

UD 7 - Apache Kafka Connect

Kafka Connect es un **componente** de Apache Kafka® que se utiliza para realizar la integración de transmisión(streaming integration) entre Kafka y otros sistemas, como bases de datos, servicios en la nube, índices de búsqueda, sistemas de archivos y almacenes de valores-clave.

Kafka Connect facilita la transmisión de datos desde numerosas fuentes a Kafka y la transmisión de datos desde Kafka a numerosos destinos. La siguiente figura enseña una pequeña muestra de estas fuentes (**sources**) y sumideros (**sinks**) objetivo. Hay literalmente cientos de conectores diferentes disponibles para Kafka Connect. Algunos de los más populares incluyen:

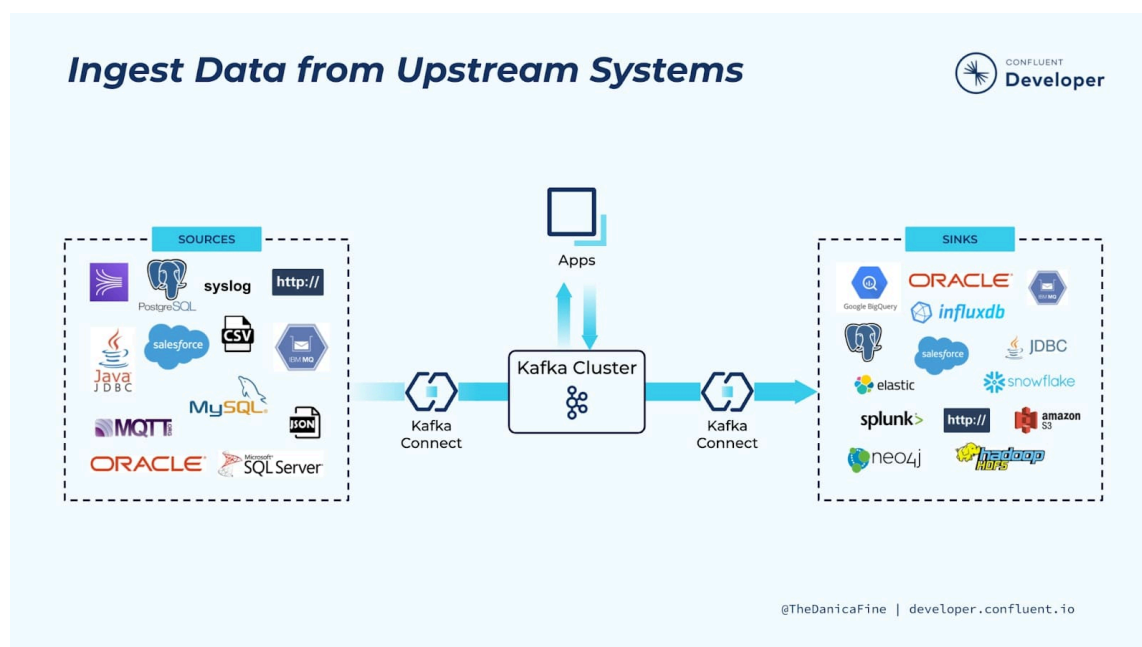


Figura 7.1_Kafka Connect: Integración entre Kafka y otros sistemas (Fuente: developer.confluent.io)

- **RDBMS** (Oracle, SQL Server, Db2, Postgres, MySQL)
- **Cloud object stores** (Amazon S3, Azure Blob Storage, Google Cloud Storage)
- **Message queues** (ActiveMQ, IBM MQ, RabbitMQ)
- **NoSQL and document stores** (Elasticsearch, MongoDB, Cassandra)
- **Cloud data warehouses** (Snowflake, Google BigQuery, Amazon Redshift)

1. Cómo funciona Kafka Connect

Kafka Connect se ejecuta en su propio proceso, **independiente de los brokers y controllers de un cluster Kafka**. Es distribuido, escalable y tolerante a fallos.

Pero la mejor parte de Kafka Connect es que **su uso no requiere programación**. Está completamente **basado en configuración**, lo que lo pone a disposición de una amplia gama de usuarios, no solo de desarrolladores. Además de la ingesta y salida de datos, Kafka Connect **también puede realizar transformaciones ligeras** en los datos a medida que pasan.

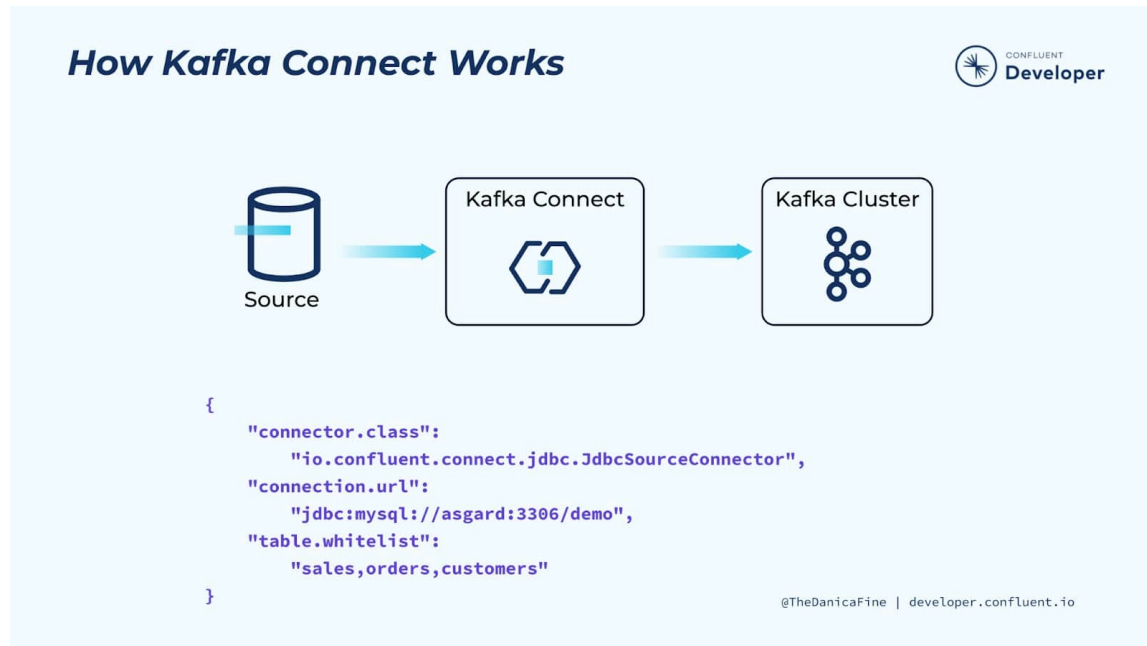


Figura 7.2_Kafka Connect: Cómo funciona Kafka Connect (Fuente: developer.confluent.io)

Siempre que desee transmitir datos a Kafka desde otro sistema, o transmitir datos desde Kafka a otro lugar, Kafka Connect debería ser lo primero que nos venga a la mente. Vamos a estudiar algunos de los casos de uso más común donde se utiliza Kafka Connect.

1.1 Streaming Pipelines

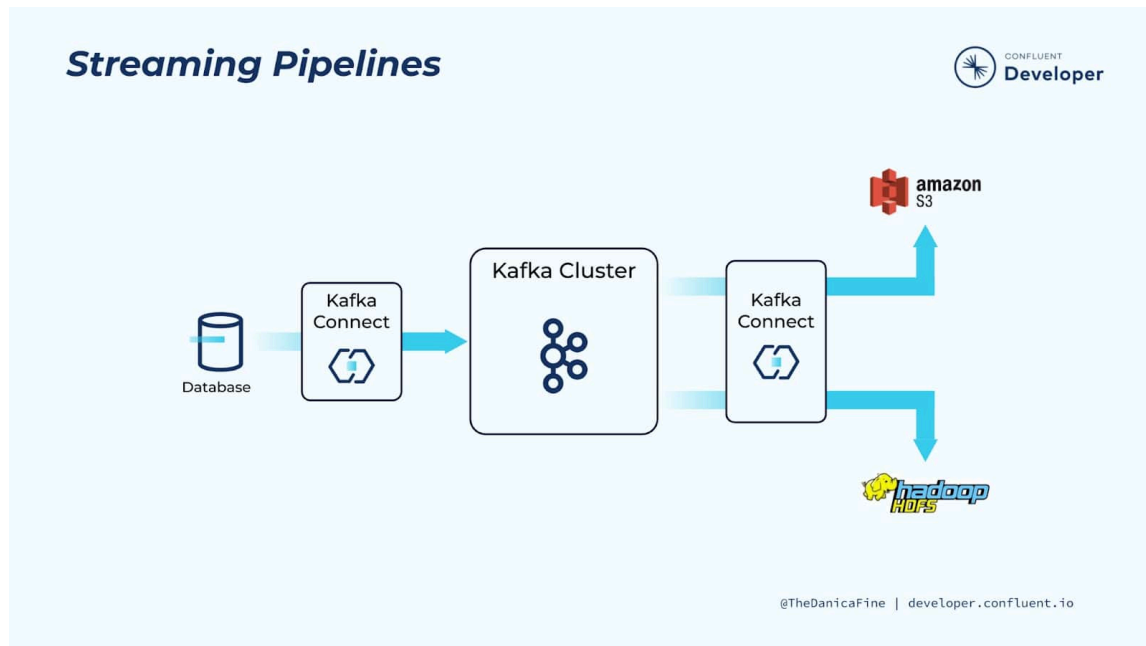


Figura 7.3_Kafka Connect, Streaming Pipelines (Fuente: developer.confluent.io)

Kafka Connect se puede utilizar para la **ingesta de flujos de eventos en tiempo real desde una fuente de datos y transmitirlos a un sistema de destino para su análisis**. En este ejemplo particular, nuestra fuente de datos es una base de datos transaccional.

Tenemos un conector Kafka que sondea la base de datos en busca de actualizaciones y traduce la información en eventos en tiempo real que produce en Kafka.

Tener a Kafka entre los sistemas de origen y de destino significa que estamos construyendo un sistema **débilmente acoplado**. En otras palabras, es relativamente fácil para nosotros cambiar la fuente o el destino sin afectar al otro.

Debido a que Kafka **almacena datos hasta un intervalo de tiempo configurable por entidad de datos (topic)**, es posible **transmitir los mismos datos originales a múltiples objetivos posteriores**. Esto significa que solo necesita mover datos a Kafka una vez y, al mismo tiempo, permitir que sean consumidos por una serie de tecnologías posteriores diferentes para una variedad de requisitos comerciales o incluso para que los mismos datos estén disponibles para diferentes áreas de un negocio.

1.2 Escribir en Datastore desde Kafka

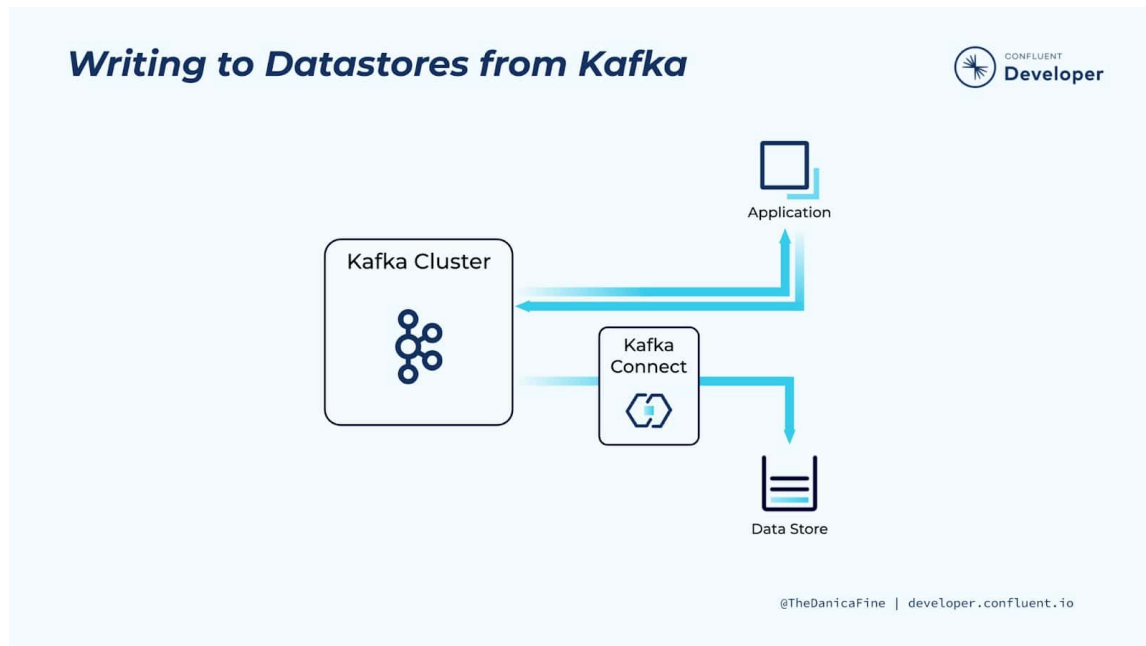


Figura 7.4_Kafka Connect, Escribir en Datastore desde Kafka (Fuente: developer.confluent.io)

Como otro caso de uso, es posible que queramos escribir datos creados por una aplicación en un sistema de destino. Por supuesto, estos podrían ser varios casos de uso de aplicaciones diferentes, pero supongamos que tenemos una aplicación que produce una serie de eventos de registro y nos gustaría que esos eventos también se escriban en un almacén de documentos o persistan en una base de datos relacional.

