola). Si el booleano resulta ‘True’ (si hay coches esperando en la cola), se crea un nuevo coche (*Car(True)*) (entra a cargar el primer vehículo de la cola). Si el booleano resulta ‘False’ (no hay vehículos esperando en la cola), no se actualiza nada.

En el caso de que la matrícula del vehículo sea distinta a *None*, eso significa que hay un vehículo cargando actualmente. Si el porcentaje de batería de este vehículo es mayor o igual a 100, se elimina el vehículo del cargador (*Car(False)*). Si el porcentaje es menor a 100, comienza

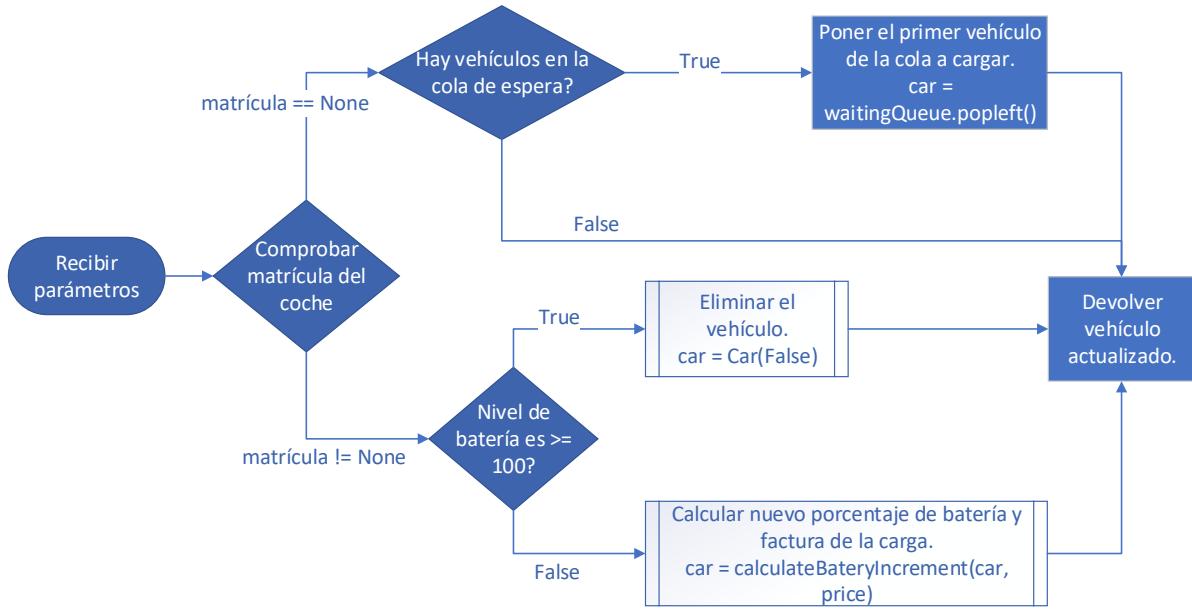


Figura 9: Diagrama de la comprobación del estado del vehículo

el subprocesso para calcular el nuevo porcentaje de batería y factura de la carga.

Al finalizar se devuelve el nuevo estado del vehículo en cuestión, ya sea ‘vacío’ o no.

4.3.5. Calcular nuevo nivel de batería y factura

Como se puede ver en la figura 10 que representa la función *CalculateBatteryIncrement()*, lo primero a realizar es calcular la velocidad de carga por segundo y los kiloWatos almacenados en la batería actualmente. A continuación, se calculan los kW incrementados sumando a los actuales la velocidad de carga por segundo y se suma la velocidad a la energía consumida. También se actualiza el precio a pagar por la electricidad. Si los kW calculados actuales superan a la capacidad de batería del vehículo, se establece el nivel de batería al 100 %. Si no, se calcula y establece el nuevo porcentaje de batería.

Por último, se devuelve el nuevo estado del vehículo.

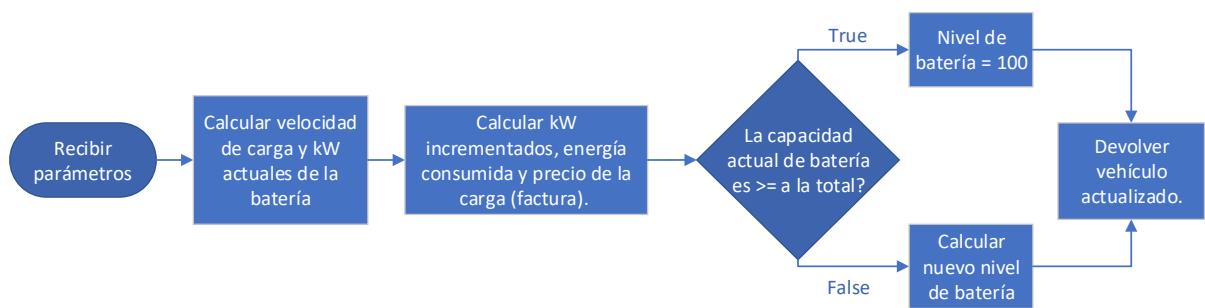


Figura 10: Diagrama del cálculo del nivel de batería y factura

5

Panel de Control

En este capítulo se expone la configuración del panel de control desarrollado en Grafana, mostrando cada una de las vistas concretas que se han creado así como otras configuraciones del sistema como son la conexión con la fuente de datos y la creación de la variable para filtrar la información según el cargador. Las métricas visualizadas se pueden diferenciar entre las relacionadas directamente con cada cargador (filtrables) y las relacionadas con la estación en general.

5.1. Conexión con InfluxDB

El primer paso para crear cualquier tipo de tablero en Grafana es conectar el sistema con la base de datos que contiene la información que se requiera monitorizar. Esto se realiza desde el apartado de ‘Connections’ seleccionando el nuevo tipo de fuente de datos a configurar y rellenando la configuración. En este caso, al tratarse de una base de datos de prueba en un entorno local y como se puede observar en la figura 11, se ha realizado una configuración sencilla con las opciones mínimas. Los datos que pide para la conexión de InfluxDB son el nombre que se le quiera asignar a la conexión, el lenguaje de consulta (dando a elegir entre InfluxQL, Flux y SQL solo para la versión 3 de Influx), opciones de HTTP como la URL al servidor en el que esté alojada, elegir el tipo de autenticación a utilizar y llenar los datos y, por último, ciertos detalles de InfluxDB entre los que están el nombre de la organización, el Token que proporciona los permisos para trabajar con la base de datos y el nombre del Bucket al que se va a consultar por defecto.

5.2. Creación de la variable de Grafana para el filtrado

The screenshot shows the Grafana interface for configuring a connection to InfluxDB. The connection is named 'influxdb' and is set to use the 'Flux' query language. Under the 'Auth' section, 'Basic auth' is selected and turned on. The 'User' field is set to 'Admin' and 'Password' is set to 'configured'. In the 'InfluxDB Details' section, the 'Organization' is listed as 'UMATFG'. Other settings like 'Token', 'Default Bucket', 'Min time Interval', and 'Max series' are also visible.

(a) Configuración 1

This screenshot shows a different configuration for the same 'influxdb' connection. Here, 'Basic auth' is turned off, while 'TLS Client Auth' is turned on with the option 'With CA Cert' selected. The 'User' field is still 'Admin' and 'Password' is 'configured'. The 'InfluxDB Details' section remains the same as in configuration 1.

(b) Configuración 2

Figura 11: Configuración de InfluxDB en Grafana

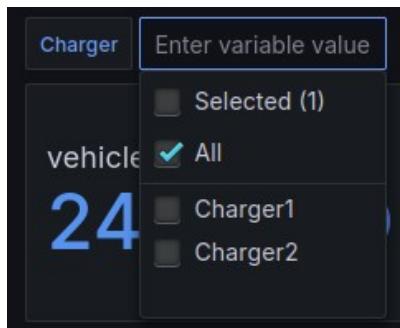


Figura 12: Filtro de cargadores disponibles

En la figura 12 se muestra el resultado del filtro agregado al tablero. Este desplegable permite filtrar parte de la información visualizada en pantalla relativa a los cargadores disponibles. Los cargadores disponibles se recuperan de la base de datos al entrar en el tablero por lo que el listado no está limitado a un número concreto de opciones dando la posibilidad de escoger uno o más cargadores. Teniendo en cuenta la posibilidad de que el gerente de la estación pueda habilitar o deshabilitar cargadores, esta función puede ser importante para no tener opciones de selección que no contengan datos actuales.

Para obtener este filtro, se ha creado una variable desde los ajustes del tablero en Grafana. Estos ajustes mostrados en la figura 13 contienen las opciones de seleccionar el tipo de variable (la cual se selecciona *Query*), el nombre de la variable, cómo se mostrará en el tablero para identificarla y usarla (en este caso el nombre y la etiqueta se han configurado con el mismo nombre pero no es necesario), descripción, la consulta a la base de datos para recopilar la información y otras opciones como la posibilidad de escoger varios resultados y que se incluya

un “todas las opciones”.

(a) Captura 1

(b) Captura 2

Figura 13: Configuración de la variable de filtrado en Grafana

La consulta realizada para la información (y que no aparece en la figura 13) de los cargadores disponibles es la representada en el listing 3.

Listing 3: Consulta de variable para filtrado

```
import "influxdata/influxdb/v1"
v1.tagValues(
    bucket: v.bucket,
    tag: "name",
    predicate: (r) => true,
    start: -10s
)
```

Listing 3: Consulta de variable para filtrado

Esta variable se invoca en las consultas de las visualizaciones concretas que la necesiten con

una línea de filtrado que indique que tiene que contener los valores de la variable en los tags. El formato de lectura es JSON ya que es el formato que utiliza Flux para estos tratamientos. Esta línea de código es la responsable de que al seleccionar los cargadores en el filtro, se actualicen de forma dinámica las visualizaciones. La línea en cuestión es el listing 4.

Listing 4: Código para filtrar la variable en Flux

```
|> filter (fn: (r) => contains( value: r[ "name" ] , set: ${Chargers:json} )) )
```

Listing 4: Código para filtrar la variable en Flux

5.3. Vista de las matrículas

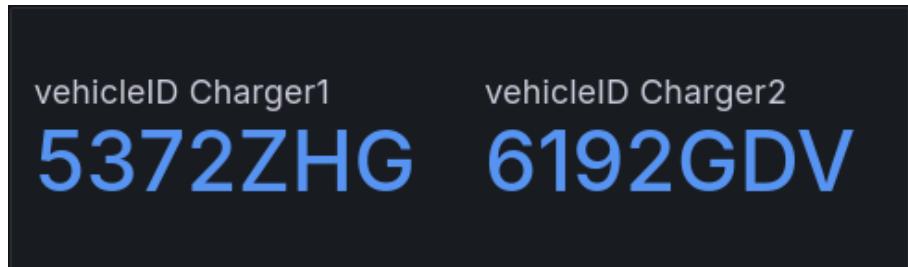


Figura 14: Vista de las matrículas de los vehículos cargando

En todo gemelo digital es importante tener identificados los agentes externos que interactúan con el gemelo físico y, en este caso, estos agentes externos son los vehículos que entran a utilizar cada cargador. Tener claro quién está haciendo uso de las instalaciones en cada momento es importante para poder identificar al causante de algún problema y para llevar un seguimiento del importe a pagar por el usuario de la estación. La vista representada en la figura 14 representa las matrículas de los vehículos que están actualmente haciendo uso de la estación de carga.

Para recuperar estos datos de la base de datos se realiza la consulta que se escribe a continuación en el listing 5.

Listing 5: Consulta de ID de vehículos cargando

```
from(bucket: "BucketTFG")  
|> range(start: v.timeRangeStart, stop: v.timeRangeStop)
```

```

|> filter(fn: (r) => r["_measurement"] == "stationA")
|> filter(fn: (r) => contains(value: r["name"], set: ${Chargers: json}))
|> filter(fn: (r) => r["_field"] == "vehicleID")
|> aggregateWindow(every: v.windowPeriod, fn: last, createEmpty: true)
|> yield(name: "last")

```

Listing 5: Consulta de ID de vehículos cargando

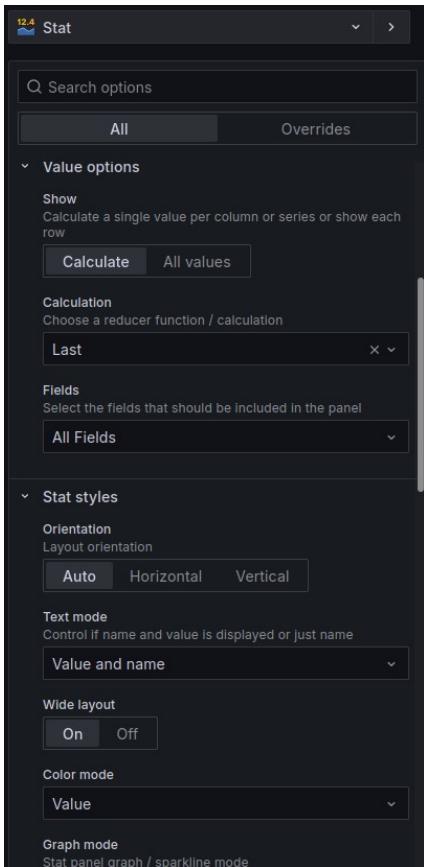


Figura 15: Configuración visualización matrículas

Además del código, se han utilizado distintas opciones de configuración de Grafana. Como

Se trata de una consulta sencilla que utiliza la variable de cargadores descrita en el apartado anterior. Comienza seleccionando el Bucket a consultar, utiliza la función *range(start, stop)* para filtrar el rango temporal de la consulta, utiliza la función *filter()* en las siguientes tres líneas para filtrar la medida a consultar, el campo buscado (*vehicleID*) y para filtrar que el resultado contenga el valor seleccionado de la variable *Chargers* utilizando la variable en formato JSON. La línea que utiliza la variable es la que permite que, al seleccionar el o los cargadores desde la interfaz de Grafana, se actualice la información mostrada en pantalla de forma dinámica. Por último, se crea una ventana agregada que se actualiza cada vez que lo mande el sistema, que la función de agregación de la consulta sea *last* (lo que significa que muestra el último resultado) y el parámetro *createEmpty* a *true* (lo que hace que si no recibe un valor en la consulta no muestre nada en vez de mostrar el último valor detectado). La función *yield()* devuelve el resultado de la consulta al usuario.

se ve en la figura 15, el modo de visualización escogido ha sido ‘Stat’ y se ha configurado, en ‘Value options’, seleccionando ‘Calculate’, la función reductora ‘Last’ y mostrar ‘All Fields’. Luego, en los estilos (‘Stat styles’), el modo de texto en ‘Value and name’ y el modo del color en valor. El color en específico se selecciona más abajo.

