Sistemas de Big Data

Conforme a contenidos del «Curso de Especialización en Inteligencia Artificial y Big Data»

Universidad de Castilla-La Mancha

Escuela Superior de Informática
Ciudad Real

Capítulo 7

Gestión y Almacenamiento de Datos - I

Julio Alberto López Gómez

Los capítulos anteriores han servido, entre otras cosas, para certificar el **crecimiento exponencial** de los datos generados por cualquier proceso y/o organización. Este crecimiento, unido a la **diversidad** en las fuentes u orígenes de datos, tipos y formatos, etc. ha conducido a la adopción de herramientas y tecnologías **Big Data** para el almacenamiento, procesamiento y analítica de todos estos datos de forma **distribuida**, con el objetivo de extraer información y conocimiento de grandes volúmenes de datos de forma **eficiente y rentable.**

Ante esta situación, las primeras cuestiones fundamentales que pueden surgir son: **¿cómo es posible introducir los datos con los que se pretende trabajar en un clúster para almacenarlos y procesarlos de forma distribuida?, ¿qué tecnologías existen a nuestra disposición para ello?**. Este capítulo y el siguiente describen **Apache Hadoop** como el framework más extendido para trabajar en entornos Big data y se centra en las herramientas **Apache Flume** y **Apache Sqoop** como servicios de **ingestión de datos** en sistemas Big Data.

7.1. Apache Hadoop

Apache Hadoop fue creado por Doug Cutting en 2006 a partir de Apache Nutch, un buscador web. Hadoop es un framework de almacenamiento y procesamiento distribuido que permite trabajar con grandes volúmenes de datos. En la actualidad, es el framework más utilizado para trabajar en entornos Big Data. Esto es así, ya que Hadoop ofrece una plataforma **fiable y escalable** para el almacenamiento y análisis distribuido. Además, se puede decir que Hadoop es un framework **asequible**, al ser de código abierto (*open-source*) y poder ejecutarse en hardware "básico" o comercial. El potencial de Hadoop en el almacenamiento y procesamiento distribuido de datos masivos está ampliamente reconocido por la industria,

lo que se refleja en la gran cantidad de productos que incorporan el uso de esta tecnología de una forma u otra, como es el caso de IBM, Oracle o Microsoft, así como en la aparición de empresas especialistas en este framework como Cloudera, MapR o Hortonworks.

Apache Hadoop: Es un framework de código abierto, fiable, escalable y asequible que da soporte al almacenamiento y procesamiento distribuido de grandes conjuntos de datos para trabajar en entornos Big Data[1]

La plataforma Hadoop ofrece un **ecosistema** en el que se integran multitud de utilidades, herramientas tecnológicas y servicios para abordar y **resolver problemas de Big Data.** Estas herramientas van desde tecnologías para la **gestión de los datos (Flume, Sqoop, Oozie...)** hasta tecnologías de **procesamiento de datos (MapReduce, Yarn)** pasando por tecnologías de **acceso a los datos (Pig, Hive, Mahout...).** La figura 7.1 muestra un esquema de algunas de las tecnologías más ampliamente utilizadas en el ecosistema Hadoop. A continuación, en las siguientes subsecciones, se profundizará en los componentes más importantes de este ecosistema.

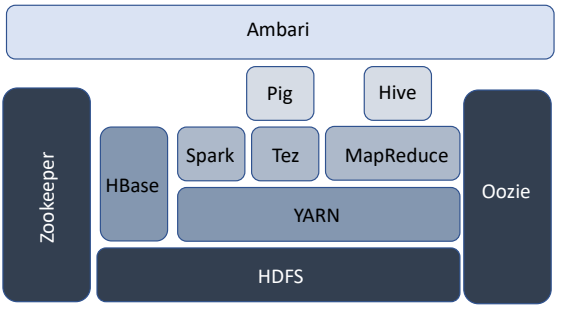

Figura 7.1: Ecosistema Hadoop: Tecnologías más utilizadas.

7.1.1. Hadoop Distributed File System (HDFS)

Es el sistema de ficheros distribuido que proporciona el framework Hadoop. Inicialmente, fue construido tomando como base el sistema de ficheros de Google (GFS). HDFS permite el almacenamiento de de ficheros en miles de servidores distribuidos que **no requieren de un hardware especializado** o inaccesible ni de unas opciones de configuración avanzadas. Por este motivo, **HDFS puede ejecutarse en cualquier centro de datos genérico.** Este sistema de archivos puede configurarse

para replicar los datos almacenados en diferentes máquinas, de forma que no se produzcan pérdidas de datos si una máquina falla. **El factor de replicación por defecto en HDFS es tres**, por lo que los datos almacenados en HDFS se replican en tres servidores dentro del clúster.