En lugar de desarrollar la lógica necesaria a esta aplicación determinada y su mantenimiento, con el coste que ello conlleva, podemos producir los datos directamente en Kafka y permitir que Kafka Connect se encargue del resto. Como vimos en el último ejemplo, al mover los datos a Kafka, somos libres de configurar conectores Kafka para mover los datos a cualquier almacén de datos posterior que necesitemos, y está completamente desacoplado de la aplicación misma.

1.3 Evolucionar el procesamiento de sistemas antiguos a nuevos

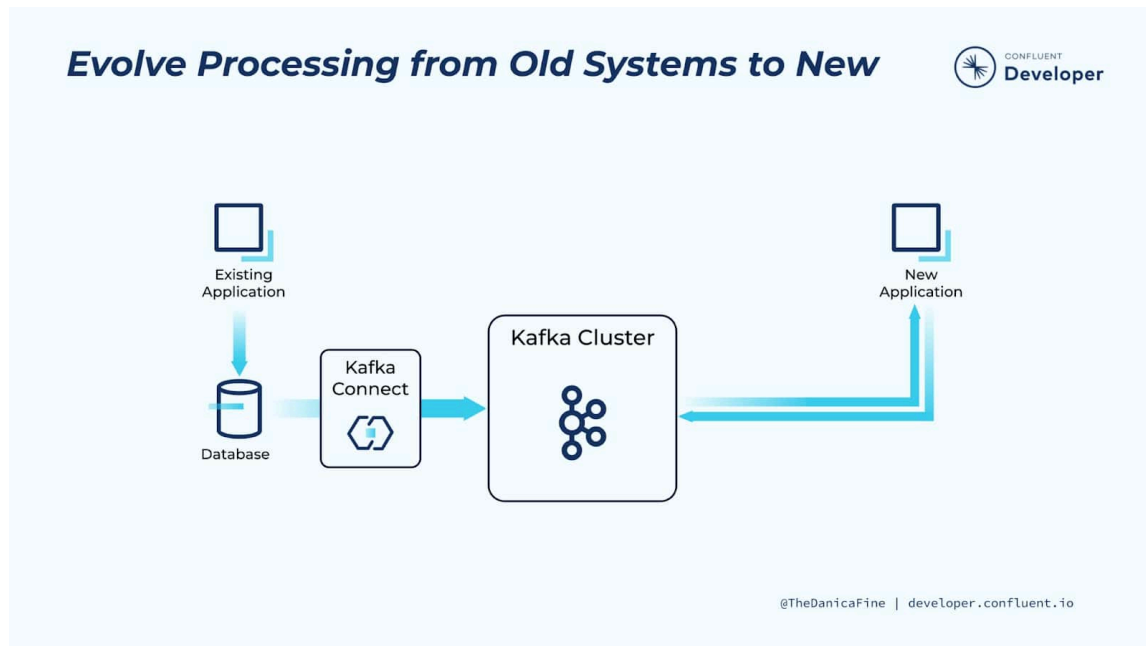


Figura 7.5_Kafka Connect, Evolucionar el procesamiento de sistemas antiguos a nuevos (Fuente: developer.confluent.io)

Antes de la llegada de tecnologías más recientes (como tiendas NoSQL, plataformas de transmisión de eventos y microservicios), las bases de datos relacionales (RDBMS) eran el lugar de facto en el que se escribían todos los datos de las aplicaciones.

Podemos usar **Ingesta de datos desde Bases de Datos hacia Kafka con Kafka Connect usando Change Data Capture (CDC)**, el cual nos permite extraer cada INSERCIÓN, ACTUALIZACIÓN e incluso ELIMINACIÓN de una base de datos en un flujo de eventos en Kafka. Y podemos hacer esto casi en tiempo real

1.4 Sistemas en tiempo real

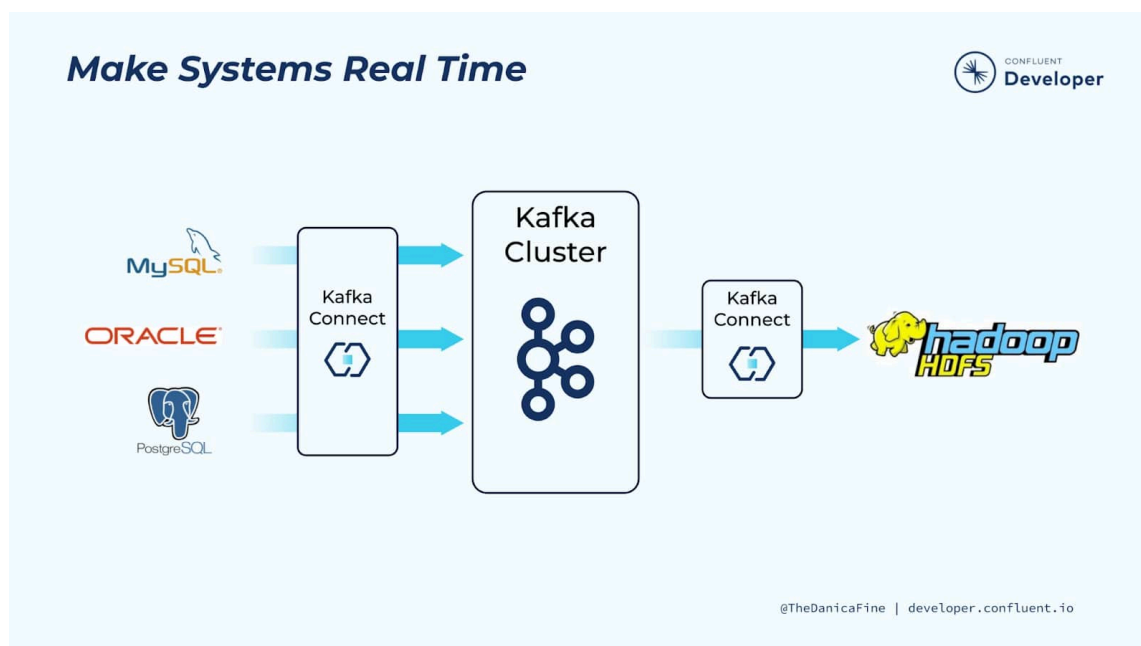


Figura 7.6_Kafka Connect, Sistemas en tiempo real (Fuente: developer.confluent.io)

Hacer que los sistemas estén en tiempo real es increíblemente valioso porque muchas organizaciones tienen datos en reposo en bases de datos y seguirán haciéndolo.

Pero el valor real de los datos radica en nuestra capacidad de acceder a ellos lo más cerca posible del momento en que se generan. Al utilizar Kafka Connect para capturar datos poco después de escribirlos en una base de datos y traducirlos en un flujo de eventos, puede crear mucho más valor. Al hacerlo, se desbloquean los datos para que pueda moverlos a otra parte, por ejemplo, agregándolos a un índice de búsqueda o a un grupo de análisis. Alternativamente, el flujo de eventos se puede utilizar para activar aplicaciones a medida que cambian los datos en la base de datos, por ejemplo, para recalcular el saldo de una cuenta o hacer una recomendación.

2. Conceptos de Kafka Connect

En esta sección describiremos los siguientes conceptos de Kafka Connect:

- **Connectors:** la abstracción de alto nivel que coordina la transmisión de datos mediante la gestión de tareas.
- **Tasks:** La implementación de cómo se copian los datos hacia o desde Kafka.
- **Workers:** los procesos en ejecución que ejecutan conectores y tareas.
- **Converters:** el código utilizado para traducir datos entre Connect y el sistema que envía o recibe datos.
- **Transforms:** Lógica simple para alterar cada mensaje producido o enviado a un conector.

- **Dead Letter Queue** (Cola de mensajes fallidos): cómo Connect maneja los errores del conector

2.1 Connectors

Kafka Connect incluye dos tipos de conectores:

- **Source connector:** Source connectors ingestan bases de datos completas y transmiten actualizaciones de tablas a topics de Kafka. También pueden recopilar métricas de todos sus servidores de aplicaciones y almacenar los datos en topics de Kafka, lo que hace que los datos estén disponibles para el procesamiento de transmisiones con baja latencia.
- **Sink connector:** Sink connectors entregan datos de topics de Kafka a índices secundarios, como Elasticsearch, o sistemas por lotes como Hadoop para análisis offline.

2.2 Tasks

Las tareas son el actor principal en el modelo de datos de Connect. Cada **instancia de conector coordina un conjunto de tareas** que copian datos. Al permitir que el conector divida un único trabajo en muchas tareas, Kafka Connect proporciona soporte integrado para paralelismo y copia de datos escalable con una configuración mínima. Las tareas se pueden iniciar, detener o reiniciar en cualquier momento para proporcionar una canalización de datos resistente y escalable.

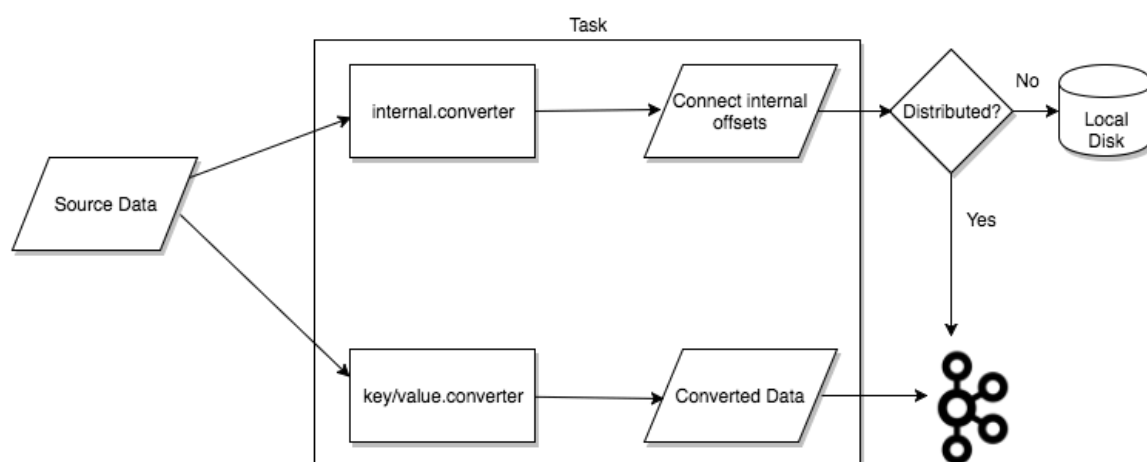


Figura 7.7_Kafka Connect: Task, Representación de alto nivel de datos que pasan a través de una tarea de origen de Connect a Kafka (Fuente: confluent.io)

Task rebalancing

Cuando un conector se envía por primera vez al clúster, los workers reequilibrán el conjunto completo de conectores en el clúster y sus tareas para que cada worker tenga aproximadamente la misma cantidad de trabajo. Este procedimiento de reequilibrio también se utiliza cuando los conectores aumentan o disminuyen la cantidad de tareas que requieren, o cuando se cambia la configuración de un conector. Cuando un worker falla, las tareas se reequilibran entre los workers activos. Cuando una tarea falla, no se activa ningún reequilibrio, ya que la falla de una tarea se considera un caso excepcional.