Además se ha realizado la transformación mostrada en a figura 16 la cual filtra el resultado para eliminar de la vista el tiempo de la métrica y solo muestre los datos de las matrículas de cada cargador.

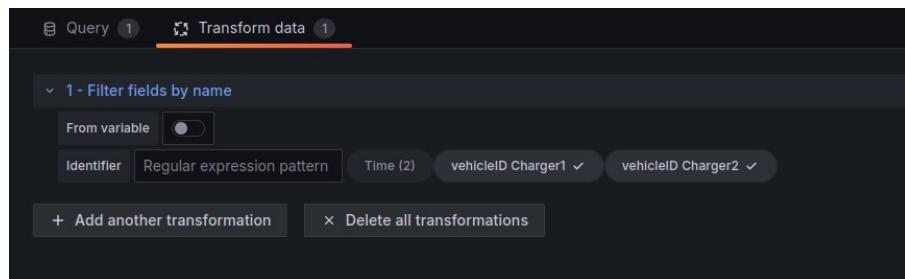


Figura 16: Transformación consulta visualización matrículas

5.4. Vista del porcentaje de batería



Figura 17: Vista del nivel de batería actual de los vehículos cargando

En esta vista se puede visualizar, tal como se muestra en la figura 17, el porcentaje de batería actual de los vehículos que se encuentran cargando en la estación. Esta vista es necesaria ya que representa la funcionalidad básica y principal de una estación de carga. Para una mejor visualización, se ha configurado para que, según el nivel de carga, este se visualice de distintos colores. En concreto cuando el nivel de carga se encuentra entre el 0-30 % se visualiza en rojo, del 31-80 % en naranja y del 81-100 % en verde.

La consulta realizada para recibir estos datos es la que se muestra a continuación en el listing 6, siendo una consulta muy simple y prácticamente igual a la realizada para las matrículas. Lo único que cambia es el campo a mostrar que, en este caso, es ‘bateryLevel’.

Listing 6: Consulta de porcentaje de batería de vehículos cargando

```
from(bucket: "BucketTFG")  
|> range(start: v.timeRangeStart, stop: v.timeRangeStop)  
|> filter(fn: (r) => r[_measurement] == "stationA")  
|> filter(fn: (r) => contains(value: r[name], set: ${Chargers: json}))  
|> filter(fn: (r) => r[_field] == "bateryLevel")  
|> aggregateWindow(every: v.windowPeriod, fn: last, createEmpty: true)  
|> yield(name: "last")
```

Listing 6: Consulta de porcentaje de batería de vehículos cargando

Esta vez, se ha elegido un gráfico de tipo ‘Gauge’. Los ajustes que se han configurado en Grafana son los representados en la figura 18. En concreto, en la figura 18a se ve que se ha elegido el campo calculado ‘Last’ y que muestre los campos numéricos únicamente y dentro de las opciones propias del gráfico ‘Gauge’, se ha activado la casilla para que muestre los marcadores de los umbrales (‘thresholds’). En las opciones estándar, como se ve en la figura 18b, se seleccionan los valores mínimo y máximo a 0 y 100 y el esquema de color dependiendo del valor y los umbrales. En la figura 18c se muestra la selección de los umbrales y colores.

5.5. Visualización del precio de la carga

En la visualización representada por la figura 19 se puede ver el precio acumulado en euros que los usuarios actuales de la estación tienen que pagar por la carga de sus vehículos. Se trata también de una vista dinámica dependiente de la variable creada en Grafana para el filtrado según los cargadores. La consulta en Flux, representada en el listing 7, es también muy parecida a las anteriores, ya que sigue siendo una consulta básica con la dificultad del uso de la variable de filtrado.

(a) Captura 1

(b) Captura 2

(c) Captura 3

Figura 18: Configuración de la vista de los niveles de batería

Listing 7: Consulta de precio a pagar por la carga de los vehículos actuales

```
from(bucket: "BucketTFG")
|> range(start: v.timeRangeStart, stop: v.timeRangeStop)
|> filter(fn: (r) => r["_measurement"] == "stationA")
|> filter(fn: (r) => contains(value: r["name"], set: ${Chargers: json}))
|> filter(fn: (r) => r["_field"] == "energyBill")
|> aggregateWindow(every: v.windowPeriod, fn: last, createEmpty: true)
|> yield(name: "last")
```

Listing 7: Consulta de precio a pagar por la carga de los vehículos actuales

Como se observa en la figura 20, la configuración es bastante sencilla. Se elige el gráfico de tipo ‘Stat’ con la función reductora ‘Last’, que muestre el valor solo de los campos numéricos y que muestre también el nombre del campo mostrado. Por último, se elige como unidad de



Figura 19: Visualización precio de la carga

(a) Configuración 1

- Orientation: Auto
- Text mode: Value and name
- Color mode: Value

(b) Configuración 2

- Unit: Euro (€)
- Min: auto
- Max: auto
- Field min/max: Off
- Decimals: auto
- Display name: none
- Color scheme: Single color (selected)
- No value: -

Figura 20: Configuración de la vista del coste de la carga energética

medida el Euro y el color de los valores.

5.6. Consumo energético por vehículo

Esta visualización está relacionada con el coste de la recarga ya que se trata de la energía consumida actualmente por cada vehículo que se encuentre haciendo uso de los cargadores.



Figura 21: Visualización consumo energético por vehículo

Como se ve en la figura 21, muestra los datos por cada cargador, en naranja, y medido en la unidad de medida típica de las baterías de vehículos eléctricos, el kilovatio hora (kWh). La consulta realizada es también similar a las anteriores y se muestra a continuación en el listing 8.

Listing 8: Consulta del consumo energético por vehículo actual en cada cargador

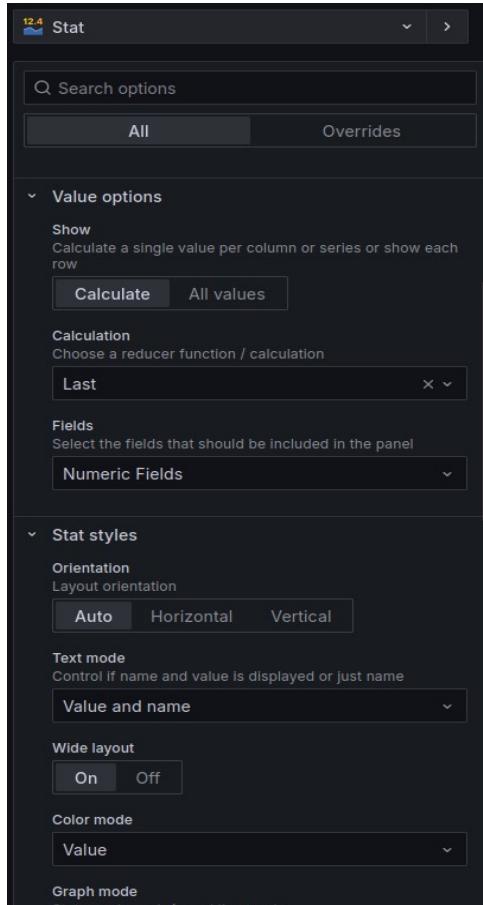
```
from(bucket: "BucketTFG")
|> range(start: v.timeRangeStart, stop: v.timeRangeStop)
|> filter(fn: (r) => r["_measurement"] == "stationA")
|> filter(fn: (r) => contains(value: r["name"], set: ${Chargers: json}))
|> filter(fn: (r) => r["_field"] == "energyConsumed")
|> aggregateWindow(every: v.windowPeriod, fn: last, createEmpty: true)
|> yield(name: "last")
```

Listing 8: Consulta del consumo energético por vehículo actual en cada cargador

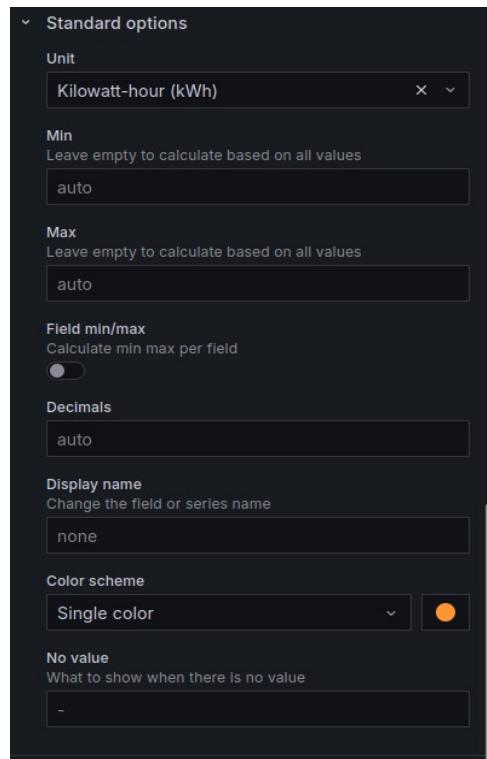
Como se observa en la figura 22, la configuración de esta visualización en Grafana es igual que la anterior, solo cambia el color mostrado y la unidad de medida que en este caso es el kWh.

5.7. Consumo total en las últimas 24 horas

En la figura 23 se observa el cálculo del consumo total de energía por cada cargador en las últimas 24 horas medido también en kWh. Este cálculo se basa en sumar los consumos totales de todos los vehículos en las últimas 24 horas agrupados por cargador. En este caso, la consulta



(a) Configuración 1



(b) Configuración 2

Figura 22: Configuración de la vista del gasto energético de los vehículos actuales por cargador



Figura 23: Visualización gasto energético por cargador

realizada en FLux es la mostrada en el listing 9.

Listing 9: Consulta del consumo energético en las últimas 24 horas por cargador

```
from(bucket: "BucketTFG")
|> range(start: -1d)
|> filter(fn: (r) => r._measurement == "stationA")
```

```

|> filter(fn: (r) => r._field == "energyConsumed" or r._field == "vehicleID")
|> filter(fn: (r) => contains(value: r["name"], set: ${Chargers: json}))
|> pivot(rowKey:[ "_time" ], columnKey: [ "_field" ], valueColumn: "_value")
|> group(columns: [ "name", "vehicleID" ])
|> sort(columns: [ "_time" ], desc: true)
|> unique(column: "vehicleID")
|> group(columns: [ "name" ])
|> sum(column: "energyConsumed")
|> yield(name: "totalEnergyConsumedXcharger")

```

Listing 9: Consulta del consumo energético en las últimas 24 horas por cargador

En esta consulta, lo primero es filtrar el rango de tiempo, la medida, el uso de la variable dinámica de Grafana y los campos a solicitar (*energyConsumed* y *vehicleID*). Teniendo todo esto se podrá averiguar la energía consumida total por cada vehículo.

La función *pivot()* permite formatear la salida actual de los datos a dos columnas, *energyConsumed* y *vehicleID*, y que la clave de cada fila sea la marca temporal de la medida. A continuación, se agrupan los resultados por vehículo y se ordenan por tiempo en orden descendente. De este modo queda en la primera fila de los datos de cada vehículo el último cálculo de la energía consumida. La función *unique* hace que el resultado solo muestre la primera línea de cada grupo de datos de los distintos vehículos.

A continuación se realiza otra agrupación en relación a los cargadores (de otro modo el resultado sería global y no por cada cargador) y, por último, se suman todas las columnas de *energyConsumed* y se devuelve el resultado al usuario.

En relación a la configuración de la vista en Grafana, como se puede ver en la figura 24, es exactamente igual a la anterior solamente cambiando la función de reducción de ‘Last’ a ‘Last*’. La diferencia entre estas dos es que ‘Last*’ excluye los valores nulos de las consultas, un cambio que en este caso no debería tener mucha relevancia.