HDFS: Sistema de archivos del framework hadoop altamente distribuido, escalable, tolerante a fallos y provisto de mecanismos de replicación y seguridad para ejecutarse en hardware comercial

Al igual que cualquier otro sistema de ficheros, HDFS almacena los datos en unidades llamadas **bloques.** Un fichero está formado por uno o más bloques en función de su tamaño. En comparación a otros sistemas de archivos, los bloques manejados por HDFS tienen un tamaño mayor, que oscila entre los **128 y los 512 MB.** La replicación se realiza **a nivel de bloque,** por lo que HDFS garantiza que cada uno de los bloques que compone un archivo se replique según el factor de replicación, posibilitando así la replicación del archivo completo y **almacenando cada copia en un servidor diferente,** incrementando la fiabilidad.

El sistema HDFS consiste en dos tipos de servidores: *name nodes* y *data nodes*. Generalmente, **un clúster Hadoop está formado por dos *name nodes* y varios *data nodes*.** Los *data nodes* son aquellos nodos del cluster en los que los datos son almacenados. Por su parte, los *name nodes* atienden las peticiones de los clientes y de los *data nodes*, almacenan los metadatos de los ficheros y bloques de datos en el sistema y se encargan de mapear cada fichero en sus correspondientes bloques. En cualquier instante de tiempo, existe siempre **un *name node* activo y otro opcional, en standby.** Este último actúa a modo de copia de seguridad del primero y toma el relevo en caso de que el *name node* activo presente alguna incidencia.

Cada vez que un cliente realiza una **escritura,** esta se escribe inicialmente en un archivo **local** en la máquina del cliente. Será cuando el cliente cierre el archivo o el tamaño del archivo local supere el límite de bloque, cuando sea crea el archivo en HDFS o se añade un nuevo bloque al archivo que se estuviera escribiendo en el sistema de archivos distribuidos. Entonces, el *name node* asigna los bloques a este archivo y se escribe cada uno de ellos, replicándolos en tantos servidores según el factor de replicación.

Una vez creado un archivo en HDFS, este no puede editarse en el sentido amplio del término, sino que a él s**solo se pueden añadir datos.** De esta forma, una vez cerrado un archivo que acaba de crearse en HDFS, solo puede ser abierto para añadir datos en él. HDFS no garantiza que las escrituras realizadas en un archivo sean visibles para todos los clientes, hasta que el cliente que escribe los datos no cierre el archivo. En cada lectura, el cliente obtiene del *name node* la ubicación de los bloques que representan el archivo y lee los datos desde el *data node*. El hecho de que cada archivo esté replicado en múltiples servidores es completamente transparente para el usuario, que no debe preocuparse por los aspectos de tolerancia a fallos y replicación a la hora de programar.

Es posible interactuar con HDFS a través de la API para clientes, que proporciona un conjunto de comandos shell que pueden ser utilizados para realizar operaciones sobre los ficheros HDFS y que se pueden consultar en la documentación de hadoop[2]. Un comando HDFS tiene la siguiente forma:

```
hdfs dfs -<command><options>
```

De esta forma, si se pretende listar el conjunto de archivos que hay en HDFS en un directorio llamado logs, se podría escribir el siguiente comando:

```
hdfs dfs -ls /logs
```

HDFS trabaja con dos tipos de archivo: **divisibles y no divisibles.** En los primeros, un archivo puede dividirse en múltiples piezas llamadas *splits*. En ellos, es posible buscar el inicio de un registro desde cualquier punto. Este tipo de archivos son **compatibles** con MapReduce. Por otra parte, los archivos en formatos no divisibles, los datos son tratados como una unidad. En cualquier caso, es recomendable trabajar en HDFS con archivos en formatos binarios, ya que este tipo de archivos incluye información que permite comprobar si el archivo está incompleto o corrupto. Algunos de los tipos de datos divisibles más comúnmente utilizados son Avro o *sequence file*, este último muy utilizado en operaciones MapReduce.

7.1.2. HBase

Se trata de un servicio **implementado sobre el sistema de ficheros HDFS** y que, por tanto, utiliza los servicios HDFS para la replicación de los datos. HBase es el **gestor de base de datos no relacional** propio del framework Hadoop. Su principal funcionalidad y uso es la **escritura y actualización de datos en tiempo real,** ofreciendo así datos en tiempo real a las aplicaciones diseñadas sobre Hadoop.

