Máster en Big Data. Tecnología y Analítica Avanzada

Trabajo Fin de Master

Arquitectura Big Data y Procesamiento en Streaming para el envío de alertas frente a condiciones meteorológicas adversas

Autor
Borja Martín Baides

Supervisado por
Adrián Sanz Rodrigo

Madrid
July 2025

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

Arquitectura Big Data y Procesamiento en Streaming para el envío de alertas frente a condiciones meteorológicas adversas

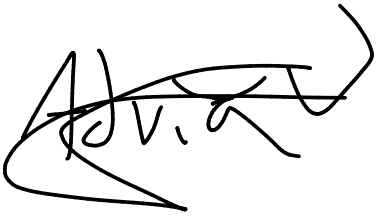
en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el curso académico 2024/25 es de mi autoría, original e inédito y no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada de otros documentos está debidamente referenciada.

Fdo.: Borja Martín Baides **Fecha: 01/07/2025**

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO



Fdo.: Adrián Sanz Rodrigo **Fecha: 01/07/2025**

Resumen

El presente Trabajo Fin de Máster (TFM) presenta el diseño e implementación de una arquitectura Big Data basada en microservicios para la monitorización en tiempo real de datos meteorológicos y la generación automática de alertas ante condiciones climáticas adversas. La motivación surge del creciente impacto de fenómenos meteorológicos extremos, tales como el acontecido con la DANA en Valencia este año pasado (2024), que ha evidenciado ciertas deficiencias en los sistemas actuales de alerta temprana.

El sistema propuesto utiliza tecnologías open source como Apache Kafka para el procesamiento en streaming, la API de OpenWeather para la recolección de datos meteorológicos, y Java con protocolo SMTP para la notificación de alertas personalizadas por correo electrónico. La arquitectura se compone de tres capas principales: ingesta, procesamiento y notificación.

El resultado es una herramienta modular, escalable, contenerizada y fácilmente desplegable, capaz de detectar en tiempo real situaciones de riesgo meteorológico según umbrales de precipitación definidos y de notificar automáticamente a los usuarios registrados. Además, se plantean mejoras futuras como una interfaz de usuario, integración de múltiples fuentes de datos, almacenamiento histórico y alertas multicanal. Este sistema contribuye a aumentar la resiliencia de la población y mejorar la capacidad de reacción ante eventos climáticos extremos.

Abstract

This Master's Thesis presents the design and development of a Big Data architecture based on microservices to monitor weather data in real time and automatically send alerts in case of severe weather conditions. The motivation comes from the growing impact of extreme weather events, such as the DANA storm in Valencia, which revealed serious issues in current early warning systems.

The proposed system uses open-source technologies like Apache Kafka for real-time data processing, the OpenWeather API to gather weather information, and Java with SMTP to send personalized email alerts. The architecture is organized in four main layers: data acquisition, ingestion, processing, and notification.

The result is a modular, scalable, and easy-to-deploy tool that can detect risky weather situations in real time based on configurable thresholds and alert registered users automatically. Future improvements include a user interface, more data sources, historical data storage, and multi-channel alerts. This system helps communities respond better and faster to extreme weather events.

Índice general

1. Introducción **1**

1.1. Descripción del Problema . . . 2
1.2. Motivación . . . 3
1.3. Objetivos . . . 4
1.4. Metodología . . . 5
1.5. Recursos . . . 6
1.6. Estructura del documento . . . 6

2. Estado del Arte **7**

2.1. Estado de la Cuestión . . . 8
2.2. Big Data . . . 9
2.3. Apache Kafka . . . 11
2.4. OpenWeather . . . 13
2.5. Protocolo SMTP . . . 13
2.6. Docker . . . 13
2.7. Github . . . 14
2.8. IntelliJ . . . 14
2.9. Apache Maven . . . 14

3. Desarrollo del Software **15**

3.1. Requisitos Funcionales . . . 16
3.2. Usuario Objetivo . . . 16
3.2.1. Cómo Registrarse . . . 16
3.3. Datos Disponibles . . . 17
3.4. Diseño y Desarrollo . . . 20
3.4.1. Data Source . . . 20
3.4.2. Data Ingestion Layer . . . 21
3.4.3. Data Processing Layer . . . 21
3.4.4. Data Serving Layer . . . 22
3.5. Pruebas de Validación . . . 22
3.5.1. weather-ingestor . . . 23
3.5.2. user-publisher . . . 24
3.5.3. alert-weather . . . 25

3.5.4. email-alert-service 26

4. Conclusiones y Mejoras **27**
4.1. Conclusiones 28
4.2. Mejoras 28

Referencias **29**

A. Modelo de datos **31**

B. Repositorio de código **32**

Índice de figuras

2.1. Cantidad de datos generados cada año. (*Fuente: explodingtopics*) 9
2.2. Arquitectura Kafka. (*Fuente: DZone Community Reseach Report*) 12

3.1. How to make an API Call. (*Fuente: OpenWeather*) 18
3.2. Diseño de Arquitectura. (*Fuente: Elaboración propia*) 20
3.3. Project Scaffold from IDE 22
3.4. Logs de ejecución *weather-ingestor* 23
3.5. kafka-console-consumer topics *raw-weather/enriched-weather* 23
3.6. Scaffold modulo *weather-ingestor* 24
3.7. Logs de ejecución *user-publisher* 24
3.8. kafka-console-consumer topic *users* 24
3.9. Scaffold modulo *user-publisher* 25
3.10. Logs de ejecución *alert-weather* 25
3.11. kafka-console-consumer topic *alerts* 25
3.12. Scaffold modulo *alert-weather* 25
3.13. Logs de ejecución *email-alert-service* 26
3.14. Email from *alertweathersystem* 26
3.15. Scaffold modulo *email-alert-service* 26

Índice de cuadros

2.1. Versiones Kafka compatibles con Zookeeper 11

Capítulo 1

Introducción

En este capítulo se presenta el Trabajo Fin de Máster (TFM), describiendo el problema que se pretende resolver, la motivación que impulsa el proyecto y los objetivos que se han planteado. Asimismo, se detalla la metodología seguida, los recursos utilizados y la estructura del documento.

1.1. Descripción del Problema

Los fenómenos meteorológicos adversos se han intensificado notablemente en los últimos años, debido principalmente al cambio climático. Episodios como lluvias torrenciales, granizadas o inundaciones son cada vez más frecuentes y provocan impactos significativos tanto en la sociedad como en el entorno natural y urbano.

Estos eventos no solo afectan infraestructuras y servicios básicos en ciudades y pueblos, sino que también suponen un grave riesgo para la vida humana y la economía local, haciendo necesaria una gestión más eficaz de los sistemas de emergencia.

