

SMARTPARKING: Sistema Inteligente de Gestión de Aparcamiento

Pedro González Fernández



SMARTPARKING

29 de octubre de 2025

Índice

1. Introducción	3
1.1. Contextualización	3
1.2. Justificación	3
1.3. Objetivos	3
1.3.1. Objetivo General	3
1.3.2. Objetivos Específicos	4
1.4. Alcance del proyecto	4
1.5. Planificación	4
2. Marco teórico	5
2.1. Big Data y su relevancia en la gestión urbana	5
2.2. Internet de las Cosas (IoT) en la monitorización de aparcamientos .	5
2.3. Tecnologías Big Data utilizadas	5
2.3.1. Apache Kafka	5
2.3.2. Apache NiFi	5
2.3.3. MongoDB	6
2.3.4. Flask	6
2.3.5. Dremio	6
3. Fase de análisis	6
3.1. Requisitos Funcionales	6
3.2. Requisitos No Funcionales	6
3.3. Análisis de tecnologías	7
4. Fase de diseño	7
4.1. Arquitectura del sistema	7
4.2. Diseño de la Base de Datos	7
4.2.1. Diseño de los documentos	8
4.3. Diseño del Flujo NiFi	8
4.4. Diseño de la Interfaz Web	9

5. Fase de implementación	9
5.1. Configuración del entorno	9
5.2. Apache Kafka	10
5.3. MongoDB	11
5.4. Apache NiFi	12
5.4.1. Instalación y configuración	12
5.4.2. Flujo de Datos	12
5.5. Parking en tiempo real con Flask	13
5.5.1. Backend	13
5.5.2. Frontend	13
5.6. Análisis y estadística de los datos con Dremio	14
5.6.1. Instalación y configuración general	14
5.6.2. Configuración específica	14
5.6.3. Consultas de ejemplo	14
6. Fase de pruebas	15
7. Conclusiones y líneas futuras	15
7.1. Conclusiones	15
7.2. Líneas futuras	15
7.3. Lecciones aprendidas	15
8. Bibliografía	16
9. Diario de trabajo	16
10. Anexo	17
10.1. Configuración del entorno	17
10.2. Apache Kafka	18
10.3. MongoDB	20

1. Introducción

1.1. Contextualización

En la era de las 'Smart Cities', la gestión eficiente de recursos urbanos se ha convertido en una prioridad. Las plazas de aparcamiento representan uno de los desafíos más significativos en entornos urbanos sobrepoblados, donde la congestión del tráfico y la disponibilidad de plazas de aparcamiento generan pérdidas económicas, contaminación y estrés en los conductores. La tecnología **Big Data**, combinada con el **Internet de las Cosas (IoT)**, ofrece soluciones innovadoras para transformar la gestión tradicional de aparcamientos en sistemas inteligentes capaces de optimizar el espacio disponible y mejorar la experiencia del usuario.

1.2. Justificación

La implementación de un sistema inteligente de gestión de aparcamiento, como **SmartParking**, es crucial para resolver la necesidad de:

- Reducir el tiempo de búsqueda de plazas de aparcamiento, disminuyendo así la congestión del tráfico y las emisiones de CO₂.
- Proporcionar datos en tiempo real para la toma de decisiones tanto por parte de los conductores como de los gestores del parking.
- Crear una base histórica de datos que permita analizar patrones de uso y optimizar la gestión del espacio.
- Servir como caso de estudio para la aplicación de tecnologías Big Data e IoT en entornos urbanos.

1.3. Objetivos

1.3.1. Objetivo General

El objetivo general es desarrollar e implementar un sistema inteligente de monitorización de plazas de aparcamiento que permita la gestión eficiente del espacio, mediante la captura, procesamiento, almacenamiento y visualización de datos en tiempo real.

1.3.2. Objetivos Específicos

- Configurar Apache Kafka para la ingesta de datos en tiempo real desde sensores IoT.
- Diseñar flujos de datos en Apache NiFi para el procesamiento y routing de la información.
- Implementar una base de datos NoSQL (MongoDB) para el almacenamiento eficiente de grandes volúmenes de datos, así como de la gestión de plazas de aparcamiento en tiempo real.
- Desarrollar una aplicación web con Flask para la visualización del parking en tiempo real.
- Integrar Dremio para el análisis avanzado de datos y generación de informes.
- Validar el rendimiento del sistema bajo condiciones de carga simuladas.