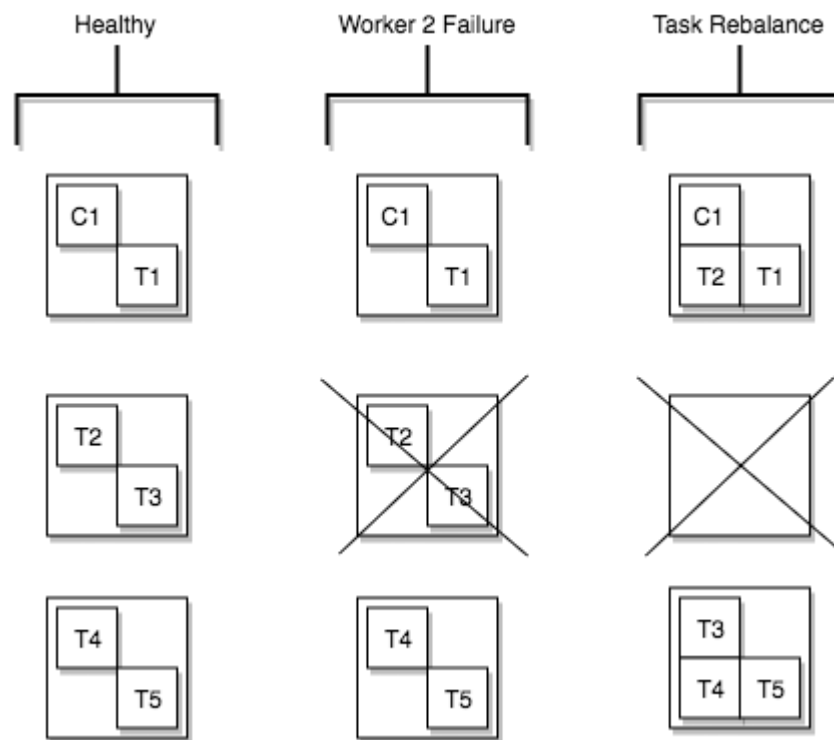


Figura 7.8_Kafka Connect: Task failover (Fuente: confluent.io)

2.3 Workers

Connectors y tasks son unidades lógicas de trabajo y deben programarse para ejecutarse en un proceso. Kafka Connect llama a estos procesos workers y tiene dos tipos de workers: standalone (independientes) y distributed (distribuidos).

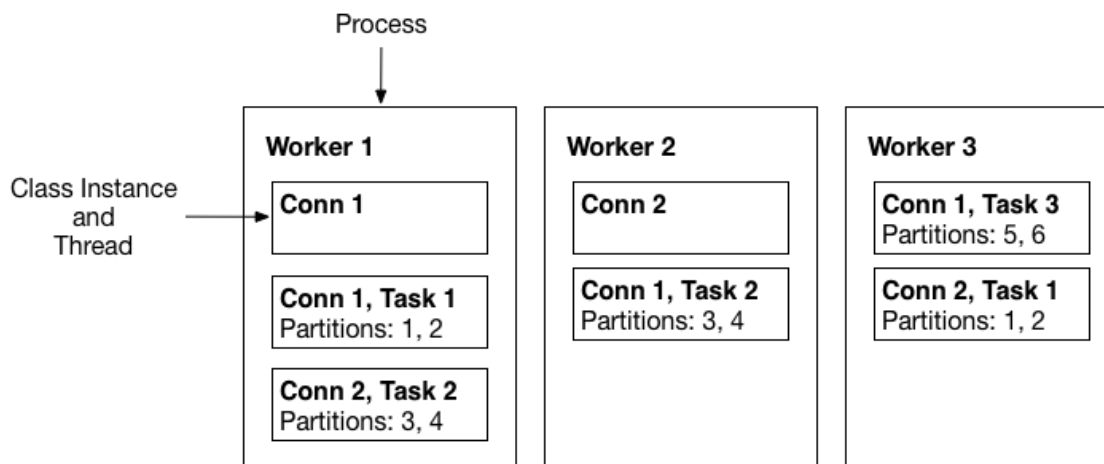


Figura 7.9_Kafka Connect: Modelo básico de worker (Fuente: confluent.io)

- Standalone workers:** Standalone workers es el modo más simple, donde un único proceso es responsable de ejecutar todos los conectores y tareas. Dado que es un proceso único, requiere una configuración mínima. Standalone workers es conveniente para comenzar, durante el desarrollo y en determinadas situaciones en las que solo un proceso tiene sentido, como la recopilación de registros de un host. Sin embargo, debido a que solo hay un proceso, también tiene una funcionalidad más limitada: la escalabilidad se limita al proceso único y no hay tolerancia a fallos más allá de cualquier monitoreo que agregue al proceso único.

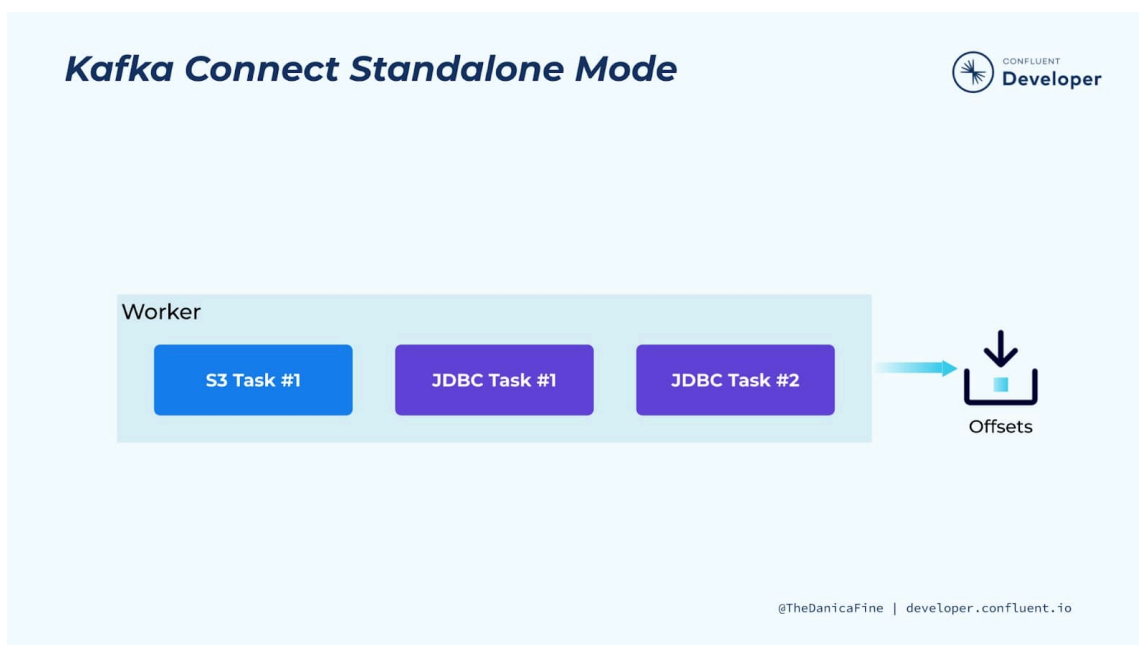


Figura 7.10_Kafka Connect: Standalone mode (Fuente: confluent.io)

- Distributed workers** Distributed workers proporciona escalabilidad y tolerancia automática a fallos para Kafka Connect. En el modo distribuido, inicia muchos procesos de trabajo

utilizando el mismo `group.id` y se coordinan para programar la ejecución de conectores y tareas entre todos los workers disponibles. Si agrega un worker, lo cierra o un worker falla inesperadamente, el resto de los workers lo reconocen y se coordinan para redistribuir conectores y tareas entre el conjunto actualizado de workers disponibles. Tenga en cuenta la similitud con el reequilibrio del grupo de consumidores. Internamente, los workers conectados utilizan grupos de consumidores para coordinarse y reequilibrarse.

Hay que tener en cuenta que todos los workers con el mismo `group.id` estarán en el mismo connect cluster. Por ejemplo, si el worker A tiene `group.id=connect-cluster-a` y el worker B tiene el mismo `group.id`, el worker A y el worker B formarán un clúster llamado `connect-cluster-a`.

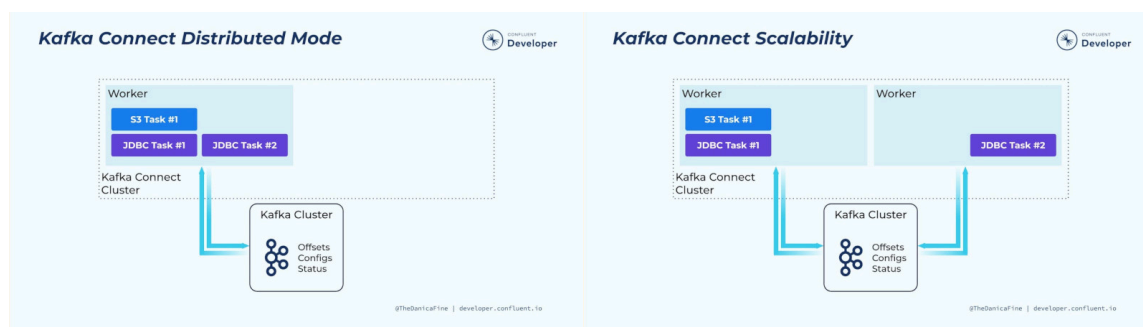


Figura 7.11_Kafka Connect: Distributed mode (Fuente: confluent.io)

2.4 Converters

Converters deben tener una implementación de Kafka Connect que admita un formato de datos particular al escribir o leer desde Kafka. Las tareas utilizan convertidores para cambiar el formato de los datos de bytes a un formato de datos interno de Connect y viceversa.

Converters están desacoplados de los propios conectores para permitir la reutilización de convertidores entre conectores. Por ejemplo (Observa la imagen), utilizando el mismo convertidor Avro, el conector de origen JDBC puede escribir datos de Avro en Kafka y el conector de receptor HDFS puede leer datos de Avro desde Kafka. Esto significa que se puede utilizar el mismo convertidor aunque, por ejemplo, la fuente JDBC devuelva un `ResultSet` que finalmente se escribe en HDFS como un archivo parquet.

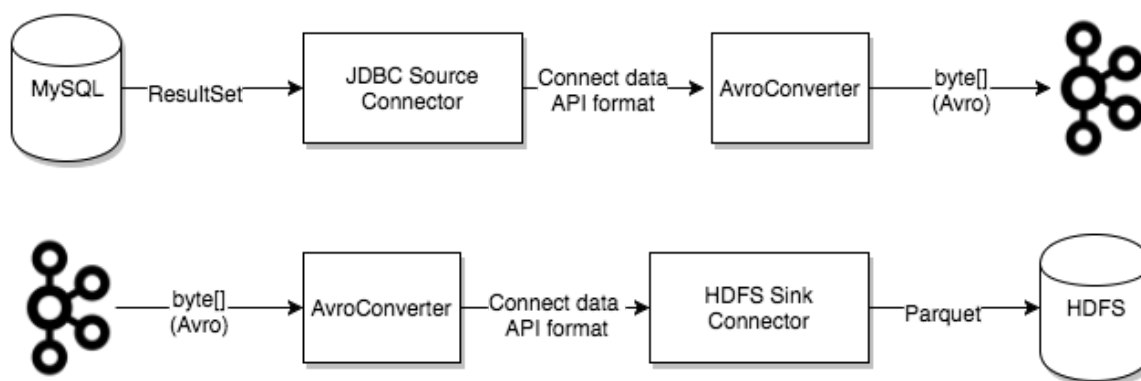


Figura 7.12_Kafka Connect: Converter básico (Fuente: confluent.io)

2.5 Transforms

Connectors se pueden configurar con transformaciones para realizar modificaciones simples y ligeras en mensajes individuales. Esto puede resultar conveniente para **ajustes menores de datos y enrutamiento de eventos**, y muchas transformaciones se pueden encadenar en la configuración del conector. Sin embargo, para transformaciones y operaciones más complejas que se aplican a muchos mensajes se implementan mejor con **ksqlDB** y **Kafka Streams**.

Una transformación es una función simple que acepta un registro como entrada y genera un registro modificado. Todas las transformaciones proporcionadas por Kafka Connect realizan modificaciones simples pero comúnmente útiles.