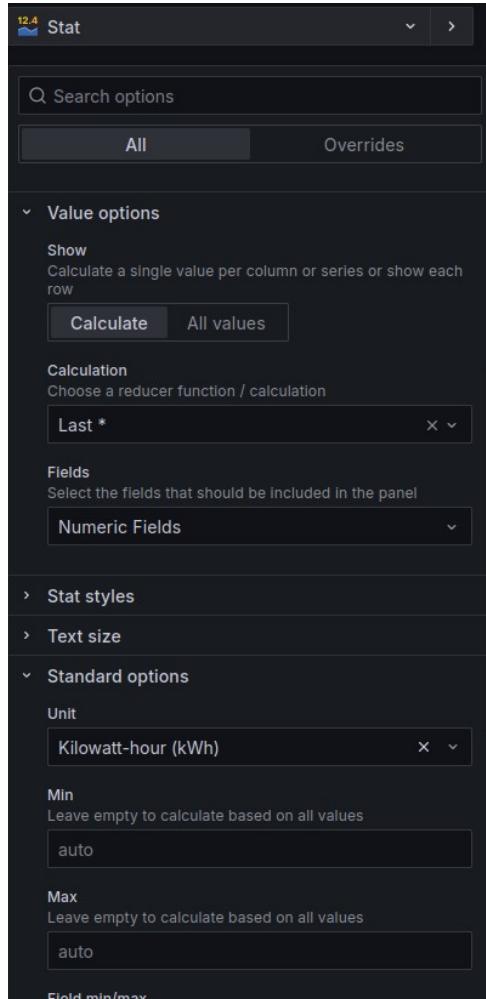


Figura 24: Configuración de la vista del gasto total por cargador

5.8. Velocidad de carga de la estación



Figura 25: Visualización velocidad de carga

En la figura 25 se observa la potencia de carga máxima (velocidad), medida también en

kilovatios hora (kWh), con la que cuenta la estación. En el caso concreto de este proyecto la velocidad es la misma para toda la estación, por eso solo se muestra un dato. La consulta escrita en Flux para este caso sería la mostrada en el listing 10.

Listing 10: Consulta de la velocidad de carga de la estación

```
from(bucket: "BucketTFG")  
|> range(start: v.timeRangeStart, stop: v.timeRangeStop)  
|> filter(fn: (r) => r["_measurement"] == "stationA")  
//|> filter(fn: (r) => contains(value: r["name"], set: ${Chargers: json}))  
|> filter(fn: (r) => r["name"] == "Charger1")  
|> filter(fn: (r) => r["_field"] == "maxPower")  
|> aggregateWindow(every: v.windowPeriod, fn: last, createEmpty:  
false)  
|> yield(name: "last")
```

Listing 10: Consulta de la velocidad de carga de la estación

Se trata también de una consulta bastante básica en la que se establece el parámetro de *createEmpty* a *false* para que evite mostrar datos nulos en caso de que haya algún problema de conexión momentáneo a la base de datos. En el filtro de las etiquetas correspondientes a los cargadores, se filtra únicamente por *Charger1* para que muestre solo un dato. En el caso de que cada cargador pudiera operar a una velocidad de carga distinta, se debería comentar esa linea y eliminar los caracteres de comentario de la anterior para utilizar la variable dinámica creada.

Como se observa en la figura 26, la configuración de la vista en Grafana es igual a las anteriores, seleccionando el tipo de gráfico ‘Stat’, la función de reducción ‘Last*’, que muestre los campos numéricos y la unidad de medida el kilovatio hora (kWh). La diferencia sería el color, esta vez elegido el amarillo.

The screenshot shows the 'Stat' configuration panel in Grafana. Under 'Value options', the 'Show' setting is set to 'Calculate a single value per column or series or show each row'. The 'Calculate' button is highlighted. The 'Calculation' dropdown is set to 'Last *'. Under 'Fields', 'Numeric Fields' is selected. Under 'Stat styles', 'Orientation' is set to 'Auto', 'Text mode' to 'Auto', 'Color mode' to 'Value', and 'Graph mode' is partially visible.

(a) Configuración 1

The screenshot shows the 'Stat' configuration panel in Grafana. Under 'Standard options', the 'Unit' is set to 'Kilowatt-hour (kWh)'. The 'Min' and 'Max' fields are both set to 'auto'. The 'Field min/max' toggle is off. The 'Decimals' field is set to 'auto'. The 'Display name' field is set to 'none'. The 'Color scheme' is set to 'Single color' with a yellow circle icon. The 'No value' field is set to 'none'.

(b) Configuración 2

Figura 26: Configuración de la vista de la velocidad de carga de la estación

5.9. Precio de la electricidad

En la figura 27 se observa la visualización en Grafana del precio en euros de la electricidad por kilovatio hora. Se trata de otra visualización y consulta simples muy similar a las anteriores. En este caso, al igual que el anterior también se refleja un solo dato para toda la estación al contar con cargadores similares. En el caso de que cada cargador pudiera utilizar una velocidad de carga o un tipo de cargador distintos entre sí, se mostraría la información del precio de la electricidad por kWh para cada cargador. La consulta en Flux a la base de datos representada en el listing 11, es similar a la anterior, cambiando el campo de datos a consultar.



Figura 27: Visualización precio de la electricidad por kWh

Listing 11: Consulta del precio del kWh

```
from(bucket: "BucketTFG")
|> range(start: v.timeRangeStart, stop: v.timeRangeStop)
|> filter(fn: (r) => r["_measurement"] == "stationA")
//|> filter(fn: (r) => contains(value: r["name"], set: ${Chargers:json}))
|> filter(fn: (r) => r["name"] == "Charger1")
|> filter(fn: (r) => r["_field"] == "price")
|> aggregateWindow(every: v.windowPeriod, fn: last, createEmpty: false)
|> yield(name: "last")
```

Listing 11: Consulta del precio del kWh

Al igual que en el caso anterior, para diferenciar el precio entre cargadores, se debería descomentar el filtro de la variable dinámica y comentar el actual filtro para el *Charger1*.

La configuración mostrada en la figura 28 es muy similar a la anterior, con la excepción de la unidad de medida siendo Euros y el color seleccionado para los valores siendo verde.

5.10. Cola de espera

En la visualización representada por la figura 29 se puede observar la cola de espera implementada para la estación. Esta cola muestra las matrículas de los vehículos que se encuentran esperando y el tiempo que llevan en la cola. Al entrar a cargar el primer vehículo, todos se desplazan en un puesto. En otras palabras, se trata de una cola FIFO (First In First Out). La

The screenshot shows the 'Stat' configuration interface. Under 'Value options', there is a 'Show' section with a dropdown for 'Calculate' set to 'All values'. Below it is a 'Calculation' section with a dropdown for 'Last *'. Under 'Fields', a dropdown is set to 'Numeric Fields'. Under 'Stat styles', there are sections for 'Orientation' (Auto), 'Text mode' (Auto), 'Color mode' (Value), and 'Graph mode'.

(a) Configuración 1

The screenshot shows the 'Standard options' configuration interface. It includes sections for 'Unit' (Euro (€)), 'Min' (auto), 'Max' (auto), 'Field min/max' (disabled), 'Decimals' (auto), 'Display name' (none), 'Color scheme' (Single color), and a 'No value' button.

(b) Configuración 2

Figura 28: Configuración de la vista del precio de la electricidad por kWh

Waiting Queue				
vehicle1	vehicle2	vehicle3	vehicle4	vehicle5
3595MIM	3229YIA	9914QQZ	4277ZNX	4923VJS
waitingTime1 39.8 mins	waitingTime2 30.8 mins	waitingTime3 22.8 mins	waitingTime4 15.8 mins	waitingTime5 1.82 mins

Figura 29: Visualización cola de espera

consulta realizada sobre la medida “stationAQueue” es la mostrada en el listing 12.

Listing 12: Consulta cola de espera

```
from(bucket: "BucketTFG")
```

```

|> range(start: -30s)
|> filter(fn: (r) => r["_measurement"] == "stationAQueue")
|> filter(fn: (r) => r["_field"] == "waitingTime1" or r["_field"]
== "waitingTime2" or r["_field"] == "waitingTime3" or r["_field"]
== "waitingTime4" or r["_field"] == "waitingTime5" or r["_field"]
== "vehicle5" or r["_field"] == "vehicle4" or r["_field"] == "
vehicle3" or r["_field"] == "vehicle2" or r["_field"] == "vehicle1"
")
|> aggregateWindow(every: v.windowPeriod, fn: last, createEmpty:
false)
|> yield(name: "last")

```

Listing 12: Consulta cola de espera

Esta consulta le pide a la base de datos la información de los campos de las matrículas de todos los vehículos de la cola de espera y su tiempo correspondiente y los devuelve al usuario.

La configuración de Grafana para esta vista, mostrada en la figura 30, consiste en la selección del tipo de gráfica ‘Stat’, función de reducción ‘Last’, mostrar todos los campos y la unidad de medida segundos. Adicionalmente, esta visualización cuenta con una transformación de los datos de Grafana. Esta transformación mostrada en la figura 31, es del tipo ‘Filter fields by name’ lo que sirve para elegir qué campos se muestran y cuáles no. En este caso se han seleccionado todos los campos menos la marca temporal.

5.11. Tiempo de espera medio hoy

La métrica mostrada en la visualización de la figura 32, se trata del tiempo de espera medio de las últimas 24 horas para comenzar a cargar el vehículo. El tiempo de espera medio se calcula con la variable *waitingTime1* únicamente ya que esta es la que contiene el último valor del tiempo de espera de cada vehículo que entra a cargar. Esta métrica podría ser útil, en caso de tener más cargadores disponibles pero apagados, si se añade una alerta que avise cuando el tiempo medio de espera alcance cierto valor y advierta que sería necesario encender un nuevo cargador para reducir ese tiempo de espera. La consulta realizada recibe los datos del tiempo en segundos y sería la siguiente:

The screenshot shows the 'Stat' configuration interface. Under 'Value options', there is a 'Show' section with 'Calculate' and 'All values' buttons, and a 'Calculation' section with 'Last' selected. Under 'Stat styles', there are sections for 'Orientation' (Auto), 'Text mode' (Auto), and 'Color mode' (Value). A search bar at the top is empty.

(a) Configuración 1

The screenshot shows the 'Standard options' section of the 'Stat' configuration. It includes fields for 'Unit' (seconds (s)), 'Min' (auto), 'Max' (auto), 'Field min/max' (disabled), 'Decimals' (auto), 'Display name' (none), and 'Color scheme' (Green-Yellow-Red (by value)).

(b) Configuración 2

Figura 30: Configuración de la vista de la cola de espera

The screenshot shows the 'Transform data' tab. It displays a transformation rule '1 - Filter fields by name' with a 'From variable' switch turned off. The 'Identifier' section lists fields: Time (10), vehicle1, vehicle2, vehicle3, vehicle4, vehicle5, waitingTime1, waitingTime2, waitingTime3, waitingTime4, and waitingTime5. Below the rule are buttons for '+ Add another transformation' and 'Delete all transformations'.

Figura 31: Transformación datos cola de espera

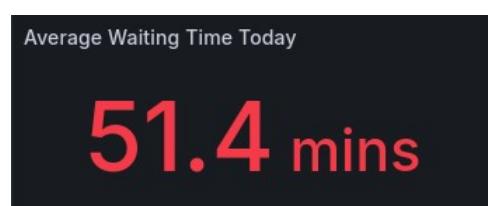


Figura 32: Visualización tiempo medio de espera hoy

Listing 13: Consulta y cálculo del tiempo medio de espera

```
from(bucket: "BucketTFG")  
|> range(start: -1d) // Selecciona el último día  
|> filter(fn: (r) => r._measurement == "stationAQueue")  
|> filter(fn: (r) => r._field == "waitingTime1" or r._field == "  
vehicle1")  
|> pivot(rowKey: ["_time"], columnKey: ["_field"], valueColumn: "  
_value")  
|> group(columns: ["vehicle1"])  
|> sort(columns: ["_time"], desc: true)  
|> unique(column: "vehicle1")  
|> group()  
|> mean(column: "waitingTime1")  
|> yield(name: "mean_waiting_time_vehicle1")
```

Listing 13: Consulta y cálculo del tiempo medio de espera

La primera acción que realiza esta consulta del listing 13 es el filtrado por rango de tiempo del último día (24 horas), a continuación filtra la medida (*stationAQueue*) y los campos a recibir que son *waitingTime1* y *vehicle1*. A continuación utiliza la función *pivot()* para reordenar la tabla de salida dejando la marca temporal como la clave de las filas y los campos consultados como columnas, siendo la clave el nombre del campo y el valor la medición del campo. El siguiente paso es agrupar la tabla por vehículo y ordenar las filas por orden temporal descendente. De esta manera se deja el dato más reciente el primero de cada grupo y se eliminan todos los demás con la función *unique()*. Por último, se vuelven a agrupar todos los datos en un mismo grupo, calcula la media de los mismos y los devuelve al usuario.

La configuración en Grafana, como se observa en la figura 33, vuelve a estar formada por el tipo de gráfico ‘Stat’, utilizando la función de reducción ‘Last’ y mostrando todos los campos pero solo su valor. La unidad de medida utilizada son los segundos, mostrando el valor con un solo decimal y el valor a mostrar cuando no se tengan datos debe ser un cero (0).

(a) Configuración 1

(b) Configuración 2

Figura 33: Configuración de la vista del tiempo medio de espera del último día

5.12. Tiempo de espera medio últimos 5 días

La última visualización, representada en la figura 34, trata de el cálculo de la media del tiempo de espera para los últimos 5 días y presentado en un gráfico de barras. Esto puede servir para comparar, por ejemplo, si el tiempo de espera de alguno de los días del gráfico se sale de la norma o no. La consulta realizada en Flux es relativamente compleja ya que involucra la creación de una función personalizada y el uso de varias variables. La consulta está representada en el listing 14.

