TRABAJO FIN DE GRADO

INGENIERÍA INFORMÁTICA

Desarrollo de un Sistema Integral para la Monitorización Inteligente de Smart Villages

Autor
Carlos Moreno Villarrubia

Directores
María Bermúdez Edo
Daniel Bolaños Martínez

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, septiembre de 2024

Desarrollo de un Sistema Integral para la Monitorización Inteligente de Smart Villages

Autor
Carlos Moreno Villarrubia

Directores
María Bermúdez Edo
Daniel Bolaños Martínez

Desarrollo de un Sistema Integral para la Monitorización Inteligente de Smart Villages

Carlos Moreno Villarrubia

Palabras clave: Cuadro de mandos, Base de datos, ETL, Orquestación de contenedores, Kubernetes, Software Libre, Monitorización

Resumen

Las smart cities utilizan tecnologías de la información y comunicación para mejorar la calidad de vida de sus habitantes. Un componente esencial de estas ciudades inteligentes es la capacidad de capturar y procesar grandes volúmenes de datos provenientes de diversas fuentes, como sensores, dispositivos IoT (Internet of Things), sistemas de transporte, servicios públicos y redes sociales. Estos datos se utilizan para monitorear y gestionar los recursos urbanos de manera eficiente.

Para llevar a cabo esta monitorización, se emplean dashboards. Los cuadros de mando, paneles de control o dashboards son herramientas fundamentales para la gestión y supervisión de distintos aspectos de las smart cities. Estos paneles ofrecen visualizaciones de datos en tiempo real, lo que facilita a los responsables de la toma de decisiones la gestión de recursos, la optimización de servicios y la mejora de la calidad de vida de los ciudadanos.

En este trabajo se presenta el análisis, diseño e implementación de una solución integral para la monitorización de estos datos, cumpliendo con los requisitos de utilizar software libre, asegurar que el sistema esté contenerizado para garantizar su desacoplamiento, y emplear un orquestador de contenedores que asegure una alta disponibilidad.

Development of a Integrated System for Intelligent Monitoring of Smart Villages

Carlos Moreno Villarrubia

Keywords: Dashboard, Database, ETL, Container Orchestration, Kubernetes, Open Source Software, Monitoring

Abstract

Smart cities use information and communication technologies to improve the quality of life of their inhabitants. An essential component of these intelligent cities is the ability to capture and process large volumes of data from various sources, such as sensors, Internet of Things (IoT) devices, transportation systems, public services, and social networks. This data is used to monitor and manage urban resources efficiently.

To carry out this monitoring, dashboards are employed. Dashboards, also known as control panels or command boards, are essential tools for the management and supervision of different aspects of smart cities. These panels provide real-time data visualizations, which help decision-makers manage resources, optimize services, and improve the quality of life for citizens.

This work presents the analysis, design, and implementation of a comprehensive solution for monitoring this data, meeting the requirements of using open-source software, ensuring that the system is containerized to guarantee decoupling, and employing a container orchestrator to ensure high availability.

Dña. **María Bermúdez Edo**, Profesor del Departamento Lenguajes y Sistemas Informáticos de la Universidad de Granada.

D. **Daniel Bolaños Martínez**, Profesor del Departamento Lenguajes y Sistemas Informáticos de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado ***Desarrollo de un Sistema Integral para la Monitorización Inteligente de Smart Villages***, ha sido realizado bajo su supervisión por **Carlos Moreno Villarrubia**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 6 de septiembre de 2024 .

Los directores:

María Bermúdez Edo **Daniel Bolaños Martínez**

Agradecimientos

Quisiera expresar mi más profundo agradecimiento a María y Daniel, mis tutores de este Trabajo de Fin de Grado, por su constante apoyo y por estar siempre disponibles cuando los he necesitado.

También deseo agradecer a mi familia por su incondicional apoyo y ánimo a lo largo de todo este proceso. Su presencia y aliento han sido fundamentales para alcanzar este logro.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

Este Trabajo de Fin de Grado (TFG) tiene como objetivo investigar herramientas y tecnologías de monitarización y gestión de información, de software libre, y aportar una solución que permita de una manera integral monitorizar y mostrar de manera visual para su análisis los datos de un proyecto en el ámbito de la monitorización de Smart Villages para hacer un seguimiento de su estado. Se utilizan los datos y la infraestructura del proyecto de investigación Smart Poqueira, que se centró en la implementación de tecnologías digitales para la gestión sostenible de la actividad turística en zonas ambientalmente sensibles.

En el proyecto Smart Poqueira, se instalaron sensores en diversas áreas del barranco de Poqueira para recopilar datos para su análisis. En este TFG, utilizaremos los datos obtenidos de sensores de tráfico vehicular y aforo en algunas zonas de Pampaneira, Bubión y Capileira, específicamente de cuatro cámaras LPR (de reconocimiento de matrículas, en inglés License-Plate Recognition). Estas cámaras detectan el paso de vehículos, registrando la hora, la matrícula y la dirección de la marcha, y están situadas en diferentes puntos de la carretera, como se puede ver en la Figura 1.1.

Figura 1.1: Localización de las cámaras de reconocimiento de matrículas con un símbolo de cámara y la codificación que recibe (1 o 0) en función del sentido de la marcha del vehículo, según las flechas.

Además, se utilizan sensores de aforo que consisten en una cámara que, mediante reconocimiento visual, controla el flujo de personas entre zonas, registrando cuántas entran o salen cada hora.

En el proyecto, debido a que se recogen las matrículas de vehículos y se proporcionan códigos postales asociados a algunas de ellas, es necesario asegurarse de que esta información esté protegida. Así, se cumple la Ley de Protección de Datos cifrando los discos donde se almacenan las matrículas y códigos postales.

1.1. Motivación

La motivación de este proyecto surge de la necesidad del equipo de investigación de visualizar conjuntos de datos de manera sencilla y eficiente. Además, se busca la carga automática de estos datos y cumplir con los requisitos de disponibilidad y uso de nuevas tecnologías no vistas en el grado.

1.2. Conocimientos previos

En esta memoria se han aplicado conocimientos adquiridos en diversas asignaturas clave del currículo académico. Las más destacadas son las siguientes:

- **Fundamentos de Programación y Metodología de la Programación:** Se han aplicado conceptos fundamentales sobre la programación de código en el desarrollo del ETL (extracción, transformación y carga de datos, en inglés). Estos conocimientos han sido esenciales para escribir y optimizar el código necesario para la extracción, transformación y carga de datos.
- **Centros de Procesamiento de Datos:** Se ha utilizado la metodología aprendida en esta asignatura para implementar el primer planteamiento del sistema, utilizando Grafana e InfluxDB containerizados mediante Vagrant, según lo visto en el curso. Esta base ha permitido la correcta visualización y gestión de los datos procesados.
- **Dirección y Gestión de Proyectos y Metodologías de Desarrollo Ágil:** Los conocimientos adquiridos sobre metodologías ágiles y SCRUM han sido aplicados para organizar y gestionar el proyecto de manera efectiva. Esto ha asegurado un desarrollo ordenado y adaptativo, permitiendo ajustes rápidos y eficientes a medida que avanzaba el proyecto.
- **Aprendizaje Automático:** Se ha utilizado el lenguaje Python, que es parte integral de esta asignatura, para diversas tareas de programación y análisis de datos. Python ha sido una herramienta fundamental para implementar algoritmos y procesos automáticos dentro del proyecto.

1.3. Objetivos

El objetivo principal consiste en aportar una solución, para recopilar, visualizar y analizar los datos de una smart village.

Para lograr este objetivo general se ha desglosado en varios objetivos:

- **OB1:** Configurar una solución de orquestación de contenedores para permitir el uso de componentes de manera contenerizada.
- **OB2:** Diseñar y desarrollar una solución ETL contenerizada para extraer los datos y cargarlos, facilitando su posterior visualización.

- **OB3:** Configurar un servicio que permita la visualización de paneles informativos y la información recopilada para su análisis.
- **OB4:** Diseñar e implementar paneles informativos para la visualización de los datos recopilados, con el fin de apoyar la toma de decisiones.
- **OB5:** Configurar una base de datos para almacenar la información recogida por el proceso ETL.

Además, se han establecido los siguientes objetivos de aprendizaje:

- **OA1:** Comprender y aplicar los conceptos fundamentales de la tecnología de contenedores utilizando Docker.
- **OA2:** Diseñar y desplegar un servicio basado en contenedores que permita la visualización de gráficos en paneles informativos.
- **OA3:** Evaluar y comparar diversas soluciones para la gestión de bases de datos, y seleccionar la más adecuada.
- **OA4:** Implementar Kubernetes para mejorar la disponibilidad y escalabilidad del servicio.
- **OA5:** Configurar mecanismos de extracción y tratamiento automático de datos (ETL pipeline) para asegurar su correcta visualización en los paneles informativos.
- **OA6:** Desplegar y configurar un sistema de control de acceso basado en permisos, asegurando que ciertos paneles sean accesibles únicamente para usuarios específicos.
- **OA7:** Desarrollar habilidades en la configuración y administración de Kubernetes, incluyendo la creación de despliegues para aprovechar las mejoras de disponibilidad que ofrece la herramienta.
- **OA8:** Adquirir experiencia práctica en la implementación de soluciones de alta disponibilidad y resiliencia en entornos de producción.
- **OA9:** Encontrar alternativas que sean abiertas y libres.
- **OA10:** Usar metodologías ágiles que permitan un desarrollo flexible y adaptativo.

En el siguiente enlace esta disponible el código del proyecto.

1.4. Presupuesto del Proyecto

En la siguiente tabla se muestra el presupuesto para la realización del proyecto.

Concepto	**Costo unitario**	**Costo total**
Ordenador (Equipo 1)	3000 €	60.15 €
Software	0 €	0 €
Informático (4 meses)	1500 €/mes	6000 €
Coste de luz	0.14 €/kWh	12.67 €
Total		6072.82 €

Cuadro 1.1: Presupuesto del proyecto

A continuación se desglosa el presupuesto mostrado en la tabla 1.1. El costo unitario se refiere al precio de una unidad del recurso (por ejemplo, un equipo o una licencia de software) o al costo por unidad de tiempo (por ejemplo, el salario mensual de un trabajador). El costo total incluye no solo el costo unitario, sino también factores adicionales como el porcentaje de utilización y los costos energéticos, que se detallan a continuación.

Equipamiento utilizado

Para calcular el uso del equipo informático, se han considerado varios factores: el costo del equipo, la fecha de adquisición, la vida útil, el porcentaje de uso y el tiempo de utilización:

- **Nombre del equipo**: Equipo 1
- **CPU**: Intel Core i9 13ª edición
- **RAM**: 4 x Corsair Dominator 16GB (64GB)
- **Potencia eléctrica**: 250 W
- **Fecha de adquisición**: 19/08/2022
- **Valor inicial**: 3000 €
- **Vida útil**: 5 años
- **Valor residual**: 3000 € / 5 años = 600 €
- **Depreciación**: (3000 € - 428.57 €) / 5 años = 480 €/año
- **Tiempo utilizado**: 4 meses = 0.33 años

- **Porcentaje de uso**: 50 %

Teniendo en cuenta estos parámetros, podemos calcular el valor residual del equipo (el valor mínimo del activo al final de su vida útil) como 600 €, y su depreciación anual es de 480 €/año. Hemos utilizado el equipo durante 4 meses con un porcentaje de uso del 50 %, por lo que el costo de uso del equipo en este proyecto es:

$$\begin{aligned}\text{Depreciación anual} \times \text{Tiempo utilizado} \times \text{Porcentaje de uso} \\ = 480\,€/\text{año} \times 0.33\,\text{años} \times 0.5 \\ = 79.20\,€\end{aligned}$$

También cabe destacar que, gracias a la utilización de software libre, el costo de este es de 0 €.

Costes Operacionales

Los costos operacionales son aquellos derivados del funcionamiento del negocio, como el uso del equipo, desplazamientos, etc. En este caso, se ha considerado el costo de la electricidad para el funcionamiento del equipo durante el desarrollo del proyecto.

- **Potencia eléctrica del equipo**: 250 W
- **Costo de la tarifa de luz actual**: 0.14 €/kWh

Podemos calcular el costo de una hora de uso de electricidad del equipo como:

$$\begin{aligned}\text{Potencia eléctrica} \times 1\ \text{hora} \times \text{Costo tarifa de luz} \\ = 250\,\text{W} \times 1\,\text{hora} \times 0.14\,€/\text{kWh} \\ = 0.035\,€\end{aligned}$$

En un proyecto de 4 meses, con 362 horas estimadas de uso, tendríamos un costo de luz de:

$$0.035\,€/\text{hora} \times 362\,\text{horas} = 12.67\,€$$

1.5. Planificación temporal

A continuación, de forma general, se especifica el transcurso de los sprints.

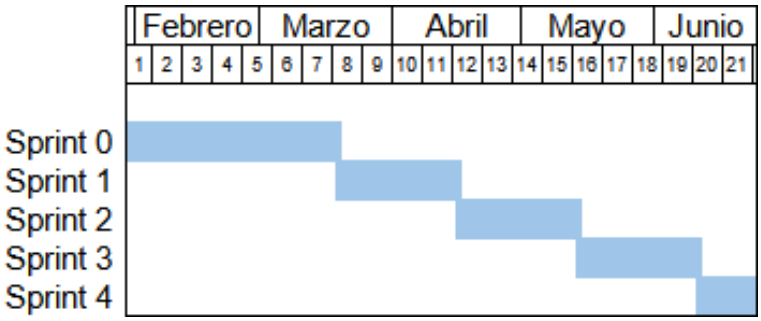

Figura 1.2: Diagrama de Gantt general donde se observa la organización temporal de los sprints por semanas del proyecto. En el capítulo 5, Diseño y Desarrollo, se desglosará esta figura.

En la figura 1.2 se observa un diagrama general del transcurso de los sprints a lo largo de las semanas. Este transcurso se desglosará más adelante en el capítulo 5, Diseño y Desarrollo.

Es importante destacar que, debido al uso de nuevas tecnologías, solo fue posible realizar una planificación detallada al finalizar el sprint 0.

1.6. Estructura de la memoria

La estructura de la memoria se organiza en varios capítulos clave que abordan diferentes aspectos del proyecto:

- **Marco teórico:** En este capítulo se aborda la importancia de los datos para las ciudades inteligentes y se describe el proceso de creación de dashboards.
- **Estado del arte:** En este capítulo se presenta cómo distintas entidades han desarrollado sus dashboards, poniendo de manifiesto el estado actual de la industria.
- **Tecnologías:** En este capítulo se describen en profundidad las tecnologías utilizadas en el proyecto y aquellas cuyo uso fue considerado o implementado, pero finalmente descartado, incluyendo los motivos de dichas decisiones.

- **Diseño y desarrollo:** Este capítulo detalla la planificación del desarrollo y analiza el transcurso de los sprints, proponiendo soluciones al problema planteado.
- **Conclusiones y trabajo futuro:** En este capítulo se presentan las conclusiones y se analiza el posible desarrollo y la explotación económica que podrían derivarse del proyecto.

Capítulo 2

Marco Teórico

En el contexto de las smart cities, la recopilación, análisis y visualización efectiva de datos son cruciales para la toma de decisiones informada. Este capítulo se centra en la teoría sobre dashboards y la captación de datos en el entorno de las smart cities.

2.1. Smart Cities y la Captación de Datos

Las smart cities utilizan tecnologías de la información y comunicación (TIC) para mejorar la calidad de vida de sus habitantes. Un componente esencial de las smart cities es la capacidad de capturar grandes volúmenes de datos de diversas fuentes, como sensores, dispositivos IoT (Internet of Things), sistemas de transporte, servicios públicos, y redes sociales. Estos datos se utilizan para monitorear y gestionar recursos urbanos de manera eficiente.

Fuentes de Datos en Smart Cities

En el contexto de Smart Cities, las fuentes de datos incluyen sensores y dispositivos IoT que capturan información en tiempo real sobre el tráfico, la calidad del aire y el consumo de energía; sistemas de transporte que proporcionan datos sobre el flujo de tráfico, horarios de transporte público y disponibilidad de estacionamientos; servicios públicos que recopilan información sobre el consumo de agua y electricidad, gestión de residuos y servicios de emergencia; y redes sociales que ofrecen datos sobre la percepción ciudadana y eventos en tiempo real. Este trabajo se centra en datos de vehículos y de aforo.

2.2. Dashboards para la Visualización de Datos

Un dashboard es una herramienta esencial para la visualización de datos, que permite mostrar información de manera clara y concisa. En el contexto de las smart cities, los dashboards se utilizan para presentar datos relevantes de forma accesible para los tomadores de decisiones. Convertir los datos en un activo valioso implica hacerlos accesibles y comprensibles a través de un dashboard. La simplicidad y la precisión son claves: un dashboard con la cantidad adecuada de información, disponible en el momento preciso, puede empoderar a las organizaciones de maneras inimaginables. Cuando se usan correctamente, los dashboards permiten tomar decisiones informadas, identificando aciertos, áreas de mejora y tendencias.

Para construir un dashboard efectivo, es fundamental seguir un proceso estructurado. El primer paso es la recolección de requisitos, donde se deben hacer las preguntas correctas a los usuarios y stakeholders para comprender sus necesidades y expectativas. Esto establece una base sólida para todo el proyecto. Es crucial determinar si la solicitud es para una extracción/análisis de datos puntual o un proyecto de visualización que requiere actualizaciones periódicas. Identificar quiénes serán los usuarios del dashboard y entender los problemas que intentan resolver o las hipótesis que desean probar es esencial para personalizar el dashboard según el tipo de usuario. También es importante considerar en qué plataformas se accederá al dashboard, dado que muchos usuarios prefieren dispositivos móviles o tablets para consultar los datos.