Transforms también se pueden utilizar con sink connectors. Para más información, consulta la página oficial [Transformation](#)

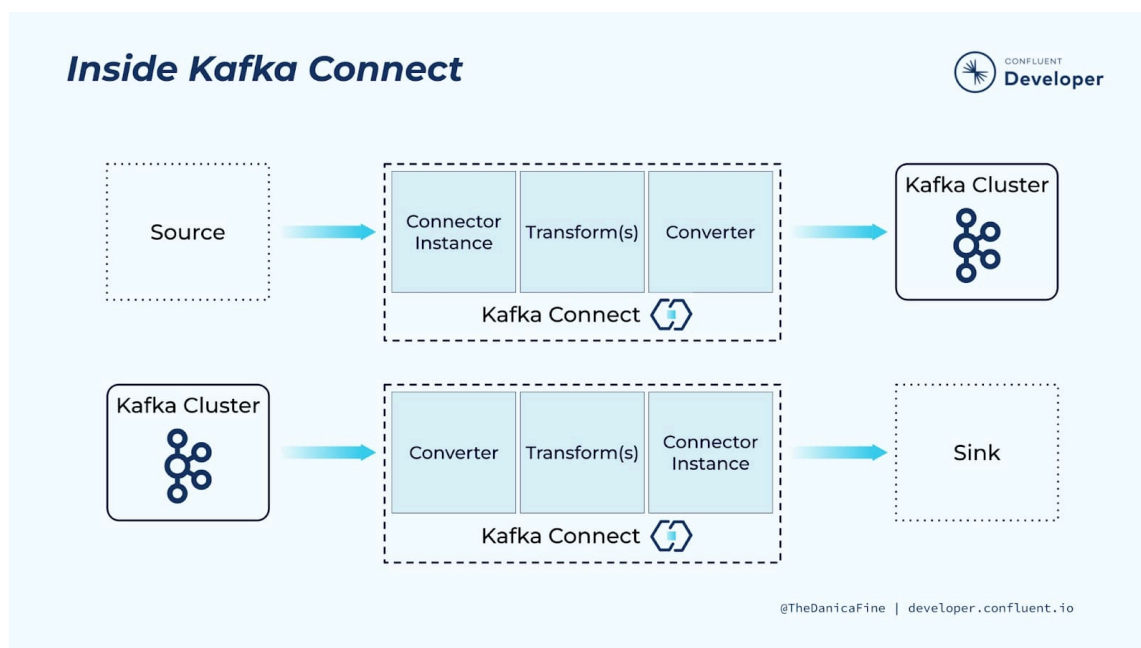
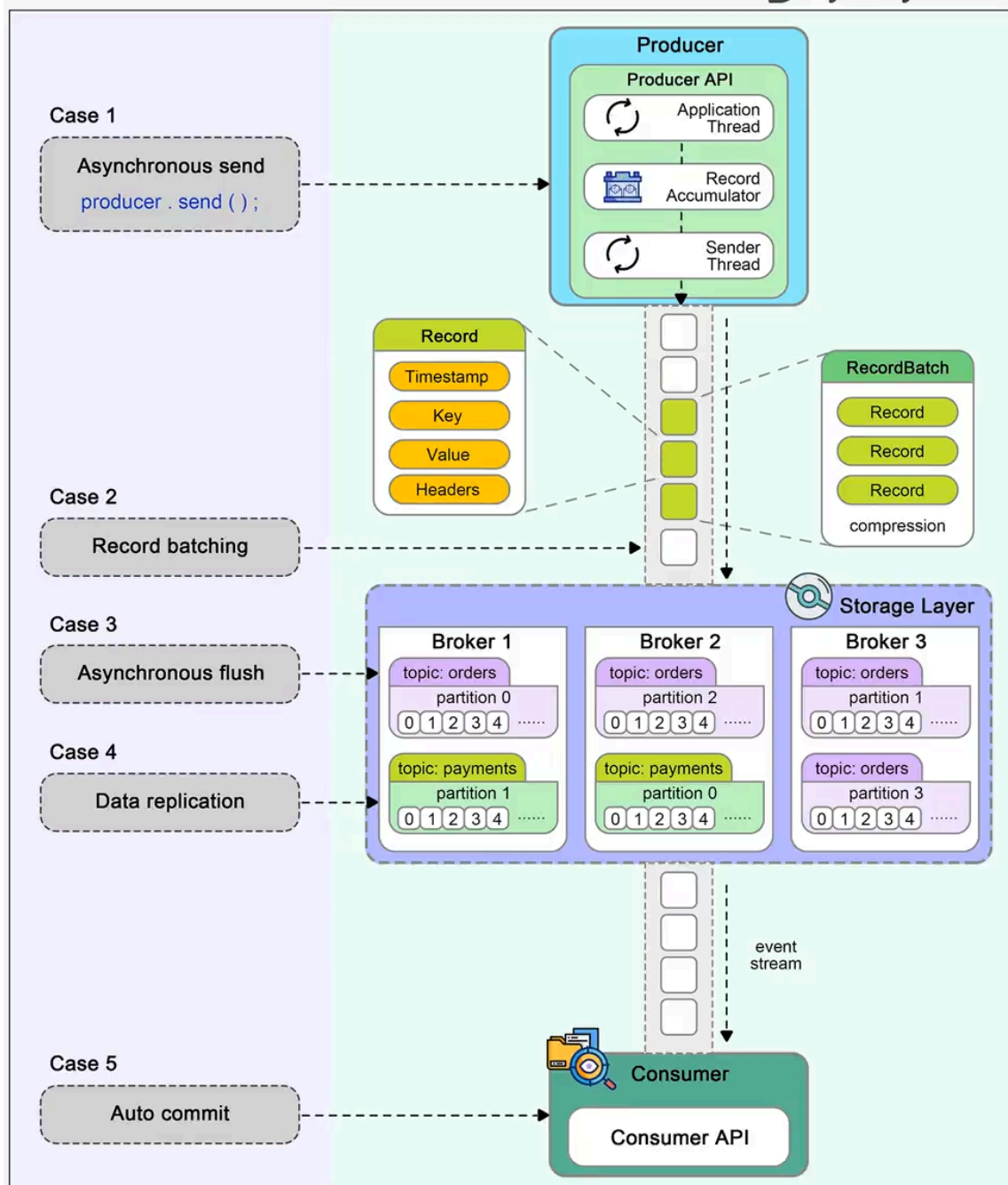


Figura 7.13_Kafka Connect: Dentro de un conector (converter y transform)
(Fuente: *confluent.io*)

2.6 Error Reporting in Connect

Kafka Connect proporciona informes de errores para manejar los errores encontrados en varias etapas del procesamiento. De forma predeterminada, **cualquier error encontrado durante la conversión o dentro de las transformaciones hará que el conector falle**. Cada configuración de conector también puede permitir tolerar dichos errores emitiéndolos, escribiendo opcionalmente cada error y los detalles de la operación fallida y el registro problemático (con varios niveles de detalle) en el registro de la aplicación Connect. Estos mecanismos también capturan errores cuando un conector receptor procesa los mensajes consumidos de sus topics Kafka, y **todos los errores se pueden escribir en un topic Kafka "dead letter queue" (DLQ) "cola de mensajes no entregados" configurable**.

Can Kafka Lose Messages ?



Animación 7.1_Kafka Connect: Kafka Lose Message? (Fuente: bytebytego.com)

3. Running Kafka Connect

Kafka Connect admite dos modos de ejecución: standalone (single process) and distributed.

- **Standalone mode:** En este modo, todo el trabajo se realiza en un único proceso. Esta configuración es más sencilla de configurar y poner en marcha, y puede resultar útil en situaciones en las que sólo tiene sentido un worker (por ejemplo, para recopilar archivos de log), pero no se beneficia de algunas de las características de Kafka Connect, como la tolerancia a fallos. Útil para desarrollar y probar Kafka Connect en una máquina local.
- **Distributed mode:** Gestiona el equilibrio automático del trabajo, permite escalar hacia arriba (o hacia abajo) de forma dinámica y ofrece tolerancia a fallos tanto en las tareas activas como en los datos de configuración y de confirmación de offset

Se recomienda para entornos de producción debido a las ventajas de escalabilidad, alta disponibilidad y gestión. Ejecuta los workers de Connect en varias máquinas (nodos), que forman un clúster de Connect. Kafka Connect distribuye los conectores en ejecución por todo su clúster. Puede añadir o eliminar nodos a medida que evolucionen sus necesidades.

3.1 Standalone mode

Podemos iniciar Kafka Connect en Standalone mode con el siguiente comando:

```
1 bin/connect-standalone.sh worker.properties connector1.properties  
[connector2.properties connector3.properties ...]
```

El **primer parámetro** `worker.properties` es la configuración para el worker. Puedes usar la configuración por defecto proporcionada por Kafka en `config/connect-standalone.properties`. Esta incluye ajustes como los **parámetros de conexión de Kafka**, el **formato de serialización** y la **frecuencia con la que se comprometen los offset**. La configuración de `config/connect-standalone.properties` debería funcionar bien con un cluster local ejecutándose con la configuración por defecto proporcionada por `config/server.properties` (Recuerda el rol: `process.roles=broker,controller`)

Será necesario modificarlo para utilizarlo con una configuración diferente o un despliegue en producción. Todos los workers (tanto autónomos -standalone- como distribuidos -distributed-) requieren algunas configuraciones:

- **bootstrap.servers** : Lista de servidores Kafka utilizados para arrancar conexiones a Kafka.
- **key.converter** : Clase Converter utilizada para convertir entre el formato Kafka Connect y la forma serializada que se escribe en Kafka. Ejemplos de formatos comunes incluyen JSON y Avro.
- **value.converter** : Clase Converter utilizada para convertir entre el formato de Kafka Connect y el formato serializado que se escribe en Kafka. Ejemplos de formatos comunes incluyen JSON y Avro.

- `plugin.path` (vacío por defecto) - una lista de rutas que contienen plugins de Connect (conectores, convertidores, transformaciones).

Las opciones de configuración importantes específicas del standalone mode son:

- `offset.storage.file.filename` : Archivo para almacenar los offsets de los conectores de origen.

El **segundo parámetro (y sucesivos, si los hubiera)** `connector1.properties` es el archivo de propiedades de configuración del conector. Todos los conectores tienen propiedades de configuración que se cargan con el worker.

3.2 Distributed mode

Podemos iniciar Kafka Connect en Distributed mode con el siguiente comando:

```
1 bin/connect-distributed.sh config/connect-distributed.properties
2 bin/connect-distributed worker.properties
```

La diferencia está en la clase que se inicia y en los parámetros de configuración que cambian el modo en que el proceso de Kafka Connect decide dónde almacenar las configuraciones, cómo asignar el trabajo y dónde almacenar los offsets y los estados de las tareas.

El **parámetro** `worker.properties` es la configuración para el worker. Puedes usar la configuración por defecto proporcionada por Kafka en `config/connect-distributed.properties`.

En el modo distribuido, Kafka Connect almacena los offsets, las configuraciones y los estados de las tareas en topics de Kafka. Se recomienda crear manualmente los topics para las offsets, las configuraciones y los estados, con el fin de alcanzar el número deseado de particiones y factores de replica. Si los topics aún no se han creado al iniciar Kafka Connect, se crearán automáticamente con el número de particiones y el factor de replicación predeterminados, que pueden no ser los más adecuados para su uso.

En particular, los siguientes parámetros de configuración, además de los ajustes comunes mencionados anteriormente, son fundamentales para establecer antes de iniciar el clúster:

- `group.id` (por defecto `connect-cluster`): Nombre único para el clúster, utilizado para formar el grupo de clústeres de Connect. Debemos tener en cuenta que no debe entrar en conflicto con los ID de los grupos de consumidores.
- `config.storage.topic` (por defecto `connect-configs`): Topic que se utilizará para almacenar las configuraciones de conectores y tareas. Debemos tener en cuenta que debe ser un topic de partición única, altamente replicado y compactado. Es posible que tengamos que crear manualmente el topic para garantizar la configuración correcta, ya

que los topics creados automáticamente pueden tener varias particiones o estar configurados automáticamente para la eliminación en lugar de la compactación.

- `offset.storage.topic` (por defecto `connect-offsets`): Topic a utilizar para almacenar offsets. Este topic debe tener muchas particiones, estar replicado y configurado para la compactación.
- `status.storage.topic` (por defecto `connect-status`): Topic a utilizar para almacenar estados. Este topic puede tener múltiples particiones, y debe estar replicado y configurado para la compactación.



Connectors on distributed mode

Hay que tener en cuenta que en el modo distribuido las configuraciones de los conectores no se pasan por la línea de comandos. En su lugar, se utiliza la *API REST* (que se describe a continuación) para crear, modificar y destruir conectores.

3.3 Configuring Connectors

Las configuraciones de conectores son simples asignaciones clave-valor. Tanto en modo standalone como distributed, se incluyen en la carga JSON de la solicitud REST que crea (o modifica) el conector. En modo standalone (como hemos visto anteriormente), también pueden definirse en un archivo de propiedades y pasarse al proceso Connect en la línea de comandos.