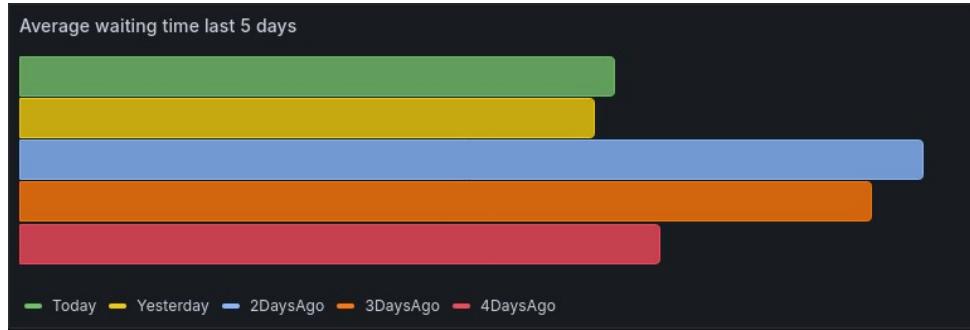


Figura 34: Visualización media tiempo espera últimos 5 días

Listing 14: Consulta y cálculo tiempo medio de espera últimos 5 días

```
averageWaitingTime = (start, stop, day) =>
  from(bucket: "BucketTFG")
    |> range(start: start, stop: stop)
    |> filter(fn: (r) => r._measurement == "stationAQueue")
    |> filter(fn: (r) => r._field == "waitingTime1" or r._field == "vehicle1")
    |> pivot(rowKey:[ "_time" ], columnKey: [ "_field" ], valueColumn: "_value")
    |> group(columns: [ "vehicle1" ])
    |> sort(columns: [ "_time" ], desc: true)
    |> unique(column: "vehicle1")
    |> group()
    |> mean(column: "waitingTime1")
    |> map(fn: (r) => ({r with day: day, mean: r.waitingTime1, media: "media"}))
    |> keep(columns: [ "day", "mean", "media" ])

avgDay1 = averageWaitingTime(start: -1d, stop: now(), day: "Today")
avgDay2 = averageWaitingTime(start: -2d, stop: -1d, day: "Yesterday")
avgDay3 = averageWaitingTime(start: -3d, stop: -2d, day: "2DaysAgo")
avgDay4 = averageWaitingTime(start: -4d, stop: -3d, day: "3DaysAgo")
avgDay5 = averageWaitingTime(start: -5d, stop: -4d, day: "4DaysAgo")
```

```

union( tables: [avgDay1, avgDay2, avgDay3, avgDay4, avgDay5])
|> group()
|> pivot(rowKey: ["media"], columnKey: ["day"], valueColumn: "mean")
|
|> yield(name: "averageWaitingTime5Days")

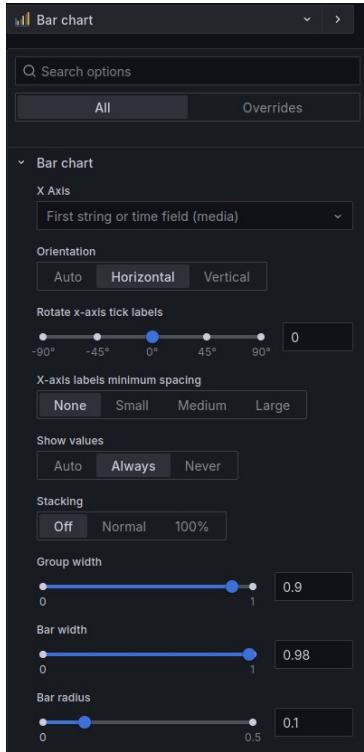
```

Listing 14: Consulta y cálculo tiempo medio de espera últimos 5 días

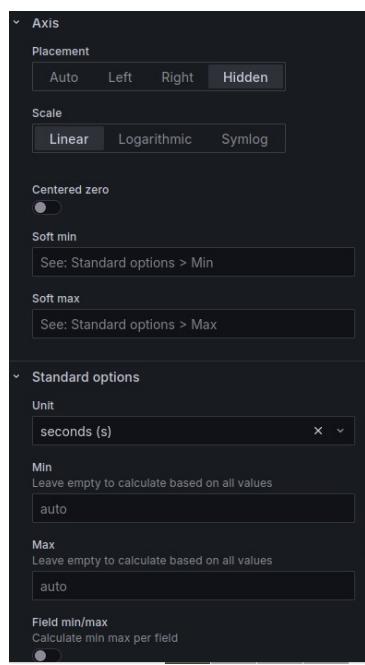
En esta consulta se crea una función que sirve para calcular la media del tiempo de espera de un periodo de tiempo que se le pase por parámetros. También recibe por parámetros un valor del tipo cadena de caracteres que se utiliza para darle nombre al cálculo de los datos. Esta función es una versión adaptada de la consulta utilizada en la visualización de la media de tiempo de espera del día actual. Primero, se establece el filtro del rango de tiempo y los filtros de la medida a consultar (*stationAQueue*) y los campos *waitingTime1* y *vehicle1*. Utiliza la función *pivot()* para reorganizar la tabla de salida de datos dejando como claves de las filas la marca temporal y en las columnas los campos con sus valores. A continuación, agrupa los datos por vehículos, los ordena en orden temporal descendente y elimina todas las filas menos la más reciente de cada vehículo. Vuelve a agrupar los datos restantes y calcula la media de los campos *waitingTime1*. Finalmente utiliza la función *map()* para asignar al campo *day* el parámetro del nombre del día pasado a la función, al campo *mean* el resultado del cálculo de la media y se crea una etiqueta *media* que se utilizará más tarde para ordenar los datos. También se usa la función *keep()* como forma de asegurarse de que solo se devuelven las columnas de la función *map()*.

La consulta utiliza cinco variables que utiliza para llamar a la función, con los rangos temporales de cada día enviados por parámetros, y almacenar sus resultados. Luego realiza la operación de unión sobre las cinco variables agrupando sus datos y reorganizándolos para dejar la etiqueta *media* como clave de las filas, los nombres asignados a los días como claves de las columnas y la media (*mean*) como los valores previamente a devolver el resultado final al usuario.

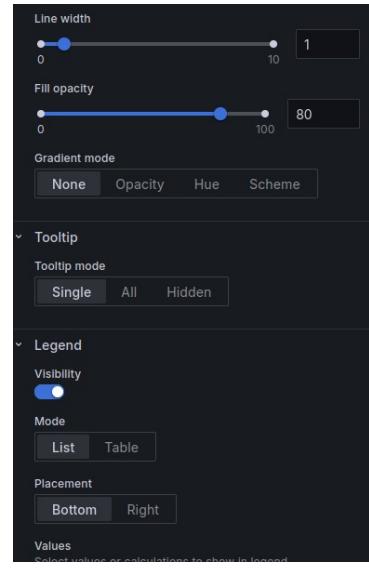
En el caso de esta visualización, como se muestra en la figura 35, se selecciona el tipo de gráfico como gráfico de barras ('Bar chart') y se rellena la configuración del mismo. Entre las opciones de configuración que se encuentran y que se han modificado para personalizar esta visualización están la orientación de las barras que se han puesto horizontales, el mostrar



(a) Configuración 1



(b) Configuración 2



(c) Configuración 3

Figura 35: Configuración de la vista del tiempo de espera medio de los últimos 5 días

los valores de las barras seleccionado a ‘Always’, el grosor de las barras, la herramienta de ayuda visual (‘Tooltip’) configurada al valor ‘Single’, la leyenda visible abajo en formato de lista, los ejes X e Y invisibles y la unidad de medida el segundo. Adicionalmente, se ha añadido una transformación de los datos en Grafana que se muestra en la figura 36. Esta opción de transformación da la opción de cambiar el orden en el que se muestran los campos y renombrarlos. Se ha utilizado para visualizar siempre el día actual en la barra superior y bajar en orden temporal.

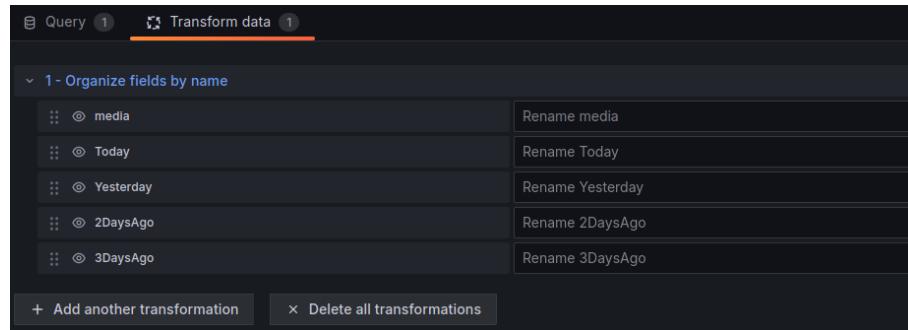


Figura 36: Transformación de los datos para la media de espera de varios días

5.13. Resultado final

En este apartado se muestra en la figura 37 el resultado final mostrando todos los datos y en la figura 38 el resultado final utilizando el filtrado de un cargador.

