PRÁCTICA 1 SD

Práctica no guiada: Sockets, Streaming de Eventos,
Colas y modularidad.
EVCharging Network

YOUSSEF SERROUKH AFELLAD 49436184F
ISRAEL IZQUIERDO SANCHIS 50387916E

CURSO: 2025/26

ÍNDICE

1. Introducción

Esta práctica consiste en el desarrollo de un sistema distribuido que simula una infraestructura de carga para vehículos eléctricos. El sistema se divide en varios procesos independientes que se ejecutan en distintas máquinas y se comunican mediante Apache Kafka y sockets TCP.

El objetivo de esta práctica es:

- Registrar y monitorizar puntos de carga
- Gestionar solicitudes de carga de los conductores
- Controlar el proceso de suministro eléctrico
- Detectar y comunicar fallos en tiempo real
- Administrar la infraestructura desde una central

1. Componentes Software Desarrollados

1.1. EV_Central

- Coordina la red de puntos de carga
- Recibe registros y actualizaciones desde los CP
- Autoriza o deniega solicitudes de carga
- Envía comandos: Para, Reanudar, Emergencia
- Guarda información en base de datos
- Incluye una interfaz gráfica que muestra el estado de cada punto de carga, sesiones en curso y log de eventos.

Funcionalidades del código

El módulo EV_Central es el encargado de coordinar y gestionar el sistema de recarga de vehículos eléctricos. Al iniciarse, configura el nivel de log para ocultar mensajes internos de Kafka y valida que se haya indicado la dirección del servidor Kafka. Luego, establece conexión con la base de datos mediante DBManager.connect() e inicializa un KafkaProducer y un KafkaConsumer, que permiten enviar y recibir mensajes hacia y desde los puntos de recarga (CP). Con estos componentes se crea un Servidor Concurrente, que se ejecuta en segundo plano escuchando solicitudes entrantes. Paralelamente, se lanza una interfaz gráfica (CentralDashboard) y un hilo que permite que el operador introduzca comandos manuales para controlar los CP: P para parar, R para reanudar y E para activar la parada de emergencia, enviándole dichas órdenes al tópico comandos-cp. La central permanece activa mediante un bucle que mantiene la ejecución hasta que se solicite su cierre, momento en el cual se

cierran correctamente los recursos de entrada y de mensajería Kafka. En resumen, este módulo actúa como núcleo del sistema, recibiendo información de los cargadores, gestionando su estado, permitiendo control manual y coordinando la comunicación a través de Kafka.

1.2. EV_CP_E (Engine)

- Controla la carga física (simulada)
- Reporta consumo y estado mediante Kafka
- Recibe órdenes desde la Central
- Genera ticket al finalizar la sesión

Funcionalidades del código

El módulo EV_CP_E es el encargado de representar el funcionamiento del punto de carga (Charging Point Engine). Al iniciarse, recibe los parámetros de identificación del CP, ubicación, precio por kWh y los datos de conexión hacia la Central y Kafka. Crea un objeto ChargingPoint que mantiene el estado interno del cargador y levanta un servidor de monitorización (MonitorServer) que permite la comunicación con el módulo EV_CP_M mediante sockets, iniciando también un hilo para escuchar periódicamente si el monitor está activo; si éste deja de responder, el CP se marca como desconectado en la base de datos. Además, el módulo crea un consumidor Kafka que escucha en el topic comandos-cp para recibir órdenes remotas de la Central, como Parar, Reanudar o activar Parada de Emergencia, y gestiona también la autorización y el comienzo automático del suministro de energía. El programa ejecuta un menú interactivo que permite al técnico realizar manualmente acciones como iniciar o finalizar la carga, activar/parar el punto, simular o reparar averías y consultar el estado. En caso de fallo simulado, se envía un mensaje al topic fallo-cp, y si se repara, se notifica por recuperacion-cp. Finalmente, el método detener() cierra conexiones, detiene hilos de servicio, actualiza el estado del CP en la base de datos y libera todos los recursos, garantizando un cierre controlado del módulo.

