**INTRODUCCIÓN**

Este proyecto es en parte la continuación o la extensión de la POC anterior, donde creábamos un Config-Server al que se conectaban todos los microservicios para obtener sus respectivas configuraciones. Además, también configuramos un Spring Cloud Bus, que nos permitía actualizar atributos concretos de los microservicios sin la necesidad de reiniciarlos. En este caso, la diferencia será el Message Broker que usaremos, en la anterior usábamos Rabbit MQ y en esta veremos como hacerlo con Kafka.

Por lo tanto para la configuración previa del proyecto (repositorio, microservicios, Config-Server y Spring Cloud Bus) podemos utilizar el documento anterior para evitar repetirlo en este. Ya que la base será la misma, simplemente cambiaremos algunas dependencias que mas adelante explicare.

El documento se llama “ConfigServer – SpringCloudBus.pdf” y deberemos leerlo hasta el apartado ***Spring Cloud Bus & Rabbit-MQ*** (pág. 11).

Lo que si que veremos en este documento es como hacer la configuración previa de Kafka para poder usarlo como Message Broker. Y seguidamente, la configuración necesaria en los microservicios para que se conecten a este.

Dicho esto vamos a entrar mas en detalle.

**CONFIGURACIÓN KAFKA**

Para este paso, hay mas de una manera en que se puede instalar Kafka para usarlo con nuestros microservicios. En un inicio trate de descargarlo directamente de la página oficial de Kafka: <https://kafka.apache.org/downloads>

Pero lo cierto es que tuve muchas complicaciones para instalarlo y al final no lo conseguí. Ya que si no recuerdo mal, algunos de los comandos que requería utilizar estaban bloqueados por el ordenador de la empresa.

De todas manera tenemos una alternativa, que es utilizar una imagen de Kafka que es lo que termine utilizando para esta POC. La imagen me lo proporciono mi equipo y estará al final del documento por si se necesita.

Primero de todo, es importante destacar que no deberíamos usar Docker para levantar Kafka, ya que este puede suponernos un coste monetario no deseado. Para ello existen otras alternativas totalmente gratuitas, al menos para el tamaño de este proyecto.

La alternativa que usaremos se llama **Rancher Desktop**, una plataforma de Kubernetes que nos permitirá desplegar con facilidad diferentes imágenes. Deberemos instalarlo en nuestro ordenador.

Una vez lo tengamos instalado y abierto, deberemos dirigirnos al botón de “Preferences”

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

Y una vez aquí, deberemos dirigirnos al apartado de Container Engine y asegurarnos de que tenemos marcada la opción de “Containerd”

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

De esta manera no usaremos Docker, si no que usaremos la alternativa gratuita **nerdctl.**

Una vez tengamos esto, con el Rancher Desktop iniciado y nuestro archivo docker-compose localizado (en mi caso esta nombrado como rancher-compose.yml). Desde una terminal de comandos ejecutaremos lo siguiente:



Esto iniciara las diferentes imágenes. Le podemos dar a Ctrl + C para parar el comando, y seguidamente escribir el siguiente comando para comprobar que los contenedores están activos: ***nerdctl ps***

Texto

El contenido generado por IA puede ser incorrecto.

Debemos fijarnos que todos los contenedores estén UP. Para comprobar que todo funciona correctamente, podemos entrar a la siguiente URL:

[**http://localhost:9021/**](http://localhost:9021/)

Deberiamos ver el Confluent con nuestro cluster activo:

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

Podemos entrar a este para comprobar la información que necesitemos. Con esto ya tenemos Kafka activo y funcionando.

**CONFIGURACIÓN MICROSERVICIOS**

Para este apartado, como ya he comentado en la introducción, la base que usaremos ya está hecha de la anterior POC, por lo que hasta aquí ya deberíamos tener las dependencias necesarias para que funcione correctamente el config-server, el repositorio con los distintos perfiles, etc. Y sobre todo tener hecha la parte del **Client Actuator**, que empieza al final de la pág. 9 del anterior documento.

