Sistema de Notificaciones para Parqueaderos Universitarios con Apache Kafka y ZooKeeper

Gilbert Nicolas Rojas Sossa - **GNicolas-Rojas** usuario GitHub

Lucas Nicolas Velasco Garzón - **Vledian** usuario GitHub

José Luis Malagón Muñoz - **nagz8** usuario GitHub

Universidad de Cundinamarca UDEC

Ing. Sistemas y computación

Ingeniero: Luis Carlos Garzón

Fusagasugá – Cundinamarca

2025

Introducción

En el contexto de la gestión de parqueaderos universitarios, se propone un sistema basado en el patrón Publicador-Suscriptor, utilizando Apache Kafka y ZooKeeper. El sistema permite registrar entradas/salidas de vehículos, emitir notificaciones, y escalar fácilmente para soportar distintos usuarios.

El sistema de parqueaderos universitarios utiliza Apache Kafka como sistema de mensajería para notificaciones en tiempo real, con ZooKeeper para la coordinación de brokers. Los componentes clave son:

Publicadores (Producers): Generan eventos (entradas/salidas, multas).

Suscriptores (Consumers): Reciben notificaciones (app móvil, dashboard admin).

Broker Kafka: Maneja los tópicos (parqueaderos.entradas, parqueaderos.multas).

ZooKeeper: Gestiona la sincronización y configuración de Kafka.

2. Casos de Uso

2.1 Iniciar Sesión

Iniciador	Estudiantes, docentes o administrativos
Otros actores	Ninguno
Precondiciones	El usuario debe estar registrado en el sistema. Debe contar con credenciales de acceso válidas.

Flujo básico	
Actor	Sistema
Ingresar su correo y contraseña	Verificar las credenciales
Esperar la validacion	Si son correctas, permite el acceso.

Flujo alternativo 1	Si las credenciales son incorrectas, se muestra un mensaje de error.
Flujo alternativo 2	Si el usuario olvida su contraseña, se ofrece la opción de recuperación
Poscondiciones	El usuario tiene acceso al sistema con su perfil correspondiente.

2.2. Registrar vehículo

Iniciador	Estudiantes, docentes o administrativos
Otros actores	Ninguno
Precondiciones	El usuario debe haber iniciado sesión.
Flujo básico	

Actor	Sistema
Selecciona la opción de registrar un vehículo.	Muestra formulario de registro.
Ingresa los datos del vehículo.	Guarda la información y confirma el registro.

Flujo alternativo 1	Si falta algún dato obligatorio, el sistema solicita completarlo
Flujo alternativo 2	Si la placa ya esta registrada, muestra un mensaje de error
Poscondiciones	El vehiculo queda asociado al usuario en el sistema.

2.3. Editar vehiculo

Iniciador	Estudiantes, docentes o administrativos

Otros actores	Ninguno
Precondiciones	El usuario debe haber iniciado sesión Debe tener al menos un vehículo registrado
Flujo básico	
Actor	Sistema
Selecciona un vehiculo registrado	Muestra los datos del vehículo
Modifica los datos deseados	Guarda los cambios y confirma la actualización
Flujo alternativo 1	Si el vehiculo no existe, muestra un mensaje de error.
Flujo alternativo 2	Si no se realizan los cambios, el sistema no permite la actualización
Poscondiciones	La información del vehiculo queda guardada en el sistema