La mayoría de las configuraciones dependen del conector, por lo que no se pueden describir aquí. Sin embargo, hay algunas opciones comunes:

- `name` - Nombre único para el conector. Si se intenta registrar de nuevo con el mismo nombre, se producirá un error.
- `connector.class` - La clase Java para el conector
- `tasks.max` - El número máximo de tareas que deben crearse para este conector. El conector puede crear menos tareas si no puede alcanzar este nivel de paralelismo.
- `key.converter` - (opcional) Anula el convertidor de clave predeterminado establecido por el worker.
- `value.converter` - (opcional) Sustituye al convertidor de valor predeterminado establecido por el worker.

Los conectores de sink también tienen algunas opciones adicionales para controlar su entrada. Cada conector de sink debe establecer una de las siguientes opciones:

- `topics` - Una lista separada por comas de topics a utilizar como entrada para este conector

- **topics.regex** - Una expresión regular de Java de los topics a utilizar como entrada para este conector

4. Ejemplo 3. QuickStart Kafka Connect

Para una primera toma de contacto práctico con Kafka Connect vamos a usar 2 conectores sencillos: uno que importa datos de un archivo (source) a un topic y otro que exporte datos (sink) de un topic a un archivo. Todo ello en standalone mode.

Por supuesto, no debemos olvidar que Kafka Connect debe estar conectado con un cluster de Kafka. Para este ejemplo, usaremos la misma configuración que el ejemplo 1 de quickstart de Kafka, con un único nodo que hace de controller y broker al mismo tiempo. Por tanto usaremos la configuración por defecto.

1. Iniciamos el cluster Kafka

```
1 KAFKA_CLUSTER_ID="$(bin/kafka-storage.sh random-uuid)"
2 bin/kafka-storage.sh format -t $KAFKA_CLUSTER_ID -c
  config/kraft/server.properties
3 bin/kafka-server-start.sh config/kraft/server.properties
```

2. Para ello usaremos un plugin de conexión de ficheros llamado `connect-file-3.9.0.jar`. Recuerda que en los ejemplos siempre estamos trabajando desde el directorio de instalación de kafka, en nuestro caso `/opt/kafka_2.13-3.9.0`. También creamos los directorios necesarios para nuestro ejemplo:

```
1 mkdir -p /opt/kafka/ejemplo3/config
2 mkdir -p /opt/kafka/ejemplo3/libs
```

3. Por tanto, en primer lugar, nos tenemos que asegurar de añadir `connect-file-3.9.0.jar` a la propiedad `plugin.path` en la configuración del Connect worker.



Rutas relativas y absolutas

Para los despliegues de producción es siempre preferible utilizar rutas absolutas. Consulta la documentación oficial de `plugin.path` para más detalle de cómo establecer esta configuración.

4. Indicamos la ruta relativa del conector, que este caso es `libs/connect-file-3.9.0.jar`
5. Hacemos 1 copia del fichero de configuración para el worker

```
1 cp config/connect-standalone.properties
  /opt/kafka/ejemplo3/config/worker1.properties
```

6. Hacemos 1 copia del plugin a usar

```
1 cp libs/connect-file-3.9.0.jar /opt/kafka/ejemplo3/libs/connect-file-3.9.0.jar
```

7. Editamos la configuración del worker para que añada el correspondiente plugin para Kafka Connect

worker1.properties

```
1 plugin.path=libs/connect-file-3.9.0.jar
```

8. Hacemos una copia de la configuración de los conectores: file-source y file-sink

```
1 cp config/connect-file-source.properties /opt/kafka/ejemplo3/config/connect-file-source.properties
2 cp config/connect-file-sink.properties /opt/kafka/ejemplo3/config/connect-file-sink.properties
```

9. Creamos algunos datos de test en el directorio de nuestro ejemplo `opt/kafka/ejemplo3`

```
1 echo -e "Hola\nmundo.\nProbando Kafka\nen el modulo Big Data Aplicado\nen el IES Gran Capitán" > test.txt
```

10. Iniciamos proporcionando tres archivos de configuración como parámetros.

- El primero es siempre la configuración para el proceso Kafka Connect, que contiene la configuración común, como los workers de Kafka para conectarse y el formato de serialización de datos.
- Los archivos de configuración restantes especifican cada uno de los conectores. Estos archivos incluyen un nombre de conector único, la clase de conector a instanciar (`connector.class`) y cualquier otra configuración requerida por el conector.
- Observa el contenido de los archivos de configuración para conocer más y familiarizarte con ellos.
- Como el source file lo tenemos en `/opt/kafka/ejemplo3/` , iniciamos esta configuración de Kafka Connect en dicho directorio

```
1 /opt/kafka_2.13-3.9.0/bin/connect-standalone.sh
/opt/kafka/ejemplo3/config/worker1.properties
/opt/kafka/ejemplo3/config/connect-file-source.properties
/opt/kafka/ejemplo3/config/connect-file-sink.properties
```

11. Ahora tenemos dos conectores: el primero es un **conector de origen que lee líneas de un archivo de entrada y produce cada una de ellas en un topic de Kafka** y el segundo es un

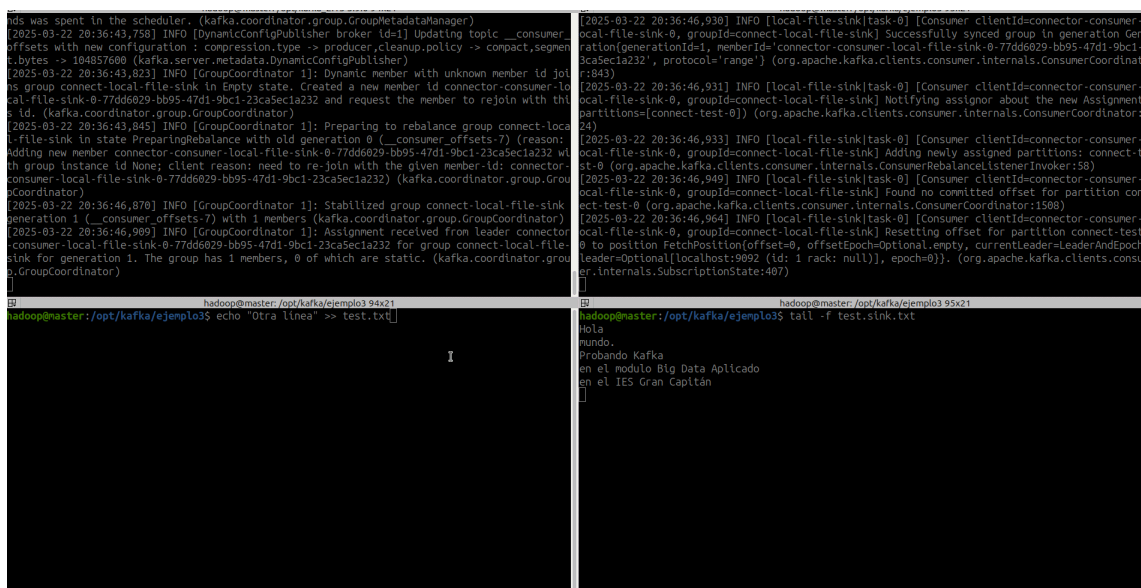
conector de destino que lee mensajes de un topic de Kafka y *produce* cada uno de ellos como una línea en un archivo de salida.

12. Desde el inicio vemos una serie de mensajes de registro, incluyendo algunos que indican que los conectores se están instanciando. Una vez que el proceso de Kafka Connect se ha iniciado, el conector de origen debería empezar a leer líneas de `test.txt` y producirlas en el topic `connect-test`, y el conector sink debería empezar a leer mensajes del topic `connect-test` y escribirlos en el archivo `test.sink.txt`. Podemos verificar que los datos han sido entregados a través de todo el pipeline examinando el contenido del archivo de salida.

```
1 tail -f test.sink.txt
2 Hola
3 mundo.
4 Probando Kafka
5 en el modulo Big Data Aplicado
6 en el IES Gran Capitán
```

13. Los conectores siguen procesando datos, por lo que podemos añadir datos al archivo y ver cómo se mueven por el pipeline:

```
1 echo "Otra línea" >> test.txt
```



Animación 7.2_Kafka Connect: Ejemplo 3

14. Debemos tener en cuenta que los datos se almacenan en el topic de Kafka `connect-test`, por lo que también podemos ejecutar un consumidor de consola para ver los datos en el topic (o utilizar código de consumidor personalizado para procesarlos):

```
1 bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic
connect-test --from-beginning
```

```

2  {"schema":{"type":"string","optional":false},"payload":"Hola"}
3  {"schema":{"type":"string","optional":false},"payload":"mundo."}
4  {"schema":{"type":"string","optional":false},"payload":"Probando Kafka"}
5  {"schema":{"type":"string","optional":false},"payload":"en el modulo Big
Data Aplicado"}
6  {"schema":{"type":"string","optional":false},"payload":"en el IES Gran
Capitán"}
7  {"schema":{"type":"string","optional":false},"payload":"Otra linea"}
8  {"schema":{"type":"string","optional":false},"payload":"Otra linea"}
9  {"schema":{"type":"string","optional":false},"payload":"Otra linea"}
10 {"schema":{"type":"string","optional":false},"payload":"Otra linea"}
11 {"schema":{"type":"string","optional":false},"payload":"Otra linea"}
12 {"schema":{"type":"string","optional":false},"payload":"Otra linea"}
13 {"schema":{"type":"string","optional":false},"payload":"Más líneas"}

```

5. Plugins y como conectarlos

Kafka Connect está diseñado para ser extensible, de modo que los **desarrolladores puedan crear conectores, transformaciones y convertidores personalizados**, y los usuarios puedan instalarlos y ejecutarlos

Un plugin de Kafka Connect es un complemento de un conjunto de archivos JAR que contienen la implementación de uno o varios conectores, transformaciones o convertidores.

Connect aísla cada plugin entre sí, de modo que las bibliotecas de un plugin no se ven afectadas por las bibliotecas de ningún otro plugin. Esto es muy importante cuando se mezclan y combinan conectores de múltiples proveedores.



Warning

Es habitual tener muchos plugins instalados en un entorno en producción de Kafka Connect. Debemos asegurarnos que sólo tenemos instalada una versión de cada plugin.

Un plugin de Kafka Connect puede ser cualquiera de los siguientes:

- Un directorio en el sistema de archivos que contiene todos los archivos JAR necesarios y las dependencias de terceros para el plugin. Esto es lo más habitual y lo preferible.
- Un único JAR que contenga todos los archivos de clase para un plugin y sus dependencias de terceros.
- Directorios que contengan inmediatamente la estructura de directorios de paquetes de clases de plugins y sus dependencias. *Nota: se seguirán enlaces simbólicos para descubrir dependencias o plugins.*

Ejemplos:

```
plugin.path=/usr/local/share/java,/usr/local/share/kafka/plugins,/opt/connectors
```

Un plugin de Kafka Connect nunca debe contener bibliotecas proporcionadas por el tiempo de ejecución de Kafka Connect.

5.1 Configuración de plugins

Kafka Connect encuentra los plugins utilizando una ruta definida como una lista separada por comas de rutas de directorio en la propiedad de configuración de worker `plugin.path`. Por ejemplo:

```
1 plugin.path=/usr/local/share/kafka/plugins
```

Para instalar un plugin, como ya hemos visto, debe colocar el directorio del plugin o el conjunto de archivos JAR (o un enlace simbólico que resuelva a uno de ellos) en un directorio que ya figure en la ruta del plugin (`plugin.path`). O bien, puede actualizar la ruta del plugin añadiendo la ruta absoluta del directorio que contiene el plugin.

Cuando se inician los workers de Connect, cada uno de ellos descubre todos los conectores, transformadores y convertidores que se encuentran en los directorios de la ruta de plugins (`plugin.path`). Cuando se utiliza un conector, una transformación o un convertidor, el worker de Connect carga primero las clases del plugin correspondiente, seguidas del tiempo de ejecución de Kafka Connect y las bibliotecas Java. Connect evita explícitamente todas las bibliotecas de otros plugins. Esto evita conflictos y hace que sea muy fácil añadir y utilizar conectores y transformaciones desarrollados por distintos proveedores.