Por eso, vamos a empezar directamente con el Spring Cloud Bus. Para esto, como ya tenemos configurado Kafka, lo primero será añadir la siguiente dependencia a todos los microservicios (no al config-server):

Texto

El contenido generado por IA puede ser incorrecto.

Y seguidamente en sus respectivos application.yml añadiremos la siguiente configuración:

**Texto

El contenido generado por IA puede ser incorrecto.**

Cada microservicio deberá tener en la parte de **group:** el siguiente formato:

**{nombre-del-servicio}-group**

Una vez tengamos esto, tendremos un nuevo endpoint.

**/actuator/busrefresh**

Esto nos permitirá llamar a un microservicio para que se actualice tanto ese como todos los demás a la vez. De esta manera no tenemos que ir uno por uno llamándolos para que se actualicen con el *actuator/refresh.*

Por ultimo solo debemos añadir dos cosas al config-server. Primero las siguientes dependencias:

Texto

El contenido generado por IA puede ser incorrecto.

Y por último su configuración para su application.yml:

**Texto

El contenido generado por IA puede ser incorrecto.**

Y con esto habremos terminado, con la dependencia de monitor nos permite crear un WEBHOOK para ponerlo en nuestro repositorio y que se llame automáticamente a http://localhost:8888/monitor

De todas manera, esta todo bien explicado en el anterior documento, en la pág. 14 se explica los del **bushrefresh**, y en la siguiente, la pág.15 se explica la dependencia de **monitor**.

**EXTRA**

Cabe destacar que para esta POC no he podido comprobar que el endpoint de monitor funcione con un webhook, ya que está en un entorno local y el ordenador de NTT no permite hacer tunneling o exponer los puertos locales. El endpoint de busrefresh si que he podido comprobarlo y todo funciona correctamente junto a Kafka, se actualizan todos los microservicios que han tenido algún cambio.

De todas maneras, la configuración es la misma que en la anterior POC y en esa si que pude comprobar que todo funcionara con un webhook, por lo tanto creo que el comportamiento debería ser el mismo.

Adjunto imagen del tópico que se crea al iniciar los microservicios de esta POC, y que se utiliza para actualizarlos:

**Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.**

**IMAGEN DE KAFKA**

Esta imagen fue compartida por mi equipo para que pudiera tener un entorno de Kafka desplegado y poder hacer así las pruebas de la POC.

*¡PARA MI PROYECTO HE COMENTADO LA ULTIMA LINEA DEL DOCUMENTO, YA QUE ME GENERABA ERRORES. PERO SI SE UTILIZA ESTA IMAGEN QUIZAS ES NECESARIO DESCOMENTARLA DE NUEVO!*

Texto

El contenido generado por IA puede ser incorrecto.

Si no me equivoco esta línea sirve para crear una terminal virtual, que en mi caso no era necesario.

**CONTENIDO:**

---

version: "3.4"

services:

broker:

container\_name: broker

depends\_on:

- zookeeper

environment:

KAFKA\_ADVERTISED\_LISTENERS: PLAINTEXT://broker:29092,PLAINTEXT\_HOST://localhost:9092

KAFKA\_BROKER\_ID: 1

KAFKA\_GROUP\_INITIAL\_REBALANCE\_DELAY\_MS: 0

KAFKA\_JMX\_HOSTNAME: localhost

KAFKA\_JMX\_PORT: 9101

KAFKA\_LISTENER\_SECURITY\_PROTOCOL\_MAP: PLAINTEXT:PLAINTEXT,PLAINTEXT\_HOST:PLAINTEXT

KAFKA\_OFFSETS\_TOPIC\_REPLICATION\_FACTOR: 1

KAFKA\_TRANSACTION\_STATE\_LOG\_MIN\_ISR: 1

KAFKA\_TRANSACTION\_STATE\_LOG\_REPLICATION\_FACTOR: 1

KAFKA\_ZOOKEEPER\_CONNECT: zookeeper:2181

KAFKA\_METRIC\_REPORTERS: io.confluent.metrics.reporter.ConfluentMetricsReporter

CONFLUENT\_METRICS\_REPORTER\_BOOTSTRAP\_SERVERS: broker:29092

CONFLUENT\_METRICS\_REPORTER\_TOPIC\_REPLICAS: 1

CONFLUENT\_METRICS\_ENABLE: 'true'