Una vez recopilados los requisitos, es momento de diseñar el producto final. Esta etapa incluye la creación de wireframes y mockups para visualizar cómo se presentará el dashboard. Esta fase puede implicar varias sesiones de brainstorming con el equipo y los stakeholders para decidir qué gráficos utilizar, dónde ubicarlos y cómo estructurar las páginas. Adicionalmente, es fundamental documentar los requisitos técnicos del dashboard, lo que incluye los cálculos de los KPIs (Indicadores Clave de Desempeño, métricas utilizadas para medir y evaluar la efectividad de las acciones en función de los objetivos establecidos), las suposiciones de datos y las fuentes de datos. La planificación de la gestión del proyecto es esencial, asignando tareas específicas, estableciendo un plan de trabajo y definiendo cronogramas.

La etapa de extracción, transformación y carga de datos (ETL) implica reunir todos los conjuntos de datos necesarios y transformarlos en una forma que sea utilizable para el dashboard. Esto incluye la preprocesación de datos, como la limpieza, codificación y generación de características. Es una buena práctica realizar la mayor cantidad de cálculos y transformaciones durante el ETL en lugar de en la herramienta de visualización para asegurar que el dashboard sea eficiente y fácil de actualizar.

Con los datos preparados, se procede a construir el dashboard conforme al wireframe aprobado. Es crucial adherirse a las guías de estilo de la organización para asegurar la consistencia y la familiaridad del usuario con el producto final. Si el dashboard requiere actualizaciones regulares, el proceso debe estar completamente automatizado para evitar intervenciones manuales. Se debe lanzar una versión beta (MVP) para recibir feedback y realizar iteraciones según sea necesario.

Antes de lanzar el dashboard, es vital realizar pruebas y control de calidad. Esto incluye verificar que los datos sean correctos y coincidan con los números de la base de datos, probar la funcionalidad de características como filtros y botones de navegación, y asegurar que el diseño cumpla con los estándares de la organización. Además, es importante documentar el proceso de construcción del dashboard, incluyendo un manual de usuario, FAQs, y detalles técnicos relevantes para facilitar su uso y mantenimiento.

Finalmente, se lanza el dashboard y se promueve su adopción entre los usuarios. Es crucial comunicar a los usuarios que el dashboard se construirá de manera iterativa, solicitando feedback continuo para mejorar el producto. Se pueden realizar sesiones de tutoría o 1:1 con los usuarios para asegurar que comprendan cómo usar el dashboard y aprovechar al máximo sus funcionalidades. Monitorear el uso del dashboard y identificar cuáles se vuelven menos populares es una estrategia efectiva para evitar que queden obsoletos y asegurar que continúen proporcionando valor a la organización.

En resumen, un dashboard bien diseñado es una herramienta poderosa para la visualización de datos en smart cities, facilitando la toma de decisiones informadas y optimizando diversos aspectos de la gestión urbana.

Capítulo 3

Estado del Arte

Los cuadros de mando, paneles de control, paneles informativos o dashboards son herramientas esenciales para la gestión y monitoreo de diversos aspectos de las ciudades inteligentes. Estos paneles proporcionan visualizaciones de datos en tiempo real que ayudan a los responsables de la toma de decisiones a gestionar recursos, optimizar servicios y mejorar la calidad de vida de los ciudadanos. A continuación, se presentan algunos casos de uso de cuadros de mando implementados por diversas empresas en distintas ciudades.

City Dashboard - Empresa: IBM

IBM ha desarrollado un cuadro de mando llamado City Dashboard que se utiliza en la ciudad de Rio de Janeiro, Brasil (Urban and Regional Innovation Research [URENIO], 2015). Este panel integra datos de múltiples fuentes, incluyendo sistemas de transporte, servicios de emergencia y monitoreo climático, proporcionando una visión integral de la operación de la ciudad. Esta solución permite a las autoridades gestionar mejor las respuestas a emergencias y optimizar los servicios públicos(Figura 3.1).

Intelligent Operations Center - Empresa: Huawei

Huawei ha implementado su Intelligent Operations Center en el distrito de Longgang, Shenzhen, China (Huawei, 2024a). Este panel de mando proporciona una plataforma integrada para la gestión de servicios públicos, seguridad ciudadana y monitoreo ambiental. La solución de Huawei facilita la coordinación entre diferentes departamentos municipales y mejora la capacidad de respuesta a incidentes(Huawei, 2024b).

Figura 3.1: Vista del panel de mando de Río de Janeiro (vineetadurani, 2011)

Figura 3.2: Vista del panel de mando de Longgang, Shenzhen, China. (Absen, 2017)

OpenDataSoft Dashboard - Empresa: OpenDataSoft

La alcaldía de París ha desarrollado dashboards personalizados utilizando OpenDataSoft, en colaboración con la comunidad de Open Data de diversas ciudades, incluyendo París, Francia (Ayuntamiento de París, 2024b).

Estos paneles permiten la visualización de datos abiertos relacionados con el transporte público, la calidad del aire y otros indicadores urbanos. Gracias a la plataforma de OpenDataSoft, tanto las autoridades como los ciudadanos pueden acceder de manera transparente y en tiempo real a información relevante sobre la ciudad.

Figura 3.3: Vista del panel de la ciudad de París para puntos de carga para vehículos eléctricos (Ayuntamiento de París, 2024a)

Km4City Smart City Control Room: Dashboard Systems - Empresa: Universidad de Florencia

El Laboratorio DISIT de la Universidad de Florencia ha desarrollado una plataforma llamada Km4City que incluye cuadros de mando para mantener bajo control los servicios y el estado de la ciudad mediante un cuadro de mandos completo y flexible. Se han hecho aplicaciciones utilizando los servicios y APIs de Km4City (Nesi, 2023). Tiene como objetivo proporcionar información actualizada de la ciudad. En Km4City las empresas e instituciones pueden integrar información abierta, privada, sensible y/o crítica de un modo contextualizado, con aquella accesible para la ciudad y crear nuevos servicios para su personal cualificado y/o para los ciudadanos. Es posible desarrollar una aplicaciones y páginas web que usen estos servicios de una manera simple y rápida. La principal herramienta de la web está accesible en http://servicemap.disit.org. Es posible acceder a cerca de 100,000 servicios en Florencia, Pisa, Prato, Pistoia, Arezzo, ciudad de Empoli y toda la

Toscana.

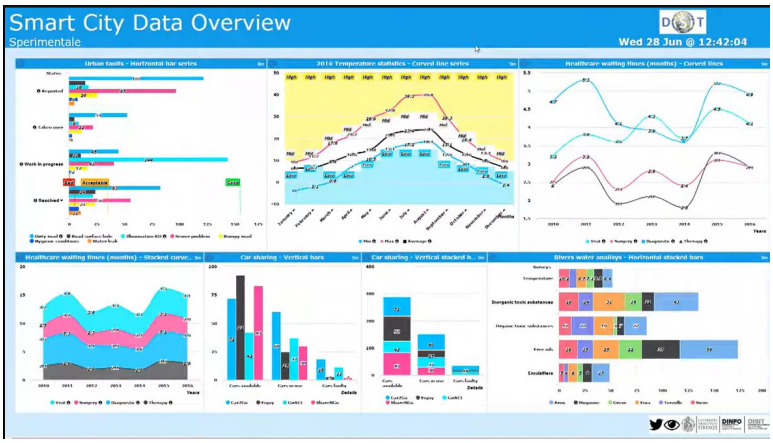

Figura 3.4: Vista del panel en la plataforma km4city (Distributed Systems and Internet Technologies LAB [DISIT LAB], 2024)

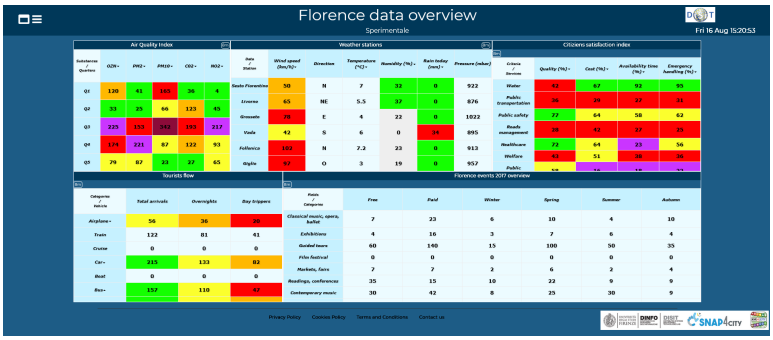

Figura 3.5: Vista del panel de la plataforma km4city (Distributed Systems and Internet Technologies LAB [DISIT LAB], 2024)

Cuadro de mandos de Paderborn - Empresa: EDAG

En cooperación con la ciudad alemana de Paderborn y su plataforma de datos ya existente, han desarrollado un Smart City Dashboard, que visualiza los datos de la Plataforma de Datos Urbanos para el público (EDAG, 2024). Al poner los datos a disposición del público, especialmente en forma de gráficos, los ciudadanos pueden hacerse una idea general de la situación actual de la ciudad. Por ejemplo, pueden ver cuántos visitantes hay actualmente en la piscina o qué valores de temperatura y humedad miden los sensores en distintas partes de la ciudad. Los operadores pueden adaptar qué datos se visualizan y cómo, en modo de funcionamiento en directo y sin esfuerzo adicional de programación. Para ello, los operadores se conectan como ad-

ministradores y utilizan un asistente integrado para realizar la configuración deseada.

Figura 3.6: Vista del panel de Paderborn (EDAG, 2024)

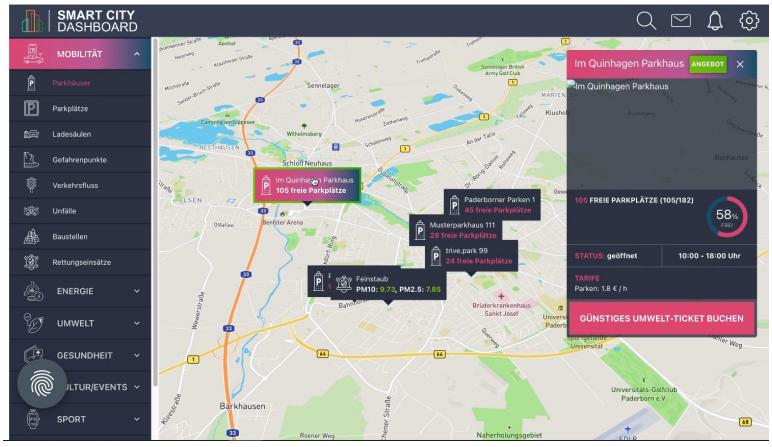

Figura 3.7: Vista del panel de Paderborn km4city (EDAG, 2024)

Capítulo 4

Tecnologías

En este apartado se describen las tecnologías que pueden ser utilizadas en el proyecto y la elección que hemos realizado basándonos en los requisitos de este trabajo.

4.1. Discusión Sobre la Elección de la Base de Datos

En el contexto de las smart cities, la recopilación, análisis y visualización efectiva de datos son cruciales para la toma de decisiones informada. Este capítulo se centra en la justificación de la elección de TimescaleDB con Apache Superset para nuestro proyecto, tras una evaluación comparativa de diversas bases de datos.

Con el fin de elegir la base de datos más adecuada para nuestro caso, realizamos una evaluación comparativa de varias bases de datos, considerando aspectos como el modelo de datos, la popularidad, la capacidad para manejar consultas complejas y la gestión de series temporales. Además, mostramos una marcada preferencia por el software libre debido a que es un requisito del cliente. A continuación, se presenta un resumen de nuestras evaluaciones sobre las bases de datos más populares encontradas en el ranking de DB-Engines "DB-Engines Ranking", 2024 sobre bases de datos más populares y la justificación final de nuestra elección. La siguiente evaluación se presenta según el orden en el que aparecen las bases de datos en el ranking por popularidad.

4.1.1. Evaluación Comparativa de Bases de Datos

Oracle

Aunque Oracle es uno de los lideres en la categoría de modelos relacionales, las consideraciones de costo y la preferencia por el software libre nos llevaron a descartarlo.

MySQL

Aunque MySQL es ampliamente utilizado y soporta tanto modelos relacionales como multi-modelo, buscábamos características específicas de series temporales que no ofrece de forma nativa. Además, nuestra prioridad por el software libre y la capacidad de integración con series temporales nos llevó a considerar otras alternativas.

Microsoft SQL Server

Microsoft SQL Server también es muy popular y soporta modelos relacionales y multi-modelo. Sin embargo, al igual que MySQL, no proporciona soporte nativo para series temporales. Además, el costo y la falta de una licencia de software libre fueron factores decisivos para descartarlo.

MariaDB

MariaDB, siendo una bifurcación de MySQL, mantiene muchas de las funcionalidades y ventajas de MySQL, pero con un mayor compromiso con el software libre. A pesar de esto, buscamos una solución con una integración más robusta y nativa para series temporales.

OpenSearch

Ante la modificación de la licencia de Elasticsearch (Elastic, 2021), la cual se volvió más restrictiva, Amazon desarrolló OpenSearch como una alternativa de código abierto. OpenSearch emergió como una solución prometedora para las necesidades de búsqueda y análisis de datos en tiempo real, ofreciendo funcionalidades similares a Elasticsearch, pero bajo una licencia de código abierto más permisiva *OpenSearch: A Distributed, RESTful Search and Analytics Suite*, 2024. Esta opción inicialmente atrajo la atención como una posible solución para superar los desafíos de licenciamiento presentados por Elasticsearch, al mismo tiempo que mantenía capacidades robustas de análisis y búsqueda.

A pesar del potencial de OpenSearch y su alineación con nuestros principios de uso de software de código libre, la decisión final se inclinó hacia TimescaleDB. Esta decisión se fundamentó en la elección de Apache Superset como nuestra herramienta principal para la visualización y análisis de datos. Aunque OpenSearch presentaba características atractivas, la integración y compatibilidad con Apache Superset no eran tan directas como las ofrecidas por soluciones basadas en SQL.

MongoDB

Como una de las bases de datos más utilizadas para almacenar documentos, MongoDB destaca gracias a su estructura basada en documentos BSON (Binary JSON), que facilita un modelo de datos flexible y dinámico. Esta característica hace de MongoDB una solución atractiva para aplicaciones que requieren la gestión de grandes volúmenes de datos no estructurados o semi-estructurados, permitiendo almacenar documentos que pueden variar en estructura y soportar arrays y documentos anidados.

A pesar de sus fortalezas en el almacenamiento y gestión de documentos gracias a su estructura BSON, MongoDB se encontró con limitaciones significativas en cuanto a su integración con Apache Superset. Apache Superset, siendo una herramienta esencial para nuestras necesidades de visualización y análisis de datos, requiere una compatibilidad robusta con la fuente de datos subyacente para aprovechar plenamente sus capacidades de análisis y generación de dashboards interactivos. Desafortunadamente, MongoDB no es compatible de manera nativa con Apache Superset, lo que representa un obstáculo considerable para la creación de visualizaciones complejas y el análisis detallado requerido por nuestro proyecto. Esto nos llevó a descartar la elección de MongoDB.

Elasticsearch

En la industria de las ciudades inteligentes, la combinación de Elasticsearch como base de datos y Kibana como plataforma de paneles de control es una solución ampliamente adoptada. Empresas líderes en el sector, como T-system, han destacado esta combinación en su charla sobre ciudades inteligentes en la Universidad de Granada como la opción que utilizan.

Sin embargo, a pesar de su reconocimiento en la industria, nuestra decisión de no adoptar Elasticsearch y Kibana para este proyecto fue influida significativamente por cambios recientes en su licenciamiento. Elasticsearch ha modificado su modelo de licencia, adoptando términos más restrictivos que limitan la capacidad de ofrecer soluciones basadas en esta tecnología como servicio Elastic, 2021. Este cambio en la política de licenciamiento

presenta desafíos significativos para proyectos que buscan desarrollar soluciones abiertas y escalables, ya que impone restricciones en la distribución y el uso del software en entornos de servicio. Cabe destacar que Kibana, el software de visualización de paneles desarrollado por Elastic, la empresa que también desarrolla Elasticsearch, ha sufrido este cambio en su licenciamiento.

InfluxDB

Reconocida como una de las bases de datos más eficientes para el manejo de series de datos temporales, InfluxDB inicialmente parecía la opción más prometedora para nuestro proyecto. Además, también fue objeto de estudio en la asignatura Centros de Procesamiento de Datos. En las fases preliminares, se contempló una implementación que combinara InfluxDB con Grafana, un poderoso dúo para la visualización de datos en tiempo real. Sin embargo, nos enfrentamos a varios desafíos críticos que finalmente nos llevaron a descartar esta opción.

InfluxDB v2.0 requiere su propio lenguaje de consultas, Flux, el cual limita ampliamente la complejidad de las consultas en comparación con lenguajes ampliamente estudiados en la carrera, como SQL. Además, esto lo hace nativamente incompatible con Apache Superset. También, la estructura de la base de datos dificulta considerablemente el cotejo de estos datos con otros que no sean series temporales, por lo que se decidió, en última instancia, descartar su uso.

Si bien es cierto que otras versiones de Influx utilizan lenguajes más similares (como Influx v1, que utiliza InfluxQL) o compatibles (Influx v3, que permite utilizar FlightArrowSQL), Influx v1 sigue siendo incompatible, mientras que Influx v3 no es de código abierto ni libre y solo está disponible en la versión de pago ofrecida como servicio por InfluxData *Documentación de InfluxDB*, 2024.

Prometheus

Prometheus es una herramienta ampliamente utilizada para la monitorización y alertas en sistemas. Se destaca por su capacidad para recopilar y almacenar métricas como series temporales, realizar consultas y generar alertas basadas en estos datos. Su integración con Grafana la convierte en una solución poderosa para la visualización de métricas en tiempo real.

En las fases iniciales, se consideró la posibilidad de utilizar Prometheus en combinación con Grafana para gestionar las métricas de nuestro sistema. Sin embargo, Prometheus está diseñado principalmente para la monitorización de métricas y no para el almacenamiento de datos de aplicación más

complejos o la realización de análisis avanzados que requieren capacidades de base de datos SQL.