5.2 Fuentes de plugins

Podemos encontrar plugins para Kafka Connect en varias fuentes confiables, cada una ofreciendo una variedad de conectores diseñados para facilitar la integración de sistemas de almacenamiento de datos, aplicaciones, y APIs con nuestro clúster de Kafka. A continuación listamos las principales:

1. **Confluent Hub:** [Confluent Hub](#) es probablemente la fuente más popular y ampliamente utilizada para encontrar conectores de Kafka Connect. Es una biblioteca digital mantenida por Confluent (la compañía liderada por los co-creadores de Kafka) que ofrece una gran variedad de conectores desarrollados por Confluent y por la comunidad. Aquí encontrarás conectores para bases de datos, sistemas de archivos, servicios en la nube, y muchas otras fuentes de datos y sumideros.
2. **Apache Kafka:** El propio proyecto Apache Kafka incluye **algunos conectores básicos** como parte de su distribución (carpeta `config`). Estos incluyen conectores para archivos de sistemas (source y sink) y para replicar topics entre clústeres de Kafka (MirrorMaker).
 - `connect-console-sink.properties`
 - `connect-file-source.properties`

- connect-console-source.properties
 - connect-log4j.properties
 - connect-distributed.properties
 - connect-mirror-maker.properties
 - connect-file-sink.properties
 - connect-standalone.properties
3. **GitHub:** Muchos desarrolladores y empresas publican sus conectores personalizados en GitHub. Puedes buscar repositorios que contengan conectores de Kafka Connect usando términos de búsqueda como "Kafka Connect Connector". Esto puede ser especialmente útil si estás buscando conectores para sistemas menos comunes o para nuevas tecnologías que aún no han sido adoptadas por grandes plataformas como Confluent Hub.
4. **Proveedores de Software y Servicios:** Algunos proveedores de bases de datos y plataformas de software ofrecen sus propios conectores de Kafka Connect diseñados para trabajar de manera óptima con sus sistemas. Por ejemplo, compañías como [MongoDB](#), [Snowflake](#), [Databricks](#) y [elastic](#) ofrecen conectores específicos para sus plataformas que están optimizados para el rendimiento y la funcionalidad.
5. **Consideraciones al elegir un conector/plugin.** Al seleccionar un conector de Kafka Connect, considera lo siguiente:
- **Compatibilidad y Requisitos:** Asegúrate de que el conector/plugin sea compatible con nuestra versión de Kafka y cumpla con tus requisitos específicos de rendimiento y seguridad.
 - **Mantenimiento y Soporte:** Elegir conectores/plugins que se mantengan activamente y tengan buen soporte.
 - **Licencia y Coste:** Verifica la licencia bajo la cual se distribuye el conector y si hay costes asociados con su uso en producción.

Utilizar la fuente correcta y seleccionar el conector adecuado puede marcar una gran diferencia en la facilidad de integración y la estabilidad de tus flujos de datos en Kafka Connect.

5.3 Instalación y uso

Para aprender a instalar un plugin en Kafka Connect, vamos a usar como ejemplo la instalación del **Conector JDBC de Confluent para Kafka Connect**, que es ampliamente utilizado para integrar bases de datos relacionales con Kafka. Este conector permite a Kafka Connect leer desde y escribir hacia bases de datos usando JDBC.

1. Preparación del sistema:

Antes de instalar el conector, asegúrate de que **Kafka** y **Kafka Connect** estén instalados y configurados correctamente en tu sistema. Debes tener acceso a las configuraciones por defecto de de Kafka Connect, normalmente encontradas en `config/connect-distributed.properties` o `config/connect-standalone.properties`, dependiendo de si estás ejecutando Kafka Connect en modo distribuido o standalone.

2. Herramienta Confluent CLI (Recomendada si usas cloud de confluent)

Podemos descargarlos de Confluent desde consola usando su herramienta `confluent-hub`. Deberás instalarla en cada máquina. Siguiendo la documentación oficial:

```
1 curl -sL --http1.1 https://cnfl.io/cli | sh -s -- latest
```

Para más detalle y uso de otros sistemas sigue los pasos que te indican en su [documentación oficial](#)

3. Descarga del conector:

Puedes descargarlo de **2 formas diferentes**:

- Usando `confluent-hub`. Este comando descarga e instala la última versión del conector en el directorio de plugins de Kafka Connect. Durante la instalación, se te preguntará si deseas instalar las dependencias del conector y si quieres actualizar el archivo de configuración de Connect automáticamente.

```
1 ./bin/confluent connect plugin install confluentinc/kafka-connect-jdbc:latest --plugin-directory /opt/kafka/ejemplo4/libs --worker-configurations /opt/kafka/ejemplo4/config/workers.properties
```

donde `--plugin-directory` es el directorio donde vas a guardar el conector y `--worker-configurations` es el fichero de configuración worker y añade automáticamente la configuración de ruta donde se se ha guardado el conector `/opt/kafka/ejemplo4/libs` al final de la lista `plugin.path`

- Descargando el fichero correspondiente y alojarlo en el directorio de configuración `plugin.path` explicado anteriormente. Accede a la página del conector y [descargarlo](#)

4. Configuración del conector:

Una vez instalado el conector, debes crear un archivo de configuración para el mismo. Crea un archivo llamado `jdbc-source.properties` con el siguiente contenido:

`jdbc-source.properties`

```
1 name=jdbc-source-connector
2 connector.class=io.confluent.connect.jdbc.JdbcSourceConnector
3 tasks.max=1
4 connection.url=jdbc:mysql://localhost:3306/mydatabase
```



```
5 connection.user=myuser
6 connection.password=mypassword
7 table.whitelist=mytable
8 mode=timestamp
9 timestamp.column.name=timestamp_column
10 topic.prefix=jdbc-
```

Este archivo configura un conector de fuente JDBC que extrae datos de una tabla específica en una base de datos MySQL. Tendríamos que asegurarnos de ajustar los parámetros de conexión y otros detalles según nuestro entorno específico.

5. Inicio de Kafka Connect con el conector:

- En standalone mode:

```
1 connect-standalone.sh config/connect-standalone.properties config/jdbc-source.properties
```

- En distributed mode:

```
1 connect-distributed.sh config/connect-distributed.properties
```

API REST

En distributed mode, la interacción de los conectores se **realiza obligatoriamente** a través de la **API REST de Kafka Connect**, que explicaremos en el [siguiente punto](#). Por este motivo, al iniciarlo en este modo no se le pasa ningún parámetro de configuración de conectores. En standalone mode, como no es en un entorno en producción, podemos usar ambos, tanto pasar por parámetro la configuración del conector como la API REST.

Registramos el conector utilizando la API REST de Kafka Connect:

```
1 curl -X POST -H "Content-Type: application/json" --data @jdbc-source.properties http://localhost:8083/connectors
```

6. Verificación

Verificamos que el conector esté funcionando correctamente revisando los logs de Kafka Connect y/o utilizando la API REST para consultar el estado del conector:

```
1 curl http://localhost:8083/connectors/jdbc-source-connector/status
```

Este paso te asegura que el conector esté extrayendo datos de la base de datos y los esté publicando en los topics de Kafka especificados.

6. API REST

Kafka Connect está pensado para ser ejecutado como un servicio, y también proporciona una API REST para gestionar conectores. Esta interfaz nos permite interactuar con un servidor Kafka Connect para administrar conectores y sus configuraciones. Proporciona una forma estándar y eficiente para configurar, gestionar y monitorear conectores en un clúster de Kafka Connect, ya sea en modo distribuido o standalone.

Kafka Connect está diseñado para ser operado de manera remota a través de esta API REST, lo que significa que todas las tareas administrativas pueden realizarse sin interactuar directamente con los servidores en sí mismos. Esto incluye:

- **Crear, modificar y eliminar conectores:** La API permite cargar nuevas configuraciones de conectores, modificar configuraciones existentes y eliminar conectores no deseados.
- **Consultar el estado y la configuración de los conectores:** Puedes obtener información sobre el estado actual de cualquier conector en el sistema, incluyendo si están corriendo, fallando, o detenidos, y ver detalles específicos de configuración y operación.
- **Reiniciar y pausar conectores:** La API también ofrece controles para pausar y reiniciar conectores, lo cual es útil para mantenimiento o para recuperarse de errores.

La API REST de Kafka Connect es accesible mediante solicitudes HTTP estándar. Por defecto, el servidor REST se ejecuta en el **puerto 8083**



API REST on HTTPS

También puede realizarse a través de HTTPS, pero entonces la configuración debe incluir la configuración SSL. Por defecto, se utilizará la configuración `ssl.*`. Para más detalle consulta la [documentación oficial de REST API](#).

6.1 Endpoints

A continuación indicamos una lista de los endpoints actualmente admitidos en la API REST:

- `GET /connectors` : Devuelve una lista de conectores activos.
- `POST /connectors` : Crea un nuevo conector. El cuerpo de la solicitud debe ser un objeto JSON que contenga un campo string `name` y un campo objeto `config` con los parámetros de configuración del conector. El objeto JSON también puede contener opcionalmente un campo string `initial_state` que puede tomar los siguientes valores: `STOPPED`, `PAUSED` o `RUNNING` (el valor por defecto).
- `GET /connectors/{name}` : Obtiene información sobre un conector específico.
- `GET /connectors/{name}/config` : Obtiene los parámetros de configuración de un conector específico

- `PUT /connectors/{name}/config` : Actualiza los parámetros de configuración de un conector específico
- `GET /connectors/{name}/status` : Obtiene el estado actual del conector, incluyendo si se está ejecutando, ha fallado, está en pausa, etc., a qué worker está asignado, información de error si ha fallado, y el estado de todas sus tareas.
- `GET /connectors/{name}/tasks` : Obtiene una lista de tareas actualmente en ejecución para un conector junto con sus configuraciones.
- `GET /connectors/{name}/tasks/{taskid}/status` : Obtiene el estado actual de la tarea, incluyendo si se está ejecutando, ha fallado, está en pausa, etc., a qué worker está asignada, e información de error si ha fallado.
- `PUT /connectors/{name}/pause` : Pausa el conector y sus tareas, lo que detiene el procesamiento de mensajes hasta que se reanude el conector. Los recursos reclamados por sus tareas se dejan asignados, lo que permite al conector comenzar a procesar datos rápidamente una vez que se reanuda.
- `PUT /connectors/{name}/stop` : Detiene el conector y cierra sus tareas, desasignando cualquier recurso reclamado por sus tareas. Esto es más eficiente desde el punto de vista del uso de recursos que pausar el conector, pero puede hacer que tarde más en empezar a procesar datos una vez reanudado.
- `PUT /connectors/{name}/resume` : Reanuda un conector en pausa o detenido (o no hace nada si el conector no está en pausa o detenido)
- `POST /connectors/{name}/restart?includeTasks=<true|false>&onlyFailed=<true|false>` : Reinicia un conector y sus instancias de tareas.
- `GET /connector-plugins` : Devuelve la lista de plugins disponibles que tienen los workers (no activos)
- `DELETE /connectors/{name}` : Eliminar un conector.

6.2 Uso API REST

Aquí están algunos ejemplos comunes de cómo puedes interactuar con Kafka Connect utilizando esta API:

1. Listar todos los conectores disponibles

```
1 curl http://localhost:8083/connectors
```

Este comando devuelve una lista de todos los conectores configurados en el sistema.

2. Obtener la configuración de un conector específico

```
1 curl http://localhost:8083/connectors/connector-name/config
```

Reemplaza `connector-name` por el nombre de tu conector. Este comando devuelve la configuración actual del conector especificado.

3. Crear un nuevo conector

Para crear un nuevo conector, puedes enviar una solicitud **POST** con la configuración del conector en formato JSON:

```
1 curl -X POST -H "Content-Type: application/json" --data '{
2   "name": "connector-name",
3   "config": {
4     "connector.class":
5     "org.apache.kafka.connect.file.FileStreamSourceConnector",
6     "tasks.max": "1",
7     "file": "/var/log/myfile.log",
8     "topic": "my-topic"
9   }
}' http://localhost:8083/connectors
```

Este comando configura y lanza un nuevo conector que lee archivos de un archivo de log y publica los datos en un topic de Kafka.

4. Modificar la configuración de un conector

Para modificar un conector existente, puedes utilizar una solicitud **PUT** :

```
1 curl -X PUT -H "Content-Type: application/json" --data '{
2   "connector.class":
3   "org.apache.kafka.connect.file.FileStreamSourceConnector",
4   "tasks.max": "2",
5   "file": "/var/log/newfile.log",
6   "topic": "new-topic"
}' http://localhost:8083/connectors/connector-name/config
```

5. Eliminar un conector

Si necesitas eliminar un conector, puedes hacerlo con una solicitud **DELETE** :

```
1 curl -X DELETE http://localhost:8083/connectors/connector-name
```

6. Consultar el estado de un conector

Para verificar el estado operativo de un conector, usa:

```
1 curl http://localhost:8083/connectors/connector-name/status
```

Este comando te dará detalles sobre si el conector está en ejecución, detenido o ha encontrado errores.