Un ejemplo reciente es la DANA que afectó a Valencia, considerada una de las más graves en la historia reciente del país [1]. Este episodio provocó más de 200 fallecimientos y sacó a la luz importantes deficiencias en los sistemas de alerta temprana, así como una falta de coordinación entre organismos responsables. Este suceso junto a la ausencia de los avisos oportunos ha generado mucha desconfianza entre la población y ha puesto en evidencia la necesidad de soluciones más automatizadas y reactivas. Pues es probable que, de haberse emitido las alertas con antelación suficiente, las consecuencias podrían haber sido considerablemente menores.[2]

El presente proyecto tiene como finalidad el desarrollo de una herramienta informática basada en tecnologías Big Data que permita monitorizar, procesar y analizar en tiempo real datos meteorológicos procedentes de fuentes abiertas. Esta herramienta estará orientada a generar alertas automáticas ante condiciones climáticas adversas, permitiendo que los ciudadanos reciban avisos de forma inmediata. Con ello, se pretende mejorar la capacidad de reacción de la población, sin depender exclusivamente de organismos oficiales, contribuyendo a mitigar los efectos negativos de estos fenómenos.

[1] adaptecca.es
[2] elpais.com

1.2. Motivación

La motivación de este proyecto surge con la intención de mejorar los sistemas de prevención y alerta temprana ante eventos meteorológicos adversos, cuya frecuencia e intensidad han aumentado significativamente en los últimos años.

La intensa urbanización del terreno ha provocado la pérdida de las llanuras de inundación, las cuales actuaban como zonas naturales de expansión del agua durante episodios de lluvias extremas. Estas zonas han sido ocupadas por infraestructuras, viviendas y carreteras, lo que ha incrementado de forma significativa el riesgo para la población y los bienes materiales frente a estos sucesos.

Ante esta realidad, resulta evidente la necesidad de contar con todos los recursos disponibles y con herramientas capaces de emitir alertas tempranas, no solo para anticipar la llegada de fenómenos meteorológicos adversos como las DANAS, sino también para proporcionar el tiempo necesario que permita activar los protocolos de emergencia, proteger las zonas críticas y minimizar los daños.

Las consecuencias de una alerta tardía pueden ser catastróficas, como se ha comprobado recientemente en la Comunidad Valenciana, donde la falta de coordinación entre organismos y la emisión tardía de las alertas intensificaron los efectos de este fenómeno.[3]

Aportar por tanto una solución tecnológica basada en Big Data, que procese datos meteorol ógicos y automatice la detección y generación de alertas, puede contribuir significativamente a la reducción de daños materiales y humanos, así como a la optimización del uso de los recursos de emergencia.

[3] elpais.com

1.3. Objetivos

Los objetivos del proyecto se enumeran de la siguiente forma:

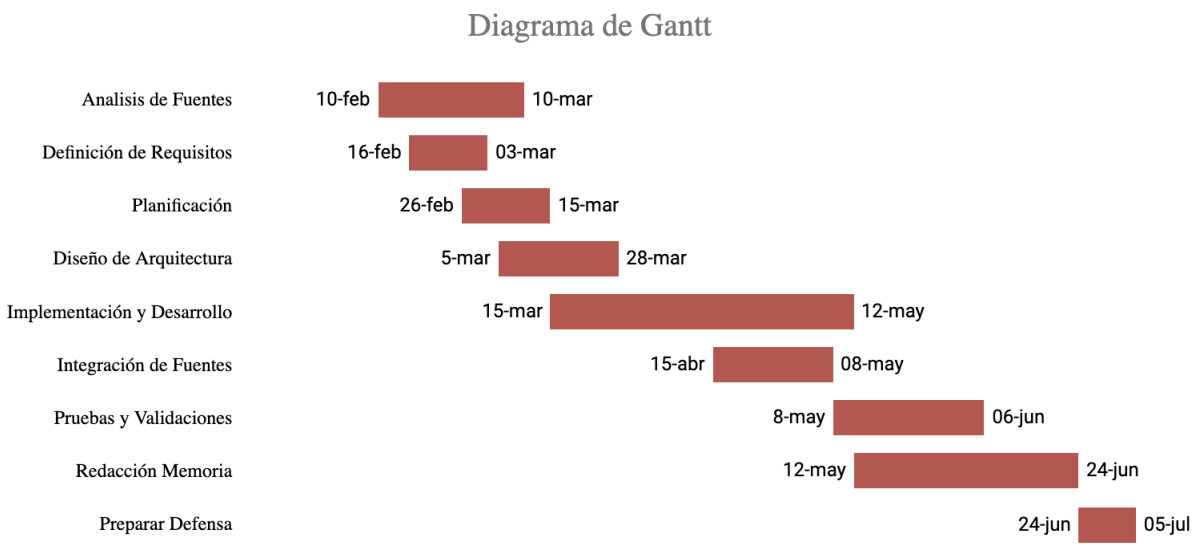
1. **Diseñar una arquitectura Big Data** para la ingesta, procesamiento y almacenamiento de datos meteorológicos en tiempo real. Que sea escalable y capaz de integrarse con distintos tipos de fuentes de datos para el envío de alertas.
2. **Desarrollar un sistema de alertas automatizado**, basado en la detección de condiciones meteorológicas adversas. Este sistema deberá utilizar reglas definidas y umbrales configurables, permitiendo la emisión de alertas personalizadas a través de correo electrónico.
3. **Integrar fuentes de datos abiertas y en tiempo real** de estaciones meteorológicas, asegurando la estandarización, limpieza y enriquecimiento de los mismos para su posterior tratamiento.
4. **Validar la utilidad y eficacia del sistema** mediante simulaciones y casos de estudio basados en eventos reales.

1.4. Metodología

Para la realización del trabajo se seguirá un enfoque incremental utilizando metodologías ágiles. Las tareas principales serán:

- Estudio y análisis de las fuentes de datos meteorológicos.
- Definición de requerimientos y planificación.
- Diseño de la arquitectura.
- Desarrollo e implementación del sistema.
- Integración de fuentes. Pruebas y validaciones.
- Redacción de memoria y preparación de la defensa.

En la siguiente figura se muestra el Diagrama de Gantt que se quiere seguir para la realización del proyecto:



1.5. Recursos

A continuación se exponen los recursos que se van a utilizar para la realización del proyecto:

1. **Apache Kafka**: Herramienta de ingesta y procesamiento de eventos en tiempo real.
2. **Protocolo SMTP**: Librería Java para el envío de alertas por correo electrónico.
3. **API meteorológica**: **OpenWeather** para la obtención de datos meteorológicos en tiempo real.
4. **Maven**: Como Build Tool para el empaquetado del proyecto.
5. Entorno de desarrollo: **Docker**, **Git**, **IntelliJ**.

Todos los recursos son de código abierto o accesibles mediante suscripciones gratuitas, por lo que son adecuados para su uso en el ámbito académico.

1.6. Estructura del documento

La estructura del documento se divide en los siguientes capítulos:

Capitulo 1. En el primer capitulo se describe la introducción al problema planteado, la motivación, los objetivos, la metodología y los recursos utilizados.

Capitulo 2. En este capitulo se analizan, describen y contextualizan las tecnologías y herramientas utilizadas.

Capitulo 3. En este capitulo se detallan las fases que se han seguido para el desarrollo, tales como la toma de requisitos funcionales, análisis de datos disponibles, diseño de la arquitectura, pruebas del flujo de procesamiento y validación de resultados.

Capitulo 4. Este capitulo recoge las conclusiones de los resultados obtenidos y plantea posibles mejoras.

Modelo de datos. En este apartado se añaden el modelo de datos que se ha aplicado en el desarrollo del proyecto.