1.4. Alcance del proyecto

El proyecto **SmartParking** se centrará en la implementación de un sistema piloto que abarque las siguientes áreas:

- Desarrollo del Backend de procesamiento de datos.
- Implementación de la base de datos.
- Creación de la interfaz web de visualización.
- Configuración de la plataforma de análisis de datos.
- Documentación técnica y memoria del proyecto.

Excluye aspectos como:

- Desarrollo de hardware específico para sensores IoT.
- Implementación en producción real.
- Sistemas de pago.
- Aplicación móvil nativa.

1.5. Planificación

El proyecto se desarrollará en varias fases a lo largo de un periodo de 4 semanas, distribuidas de la siguiente manera:

- Semana 1: Análisis y diseño del sistema.
- Semana 2: Configuración de Apache Kafka, Apache NiFi e implementación de la base de datos.
- Semana 3: Desarrollo del backend y la interfaz web e integración con dremio.
- Semana 4: Pruebas finales, desarrollo de la memoria y video demostrativo.

2. Marco teórico

2.1. Big Data y su relevancia en la gestión urbana

El Big Data se refiere al manejo y análisis de grandes volúmenes de datos que no pueden ser procesados mediante herramientas tradicionales. En el contexto urbano, el Big Data permite la recopilación y análisis de datos en tiempo real provenientes de diversas fuentes, como sensores IoT, cámaras y dispositivos móviles. Esta capacidad es fundamental para la gestión eficiente de recursos urbanos, incluyendo la optimización del tráfico y la gestión de aparcamientos.

2.2. Internet de las Cosas (IoT) en la monitorización de aparcamientos

El Internet de las Cosas (IoT) se refiere a la interconexión de dispositivos físicos a través de internet, permitiendo la recopilación e intercambio de datos. En el ámbito de la gestión de aparcamientos, los sensores IoT pueden detectar la ocupación de plazas en tiempo real, proporcionando datos valiosos para la optimización del uso del espacio y la mejora de la experiencia del usuario.

2.3. Tecnologías Big Data utilizadas

2.3.1. Apache Kafka

Apache Kafka es una plataforma de streaming distribuida que permite la ingesta y procesamiento de grandes volúmenes de datos en tiempo real. En el proyecto SmartParking, Kafka se utilizará para recibir datos de los sensores IoT y distribuirlos a los componentes del sistema.

- Tópico: Categoría o canal donde se publican los mensajes.
- Productor: Componente que envía datos a un tópico.
- Consumidor: Componente que recibe datos de un tópico.
- Broker: Servidor que almacena y distribuye los mensajes.

2.3.2. Apache NiFi

Apache NiFi es una herramienta de integración de datos que facilita el flujo, transformación y enrutamiento de datos entre sistemas. En SmartParking, NiFi se empleará para procesar los datos recibidos de Kafka y enviarlos a la base de datos y otros servicios.

- Procesadores: Componentes que realizan operaciones específicas sobre los datos.
- Flujos de datos: Definiciones de cómo los datos se mueven y transforman dentro de NiFi.

2.3.3. MongoDB

MongoDB es una base de datos NoSQL orientada a documentos que ofrece alta escalabilidad y flexibilidad en el almacenamiento de datos. En este proyecto, MongoDB se utilizará para almacenar la información de las plazas de aparcamiento en tiempo real y los datos históricos de ocupación.

- Documentos: Estructuras de datos similares a JSON que almacenan la información.
- Colecciones: Conjuntos de documentos relacionados.
- Consultas: Operaciones para recuperar y manipular datos almacenados.
- Índices: Estructuras que mejoran la velocidad de las consultas.

2.3.4. Flask

Flask es un microframework web para Python que permite el desarrollo rápido de aplicaciones web. En SmartParking, Flask se utilizará para crear la interfaz web que mostrará el estado del aparcamiento en tiempo real.

2.3.5. Dremio

Dremio es una plataforma de análisis de datos que facilita la consulta y visualización de grandes volúmenes de datos. En el proyecto, Dremio se empleará para analizar los datos almacenados en MongoDB y generar consultas sobre el uso del aparcamiento.