HBase utiliza un sistema de almacenamiento de tipo **clave-valor.** Este modelo de datos es similar al de una base de datos relacional (tablas, campos, registros, etc) con algunas diferencias significativas como que las columnas no son fijas en el esquema (dos filas o registros pueden no presentar el mismo número de columnas o campos) y que las columnas o campos pueden ser creados dinámicamente por un cliente. Cada una de las filas es accesible a través de una **clave de fila,** concepto análogo al de clave primaria. Al igual que ocurre en el modelo relacional, para cada una de las columnas solo puede haber un valor almacenado, si bien es cierto que **HBase puede mantener los últimos n valores de dicha columna (versiones).** Además, HBase permite agrupar las columnas en **familias de columnas,** por lo que es común agrupar aquellos columnas cuyos datos se escriben y se acceden de forma similar.

[2] https://hadoop.apache.org/docs/current2/hadoop-project-dist/hadoop-common/CommandsManual.html

HBase: Sistema gestor de base de datos distribuido propio del framework Apache Hadoop. Implementado sobre HDFS, sigue un modelo datos de tipo clave-valor que permite la escritura y actualización de datos en tiempo real.

La interacción con HBase en un cluster se puede realizar a través de una **API Java** diseñada a tal efecto. La escritura de datos en HBase se realiza en base a dos operaciones fundamentales: *Put*, que se corresponde con la escritura de una única fila y *Increment* que es la operación que se utiliza para añadir valores en alguna de las columnas. Ambas operaciones representan una llamada a un procedimiento remoto que permite la escritura en HBase. Además de la interacción por medio de la API, HBase dispone de una serie de **comandos shell** para la interacción con el clúster. Existen multitud de comandos que permiten realizar operaciones de tipo *put*, *increment*, *get*, *delete*, *scan*...[3] Para comenzar una sesión del shell HBase, es posible escribir el comando:

```
hbase shell
```

7.1.3. Otras tecnologías

El ecosistema de Hadoop aglutina un gran número de tecnologías y servicios para el acceso, gestión y procesamiento de grandes volúmenes de datos. A continuación, y de manera más resumida, se describen otras tecnologías ampliamente utilizadas y que se mostraban en la figura 7.1.

- **Zookeeper:** Cualquier aplicación distribuida requiere de una correcta gestión de la sincronización, información de configuración y otros servicios de grupo. La implementación de todos estos servicios no es una tarea fácil, lo que hace que haya mucho trabajo para corregir errores de sincronización y condiciones de carrera. Zookeper es un servicio centralizado de Hadoop para la gestión de la sincronización.
- **Oozie:** Herramienta que permite la programación de flujos de trabajo en Hadoop. De esta forma, los administradores de los cluster pueden definir y programar flujos de trabajo para, de forma secuencial, gestionar y manipular los datos.
- **Yarn:** Es una de las principales herramientas del ecosistema Hadoop. Yarn es la aplicación de gestión de recursos de Hadoop, que permite asignar los recursos a las aplicaciones en ejecución, programando procesos para que se ejecuten en diferentes máquinas del cluster.
- **MapReduce:** Es el framework de procesamiento paralelo y distribuido de datos que incorpora el ecosistema Hadoop. Se implementa sobre HDFS.

[3] `http://wiki.apache.org/hadoop/Hbase/Shell`

- **Spark:** Framework de procesamiento de datos distribuido que, frente a MapReduce, supone una mejora al trabajar con memoria RAM y ser compatible con gran cantidad de lenguajes. Sin embargo, Spark no cuenta con su propio sistema de archivos distribuido, por lo que se utiliza sobre HDFS, aprovechando el sistema de ficheros de Hadoop.
- **Tez:** Herramienta que permite al administrador diseñar aplicaciones de alto rendimiento para la gestión de datos interactivos, mejorando la velocidad de Map Reduce.
- **Pig:** Compatible con MapReduce, ofrece un lenguaje de alto nivel para analizar grandes conjuntos de datos, implementando programas de análisis y ejecutándolos en la infraestructura necesaria para evaluar los programas implementados.
- **Hive:** Herramienta de almacén de datos que permite la lectura, gestión y acceso a grandes conjuntos de datos distribuidos mediante SQL. Proporciona a los usuarios una herramienta de línea de comandos y un controlador JDBC para la conexión con Hive.
- **Ambari:** Framework que permite el aprovisionamiento, gestión y supervisión de clusters Hadoop. Proporciona una interfaz web de gestiónd de Hadoop intuitiva, simplificando la gestión.

7.2. Apache Flume

Apache Hadoop se ha convertido en los últimos años en el framework estándar de almacenamiento y procesamiento distribuido de Big Data en las grandes empresas. Los procesos y aplicaciones que se dan en ellas producen grandes cantidades de datos que se almacenan en **HDFS y/o HBase.** El almacenamiento de todos estos datos debe realizarse de forma **fiable** y de tal manera que la escritura y el almacenamiento de los ficheros sea compatible con herramientas de procesamiento como MapReduce, Hive, Pig... En esta sección, se describe detalladamente **Apache Flume** como sistema **fiable y escalable** de **ingesta** de datos en entornos distribuidos propios del Big Data, como es Hadoop.