Conclusión

La API REST de Kafka Connect es una herramienta poderosa para la gestión automatizada y remota de conectores en un clúster de Kafka Connect. Facilita una amplia gama de operaciones administrativas y de monitoreo, lo que la hace indispensable para operar Kafka Connect a escala y en producción. La habilidad para interactuar con Kafka Connect a través de esta API simplifica la integración con herramientas y sistemas de gestión, y permite a los desarrolladores y administradores mantener sus sistemas de manera más eficiente y efectiva.

Para una información más detallada consulta la [documentación oficial de REST API](#).

7. Ejemplo 4. Usando Kafka Connect

Supongamos que tenemos el siguiente escenario

- Tenemos un cluster de Kafka con 3 nodos usando KRaft (Kafka Raft Metadata Mode) con 2 brokers y un controlador (mismo que el [ejemplo 2](#))
- Configuraremos un cluster kafka Connect con varios workers utilizando 2 conectores(1 source y un sink)
 - **Source:** obtendremos datos de una Base de Datos Mysql consumiendo por CDC (Change Data Capture)
 - **Sink:** Guardaremos los datos en HDFS
- Este ejemplo estará orientado a un entorno de producción, asegurando alta disponibilidad y escalabilidad
- *Aunque no olvidemos que estamos realizando una prueba de concepto. Realizaremos la configuración en un sola máquina. En un entorno real nuestros servidores deberían estar en máquinas diferentes.*
- *En un entorno en producción, habría que establecer correctamente las particiones, réplicas, offset, storage, ... más convenientes dependiendo de tu entorno real. Para el ejemplo, configuraremos algunas de ellas sin entrar en más detalle.*
- *Vamos a aprovechar nuestro cluster de Hadoop para la realización de este ejemplo*

7.1 Preparación del entorno

1. Voy a usar como máquina donde ejecutaremos Kafka y Kafka Connect la misma que en el ejemplo 2, donde ya tenemos todas las configuraciones preparadas para ejecutar el cluster de Kafka.
2. Todos los conectores los voy a descargar en la carpeta `/opt/kafka/plugins`. De aquí los ire copiando a la carpeta correspondiente según el ejemplo/ejercicio en cuestión.
3. Descargamos el *source connector*. Como hemos visto en el [apartado anterior](#), tenemos 2 formas de descargar el conector, con la herramienta confluent-cli o descargando el

conector directamente. Nosotros optaremos por la segunda.

```
1 wget https://hub-downloads.confluent.io/api/plugins/confluentinc/kafka-connect-jdbc/versions/10.8.2/confluentinc-kafka-connect-jdbc-10.8.2.zip
```

Si en enlace no funciona, [descárgalo manualmente](#) y cópialo en tu sistema

4. Descomprimos

```
1 unzip confluentinc-kafka-connect-jdbc-10.8.2.zip
```

5. Observa la [documentación oficial del conector](#). Allí se indica que también necesitamos el [conector específico del SGBD](#). Seguimos los pasos y descargamos el conector de Mysql(Observa siempre la documentación oficial de cada conector, que te indicará toda la información necesaria para su configuración.)

```
1 wget https://dev.mysql.com/get/Downloads/Connector-J/mysql-connector-j-9.2.0.tar.gz
2 tar -xzf mysql-connector-j-9.2.0.tar.gz
```

6. Descargamos el *sink connector* para guardar los datos en nuestro HDFS.

```
1 wget https://hub-downloads.confluent.io/api/plugins/confluentinc/kafka-connect-hdfs3/versions/1.2.3/confluentinc-kafka-connect-hdfs3-1.2.3.zip
```

Si en enlace no funciona, [descárgalo manualmente](#) y cópialo en tu sistema

7. Descomprimos

```
1 unzip confluentinc-kafka-connect-hdfs3-1.2.3.zip
```

8. Observa la [documentación oficial del conector](#). Observa también la [configuración del conector](#), para ver los diferentes parámetros de configuración.

9. Creamos los directorios necesarios para todo el ejemplo

```
1 mkdir -p /opt/kafka/ejemplo4/libs
2 mkdir -p /opt/kafka/ejemplo4/config
3 mkdir -p /opt/kafka/ejemplo4/logs
```

10. Copiaremos los directorios completos y el fichero `jar` de `mysql-connect` en nuestro directorio `libs` del ejemplo4:

```

1 | cp -r confluentinc-kafka-connect-jdbc-10.8.2/ /opt/kafka/ejemplo4/libs/
2 | cp mysql-connector-j-9.2.0/mysql-connector-j-9.2.0.jar
  /opt/kafka/ejemplo4/libs/confluentinc-kafka-connect-jdbc-10.8.2/lib
3 | cp -r confluentinc-kafka-connect-hdfs3-1.2.3/ /opt/kafka/ejemplo4/libs/

```

11. Para mysql usaremos la imagen de [docker mysql](#). Lanzaremos la instancia en nuestro host



Docker context

Si estas usando docker desktop, **no podrás ejecutar docker y VirtualBox simultáneamente**, ya que ambos usan el mismo hipervisor para virtualizar. Por tanto, tendrás que cambiar el contexto de docker a default, para usar directamente el demonio docker desde la consola. *No olvides volver a cambiarlo cuando termines si quieres usar tu configuración original de Docker Desktop*

```

1 | $ docker context ls
2 | NAME                TYPE                DESCRIPTION
   DOCKER ENDPOINT      KUBERNETES ENDPOINT
   ORCHESTRATOR
3 | default *           moby                Current DOCKER_HOST based
   configuration        unix:///var/run/docker.sock
4 | desktop-linux       moby                Docker Desktop
   unix:///home/jaime/.docker/desktop/docker.sock
5 | $ docker context use desktop-linux
6 | desktop-linux
7 | Current context is now "desktop-linux"
8 | $ docker context ls
9 | NAME                TYPE                DESCRIPTION
   DOCKER ENDPOINT      KUBERNETES ENDPOINT
   ORCHESTRATOR
10 | default             moby                Current DOCKER_HOST based
   configuration        unix:///var/run/docker.sock
11 | desktop-linux *     moby                Docker Desktop
   unix:///home/jaime/.docker/desktop/docker.sock
12 | $ docker context use default
13 | default
14 | Current context is now "default"
15 | $ docker context ls
16 | NAME                TYPE                DESCRIPTION
   DOCKER ENDPOINT      KUBERNETES ENDPOINT
   ORCHESTRATOR
17 | default *           moby                Current DOCKER_HOST based
   configuration        unix:///var/run/docker.sock
18 | desktop-linux       moby                Docker Desktop
   unix:///home/jaime/.docker/desktop/docker.sock

```

```

1  docker run -d \
2  --rm \
3  --name mysql-kafka \
4  -p 3306:3306 \
5  -e MYSQL_ROOT_PASSWORD=pass \
6  -v mysql_data:/var/lib/mysql \
7  mysql:latest

```

12. Usaremos también la imagen de [docker phpmyadmin](#) para mayor comodidad.

```

1  docker run -d \
2  --rm \
3  --name phpmyadmin-kafka \
4  --link mysql-kafka \
5  -e PMA_HOST=mysql-kafka \
6  -p 8080:80 \
7  phpmyadmin

```

Podemos entrar con las credenciales `root/pass`

7.2 Cluster Kafka

Recuerda que usaremos la misma configuración y máquina que en el ejemplo 2

1. Hacemos 2 y 1 copia de los ficheros correspondientes de configuración

```

1  cp config/kraft/controller.properties
   /opt/kafka/ejemplo4/config/controller1.properties
2  cp config/kraft/broker.properties
   /opt/kafka/ejemplo4/config/broker1.properties
3  cp config/kraft/broker.properties
   /opt/kafka/ejemplo4/config/broker2.properties

```

2. Asignamos la configuración al controller

controller1.properties

```

1  # Server Basics
2  process.roles=controller
3  node.id=1
4  controller.quorum.voters=1@localhost:9093
5  # Socket Server Settings
6  listeners=CONTROLLER://localhost:9093
7  controller.listener.names=CONTROLLER
8  # Log Basics
9  log.dirs=/opt/kafka/ejemplo4/logs/controller1

```

3. Asignamos la siguiente configuración para cada broker

broker1**broker1.properties**

```

1  # Server Basics
2  process.roles=broker
3  node.id=2
4  controller.quorum.voters=1@localhost:9093
5  # Socket Server Settings
6  listeners=PLAINTEXT://localhost:9094
7  advertised.listeners=PLAINTEXT://localhost:9094
8  # Log Basics
9  log.dirs=/opt/kafka/ejemplo4/logs/broker1

```

broker2**broker2.properties**

```

1  # Server Basics
2  process.roles=broker
3  node.id=3
4  controller.quorum.voters=1@localhost:9093
5  # Socket Server Settings
6  listeners=PLAINTEXT://localhost:9095
7  advertised.listeners=PLAINTEXT://localhost:9095
8  # Log Basics
9  log.dirs=/opt/kafka/ejemplo4/logs/broker2

```

4. Iniciamos Kafka con KRaft

```

1  #Genera un cluster UUID
2  KAFKA_CLUSTER_ID="$(bin/kafka-storage.sh random-uuid)"
3  echo $KAFKA_CLUSTER_ID
4
5  #Formateamos los directorios de log
6  bin/kafka-storage.sh format -t $KAFKA_CLUSTER_ID -c
/opt/kafka/ejemplo4/config/controller1.properties
7  bin/kafka-storage.sh format -t $KAFKA_CLUSTER_ID -c
/opt/kafka/ejemplo4/config/broker1.properties
8  bin/kafka-storage.sh format -t $KAFKA_CLUSTER_ID -c
/opt/kafka/ejemplo4/config/broker2.properties

```

5. Iniciamos los server(1 controller y 2 brokers) cada uno en una terminal

```

1  #Ejecuta el servidor Kafka
2  bin/kafka-server-start.sh
/opt/kafka/ejemplo4/config/controller1.properties
3  bin/kafka-server-start.sh /opt/kafka/ejemplo4/config/broker1.properties
4  bin/kafka-server-start.sh /opt/kafka/ejemplo4/config/broker2.properties

```

7.3 Creamos el topic

1. Creamos el topic con factor de replica 2 y 2 particiones. El topic debe conectarse a un broker.

```
1 bin/kafka-topics.sh --create --topic empleados-employees --bootstrap-server localhost:9094 --replication-factor 2 --partitions 2
```

2. Podemos ver la descripción del topic creado

```
1 bin/kafka-topics.sh --describe --topic empleados-employees --bootstrap-server localhost:9094
```

```
1 Topic: empleados-employees TopicId: UomyAfUiRN6vhwHtKET2fg
PartitionCount: 2 ReplicationFactor: 2 Configs: segment.bytes=1073741824
2 Topic: empleados-employees Partition: 0 Leader: 3 Replicas: 3,2
Isr: 3,2
3 Topic: empleados-employees Partition: 1 Leader: 2 Replicas: 2,3
Isr: 2,3
```

3. Incluso podemos ver los topics existentes

```
1 bin/kafka-topics.sh --list --bootstrap-server localhost:9094
```

Los topics `__consumer_offsets`, `connect-configs`, `connect-offsets` y `connect-status` son creados por el propio Kafka Connect

4. Aprovecharemos la Consumer API para ver como Kafka Connect está consumiendo los datos correctamente

```
1 bin/kafka-console-consumer.sh --topic empleados-employees --from-beginning --bootstrap-server localhost:9094
```

7.4 Cluster Kafka Connect

Siguiendo nuestro escenario, vamos a levantar 3 workers, para cumplir una correcta Alta disponibilidad, Balanceo de carga y Escalabilidad.