Además, Prometheus utiliza PromQL, un lenguaje de consultas específico para métricas, que, aunque es muy eficaz para su propósito, no es tan versátil como SQL para otros tipos de consultas analíticas. Por estas razones, se decidió no utilizar Prometheus como base de datos principal en este proyecto, aunque sigue siendo una opción excelente para la monitorización de sistemas.

No obstante, se está considerando la posibilidad de utilizar Prometheus en combinación con Grafana para la gestión de logs y la monitorización del clúster.

En la elección se tuvo en cuenta la facilidad de integración con las herramientas de visualización, el soporte para consultas complejas y la preferencia por software de código abierto y libre.

Para este proyecto, se ha elegido PostgreSQL, cuya documentación completa se encuentra disponible en línea *PostgreSQL Documentation*, s.f., junto con la documentación de TimescaleDB *TimescaleDB Documentation*, 2024.

4.1.2. Justificación de la Elección de PostgreSQL con TimescaleDB

La decisión de adoptar TimescaleDB se basó en varias razones clave:

- **Modelo Relacional y Optimización para Series Temporales:** PostgreSQL ofrece un robusto sistema de gestión de bases de datos relacionales que, cuando se complementa con TimescaleDB, se especializa en el manejo eficiente de series temporales.
- **Escalabilidad y Flexibilidad:** Esta combinación proporciona una solución escalable que puede manejar grandes conjuntos de datos y consultas complejas de análisis, adaptándose perfectamente a las necesidades de proyectos de ciudades inteligentes.
- **Soporte Comunitario y Extensibilidad:** La amplia comunidad de usuarios y el soporte para diversas extensiones permiten adaptar la base de datos a necesidades específicas, asegurando una plataforma robusta para el desarrollo futuro.
- **Extensión de PostgreSQL:** TimescaleDB es una extensión de PostgreSQL, lo que implica que si existe compatibilidad con PostgreSQL, también la habrá con TimescaleDB. En caso de ser necesario, se podrá utilizar PostgreSQL en lugar de TimescaleDB, situación que se presentará más adelante en el proyecto.

4.2. Herramientas comúnmente utilizadas en ETL

A continuación se especifican algunas tenologías usadas para crear ETLs.

4.2.1. Apache Kafka

Apache Kafka es un sistema de intermediación de mensajes que admite suscripciones. Su utilización es especialmente relevante para desacoplar aplicaciones, permitiendo que diversos componentes produzcan o consuman datos a diferentes velocidades, además de redirigir la información a diversos puntos de nuestro sistema y tratarla.

Apache Kafka se define como una plataforma de transmisión de datos distribuida, diseñada para permitir la publicación, suscripción, almacenamiento y procesamiento de flujos de registros (eventos) en tiempo real. Es frecuentemente empleado en la construcción de infraestructuras de datos que requieren robustez y escalabilidad, siendo capaz de manejar volúmenes elevados de mensajes con mínima latencia*Apache Kafka Documentation*, 2024.

La utilización de Kafka fue descartada debido a las características particulares de nuestro caso, en el cual el sistema ya está desacoplado y no hay necesidad de una extracción de datos en tiempo real, ya que utilizamos una extracción por lotes.

4.2.2. Apache Airflow

Apache Airflow es una plataforma avanzada de orquestación de flujos de trabajo diseñada para programar, coordinar y monitorear tareas dentro de flujos de trabajo complejos. Esta herramienta es particularmente eficaz en la gestión de dependencias entre tareas, actuando como una versión significativamente potenciada de un sistema cron, gracias a sus amplias funcionalidades. A pesar de que en algunos contextos se pueda presentar como una herramienta ETL, Apache Airflow no se clasifica como tal*Apache Airflow Documentation*, 2024.

Se descartó su uso debido a que el tiempo necesario para su correcta configuración incrementaba significativamente el tiempo planificado para una única funcionalidad que ya estaba implementada. Es decir, no resulta rentable para una funcionalidad ya existente en Kubernetes, además de que solo tenemos una tarea que requiere ser monitorizada.

4.2.3. Apache NiFi

Apache NiFi es una plataforma de procesamiento y distribución de datos automatizada que permite diseñar, controlar y monitorear flujos de datos en tiempo real. NiFi es altamente configurable y soporta una variedad de fuentes y destinos de datos, incluyendo sistemas de archivos, bases de datos y sistemas de mensajería*Apache NiFi Documentation*, 2024.

Fue descartado debido a que la extracción de datos se realiza por lotes.

4.3. Orquestación de contenedores

A continuación, describo algunos de los softwares utilizados para la orquestación de contenedores.

Docker Swarm

Docker Swarm es una herramienta de orquestación de contenedores que viene con Docker. Es una opción más sencilla en comparación con Kubernetes y está mejor orientada para despliegues más pequeños.

Entre los puntos fuertes de Docker Swarm se encuentran su interfaz fácil de usar y su sencilla configuración, lo que lo convierte en una buena opción para equipos con menos experiencia en la contenedorización. Sin embargo, una de sus contras es que, al ser más simple, puede no ser tan robusto ni escalable como Kubernetes para proyectos de mayor envergadura.

Nomad

Nomad es una herramienta de orquestación de contenedores desarrollada por HashiCorp. Es ligera y flexible, y puede ejecutarse en cualquier infraestructura, incluidas nubes públicas, nubes privadas y servidores físicos.

Nomad es fácil de usar y no requiere una configuración compleja, lo que la convierte en una buena opción para equipos más pequeños. Sin embargo, su simplicidad puede ser una limitación para despliegues más grandes o complejos, donde se podrían necesitar características avanzadas no presentes en Nomad.

Mesos

Apache Mesos es una herramienta de código abierto que permite gestionar y orquestar contenedores, máquinas virtuales y otros recursos. Tiene una

arquitectura modular y puede integrarse con diferentes entornos de contenedores, lo que la convierte en una opción flexible para despliegues complejos.

Mesos es una buena elección para equipos con necesidades avanzadas y la experiencia para gestionar su complejidad. Sin embargo, su configuración y mantenimiento pueden ser complicados, requiriendo un nivel significativo de experiencia técnica.

OpenShift

OpenShift es una plataforma de aplicaciones de contenedores de código abierto desarrollada por Red Hat. Está basada en Kubernetes y proporciona características y herramientas adicionales para simplificar el despliegue y la gestión de contenedores.

OpenShift es una buena opción para equipos que buscan una plataforma de contenedores más completa que incluya características como CI/CD integrado, monitorización integrada y seguridad incorporada. Sin embargo, puede ser más costoso y complejo de implementar que soluciones más sencillas como Docker Swarm.

Rancher

Rancher es una plataforma completa de gestión de contenedores que proporciona Kubernetes y otras herramientas de orquestación de contenedores como servicio. Es fácil de usar y ofrece una interfaz amigable para gestionar los contenedores.

Rancher es una buena elección para equipos que necesitan una plataforma completa de contenedores y prefieren un servicio gestionado. No obstante, aunque facilita muchas tareas, puede ser redundante para equipos que ya utilizan otras herramientas de gestión de Kubernetes.

Amazon ECS

Amazon Elastic Container Service (ECS) es un servicio de contenedores completamente gestionado que permite ejecutar contenedores en la nube de Amazon Web Services (AWS). Proporciona una manera simple y rentable de desplegar y gestionar contenedores a gran escala.

ECS es una buena opción para equipos que ya utilizan AWS y buscan una solución de contenedores totalmente gestionada. Sin embargo, su uso está muy ligado al ecosistema de AWS, lo que puede ser una limitación para equipos que prefieren una solución más agnóstica respecto a la nube.

Apache Aurora

Apache Aurora es una herramienta de código abierto para gestionar servicios de larga duración y trabajos cron. También puede utilizarse para gestionar contenedores y es una buena opción para equipos que buscan una opción más flexible y personalizable para la orquestación de contenedores.

Aurora es una buena elección para equipos con necesidades avanzadas y la experiencia para gestionar su complejidad. Sin embargo, su uso puede ser complejo y requiere una inversión significativa en tiempo y recursos para su configuración y mantenimiento.

4.4. Kubernetes

Para este fin hemos utilizado Kubernetes que es una plataforma de código abierto utilizada para automatizar la implementación, el escalado y la gestión de aplicaciones contenerizadas. Actúa como un orquestador de contenedores, lo que significa que permite a los usuarios gestionar sus aplicaciones construidas con contenedores (como Docker) de manera más eficiente, asegurando que se ejecuten donde y cuando sean necesarios, y ayudando a optimizar el uso de los recursos subyacentes.

La integración de Kubernetes en sistemas y procesos permite a las organizaciones adoptar un enfoque más ágil y flexible en el despliegue y la gestión de sus aplicaciones. Esto es especialmente útil en entornos de microservicios, donde diferentes componentes de una aplicación se desarrollan, despliegan y escalan de manera independiente. Kubernetes facilita esta gestión al proporcionar:

- **Automatización del despliegue**: Permite la implementación automática de contenedores según las necesidades de la aplicación, sin intervención manual.
- **Escalabilidad**: Soporta el escalado automático de aplicaciones hacia arriba o hacia abajo, en función de la demanda, mejorando así la eficiencia y la disponibilidad de los recursos.
- **Balanceo de carga**: Distribuye automáticamente el tráfico entre los contenedores de la aplicación para asegurar un rendimiento óptimo y la disponibilidad del servicio.
- **Autoreparación**: Tiene la capacidad de reiniciar contenedores que fallan, reemplazarlos y eliminar contenedores que no responden a las verificaciones de estado definidas por el usuario, garantizando la estabilidad y fiabilidad de las aplicaciones.

- **Gestión de configuraciones y secretos**: Permite a los desarrolladores almacenar y gestionar información sensible, como contraseñas, tokens OAuth y claves SSH, de manera segura.

En la figura 4.1 se muestran las partes fundamentales de la arquitectura de Kubernetes. Estas partes son comunes a la mayoría de distribuciones de Kubernetes.

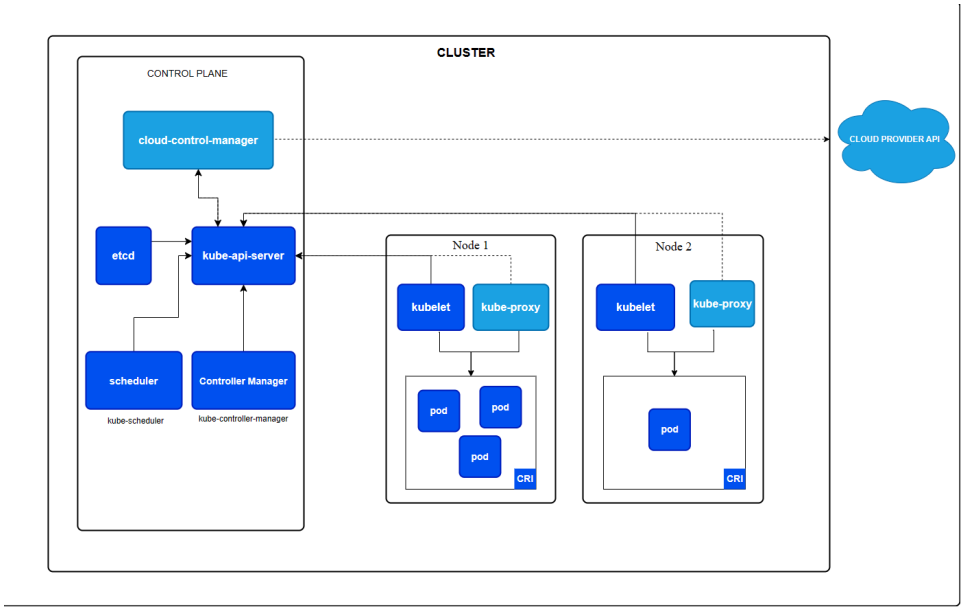

Figura 4.1: Partes comunes a una distribución estándar de Kubernetes.(The Kubernetes Authors, 2024)

Existen varias distribuciones de kubernetes, algunas ofrecidas como servicio en la nube como Google Kubernetes Engine (GKE) o Amazon Elastic Kubernetes Service (EKS) y otras autogestionadas como Rancher, OpenShift, K3s y K0s entre otras.

En nuestro caso, utilizamos la distribución de Kubernetes K0s desarrollada por Mirantis porque permite montar un cloud privado en local de manera sencilla y rápida, además de ser altamente configurable en caso de ser necesario.

4.4.1. Componentes de Kubernetes

Kubernetes está compuesto por varios componentes clave que facilitan la gestión de aplicaciones contenerizadas. A continuación, se describen algunos de los componentes más importantes o que han sido utilizados en el proyecto:

Pods

Un pod es la unidad de despliegue más pequeña que se puede crear y gestionar en Kubernetes. Representa un grupo de uno o más contenedores que comparten recursos de almacenamiento y red, así como una especificación sobre cómo deben ejecutarse estos contenedores. Los Pods están diseñados para funcionar como un "host lógico" específico de la aplicación, lo que significa que los contenedores dentro de un Pod están estrechamente acoplados y deben ser co-localizados y co-programados para ejecutarse en un contexto compartido.

Cada Pod puede contener múltiples contenedores de aplicaciones que funcionan juntos de manera cohesiva. Esta agrupación permite a los contenedores compartir el mismo espacio de red y almacenamiento, lo que facilita la comunicación y la gestión de datos entre ellos. En esencia, los Pods permiten a Kubernetes gestionar y orquestar aplicaciones contenerizadas de manera eficiente, proporcionando un entorno en el que múltiples contenedores pueden operar como una sola unidad lógica.

En comparación con contextos no basados en la nube, los Pods en Kubernetes son análogos a ejecutar múltiples aplicaciones en la misma máquina física o virtual, pero en un entorno de nube. Esta arquitectura permite una mayor flexibilidad y escalabilidad, ya que los recursos pueden ser gestionados de manera dinámica según las necesidades de la aplicación*Kubernetes Pod Documentation*, 2024.

etcd

etcd es una base de datos de clave-valor distribuida y consistente que Kubernetes utiliza para almacenar todos los datos del clúster. Es fundamental para la configuración y el estado del clúster. etcd guarda la información sobre la configuración de los nodos, el estado de los Pods y otros datos necesarios para la operación del clúster.

Despliegue de Aplicaciones

En Kubernetes, el despliegue de aplicaciones se puede clasificar en dos tipos principales: despliegue sin estado (*stateless*) y despliegue con estado (*stateful*).

- **Despliegues Stateless**: Este tipo de despliegue se utiliza para aplicaciones que no mantienen un estado persistente. Un ejemplo típico de estas aplicaciones son los servidores web que no gestionan sesiones persistentes. Los despliegues *stateless* permiten una fácil replicación y

escalabilidad, ya que cada instancia de la aplicación puede funcionar de manera independiente sin requerir sincronización con otras instancias.

- **Despliegues Stateful**: Este tipo de despliegue se utiliza para aplicaciones que necesitan mantener un estado persistente, como las bases de datos. Los despliegues *stateful* gestionan la persistencia y el orden de despliegue de los Pods, garantizando que cada Pod tenga una identidad única y que los datos se mantengan entre reinicios o reprogramaciones. Esto es crucial para aplicaciones donde la consistencia y la persistencia de los datos son fundamentales.

Persistent Volumes (PV) y Persistent Volume Claims (PVC)

- **Persistent Volumes (PV)**: Son recursos de almacenamiento en el clúster que existen independientemente del ciclo de vida de los Pods. Proporcionan una forma de abstraer los detalles del almacenamiento subyacente.
- **Persistent Volume Claims (PVC)**: Son solicitudes de almacenamiento por parte de un usuario. Un PVC solicita un PV con ciertas características y Kubernetes se encarga de enlazarlos.

Servicios (Services)

Un Servicio en Kubernetes es una abstracción que define un conjunto lógico de Pods y una política para acceder a ellos. Kubernetes soporta varios tipos de servicios:

- **ClusterIP**: Expone el servicio en una dirección IP interna del clúster. Este tipo de servicio es accesible solo dentro del clúster.
- **NodePort**: Expone el servicio en cada IP de nodo en un puerto estático específico. Hace que el servicio sea accesible desde fuera del clúster.
- **LoadBalancer**: Crea un balanceador de carga externo en el proveedor de nube y asigna una IP fija al servicio.
- **ExternalName**: Mapea el servicio a un nombre DNS (Domain Name System) especificado (funciona solo para nombres DNS externos).

StorageClass

El objeto **StorageClass** en Kubernetes proporciona una forma de describir las "clases" de almacenamiento disponibles. Permite a los administra-

dores definir diferentes niveles de almacenamiento y características específicas, como la calidad del servicio, el rendimiento y el método de aprovisionamiento. Los StorageClasses permiten el aprovisionamiento dinámico de volúmenes, lo que significa que Kubernetes puede crear Persistent Volumes (PV) automáticamente cuando un Persistent Volume Claim (PVC) lo solicita. Cada StorageClass tiene un `provisioner` que determina el proveedor de almacenamiento, y parámetros adicionales que definen sus características y comportamiento.

- **provisioner**: Especifica el plugin de almacenamiento que debe usarse para crear el volumen.
- **parameters**: Un conjunto de parámetros que se pasan al provisioner para definir el comportamiento del almacenamiento.
- **reclaimPolicy**: Define qué sucede con el PV cuando un PVC es eliminado (por ejemplo, `Retain`, `Delete`).
- **bindingMode**: Determina cuándo los PVs se asignan a los PVCs (por ejemplo, `Immediate` o `WaitForFirstConsumer`).

4.4.2. NGINX Ingress Controller: Funcionalidad y Uso

El NGINX Ingress Controller es una herramienta clave en Kubernetes para gestionar el acceso externo a los servicios desplegados en el clúster. Actúa como un controlador que gestiona las reglas de entrada definidas a través de recursos Ingress, proporcionando un único punto de entrada para el tráfico HTTP y HTTPS.