3. Fase de análisis

3.1. Requisitos Funcionales

- RF1: El sistema debe capturar datos de sensores IoT en tiempo real.
- RF2: El sistema debe almacenar un histórico completo de los eventos.
- RF3: El sistema debe mantener el estado actual de cada plaza.
- RF4: El sistema debe visualizar la ocupación en tiempo real.
- RF5: El sistema debe permitir el análisis de datos históricos.

3.2. Requisitos No Funcionales

- RNF1: El sistema debe ser escalable para manejar un aumento en el número de sensores.
- RNF2: El sistema debe garantizar la integridad y consistencia de los datos.

3.3. Análisis de tecnologías

La selección de tecnologías se basa en:

- Apache Kafka: Ideal para la ingesta de datos en tiempo real debido a su alta capacidad de manejo de mensajes.
- Apache NiFi: Facilita la integración y procesamiento de datos con una interfaz visual intuitiva.
- MongoDB: Proporciona flexibilidad y escalabilidad para almacenar grandes volúmenes de datos no estructurados.
- Flask: Permite un desarrollo rápido y sencillo de la interfaz web.
- Dremio: Ofrece capacidades avanzadas de análisis y visualización de datos.

4. Fase de diseño

4.1. Arquitectura del sistema

La arquitectura del sistema SmartParking se compone de los siguientes componentes principales:

Sensores IoT → Apache Kafka → Apache NiFi → MongoDB → Flask (Interfaz Web) & Dremio (Análisis de Datos)

4.2. Diseño de la Base de Datos

La base de datos MongoDB se diseñará con las siguientes colecciones principales:

- **Events:** Documentos que registran cada evento de ocupación o liberación de una plaza, con marca temporal y detalles del sensor.
- **Bays:** Documentos que representan cada plaza de aparcamiento, con su estado actual (ocupada/libre).

4.2.1. Diseño de los documentos

Ambas colecciones tendrán la misma estructura de documento base:

```
{
  "_id": ObjectId("..."),
  "bay_id": "L1-A-023",
  "parking_id": "PK-CADIZ-01",
  "level": "L1",
  "occupied": true,
  "last_event_ts": "2025-10-07T10:15:30Z",
  "metrics": {
    "temperature_c": 23.4,
    "battery_pct": 78
  },
  "updated_at": "2025-10-07T10:15:31Z"
}
```

Dónde:

- `_id`: Identificador único del documento generado por MongoDB.
- `bay_id`: Identificador único de la plaza de aparcamiento.
- `parking_id`: Identificador del parking al que pertenece la plaza.
- `level`: Nivel del parking donde se encuentra la plaza.
- `occupied`: Estado actual de la plaza (ocupada o libre).
- `last_event_ts`: Marca temporal del último evento registrado para la plaza.
- `metrics`: Objeto que contiene métricas adicionales proporcionadas por el sensor.
 - `temperature_c`: Temperatura medida por el sensor en grados Celsius.
 - `battery_pct`: Porcentaje de batería restante del sensor
- `updated_at`: Marca temporal de la última actualización del documento.

4.3. Diseño del Flujo NiFi

El flujo de datos en Apache NiFi se diseñará para:

- Consumir mensajes desde el tópico de Kafka.
- Procesar y transformar los datos según sea necesario.
- Enviar los datos procesados a MongoDB para su almacenamiento en ambas colecciones.

4.4. Diseño de la Interfaz Web

La interfaz web desarrollada con Flask mostrará:

- Mapa en tiempo real del estado de las plazas de aparcamiento.
- Codificación de colores: verde (libre), rojo (ocupado).
- Estadísticas básicas sobre la ocupación del parking.
- Diseño responsive para múltiples dispositivos

5. Fase de implementación

5.1. Configuración del entorno

Para la implementación del sistema SmartParking, he creado una máquina virtual con las siguientes características:

- Sistema Operativo: Ubuntu 24.04.03 LTS
- Memoria RAM: 8 GB
- Almacenamiento: 50 GB SSD
- Procesador: 4 núcleos

Tras la creación de la máquina, instalé algunos paquetes necesarios para su posterior uso, como Python y sus componentes.

(Anexo: Paquetes necesarios).