1. Configuración de los workers. Estos apuntan a los brokers del cluster de Kafka(`bootstrap.servers`)

worker1

worker1.properties

```
1 bootstrap.servers=localhost:9094,localhost:9095
2 group.id=connect-cluster
3 key.converter=org.apache.kafka.connect.json.JsonConverter
4 value.converter=org.apache.kafka.connect.json.JsonConverter
```

```
5 key.converter.schemas.enable=true
6 value.converter.schemas.enable=true
7 offset.storage.topic=connect-offsets
8 offset.storage.replication.factor=2
9 config.storage.topic=connect-configs
10 config.storage.replication.factor=2
11 status.storage.topic=connect-status
12 status.storage.replication.factor=2
13 plugin.path=/opt/kafka/ejemplo4/libs
14 listeners=http://localhost:8083
```

worker2

worker2.properties

```
1 bootstrap.servers=localhost:9094,localhost:9095
2 group.id=connect-cluster
3 key.converter=org.apache.kafka.connect.json.JsonConverter
4 value.converter=org.apache.kafka.connect.json.JsonConverter
5 key.converter.schemas.enable=true
6 value.converter.schemas.enable=true
7 offset.storage.topic=connect-offsets
8 offset.storage.replication.factor=2
9 config.storage.topic=connect-configs
10 config.storage.replication.factor=2
11 status.storage.topic=connect-status
12 status.storage.replication.factor=2
13 plugin.path=/opt/kafka/ejemplo4/libs
14 listeners=http://localhost:8084
```

worker3

worker3.properties

```
1 bootstrap.servers=localhost:9094,localhost:9095
2 group.id=connect-cluster
3 key.converter=org.apache.kafka.connect.json.JsonConverter
4 value.converter=org.apache.kafka.connect.json.JsonConverter
5 key.converter.schemas.enable=true
6 value.converter.schemas.enable=true
7 offset.storage.topic=connect-offsets
8 offset.storage.replication.factor=2
9 config.storage.topic=connect-configs
10 config.storage.replication.factor=2
11 status.storage.topic=connect-status
12 status.storage.replication.factor=2
13 plugin.path=/opt/kafka/ejemplo4/libs
14 listeners=http://localhost:8085
```



Entorno de producción real

Recuerda que estamos simulando un entorno real pero en una sola máquina. Debido a esto, hemos tenido que añadir en la configuración que la API REST de cada worker escuche en puertos diferentes con la propiedad `listeners`, que puedes consultar en la [documentación oficial](#). En un entorno real, para asegurar la Alta Disponibilidad, Redundancia, Tolerancia a Fallos y Balanceo de Carga habría que desplegar los workers en diferentes nodos y/o zonas de disponibilidad. Y por tanto, no tendríamos que modificar este puerto. En ese caso, el fichero de configuración podría ser el mismo.

2. Guardamos en `/opt/kafka/ejemplo4/config`

3. Iniciamos los 3 workers de Kafka Connect (cada uno en una terminal)

```
1 bin/connect-distributed.sh /opt/kafka/ejemplo4/config/worker1.properties
2 bin/connect-distributed.sh /opt/kafka/ejemplo4/config/worker2.properties
3 bin/connect-distributed.sh /opt/kafka/ejemplo4/config/worker3.properties
```

4. Puedes comprobar en el log que los workers están levantados (cada uno escuchando en su puerto correspondiente) y podemos hacer una consulta a través de la API REST, que evidentemente, devolverá que no hay conectores todavía:

```
1 curl http://localhost:8083/connectors
2 curl http://localhost:8084/connectors
3 curl http://localhost:8085/connectors
```

5. Podemos ver también la lista de plugins y conectores que los workers tienen a su disposición

```
1 curl http://localhost:8083/connector-plugins
```

Si quieres mejorar el formato de salida

```
1 curl http://localhost:8083/connector-plugins | jq
```

7.5 Preparación de Source y Sink

Preparación de nuestros datos en mysql

Para nuestra Base de Datos de prueba, vamos a usar una propia desarrollada por Mysql para entornos de pruebas, llamada "employee data". [Mysql dispone de varias](#). Nuestra BD elegida tiene su [propia documentación](#) y [propio repositorio en github](#), que vamos a usar.

1. Descargamos los datos y los copiamos en el contenedor

```
1 git clone https://github.com/datacharmer/test_db.git
2 docker cp test_db/ mysql-kafka:/
```

2. Entramos en el contenedor

```
1 docker exec -it mysql-kafka bash
2 cd test_db
```

3. Creamos la Base de datos y el dataset. Este comando nos pedirá la contraseña de tu usuario MySQL y luego creará una nueva base de datos llamada employees con varias tablas llenas de datos.

```
1 mysql -u root -p < employees.sql
```

4. Comprobamos entrando

```
1 mysql -u root -p
```

5. Mostramos las bases de datos y elegimos employees

```
1 mysql> show databases;
2 +-----+
3 | Database |
4 +-----+
5 | employees |
6 | information_schema |
7 | mysql |
8 | performance_schema |
9 | sys |
10 +-----+
11 5 rows in set (0.00 sec)
12
13 mysql> use employees;
```

6. Observamos la tabla employees (SELECT * FROM employees;) o mediante phpmyadmin que los datos de la base de datos se han cargado correctamente

```
1 mysql> SELECT * FROM employees;
2 ...
3 ...
4 ...
5 | 499998 | 1956-09-05 | Patricia | Breugel | M | 1993-
10-13 |
6 | 499999 | 1958-05-01 | Sachin | Tsukuda | M | 1997-
11-30 |
7 +-----+-----+-----+-----+-----+-----+
8 300024 rows in set (0.13 sec)
```

7. También podemos mediante phpmyadmin (<http://localhost:8080/>) que los datos de la base de datos se han cargado correctamente

Preparación de HDFS

1. Tener Hadoop levantado y que no esté en modo seguro.
2. Crear el directorio correspondiente en nuestro HDFS: `hdfs dfs -mkdir -p /bda/kafka/ejemplo4`.

7.6 Conectores

1. **Source Conector (JDBC):** Debemos tener en cuenta la configuración de nuestro mysql. Guárdalo en `/opt/kafka/ejemplo4/config`

mysql-employees-source-connector.json

```

1  {
2    "name": "mysql-employees-source-connector",
3    "config": {
4      "connector.class": "io.confluent.connect.jdbc.JdbcSourceConnector",
5      "tasks.max": "3",
6      "connection.url": "jdbc:mysql://192.168.56.1:3306/employees",
7      "connection.user": "root",
8      "connection.password": "pass",
9      "table.whitelist": "employees",
10     "mode": "incrementing",
11     "incrementing.column.name": "emp_no",
12     "topic.prefix": "empleados-",
13     "logs.dir": "/opt/kafka/ejemplo4/logs/JdbcSource",
14     "poll.interval.ms": "5000"
15   }
16 }
```

Cuando este conector utiliza tablas de base de datos como fuente de datos, asigna a cada tabla un topic de Kafka con el formato `<prefix><table_name>`. Si el conector está configurado para utilizar una consulta personalizada, el nombre del tema es simplemente `<prefix>`. El prefijo es obligatorio mediante la propiedad `topic.prefix`. Puedes consultar esta información en la [documentación del conector](#)

Con la propiedad `table.whitelist` indicamos que queremos copiar toda la tabla y `"mode": "incrementing"` para que esa replica esté basada en una columna de incremento. Podríamos especificar una consulta determinada, para ellos eliminamos la propiedad `table.whitelist` y añadimos la query específica, por ejemplo `"query": "SELECT * FROM employees WHERE department_id = 5"`

1. **Sink Conector (HDFS):** Debemos tener en cuenta la configuración de nuestro HDFS. Guárdalo en `/opt/kafka/ejemplo4/config`

hdfs3-employees-sink-connector.json

```

1  {
2    "name": "hdfs3-employees-sink-connector",
3    "config": {
4      "connector.class": "io.confluent.connect.hdfs3.Hdfs3SinkConnector",
5      "tasks.max": "3",
6      "confluent.topic.bootstrap.servers": "localhost:9094",
7      "topics": "empleados-employees",
8      "store.url": "hdfs://cluster-bda:9000/bda/kafka/ejemplo4",
9      "logs.dir": "logs/hdfs3sink",
10     "format.class": "io.confluent.connect.hdfs3.avro.AvroFormat",
11     "path.format": "'year'=YYYY/'month'=MM/'day'=dd",
12     "flush.size": "1000",
13     "hadoop.conf.dir": "/opt/hadoop-3.4.1/etc/hadoop/",
14     "hadoop.home": "/opt/hadoop-3.4.1/"
15   }
16 }

```

No olvides crear el directorio correspondiente en nuestro HDFS: `hdfs dfs -mkdir -p /bda/kafka/ejemplo4`. Por supuesto, asegúrate de tener Hadoop levantado y no esté en modo seguro.

7.7 Iniciar conectores

Usando la API REST vamos a iniciar estos conectores

1. Primero comprobamos los conectores activos, que no debería haber ninguno

```
1 curl http://localhost:8083/connectors
```

2. Justo antes de iniciar los conectores, observa la ventana donde hemos usado Consumer API para observar como consume los datos de la Base de Datos.
3. Iniciamos los conectores. *Recuerda que puedes pasar directamente la configuración en el endpoint*

```

1 curl -X POST -H "Content-Type: application/json" --data
  @/opt/kafka/ejemplo4/config/mysql-employees-source-connector.json
  http://localhost:8083/connectors
2 curl -X POST -H "Content-Type: application/json" --data
  @/opt/kafka/ejemplo4/config/hdfs3-employees-sink-connector.json
  http://localhost:8083/connectors

```

4. Comprobamos que están activos

```

1 curl http://localhost:8083/connectors
2 curl http://localhost:8083/connectors/mysql-employees-source-
  connector/status

```

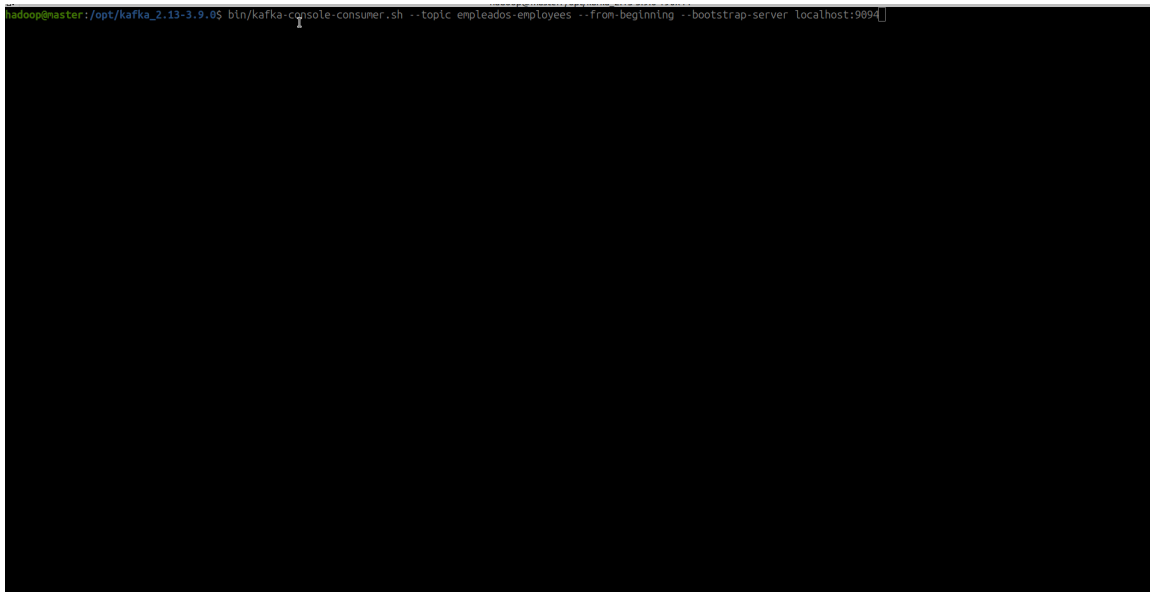
```
3 | curl http://localhost:8083/connectors/hdfs3-employees-sink-  
connector/status
```

5. También puedes usar aplicaciones de terceros como POSTMAN

7.8 Ejecución

Observa el video de la ejecución de nuestro ejemplo:

1. Usamos Consumer API para observar como Kafka consume los registros de la Base de Datos.
2. Iniciamos el primer conector, que accede a la base de datos. En ese momento, Consumer API muestra como procesa los datos Kafka.
3. Iniciamos el segundo conector, que accede a HDFS, transforma los datos en formato Avro y los almacena en el directorio indicado.
4. Podemos ver como son almacenados los datos en HDFS



```
hadoop@master:/opt/kafka_2.13-3.9.0$ bin/kafka-console-consumer.sh --topic empleados-employees --from-beginning --bootstrap-server localhost:9094
```

Animación 7.4_Kafka Connect: Ejemplo 4