2.4. Ver vehículos registrados

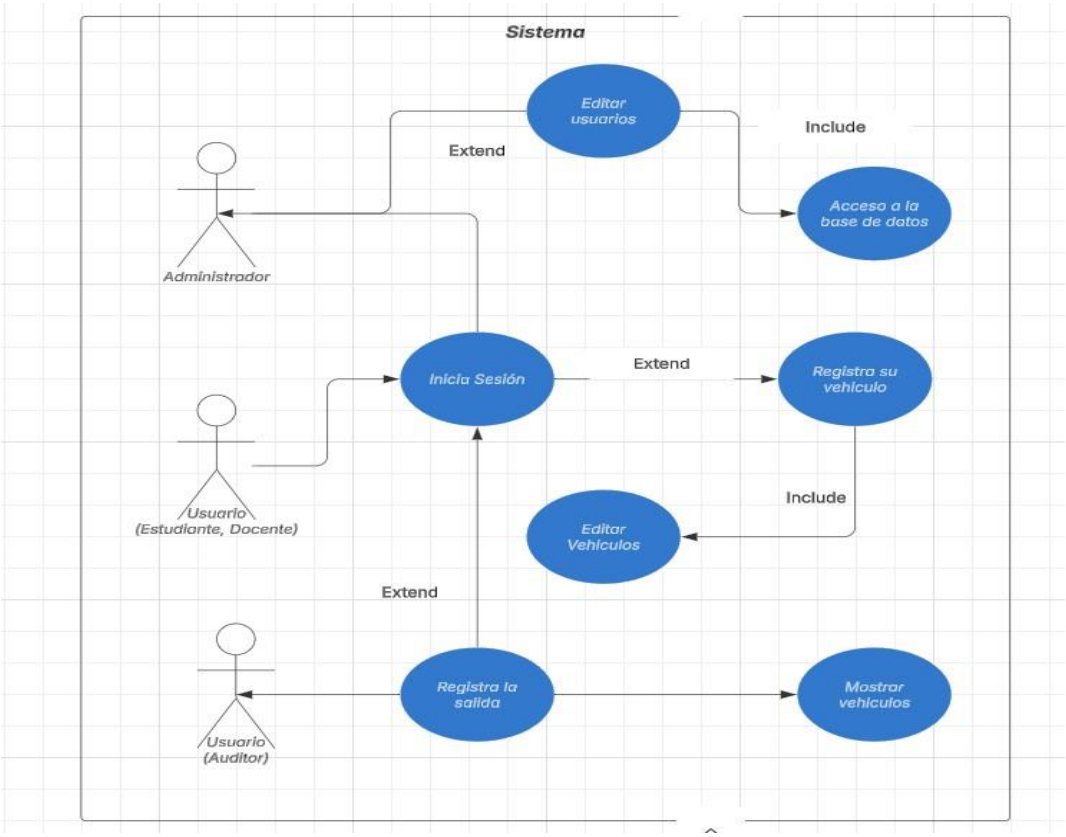
Iniciador	Estudiantes, docentes o administrativos	
Otros actores	Ninguno	
Precondiciones	El usuario debe haber iniciado sesión	
Flujo básico		
Actor		Sistema
Accede a la opción ver vehículos.		Muestra la lista de vehículos registrados
Flujo alternativo 1	Si no hay vehículos registrados, se muestra un mensaje indicando la lista esta vacía.	
Poscondiciones	El usuario visualiza los vehículos asociados con su cuenta.	

2.5. Habilitar/deshabilitar puesto de parqueadero

Iniciador	Administrador del sistema
Otros actores	Ninguno

Precondiciones	El usuario debe haber iniciado sesión como administrador. Debe existir al menos un puesto registrado.	
Flujo básico		
Actor		Sistema
Accede a la gestión de puestos de parqueo.		Muestra la lista de puestos disponibles.
Selecciona un puesto y cambia su estado.		Guarda los cambios y actualiza la disponibilidad.
Flujo alternativo 1	Si el puesto no existe, se muestra el mensaje de error.	
Flujo alternativo 2	Si ya está en el estado deseado, no se permiten cambios.	
Poscondiciones	El puesto queda habilitado o deshabilitado según la acción tomada.	