Posteriormente, instalé Java en su versión 11 para el sistema, y Java 21 para su uso en NiFi y Kafka.

```
smartparking@smartparking:~$ java -version
openjdk version "11.0.28" 2025-07-15
OpenJDK Runtime Environment (build 11.0.28+6-post-Ubuntu-1ubuntu124.04.1)
OpenJDK 64-Bit Server VM (build 11.0.28+6-post-Ubuntu-1ubuntu124.04.1, mixed mode, sharing)
```

(Anexo: Instalación y configuración de Java 11 & 21).

Finalmente, para finalizar con la configuración del sistema, creé el directorio dónde voy a trabajar y montar casi todo el sistema.

```
smartparking@smartparking:~$ mkdir -p smartparking
smartparking@smartparking:~$ cd smartparking/
smartparking@smartparking:~/smartparking$ ll
total 8
drwxrwxr-x  2 smartparking smartparking 4096 oct 24 15:55 ./
drwxr-x--- 14 smartparking smartparking 4096 oct 24 15:55 ../
```

5.2. Apache Kafka

Con el sistema listo, lo primero que hice fue preparar Apache Kafka para su posterior uso, dado que es el primer componente de este proyecto.

Tras descargarlo y ubicarlo en el directorio de trabajo, añadí la variable de entorno `KAFKA_HOME` al archivo `.bashrc` (Anexo: Apache Kafka).

Posteriormente, configuré Kafka, para usarlo en modo **KRaft** (sin Zookeeper):

```
GNU nano 7.2 kafka/config/server.properties*
# The rebalance will be further delayed by the value of group.initial.rebalance.delay.ms as new members join the group, up to
# The default value for this is 3 seconds.
# We override this to 0 here as it makes for a better out-of-the-box experience for development and testing.
# However, in production environments the default value of 3 seconds is more suitable as this will help to avoid unnecessary,
group.initial.rebalance.delay.ms=0

process.roles=broker,controller
node.id=1

listeners=PLAINTEXT://:9092,CONTROLLER://:9093
advertised.listeners=PLAINTEXT://localhost:9092
controller.listener.names=CONTROLLER
listener.security.protocol.map=PLAINTEXT:PLAINTEXT,CONTROLLER:PLAINTEXT
inter.broker.listener.name=PLAINTEXT

# Quorum KRaft (node.id@host:puerto_del_listener_CONTROLLER)
controller.quorum.voters=1@localhost:9093

# Carpeta de logs y metadatos
log.dirs=/home/smartparking/smartparking/kafka-data
```

(Anexo: Apache Kafka).

Tras realizar la configuración de Apache Kafka tal y como se puede ver al final del anexo Apache Kafka, arranqué el servicio Y creé el tópic que voy a usar para la ingesta de datos de SmartParking:

```
smartparking@smartparking:~$ kafka-topics.sh --create --topic smartparking.events --bootstrap-server localhost:9092
WARNING: Due to limitations in metric names, topics with a period ('.') or underscore ('_') could collide. To avoid issues it
is best to use either, but not both.
Created topic smartparking.events
smartparking@smartparking:~$
```

Con todo esto, ya estaría finalizada la configuración de Kafka, así que realicé una prueba básica en la que creo un **productor** que envía mensajes de prueba y un **consumidor** que los va a consumir, y el resultado fue el siguiente:

```
smartparking@smartparking:~$ kafka-console-consumer.sh --topic smartparking.events --bootstrap-server localhost:9092 --from-beginning
mensaje 1
mensaje 2
mensaje 3
mensaje 4
```

(Anexo: Apache Kafka).

5.3. MongoDB

Siguiendo el orden natural del proyecto, lo siguiente sería **Apache NiFi**, pero sería complicado hacer el flujo sin tener listo el destino de los datos, así que antes me encargué de configurar MongoDB.

Lo primero fue añadirlo a los repositorios por defecto de linux e instalarlo con apt.

```
if [ $(dpkg-query -f='${Package} ${Version} ${Architecture}\n' -W -f='${Package} ${Version} ${Architecture}\n' | grep -c mongodb-org) = 0 ]; then\n  echo \"Installing mongodb-org...\"\n  sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com --recv-keys 4F4E044E88D99C8436DD04893B62014602B60955\n  sudo apt install -y mongodb-org\n  echo \"Installation complete.\"\nfi
```

(Anexo: MongoDB).