1.3. EV_CP_M (Monitor)

- Supervisa el Engine
- Detecta fallos y reporta averías a la Central
- Se comunica mediante sockets TCP

Funcionalidades del código

El módulo EV_CP_M, que actúa como monitor del punto de carga y cuya función principal es supervisar continuamente el estado del módulo EV_CP_E (Engine) e

informar a la Central de posibles fallos o recuperaciones. Al iniciarse, el monitor registra su presencia en la Central enviando un mensaje al topic monitor-registro y establece posteriormente una conexión por socket con el Engine para realizar un intercambio periódico de señales de vida mediante mensajes MONITOR_ACTIVO y MONITOR_ACTIVO_ACK. Si el Engine deja de contestar o no puede conectarse, el monitor genera automáticamente un evento de avería enviándolo al topic monitor-averías, diferenciando entre avería inicial (si el Engine nunca estuvo disponible al arrancar) o avería durante la ejecución (detectada en funcionamiento normal). Igualmente, cuando el Engine vuelve a estar operativo, el monitor detecta la recuperación y envía un mensaje Recuperacion_Reporte a la Central. Todo este proceso se realiza mediante un hilo que comprueba continuamente el estado del Engine sin detener la ejecución del resto del sistema. Finalmente, cuando se detiene el monitor, se cierran conexiones y se liberan recursos de forma controlada.

1.4. EV_Driver

- Simula un usuario solicitando carga
- Puede funcionar en modo menú, de forma interactiva, o en modo automático, con un archivo de servicios.
- Recibe autorización o denegación desde la Central

Funcionalidades del código

El módulo EV_Driver es el encargado de simular el comportamiento de un conductor solicitando servicios de recarga en los puntos de carga del sistema. Al iniciarse, el programa configura un KafkaProducer y un KafkaConsumer, suscribiéndose a los tópicos privados del conductor (driver-autorizacion-ID, driver-estado-ID, driver-ticket-ID y driver-error-ID) para recibir respuestas personalizadas desde la Central y los puntos de carga. El consumo de mensajes se realiza en un hilo independiente para permitir que el usuario interactúe simultáneamente con el menú. El conductor puede operar en dos modos: modo manual, donde se muestra un menú interactivo para solicitar recarga y consultar el estado actual; y modo automático, donde el programa lee un archivo con una lista de puntos de carga y realiza solicitudes secuenciales, esperando a que cada servicio finalice antes de iniciar el siguiente. Cuando se solicita un servicio, el módulo envía un mensaje Solicitud_Servicio al topic driver-solicitud. Si la Central autoriza la recarga, el driver recibe un mensaje Autorizado, mostrando la sesión asignada y el punto donde debe conectar el vehículo; si se deniega, se notifica igualmente. Durante la carga, el usuario puede recibir actualizaciones periódicas del consumo y coste, y al finalizar se presenta automáticamente un ticket completo con los valores finales. Finalmente, al cerrar el driver se realiza

un apagado ordenado deteniendo hilos, cerrando el consumidor y el productor, y liberando recursos.

2. Base de Datos

Se utiliza MariaDB con Docker
Tablas utilizadas:

- driver: Guarda la información de los drivers
- charging_point: Guarda la información y estado de los puntos de carga
- charging_session: Guarda la información de las sesiones activas y finalizadas
- event_log: Registro de eventos reportados

3.Guía de Despliegue Paso a Paso

3.1. Preparación del entorno Kafka

cd ~/Kafka
bin/kafka-server-start.sh config/server.properties

3.2. Creación de topics

cd ~/Kafka
bin/kafka-topics.sh --create --topic cp-registro --bootstrap-server localhost:9092 --partitions 1 --replication-factor 1
bin/kafka-topics.sh --create --topic cp-estado --bootstrap-server localhost:9092 --partitions 1 --replication-factor 1
bin/kafka-topics.sh --create --topic cp-autorizacion --bootstrap-server localhost:9092 --partitions 1 --replication-factor 1
bin/kafka-topics.sh --create --topic actualizacion-recarga --bootstrap-server localhost:9092 --partitions 1 --replication-factor 1
bin/kafka-topics.sh --create --topic fallo-cp --bootstrap-server localhost:9092 --partitions 1 --replication-factor 1
bin/kafka-topics.sh --create --topic recuperacion-cp --bootstrap-server localhost:9092 --partitions 1 --replication-factor 1
bin/kafka-topics.sh --create --topic central-to-cp --bootstrap-server localhost:9092 --partitions 1 --replication-factor 1
bin/kafka-topics.sh --create --topic sistema-eventos --bootstrap-server localhost:9092 --partitions 1 --replication-factor 1
bin/kafka-topics.sh --create --topic monitor-registro --bootstrap-server localhost:9092 --partitions 1 --replication-factor 1
bin/kafka-topics.sh --create --topic ticket --bootstrap-server localhost:9092 --partitions 1 --replication-factor 1

bin/kafka-topics.sh --create --topic comandos-cp --bootstrap-server localhost:9092 --partitions 1 --replication-factor 1
bin/kafka-topics.sh --create --topic monitor-averias --bootstrap-server localhost:9092 --partitions 1 --replication-factor 1