Repositorio de código. Este anexo contiene el enlace al repositorio de código.

Capítulo 2

Estado del Arte

En este capitulo haremos una introducción a diversos aspectos relacionados con los estudios realizados para abordar el proyecto, así como las herramientas/tecnologías que se han empleado.

2.1. Estado de la Cuestión

Actualmente, existen diversas soluciones de monitorización meteorológica, tales como AEMET OpenData, plataformas privadas como Meteoclimatic y redes de sensores meteorológicos como AVAMET. Estas herramientas se enfocan principalmente en la recopilación y publicación de datos meteorológicos, generalmente en formatos estáticos o con cierta latencia en la actualización de la información. Aunque resultan útiles como referencia o consulta, su capacidad para anticipar eventos críticos o para reaccionar de forma automatizada es limitada.

En la mayoría de los casos, no cuentan con arquitecturas avanzadas basadas en Big Data que permitan un análisis predictivo dinámico ni el envío de alertas en tiempo real personalizadas para cada usuario o región. Esto deja un vacío importante en la detección proactiva de condiciones meteorológicas extremas que puedan representar un riesgo para la población.

En otros sectores, sin embargo, ya existen soluciones consolidadas que podrían ser aprovechadas en este contexto. Tecnologías como Apache Kafka, Apache NiFi y Apache Spark han sido utilizadas ampliamente en entornos industriales, financieros y científicos para el procesamiento de grandes volúmenes de datos en tiempo real, demostrando su eficiencia, escalabilidad y robustez. Estas herramientas permiten construir sistemas de monitorización reactiva con respuestas automatizadas, integrando diversas fuentes de datos, tanto estructuradas como no estructuradas, y ejecutar reglas complejas o modelos analíticos que podrían resultar decisivos en contextos de emergencia como los provocados por eventos meteorológicos extremos.

En cuanto a las herramientas y aplicaciones que permiten a los ciudadanos recibir alertas sobre fenómenos meteorológicos adversos, su efectividad depende en gran medida de la coordinación entre los organismos oficiales. Además, debe tenerse en cuenta que los sistemas operativos móviles como iOS y Android incluyen la capacidad de activar alertas de emergencia gubernamentales para avisos sobre condiciones meteorológicas extremas. Sin embargo, en algunos dispositivos estas alertas están desactivadas por defecto, lo que puede limitar su efectividad.

Como se ha experimentado durante la DANA en Valencia, a pesar de la existencia de herramientas de alerta, la dependencia exclusiva de los organismos oficiales para la emisión de avisos puede resultar insuficiente en situaciones críticas. Por ello, el desarrollo de una herramienta independiente y open source que permita a los ciudadanos recibir alertas tempranas en tiempo real a través de otros canales, sin depender exclusivamente de las decisiones de entidades oficiales, se presenta como una oportunidad para mejorar la capacidad de reacción ante fenómenos meteorológicos adversos.

2.2. Big Data

Hoy en día, la cantidad de datos que se genera mediante los dispositivos y sistemas que usamos cotidianamente ha crecido de forma exponencial:

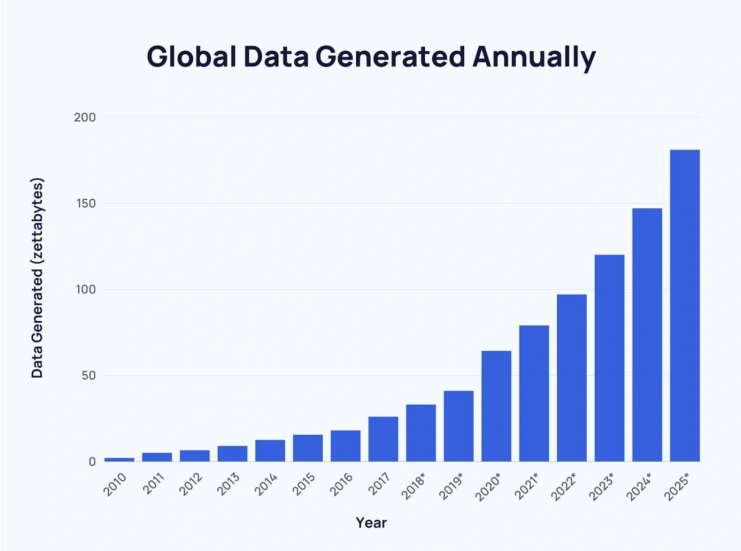


Figura 2.1: Cantidad de datos generados cada año. (*Fuente: explodingtopics*)

Esta situación hace que sea difícil analizar y tratar toda esta información de forma convencional. Hace unos años, manejar tal volumen de datos era algo complejo y muy costoso, pero gracias al avance de la tecnología en procesamiento y almacenamiento —además de la reducción de costes— ahora resulta viable y rentable trabajar con grandes cantidades de datos.

El Big Data se define como el conjunto de tecnologías y métodos que permiten procesar y analizar grandes cantidades de datos que, por su complejidad o cantidad, no pueden ser tratados de manera eficiente con herramientas tradicionales. El objetivo principal es extraer valor de esa información para apoyar en la toma de decisiones y descubrir patrones o tendencias útiles.

Es importante resaltar que los datos, por sí solos, no aportan valor directamente a la toma de decisiones. Sin el análisis adecuado no explican el porqué de los hechos, es a través de la interpretación y el análisis contextual que los datos se transforman en información, es decir, en datos procesados que tienen significado y utilidad para la toma de decisiones.

Por tanto, conviene distinguir entre datos, información y conocimiento.

- **Dato**: Unidad mínima que representa un hecho sin interpretación.
- **Información**: Conjunto de datos procesados que tienen un significado para quien los recibe.
- **Conocimiento**: Resultado de combinar información con la experiencia, el contexto y el análisis. A partir de la información se genera conocimiento.

Poder extraer información útil de los datos permite mejorar muchos aspectos del día a día: por ejemplo, optimizar rutas de transporte o detectar zonas con alta contaminación. En este sentido, los datos se han convertido en un recurso muy valioso que impulsa soluciones prácticas e innovadoras en múltiples ámbitos.

2.3. Apache Kafka

Dentro del ecosistema Big Data existen diversas herramientas de procesamiento en tiempo real, la más destacada es **Apache Kafka**.

Kafka es una plataforma de mensajería distribuida diseñada para trabajar con grandes volúmenes de datos en tiempo real. Fue creada originalmente por LinkedIn y posteriormente donada a la fundación **Apache**. A raíz de su éxito, se fundó la empresa **Confluent**, centrada en el desarrollo de soluciones basadas en Kafka.

Estas soluciones destacan por su alto rendimiento, baja latencia y escalabilidad. Dichas características lo convierten en una herramienta fundamental en arquitecturas orientadas a eventos. Es tolerante a fallos, soporta la replicación de datos y permite su despliegue en clústeres de múltiples nodos. Para el procesamiento de datos en tiempo real, se utiliza Kafka Streams, una librería que permite construir aplicaciones que transforman, filtran y combinan flujos de datos dentro del propio ecosistema de Kafka. Es especialmente útil para casos donde se requiere actuar sobre la información conforme llega, como en la generación de alertas meteorológicas.