7.2.1. Motivación y necesidad

¿Es necesario disponer de herramientas de ingestión de datos en sistemas de archivos distribuidos?. Una alternativa podría pasar por **la escritura directa** de los datos que se pretenden inyectar en HDFS o HBase. Sin embargo, en entornos de tipo cluster, existen generalmente una gran cantidad de servidores produciendo datos (cientos e incluso miles). La escritura simultánea de varios servidores en HDFS o HBase podría causar problemas, como pueden ser el **aumento de la latencia, sobrecarga y pérdida de funcionamiento en el *name node*** que es el componente encargado de realizar estas operaciones...

HDFS permite la **escritura de un archivo por parte de un único cliente en un mismo instante de tiempo.** En caso contrario, cientos de ficheros podrían ser escritos al mismo tiempo. El componente encargado de realizar las operaciones necesarias para que un archivo sea creado o se asignen nuevos bloques es el *name node*. La escritura de muchos archivos al mismo tiempo en un único servidor podría causar que el servidor cayera. Además, cuando una gran cantidad de máquinas están escribiendo datos en una máquina o un número pequeño de ellas, la conectividad puede verse afectada, aumentando la latencia. Por todo lo anterior, es necesario **programar la ingesta de datos en HDFS o HBase** para que los datos se inyecten de forma organizada y controlada.

Apache Flume: Servicio distribuido, disponible y fiable para recolectar, agregar y mover grandes cantidades de datos.

Apache Flume es una herramienta de **ingestión de datos** en sistemas de archivos distribuidos que puede escalar fácilmente a un gran número de datos y que altamente personalizable. Este servicio permite definir agentes y *pipelines* de agentes que se encargan de recolectar los datos que se pretenden inyectar (eventos) y transmitirlos hacia el repositorio de destino (generalmente HDFS o HBase). Flume está diseñado principalmente para la extracción e ingestión de datos **no estructurados,** como los provenientes de redes sociales, smartphones, etc. en sistemas de archivos distribuidos. La figura 7.2 muestra de forma esquemática el proceso de ingestión de datos.

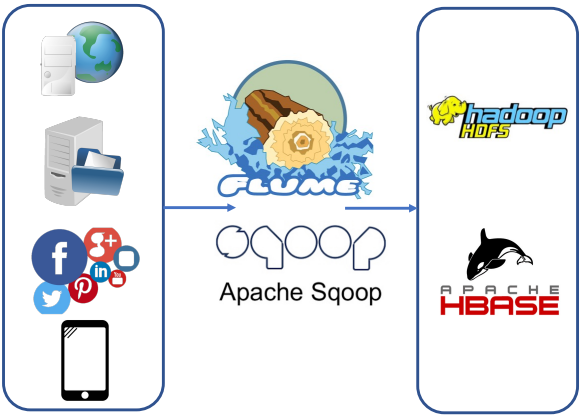

Figura 7.2: Esquema del proceso de ingestión de datos en sistemas de ficheros distribuidos

7.2.2. Arquitectura de agentes

La unidad básica de despliegue de **Apache Flume** es el **agente.** Un agente es un **proceso JVM** que se encarga de transmitir la información que se pretende inyectar **(eventos)** al **repositorio terminal (generalmente HDFS o HBase)** o bien a otro agente, en cuyo caso se dice que se ha implementado un ***pipeline* de agentes Flume** para la inyección de los datos. Grosso modo, un agente Flume se encarga de recolectar o recibir informacion por parte de los servidores de aplicación y transmitirla hacia el siguiente agente del flujo o bien directamente al repositorio terminal.

La arquitectura de un agente flume consta de tres componentes principales: la fuente o *source*, el canal (*channel*) y el sumidor o *sink*. La figura 7.3 muestra de forma esquemática la arquitectura de un agente Flume.