CONFLUENT\_SUPPORT\_CUSTOMER\_ID: 'anonymous'

hostname: broker

image: confluentinc/cp-server:7.3.1

ports:

- 29092:29092

- 9092:9092

- 9101:9101

schema-registry:

container\_name: schema-registry

depends\_on:

- broker

environment:

SCHEMA\_REGISTRY\_HOST\_NAME: schema-registry

SCHEMA\_REGISTRY\_KAFKASTORE\_BOOTSTRAP\_SERVERS: broker:29092

SCHEMA\_REGISTRY\_LISTENERS: http://0.0.0.0:8081

hostname: schema-registry

image: confluentinc/cp-schema-registry:7.3.1

platform: linux/arm64/v8

ports:

- 8081:8081

zookeeper:

container\_name: zookeeper

environment:

ZOOKEEPER\_CLIENT\_PORT: 2181

ZOOKEEPER\_TICK\_TIME: 2000

hostname: zookeeper

image: confluentinc/cp-zookeeper:7.3.1

ports:

- 2181:2181

control-center:

container\_name: control-center

depends\_on:

- broker

- connect

environment:

# CONTROL\_CENTER\_BOOTSTRAP\_SERVERS: 'broker:9092'

# CONTROL\_CENTER\_CONNECT\_DC1\_CLUSTER: http://connect:8083

# CONTROL\_CENTER\_CONNECT\_CONNECT-DC1\_CLUSTER: 'connect:8083'

# CONTROL\_CENTER\_KAFKA\_DC1\_BOOTSTRAP\_SERVERS: 'broker:9092'

# CONTROL\_CENTER\_REPLICATION\_FACTOR: 1

# CONTROL\_CENTER\_INTERNAL\_TOPICS\_PARTITIONS: 1

# CONTROL\_CENTER\_MONITORING\_INTERCEPTOR\_TOPIC\_PARTITIONS: 1

# CONTROL\_CENTER\_DEPRECATED\_VIEWS\_ENABLE: "true"

# CONFLUENT\_METRICS\_TOPIC\_REPLICATION: 1

# PORT: 9021

CONTROL\_CENTER\_BOOTSTRAP\_SERVERS: broker:29092

CONTROL\_CENTER\_SCHEMA\_REGISTRY\_URL: http://schema-registry:8081

CONTROL\_CENTER\_REPLICATION\_FACTOR: 1

CONTROL\_CENTER\_INTERNAL\_TOPICS\_PARTITIONS: 1

CONTROL\_CENTER\_MONITORING\_INTERCEPTOR\_TOPIC\_PARTITIONS: 1

CONFLUENT\_METRICS\_TOPIC\_REPLICATION: 1

PORT: 9021

CONTROL\_CENTER\_CONNECT\_CONNECT-DEFAULT\_CLUSTER: 'connect:8083'

CONTROL\_CENTER\_KSQL\_KSQLDB1\_URL: "http://ksqldb-server:8088"

CONTROL\_CENTER\_KSQL\_KSQLDB1\_ADVERTISED\_URL: "http://localhost:8088"

hostname: control-center

image: confluentinc/cp-enterprise-control-center:7.3.1

ports:

- 9021:9021

rest-proxy-local:

image: confluentinc/cp-kafka-rest:7.3.1

depends\_on:

- broker

- schema-registry

ports:

- 8082:8082

hostname: rest-proxy-local

container\_name: rest-proxy-local

environment:

KAFKA\_REST\_HOST\_NAME: rest-proxy-local

KAFKA\_REST\_BOOTSTRAP\_SERVERS: 'broker:29092'

KAFKA\_REST\_LISTENERS: "http://0.0.0.0:8082"