Tras la instalación, habilité el servicio y accedí a Mongo con `mongosh`. Dentro creé la base de datos y ambas colecciones para el proyecto:

- **events** para almacenar el histórico de las plazas del parking.
- **bays** para almacenar el estado de cada plaza en tiempo real.

```
test> use smartparking\nswitched to db smartparking\nsmartparking> db.createCollection("events")\n{ ok: 1 }\nsmartparking> db.createCollection("bays")\n{ ok: 1 }\nsmartparking> show collections\nbays\nevents\nsmartparking>
```

Con esto, MongoDB ya estaría listo para su uso, pero antes de continuar hice algunas pruebas de inserción de datos consultas y upsert. Finalmente añadí índices para facilitar las consultas:

```
smartparking> db.events.find().pretty()\n[\n  {\n    _id: ObjectId('68fb8fd51728a92eb5ce5f47'),\n    bay_id: 'L1-A-023',\n    parking_id: 'PK-CADIZ-01',\n    level: 'L1',\n    occupied: true,\n    last_event_ts: ISODate('2025-10-07T10:15:30.000Z'),\n    metrics: { temperature_c: 23.4, battery_pct: 78 },\n    updated_at: ISODate('2025-10-07T10:15:31.000Z')\n  }\n]
```

(Anexo: MongoDB).

5.4. Apache NiFi

Teniendo listos tanto el origen como el destino de los datos, configuré la conexión entre ambos, es decir, el flujo de **Apache NiFi**.

5.4.1. Instalación y configuración

5.4.2. Flujo de Datos

5.5. Parking en tiempo real con Flask

Con el componente principal del sistema funcionando, me encargué de la parte relacionada con **representar los datos**, primero con Flask para mostrar las plazas de parking en tiempo real y posteriormente con Dremio para obtener estadísticas y patrones de los datos históricos.

Este apartado se divide en dos partes:

1. El **backend** en python que obtiene datos de Mongo y los envía al front.
2. El **frontend** que recibe los datos del backend y representa un mapa del parking en tiempo real.

5.5.1. Backend

5.5.2. Frontend

5.6. Análisis y estadística de los datos con Dremio

El último punto de este proyecto, como ya mencioné anteriormente, es configurar Dremio para consultar los datos históricos de las plazas del parking, de forma que podamos **analizar y sacar conclusiones** de los mismos, como patrones que se repiten, estadísticas de interés, etc.

5.6.1. Instalación y configuración general

5.6.2. Configuración específica

5.6.3. Consultas de ejemplo

6. Fase de pruebas

// TODO: Describir las pruebas realizadas y los resultados obtenidos aquí.

7. Conclusiones y líneas futuras

7.1. Conclusiones

Este proyecto ha demostrado la viabilidad de uso de tecnologías Big Data e IoT para la gestión inteligente de aparcamientos urbanos. La integración de Apache Kafka, Apache NiFi, MongoDB, Flask y Dremio ha permitido desarrollar un sistema capaz de capturar, procesar y visualizar datos en tiempo real, mejorando la eficiencia en la gestión del espacio de aparcamiento.

7.2. Líneas futuras

Las futuras mejoras y expansiones del proyecto SmartParking podrían incluir:

- Integración con sistemas de pago y reservas de plazas.
- Desarrollo de una aplicación móvil nativa para usuarios.
- Implementación de algoritmos de predicción basados en aprendizaje automático.
- Expansión del sistema a múltiples ubicaciones y ciudades.
- Incorporación de análisis avanzados y generación de informes personalizados.
- Optimización del rendimiento y escalabilidad del sistema.

7.3. Lecciones aprendidas

El desarrollo del proyecto SmartParking ha proporcionado valiosas lecciones en la integración de tecnologías Big Data e IoT, destacando la importancia de una planificación cuidadosa, pruebas exhaustivas y la adaptabilidad a los desafíos técnicos que surgen durante la implementación.

- La orquestación con NiFi simplifica significativamente el manejo de flujo de datos complejos.
- MongoDB ofrece el equilibrio adecuado entre flexibilidad y rendimiento para datos de sensores IoT.
- La documentación y pruebas continuas son esenciales para garantizar la calidad del sistema.