Funcionamiento de NGINX Ingress Controller

El NGINX Ingress Controller se despliega como un pod dentro del clúster de Kubernetes y escucha las definiciones de Ingress para dirigir el tráfico a los servicios correspondientes. Las definiciones de Ingress especifican cómo debe redirigirse el tráfico a los diferentes servicios basados en rutas URL y nombres de host.

El flujo de trabajo típico de NGINX Ingress Controller es el siguiente:

1. Los administradores definen reglas de Ingress que especifican rutas URL y nombres de host, junto con los servicios de destino.
2. El NGINX Ingress Controller lee estas definiciones y configura internamente el servidor NGINX para manejar las solicitudes según las reglas especificadas.

3. Cuando llega una solicitud al punto de entrada del Ingress Controller, NGINX dirige el tráfico al servicio correspondiente basado en las reglas de Ingress.

Usos Comunes de NGINX Ingress Controller

NGINX Ingress Controller es ampliamente utilizado en entornos Kubernetes debido a su capacidad para gestionar y balancear el tráfico de manera eficiente. Algunos de los usos más comunes incluyen:

- **Balanceo de Carga:** NGINX Ingress Controller distribuye el tráfico entrante entre múltiples réplicas de un servicio, proporcionando alta disponibilidad y balanceo de carga.
- **Terminación SSL:** NGINX Ingress Controller puede manejar la terminación SSL, gestionando certificados y encriptación para asegurar las comunicaciones HTTP y HTTPS.
- **Reescritura de URL:** Permite la reescritura y redirección de URL, facilitando la gestión de rutas y la implementación de políticas de acceso.
- **Control de Acceso:** NGINX Ingress Controller puede integrar autenticación y autorización, proporcionando control de acceso basado en políticas.
- **Monitoreo y Logging:** Proporciona capacidades de monitoreo y logging detalladas, permitiendo a los administradores rastrear el tráfico y diagnosticar problemas.

NGINX Ingress Controller en el TFG

En el contexto de este trabajo de fin de grado, NGINX Ingress Controller se utiliza para gestionar el tráfico de entrada a los servicios desplegados en el clúster Kubernetes. Trabaja en conjunto con MetalLB para proporcionar una solución completa de balanceo de carga y gestión de tráfico.

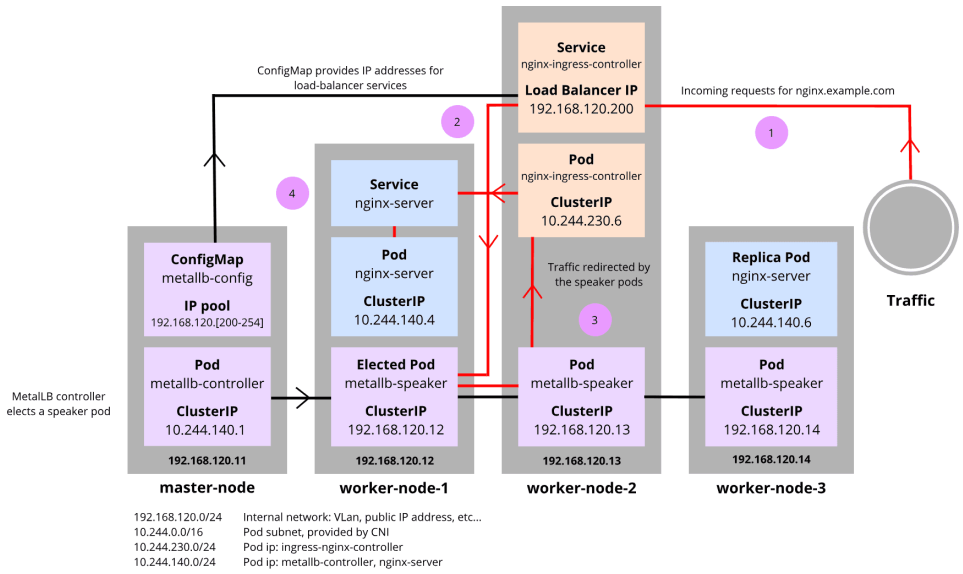

Figura 4.2: Esquema de integración de MetalLB y NGINX Ingress.(Cottart, 2022)

El flujo de la petición se desarrolla de la siguiente manera (ver Figura Cottart, 2022, fig. 4.2):

1. Una solicitud llega a http://nginx.example.com, que se resuelve como la IP del balanceador de carga del controlador NGINX Ingress.
2. La petición sale del servicio hacia el pod de metallb-speaker seleccionado, el cual se comunica con los otros pods de metallb-speaker para redirigir el tráfico de datos al pod del controlador NGINX Ingress.
3. El tráfico de datos llega al pod del controlador NGINX Ingress.
4. El pod del controlador NGINX Ingress envía el tráfico de datos al servicio solicitado con el nombre de host **nginx.example.com**. El servidor NGINX es contactado y sirve el sitio web al cliente externo.

En resumen, NGINX Ingress Controller es una herramienta esencial en este proyecto, permitiendo la gestión eficiente y segura del tráfico HTTP y HTTPS hacia los servicios del clúster Kubernetes. Su integración con MetalLB proporciona una solución robusta para el balanceo de carga y la exposición de servicios.

4.4.3. MetalLB: Funcionalidad y Uso

MetalLB es una implementación de balanceador de carga para clústeres Kubernetes en entornos donde la funcionalidad de balanceo de carga no está

disponible nativamente. Está diseñado para proporcionar a Kubernetes la capacidad de asignar IPs externas a servicios, permitiendo que sean accesibles fuera del clúster. Esto es especialmente útil en entornos on-premise o en nubes privadas que no ofrecen servicios de balanceo de carga integrados.

Funcionamiento de MetalLB

MetalLB opera en uno de dos modos: *Layer 2* y *BGP*.

- **Layer 2:** En este modo, MetalLB responde a las solicitudes ARP (Address Resolution Protocol) para las IPs asignadas a servicios. Esto permite que cualquier tráfico dirigido a estas IPs sea redirigido a los pods correspondientes dentro del clúster Kubernetes.

- **BGP (Border Gateway Protocol):** En este modo, MetalLB utiliza BGP para anunciar las IPs asignadas a servicios a los routers en la red. Esto permite una integración más avanzada con la infraestructura de red existente, distribuyendo el tráfico de manera eficiente y escalable.

El flujo de trabajo típico de MetalLB es el siguiente:

1. MetalLB escucha las solicitudes de servicios que requieren una IP externa.

2. MetalLB asigna una IP externa a estos servicios desde un rango preconfigurado.

3. Dependiendo del modo operativo (Layer 2 o BGP), MetalLB asegura que el tráfico dirigido a estas IPs sea redirigido al servicio correspondiente dentro del clúster.

4.4.4. Distribución de Bitnami de PostgreSQL

En el contexto de este proyecto, PostgreSQL se utiliza para almacenar los datos que se mostrarán en el panel de control de Apache Superset. La distribución de PostgreSQL de Bitnami proporciona una configuración que permite aprovechar el entorno cloud de manera eficiente.

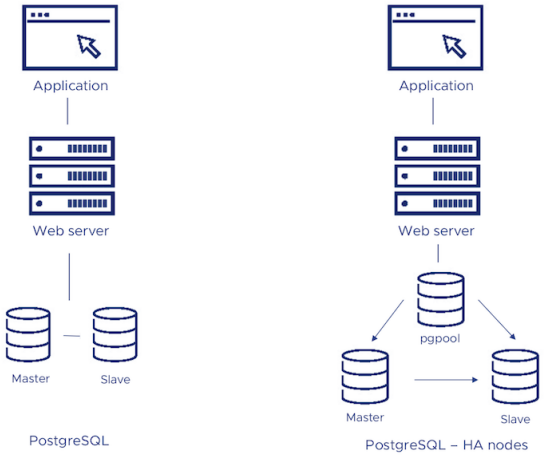

Figura 4.3: Topología de la distribución de Bitnami de PostgreSQL. Actualmente se usa la topología de la izquierda. (VMware, 2024)

PostgreSQL sigue la arquitectura, que se muestra en la figura 4.3, *master-slave* en la cual solo la instancia de la base de datos *master* puede atender peticiones de escritura. Las peticiones de lectura se distribuyen entre las instancias *slave*, permitiendo así un balanceo de carga y mejorando el rendimiento del sistema. Los datos se sincronizan entre las instancias mediante un proceso de replicación.

4.4.5. Redis: Funcionalidad y Uso

Redis es una base de datos en memoria, de código abierto, que se utiliza principalmente como caché y agente de mensajes. Es conocida por su alta velocidad y flexibilidad, ya que soporta estructuras de datos como cadenas, hashes, listas, conjuntos y más. En este proyecto, se utiliza la distribución de Redis proporcionada por Bitnami, que incluye varias optimizaciones y configuraciones predefinidas para facilitar su despliegue y administración.

Funcionamiento de Redis

Redis opera completamente en memoria, lo que le permite acceder y manipular datos a una velocidad extremadamente alta. Aunque puede persistir datos en disco, su diseño principal está orientado a operaciones rápidas en memoria. Redis es monothread, lo que significa que maneja todas las operaciones de manera secuencial con un único hilo, eliminando la necesidad de gestionar la concurrencia multihilo y evitando los bloqueos.

Redis utiliza un modelo de cliente-servidor, donde los clientes se conectan al servidor Redis para realizar operaciones de lectura y escritura en la base de datos. Redis puede configurarse en varias topologías, incluyendo la replicación master-slave, donde una instancia master maneja todas las operaciones de escritura y las réplicas slave manejan las operaciones de lectura, proporcionando redundancia y escalabilidad.

Usos Comunes de Redis

Redis es utilizado en una amplia variedad de aplicaciones debido a su velocidad y versatilidad. Algunos de los usos más comunes incluyen:

- **Caché:** Redis se utiliza comúnmente como caché para almacenar datos temporales que necesitan ser accesibles rápidamente. Esto incluye sesiones de usuario, resultados de consultas frecuentes y otros datos que requieren acceso rápido.

- **Colas de Mensajes:** Redis puede actuar como un agente de mensajes para colas de trabajo, donde las tareas se encolan y los trabajadores las procesan. Esto es útil para manejar tareas asíncronas y distribuir la carga de trabajo.

- **Sesiones de Usuario:** En aplicaciones web, Redis se utiliza para almacenar las sesiones de los usuarios, permitiendo un acceso rápido y eficiente a la información de la sesión.

- **Análisis en Tiempo Real:** La capacidad de Redis para manejar grandes volúmenes de datos a alta velocidad lo hace ideal para aplicaciones que requieren análisis en tiempo real, como el monitoreo de eventos y la agregación de datos en tiempo real.

Redis en el Proyecto

En el contexto de este proyecto, Redis se utiliza como broker de mensajes para Celery y para almacenar las sesiones de los usuarios que acceden a Apache Superset. La distribución de Redis de Bitnami incluye configuraciones optimizadas para mejorar el rendimiento y la fiabilidad, así como herramientas para facilitar su despliegue y gestión.

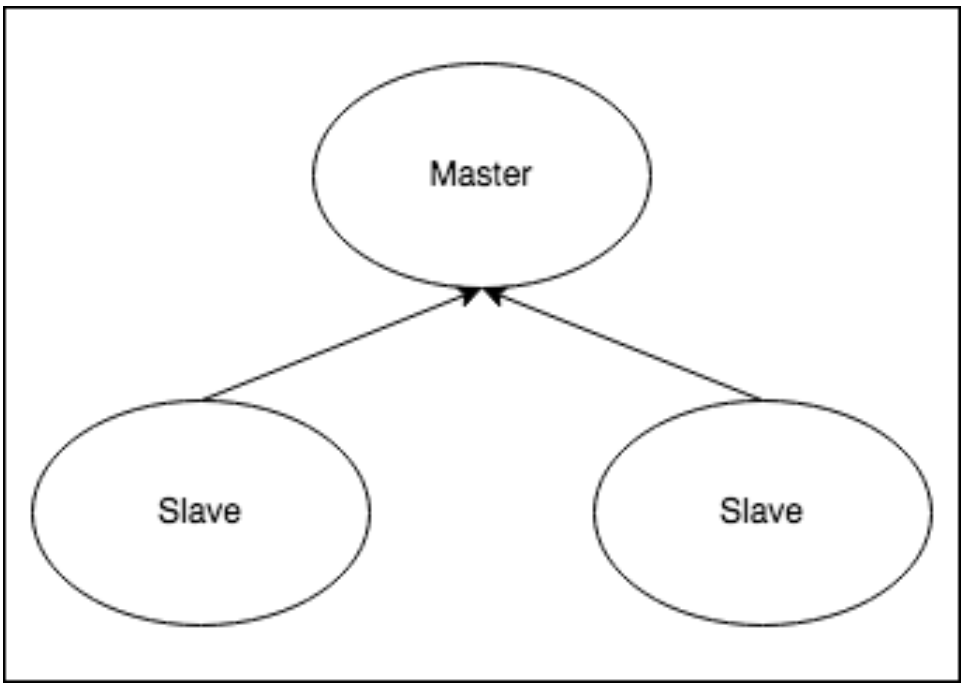

Figura 4.4: Arquitectura de Redis utilizada, extraída de (vmware, 2024). Esta arquitectura tiene un único punto de escritura y soporta varias bases de datos en modo esclavo.

Redis sigue una arquitectura master-slave en este proyecto, lo que significa que todas las operaciones de escritura se realizan en la base de datos master y luego se replican a las instancias slave. Las instancias slave pueden manejar operaciones de lectura, proporcionando redundancia y mejorando el rendimiento de lectura.

Polémica

Redis está enfrentando dificultades relacionadas con la intención de transformar el software en un producto que deje de ser libre y de código abierto. La empresa que adquirió Redis ha encontrado obstáculos en este proceso (Khawaja Shams, 2024; Proven, 2024) debido al antiguo licenciamiento de redis.

En este proyecto se utiliza una distribución mantenida y ofrecida por Bitnami, la cual se conserva como software libre bajo la licencia Apache 2.0.

4.4.6. Celery

Celery es una librería de código abierto utilizada para la gestión de tareas asíncronas y la programación de trabajos periódicos en aplicaciones distribuidas. Principalmente, Celery permite delegar la ejecución de tareas a trabajadores (*workers*) que pueden ejecutarlas de manera independiente, permitiendo que el sistema principal continúe operando sin bloqueos.

El funcionamiento de Celery se basa en tres componentes principales: tareas, trabajadores y un *broker* de mensajes.

- **Tareas (Tasks):** Son las unidades de trabajo que se desean ejecutar de manera asíncrona. Estas tareas pueden ser funciones definidas en el código que realizan operaciones que pueden llevar tiempo, como el procesamiento de datos, el envío de correos electrónicos o cualquier otra operación que pueda beneficiarse de ser ejecutada en segundo plano.
- **Trabajadores (Workers):** Son los procesos que se encargan de ejecutar las tareas. Un *worker* puede ejecutar múltiples tareas de manera concurrente, mejorando así la eficiencia y el rendimiento de la aplicación. Los *workers* pueden estar distribuidos en múltiples máquinas, lo que permite escalabilidad horizontal.
- **Broker de Mensajes:** Es el intermediario que recibe las tareas desde la aplicación principal y las distribuye a los *workers*. El *broker* de mensajes asegura que las tareas se envíen de manera fiable y ordenada. Celery soporta varios *brokers* de mensajes, siendo Redis y RabbitMQ los más comunes.

El flujo de trabajo típico de Celery es el siguiente:

1. La aplicación principal define una tarea y la envía al *broker* de mensajes.
2. El *broker* de mensajes coloca la tarea en una cola.
3. Un *worker* recibe la tarea de la cola, la ejecuta y, una vez completada, devuelve el resultado al *broker*.
4. La aplicación principal puede recibir notificaciones de finalización de tarea o consultar el estado y resultado de la tarea según sea necesario.

En el contexto de este proyecto, Celery es utilizado por Apache Superset para gestionar los trabajos activos, distribuyendo y ejecutando tareas de manera eficiente. Redis actúa como el *broker* de mensajes, facilitando la comunicación entre la aplicación principal y los *workers* de Celery.

4.4.7. Helm: Gestión de Paquetes para Kubernetes

Helm es una herramienta de gestión de paquetes diseñada específicamente para Kubernetes, un sistema de orquestación de contenedores ampliamente utilizado en la gestión de aplicaciones en entornos de producción. Helm facilita la instalación, actualización y gestión de aplicaciones Kubernetes mediante el uso de *charts*, que son paquetes preconfigurados que contienen todos los recursos necesarios para desplegar una aplicación.

Un *chart* de Helm incluye plantillas de Kubernetes que permiten personalizar y parametrizar despliegues complejos, lo que simplifica la gestión y reduce el riesgo de errores de configuración. Esta herramienta se ha convertido en un estándar en la industria para la gestión de aplicaciones en Kubernetes debido a su capacidad para gestionar la complejidad y su facilidad de uso.

Helm sirve para varios propósitos fundamentales en la gestión de aplicaciones en Kubernetes:

- **Automatización del Despliegue**: Permite automatizar el proceso de despliegue de aplicaciones, asegurando que todas las configuraciones y dependencias se apliquen de manera consistente.
- **Gestión de Versiones**: Facilita la actualización y la reversión de aplicaciones a versiones anteriores, lo que es crucial para la continuidad del negocio y la gestión de cambios.
- **Configuración Centralizada**: Ofrece una manera de gestionar configuraciones a través de archivos de valores, lo que permite reutilizar configuraciones y aplicar cambios de manera centralizada.
- **Eficiencia Operacional**: Reduce la complejidad de los despliegues manuales y minimiza los errores humanos, lo que mejora la eficiencia operativa.