En versiones anteriores, Kafka requería de un componente adicional llamado **Zookeeper**, encargado de coordinar el clúster, gestionar los brokers y mantener la integridad del sistema distribuido. Sin embargo, desde la versión 2.8.0, Kafka ha comenzado a integrar de forma nativa estas funciones mediante un nuevo modo llamado KRaft (Kafka Raft Metadata mode), eliminando de este modo la necesidad de Zookeeper. Actualmente, la versión más reciente es la 4.0.0 que ya elimina por defecto el uso de Zookeeper. En la siguiente tabla se muestra la evolución de las versiones de manera resumida.

Versión Kafka	**Zookeeper**	**Soporta KRaft**
2.8.0	Sí (por defecto)	Experimental (No por defecto)
3.0 – 3.3	Sí (por defecto)	Parcial (No por defecto)
3.4 – 3.9	Sí (por defecto)	Estable (No por defecto)
4.0.0	No (eliminado)	Total (por defecto)

Cuadro 2.1: Versiones Kafka compatibles con Zookeeper

Aunque el enfoque KRaft simplifica la arquitectura y reduce la complejidad, para este proyecto se ha utilizado una versión de Kafka compatible con Zookeeper, concretamente la 3.5.1 que viene integrada en la versión 7.5.0 gratuita del ***Confluence Community Software***.

Kafka permite transmitir datos de forma rápida y fiable entre diferentes componentes de un sistema. Utiliza una arquitectura basada en el modelo "publish-subscribe", en la que los productores (producers) envían mensajes a canales llamados topics, y los consumidores (consumers) se suscriben a esos topics para recibir la información.

En la siguiente figura se muestra un esquema de sus arquitectura.

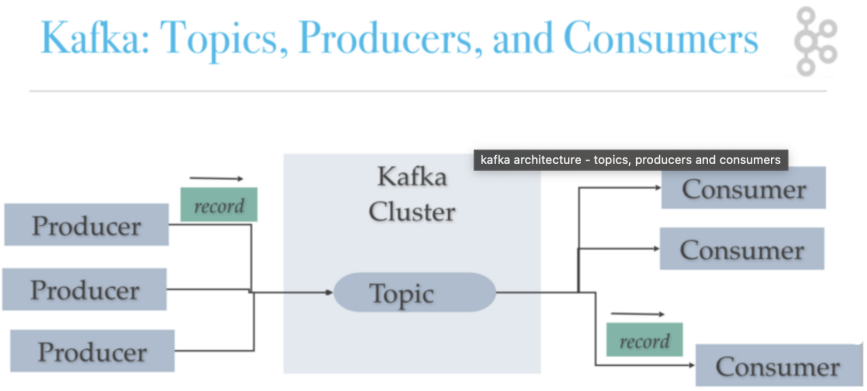


Figura 2.2: Arquitectura Kafka. (*Fuente: DZone Community Reseach Report*)

Una de sus claves más destacadas es que actúa como un çommit-log"distribuido, es decir, un registro inmutable en el que los datos se van añadiendo secuencialmente. Esto garantiza la consistencia y permite que múltiples consumidores accedan a la misma información en momentos distintos.

Características de Kafka:

- **Alto rendimiento**: Trabaja con baja latencia gracias a su arquitectura basada en escritura secuencial.
- **Distribuido**: Opera en clústeres de múltiples nodos.
- **Escalable**: Permite añadir más nodos fácilmente.
- **Tolerancia a fallos**: Soporta replicación y recuperación automática.
- **Multilenguaje**: Compatible con varios lenguajes de programación.
- **Integración sencilla**: Posee un ecosistema maduro y herramientas como Kafka Connect y REST Proxy.
- **Retrocompatibilidad**: Garantiza compatibilidad entre versiones anteriores.

En este proyecto, Kafka actúa como el canal principal de comunicación, encargándose de la etapa de ingesta, procesamiento y envío de alertas a los usuarios finales. Usando un total de 4 topics (*raw-weather*, *enriched-weather*, *users*, *alerts*).

2.4. OpenWeather

Para obtener los datos meteorológicos en tiempo real, se ha recurrido a la API proporcionada por **OpenWeather**, una de las plataformas más utilizadas a nivel mundial para acceder a información climática.

La API OpenWeather permite consultar datos actualizados sobre temperatura, humedad, presión atmosférica, velocidad y dirección del viento, y condiciones generales como lluvia o nubosidad. A través de solicitudes HTTP tipo REST, el sistema puede consultar información en formato JSON para una o varias ciudades, utilizando parámetros como el nombre, el ID o las coordenadas geográficas de cada localidad.

En este proyecto, la API se utiliza para recopilar datos actuales de múltiples ciudades, los cuales son procesados por el módulo *weather-ingestor*. Esta información se recoge en el topic *raw-weather* para su tratamiento y enriquecimiento.

El uso de OpenWeather ha sido fundamental por su fiabilidad, simplicidad de integración y disponibilidad de un plan gratuito con acceso suficiente para cubrir las necesidades académicas del proyecto.

2.5. Protocolo SMTP

Para la notificación de alertas meteorológicas por correo electrónico, se ha implementado un sistema de envío de emails utilizando el protocolo SMTP (Simple Mail Transfer Protocol) de java. Se ha decidido utilizar este protocolo al ser compatible con la versión **java 8** que es la que hemos utilizado para el desarrollo del sistema.

Se ha configurado una clase de servicio encargada de construir y enviar los correos a través de una cuenta configurada en un servidor SMTP externo de Gmail. Esta implementación se apoya en las librerías estándar de JavaMail [1], integradas en el microservicio *email-alert-service*. El sistema permite parametrizar datos como el remitente, las credenciales, el servidor SMTP y el contenido del mensaje, que puede incluir texto personalizado en función de la alerta generada (localización, intensidad de lluvia, umbral superado, etc.).

Esta solución ha demostrado ser sencilla, fiable y efectiva para el propósito del proyecto, y permite que cada usuario reciba notificaciones automáticas y personalizadas ante condiciones meteorológicas críticas en su zona de interés.

2.6. Docker

Docker es una tecnología basada en contenedores que permite empaquetar aplicaciones y sus dependencias en entornos aislados y portables. Su uso en el proyecto facilita

[1] javamail-que-es

la orquestación de los distintos microservicios desarrollados, asegurando coherencia en los entornos de desarrollo, prueba y producción. Esto permite que las aplicaciones se comporten de la misma forma sin importar el sistema operativo o el entorno en el que se ejecuten.

2.7. Github

Git es el sistema de control de versiones más extendido actualmente. Se trata de una herramienta fundamental en el desarrollo de software moderno, ya que permite llevar un registro detallado de los cambios realizados en el código fuente.

Es imprescindible para trabajar de forma colaborativa sin conflictos, revertir cambios cuando sea necesario, y crear ramas para nuevas funcionalidades sin afectar el código principal. Además, facilita la integración con plataformas como GitHub o GitLab, que permiten gestionar repositorios remotos, realizar revisiones de código mediante pull requests y automatizar procesos de despliegue.

En este proyecto, se ha utilizado **GitHub** para organizar las versiones del sistema, documentar el avance del desarrollo de forma trazable y ordenada, y asegurar la durabilidad del trabajo realizado.