Para ver que los topics se han creado de manera correcta:
bin/kafka-topics.sh --list --bootstrap-server localhost:9092

3.3. Despliegue de MariaDB en Docker

cd p2/
docker compose build
docker compose up -d
docker compose ps

Entrar en la base de datos (la contraseña de la base de datos es evpass):
docker exec -it p2-mariadb-1 mysql -u evuser -p ev_charging_system

Insertar datos iniciales:
docker exec -i p2-mariadb-1 mysql -u evuser -p evpass ev_charging_system < p2/central/sql/inserts.sql

3.4. Compilación del Proyecto

cd ~/eclipse-workspace/SD-base/src
javac -cp "p2/libs/*" p2/db/DBManager.java p2/evcharging/cp/*.java p2/evcharging/cp/network/*.java p2/evcharging/cp/service/*.java
javac -cp "p2/libs/*" p2/db/DBManager.java p2/central/*.java
javac -cp "p2/libs/*" p2/driver/EV_Driver.java

Esta es la compilación manual, tambien tenemos un compilar.sh que te lo hace de manera automática, si te pide permisos, tienes que darselo con chmod -x compilar.sh, esto se hace con todos los sh que voy a decir que

3.5. Ejecución

Central:
java -cp ".:p2/libs/*:p2" p2.central.EV_Central localhost:9092
Engine:

java -cp ".:p2/libs/*" p2.evcharging.cp.EV_CP_E CP001 "Calle Principal 123" 0.15 localhost:9090 localhost:9092 8080

Monitor:
java -cp ".:p2/libs/*:p2" p2.evcharging.cp.EV_CP_M localhost 8080 CP001 localhost:9092 8080

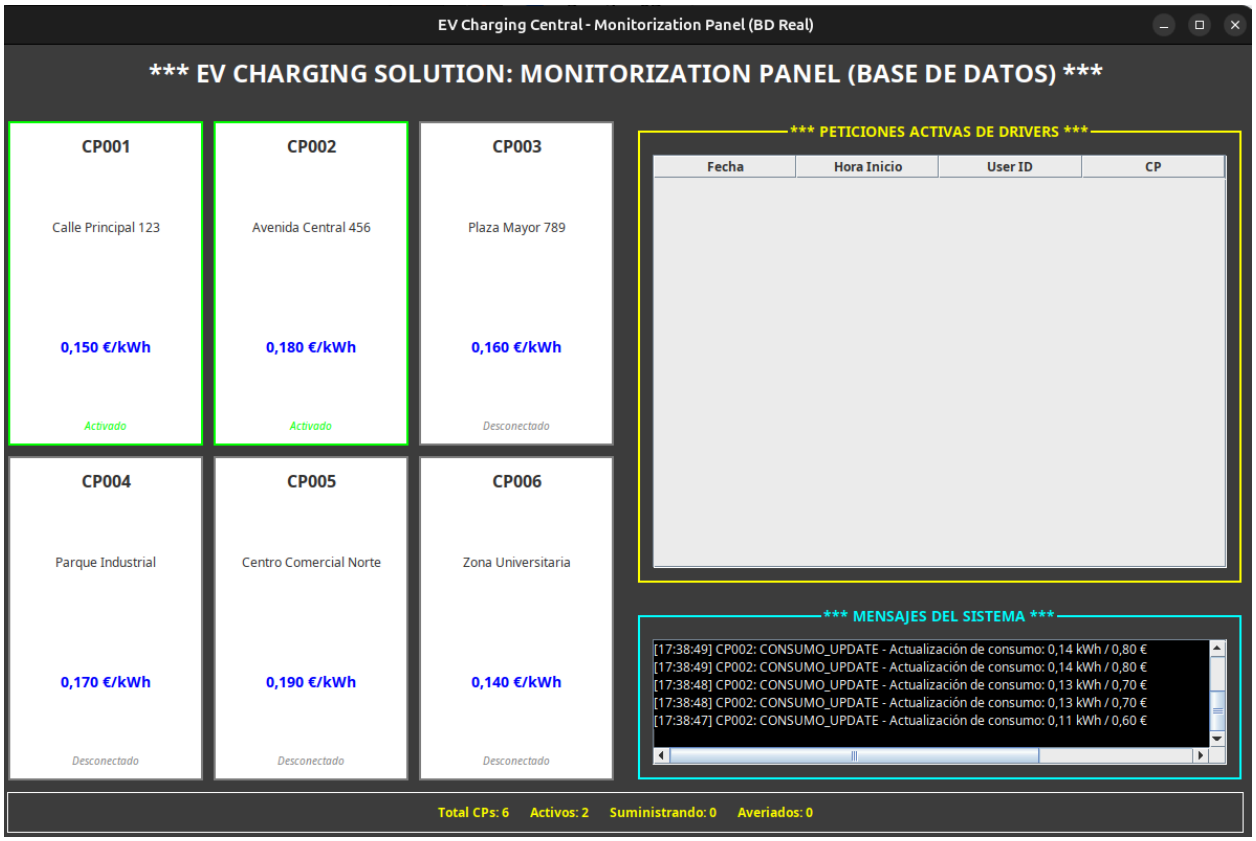
Driver (modo menú):
java -cp ".:p2/libs/*:p2" p2.driver.EV_Driver localhost:9092 DRIVER001