4.4.8. Apache Superset: Funcionalidad y Uso

Apache Superset es una plataforma de visualización de datos de código abierto que permite a los usuarios explorar y visualizar datos de manera interactiva. Ofrece una interfaz de usuario intuitiva que facilita la creación de dashboards dinámicos, gráficos y reportes personalizados. Superset está diseñado para manejar grandes volúmenes de datos y puede integrarse con múltiples fuentes de datos, lo que lo convierte en una herramienta robusta para la inteligencia de negocios.

Componentes de Apache Superset

El funcionamiento de Apache Superset se basa en varios componentes. A continuación describo algunos de los más importantes que se utilizan:

- **SQLAlchemy:** Es una biblioteca de Python que facilita la interacción con bases de datos mediante un ORM (Object-Relational Mapping). SQLAlchemy permite a los desarrolladores trabajar con bases de datos usando clases y objetos de Python en lugar de escribir consultas SQL directamente. En Superset, SQLAlchemy es fundamental para conectar y extraer datos de diversas fuentes de datos, proporcionando una capa de abstracción que facilita la manipulación de datos independientemente del sistema de base de datos subyacente.
- **Flask:** Es un microframework web para Python que gestiona la interacción con la interfaz de usuario y las solicitudes HTTP en Superset. Flask permite la creación y manejo de vistas web, autenticación de usuarios, gestión de sesiones y configuración de API RESTful. Su flexibilidad y ligereza lo hacen ideal para aplicaciones como Superset, donde se requiere una base sólida para el desarrollo de funcionalidades personalizadas y extensiones.
- **Prophet:** Es una biblioteca de previsión de series temporales desarrollada por Facebook. En Superset, Prophet se utiliza para realizar análisis predictivo y modelado de datos. Permite a los usuarios crear modelos predictivos directamente desde los dashboards, visualizando previsiones y tendencias futuras basadas en datos históricos. Esto es especialmente útil para la planificación empresarial y la toma de decisiones informadas.
- **Celery:** Una biblioteca de colas de tareas distribuidas que permite la ejecución de tareas en segundo plano, manejando trabajos pesados y tareas asíncronas en Superset.

4.5. Otras tectnologías

En esta sección se describen algunas tecnologías notables que fueron planteadas o utilizadas en algún punto del proyecto pero que posteriormente su uso fue descartado.

4.5.1. Fluent Bit

Fluent Bit es una utilidad similar a Logstash que permite extraer datos de un archivo de logs y enviarlos a una cola de Kafka. La principal diferencia

con Fluentd es que Fluent Bit es un binario sin dependencias. Se descartó su uso cuando se decidió no utilizar Kafka, ya que Fluent Bit dependía de Kafka en este proyecto.

4.5.2. Telegraf

Telegraf es un agente de recopilación de métricas y datos de eventos diseñado para ser ligero y fácil de configurar. Desarrollado como parte del ecosistema InfluxDB, su función principal es recoger, procesar y enviar datos a diferentes destinos de almacenamiento de series temporales o bases de datos para su análisis. Telegraf soporta una amplia variedad de fuentes de datos gracias a su sistema de *plugins*, que le permite recoger métricas de una amplia gama de sistemas, servicios y aplicaciones. Su uso fue descartado cuando se decidió abandonar la implementación del proyecto que utilizaba InfluxDB.

4.5.3. Arrow Flight SQL

Arrow Flight SQL es un protocolo diseñado para interactuar con bases de datos SQL utilizando el formato en memoria Arrow y el marco de trabajo RPC (Remote Procedure Call) de Flight. Este protocolo permite que una base de datos implemente métodos RPC según las especificaciones, permitiendo a los clientes de la base de datos utilizar el cliente Flight SQL para interactuar con cualquier base de datos que soporte los endpoints necesarios. Flight SQL proporciona métodos para la ejecución de consultas SQL y la gestión de declaraciones preparadas, mejorando significativamente la eficiencia en el acceso a datos en comparación con los métodos tradicionales.

Una de las principales características de Arrow Flight SQL es su capacidad para manejar metadatos SQL. A través de una serie de comandos, los usuarios pueden recuperar información sobre los catálogos, esquemas, tablas, tipos de tablas y otras estructuras de base de datos. Esta funcionalidad es compatible con los métodos GetFlightInfo y GetSchema, que devuelven la información en el formato de datos Arrow, asegurando que los metadatos se presenten de manera consistente y eficiente junto con los resultados de las consultas.

Además, Arrow Flight SQL incluye comandos para la gestión de la sesión del servidor y la ejecución de consultas SQL ad-hoc o preparadas. Esto incluye la capacidad de crear, ejecutar y cerrar declaraciones preparadas, así como la ejecución de consultas SQL directas. A pesar de que Flight SQL todavía se considera experimental y puede estar sujeto a cambios en el protocolo, su diseño modular y orientado a mejorar la interoperabilidad y el rendimiento lo convierte en una herramienta poderosa para la integración y

gestión de bases de datos en entornos distribuidos.

Su uso fue descartado debido a que ya contamos con buenas integraciones con Apache Superset y no utilizamos InfluxDB v3.

4.5.4. Grafana

Grafana es una plataforma de análisis y visualización de datos de código abierto, ampliamente reconocida por su capacidad para permitir a los usuarios crear paneles de control dinámicos y visualmente atractivos. Estos paneles facilitan la comprensión de complejos conjuntos de datos mediante gráficos, mapas y alertas en tiempo real. Grafana es particularmente valorada en el ámbito del monitoreo de infraestructuras y aplicaciones, así como en el análisis de métricas operativas y estadísticas.

Una de las principales ventajas de Grafana es su compatibilidad con múltiples fuentes de datos, incluyendo bases de datos populares como MySQL, PostgreSQL, Graphite, InfluxDB y Prometheus, entre otras. Esto permite a los usuarios integrar datos de diversas fuentes para obtener una visión holística del rendimiento de sus sistemas o procesos de negocio.

El uso de Grafana fue descartado debido a que su aspecto visual no convenció al equipo de investigación y no ofrecía tablas enfocadas al análisis de manera nativa.

4.6. Scrum

Scrum es un marco de trabajo ágil utilizado para gestionar proyectos y desarrollar productos de manera iterativa e incremental. Este enfoque es especialmente efectivo en entornos complejos donde los requisitos pueden cambiar rápidamente. Scrum proporciona una estructura que permite a los equipos trabajar de manera colaborativa, entregando incrementos funcionales del producto de forma regular y frecuente.

4.6.1. Partes de Scrum

Scrum se compone de varios elementos clave que facilitan la organización y ejecución del trabajo del equipo:

- **Sprint**: Es el corazón de Scrum, un ciclo de trabajo de duración fija (generalmente de una a cuatro semanas) durante el cual se desarrolla un incremento del producto. Cada Sprint comienza con una planificación y termina con una revisión y retrospectiva.

- **Backlog del Producto**: Es una lista priorizada de todo lo que se necesita en el producto, gestionada por el Product Owner. Contiene características, funciones, requisitos, mejoras y arreglos.
- **Backlog del Sprint**: Es una lista de tareas seleccionadas del Backlog del Producto que se comprometen a completar en un Sprint específico. Es gestionada por el equipo de desarrollo.
- **Incremento**: Es la suma de todos los elementos del Backlog del Producto completados durante un Sprint y todos los Sprints anteriores. Debe ser un producto utilizable y que cumpla con la definición de "hecho".
- **Reunión de Planificación del Sprint**: Al inicio de cada Sprint, el equipo de Scrum se reúne para planificar el trabajo a realizar. Se definen los objetivos del Sprint y se seleccionan los elementos del Backlog del Producto que se incluirán en el Backlog del Sprint.
- **Reunión Diaria de Scrum (Daily Scrum)**: Es una breve reunión diaria de aproximadamente 15 minutos donde el equipo de desarrollo sincroniza sus actividades y crea un plan para las siguientes 24 horas.
- **Revisión del Sprint (Sprint Review)**: Al final de cada Sprint, el equipo presenta el trabajo completado al Product Owner y a otros stakeholders para recibir feedback.
- **Retrospectiva del Sprint (Sprint Retrospective)**: Después de la revisión del Sprint, el equipo reflexiona sobre el Sprint que acaba de terminar para identificar mejoras para futuros Sprints.

4.6.2. Roles en Scrum

Scrum define tres roles principales que son esenciales para el éxito del proyecto:

- **Product Owner**: Es el responsable de maximizar el valor del producto y del trabajo del equipo de desarrollo. Gestiona el Backlog del Producto, asegurando que esté visible, claro y priorizado.
- **Scrum Master**: Es el facilitador del equipo de Scrum. Ayuda a todos a entender y aplicar las prácticas de Scrum, elimina impedimentos y asegura un entorno de trabajo productivo.
- **Equipo de Desarrollo**: Es un grupo auto-organizado de profesionales que trabajan juntos para entregar incrementos del producto que cumplan con la definición de "hecho". Los equipos de desarrollo son multifuncionales y responsables de todos los aspectos del trabajo.

Scrum proporciona un marco sólido para la colaboración efectiva y la entrega continua de valor en proyectos de desarrollo de software y otros campos. Su estructura flexible y su enfoque en la mejora continua lo convierten en una elección popular para equipos que buscan adaptarse rápidamente a los cambios y maximizar su eficiencia.

Cabe destacar que también existen metodologías centradas en la gestión de servicios como ITIL (*ITIL Foundation: ITIL 4 Edition*, 2019), que se centran en la alineación de los servicios de TI con las necesidades del negocio y en la mejora continua de los procesos. Estas metodologías pueden ser útiles posteriormente para el mantenimiento del proyecto. De hecho, muchas empresas optan por combinar lo mejor de ambas metodologías, un enfoque conocido como Agile Service Management (TOPdesk, 2024).

Capítulo 5

Diseño y Desarrollo

En este capítulo se explicará el curso del desarrollo y diseño del proyecto, siguiendo una metodología lo más parecida posible a SCRUM, ampliamente abordada en asignaturas como Desarrollo y Gestión de Proyectos (DGP) y Metodologías del Desarrollo Ágil (MDA). SCRUM, que se basa en sprints, ha sido la estructura utilizada para dividir el desarrollo del proyecto. A lo largo de los siguientes apartados, los sprints se agrupan en función de los cambios significativos realizados en el proyecto, el cual comprende un total de 9 sprints, siendo los 5 primeros de 2 semanas y los 4 restantes de 1 semana cada uno.

5.1. Inicio: Sprint 0

5.1.1. Análisis del Problema

El equipo de investigación necesita una aplicación para mostrar datos en un panel informativo. Requieren que la aplicación se conecte automáticamente a la API de la empresa Innovasur, cargue los datos y los muestre. Además, necesitan ser capaces de filtrar estos datos al mostrarlos.

Es necesario recopilar datos de dos tipos de sensores: las cámaras de aforo y las cámaras LPR. Las cámaras LPR proporcionan información sobre la matrícula del vehículo, el momento en que este pasó y el sentido de la marcha. Por otro lado, las cámaras de control de aforo registran el momento del reporte y un contador que indica la cantidad de personas que han cruzado una determinada frontera. Estas cámaras de aforo se encargan exclusivamente de monitorizar el flujo de personas que atraviesan una frontera y de mantener un registro de cuántas personas hay en una zona específica, sumando a un contador según el tipo de evento, ya sea entrada o salida.

Se recogieron los requisitos funcionales con la finalidad de crear historias

de usuario para poder comprobar si la solución planteada cumple con los problemas presentados por el equipo de investigación.

5.1.2. Requisitos

Los requisitos funcionales del sistema quedan recogidos en la siguiente tabla 5.1.

Requisito Funcional	Descripción
RF1	Gestion de Dashboards
RF1.1	Creacion de Dashboards
RF1.2	Eliminar Dashboard
RF1.3	Modificar Dashboard
RF1.4	Visualizar Dashboard
RF1.5	Gestion de tablas en los Dashboards
RF1.5.1	Creacion de tablas
RF1.5.2	Eliminacion de tablas
RF1.5.3	Modificacion de tablas
RF1.5.4	Asociacion de una tabla a un dashboard
RF1.5.5	Filtrado de datos en una tabla
RF1.5.5.1	Filtrado por codigo postal
RF1.5.5.2	Filtrado por la matricula del vehiculo
RF2	Carga de datos automatica desde la API
RF2.1	Extraccion de los datos
RF2.2	Transformacion de los datos
RF2.3	Carga de los datos en la base de datos
RF3	Permisos de vision y edicion para los paneles (Dashboards)
RF3.1	Gestion de usuarios
RF3.1.1	Crear usuarios
RF3.1.2	Modificar usuarios
RF3.1.3	Borrar usuarios
RF3.2	Gestion de perimsos para usuarios usando roles
RF3.2.1	Crear roles
RF3.2.2	Borrar roles
RF3.2.3	Modificar roles
RF3.2.4	Asignar roles a los usuarios
RF3.2.5	Asignar roles a los paneles

Cuadro 5.1: Tabla con los requisitos funcionales del proyecto

y en la tabla 5.2 quedan recogidos los requisitos no funcionales:

Identificador	Requisito
RNF1	Seguridad cifrando los discos donde estarán almacenados datos personales
RNF2	Solo los administradores pueden configurar los dashboard
RNF3	Existirán usuarios que solo puedan ver determinados dashboards
RNF4	Compartimentalización en contenedores para el uso de kubernetes
RNF5	Utilización de kubernetes para mejor disponibilidad del sistema y reparto de carga

Cuadro 5.2: Tabla con los requisitos no funcionales del proyecto

De donde se han sacado las historias de usuario que se muestran en la tabla 5.3.

5.1.3. Propuesta Inicial

Se planteo una implementación utilizando InfluxDB junto con el panel de monitorización de Grafana. Para cargar los datos en la base de datos InfluxDB se utilizaría Telegraf con *addons* específicos. Se seguiría el flujo de datos representado en la figura.

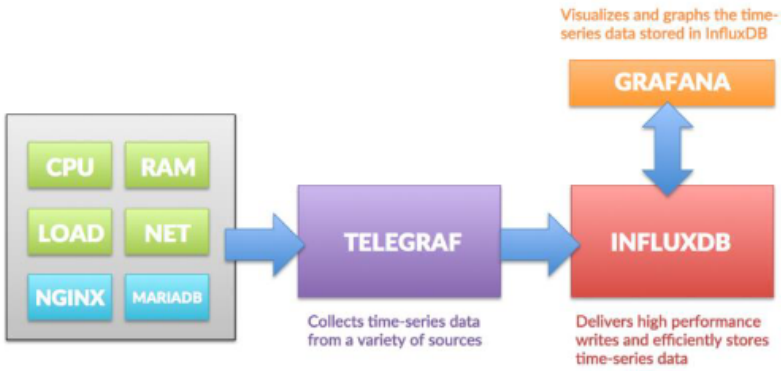

Figura 5.1: Flujo de datos de un *stack* construido por Telegraf, InfluxDB y Grafana. (InfluxData, 2020)

Para la representación de los datos se utilizaría Grafana, que tiene compatibilidad nativa con InfluxDB v2. También cabe destacar que InfluxDB v2 tiene un panel integrado desde donde se puede acceder a su configuración, visualización de variables y control de permisos. También todos estos componentes funcionarían utilizando docker-compose para aprovechar sus beneficios. La selección inicial se fundamentó en la utilización de estos componentes en la asignatura de Centros de Procesamiento de Datos y el énfasis en series de datos temporales.

Pero tras el intento de genererar una prueba de concepto para este sistema se vio que era muy complicado utilizar el lenguaje de la base de datos (flux) para realizar consultas complejas requeridas por el equipo de desarrollo. Luego se opto por profundizar mas en la investigación de otras altenativas para la base de datos junto con alternativas para el software para la muestra del panel y el software necesario para la carga de datos.

Durante este periodo de pruebas, se decidió descartar Grafana como pa-

nel de información debido a que su *look and feel* no convencía al equipo de desarrollo. Además, durante la configuración y carga de datos de prueba en InfluxDB v2 se encontraron grandes dificultades para conseguir la funcionalidad deseada por el equipo de desarrollo. Debido a esto, se descartaron Grafana e InfluxDB como componentes del proyecto y se decidió buscar alternativas tanto para el panel de información como para la base de datos.

Durante la investigación de diferentes componentes, se seleccionaron Apache Superset y TimescaleDB por los motivos mencionados anteriormente. Finalmente se propuso como propuesta inicial una composición de paneles usando Apache Superset y TimescaleDB y para cargar los datos se propuso un cargador hecho en python debido a que aplicaciones como apache airflow utilizan python para la configuracion de sus flujos.

5.1.4. Tareas realizadas

En este sprint se han realizado las tareas mostradas en 5.2 y se han desarrollado temporalemnte según figura 5.4.

Identificador	Tarea
T1	Asistencia a 1ª Ed Taller TIC
T2	Instalación y configuración de docker compose
T3	Instalación y configuración de influxdb (v2)
T4	Instalación y configuración de Grafana
T5	Búsqueda de alternativas y herramientas usadas en mercado
T6	Investigación sobre las bases de datos
T7	Investigar herramientas para ETL
FS0	Final Sprint 0

Cuadro 5.4: Backlog del Sprint 0.

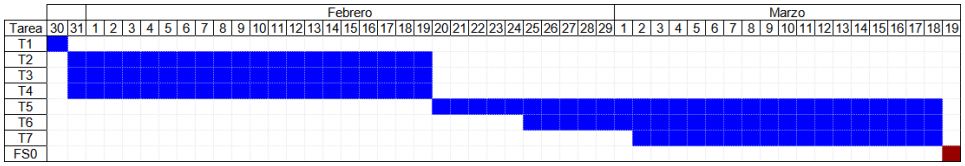

Figura 5.2: Organización temporal del Sprint 0.