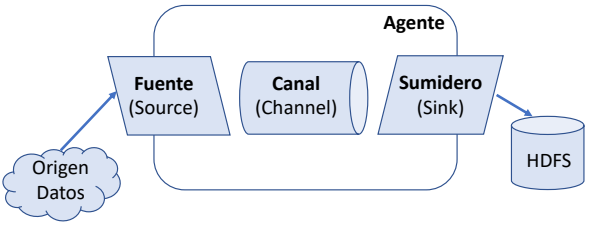

Figura 7.3: Arquitectura de un agente Flume

- **Fuente (*Source)*:** Se trata del primer componente activo de la arquitectura del agente, que se encarga de recibir o recolectar los eventos de las aplicaciones que están produciendo los datos. Un agente Flume puede tener uno o más *sources* que pueden escuchar en uno o más puertos de la red e incluso leer datos del sistema de ficheros local. Cada *source* debe estar conectado con al menos un canal (*channel*), sobre los cuales el *source* puede realizar escrituras, replicando los eventos a todos o solo a uno de los canales a los que está conectados, en función del criterio del desarrollador.
- **Canal (*Channel*):** Aunque el canal/es pueden ejecutar sus propios procesos de limpieza o recolección de basura, se le considera un componente pasivo de la arquitectura de un agente Flume. El canal se comporta como una cola o búfer de información en el que se almacenan los eventos que han sido recibidos por el *source* del agente pero que todavía no se han transmitido a otro agente o escrito en un sistema de almacenamiento (HDFS o HBase) y que serán leídos por el sumidero (*sink*). Respecto a su funcionamiento, múltiples *sources* pueden escribir información y transmitirla a través del canal, del mismo modo que múltiples *sinks* pueden leer eventos del canal. No obstante, si múltiples *sinks* leen del mismo canal, se garantiza que exactamente solo uno de ellos leerá la información de un evento específico del canal, lo que implica que un mismo evento no puede ser leído de forma simultánea por más de un sumidero.

- **Sumidero (*Sink*):** El sumidero es el último componente activo de la arquitectura de un agente Flume. Los sumideros sondean continuamente el canal o los canales a los que están conectados para leer y eliminar eventos. De esta forma, los *sinks* envían los eventos al siguiente salto (en caso de tener un flujo de agentes Flume) o directamente al repositorio terminal. Cuando los datos se encuentran en el siguiente salto o en el destino, los *sinks* notifican a los canales para que estos últimos puedan eliminar los eventos.

Flume **no restringe** el número de *sources*, *channels* y *sinks* dentro de un agente. Por tanto, es posible que un *source* reciba eventos y, a través de la configuración, replique los eventos en múltiples destinos. Esto es posible gracias al esquema de funcionamiento a través de **procesadores de canal, interceptores y selectores de canal.**

Cada uno de los *source* definidos en un agente tiene su propio **procesador de canal** (*channel processor*). El procesador es de canales es quien escribe los eventos en los canales, transmitiendo estos eventos a los interceptores configurados en el *source*. Los **interceptores,** por su parte, son pequeñas piezas de código que leen un determinado evento y lo modifican o incluso lo eliminan según el criterio que se ha programado en el interceptor. Por ejemplo, un evento puede eliminarse si cumple una expresión regular, se pueden añadir cabeceras a determinados eventos en función de un criterio establecido... Un mismo *source* puede utilizar **múltiples interceptores,** que son llamados en el orden definido en su configuración, lo que recibe el nombre de **cadena de responsabilidad**. Después de que los interceptores hayan realizado su función, éstos devuelven una lista de eventos que son los que se deben escribir en los canales a los que está conectado el *source*. Es tarea del **selector de canal** el determinar qué eventos se van a escribir en qué canales, aplicando los criterios que correspondan en cada caso y decidiendo en qué canales se debe escribir obligatoriamente un evento y en qué canales es opcional que se escriba un determinado evento.

La utilización de los *sources* y *sinks* configurados en Flume permite garantizar la **durabilidad de la transmisión,** asegurando que los eventos no se perderán durante la transmisión. Además, Flume puede escribir **datos duplicados** si se producen errores o tiempos de espera inesperados en el *pipeline*. No obstante, Flume podría **perder los datos** si se producen fallos en los discos que contienen los canales. Ante esta posibilidad, Flume permite replicar eventos a través de flujos redundantes, aunque esto implicaría realizar algún tipo de post-procesamiento para procesar los datos duplicados

7.2.3. Implementación de agentes

En este apartado se pretende ilustrar la estructura e implementación de un agente Flume básico. Para ello, se creará un fichero de texto con extensión .conf en el cual se especificarán todas las opciones de configuración de un agente. Merece la pena destacar que dentro de un mismo archivo de configuración se pueden definir y configurar tantos agentes como se desee. Será en la fase de ejecución, cuando por línea de comandos se especifique cuál de los agentes configurados en el archivo es el que se desea ejecutar.

La configuración de un agente consta, fundamentalmente, de los siguientes elementos:

1. Definición de los componentes del agente: sources, channels y sinks
2. Configuración de las propiedades de cada componente
3. Vinculación de sources y sinks a los channels correspondientes

El listado 7.1 muestra una plantilla de fichero de configuración de agente Flume.