2.8. IntelliJ

IntelliJ IDEA es el entorno de desarrollo integrado (IDE) que se ha utilizado para programar los servicios en Java. Es uno de los IDEs más utilizados en el desarrollo del Software, su compatibilidad con Maven, Git y Docker lo convierte en una herramienta ágil para el desarrollo de aplicaciones complejas.

2.9. Apache Maven

Maven es la build tool que se ha utilizado para la construcción y gestión del proyecto multimodulo, la cual ha facilitado la compilación, empaquetado, fase de testeo y gestión de dependencias. Gracias a su estructura basada en el archivo pom.xml, es posible definir de forma sencilla todos los paquetes externas necesarios, así como plugins de compilación, pruebas y despliegue.

Maven permite estructurar de forma clara y sencilla los distintos módulos del sistema, facilitando su mantenimiento y construcción de forma independiente o conjunta. Además, su integración con el IDE IntelliJ y simplifica enormemente las tareas de desarrollo y pruebas.

Capítulo 3

Desarrollo del Software

Este capítulo recoge el desarrollo, diseño y construcción del sistema de alertas meteorológicas. Se detallan los requisitos funcionales, el perfil de los usuarios a los que va dirigido, las fuentes de datos que se han empleado y cada una de las capas del proceso.

3.1. Requisitos Funcionales

El objetivo principal del proyecto es desarrollar una herramienta que permita detectar condiciones meteorológicas adversas —como lluvias intensas— en tiempo real, y generar alertas automatizadas que se notifiquen a los usuarios por correo electrónico.

Esta herramienta se plantea como un complementaria a la emisión de alertas de los organismos oficiales. Debe ser modular, escalable, de fácil despliegue y capaz de integrarse con distintas fuentes de datos meteorológicos abiertas.

Entre los requisitos funcionales destacan:

1. Ingestar datos meteorológicos en tiempo real.
2. Enriquecer los datos con información contextual (zonas, usuarios, umbrales).
3. Generar alertas personalizadas si se superan determinados umbrales.
4. Notificar automáticamente de las alertas vía email a los usuarios registrados.

3.2. Usuario Objetivo

La herramienta está dirigida a usuarios que viven en zonas susceptibles de verse afectadas por fenómenos meteorológicos adversos y desean recibir alertas de manera automatizada y personalizada.

También puede ser útil para ayuntamientos, organismos de protección civil o entidades interesadas en mejorar su capacidad de respuesta frente a eventos climáticos extremos.

El perfil del usuario final no requiere conocimientos técnicos: el sistema automatiza la recolección de datos y el envío de alertas, haciendo que la experiencia sea completamente pasiva y transparente desde el primer momento.

3.2.1. Cómo Registrarse

Para darse de alta en el sistema de alertas, el usuario deberá enviar un correo electrónico a la dirección alertweathersystem@gmail.com con **ASUNTO: ALTA USUARIO** y los siguientes datos:

- **Nombre**
- **Apellidos**
- **DNI/NIF**
- **Correo electrónico**
- **Localización donde vive o desea monitorizar**

Una vez procesada la solicitud, el usuario quedará registrado en el sistema y comenzará a recibir alertas en función de su ubicación y los umbrales definidos según su zona.

Del mismo modo para darse de baja, el usuario deberá enviar un correo con su nombre y apellidos, dni y correo electrónico a la misma dirección con ASUNTO: BAJA USUARIO.

3.3. Datos Disponibles

Durante la fase de análisis se han evaluado diversas fuentes de datos meteorológicos abiertos, tales como:

- **AEMET**: Agencia Estatal de Meteorología en España.
- **AVAMET**: Associación Valenciana de Meteorología.
- **OpenWeather**: Plataforma internacional que proporciona datos meteorológicos actualizados mediante una API REST de fácil integración.

La fuente elegida para el proyecto ha sido **OpenWeather** debido a su facilidad de integración mediante peticiones API REST. Esta plataforma cuenta con una documentación clara, ejemplos prácticos y un plan gratuito que ofrece un número suficiente de solicitudes para el contexto académico. Además, al tratarse de un servicio con cobertura internacional, permite monitorizar ubicaciones fuera del territorio nacional, lo cual añade flexibilidad y escalabilidad a la solución.

Los datos obtenidos incluyen precipitaciones, temperatura, presión, viento y humedad, entre otros. A continuación se muestra una imagen de su portal de desarrollo con las instrucciones para llamar a la API y un ejemplo del contenido de la respuesta en formato json.

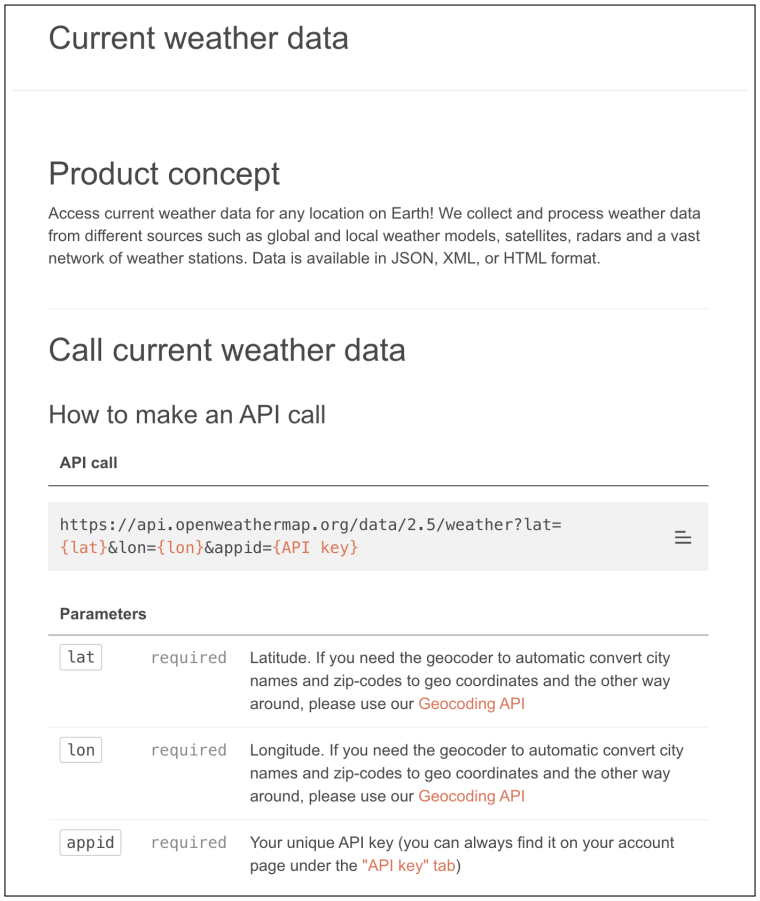


Figura 3.1: How to make an API Call. (*Fuente: OpenWeather*)

Listing 3.1: JSON format API response example


```
{
   "coord": {
      "lon": 7.367,
      "lat": 45.133
   },
   "weather": [
      {
         "id": 501,
         "main": "Rain",
         "description": "moderate␣rain",
         "icon": "10d"
      }
   ],
   "base": "stations",
   "main": {
      "temp": 284.2,
      "feels_like": 282.93,
      "temp_min": 283.06,
      "temp_max": 286.82,
      "pressure": 1021,
      "humidity": 60,
      "sea_level": 1021,
      "grnd_level": 910
   },
   "visibility": 10000,
   "wind": {
      "speed": 4.09,
      "deg": 121,
      "gust": 3.47
   },
   "rain": {
      "1h": 2.73
   },
   "clouds": {
      "all": 83
   },
   "dt": 1726660758,
   "sys": {
      "type": 1,
      "id": 6736,
      "country": "IT",
      "sunrise": 1726636384,
      "sunset": 1726680975
   },
   "timezone": 7200,
   "id": 3165523,
   "name": "Province␣of␣Turin",
   "cod": 200
}
```

3.4. Diseño y Desarrollo

Antes de abordar la fase de desarrollo, es importante previsualizar el diseño de la arquitectura definido.