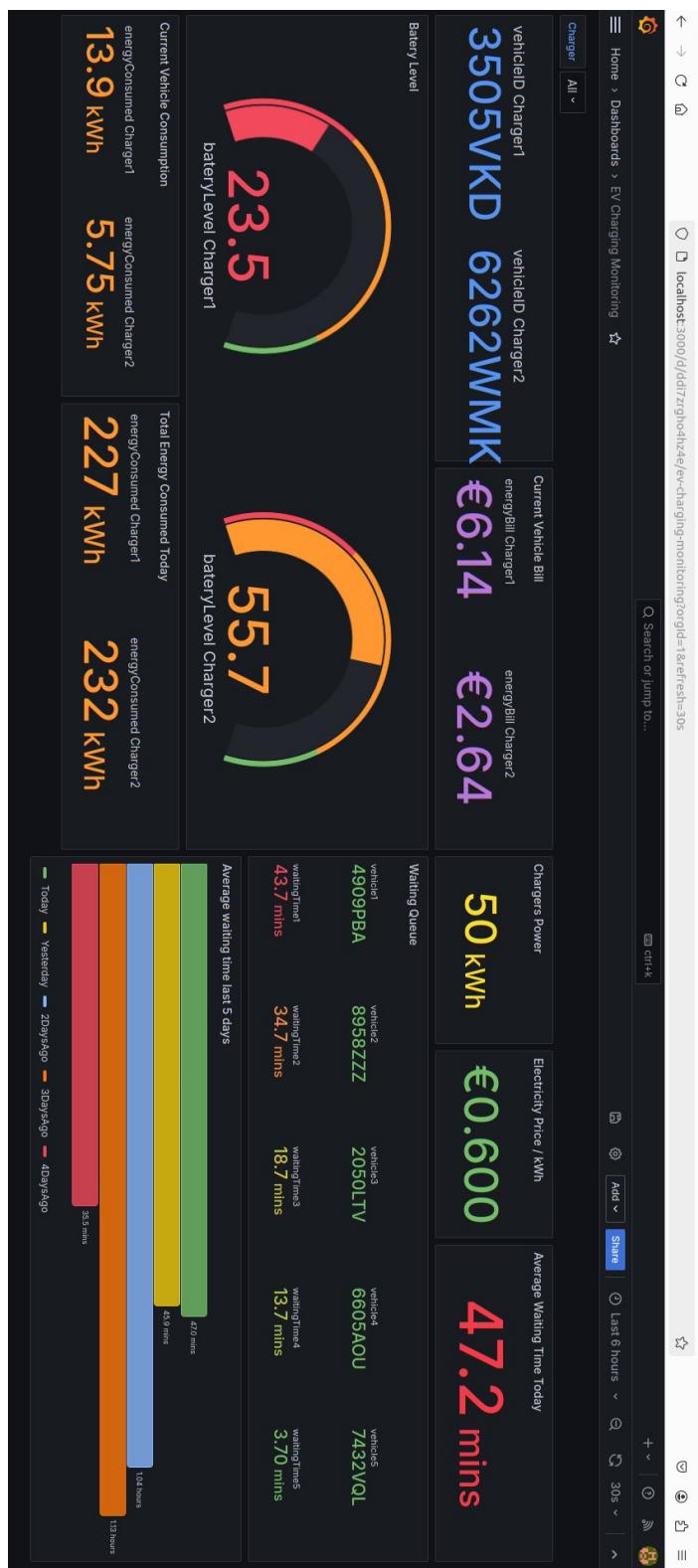


Figura 37: Resultado final

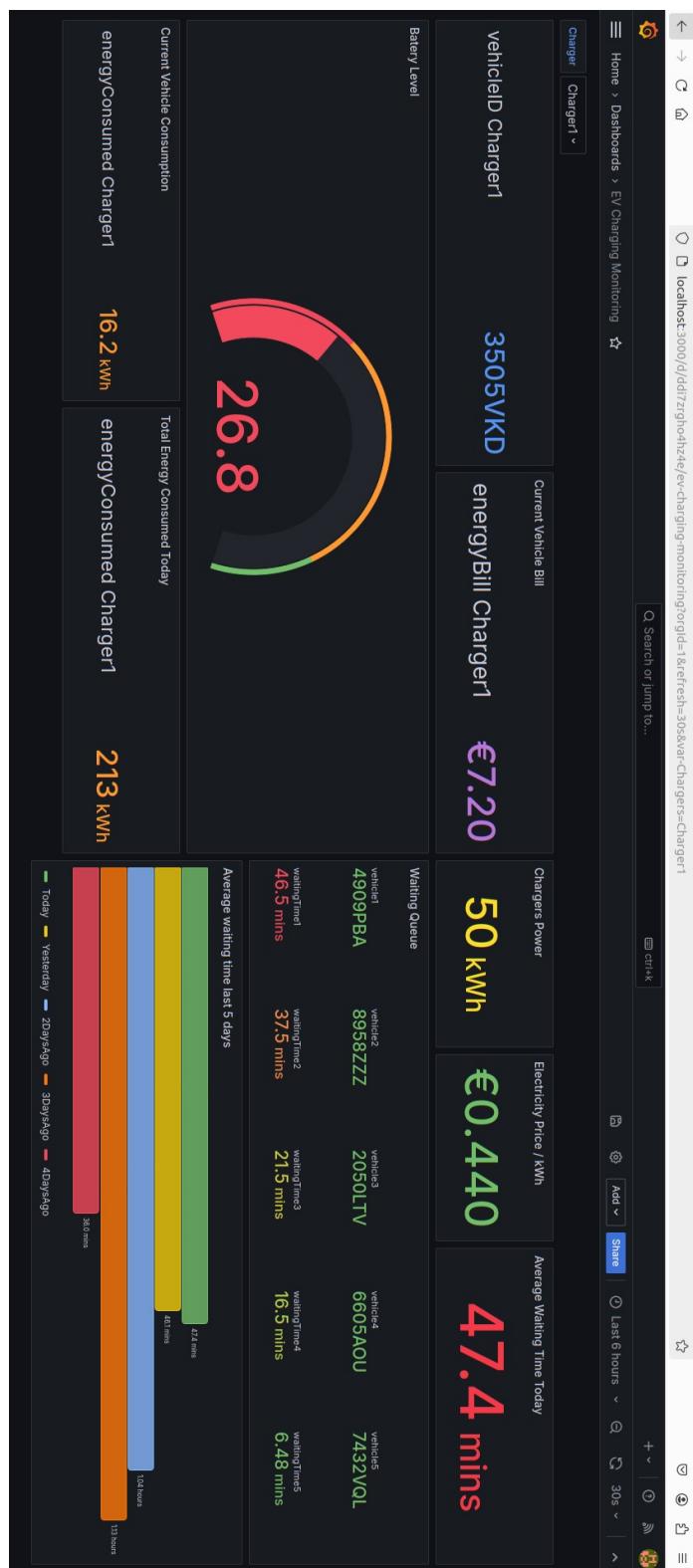


Figura 38: Resultado final usando filtro

6

Contenedorización de la aplicación

Para proporcionar un sistema fácil de instalar y probar, se ha empaquetado toda la infraestructura del sistema en contenedores que ejecutan las instancias de Influx y Grafana y el programa ‘Datafaker.py’, mostrado en el Apéndice B de este documento, que alimenta la base de datos. En este capítulo se describe el proceso de empaquetado en contenedores que se ha realizado sobre el sistema utilizando la tecnología de Docker y Docker-compose.

6.1. Estructura y contenido de los contenedores

La estructura del proyecto, mostrada en forma de árbol de directorios para que sea más sencillo de entender todo, es la siguiente:

```
TFG_DASHBOARD_DT
|   docker-compose.yml
|   grafana
|       └── provisioning
|           ├── dashboards
|           |   └── dashboard.yml
|           |   └── EVChargingMonitoring.json
|           └── datasources
|               └── datasource.yml
|
|   python
|       dataFaker.py
|       Dockerfile
|       requirements.txt
```

La creación de los distintos contenedores se ha definido en el archivo ‘docker-compose.yml’. Este archivo se muestra en el listing 15 y define tres contenedores. El primer contenedor que crea es el de InfluxDB. Define su imagen como la última versión disponible e indica que transmite a través del puerto 8086 (el puerto por defecto para Influx). También define las variables de entorno necesarias para la creación y posible uso de la base de datos y define un volumen de almacenamiento *influx-storage*. Por último se establece una prueba de salud que utilizarán los contenedores de Python y Grafana para determinar si pueden ejecutarse o no.

El segundo contenedor definido es el de Grafana. Utilizando la imagen de su versión más reciente y transmitiendo en el puerto por defecto de las instalaciones de Grafana (puerto 3000), también define un volumen de almacenamiento *grafana-storage* y además una ruta para los archivos de aprovisionamiento como la conexión con la base de datos y la creación del panel de control. Por último, se define la dependencia con el contenedor de InfluxDB bajo la condición de que el servicio de Influx esté “sano”.

El tercer y último contenedor definido es el encargado de ejecutar el programa de python. Este no utiliza directamente una imagen predefinida, sino un archivo Dockerfile para la definición de esta. Indica la ubicación el contexto dentro del árbol de directorios y la ubicación del archivo Dockerfile. Además indica también la dependencia con el contenedor de Influx de la misma forma que el de Grafana y define unas variables de entorno necesarias para la ejecución de ‘dataFaker.py’.

Listing 15: Archivo docker-compose

```
version: '3.8'

services:
  influxdb:
    image: influxdb:latest
    ports:
      - 8086:8086
    environment:
      - INFLUXDB_DB=db0
      - DOCKER_INFLUXDB_INIT_MODE=setup
      - DOCKER_INFLUXDB_INIT_USERNAME=admin
```

```

- DOCKER_INFLUXDB_INIT_PASSWORD=adminpassword1234
- DOCKER_INFLUXDB_INIT_ORG=UMATFG
- DOCKER_INFLUXDB_INIT_BUCKET=BucketTFG
- DOCKER_INFLUXDB_INIT_RETENTION=1w
- DOCKER_INFLUXDB_INIT_ADMIN_TOKEN=ADMIN_TOKEN

volumes:
- influxdb-storage:/var/lib/influxdb

healthcheck:
  test: "curl -f http://localhost:8086/ping"
  interval: 10s
  timeout: 5s
  retries: 5

grafana:
  image: grafana/grafana:latest
  ports:
  - 3000:3000
  volumes:
  - grafana-storage:/var/lib/grafana
  - ./grafana/provisioning/:/etc/grafana/provisioning
depends_on:
  influxdb:
    condition: service_healthy

python:
  build:
    context: ./python
    dockerfile: ./Dockerfile
  volumes:
  - ./python:/app
depends_on:
  influxdb:
    condition: service_healthy

```

```

environment:
  - DOCKER_INFLUXDB_INIT_ADMIN_TOKEN=ADMIN_TOKEN
  - DOCKER_INFLUXDB_INIT_ORG=UMATFG
  - DOCKER_INFLUXDB_INIT_BUCKET=BucketTFG

volumes:
  influxdb-storage:
  grafana-storage:

```

Listing 15: Archivo docker-compose

Dentro del directorio ‘Grafana/provisioning’, destinado al aprovisionamiento del contenido de Grafana, se encuentran los directorios de ‘dashboards’ (tableros) y ‘datasources’ (fuentes de datos). En el directorio de ‘dashboards’ se encuentra el archivo ‘EVCharginMonitoring.json’ que corresponde al panel de control desarrollado y el archivo ‘dashboard.yml’ (listing 16), definido según la documentación oficial de Grafana, que define una serie de parámetros del tablero como son el proveedor (fuente de datos) y la ruta al archivo json.

Listing 16: Archivo dashboard.yml

```

apiVersion: 1
providers:
- name: InfluxDB
  folder: ''
  type: file
  disableDeletion: false
  editable: true
  options:
    path: /etc/grafana/provisioning/dashboards

```

Listing 16: Archivo dashboard.yml

En el directorio ‘datasources’ se encuentra únicamente el archivo ‘datasource.yml’ (listing 17) que define la información necesaria de la base de datos para su conexión con Grafana. Esta información incluye, entre otras cosa, la url de la base de datos, las credenciales, el ID de la base de datos dentro de Grafana y el lenguaje de consulta a utilizar.

Listing 17: Archivo datasource.yml

```
apiVersion: 1

datasources:
  - name: InfluxDB
    type: influxdb
    access: proxy
    database: db0
    user: admin
    password: admin
    basicAuth: true
    basicAuthUser: admin
    url: http://influxdb:8086
    isDefault: true
    editable: true
    uid: adi7yqk7zvvggd
    jsonData:
      version: Flux
      organization: UMATFG
      defaultBucket: BucketTFG
      #tlsSkipVerify: true
    secureJsonData:
      token: ADMIN_TOKEN
      password: adminpassword1234
      basicAuthPassword: adminpassword1234
```

Listing 17: Archivo datasource.yml

Dentro del directorio del contexto de Python, se encuentran tres archivos, ‘Dockerfile’ (listing 18), ‘dataFaker.py’ (el programa a ejecutar) y un documento de texto con los requisitos del contenedor, que en este caso el único requisito a instalar es ‘*influxdb_client*’. El archivo ‘Dockerfile’ define la versión de Python a utilizar y una serie de instrucciones. Establece el directorio de trabajo a ‘/app’ e indica que debe instalar los requisitos y ejecutar el comando “python3 dataFaker.py”.

Listing 18: Archivo Dockerfile

```
FROM python:3.10

WORKDIR /app

COPY requirements.txt requirements.txt
RUN pip install --no-cache-dir -r requirements.txt

COPY . .

CMD [ "python3" , "dataFaker.py" ]
```

Listing 18: Archivo Dockerfile

7

Conclusiones y Líneas Futuras

En este capítulo se describen las conclusiones extraídas del trabajo realizado en este proyecto, haciendo un pequeño resumen de lo desarrollado en el mismo y las posibles líneas de trabajo futuras.

7.1. Conclusiones

Con el objetivo de lograr construir las llamadas ciudades inteligentes, el uso de los vehículos eléctricos es esencial y si no se cuenta con abundantes puntos de recarga, el uso de estos se vuelve realmente complicado. Para ello siempre es importante conocer qué es lo que ocurre en estas estaciones de recarga en todo momento. Embarcarme en este proyecto me ha hecho aprender e interesarme más por el tema de las ciudades inteligentes y de los gemelos digitales, siendo estas herramientas tan amplias como útiles y potentes.

El sistema desarrollado se encuentra, dentro de un gemelo digital, en la parte de la monitorización. Se ha desarrollado un sistema capaz de monitorizar todas las métricas que una estación de carga de vehículos eléctricos pueda emitir, siendo posible la ampliación o la eliminación de visualizaciones según convenga. Además de la funcionalidad de la monitorización, también se ha desarrollado un programa capaz de simular el funcionamiento de una estación de carga real. Este programa puede servir tanto para utilizarlo para alimentar la base de datos con información de prueba como para realizar un primer acercamiento a lo que podrían ser otras áreas del gemelo digital, la simulación por ejemplo.

El uso de Grafana ha aportado una forma de presentar los distintos gráficos así como de publicarlos y de manejar a los posibles usuarios del sistema y sus permisos. Además de la visualización, con el uso de Grafana también es posible la creación de alertas según una serie

de parámetros seleccionados para las métricas deseadas. Esto puede ser útil, por ejemplo, para habilitar otro cargador si el tiempo medio de espera está siendo muy elevado.

Por otra parte, la elección de InfluxDB como sistema de bases de datos, ha aportado un sistema robusto y optimizado para el uso necesario en este proyecto, capaz de recibir miles de peticiones de ingesta de datos sin problema y con una documentación bastante amplia y bien redactada. Esta documentación, complementada con la gran comunidad con la que cuenta esta plataforma hacen que la resolución de problemas se simplifique. Además también cuenta con muy buenas opciones de escalabilidad para un futuro.

En definitiva, considero que el desarrollo de este proyecto se puede considerar un éxito ya que, dejando espacio para la ampliación, se han logrado todos los objetivos y requisitos planteados inicialmente (ver capítulo 3), además de haber sido desarrollado con tecnologías de código abierto.

En cuanto a mi aprendizaje personal, considero que ha sido un proyecto muy productivo ya que he podido desarrollar una aplicación utilizando tecnologías de código abierto y tan actuales e importantes en el mundo laboral como son Python, la base de datos NoSQL InfluxDB y Grafana. La mayor dificultad del proyecto ha sido delimitar sus límites, estructura y las tecnologías a utilizar para su desarrollo. Una vez se tenía todo esto claro, las pocas dificultades destacables vinieron con el uso interno de Grafana, por ejemplo, para conseguir hacer uso correctamente de la variable de filtrado en el lenguaje de consulta Flux. Consultando en distintos foros, descubrí que era un problema común y no llegué a encontrar una respuesta que terminara de adaptarse a las necesidades del proyecto. Aún así, con lo aprendido en la documentación oficial y en los foros de la comunidad, se consiguió solucionar el problema logrando correctamente el requisito del filtrado de los cargadores.

7.2. Líneas Futuras

En el desarrollo de este proyecto se ha intentado abarcar el mayor número de funcionalidades desarrolladas para una primera versión completa de la herramienta. Sin embargo, aún queda camino para la mejora y el alcance de un sistema más completo y útil.

7.2.1. Creación visualización modelo físico

Los gemelos digitales suelen contar con una representación gráfica del modelo físico y del virtual que facilite la visualización y el entendimiento de los datos relacionados con el mismo. En un primer momento se pensó en incluir esta representación a la primera versión de este proyecto pero se decidió no hacerlo ya que iba a requerir una inversión notable en esfuerzo de desarrollo, relativo a un valor limitado para el usuario debido a la redundancia de información con lo ya presentado. Sin embargo, sigue siendo buena idea incluirlo en versiones futuras del proyecto ya que ayudaría a obtener una visualización más intuitiva de algunas métricas.