5.1.5. Tareas identificadas

Cabe destacar que, debido a la naturaleza de este proyecto, la configuración para el despliegue de Apache Superset cumple con todos los requisitos de las historias de usuario, a excepción de HU-20 y HU-21, para los cuales es necesaria la implementación del cargador automático, además del despliegue

de Apache Superset. Además, a pesar de que el despliegue en Kubernetes es un requisito no funcional, debido a la inexperiencia y a la mala documentación del despliegue de la herramienta Apache Superset en Kubernetes, se decidió comenzar con un despliegue contenerizado de Superset junto con su base de datos, de acuerdo con los conocimientos adquiridos en la asignatura de Centro de Procesamiento de Datos, para poder comprender su funcionamiento y enfrentar cualquier problema que pudiera surgir en su despliegue.

En la tabla 5.5 se relacionan las tareas necesarias para completar cada HU.

Tarea	Historia de usuario
T8, T9, T10, T11, T18	HU-1
T8, T9, T10, T11	HU-2
T8, T9, T10, T11	HU-3
T8, T9, T10, T11	HU-4
T8, T9, T10, T11	HU-5
T8, T9, T10, T11	HU-6
T8, T9, T10, T11	HU-7
T8, T9, T10, T11	HU-8
T8, T9, T10, T11	HU-9
T8, T9, T10, T11	HU-10
T8, T9, T10, T11	HU-11
T8, T9, T10, T11, T12, T13, T14, T15, T16, T17	HU-12
T8, T9, T10, T11, T12, T13, T14, T15, T16, T17, T18	HU-13
T8, T9, T10, T11	HU-14
T8, T9, T10, T11	HU-15
T8, T9, T10, T11	HU-16
T8, T9, T10, T11	HU-17
T8, T9, T10, T11	HU-18
T8, T9, T10, T11	HU-19
T8, T9, T10, T11	HU-20
T8, T9, T10, T11, T18	HU-21

Cuadro 5.5: Tareas necesarias para completar cada HU asociada.

5.1.6. Organización temporal

Se ha realizado una planificación para cuatro sprints, distribuyendo las tareas en cada sprint en función de las tareas previas necesarias y su estimación de coste temporal. Debido a la necesidad de utilizar nuevas tecnologías desconocidas, no se pudo completar la planificación hasta el final del sprint 0; por ello, la figura 5.2 muestra las tareas ya realizadas en azul oscuro a diferencia de los siguientes diagramas de gantt que muestran las tareas planificadas

Sprint 1

Identificador	Tarea	Duración	Fecha inicio	Fecha Fin
T8	Instalación y configuración Superset (docker compose)	14	19/03/24	01/04/24
T9	Instalación y configuración Timescaledb (docker compose)	7	02/04/24	08/04/24
T10	Crear dashboard datos arreglados	7	09/04/24	15/04/24
FS1	Final Sprint 1	1	16/04/24	16/04/24

Cuadro 5.6: Tareas planificadas en el sprint 1. Se han creado artefactos precedios por FS en el identificador para indicar el final de un sprint. Los identificadores con prefijo T se utilizan para las tareas.

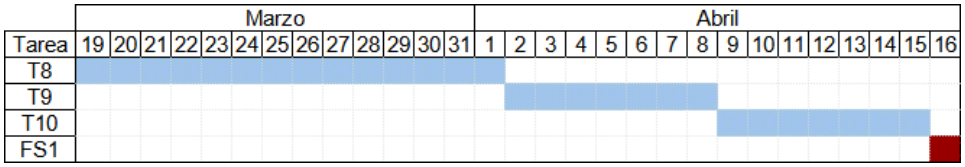

Figura 5.3: Planificación del sprint 1.

Sprint 2

Identificador	Tarea	Duración	Fecha inicio	Fecha Fin
T11	Configuración y despliegue del entorno en kubernetes	21	16/04/24	06/05/24
T12	Implementación de ETL en python	7	07/05/24	13/05/24
FS2	Final Sprint 2	1	14/05/24	14/05/24

Cuadro 5.7: Tareas asignadas al Sprint 2. El prefijo FS se usa para indicar un artefacto que marca la finalización del sprint. El prefijo T marca las tareas.

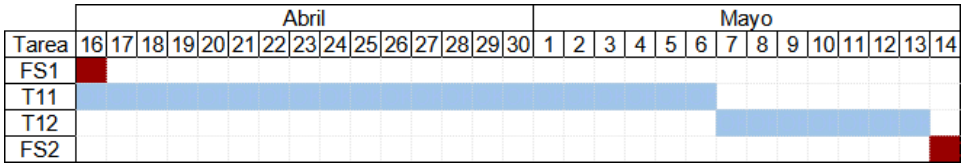

Figura 5.4: Planificación del Sprint 2. Tas tareas planificadas se muestran en azul claro y los artefactos que indican la finalización de los sprints se muestran en rojo.

Sprint 3

Identificador	Tarea	Duración	Fecha inicio	Fecha Fin
T13	Configuración para despliegue de ETL en kubernetes	15	14/05/24	28/05/24
T14	Gestión de autenticación del ETL	5	29/05/24	02/06/24
T15	Adaptación a la API	3	03/06/24	05/06/24
T16	Creación de Dashboard de monitorización para trafico	1	06/06/24	06/06/24
T17	Creación de Dashboard de monitorización para aforo	1	07/06/24	07/06/24
T18	Creación de usuario para ver y no editar	1	08/06/24	08/06/24
T19	Investigación y despliegue de Prophet para predicciones	2	09/06/24	10/06/24
FS3	Final Sprint 3	1	11/06/24	11/06/24

Cuadro 5.8: Tareas asignadas al Sprint 3. El prefijo FS se usa para indicar un artefacto que marca la finalización del sprint. El prefijo T marca las tareas.

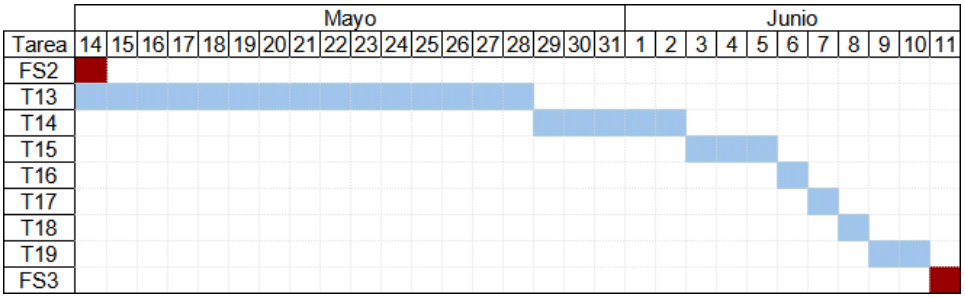

Figura 5.5: Organización temporal del Sprint 3. Las tareas planificadas se muestran en azul claro y los artefactos que indican la finalización de los sprints se muestran en rojo.

Sprint 4

Identificador	Tarea	Duración	Fecha inicio	Fecha Fin
T20	Elaboración de documentación de despliegue	7	11/06/24	17/06/24
T21	Elaboración del manual de usuario	7	18/06/24	24/06/24
FS4	Final Sprint 4	1	24/06/24	24/06/24

Cuadro 5.9: Tareas asignadas al Sprint 4. El prefijo FS se usa para indicar un artefacto que marca la finalización del sprint. El prefijo T marca las tareas.

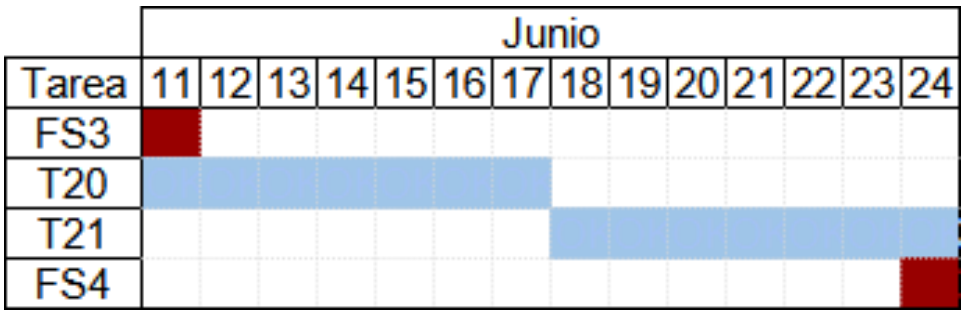

Figura 5.6: Organización temporal del Sprint 4. Las tareas planificadas se muestran en azul claro y los artefactos que indican la finalización de los sprints se muestran en rojo.

5.2. Transcurso del Sprint 1

En este sprint se planteó la configuración y se desplegó un entorno Docker Compose con contenedores de Apache Superset y TimescaleDB, junto con la creación de un panel para un conjunto de datos de ejemplo. Durante este sprint no se tuvo en cuenta el requisito no funcional de Kubernetes debido a su gran complejidad, ya que se trataba de una nueva tecnología desconocida. Esto fue planificado desde el principio y se implementó posteriormente.

Identificador	Horas Totales Estimadas	Horas Totales Reales	Horas extra
T8	56	72	16,00
T9	28	20	-8,00
T10	28	28	0,00

Cuadro 5.10: Se presentan las horas reales, las horas estimadas y las horas extra que se han tenido que invertir por cada tarea en el primer sprint.

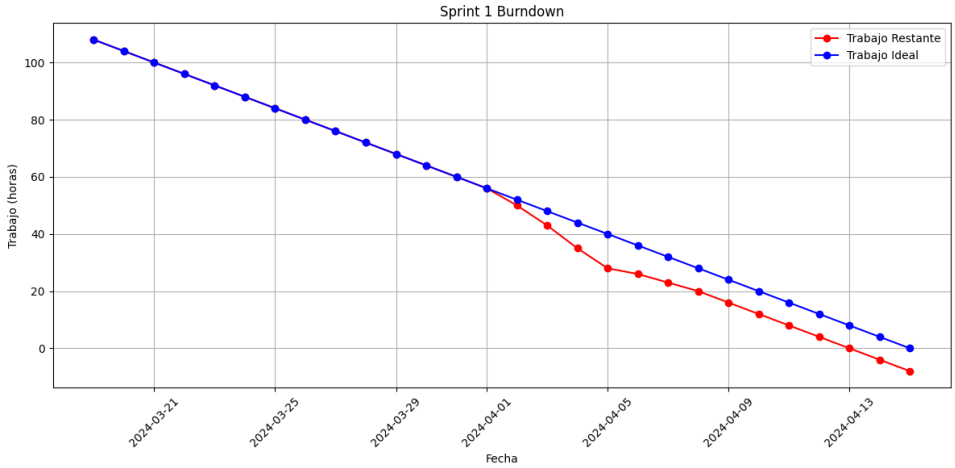

Figura 5.7: Se presenta el burndown del primer sprint, en el que se muestra en azul la carga de trabajo total planificada en horas y en rojo la carga de trabajo real.

Debido a errores inesperados en la configuración de Apache Superset, la tarea T8 tomó más tiempo del planificado. Por el contrario, la configuración de TimescaleDB (T9) llevó menos tiempo del esperado. En total, se emplearon 8 horas más de lo inicialmente planificado.

5.3. Transcurso del Sprint 2

En este sprint se llevó a cabo la transición de la configuración previa a Kubernetes. Con el fin de aprovechar mejor las capacidades de Kubernetes, se cambió el stack tecnológico y se utilizó PostgreSQL en lugar de TimescaleDB. Además, se añadieron los componentes restantes del despliegue de Apache Superset para Kubernetes, incluyendo Redis y Celery para los workers. También se desarrolló la funcionalidad básica de carga de datos a través de un programa en Python.

Identificador	Horas Totales Estimadas	Horas Totales Reales	Horas extra
T11	84	100	16,00
T12	28	16	-12,00

Cuadro 5.11: Se presentan las horas reales, las horas estimadas y las horas extra que se han tenido que invertir por cada tarea en el segundo sprint.

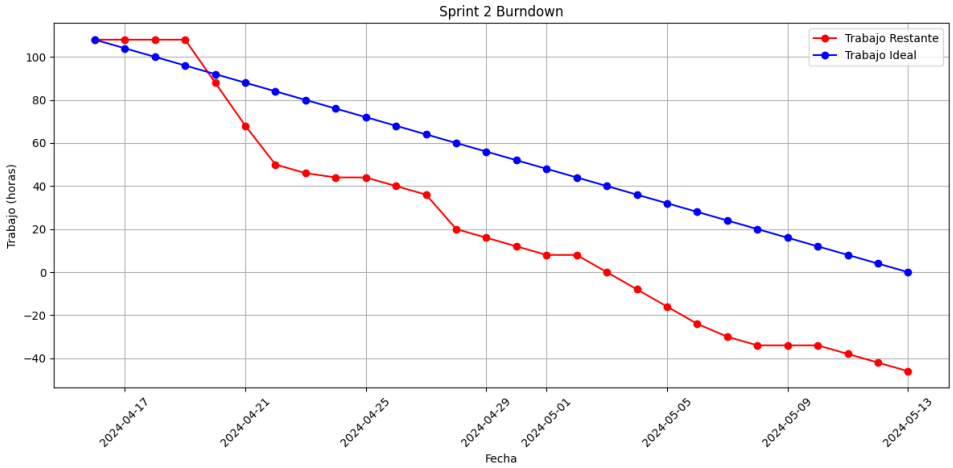

Figura 5.8: Se presenta el burndown del segundo sprint, en el que se muestra en azul la carga de trabajo total planificada en horas y en rojo la carga de trabajo real.

Debido a la mala documentación, la complejidad de configuración y al hecho de ser una tecnología nueva para el equipo de desarrollo, la configuración de Kubernetes y el despliegue de la aplicación (T11) tomaron mucho más tiempo del esperado. Por otro lado, la implementación de la funcionalidad básica para el cargador (T12) llevó menos tiempo del previsto. Como resultado, el sprint tardó 46 horas más de lo inicialmente planificado.

5.4. Transcurso del Sprint 3

Se desplegó el contenedor destinado a la ejecución del script ETL en el clúster, asegurando que este se ejecutara de manera periódica. Además, se mejoró su funcionalidad. Durante este sprint, también se desarrollaron los paneles para la visualización de los datos cargados.

Identificador	Horas Totales Estimadas	Horas Totales Reales	Horas extra
T13	60	74	14,00
T14	20	20	0,00
T15	12	12	0,00
T16	4	4	0,00
T17	4	4	0,00
T18	4	4	0,00
T19	8	8	0,00

Cuadro 5.12: Se presentan las horas reales, las horas estimadas y las horas extra que se han tenido que invertir por cada tarea en el tercer sprint.

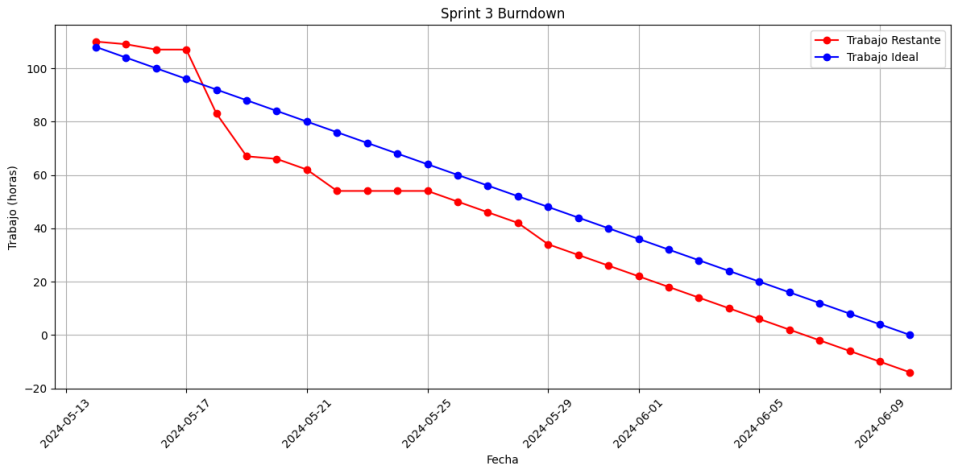

Figura 5.9: Se presenta el burndown del tercer sprint, en el que se muestra en azul la carga de trabajo total planificada en horas y en rojo la carga de trabajo real.

Dado que el equipo carecía de experiencia en el desarrollo de componentes en un entorno Kubernetes, se incrementó significativamente el tiempo requerido para llevar a cabo cada prueba del desarrollo. Esto, a su vez, prolongó el tiempo necesario para completar la tarea (T13), resultando en un aumento de 14 horas en la duración del sprint.

5.5. Transcurso del Sprint 4

En este sprint se finalizo la redacción de la guía del despliegue y del usuario.

Identificador	Horas Totales Estimadas	Horas Totales Reales	Horas extra
T20	28	36	8,00
T21	28	20	-8,00

Cuadro 5.13: Se presentan las horas reales, las horas estimadas y las horas extra que se han tenido que invertir por cada tarea en el cuarto sprint.

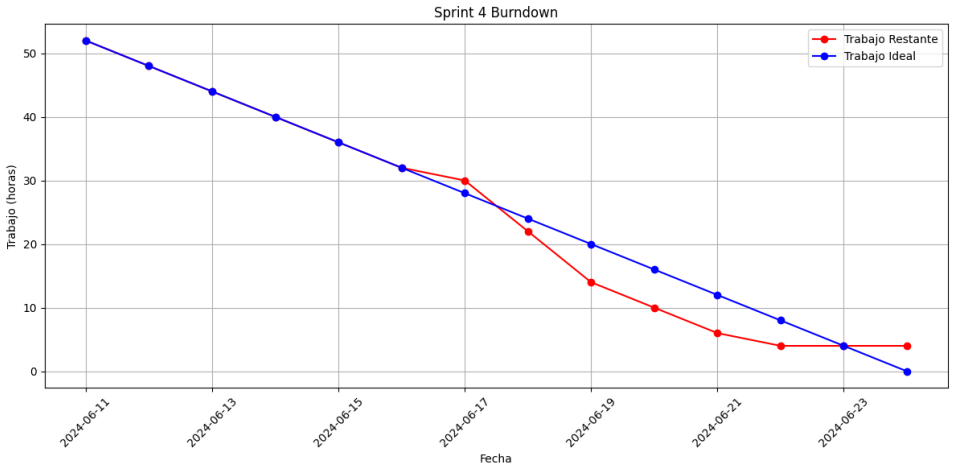

Figura 5.10: Se presenta el burndown del cuarto sprint, en el que se muestra en azul la carga de trabajo total planificada en horas y en rojo la carga de trabajo real.