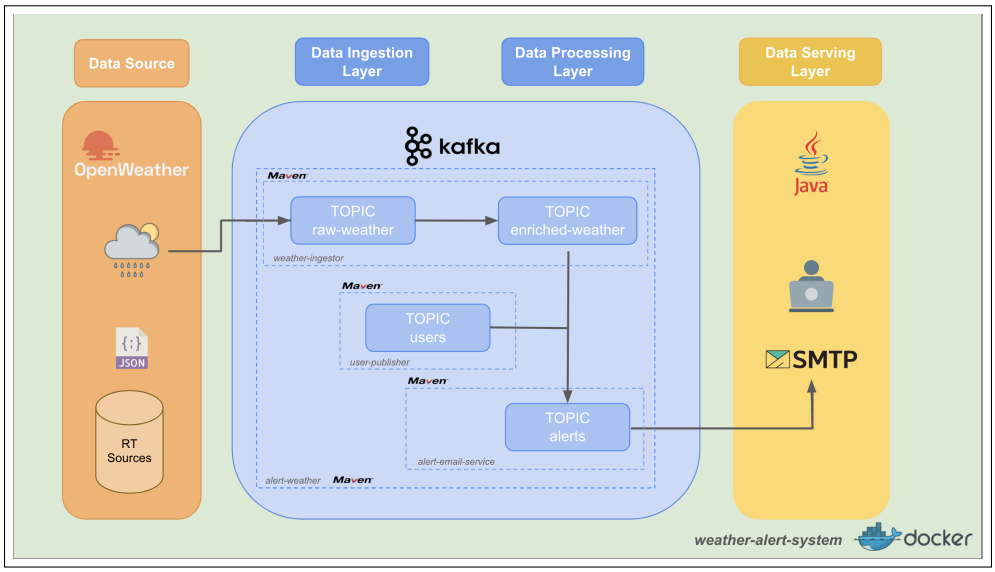


Figura 3.2: Diseño de Arquitectura. (*Fuente: Elaboración propia*)

Como se aprecia en la figura se distinguen 4 fases o capas dentro del flujo de procesamiento. Las cuales son:

- **Data Source**: Capa de la fuente de datos
- **Data Ingestion Layer**: Capa de ingesta de datos
- **Data Processing Layer**: Capa de procesamiento de datos
- **Data Sercing Layer**: Capa de servicio de datos

El desarrollo se ha estructurado siguiendo una arquitectura por módulos basada en microservicios. Cada modulo o servicio representa una parte de este flujo de procesamiento, desde su origen hasta la notificación de alertas.

A continuación, se explica en detalle cada una de estas capas y los módulos/servicios que las engloba.

3.4.1. Data Source

Esta capa habilita los datos meteorológicos desde la **API** para un conjunto de ciudades definidas. Se trata de un flujo continuo de datos en Real Time que se refresca minuto a minuto.

3.4.2. Data Ingestion Layer

Esta capa procesa los datos mediante el servicio **weather-ingestor**, que publica los eventos en el topic ***raw-weather***. Las consultas se realizan mediante un cliente Java que lanza consultas periódicamente a la API y obtiene los datos en formato JSON.

Las ciudades que se procesan están recogidas en el fichero *cities.json* con información de su latitud y longitud para construit la petición a la API.

Después de recibir los eventos en raw-weather se hace un tratamiento de los datos para obtener un subset con la información necesaria para el alertado de precipitación y se publica en el topic ***enriched-weather***.

Desde el servicio **user-publisher** y por medio del fichero *users.json* se publican los usuarios en el topic ***users***. Los usuarios recogidos en este fichero serán aquellos que hayan solicitado el alta.

A continuación se muestra un ejemplo de ambos ficheros.

Listing 3.2: Ejemplo users.json

```
[
  { "name": "Borja", "id": 1, "mail": "borja@gmail.com", "
     zone": "Madrid", "threshold": 0 },
  { "name": "Adrian", "id": 2, "mail": "adrian@gmail.com",
     "zone": "Madrid", "threshold": 3}
]
```

Listing 3.3: Ejemplo cities.json

```
[
  { "name": "Madrid", "lat": 40.4168, "lon": -3.7038 },
  { "name": "Toledo", "lat": 39.47391, "lon": -4.02263 },
  { "name": "Valencia", "lat": 39.47391, "lon": -0.37966 }
]
```

3.4.3. Data Processing Layer

En esta capa actúa el servicio **alert-weather**, que mediante Kafka Streams procesa los datos meteorológicos y cruza los datos con los usuarios registrados en el topic ***users***. Si el valor de precipitación supera el umbral configurado, se genera un evento de tipo *alert* y se publica en el topic ***alerts***.

3.4.4. Data Serving Layer

Por último, el servicio **email-alert-service** escucha el topic ***alerts*** y se encarga de componer y enviar los correos electrónicos de alerta a los usuarios afectados, utilizando el paquete SMTP de Java.

En la siguiente figura se muestra el scaffolder del proyecto con todos los módulos comentados. El modulo adicional **common** es un modulo común para todos que contiene utilidades y modelos de datos compartidos entre los distintos módulos.

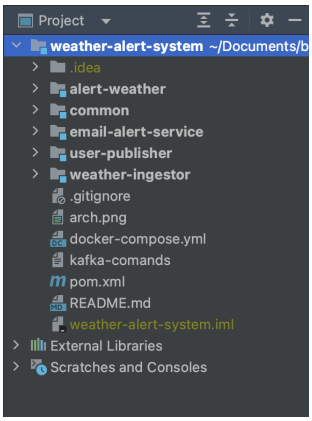


Figura 3.3: Project Scaffold from IDE

3.5. Pruebas de Validación

Se han realizado pruebas unitarias y de integración para verificar que cada componente funciona correctamente de forma individual y en conjunto. Entre las pruebas destacadas se incluyen:

- Verificación de conexión y formato de datos de la API.
- Validación de publicación y consumo de eventos en Kafka.
- Generación de alertas a partir de datos consumidos.
- Envío de correos con contenido dinámico en función del evento.

A continuación se muestran las evidencias de estás validaciones así como la ejecución de cada uno de los módulos en el proceso.