KAFKA\_REST\_SCHEMA\_REGISTRY\_URL: 'http://schema-registry:8081'

connect:

image: cnfldemos/cp-server-connect-datagen:0.5.3-7.1.0

hostname: connect

container\_name: connect

depends\_on:

- broker

- schema-registry

ports:

- "8083:8083"

environment:

CONNECT\_BOOTSTRAP\_SERVERS: 'broker:29092'

CONNECT\_REST\_ADVERTISED\_HOST\_NAME: connect

CONNECT\_GROUP\_ID: compose-connect-group

CONNECT\_CONFIG\_STORAGE\_TOPIC: docker-connect-configs

CONNECT\_CONFIG\_STORAGE\_REPLICATION\_FACTOR: 1

CONNECT\_OFFSET\_FLUSH\_INTERVAL\_MS: 10000

CONNECT\_OFFSET\_STORAGE\_TOPIC: docker-connect-offsets

CONNECT\_OFFSET\_STORAGE\_REPLICATION\_FACTOR: 1

CONNECT\_STATUS\_STORAGE\_TOPIC: docker-connect-status

CONNECT\_STATUS\_STORAGE\_REPLICATION\_FACTOR: 1

CONNECT\_KEY\_CONVERTER: org.apache.kafka.connect.storage.StringConverter

CONNECT\_VALUE\_CONVERTER: io.confluent.connect.avro.AvroConverter

CONNECT\_VALUE\_CONVERTER\_SCHEMA\_REGISTRY\_URL: http://schema-registry:8081

# CLASSPATH required due to CC-2422

CLASSPATH: /usr/share/java/monitoring-interceptors/monitoring-interceptors-7.3.1.jar

CONNECT\_PRODUCER\_INTERCEPTOR\_CLASSES: "io.confluent.monitoring.clients.interceptor.MonitoringProducerInterceptor"

CONNECT\_CONSUMER\_INTERCEPTOR\_CLASSES: "io.confluent.monitoring.clients.interceptor.MonitoringConsumerInterceptor"

CONNECT\_PLUGIN\_PATH: "/usr/share/java,/usr/share/confluent-hub-components"

CONNECT\_LOG4J\_LOGGERS: org.apache.zookeeper=ERROR,org.I0Itec.zkclient=ERROR,org.reflections=ERROR

ksqldb-server:

image: confluentinc/cp-ksqldb-server:7.3.1

hostname: ksqldb-server

container\_name: ksqldb-server

depends\_on:

- broker

- connect

ports:

- "8088:8088"

environment:

KSQL\_CONFIG\_DIR: "/etc/ksql"

KSQL\_BOOTSTRAP\_SERVERS: "broker:29092"

KSQL\_HOST\_NAME: ksqldb-server

KSQL\_LISTENERS: "http://0.0.0.0:8088"

KSQL\_CACHE\_MAX\_BYTES\_BUFFERING: 0

KSQL\_KSQL\_SCHEMA\_REGISTRY\_URL: "http://schema-registry:8081"

KSQL\_PRODUCER\_INTERCEPTOR\_CLASSES: "io.confluent.monitoring.clients.interceptor.MonitoringProducerInterceptor"

KSQL\_CONSUMER\_INTERCEPTOR\_CLASSES: "io.confluent.monitoring.clients.interceptor.MonitoringConsumerInterceptor"

KSQL\_KSQL\_CONNECT\_URL: "http://connect:8083"

KSQL\_KSQL\_LOGGING\_PROCESSING\_TOPIC\_REPLICATION\_FACTOR: 1

KSQL\_KSQL\_LOGGING\_PROCESSING\_TOPIC\_AUTO\_CREATE: 'true'

KSQL\_KSQL\_LOGGING\_PROCESSING\_STREAM\_AUTO\_CREATE: 'true'

ksqldb-cli:

image: confluentinc/cp-ksqldb-cli:7.3.1

container\_name: ksqldb-cli

depends\_on:

- broker

- connect

- ksqldb-server

entrypoint: /bin/sh

#tty: true