Listado 7.1: Plantilla de definición de Agentes Flume

```
# Plantilla para la definición de agentes Flume

# Definición de sources, channels y sinks
# Agente basico con un unico source y sink
<Agent>.sources  = <Source>
<Agent>.sinks    = <Sink>
<Agent>.channels = <Channel1> <Channel2>

# Propiedades del source
<Agent>.sources.<Source>.<someProperty> = <someValue>

# Propiedades del canal/es
<Agent>.channel.<Channel>.<someProperty> = <someValue>
...

# Propiedades del sink
<Agent>.sources.<Sink>.<someProperty> = <someValue>
...

# Vincular source al canal/es correspondiente
<Agent>.sources.<Source>.channels = <Channel1> <Channel2> ...

# Vincular sink al canal/es correspondiente
<Agent>.sinks.<Sink>.channel = <Channel1>
```

A continuación, se muestran algunos de los tipos de componentes que incorpora Apache Flume para la definición de *sources*, *channels* y *sinks*. En el caso de los *sources* algunos de los más utilizados son:

- **Avro:** Es un formato *open-source* para la seralización de datos, que utiliza una estructura de tipo JSON. El *source* Avro recibe los eventos de un cliente Avro a través de un número de puerto. Cuando es utilizado, se empareja con un *sink* de tipo Avro.
- **Thrift:** Permite la creación de servicios web, facilitando la recolección de datos a través de un cliente Thrift. Cuando se utiliza este *source* se debe emparejar con un *sink* también de tipo Thrift.
- **Exec:** Permite ejecutar un comando Unix, que ejecutará la funcionalidad del agente. Este proceso producirá los eventos en la salida estándar. Si por algún motivo el proceso se finaliza, el *source* también lo hace, dejando de producir datos.
- **Netcat:** De similar funcionalidad al comando netcat de Unix, convierte cada línea de texto en un evento de Flume, mediante la conexión establecida en un puerto especificado, en el que escucha los datos suministrados, que deben ir separados por nuevas líneas, convirtiendo cada una de ellas en un evento Flume y enviándolo al canal correspondiente.
- **Http:** Permite recolectar eventos a través de los métodos GET y POST. De esta forma, se convierten las peticiones hTTP en eventos de Flume y se envían al canal en una única transacción incrementando la eficiencia del mismo. Por otra parte, los eventos enviados en una petición POST se asumen como un lote y se insertan en el canal en una sola transacción.
- **Spooling directory:** Permite ingestar datos colocando los archivos que van a ser ingeridos dentro de un directorio llamado *spooling directory*. De esta forma, se analizan los datos de los nuevos archivos a medida que van apareciendo y, cuando son leídos completamente por el canal, se indica la escritura en el directorio y la finalización.

Respecto a los canales, los más utilizados son:

- **Memory Channel:** También conocido como canal de memoria, es una cola de memoria sobre la que los *sources* escriben eventos en su cola (*tail*) y los *sinks* leen eventos de su cabeza (*head*). El tamaño máximo de la cola puede definirse a priori por el desarrollador. Al almacenar los datos en memoria, el rendimiento es muy alto, aunque no es recomendable en aquellos flujos de datos donde la pérdida de eventos es crítica.
- **File Channel:** Similar al canal de memoria, este canal escrite todos los eventos en el disco, asegurando que cualquier evento será transmitido correctamente sin pérdida de datos. Este canal es altamente concurrente y permite manejar distintos *sources* y *sinks* de manera simultánea, por lo que es una buena alternativa para flujos de datos en los que no se pueden producir pérdidas de datos.

- **JDBC Channel:** Este canal almacena los eventos de Flume en un almacenamiento persistente, concretamente, en una base de datos. Cuando la recuperabilidad de los eventos y de la información transmitida es importante, este canal es una buena opción.

Finalmente, los sumideros (*sinks*) más utilizados son:

- **HDFS:** Permite escribir los eventos Flume en HDFS. Los archivos de texto y archivos de secuencia son los formatos permitidos en este tipo de *sink*. Para utilizar este tipo de sumidero, es necesario instalar y ejecutar Hadoop. A la hora de escribir los archivos, es posible incluir secuencias de escape en la ruta, lo que permite crear directorios que almacenen los eventos.
- **Logger sink:** Se trata de un tipo de *sink* muy utilizado para pruebas y/o depuración. Este sumidero permite registrar eventos a nivel de información y, al contrario que el resto, no necesita configuración adicional.
- **Avro sink:** Emparejado con el *source* Avro correspondiente, convierte a formato Avro los eventos y los envía al puerto especificado en la configuración. También es posible configurar el tamaño de lote para tomar los eventos de Flume en lotes de dicho tamaño.
- **Thrift sink:** Emparejado con el correspondiente *source* Thrift, funciona de manera similar al sumidero Avro, solo que a través de Thrift.
- **Custom sink:** Se trata de un tipo de sumidero que puede ser implementado a medida por el desarrollador, incluyendo la clase del *sink* personalizado así como las dependencias que pudieran darse en el classpath a la hora de ejecutar el agente Flume que haga uso de este tipo de *sink*.