3.5.1. weather-ingestor

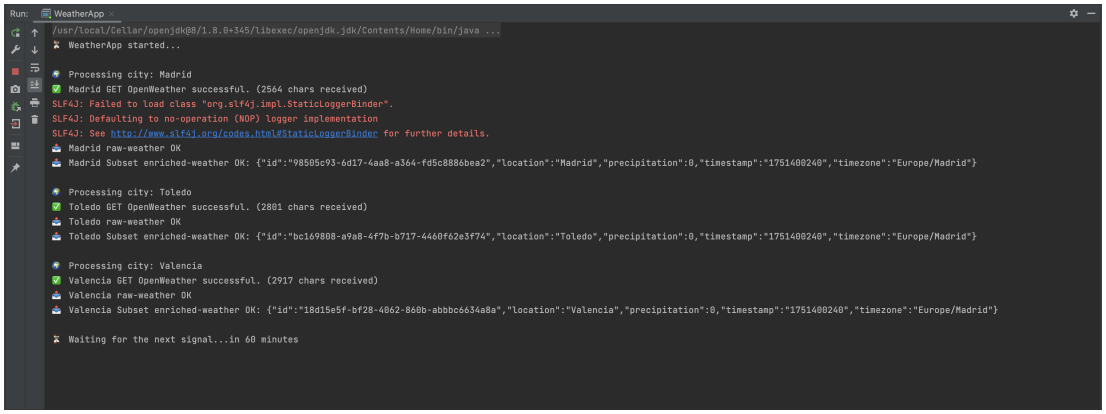


Figura 3.4: Logs de ejecución *weather-ingestor*

Figura 3.5: kafka-console-consumer topics *raw-weather* / *enriched-weather*

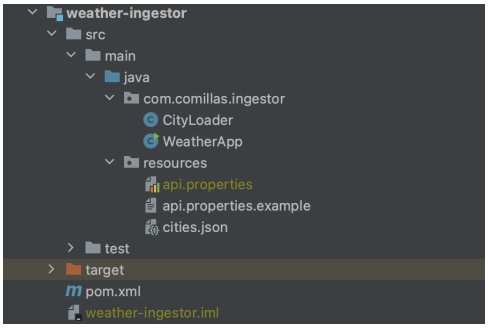


Figura 3.6: Scaffold modulo *weather-ingestor*

3.5.2. user-publisher

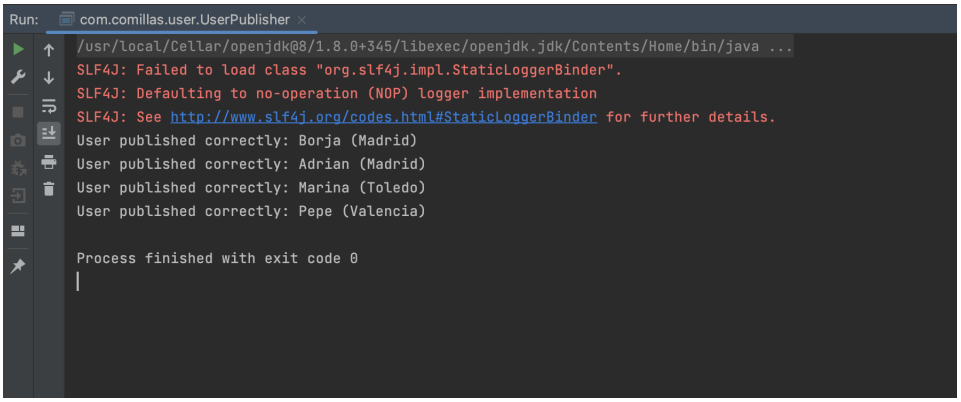


Figura 3.7: Logs de ejecución *user-publisher*

```
[appuser@258e7432e936 ~]$ kafka-topics --list --bootstrap-server localhost:9092
__consumer_offsets
enriched-weather
raw-weather
users
[appuser@258e7432e936 ~]$ kafka-console-consumer --topic users --from-beginning --bootstrap-server localhost:9092
{"name":"Borja","id":1,"mail":"borjamartinbaides@gmail.com","zone":"Madrid","threshold":0}
{"name":"Adrian","id":2,"mail":"adriansanzoficial@gmail.com","zone":"Madrid","threshold":3}
{"name":"Marina","id":3,"mail":"garciamolineromarina@gmail.com","zone":"Toledo","threshold":0}
{"name":"Pepe","id":4,"mail":"borjamartinbaides@gmail.com","zone":"Valencia","threshold":0}
```

Figura 3.8: kafka-console-consumer topic *users*

Figura 3.9: Scaffold modulo *user-publisher*

3.5.3. alert-weather

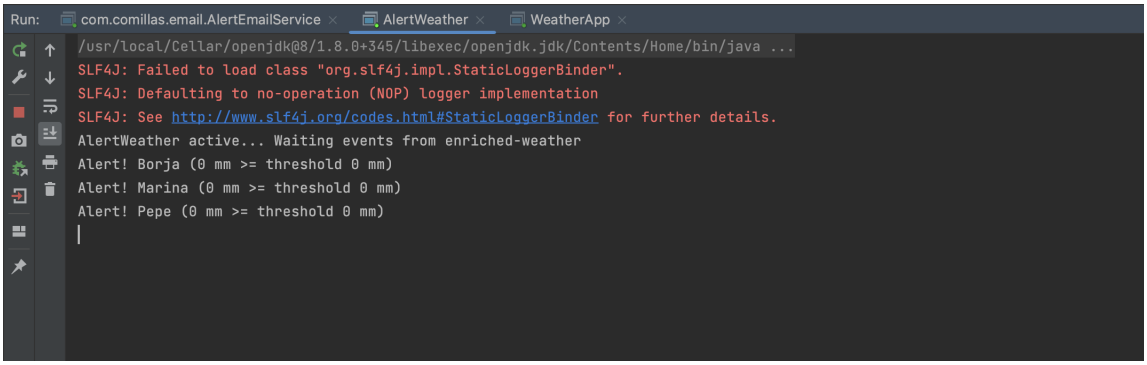


Figura 3.10: Logs de ejecución *alert-weather*

```
[appuser@258e7432e936 ~]$ kafka-topics --list --bootstrap-server localhost:9092
__consumer_offsets
alerts
enriched-weather
raw-weather
users
[appuser@258e7432e936 ~]$ kafka-console-consumer --topic alerts --from-beginning --bootstrap-server localhost:9092
{"name":"Borja","mail":"borjamartinbaides@gmail.com","zone":"Madrid","precipitation":0,"threshold":0,"timestamp":"1751401140"}
{"name":"Marina","mail":"garciamolineromarina@gmail.com","zone":"Toledo","precipitation":0,"threshold":0,"timestamp":"1751401140"}
{"name":"Pepe","mail":"borjamartinbaides@gmail.com","zone":"Valencia","precipitation":0,"threshold":0,"timestamp":"1751401140"}
```