7.2.2. Alojamiento servidor remoto

Algo esencial en todo sistema web es el alojamiento. Actualmente el sistema se encuentra únicamente en local pero sería de crucial importancia mudar el sistema a un servidor remoto, tanto su base de datos como la parte de Grafana. Esto permitiría, en cuanto a la base de datos, la recepción de métricas con una disponibilidad casi del 100 % del tiempo ya que actualmente solo genera y recibe datos al tener la máquina virtual ejecutándose. En cuanto a Grafana, tener el sistema, con las configuraciones adecuadas, en un servidor remoto permitiría que cualquier usuario con acceso y credenciales pudiera acceder al sistema desde su propio equipo.

Esta tarea se podría realizar haciendo uso de los servicios de alojamiento de InfluxData y de Grafana Labs. Sin embargo, si se quiere mantener el uso de las versiones de código abierto de estas tecnologías y toda la capacidad de control sobre el sistema que ello otorga, se debe instalar en servidores de uso general que corran un sistema operativo sobre el que poder instalar el sistema.

7.2.3. Conexión con fuentes de datos reales

El uso de los datos simulados con un programa Python es muy buena práctica para diseñar inicialmente el sistema y realizar pruebas sobre el mismo, incluso para un primer acercamiento con la función de simulación de un gemelo digital, pero no para el fin planeado para este sistema, la monitorización de un sistema real.

7.2.4. Extensión a manejo de múltiples estaciones

En caso de que se deseen manejar múltiples estaciones de carga de vehículos eléctricos con este sistema, deberían crearse un tablero o panel de control para cada estación. Una posible forma de mantenerlos todos organizados gráficamente sería la creación de un tablero de Grafana que implemente una visualización de un mapa usando el complemento nativo “Geomap”. Utilizando este complemento se podría situar en el mapa, mediante coordenadas, el lugar en el que se encuentra cada estación y asignar el tablero de cada estación a su punto en el mapa correspondiente. De esta forma, al seleccionar el punto del mapa, llevaría al panel de control correspondiente de la estación.

7.2.5. Añadir varios tipos de cargadores

La realidad es que existen cargadores de vehículos eléctricos de distintos tipos. Existen cargadores para estaciones o la vía pública que, al igual que los surtidores de gasolina, cuentan con varios tipos de conectores [25] pero también existen cargadores que solo cuentan con un tipo de conector.

En caso de trabajar con una estación formada por varios cargadores distintos entre sí y que solo cuenten con un tipo de cargador cada uno, sería necesario comenzar a manejar la variable del tipo de cargador. Esto podría conllevar no solo añadir una variable más al sistema, sino modificar el sistema de colas para que exista una por cada tipo de cargador disponible.

Referencias

- [1] *¿Qué es la tecnología de gemelos digitales? - Explicación de la tecnología de gemelos digitales - AWS.* URL: <https://aws.amazon.com/es/what-is/digital-twin/>.
- [2] Kent Beck et al. *Manifiesto por el Desarrollo Ágil de Software.* URL: <https://agilemanifesto.org/iso/es/manifesto.html>.
- [3] Kent Beck et al. *Principios del Manifiesto Ágil.* 2001. URL: <https://agilemanifesto.org/iso/es/principles.html>.
- [4] Igor Bobriakov. *Prometheus vs InfluxDB / MetricFire.* Oct. de 2023. URL: <https://www.metricfire.com/blog/prometheus-vs-influxdb/>.
- [5] *Chronograf/InfluxData.* URL: <https://www.influxdata.com/time-series-platform/chronograf/>.
- [6] *Docker Hub Container Image Library / App Containerization.* URL: <https://hub.docker.com/>.
- [7] *Flux Documentation.* URL: <https://docs.influxdata.com/flux/v0/>.
- [8] *Get started with InfluxDB API client libraries / InfluxDB OSS v2 Documentation.* URL: <https://docs.influxdata.com/influxdb/v2/api-guide/tutorials/>.
- [9] *GitHub - ertis-research/opentwins: Innovative open-source platform that specializes in developing next-gen compositional digital twins.* URL: <https://github.com/ertis-research/opentwins>.
- [10] *GitHub - OpenTSDB/opensdb: A scalable, distributed Time Series Database.* URL: <https://github.com/OpenTSDB/opensdb>.
- [11] *Grafana Cloud / Observability platform overview.* URL: <https://grafana.com/products/cloud/>.
- [12] *InfluxDB / InfluxData.* URL: <https://www.influxdata.com/products/influxdb/>.
- [13] *InfluxDB schema design / InfluxDB OSS v2 Documentation.* URL: <https://docs.influxdata.com/influxdb/v2/write-data/best-practices/schema-design/>.

- [14] *Ingeniería de requisitos / Marco de Desarrollo de la Junta de Andalucía.* URL: <https://www.juntadeandalucia.es/servicios/madeja/contenido/subsistemas/ingenieria/ingenieria-requisitos>.
- [15] *Install Grafana on Debian or Ubuntu / Grafana documentation.* URL: <https://grafana.com/docs/grafana/latest/setup-grafana/installation/debian/>.
- [16] *Install InfluxDB / InfluxDB OSS v2 Documentation.* URL: <https://docs.influxdata.com/influxdb/v2/install/>.
- [17] *Introduction / Grafana documentation.* URL: <https://grafana.com/docs/grafana/latest/fundamentals/?pg=oss-graf&plcmt=hero-btn-2>.
- [18] *Overview/Prometheus.* URL: <https://prometheus.io/docs/introduction/overview/>.
- [19] *Renault Group's vehicles are now built by using a digital twin - Renault Group.* Jun. de 2022. URL: <https://www.renaultgroup.com/en/news-on-air/news/vehicle-digital-twin-when-physical-and-digital-models-unite/>.
- [20] Raquel Sánchez, Javier Troya y Javier Cámara. *Automated Planning for Adaptive Cyber-Physical Systems under Uncertainty in Temporal Availability Constraints.* Inf. téc. 2018. URL: <https://dl.acm.org/doi/10.1145/3643915.3644083>.
- [21] *Smart City Digital Twin by Snap4City / Snap4City.* URL: <https://www.snap4city.org/drupal/node/749>.
- [22] *Telegraf / InfluxData.* URL: <https://www.influxdata.com/time-series-platform/telegraf/>.
- [23] *Timescale documentation.* URL: <https://docs.timescale.com/home>.
- [24] *TimescaleDB vs. InfluxDB: Purpose-built for time-series data.* Ago. de 2018. URL: <https://www.timescale.com/blog/timescaledb-vs-influxdb-for-time-series-data-timescale-influx-sql-nosql-36489299877/>.
- [25] *TIPOS DE CONECTORES – Bilbao Movilidad Eléctrica.* URL: <https://electromovilidad.bilbao.eus/conceptos-basicos/tipos-de-conectores/>.
- [26] *Visualize data with the InfluxDB UI / InfluxDB OSS v2 Documentation.* URL: <https://docs.influxdata.com/influxdb/v2/visualize-data/>.

[27] *What is MongoDB? - MongoDB Manual v7.0.* URL: <https://www.mongodb.com/docs/manual/>.

Apéndice A

Manual de

Usuario

En este apéndice se detalla el manual de usuario que incluye las instrucciones de instalación y de uso del sistema.

A.1. Instalación

Para simplificar la instalación, se proporciona el sistema encapsulado en contenedores utilizando Docker. En esta versión, el sistema ejecuta el programa de python *dataFaker.py* para alimentar la base de datos con los datos de prueba y facilitar la realización de pruebas sobre el sistema.

A.1.1. Requisitos

Al utilizar Docker, el único requisito para poder ejecutar el sistema es tener instalado docker y docker-compose. Para sistemas basados en Linux con sistema de gestión de paquetes *apt*, la instalación se realizaría sencillamente con estos dos comandos:

```
sudo apt-get install docker.io  
sudo apt-get install docker-compose
```

A.1.2. Instalación y arranque del sistema

Una vez se tengan instalados los prerequisitos, hay que seguir una serie de sencillos pasos para arrancar y visualizar el sistema:

1. Descargar o clonar el [repositorio](#) con los archivos.
2. Ejecutar el comando de Docker en la terminal para arrancar el contenedor.
3. En el navegador, entrar en la url <https://localhost:3000> para acceder a la interfaz de Grafana.

El comando de Docker para arrancar el sistema es el siguiente.

```
sudo docker-compose up -d
```

Una vez haya terminado de arrancar los contenedores, se podrá acceder al sistema. Cuando se termine de utilizar el sistema y se quiera apagar, se debe escribir el siguiente comando en la consola.

```
sudo docker-compose down
```

Con esta instalación se logra una versión completamente funcional del sistema actual que utiliza los datos de prueba.

A.1.3. Alternativas

Si se desea personalizar el sistema para su integración en un entorno de producción real se debe reconfigurar la arquitectura del sistema para adaptarla a las nuevas necesidades del proyecto. Esto se puede lograr mediante una instalación limpia de InfluxDB [16] y de Grafana [15] siguiendo los manuales oficiales, creando un nuevo panel de control e importando el archivo JSON del proyecto que se encuentra en la ruta ‘./grafana/provisioning/dashboard-s/EVChargingMonitoring.json’ y adaptando las consultas para que se adapten a la nueva base de datos. Otra opción sería modificar los contenedores de Docker para eliminar la ejecución de *dataFaker.py* y añadir las imágenes que necesitara el nuevo sistema.

A.2. Manejo del software

En este apartado se detallan los pasos a seguir para la utilización correcta del sistema una vez esta arrancado como se ha explicado en el apartado anterior.

A.2.1. Entrar al sistema

Al tratarse de un sistema web, se puede acceder desde los principales navegadores del mercado. El primer paso es abrir el navegador elegido e ingresar la url del sistema. En caso de la instalación local, Grafana se ejecuta a través del puerto 3000 por defecto así que la url sería ‘<http://localhost:3000>’.

Una vez dentro del sistema, aparecerá la ventana de inicio de sesión (figura 39) que pedirá los credenciales del usuario (nombre de usuario/email y contraseña). Introducir las credenciales proporcionadas por el gestor y pulsar en iniciar sesión.

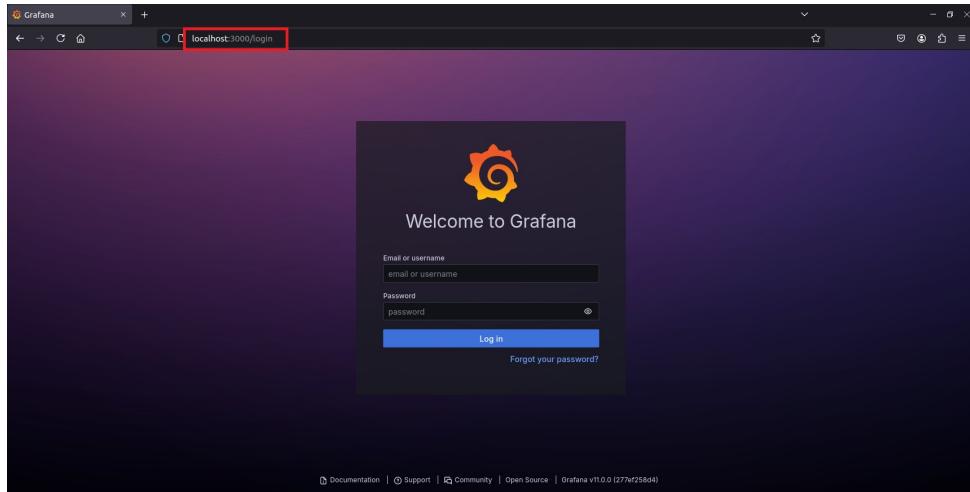


Figura 39: Pantalla de inicio de sesión

Nota: En el caso de ser el primer inicio de sesión del sistema (para el usuario Gestor), el usuario y contraseña por defecto son *admin/admin*. Acto seguido de la verificación de los credenciales, el sistema dará la opción de cambiar la contraseña del usuario.

A.2.2. Visualizar un panel de control

Para visualizar el panel de control, primero hay que acceder a la lista de todos los paneles. Desde el menú desplegable, se selecciona ‘Dashboards’ (figura 41). Esto dará acceso a la lista de los paneles de control disponibles (figura 40).

Para visualizar un panel de control concreto solo hay que seleccionar el panel deseado de la lista.

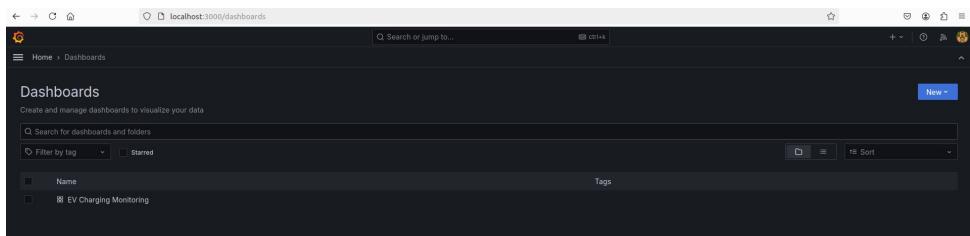


Figura 40: Lista de tableros disponibles

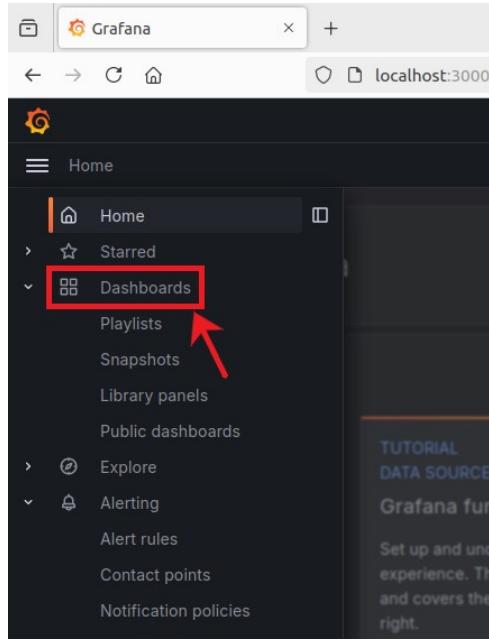


Figura 41: Acceso a la lista de tableros desde el menú desplegable

A.2.3. Visualizar métricas

Para visualizar el conjunto global de las métricas del tablero (panel de control), solo hay que entrar al tablero como se ha explicado en el paso anterior.