¡Hola Mundo!. El primer agente Flume

A continuación, se muestra cómo proceder a la configuración de un agente básico Flume. La configuración y ejecución de este agente permitirá tener una toma de contacto con Apache Flume y servirá como guía para la definición de agentes posteriormente.

El agente creado define un único *source* de tipo **netcat.** Esto es así, ya que lo que se pretende es que, una vez se ejecute el agente Flume, el texto que se vaya introduciendo en cada una de las líneas escritas a través de la terminal de comandos se conviertan en eventos Flume que se transmitan por un canal y se muestren en otra terminal. Para ello, la configuración del *source* debe incluir la conexión que ha de realizarse en local en un puerto especificado. Una vez definido el *source*, se define un único *channel* de tipo ***memory*.** En este canal se configuran las propiedades de capacidad y capacidad de transacción con dos valores que definen, respectivamente, el número máximo de eventos que el canal puede almacenar y el número máximo de eventos que el canal puede tomar del *source* o enviar al *sink* en una

única transacción. Finalmente, se define un *sink* de tipo **logger,** ya que el agente que se está implementando es un agente de pruebas que mostrará los eventos en una terminal a modo de log. El listado 7.2 muestra el archivo de configuración de dicho agente.

Listado 7.2: Configuración del Agente Hola Mundo

```
# Configuración del Agente "Hola Mundo"

# Definición de componentes del agente
a1.sources  = r1
a1.sinks    = k1
a1.channels = c1

# Configuración de propiedades del source
a1.sources.r1.type = netcat
a1.sources.r1.bind = localhost
a1.sources.r1.port = 44444

# Configuración de propiedades del canal
a1.channels.c1.type = memory
a1.channels.c1.capacity = 1000
a1.channels.c1.transactionCapacity = 100

# Configuración de porpiedades del sink
a1.sinks.k1.type = logger

# Vincular source y sink al canal creado
a1.sources.r1.channels = c1
a1.sinks.k1.channel = c1
```

Para ejecutar el agente anterior, se deberá abrir una terminal y ejecutar el comando flume-ng, que permite ejecutar un agente Flume.

```
flume-ng agent -n a1 -f /Users/grupomat/Desktop/ejemplo.conf -Dflume.root.logger=INFO,console –Xmx512m
```

La opción **-n** permite especificar qué agente se ejecutará, ya que dentro de un mismo archivo de configuración se pueden definir más de un agente. La opción **-f** especifica la ruta del archivo de configuración. La opción **-Dflume** permite configurar la salida estándar para que muestre el log de los eventos registrados y la opción **-Xmx512m** permite aumentar la memoria del canal para evitar desbordamientos (no es necesaria, salvo que la ejecución del agente arroje una excepción de problemas de memoria).

Monitorización de archivos

En este apartado se mostrará la configuración de un agente Flume que permita **monitorizar los cambios producidos en un archivo.** De esta forma, una vez que el agente se ejecute, cada cambio que se produzca en el archivo y se añadan nuevos datos, los cambios incorporados se incorporarán a un evento Flume que se mostrará por terminal.

La configuración del agente en este caso utiliza un *source* de tipo **exec**, ya que la funcionalidad del agente se corresponde con la ejecución de un comando Unix (*tail*). Para dicho *source* se han configurado las propiedades *command* y *shell* que especifican, respectivamente, el comando a ejecutar (en este caso tail -f y a continuación la ruta al archivo que se pretende monitorizar) y la invocación a la ruta donde se almacena el comando, opción que es necesario especificar cuando la funcionalidad será la ejecución de un comando shell. A continuación, se define y configura un canal de tipo **memoria,** aunque en este caso no se ha configurado ninguna propiedad, y un *sink* de tipo **logger,** ya que los eventos se mostrarán por el terminal. El listado muestra la configuración de este agente.