8. Bibliografía

- Documentación oficial de Apache NiFi: <https://nifi.apache.org/docs.html>
- Documentación oficial de Apache Kafka: <https://kafka.apache.org/documentation/>
- Documentación oficial de MongoDB: <https://docs.mongodb.com/>
- Documentación oficial de Flask: <https://flask.palletsprojects.com/en/2.0.x/>
- Documentación oficial de Dremio: <https://docs.dremio.com/>
- Herramientas de IA Generativa:
 - ChatGPT de OpenAI: <https://openai.com/chatgpt>
 - DeepSeek: <https://www.deepseek.com/>
 - Copilot de GitHub (Integrado en VS Code).

Nota sobre el uso de IA Generativa

El uso de estas herramientas se ha limitado a la resolución de dudas puntuales, generación de código (Script de envío de mensajes y Flask) y apoyo en la redacción de ciertos apartados del documento. Las respuestas obtenidas de estas herramientas se han contrastado con fuentes oficiales y se han usado a modo orientativo.

9. Diario de trabajo

- Semana 1: Análisis y diseño del sistema:
 - Investigación de tecnologías requeridas.
 - Definición de requisitos funcionales y no funcionales.
 - Diseño de la arquitectura del sistema y la base de datos.
- Semana 2: Configuración de Apache Kafka, Apache NiFi e implementación de la base de datos.
 - Configuración de Kafka y creación de tópicos.
 - Implementación de la base de datos MongoDB y creación de colecciones.
 - Diseño e implementación del flujo de datos en NiFi.
 - Pruebas iniciales de ingesta y almacenamiento de datos.
- Semana 3: Desarrollo del backend y la interfaz web e integración con dremio.
 - Investigación de Flask y desarrollo de la aplicación web.
 - Integración de Dremio para análisis de datos y desarrollo de consultas descriptivas.

- Semana 4: Pruebas finales, desarrollo de la memoria y video demostrativo.
 - Realización de pruebas de carga y rendimiento.
 - Documentación del proyecto y elaboración de la memoria.
 - Creación del video demostrativo del sistema.

10. Anexo

10.1. Configuración del entorno

Paquetes necesarios

Instalación de paquetes necesarios:

```
$ sudo apt update && sudo apt upgrade -y
$ sudo apt install -y curl wget git unzip build-essential
jq python3 python3-venv python3-pip net-tools
```

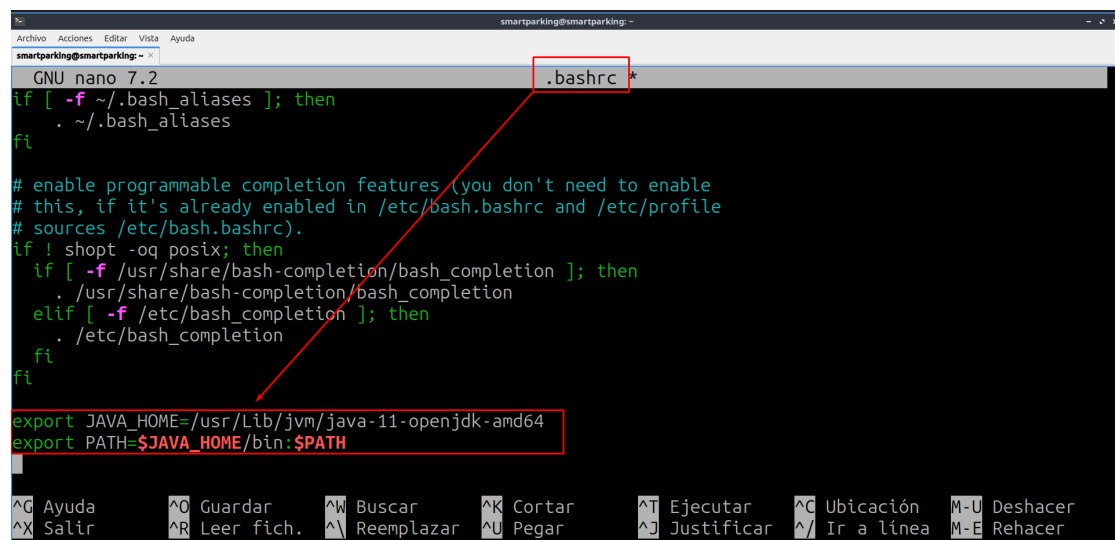
Instalación y configuración de Java 11 & 21

