JUSTIFICACIÓN DE LAS DECISIONES TECNOLÓGICAS DEL PROYECTO.

La arquitectura del pipeline diseñado para el proyecto de calidad del aire responde a la necesidad de construir un sistema **automático, escalable y fácilmente ampliable**, que permita integrar datos procedentes de distintas fuentes, procesarlos en tiempo casi real y presentarlos de forma clara para su análisis. Cada componente ha sido seleccionado para cubrir una función específica dentro del flujo de ingesta, transformación y visualización, siguiendo un enfoque modular y basado en buenas prácticas actuales de ingeniería de datos.

1. Ingesta de datos mediante Python y APIs

El proceso comienza con la consulta periódica de dos fuentes de datos públicas:

- **API de contaminación**
- **API de meteorología**

La elección de **Python** para esta fase se debe a:

- Su **amplio ecosistema de librerías** que facilitan la llamada a APIs, el manejo de JSON y el formateo de datos.
- Su **sencillez y rapidez** para desarrollar scripts que automatizan tareas recurrentes.
- Su **compatibilidad directa con Kafka y PostgreSQL**, permitiendo mantener una arquitectura homogénea.

Además, se ha configurado una **frecuencia de actualización de 10 minutos**, adecuada para el tipo de dato (ambiental) y lo suficientemente frecuente para detectar variaciones relevantes sin sobrecargar el sistema.

Todo ello permite contar con una ingesta **estable, automatizada y controlada**, base imprescindible para el resto del pipeline.

2. Uso de Kafka como sistema de mensajería

La incorporación de **Apache Kafka** entre los scripts de ingesta y el procesamiento posterior responde a tres motivos principales:

1. **Desacoplamiento entre productores y consumidores:** Esto nos permite hacer evolucionar cada parte del sistema por separado sin generar dependencias rígidas.
2. **Capacidad para gestionar flujos continuos:** Dado que recibimos datos cada 10 minutos y potencialmente podríamos aumentar la frecuencia o añadir nuevas

fuentes, Kafka actúa como un buffer robusto que garantiza que no haya pérdida de datos.
3. **Escalabilidad a futuro:** Si el proyecto creciera (más APIs, más estaciones, datos en tiempo real…), Kafka absorbería el aumento sin necesidad de rediseñar el pipeline.

Por tanto, Kafka aporta no solo robustez, sino también **flexibilidad arquitectónica**.

3. PostgreSQL como almacenamiento central

Una vez consumidos los mensajes de Kafka, los datos se almacenan en **PostgreSQL**. El proyecto utiliza dos capas diferenciadas:

- **raw_pollution** (datos brutos descargados directamente de Kafka)
- **clean_pollution** (datos procesados y listos para análisis)

Se eligió PostgreSQL por:

- Su **estabilidad y madurez** como base de datos relacional.
- Su **facilidad para trabajar con datos estructurados**, tal como los que obtenemos de las APIs.
- Su integración natural con dbt y con las herramientas Python utilizadas en las fases previas y posteriores.

Con esta estructura, el proyecto asegura la **trazabilidad completa** del dato y permite reproducir o depurar cualquier parte del proceso si fuese necesario.

4. Transformación mediante dbt

Para la limpieza y estandarización del dato se ha utilizado **dbt (data build tool)**. Esta herramienta resulta especialmente adecuada porque:

- Permite separar claramente los modelos **raw** de los modelos **clean**.
- Estructura la lógica de transformación de forma **ordenada y modular**.
- Facilita la **documentación automática** del flujo de datos.
- Asegura que cada ejecución sea **reproducible y controlada**, algo clave en proyectos de automatización.

La salida de esta etapa es la tabla **clean_pollution**, que constituye la capa final de análisis.

5. Visualización interactiva con Plotly/Dash

Para la fase final se ha optado por construir un **dashboard interactivo en Dash**, utilizando **Plotly** para las visualizaciones. Esta decisión se justifica por:

- La **flexibilidad total** que ofrece Dash para adaptar el dashboard al diseño del proyecto.
- La **integración directa con Python**, lo que evita depender de herramientas externas de BI y mantiene la coherencia tecnológica.
- El soporte de visualizaciones avanzadas como mapas, gráficos de tendencia o indicadores tipo velocímetro.

Un script adicional de Python convierte coordenadas, calcula distancias y prepara los datos que se representan visualmente, integrando tanto la información de contaminación como la meteorológica.

Esta elección otorga al proyecto un dashboard **personalizable, dinámico y completamente alineado con la lógica del pipeline**.

6. Arquitectura modular del pipeline

El diseño general sigue una estructura clara:

Ingesta → Streaming → Almacenamiento → Transformación → Visualización

Este enfoque modular permite:

- Sustituir cualquier pieza sin afectar a las demás.
- Facilitar el mantenimiento y la ampliación del sistema.
- Escalar fácilmente si se incorporan nuevas APIs o volúmenes mayores de datos.
- Garantizar una trazabilidad completa desde el origen hasta el dashboard.

Cada herramienta cumple una función bien definida dentro del pipeline y contribuye a que el flujo completo sea **fiable, automatizado, escalable y fácilmente interpretable**.

El resultado es una arquitectura robusta que integra correctamente la ingesta, procesamiento y visualización de los datos ambientales, permitiendo ofrecer un producto final coherente, funcional y alineado con los objetivos iniciales del proyecto.