3. Diagrama casos de uso



4. Flujo simplificado de los mensajes

Link lucidchart diagrama de componentes:

https://lucid.app/lucidchart/a9020f71-3d9b-4fea-af77-5ddc1c217216/edit?invitationId=inv_d736cc0b-b2bf-4d48-9eb4-885ec14a6be0&page=0_0#

4.1 Iniciar Sesión

Iniciador: Estudiantes, docentes o administrativos

Precondiciones: Usuario registrado con credenciales válidas

Flujo básico:

Usuario ingresa correo y contraseña

Sistema verifica credenciales

Permite acceso si son correctas

Flujo alternativo:

Credenciales incorrectas → mensaje de error

Opción de recuperación de contraseña

Poscondición: Usuario autenticado

4.2 Registrar Vehículo

Iniciador: Usuario autenticado

Precondiciones: Haber iniciado sesión

Flujo básico:

Selecciona “registrar vehículo”

Llena formulario

El sistema valida y guarda

Flujo alternativo:

Falta algún dato → aviso

Placa ya registrada → error

Poscondición: Vehículo asociado al usuario

5. Patrón Arquitectónico: Publicador-Suscriptor (Publish-Subscribe)

Componentes

Producers (Publicadores): Scripts que envían eventos (ej. producer.py)

Consumers (Suscriptores): Scripts que reciben eventos (ej. consumer.py)

Broker Kafka: Tópicos: parqueaderos.entradas, parqueaderos.multas

ZooKeeper: Coordina los brokers Kafka

---

6. Configuración en Windows

6.1 Instalación y Estructura de Carpetas

Para configurar Apache Kafka en Windows, primero es necesario descargar tanto Kafka como ZooKeeper. ZooKeeper es un servicio que Kafka utiliza para coordinarse

internamente. Una vez descargados, los archivos deben organizarse en una estructura de carpetas clara para facilitar la administración. Generalmente, se recomienda tener una carpeta principal para Kafka y dentro de ella subcarpetas para binarios, configuraciones, logs y datos. Esta organización permite mantener el sistema ordenado y facilita el acceso a los diferentes componentes

Descargas:

Apache Kafka.

ZooKeeper.

6.2 Comandos para Iniciar Servicios

La ejecución de Kafka en Windows requiere abrir diferentes terminales o ventanas de comandos para iniciar cada servicio por separado. Primero, se debe levantar el servicio de ZooKeeper, ya que Kafka depende de este para funcionar correctamente. Luego, en una nueva terminal, se inicia el broker de Kafka, que es el encargado de recibir, almacenar y enviar los mensajes entre productores y consumidores. Esta separación permite que los

servicios se ejecuten de forma independiente y que se puedan monitorear individualmente.

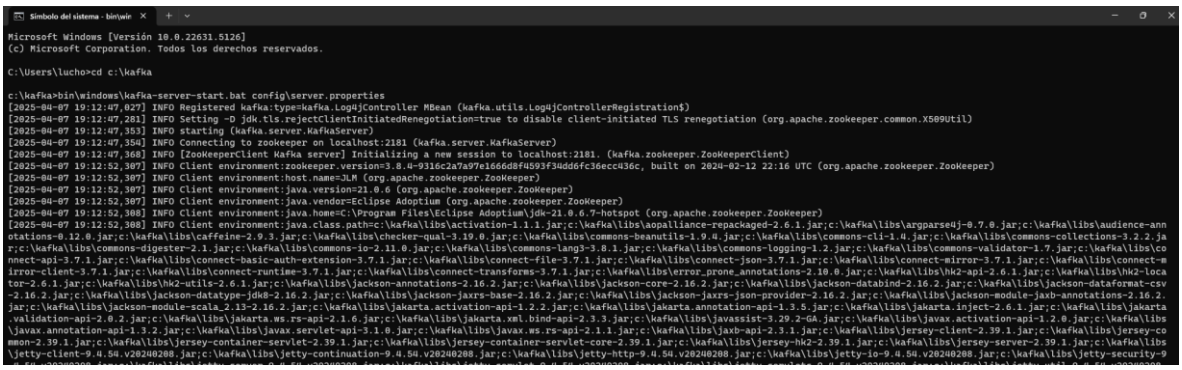
```
cd c:\kafka
bin\windows\zookeeper-server-start.bat config\zookeeper.properties
bin\windows\kafka-server-start.bat config\server.properties

cd c:\kafka
bin\windows\kafka-topics.bat --create -topic labrafatopic --bootstrap-server localhost:9092
bin\windows\kafka-topics.bat --describe --topic labrafatopic --bootstrap-server localhost:9092

cd c:\kafka
bin\windows\kafka-console-producer.bat --topic labrafatopic --bootstrap-server localhost:9092
bin\windows\kafka-console-consumer.bat --topic labrafatopic --from-beginning --bootstrap-server localhost:9092

server:
log.dirs=C:\kafka\kafka-logs
zookeeper:
dataDir=C:\kafka\zookeeper
```