5.6. Propuesta Final

5.6.1. Almacenamiento de Datos

La estructura del paquete de despliegue de Superset en Kubernetes requiere varias bases de datos. En primer lugar, se necesita una base de datos Redis (distribución de Bitnami adaptada para su despliegue en Kubernetes), que es muy rápida y opera solo en memoria. Esta base de datos es utilizada por Flask, el componente de Superset encargado de la autenticación, para almacenar las sesiones activas y la caché, y por Celery como broker de mensajes. En segundo lugar, se requiere una base de datos para almacenar los datos que se mostrarán en los diferentes paneles; en este caso, se ha decidido utilizar PostgreSQL, que es la opción predeterminada en la configuración de Superset (esto es posible dado que TimescaleDB se basa en PostgreSQL). Adicionalmente, Apache Superset incluye internamente una base de datos

SQLite, cuyo uso está desaconsejado y se reserva principalmente para el funcionamiento interno de la herramienta.

El despliegue de estas bases de datos se realiza mediante Helm, un gestor de paquetes que, al desplegar Superset, requiere la configuración previa de estas bases de datos si se desea utilizarlas.

Sobre la imagen de Superset, se ha configurado un PVC (Persistent Volume Claim) donde los PV (Persistent Volumes) se han configurado en los nodos y están en discos duros cifrados para cumplir con la ley de protección de datos. Estos discos están montados en máquinas locales siguiendo la estructura de montaje mostrada en la figura5.11 y están cifrados utilizando 'cryptsetup'. Según se ha dado en la asignatura Ingeniería de Servidores.

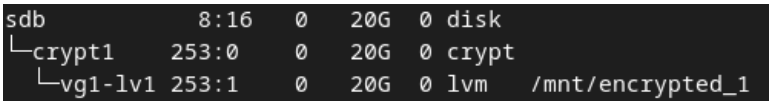

Figura 5.11: Estructura de cifrado utilizada en el proyecto. Esta estructura se basa en conceptos y herramientas de la asignatura de Ingeniería de Servidores (ISE).

5.6.2. Funcionamiento del cargador para ETL

Se ha implementado un cargador por lotes para ETL (Extract, Transform, Load) mediante un contenedor Python Slim con un script de Python que se ejecuta en el contenedor.

Este contenedor se conecta al endpoint de Innovasur para autenticarse. Primero, verifica si hay un token guardado en la base de datos. En caso de que lo haya, realiza una petición sin parámetros al endpoint de datos. Si recibe un error indicando que el token está expirado, o si no tiene token, se autentica y guarda el nuevo token en la base de datos. Si la autenticación es exitosa, continúa con el proceso. Si recibe algún otro tipo de error, lo registra en el log y detiene su ejecución.

A continuación, se procede a descargar los datos utilizando el endpoint de datos. Este script verifica en la base de datos cuál es la última y la primera fecha registradas, y si no existen, crea las tablas correspondientes. El proceso de descarga se realiza en lotes de cinco meses, obteniendo las fechas más antiguas y las más recientes mediante peticiones a la API. Este proceso continúa hasta que la API devuelve algún tipo de error. Según las pruebas realizadas, un error 500 suele indicar que ya no hay más datos que cargar, pero a veces, sin una explicación clara, la API devuelve otros errores; en estos casos, la carga en esa dirección (hacia atrás en el tiempo o hacia adelante) se detiene.

Además, la API para el endpoint que devuelve los datos de aforo, aunque no haya datos de los contadores, la API devuelve filas por horas de días futuros (esto se ha tenido que mitigar y comprobar en el ETL). En los ETL se han desarrollado dos versiones: una para los datos de aforo y otra para los datos de tráfico, debido a las diferencias en el comportamiento de cada endpoint y que la carga se verifica en el insert para las columnas planificadas. En caso de que la API devuelva otras columnas, esto debería resultar en un error y no cargará esos datos para evitar un estado incorrecto en la tabla. Los datos se insertan en la base de datos PostgreSQL. De esta manera, en caso de que la API devuelva datos incompletos en alguna ocasión, se pueden borrar todos los datos desde el punto donde se realizó una carga incorrecta hacia adelante o hacia atrás, y ejecutando más veces el ETL se logrará cargar todos los datos correctamente.

Se ejecutan dos contenedores con parámetros específicos: uno para el tráfico y otro para el aforo. El contenedor de tráfico se ejecuta cada 7 minutos, una frecuencia seleccionada porque los registros se actualizan cada vez que pasa un vehículo, aunque el retraso varía según la cámara LPR (Lectura de Matrículas). La cámara con mayor retraso tiene un desfase de 2 horas, mientras que la de menor retraso tiene uno de 30 minutos. Elegimos un intervalo de 7 minutos de manera arbitraria para evitar saturar la red con un tiempo demasiado corto y, al mismo tiempo, mantener la base de datos lo más actualizada posible, dado que la API es inestable y podría dejar de enviar datos o devolver respuestas incorrectas. Por otro lado, el sensor de aforo se ejecuta cada 30 minutos, ya que la actualización de los datos de aforo ocurre una vez por hora de manera sincronizada entre todos los sensores. La frecuencia de 30 minutos permite una actualización intermedia por si la API no devuelve datos debido a su inestabilidad. Estos contenedores se ejecutan mediante cronjobs, que inician un pod con un contenedor que contiene un entorno Python ligero y el script correspondiente. Este enfoque es eficiente, ya que al finalizar la ejecución, el pod se elimina y los recursos se liberan.

5.6.3. Uso y Funcionamiento

Todo el sistema se está ejecutando en un ordenador personal, donde todos los nodos que operan en el clúster de Kubernetes están virtualizados, al igual que el control externo, que corresponde a un PC en la misma red que el clúster, pero independiente de este. Todas las máquinas están virtualizadas en el host mediante VirtualBox. Como se puede observar en la figura 5.12, la red `natnetwork1` conecta las máquinas; esta es una red con NAT y DHCP habilitados, lo que permite al clúster tener acceso a Internet.

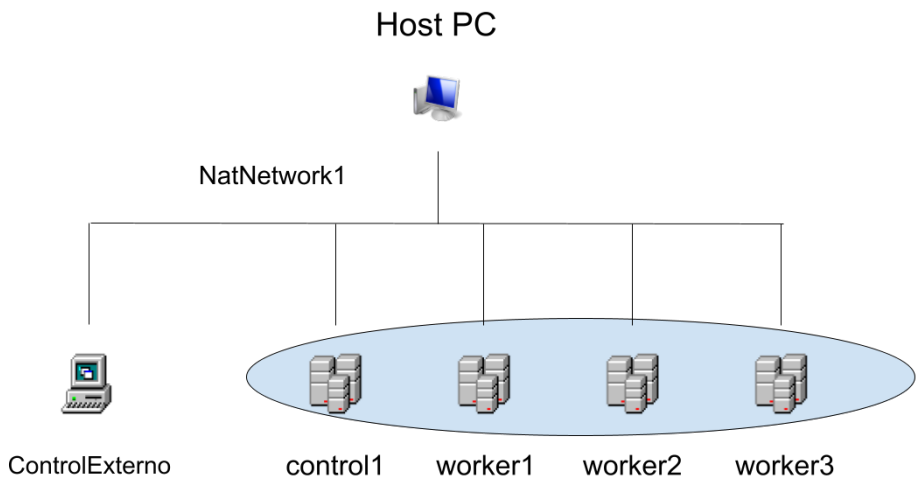

Figura 5.12: En esta figura, el óvalo azul agrupa las máquinas que están en el clúster de Kubernetes. `natnetwork1` es la red que las conecta, una red con NAT y DHCP habilitados, lo que permite al clúster tener acceso a Internet.

La base de datos utiliza una única tabla para cada serie de datos temporales. Junto con otras tablas, se pueden generar vistas para crear consultas más complejas que se actualizan automáticamente. De esta manera, en Apache Superset se pueden utilizar estas vistas como fuente de datos.

Superset incluye un asistente basado en SQLAlchemy para manejar consultas SQL directamente, como se muestra en la Figura 5.13

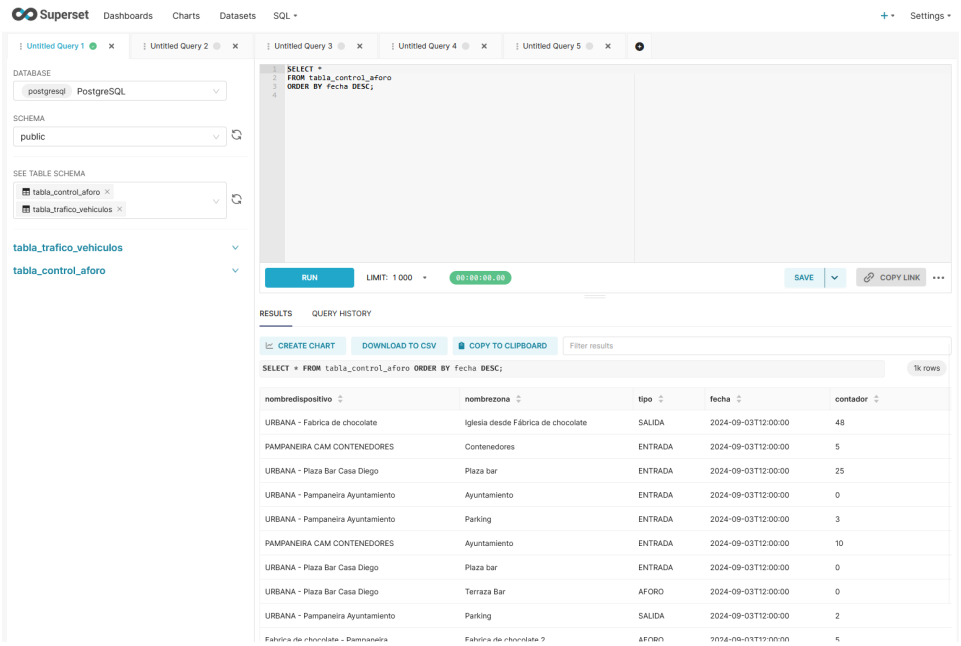

Figura 5.13: Muestra del asistente de consultas SQL

Además, es posible cargar datos y crear tablas directamente en Apache Superset mediante sentencias SQL. Superset también incorpora un asistente para la importación de datos desde archivos CSV, como se puede observar en la Figura 5.14

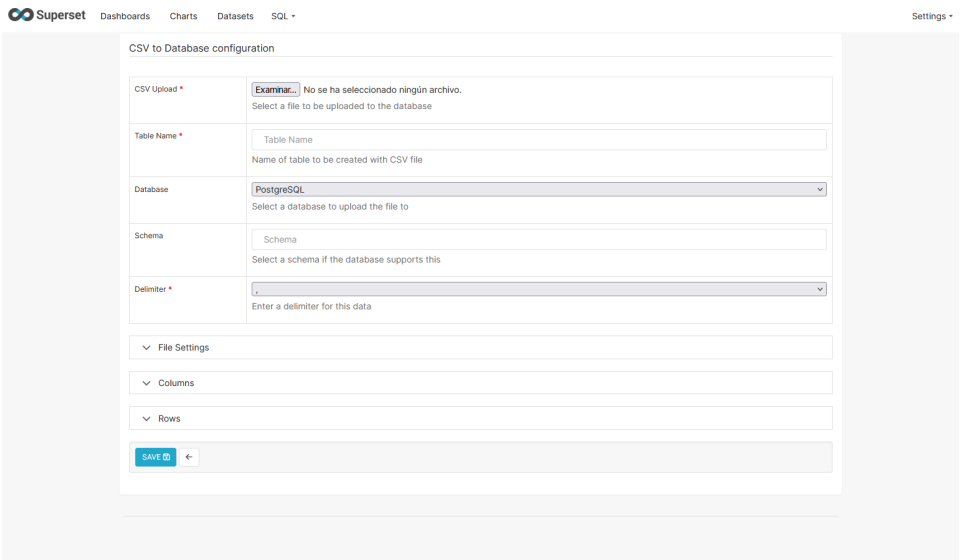

Figura 5.14: Muestra del asistente de importación desde csv

El diseño de los paneles de datos en Apache Superset está orientado a mostrar de manera clara y eficiente la información recibida de los sensores. Cada panel se crea con un propósito específico. Por ejemplo, el Panel de Tránsito General se utiliza para mostrar los datos sobre el tránsito de vehículos en diferentes zonas, lo cual es fundamental para entender los patrones de tráfico y gestionar los recursos de manera adecuada. Este panel se puede observar en la Figura 5.15.

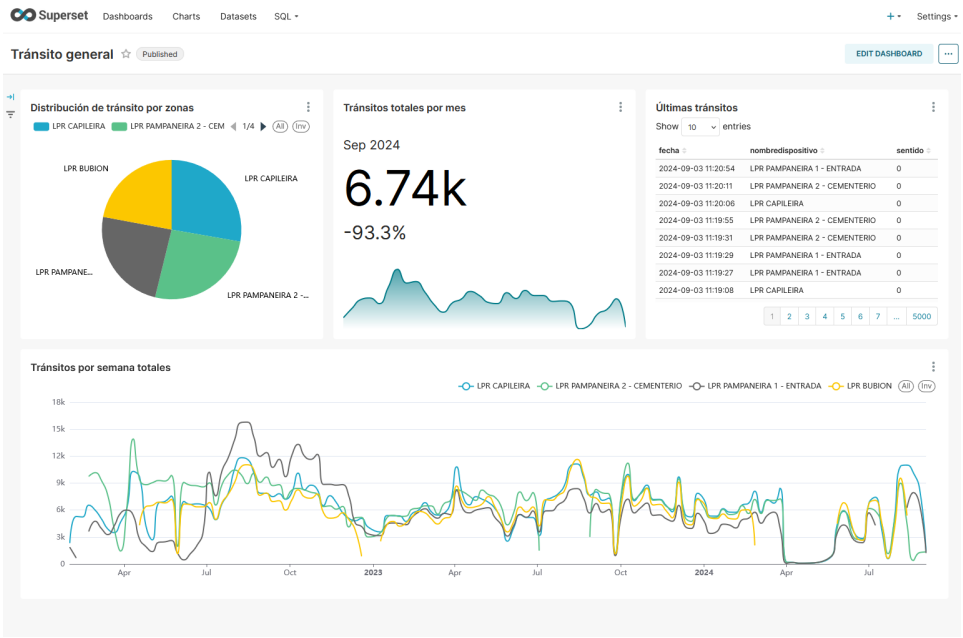

Figura 5.15: Muestra del panel desarrollado para mostrar los datos de tránsito

En el Panel de Tránsito General, se incluyen indicadores cuantitativos que permiten un análisis detallado de los datos. Un aspecto clave es la distribución del tránsito por zonas, que mide los tránsitos totales agrupados por cada dispositivo. Esto ayuda a identificar si hay significativamente más tránsito en una cámara en comparación con las demás, lo que podría indicar un uso desigual de las rutas o la necesidad de ajustes en la infraestructura. Además, se presentan los tránsitos totales por mes, proporcionando una visión clara de cómo varía el número de tránsitos de un mes a otro, lo que es útil para la planificación y análisis a largo plazo.

Otra funcionalidad esencial es el indicador de últimos tránsitos, que muestra una tabla detallada donde cada fila representa un tránsito y cada columna ofrece información relevante sobre cada evento. Esto es crucial para monitorear las operaciones diarias y responder rápidamente a cualquier anomalía o patrón inesperado. También se incluye un análisis histórico a

través del indicador de tránsitos totales por semana, que permite observar tendencias a lo largo del tiempo y detectar cambios significativos en el comportamiento del tránsito, así como identificar periodos en los que alguna cámara podría no haber estado operativa.

Para complementar el análisis, se ha desarrollado un panel adicional que se centra específicamente en los datos de tránsito de las últimas 24 horas. Este panel, que se muestra en la Figura 5.16, proporciona una vista más granular de la actividad reciente, facilitando la identificación rápida de problemas como cámaras caídas o variaciones abruptas en el flujo de tránsito, que pueden requerir atención inmediata.

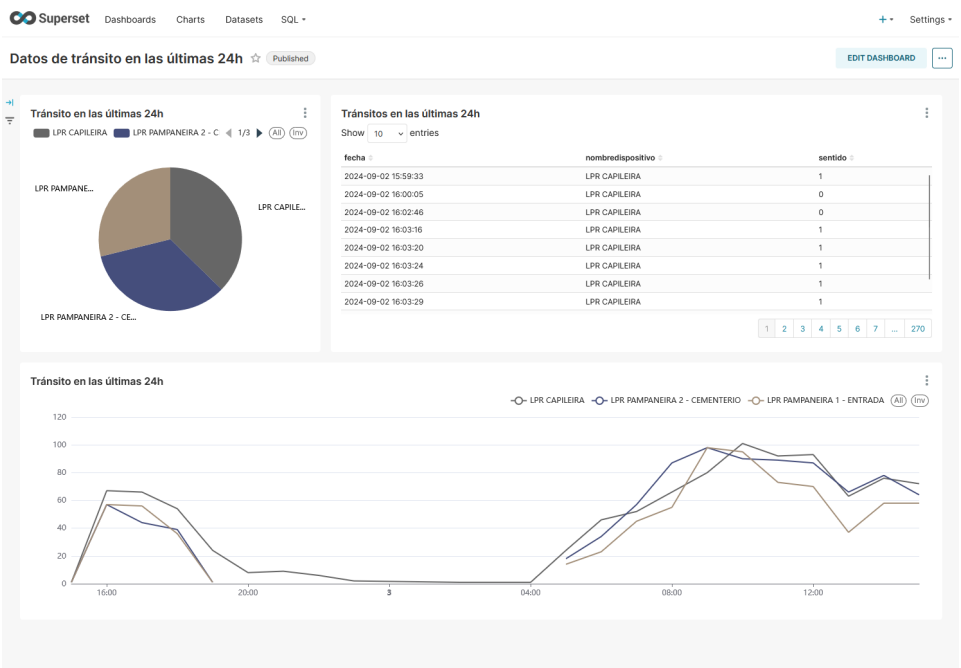

Figura 5.16: Vista del panel de las últimas 24 horas

La Figura 5.17 muestra el panel diseñado para visualizar los datos de aforo en diversas zonas. Este panel es fundamental para monitorear y gestionar la capacidad de ocupación de diferentes áreas, permitiendo una visión detallada del flujo de personas y su distribución.