Para iniciar el driver con el archivo de texto lo que hay que ejecutar es lo siguiente: java -cp ".:p2/libs/*:p2" p2.driver.EV_Driver localhost:9092 DRIVER001 ruta_del_archivo/servicios.txt

Aunque todo esto se puede ejecutar con el central.sh para levantar la central y luego ejecutamos el run.sh que levanta todo lo demás donde para elegir las cosas que levantar se tienen que comentar o descomentar las líneas correspondientes, esto tiene incorporados los drivers en modo menú si se quiere hacerlos en modo con el archivo de texto se tiene que lanzar manualmente con el comando anterior

4.Funcionamiento

Esta es la interfaz gráfica en de la central que está sincronizada con la base de datos, y como se puede observar hay dos CPs activos, con sus engines y monitores respectivos.



Aquí se observa cómo coincide la información del terminal con la interfaz gráfica, y a través del terminal, se puede iniciar un suministro y finalizarlo. Al igual que se puede activar, desactivar un CP, simular y arreglar una avería, o simplemente ver el estado completo del CP. El estado completo también se observa en la interfaz gráfica.

```
=== ESTADO COMPLETO CP CP001 ===
Ubicación: Calle Principal 123
Precio: 0.15 €/kWh
Estado: ACTIVADO (VERDE)
Salud: OK
Registrado: SÍ
================================

--- MENÚ PRINCIPAL ---
1.  Iniciar suministro manual
2.  Finalizar suministro
3.  Activar CP
4.  Parar CP
5.  Simular avería
6.  Reparar avería
7.  Estado completo
8.  Salir
Seleccione opción:
```

```
=== ESTADO COMPLETO CP CP002 ===
Ubicación: Avenida Central 456
Precio: 0.18 €/kWh
Estado: ACTIVADO (VERDE)
Salud: OK
Registrado: SÍ
================================

--- MENÚ PRINCIPAL ---
1.  Iniciar suministro manual
2.  Finalizar suministro
3.  Activar CP
4.  Parar CP
5.  Simular avería
6.  Reparar avería
7.  Estado completo
8.  Salir
Seleccione opción:
```

El driver también puede solicitar un servicio al CP, como se observa en la imagen. Al hacerlo, comienza el suministro en el CP, y al finalizarlo, se imprime un ticket con el id del driver y lo que ha gastado.

```
--- MENÚ DRIVER DRIVER001 ---
1. Solicitar servicio en CP
2. Ver estado actual
3. Salir
Seleccione opción:
```

```
Ticket enviado a DRIVER001 - CP001: 0,93 kW (6,20 €)
[CENTRAL] Solicitud recibida de driver: CP001 -> Ticket|DRIVER001|6,20|0,93
```