Listado 7.3: Configuración de un agente para la monitorización de archivos

```
# Configuración del Agente para monitorización de archivos

#Definición del agente: nombre y componentes
Ag_Monitor.sources  = s1
Ag_Monitor.sinks    = k1
Ag_Monitor.channels = c1

#Definición de propiedades Source
Ag_Monitor.sources.s1.type    = exec
Ag_Monitor.sources.s1.command = tail -F /tmp/log.txt
Ag_Monitor.sources.s1.shell   = /bin/bash -c

# Configuración de propiedades del canal
a1.channels.c1.type = memory

# Configuración de propiedades del sink
a1.sinks.k1.type = logger

# Vinculación de source y sink al canal creado
Ag_Monitor.sources.s1.channels = c1
a1.sinks.k1.channel            = c1
```

7.2.4. Ingesta de datos en HDFS

Tal y como se ha visto anteriormente, el/los *sinks* de un agente Flume envían los eventos leídos del canal bien a otro agente (cuando hay un *pipeline* de agentes) o bien a un repositorio terminal, que normalmente suele ser HDFS o HBase. En los ejemplos vistos anteriormente, los eventos registrados se enviaban al **terminal.** Esto puede ser útil en algunas aplicaciones pero presenta una serie de **inconvenien-**

tes: ¿qué ocurre si se llena el canal y la memoria se desborda?, en ese caso, ¿cómo puede Flume informar a la aplicación en cuestión de que un evento no ha sido enviado?. En los ejemplos vistos anteriormente, por tanto, no hay una garantía de que los eventos se hayan recibido.

Por todo lo anterior, lo habitual para resolver los problemas anteriores es la **escritura de los eventos en un repositorio distribuido**, como puede ser HDFS o HBase. A continuación, se muestra un ejemplo de configuración de agente Flume en el que los eventos se escriben en HDFS.

Monitorización de un directorio

El primer ejemplo de agente que se mostrará en este apartado permite monitorizar los cambios producidos en un directorio. Este agente define un *source* de tipo *spooldir*. La configuración de dicho *source* incluye la definición de las propiedades que indican el directorio que se monitorizará, así como dos propiedades que indican que cuando se transmite el archivo, el nombre del mismo se agrega antes de cada línea de datos al evento correspondiente. A continuación se define un canal de tipo memoria y, finalmente, se configuran las propiedades del *sink*. En este caso, se ha utilizado un *sink* de tipo HDFS, ya que se pretende escribir los datos en este sistema de archivos. Además, se han configurado las opciones: *path*, que indica la ruta al directorio dentro de HDFS donde se escribirán los eventos, incluyendo la expresión que permite crear un directorio con la fecha (año mes y día) en el que se escribe el evento, *fileprefix*, que especifica el prefijo que se incluirá en los archivos creados por Flume, *fileType* que especifica el tipo de archivo que se almacenará en HDFS y, por último, *uselocaltimestamp* que utilizará el reloj local para incluir la fecha y hora de modificación de los archivos. En el listado 7.4 se muestra el archivo de configuración de dicho agente.

Listado 7.4: Configuración de un agente para la monitorización de un directorio

```
# Configuración del Agente para monitorización de un directorio

#Definición del agente: nombre y componentes
Agente_Dir.sources  = s1
Agente_Dir.sinks    = k1
Agente_Dir.channels = c1

#Configuración de propiedades Source
Agente_Dir.sources.s1.type              = spooldir
Agente_Dir.sources.s1.spoolDir          = /tmp/logs
Agente_Dir.sources.s1.basenameHeader    = true
Agente_Dir.sources.s1.basenameHeaderKey = fileName

# Configuración de propiedades del canal
Agente_Dir.channels.c1.type    = memory

# Configuración de propiedades del sink
Agente_Dir.sinks.k1.type                    = hdfs
Agente_Dir.sinks.k1.hdfs.path               = /flume/events/%y-%m-%d/%H/
Agente_Dir.sinks.k1.hdfs.filePrefix         = %{fileName}
```

```
21 Agente_Dir.sinks.k1.hdfs.fileType          = DataStream
22 Agente_Dir.sinks.k1.hdfs.useLocalTimeStamp = true
23
24 #Vinculación de source y sink al canal creado
25 Agente_Dir.sources.s1.channels = c1
26 Agente_Dir.sinks.k1.channel    = c1
```

A la hora de ejecutar este agente, se utilizará el siguiente comando:

> flume-ng –n Agente_Dir –f /home/cloudera/Downloads/apache-flume-1.9.0-bin/examples/ejemplo_hdfs.conf

Para comprobar que el agente funciona correctamente, se puede copiar cualquier archivo al directorio que se pretende monitorizar. Para ello, se utilizará el comando:

> cp <Ruta archivo a copiar>/tmp/logs

Para comprobar si el archivo se ha cargado en el directorio HDFS especificado a tal efecto, se utiliza el comando -cat, aplicado sobre HDFS.

> hdfs –cat flume/events/21-10-15/11/log.txt.1554788567801