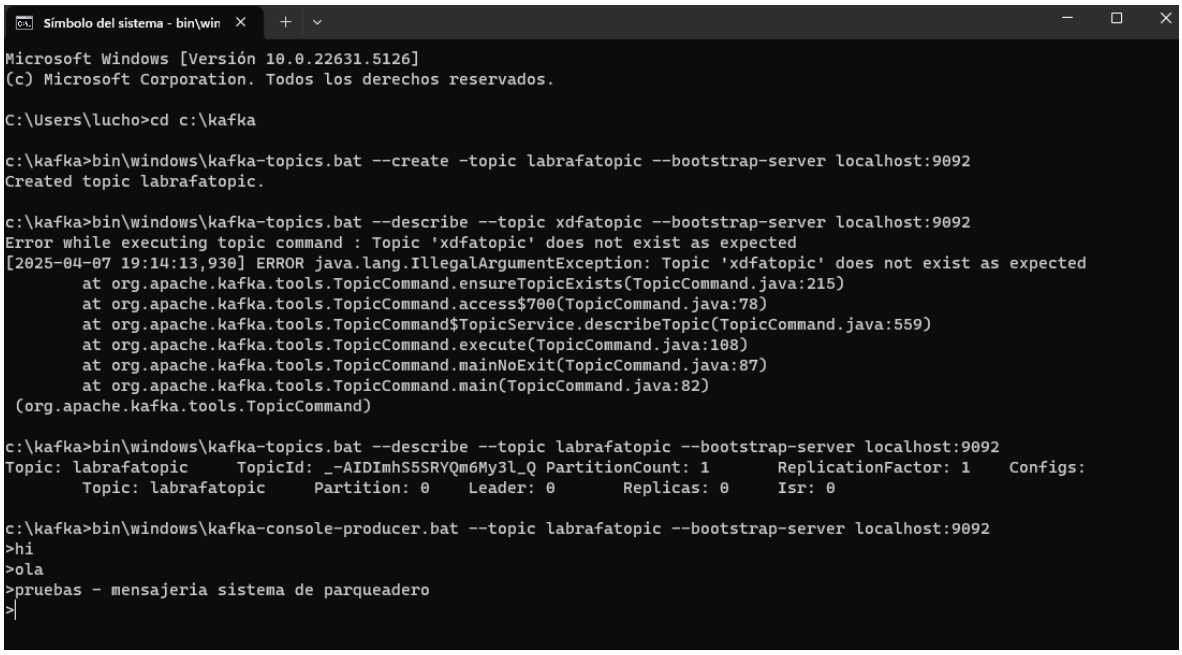
Terminal 1 (ZooKeeper):