En caso de querer ampliar y focalizar la pantalla en una visualización concreta, es necesario seleccionar el botón de opciones de la visualización deseada y seleccionar la opción de Ver ('View') como se muestra en la figura 42.

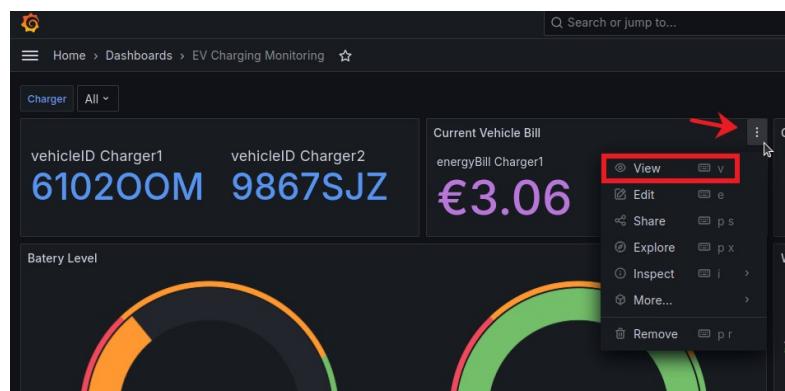


Figura 42: Selección vista focalizada en métrica

De esta forma, se accede a la visualización ampliada (a pantalla completa) de la métrica.

A.2.4. Filtrar los datos a mostrar

Para filtrar los datos a mostrar se debe utilizar el selector desplegable que se encuentra en la parte superior izquierda del tablero (figura 43) y seleccionar el o los cargadores que se deseen visualizar.

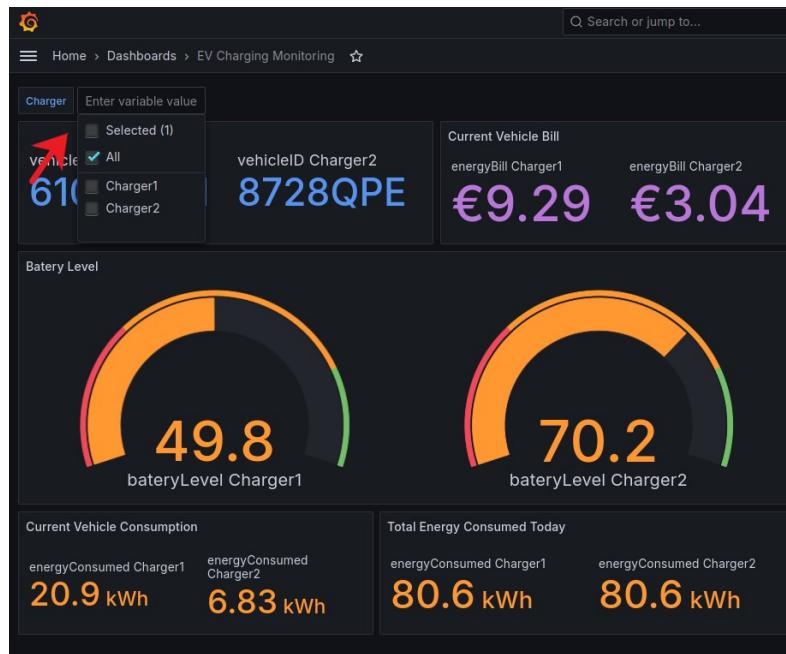


Figura 43: Selector desplegable para filtrar los cargadores

A.2.5. Cambiar temporización actualización

Otra característica que ofrece el software, es la actualización automática o manual de los datos del tablero. Esta funcionalidad se configura en la zona superior derecha del tablero. En el botón desplegable que se resalta en la figura 44, se puede configurar el temporizador para la actualización o apagarlo y dejarlo en manual. En caso de dejarlo manual, para actualizar los datos hay que pulsar el botón que se encuentra a la izquierda del temporizador resaltado en la imagen.

A.2.6. Visualizar todos los usuarios del sistema

Para acceder a la lista de todos los usuarios del sistema, hay que seleccionar la sección 'Administration - Users and Access - Users' del menú desplegable de la izquierda (figura 45).

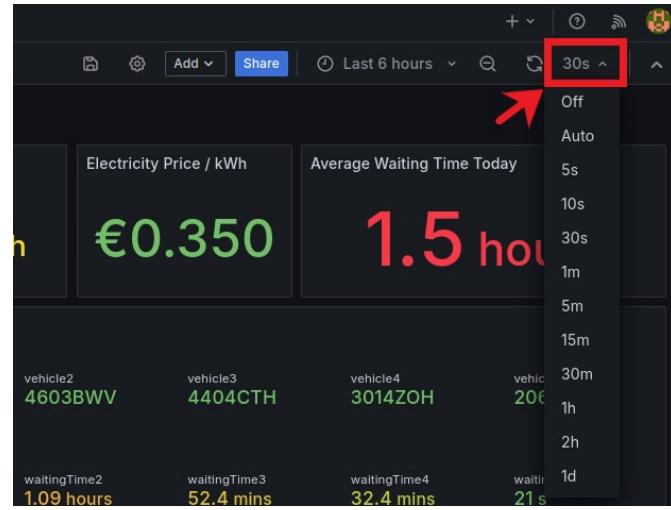


Figura 44: Temporizador de actualización de las vistas

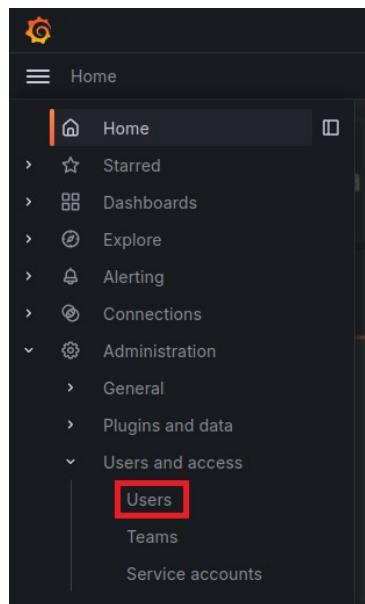


Figura 45: Acceso a la lista de usuarios desde el menú desplegable

Esto lleva al listado completo de usuarios del sistema (figura 46), desde el que se pueden añadir, modificar y eliminar usuarios.

A.2.7. Añadir usuarios

Para añadir usuarios nuevos, hay que pulsar en el botón azul 'New User' de la lista de usuarios que se muestra en la figura 46. Esto lleva a la pantalla mostrada en la figura 47 para crear un nuevo usuario dotándolo de nombre, dirección de correo electrónico, nombre de usuario

Users					
Manage users in Grafana					
All users	Organization users				
		Q. Search user by login, email, or name.	All users	Active last 30 days	New user
Login	Email	Name	Last active	Origin	
admin	admin@localhost		3 minutes		
rober	rnginformatica@uma.es	Roberto	Never		

Figura 46: Lista de usuarios del sistema

y contraseña. Inicialmente se le otorgan permisos de visualización solamente. Esto se podrá cambiar en el siguiente paso.

The screenshot shows the 'New user' creation interface. At the top, it says 'New user' and 'Create a new Grafana user.' Below are four input fields with labels: 'Name *' (containing 'Roberto'), 'Email' (containing 'rnginformatica@uma.es'), 'Username' (containing 'rober'), and 'Password *' (containing a redacted password). At the bottom is a blue 'Create user' button.

Figura 47: Creación de un nuevo usuario

A.2.8. Editar usuarios

Si se pincha en el botón que tiene forma de lápiz asociado a un usuario de la lista, se accede a la pantalla de edición del mismo (figura 48). Esta misma pantalla es la que aparece directamente cuando se crea un usuario nuevo. En esta pantalla es posible consultar y cambiar todos los datos y permisos del usuario, así como forzar un cerrado de sesión, inhabilitar o eliminar el usuario.

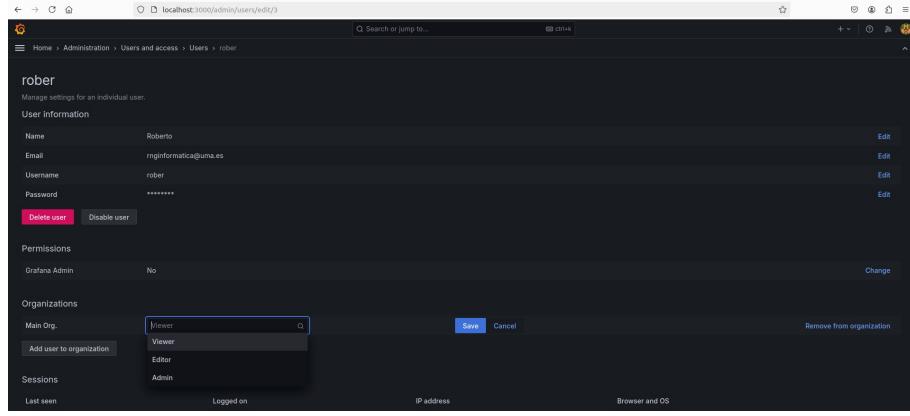


Figura 48: Modificar un usuario

A.2.9. Añadir nuevas fuentes de datos

Para añadir nuevas fuentes de datos, al igual que en los casos anteriores, hay que acceder al menú desplegable de la izquierda. En él, seleccionar ‘Connections - Add new Connection’ como indica la figura 50. Esto lleva a un listado de todos los plugins de conexiones a fuentes de datos disponibles (figura 49). Una vez localizado el plugin asociado a la fuente de datos necesaria, se selecciona y se accede a una página de vista previa del plugin que ofrece algo de información sobre el mismo. En la figura 51 se muestra la vista previa del plugin de PostgreSQL. Para añadir y comenzar a configurar esa nueva fuente de datos hay que pulsar el botón ‘Add new data source’ lo que llevará a una pantalla similar a la de la figura 11 (cada fuente de datos necesita una configuración distinta).

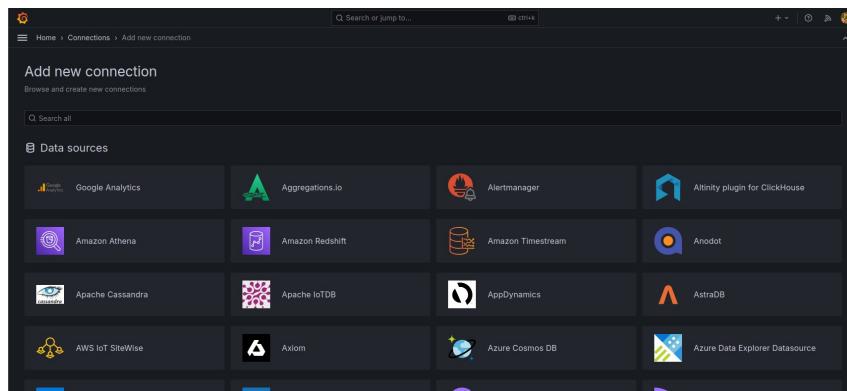


Figura 49: Lista de plugins para fuentes de datos

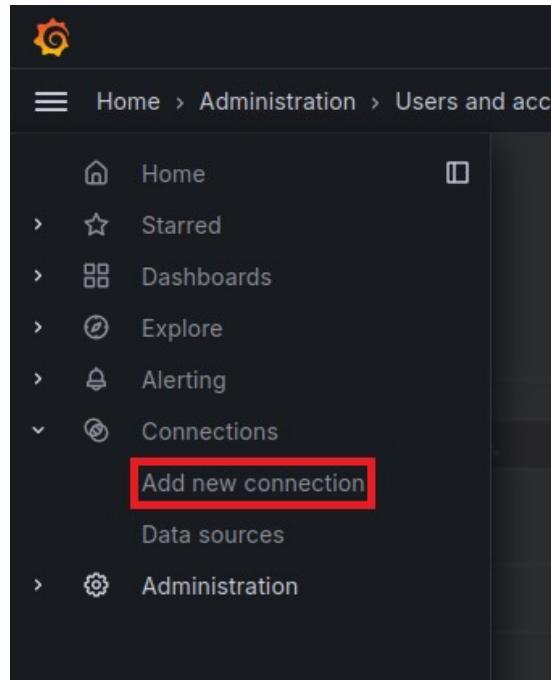


Figura 50: Acceso a la lista de conexiones de fuentes de datos

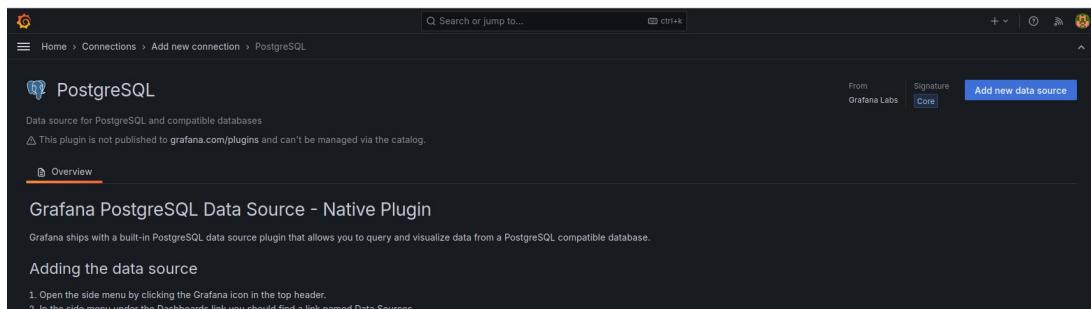


Figura 51: Vista previa del plugin para PostgreSQL

A.2.10. Modificar fuentes de datos

Para modificar una fuente de datos, primero hay que acceder al listado de fuentes de datos configuradas en el sistema. Este listado se obtiene seleccionando la opción ‘Data sources’ que se encuentra en el menú desplegable (justo debajo de la opción señalada en la figura 50). Esto lleva, tal como se muestra en la figura 52, al listado de fuentes de datos disponibles. Una vez localizada la que se quiere modificar, se selecciona y lleva de nuevo a una vista como la vista en la figura 11 en la que se pueden modificar toda la configuración de la conexión.