Instalación de Java 11:

```
$ sudo apt install -y openjdk-11-jdk
```

```
smartparking@smartparking:~$ java -version
openjdk version "11.0.28" 2025-07-15
OpenJDK Runtime Environment (build 11.0.28+6-post-Ubuntu-1ubuntu124.04.1)
OpenJDK 64-Bit Server VM (build 11.0.28+6-post-Ubuntu-1ubuntu124.04.1, mixed mode, sharing)
smartparking@smartparking:~$ javac -version
javac 11.0.28
```

Se incluye la variable JAVA_HOME en el archivo .bashrc:



```
smartparking@smartparking: ~
GNU nano 7.2 .bashrc *
if [ -f ~/.bash_aliases ]; then
  . ~/.bash_aliases
fi

# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
  if [ -f /usr/share/bash-completion/bash_completion ]; then
    . /usr/share/bash-completion/bash_completion
  elif [ -f /etc/bash_completion ]; then
    . /etc/bash_completion
  fi
fi

export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64
export PATH=$JAVA_HOME/bin:$PATH
```

Instalación de Java 21:

```
$ sudo apt install -y openjdk-21-jdk
```

Especificar la versión 11 como predeterminada del sistema:

```
$ sudo update-alternatives --config java
```

```
smartparking@smartparking:~$ sudo update-alternatives --config java
Existen 2 opciones para la alternativa java (que provee /usr/bin/java).

Selección    Ruta
-----
* 0          /usr/lib/jvm/java-21-openjdk-amd64/bin/java    2111    modo automático
1          /usr/lib/jvm/java-11-openjdk-amd64/bin/java    1111    modo manual
2          /usr/lib/jvm/java-21-openjdk-amd64/bin/java    2111    modo manual

Pulse <Intro> para mantener el valor por omisión [*] o pulse un número de selección: 1
update-alternatives: utilizando /usr/lib/jvm/java-11-openjdk-amd64/bin/java para proveer /
```

```
smartparking@smartparking:~$ java -version
openjdk version "11.0.28" 2025-07-15
OpenJDK Runtime Environment (build 11.0.28+6-post-Ubuntu-1ubuntu124.04.1)
OpenJDK 64-Bit Server VM (build 11.0.28+6-post-Ubuntu-1ubuntu124.04.1, mixed mode, sharing)
```

10.2. Apache Kafka

Descarga y ubicación de Apache Kafka:

```
smartparking@smartparking:~/smartparking$ wget https://downloads.apache.org/kafka/3.8.0/kafka_2.13-3.8.0.tgz
--2025-10-24 15:59:46-- https://downloads.apache.org/kafka/3.8.0/kafka_2.13-3.8.0.tgz
Resolviendo downloads.apache.org (downloads.apache.org)... 135.181.214.104, 88.99.208.237, 2a01:4f9:3a:2c57::2, ...
Conectando con downloads.apache.org (downloads.apache.org)[135.181.214.104]:443... conectado.
Petición HTTP enviada, esperando respuesta... 200 OK
Longitud: 120735482 (115M) [application/x-gzip]
Guardando como: 'kafka_2.13-3.8.0.tgz'

kafka_2.13-3.8.0.tgz 100%[=====] 115,14M 33,6MB/s en 4,1s

2025-10-24 15:59:50 (28,3 MB/s) - 'kafka_2.13-3.8.0.tgz' guardado [120735482/120735482]

smartparking@smartparking:~/smartparking$ tar xzf kafka_2.13-3.8.0.tgz
smartparking@smartparking:~/smartparking$ mv kafka_2.13-3.8.0 kafka
smartparking@smartparking:~/smartparking$ ls
kafka  kafka_2.13-3.8.0.tgz
smartparking@smartparking:~/smartparking$
```