Figura 3.11: kafka-console-consumer topic *alerts*

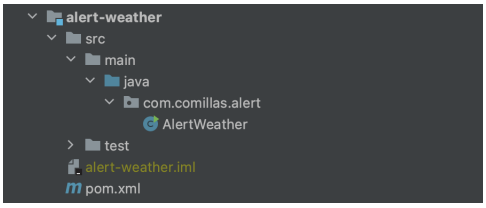


Figura 3.12: Scaffold modulo *alert-weather*

3.5.4. email-alert-service

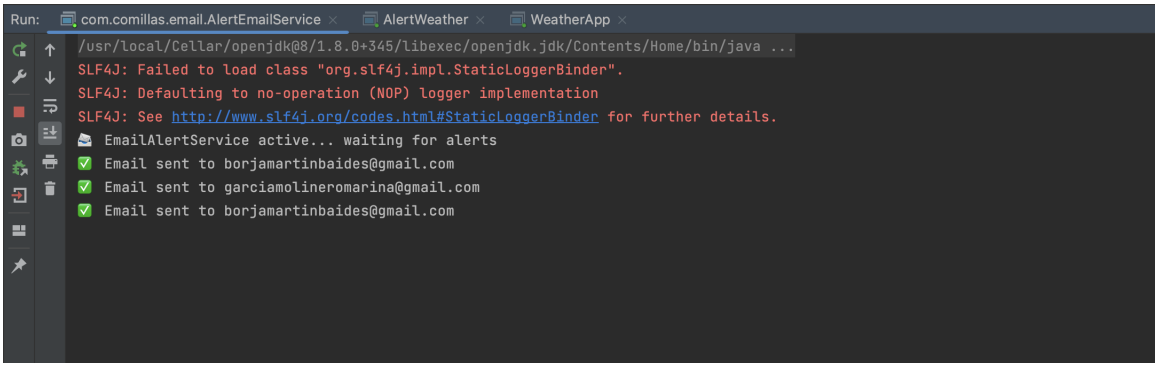


Figura 3.13: Logs de ejecución *email-alert-service*

Figura 3.14: Email from *alertweathersystem*

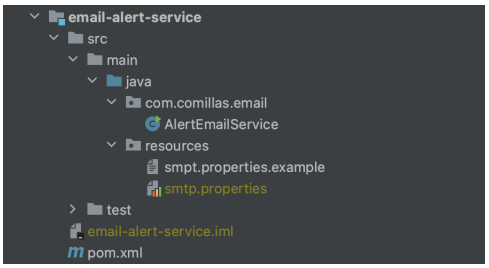


Figura 3.15: Scaffold modulo *email-alert-service*

Capítulo 4

Conclusiones y Mejoras

Este capítulo presenta una reflexión final sobre los resultados obtenidos con el desarrollo del sistema, así como una serie de propuestas de mejora que podrían aplicarse en futuras iteraciones del proyecto.

4.1. Conclusiones

En este TFM se ha conseguido diseñar e implementar un sistema para la recolección, procesamiento y notificación de alertas meteorológicas en tiempo real. Se ha seguido una arquitectura basada en microservicios, y utilizado tecnologías open source como Kafka y Docker. Además, el sistema se ha integrado con datos de la API OpenWeather lo que ha permitido construir una solución escalable, modular y fácilmente desplegable.

En resumen, el sistema es capaz de:

- Consultar datos meteorológicos actualizados para múltiples ciudades por medio de peticiones API REST.
- Enviar datos en tiempo real a topics de Kafka.
- Detectar automáticamente situaciones de riesgo según los umbrales definidos por cada zona/ciudad.
- Enviar alertas personalizadas por correo electrónico a los usuarios afectados.

Por tanto, podemos concluir que se han alcanzado los objetivos propuestos y que puede resultar de utilidad para prevenir a los ciudadanos de eventos climatológicos con precipitación elevada.

4.2. Mejoras

A continuación se plantean posibles mejoras de cara a realizar la herramienta más útil y atractiva:

- **Panel de administración para usuarios**: Crear una interfaz web/App que permita a los usuarios registrarse, modificar sus umbrales o zonas de interés, y consultar su historial de alertas.
- **Soporte para más fuentes de datos**: Incorporar más fuentes de datos como AEMET, AVAMET que permitan mejorar la precisión.
- **Base de datos histórica**: Almacenar los eventos meteorológicos y las alertas generadas para análisis posterior y visualización.
- **Alertas multicanal**: Añadir la posibilidad de envíar notificaciones no solo por correo, sino también por SMS, Whatshapp, Telegram u otras plataformas.
- **Sistema de predicción**: Integrar modelos de predicción meteorológica o aprendizaje automático para anticiparse a eventos antes de que ocurran.

Estas mejoras permitirían ampliar el alcance del sistema y su utilidad práctica, haciendo que sea más adaptable a distintos contextos y usuarios finales.

Referencias

[1] AEMET OpenData. https://opendata.aemet.es/centrodedescargas/inicio

[2] AVAMET OpenData. https://www.avamet.org/?utm_source=chatgpt.com

[3] Meteoclimatic. https://www.meteoclimatic.net/?screen_width=1680

[4] OpenWeather. https://openweathermap.org/

[5] adapteca. *adapteca noticias "la dana de valencia entre los 10 desastres más costosos del 2024"*. https://adaptecca.es/recursos/noticias/la-dana-de-valencia-entre-los-10-desastres-naturales-mas-costosos-del-2024

[6] elpais. *elpais noticias .ºcho horas que pudieron cambiarlo todo"* https://elpais.com/expres/2025-04-15/ocho-horas-que-pudieron-cambiarlo-todo-claves-de-la-declaracion-de-la-delegada-del-g
html?utm_source=chatgpt.com.

[7] elpais. *elpais noticias ün experto confirma que la alerta de la generalitat fue tardia y confusa"*. https://elpais.com/espana/comunidad-valenciana/2025-04-01/un-experto-confirma-a-la-jueza-de-la-dana-que-la-alerta-de-la-generalitat-fue-tardi
html?event=regonetap&event_log=regonetap&prod=REGONETAP&o=regonetap

[8] explodingtopics. *explodingtopics - Data Generated Per Day.* https://explodingtopics.com/blog/data-generated-per-day?utm_source=chatgpt.com

[9] Apache Kafka. https://kafka.apache.org/

[10] dzone. *DZone Community Reseach Report - kafka architecture* https://dzone.com/articles/kafka-architecture.

[11] Fundación Apache. https://apache.org/

[12] Confluent. https://www.confluent.io/es-es/

[13] Platform Confluent. *Platform Confluent Versions* https://docs.confluent.io/platform/current/installation/versions-interoperability.html

[14] Docker. https://www.docker.com/

[15] Git. *GitHub* - *GitLab*. https://git-scm.com/

[16] IntelliJ IDEA. https://www.jetbrains.com/es-es/idea/

[17] Apache Maven. https://maven.apache.org/

Anexo A

Modelo de datos

A continuación se muestra el modelo de datos dentro del proyecto:

`User`: name, id, mail, zone, threshold
`City`: name, lat, lon
`WeatherRaw`: datos completos de la API OpenWeather
`WeatherEnriched`: id, location, precipitation, timestamp, timezone
`Alert`: name, mail, zone, precipitation, threshold, timestamp

Anexo B

Repositorio de código

El código fuente completo del sistema desarrollado está disponible en el siguiente repositorio de GitHub:

```
https://github.com/borjabaides/weather-alert-system
```

Este repositorio contiene todos los módulos del proyecto, la configuración Docker, archivos de ejemplo y documentación adicional para su instalación y testeo.