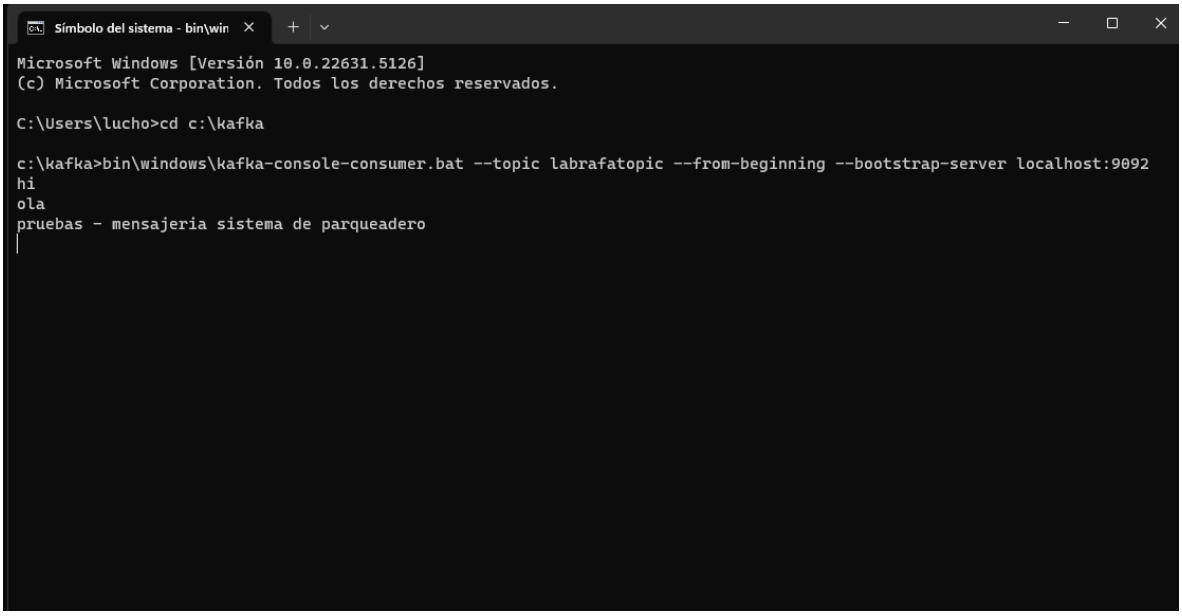


Terminal 2 (Kafka Broker):

6.3 Creación de Tópicos

Una vez que los servicios de ZooKeeper y Kafka están en funcionamiento, el siguiente paso es crear los tópicos. Un tópico es un canal o categoría donde se publican y consumen los mensajes dentro de Kafka. La creación de tópicos permite establecer cómo se organizará la información en el sistema, definiendo aspectos como el nombre del tópico, el número de particiones y el factor de replicación. Estos parámetros influyen en la distribución, tolerancia a fallos y rendimiento del sistema de mensajería.





7. Implementación con Python

7.1 productor

El productor Kafka desarrollado en Python permite enviar mensajes a un tópico previamente creado. El código establece conexión con el broker Kafka utilizando la

librería kafka-python, define el tópico de destino y estructura los mensajes que serán enviados (por ejemplo, información de vehículos o usuarios). Este componente es clave para simular entradas al sistema en tiempo real, y puede integrarse fácilmente con formularios web, sensores u otras fuentes de datos

7.2 consumidor

El consumidor Kafka está diseñado para recibir mensajes desde un tópico específico. Una vez conectado al broker, escucha continuamente y procesa cada mensaje recibido. Dependiendo del diseño, puede mostrar los datos en consola, almacenarlos en una base de datos, o generar alertas. Este componente representa la parte que interpreta y utiliza la información publicada por el productor.

Repositorio de GitHub

el enlace al repositorio de GitHub con ambos códigos en Python: Productor y Consumidor Kafka.

https://github.com/nagz8/Kafka

---

9. Conclusiones y Recomendaciones

Ventajas:

Escalabilidad: Soporta múltiples sensores y usuarios.

Tolerancia a Fallos: Mensajes persisten incluso si un servicio se detiene.

Mejoras Futuras:

Añadir tópicos para emergencias (parqueaderos.alertas).

Integrar con una base de datos para historial de eventos.

Referencias

Referencias IEEE Computer Society. (2014). SWEBOK V3.0: Guide to the Software Engineering Body of Knowledge. Https://pregrado.ucundinamarca.edu.co/pluginfile.php/987233/mod_assign/introattachment/0/SWEBOK%203.0.pdf?Forcedownload=1

Lucid Software (2024). Diagrama en Lucidchart. https://www.googleadservices.com/pagead/aclk?sa=L&ai=DChsSEwjr7uT3zceMAxUOmloFHWCRAbwYACICCAEQABoCdnU&co=1&gclid=EAIaIQobChMI6-7k983HjAMVDppaBR1gkQG8EAAYASAAEgKriPD_BwE&ohost=www.google.com&cid=CAASJeRotM53md-c-

Guia instalación Kafka (2024) https://youtu.be/E3kfS5kWL5c?si=-vfm5e3hXKGNj4iP