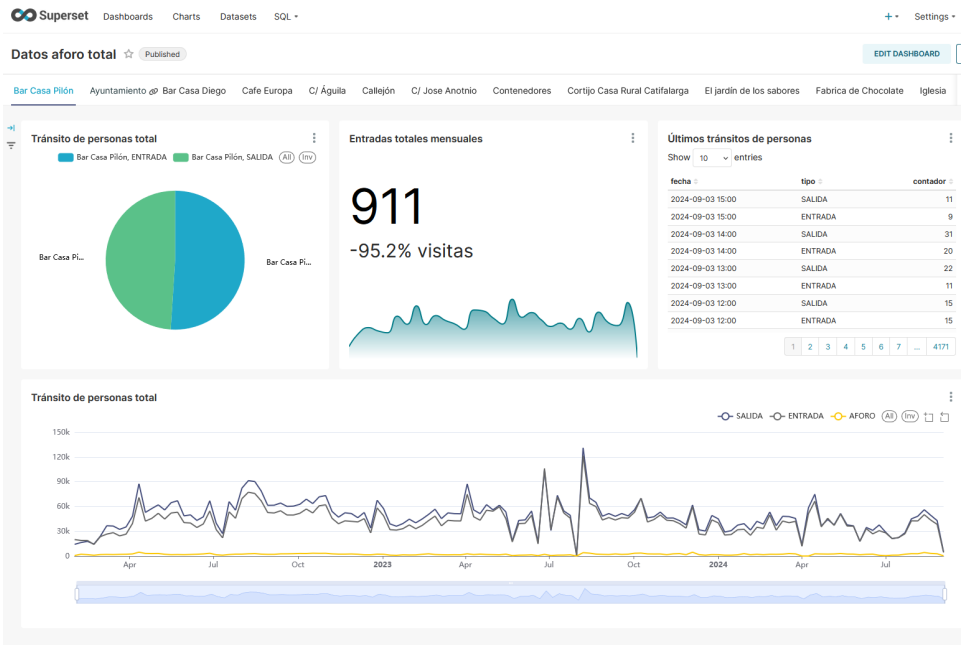

Figura 5.17: Se muestra el panel para el aforo de las diversas zonas

En este panel, uno de los indicadores clave es el tránsito de personas total, que mide el número de personas que pasan por una zona de salida en comparación con la entrada. Este indicador es crucial para entender si una zona está siendo más utilizada de lo esperado, al mostrar claramente si el número de personas que ingresan es superior al de las que salen.

Otro indicador relevante es el de entradas totales mensuales, que presenta la suma total del conteo de entradas por mes. Este indicador ayuda a identificar si una zona en particular está experimentando un aumento en el número de visitantes, comparando el número de entradas actuales con los datos de los diez meses previos. Esto permite detectar tendencias, así como aumentos o disminuciones significativas en la afluencia de personas en un periodo determinado.

Además, el panel incluye un indicador denominado últimos tránsitos de personas, que muestra las actualizaciones más recientes en forma de tabla. Esta función permite un seguimiento preciso de cuántas personas han entrado y a qué hora, proporcionando una visión inmediata de la actividad actual en cada zona. Esta información es esencial para la toma de decisiones en tiempo real, especialmente en situaciones que requieren una respuesta rápida, como la gestión de emergencias o eventos especiales.

En el siguiente enlace esta disponible el código del proyecto.

RF	Descripción	Historia de Usuario
RF1.4	como usuario quiero poder ver mis paneles asociados con información	HU-1
RF1.1	como administrador quiero poder crear paneles	HU-2
RF1.2	como administrador quiero poder eliminar cualquier panel	HU-3
RF1.3	como administrador quiero poder modificar cualquier panel	HU-4
RF1.4	como administrador quiero poder ver todos los paneles	HU-5
RF1.5.1	como administrador quiero poder crear tablas	HU-6
RF1.5.2	como administrador quiero poder eliminar tablas	HU-7
RF1.5.3	como administrador quiero poder modificar cualquier tabla	HU-8
RF1.5.4	como administrador quiero poder asociar tablas a cualquier dashboard	HU-9
RF1.5.5.2	como administrador quiero poder filtrar los datos de la tabla por matricula del vehículo	HU-10
RF1.5.5.2	como administrador quiero poder filtrar los datos de la tabla por código postal	HU-11
RF2	como usuario quiero ver los datos extraídos de la API en una tabla básica	HU-12
RF2	como administrador quiero ver los datos extraídos de la API en una tabla básica	HU-13
RF3.1.1	como administrador quiero poder crear usuarios	HU-14
RF3.1.2	como administrador quiero poder borrar usuarios	HU-15
RF3.1.3	como administrador quiero poder modificar los usuarios	HU-16
RF3.2.1	como administrador quiero poder crear roles	HU-17
RF3.2.2	como administrador quiero poder borrar roles	HU-18
RF3.2.3	como administrador quiero poder modificar los roles	HU-19
RF3.2.4	como administrador quiero poder asignar roles a los usuarios	HU-20
RF3.2.5	como administrador quiero poder asignar paneles para que solo determinados usuarios puedan verlos	HU-21

Cuadro 5.3: Tabla con las historias de usuario.

Capítulo 6

Conclusiones y trabajo futuro

En esta sección se darán las conclusiones del trabajo realizado y se discutirá el desarrollo futuro que podría tener este proyecto o derivar de él.

6.1. Conclusión

En este proyecto se ha desarrollado una herramienta de recolección, visualización y análisis de datos de una smart village. Se ha desarrollado una solución acorde con un enfoque didáctico que fomenta el uso de herramientas y tecnologías contemporáneas, así como de software libre y de código abierto, en línea con los objetivos de aprendizaje establecidos. Este enfoque no solo busca resolver el problema planteado, sino también ofrecer una experiencia de aprendizaje enriquecedora mediante la aplicación práctica de conocimientos teóricos y la implementación de nuevas herramientas.

En primer lugar, se evaluaron diversas herramientas para la gestión de bases de datos y dashboards, seleccionando TimescaleDB como base de datos de almacenamiento y Superset para la visualización de datos en los dashboards. Inicialmente, se diseñó y desplegó un servicio basado en contenedores utilizando Docker Compose, con el fin de mostrar el dashboard y almacenar los datos.

Además, se desarrolló un mecanismo de extracción y tratamiento de datos (ETL) implementado en Python. Para ejecutar este ETL, se utilizó una imagen de Python Slim, contenerizada con Docker.

Posteriormente, se estudió el uso de Kubernetes y se reconfiguró la solución utilizando despliegues, con el objetivo de aprovechar las mejoras en rendimiento y disponibilidad que Kubernetes ofrece en comparación con

la implementación inicial. Este cambio requirió sustituir TimescaleDB por PostgreSQL, una decisión que había sido anticipada durante la planificación, considerando la flexibilidad y escalabilidad necesarias para el proyecto.

La mayor dificultad encontrada durante el desarrollo de este proyecto ha sido la identificación y selección de las herramientas adecuadas, que no solo sean comúnmente utilizadas en la industria, sino que también sean relevantes y aplicables a nuestro caso específico. Además, ha sido un desafío significativo adaptar y utilizar estas herramientas de manera efectiva dentro del contexto de nuestro proyecto.

A través de este proceso, se ha adquirido una comprensión profunda de diversas tecnologías, lo que ha permitido no solo cumplir con los objetivos del proyecto, sino también enriquecer el conocimiento práctico y teórico en el campo de la ingeniería de software y la gestión de proyectos.

6.2. Trabajo futuro y explotación económica

Debido a los problemas y al comportamiento inesperado de la API de Innovasur, se podría mejorar el cargador de datos para que fuera más inteligente, realizando comprobaciones temporales antes y después de la carga de datos. Esto permitiría verificar que la API ha enviado los datos solicitados o, en caso de cambios en las columnas o de falta de ciertos datos, almacenarlos en una nueva tabla automáticamente. Además, se podrían implementar avisos específicos según la situación con la API o con los datos. Para optimizar aún más, se podría considerar la adaptación de este ETL a un gestor de flujos de trabajo más avanzado, como Argo Workflows o Apache Airflow.

Otra mejora sería la integración de un gran modelo de lenguaje (LLM) para la creación de tablas y la gestión general de Apache Superset, lo que permitiría una administración más eficiente y automatizada. Esta funcionalidad ya está disponible en la versión de pago de Apache Superset ofrecida por Prophet.

En cuanto al manejo de series de datos temporales, se podría mejorar utilizando la distribución HA de TimescaleDB en el clúster en lugar de la actual distribución de Bitnami de PostgreSQL. Esto optimizaría la gestión y el rendimiento de los datos.

La implementación de una API Gateway sería beneficiosa para desarrollar soluciones en nuestro clúster que no requieran autenticación explícita para cada aplicación. De esta forma, podríamos ofrecer servicios a través de una API, similar a cómo Innovasur nos ofrece los suyos, y utilizar los webhooks de Apache Superset para integrar tablas o paneles completos en otras páginas web.

En términos de seguridad, aunque actualmente se utiliza un certificado TLS autofirmado, sería posible desplegar el clúster en otro entorno y obtener un certificado gratuito de Let's Encrypt. Let's Encrypt, con su gran comunidad de soporte y aplicaciones como Cert-manager, permitiría la renovación automática del certificado, mejorando la seguridad del clúster.

Para la monitorización del clúster y de los logs, la instalación de Grafana y Prometheus sería una solución ideal. Apache Superset ya está preparado para la integración con Prometheus, lo que facilitaría la supervisión de su uso y de los datos de registro. Asimismo, Nginx podría ser monitorizado con Prometheus, proporcionando una visión completa del rendimiento y del estado del clúster.

Finalmente, las posibles explotaciones económicas del clúster y de la solución ofrecida son variadas. Se podría ofrecer un servicio de soporte de pago o alquilar el sistema completo o partes del clúster, dado que existen herramientas que permiten estas opciones.

Bibliografía

Absen. (2017). *Longgang New-type Smart City Operation Center*. https://www.absen.com/case/longgang-new-type-smart-city-operation-center/

Apache Airflow Documentation. (2024). Apache Software Foundation. https://airflow.apache.org/docs/

Apache Kafka Documentation. (2024). Apache Software Foundation. https://kafka.apache.org/documentation/

Apache NiFi Documentation. (2024). Apache Software Foundation. https://nifi.apache.org/docs.html

Apache Superset Documentation. (2024). Apache Software Foundation. https://superset.apache.org/docs/

Arrow Flight SQL Documentation. (2024). Apache Software Foundation. https://arrow.apache.org/docs/java/flight_sql.html

Ayuntamiento de París. (2024a). *Belib' - Points de recharge pour véhicules électriques - Données statiques* [Recuperado: Septiembre, 2024]. https://opendata.paris.fr/explore/dataset/belib-points-de-recharge-pour-vehicules-electriques-donnees-statiques/map/

Ayuntamiento de París. (2024b). *Qu'est ce que l'Open Data?* [Recuperado: Septiembre, 2024]. https://opendata.paris.fr/pages/faq/

Ayuntamiento de París. (2024c). *Vivre à Paris* [Recuperado: Septiembre, 2024]. https://dashboard.paris/pages/home/

Cottart, K. (2022). *Ingresses and Load Balancers in Kubernetes with MetalLB and nginx-ingress*. https://www.adaltas.com/en/2022/09/08/kubernetes-metallb-nginx/

DB-Engines Ranking [Accedido: Marzo, 2024]. (2024). https://db-engines.com/en/ranking

Distributed Systems and Internet Technologies LAB [DISIT LAB]. (2024). *Florence data overview* [Recuperado: Septiembre, 2024]. https://dashboard.km4city.org/dashboardSmartCity/view/index.php?iddasboard=Nzc=&nome_dashboard=Florence%20data%20overview

Documentación de InfluxDB [Accedido: Junio, 2024]. (2024). InfluxData. https://docs.influxdata.com/

EDAG. (2024). *Smart City Dashboard, THE DASHBOARD FRAMEWORK POWERED BY EDAG* [Recuperado: Septiembre, 2024]. https://smartcity.edag.com/en/referenzen/smart-city-dashboard/

Elastic. (2021). *Preguntas frecuentes sobre el cambio de licencia 2021*. https://www.elastic.co/es/pricing/faq/licensing

Films, S. (2016). *'Smarter Cities -Rio' — IBM* [Archivo de video]. https://www.youtube.com/watch?v=TwBbWVni5rA

Fitzsimmons, J. A., & Fitzsimmons, M. J. (2013). *Service Management: Operations, Strategy, Information Technology* (8th) [Explora los principios y prácticas de la gestión de servicios en diversos sectores.]. McGraw-Hill Education.

Fluent Bit Documentation. (2024). Fluent. https://docs.fluentbit.io/

Grafana Documentation. (2024). Grafana Labs. https://grafana.com/docs/

Huawei. (2024a). *Intelligent Operations Center: A Smart Brain for City Management* [Recuperado: Septiembre, 2024]. https://e.huawei.com/en/ict-insights/global/ict_insights/201908281022/focus/201911081641

Huawei. (2024b). *Smart City Intelligent Operations Center* [Recuperado: Septiembre, 2024]. https://e.huawei.com/es/videolist/video/40f4d3d702cf4cf4afa97fbf78b2fa3b

InfluxData. (2020). *Two Approaches to vSphere Monitoring Using InfluxDB and Telegraf*. https://get.influxdata.com/rs/972-GDU-533/images/Two_Approaches_to_vSphere_Monitoring_Using_InfluxDB_and_Telegraf.pdf

Institute, P. M. (2017). *A Guide to the Project Management Body of Knowledge (PMBOK Guide)* (6th) [Proporciona una visión general sobre la gestión de proyectos y las buenas prácticas asociadas.].

ITIL Foundation: ITIL 4 Edition [Accedido: Junio, 2024]. (2019). AXELOS. https://www.axelos.com/best-practice-solutions/itil

Khawaja Shams, T. V. (2024). RIP Redis: How Garantia Data Pulled Off the Biggest Heist in Open Source History. *Gomomento*. https://www.gomomento.com/blog/rip-redis-how-garantia-data-pulled-off-the-biggest-heist-in-open-source-history

Kubernetes Documentation. (2024). The Linux Foundation. https://kubernetes.io/docs/

Kubernetes Pod Documentation. (2024). The Linux Foundation. https://kubernetes.io/docs/concepts/workloads/pods/

Licencia de Elasticsearch [Accedido: Junio, 2024]. (2024). Elastic. https://www.elastic.co/es/licensing/elastic-license

Manifesto for Agile Software Development. (2001). Agile Alliance. https://agilemanifesto.org/

Nesi, P. (2023). *DISIT Lab, Snap4City, 5 Minute competences*. https://www.youtube.com/watch?v=GqwHUqK3As8

OpenSearch: A Distributed, RESTful Search and Analytics Suite [Accedido: Marzo, 2024]. (2024). Amazon. https://opensearch.org

PostgreSQL Documentation [Accedido: Marzo, 2024]. (s.f.). PostgreSQL. https://www.postgresql.org/docs/

Prometheus Documentation. (2024). Prometheus Authors. https://prometheus.io/docs/introduction/overview/

Proven, L. (2024). Redis changes license. *The Register*. https://www.theregister.com/2024/03/22/redis_changes_license/

Schwaber, K., & Sutherland, J. (2020). The Scrum Guide [El documento oficial que define el marco de trabajo Scrum, usado en la gestión ágil de proyectos.].

SQLAlchemy Documentation. (2024). SQLAlchemy. https://docs.sqlalchemy.org/

Telegraf Documentation. (2024). InfluxData. https://docs.influxdata.com/telegraf/

The Kubernetes Authors. (2024). *Kubernetes Documentation* [Recuperado: Marzo, 2024]. https://kubernetes.io/docs/concepts/architecture/

TimescaleDB Documentation. (2024). Timescale. https://docs.timescale.com/

TOPdesk. (2024). *What is Agile Service Management?* [Recuperado: Marzo, 2024]. https://www.topdesk.com/en/glossary/what-is-agile-service-management/

Urban and Regional Innovation Research [URENIO]. (2015). *Smart City Strategy: Rio de Janeiro, Brazil*. https://urenio.org/2015/03/23/smart-city-strategy-rio-de-janeiro-brazil/

vineetadurani. (2011). *IBM Helps Rio Become a Smarter City* [Archivo de video]. https://www.youtube.com/watch?v=vuBBGYFonXM

VMware. (2024). *Bitnami package for PostgreSQL HA* [Recuperado: Enero, 2024]. https://docs.vmware.com/en/VMware-Tanzu-Application-Catalog/services/apps/GUID-apps-postgresql-ha-index.html

vmware. (2024). *Bitnami package for Redis(R)* [Recuperado: Septiembre, 2024]. https://docs.vmware.com/en/VMware-Tanzu-Application-Catalog/services/apps/GUID-apps-redis-index.html