Adición de la variable de entorno KAFKA_HOME al archivo .bashrc:

```
GNU nano 7.2 _bashrc *
# You may want to put all your additions into a separate file like
# ~/.bash_aliases, instead of adding them here directly.
# See /usr/share/doc/bash-doc/examples in the bash-doc package.

if [ -f ~/.bash_aliases ]; then
. ~/.bash_aliases
fi

# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
if [ -f /usr/share/bash-completion/bash_completion ]; then
. /usr/share/bash-completion/bash_completion
elif [ -f /etc/bash_completion ]; then
. /etc/bash_completion
fi
fi

export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64
export PATH=$JAVA_HOME/bin:$PATH
export KAFKA_HOME=/home/smartparking/smartparking/kafka
export PATH=$KAFKA_HOME/bin:$PATH
```

Explicación del archivo de configuración:

```
GNU nano 7.2 kafka/config/server.properties
# The rebalance will be further delayed by the value of group.initial.rebalance.delay.ms as new members join the group, up to
# The default value for this is 3 seconds.
# We override this to 0 here as it makes for a better out-of-the-box experience for development and testing.
# However, in production environments the default value of 3 seconds is more suitable as this will help to avoid unnecessary
group.initial.rebalance.delay.ms=0

process.roles=broker,controller
node.id=1

listeners=PLAINTEXT://:9092,CONTROLLER://:9093
advertised.listeners=PLAINTEXT://localhost:9092
controller.listener.names=CONTROLLER
listener.security.protocol.map=PLAINTEXT:PLAINTEXT,CONTROLLER:PLAINTEXT
inter.broker.listener.name=PLAINTEXT

# Quorum KRaft (node.id@host:puerto_del_listener_CONTROLLER)
controller.quorum.voters=1@localhost:9093

# Carpeta de logs y metadatos
log.dirs=/home/smartparking/smartparking/kafka-data
```

Este archivo configura Kafka en modo autónomo (sin Zookeeper) para desarrollo local. Define un único nodo que actúa como broker y controller, escuchando en el puerto 9092 para clientes y en el 9093 para comunicación interna, con los datos almacenados en una carpeta específica.

Configuración final de Kafka:

Generar el `cluster.id`:

```
smartparking@smartparking:~$ UUID=$(KAFKA_HOME/bin/kafka-storage.sh random-uuid)
smartparking@smartparking:~$ echo $UUID
8CI4SCDmQKGOR0vFvVrGQ
```

Formatear el almacenamiento:

```
smartparking@smartparking:~$ KAFKA_HOME/bin/kafka-storage.sh format -t $UUID -c KAFKA_HOME/config/server.properties
Formatting /home/smartparking/smartparking/kafka-data with metadata.version 3.8-IV0.
```

Verificar el `meta.properties` para ver que está todo correcto y arranco el servicio:

```
smartparking@smartparking:~$ cat /home/smartparking/smartparking/kafka-data/meta.properties
#
#Fri Oct 24 16:10:51 CEST 2025
node.id=1
directory.id=Q8rv0RdZ1TMJA_JErkr7zw
version=1
cluster.id=8CI4SCDmQKGOR0vFvVrGQ
smartparking@smartparking:~$ kafka-server-start.sh KAFKA_HOME/config/server.properties &
[1] 22567
```

Explicación de la prueba final de Kafka:

Crear un **productor** que usado para enviar algunos mensajes de prueba, usando el topico creado para el proyecto:

```
smartparking@smartparking:~$ kafka-console-producer.sh --topic smartparking.events --bootstrap-server localhost:9092
>mensaje 1
>mensaje 2
>mensaje 3
>mensaje 4
>
```

Crear un **consumidor** que lee los mensajes desde el inicio del canal de datos, para comprobar si se han enviado correctamente y si llegan a través del canal:

```
smartparking@smartparking:~$ kafka-console-consumer.sh --topic smartparking.events --bootstrap-server localhost:9092 --from-beginning
mensaje 1
mensaje 2
mensaje 3
mensaje 4
```

10.3. MongoDB

Adición del repositorio de MongoDB

```
$ wget -qO -
https://www.mongodb.org/static/pgp/server-6.0.asc | sudo
apt-key add -
$ echo "deb [ arch=amd64,arm64 ]
https://repo.mongodb.org/apt/ubuntu jammy/mongodb-org/6.0
multiverse" | sudo tee
/etc/apt/sources.list.d/mongodb-org-6.0.list
$ sudo apt update
```