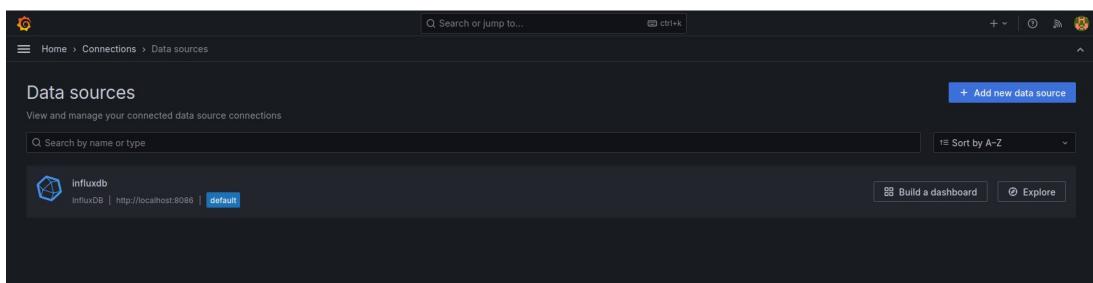


Figura 52: Listado de conexiones configuradas en Grafana

Apéndice B

Código Base

de Datos

En esta sección se muestra el código escrito en Python utilizado para simular los datos emitidos por una estación de carga de vehículos eléctricos a la base de datos InfluxDB.

Listing 19: Código en Python de la simulación

```
import os
import random
import time
import string

import influxdb_client
from influxdb_client.client.write_api import SYNCHRONOUS
from collections import deque
from dataclasses import dataclass

"""
Clase para crear los datos de los cargadores a enviar a InfluxDB.
Measure Key: location
Tags: name
Fields: vehicleID, energyConsumed, bateryLevel, maxPower, price
"""

@dataclass
class Charger:
    name: str          # Nombre/ID del cargador
    location: str      # Ubicacion del cargador
    vehicleID: str     # Matricula del vehiculo usuario
```

```

energyConsumed: float      # Energia consumida en la carga total en
                           kWh

bateryLevel: float          # Nivel de bateria del vehiculo (%)

maxPower: int                # Potencia maxima del cargador (kWh).

price: float                 # Precio por kWh.

energyBill: float            # Coste total de la carga

timestamp: int                # Marca temporal de la medicion (ns)

"""

Clase para crear los datos de la cola de espera a enviar a InfluxDB.

Measure Key: location
Tags: None
Fields: vehicle1, vehicle2, vehicle3, vehicle4, vehicle5
"""

@dataclass
class dataQueue:

    location: str               # Ubicacion del cargador

    vehicle1: str                # Matricula del vehiculo usuario 1

    vehicle2: str                # Matricula del vehiculo usuario 2

    vehicle3: str                # Matricula del vehiculo usuario 3

    vehicle4: str                # Matricula del vehiculo usuario 4

    vehicle5: str                # Matricula del vehiculo usuario 5

    waitingTime1: int             # Tiempo de espera del vehiculo 1

    waitingTime2: int             # Tiempo de espera del vehiculo 2

    waitingTime3: int             # Tiempo de espera del vehiculo 3

    waitingTime4: int             # Tiempo de espera del vehiculo 4

    waitingTime5: int             # Tiempo de espera del vehiculo 5

    timestamp: int                # Marca temporal de la medicion (ns)

"""

Clase para crear y manejar los datos de los vehiculos.

```

Genera un vehiculo con una matricula aleatoria , un nivel y capacidad de bateria aleatorio , la energia que gasta en relacion a la bateria y el coste de la carga.

Depende de un booleano (enter) para inicializar el vehiculo con datos (hay un vehiculo) o sin ellos (no hay un vehiculo).

```

"""
class Car:

    vehicleID: str
    bateryLevel: float
    bateryCapacity: int
    energyConsumed: float
    energyBill: float
"""

    Genera la matricula del vehiculo que entra a cargar
"""

def generateVehicleID(self):
    letras = ""
    for _ in range(3):
        letras += random.choice(string.ascii_uppercase)
    return str(random.randint(1000, 9999)) + letras

def __init__(self, enter):
    if (enter):
        self.vehicleID = self.generateVehicleID()
        self.bateryLevel = float(random.randint(0,99))
        self.bateryCapacity = random.randint(40,100)
        self.energyConsumed = 0.0
        self.energyBill = 0.0
    else:
        self.vehicleID = None
        self.bateryLevel = None
        self.bateryCapacity = None
        self.energyConsumed = None

```

```

        self.energyBill = None

# Configuración de InfluxDB
myToken = os.environ['DOCKER_INFLUXDB_INIT_ADMIN_TOKEN']
org = os.environ['DOCKER_INFLUXDB_INIT_ORG']
url = "http://influxdb:8086"
bucket = os.environ['DOCKER_INFLUXDB_INIT_BUCKET']

# Conexión a InfluxDB
client = influxdb_client.InfluxDBClient(url=url, token=myToken, org=org)
write_api = client.write_api(write_options=SYNCHRONOUS)

#####
# Variables globales:
maxPower = random.randint(20, 70)
maxCarsInQueue = 5
waitingQueue = deque([Car(False) for _ in range(maxCarsInQueue)],
                     maxCarsInQueue) # Cola de espera de coches
timeCounter = deque([None for _ in range(maxCarsInQueue)],
                     maxlen=maxCarsInQueue) # Contador de tiempo para cada coche en la
# cola en segundos

#####
"""

Genera los datos que se enviaran a la base de datos
inputs:
    car : Car
    chargerID : str
    price : float
output:
    data : Charger
"""

def generateData(car, chargerID, price):

```

```

timestamp = int(time.time()) * 1000000000 # Convertir a
nanosegundos

# Estructura del dato para InfluxDB
data = Charger(name = chargerID ,
                location ="stationA",
                vehicleID = car.vehicleID,
                energyConsumed = car.energyConsumed,
                bateryLevel = car.bateryLevel,
                maxPower = maxPower,
                price = price,
                energyBill = car.energyBill,
                timestamp = timestamp)

return data

"""
Genera los datos de la cola que se enviaran a la base de datos
inputs:
-
output:
    data : dataQueue
"""

def generateDataQueue():
    timestamp = int(time.time()) * 1000000000 # Convertir a
nanosegundos

# Estructura del dato para InfluxDB
data = dataQueue(location ="stationAQueue",
                  vehicle1 = waitingQueue[0].vehicleID,
                  vehicle2 = waitingQueue[1].vehicleID,
                  vehicle3 = waitingQueue[2].vehicleID,
                  vehicle4 = waitingQueue[3].vehicleID,
                  vehicle5 = waitingQueue[4].vehicleID,

```

```

        waitingTime1 = timeCounter[0] ,
        waitingTime2 = timeCounter[1] ,
        waitingTime3 = timeCounter[2] ,
        waitingTime4 = timeCounter[3] ,
        waitingTime5 = timeCounter[4] ,
        timestamp = timestamp)

    return data

```

"""

Calcula el nuevo porcentaje de bateria segun la potencia del cargador (maxPower). Calcula la velocidad de carga por segundo del cargador usando la potencia maxima de este , la carga actual de la bateria en kWh segun su porcentaje y realiza los calculos para sumar los kWh y devolver el coche con el nuevo porcentaje. Tambien actualiza la energia consumida por el coche y el coste de la carga .

inputs :

car : Car

price : float

output :

Car con el nuevo Porcentaje actual de bateria <= 100 y consumos .

"""

```

def calculateBateryIncrement(car, price):
    velocity = maxPower/3600 #kW por segundo
    actualKW = car.bateryCapacity * (car.bateryLevel/100) #kWh
    almacenados en la bateria actualmente
    incremented = actualKW + velocity
    car.energyConsumed += velocity
    car.energyBill += price * velocity
    if (incremented >= car.bateryCapacity):
        car.bateryLevel = float(100)
    else:

```

```

    car.bateryLevel = (incremented * 100) / car.bateryCapacity # Porcentaje actual
    return car

"""

Calcula el estado del coche en la siguiente iteracion. Si el coche esta cargando, se comprueba si ha llegado al 100%. En caso de estar al 100%, se elimina el coche, si no, se actualiza el porcentaje de bateria.

 : Car (estado actual)
price : float (precio del kWh)

 : Car (estado siguiente)

"""

def calculateCarState(car, price):
    waitingQueue
    timeCounter
    # Comprobar si hay un coche cargando
    if (car.vehicleID == None): #No hay coche
        # Preparar siguiente iteracion
        if (waitingQueue[0].vehicleID != None): #Si hay coches en la cola, entra el primero a cargar
            car = waitingQueue.popleft()
            waitingQueue.append(Car(False))
            timeCounter.popleft()
            timeCounter.append(None)
    else:
        #Actualizar los datos del coche y eliminarlo en caso de que llegue al 100%
        if (car.bateryLevel >= 100):
            car = Car(False)

```

```

    #No se actualiza el siguiente estado de isThereAcar aqui
para dejar que haya al menos una iteracion sin coche

else :
    car = calculateBateryIncrement(car , price)

return car

"""
Cuenta las plazas libres en la cola de espera
"""

def countFreeQueueSpaces () :
    count = 0
    for c in waitingQueue:
        if c . vehicleID == None:
            count += 1
    return count

def main () :
    waitingQueue
    timeCounter

    # Inicializar los coches iniciales para los cargadores
    car1 = Car(True)
    car2 = Car(True)

    nextCar = random . randint (5 , 15) #Tiempo en minutos que tarda el
siguiente coche en entrar
    nextCounter = 0 #Contador para saber cuando entra el siguiente
coche

    while True:
        price = round(random . uniform (0.15 , 0.75) ,2)
        data = generateData(car1 , "Charger1" , price) #generateData(
car1 )

```

```

data2 = generateData(car2, "Charger2", price) #generateData(
car2)

dataQueue = generateDataQueue()

# Escribir los datos en InfluxDB
write_api.write(bucket = bucket,
                record = data,
                record_measurement_key="location",
                record_time_key = "timestamp",
                record_tag_keys=[ "name" ],
                record_field_keys=[ "vehicleID" , "energyConsumed" , "bateryLevel" , "maxPower" , "price" , "energyBill" ])
                write_api.write(bucket = bucket,
                record = data2,
                record_measurement_key="location",
                record_time_key = "timestamp",
                record_tag_keys=[ "name" ],
                record_field_keys=[ "vehicleID" , "energyConsumed" , "bateryLevel" , "maxPower" , "price" , "energyBill" ])
#Enviar los Datos de la cola (dataQueue) a InfluxDB
write_api.write(bucket = bucket,
                record = dataQueue,
                record_measurement_key="location",
                record_time_key = "timestamp",
                record_tag_keys=[],
                record_field_keys=[ "vehicle1" , "vehicle2" , "vehicle3" , "vehicle4" , "vehicle5" , "waitingTime1" , "waitingTime2" , "waitingTime3" , "waitingTime4" , "waitingTime5" ])

#Actualizar el estado del coche
car1 = calculateCarState(car1, price)

```

```

car2 = calculateCarState(car2, price)

#Actualizar el estado de la cola
freeSpaces = countFreeQueueSpaces()
nextCounter = nextCounter + 1
for i in range(maxCarsInQueue-freeSpaces):
    timeCounter[i] += 1

if(nextCounter == nextCar*60): #El contador ha llegado al
tiempo de llegada
    if(waitingQueue[maxCarsInQueue-1].vehicleID == None): #
Comprobar si hay una posición vacía en la cola
        #Añadir un coche a la cola en la primera posición
        vacía
        waitingQueue[maxCarsInQueue - freeSpaces] = Car(True)
        timeCounter[maxCarsInQueue - freeSpaces] = 0
        #Nuevo tiempo de llegada para el siguiente coche
        nextCar = random.randint(5, 15)
        nextCounter = 0

# Esperar un segundo antes de generar el siguiente dato
time.sleep(1)

if __name__ == "__main__":
    main()

```

Listing 19: Código en Python de la simulación



UNIVERSIDAD
DE MÁLAGA | **uma.es**

E.T.S. DE INGENIERÍA INFORMÁTICA

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga