Flume

Orígenes de datos

Hoy en día existen multitud de fuentes de datos

- Servidores de ficheros, bbdds, apps de empresa, feeds de noticias, tfno. móvil, sensores
- Pero sobre todo RRSS

Sabemos que hay dos maneras de catalogar los datos en función del tiempo de espera hasta obtener resultados

- Batch (lotes)
- Real Time (tiempo real)

Carga de datos

Enfoque tradicional
- Casi obsoleto: trabajar con cintas

Un poco menos tradicional
- Copiar datos de un servidor a otro mediante comandos unix
  - Secury copy (scp)
  - ***scp source_file_name user@destination_host:folder***
  - Rsync
  - ***rsync [OPTION] … SRC [SRC] … [USER@]HOST:DEST***
  - Rcp
  - ***rcp source_file_name user@:destination_host:folder***

Automatizado
- Usando **Crontab**

Ejecuta *copiasemanal.sh* cada domingo a las 23:55

Hoy en día hay soluciones más eficientes

Flume: definición

- Apache Flume es un sistema distribuido, confiable y preparado para recopilar, agregar y mover de forma eficiente grandes cantidades de datos de diversas fuentes a un almacén de datos centralizado.

- Posee una arquitectura sencilla y flexible basada en flujos de datos en streaming.

- Es robusto y tolerante a fallos con múltiples mecanismos de configuración.

- Utiliza la transaccionalidad en la comunicación entre sus componentes.

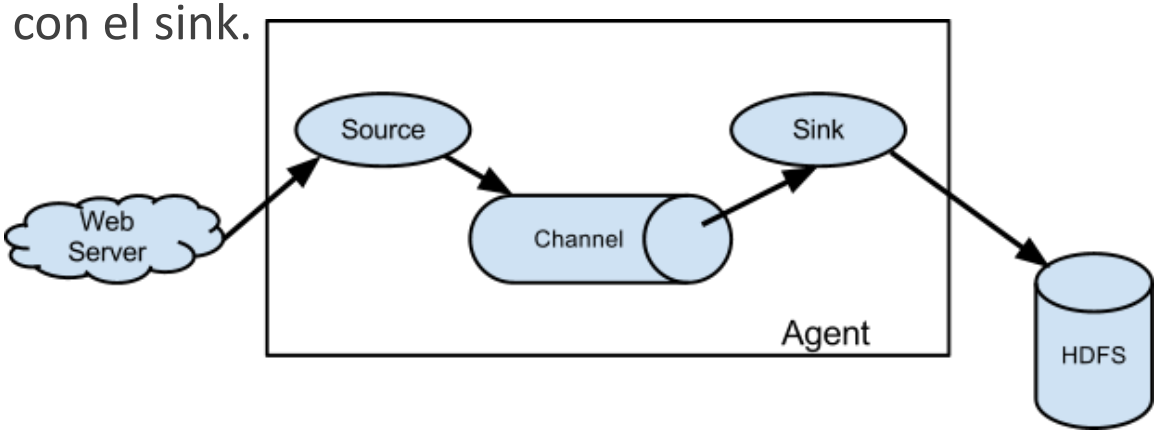
Flume: conceptos básicos

- **Flume** significa “Canal Artificial”
- **Log** se puede traducir, entre otras cosas, como “Tronco”
- La idea está clara

Flume: componentes

Flume se compone de los siguientes componentes principales:

- **Event**: es la unidad de dato que se va propagando a través de la arquitectura
- **Source**: es el origen de los datos
- **Sink**: es el destino de los datos
- **Channel**: es el buffer de almacenamiento intermedio que conecta el source con el sink.



Flume: Setup

- Flume funciona como un **agente** levantado en la máquina desde la que se quiere recoger la información.
- La configuración de este agente se realiza en un fichero local.
- Este fichero de configuración contiene las propiedades de cada source, channel y sink que vayamos a definir.
- Puede haber varios channels y sinks en un mismo fichero de configuración.
- Cada source, channel y sink tienen su nombre, tipo y un set de propiedades.
- Un agente, para funcionar correctamente, debe conocer cada componente y la forma en la que están conectados. A esto se le llama **flow** (flujo)**.**
- Para inicial un agente se utiliza un Shell script llamado *flume-ng* que está localizado en el directorio de instalación de flume. Necesitamos especificar el nombre del agente, el directorio de configuración y el archivo de configuración de esta manera:
  - $ bin/flume-ng agent -n $agent_name -c conf -f conf/flume-conf.properties.template

Flume: Ejemplo

```
# Definimos el Source, Channel y Sink

a1.sources = r1

a1.sinks = k1

a1.channels = c1

# Configuramos el Source

a1.sources.r1.type = netcat

a1.sources.r1.bind = localhost

a1.sources.r1.port = 44444
```

```
# Configuramos en sink
a1.sinks.k1.type = logger
# Configuramso el channel
a1.channels.c1.type = memory
a1.channels.c1.capacity = 1000
a1.channels.c1.transactionCapacity = 100
# Unimos source y el sink a través del channel
a1.sources.r1.channels = c1
a1.sinks.k1.channel = c1
```

Flume: opciones de configuración

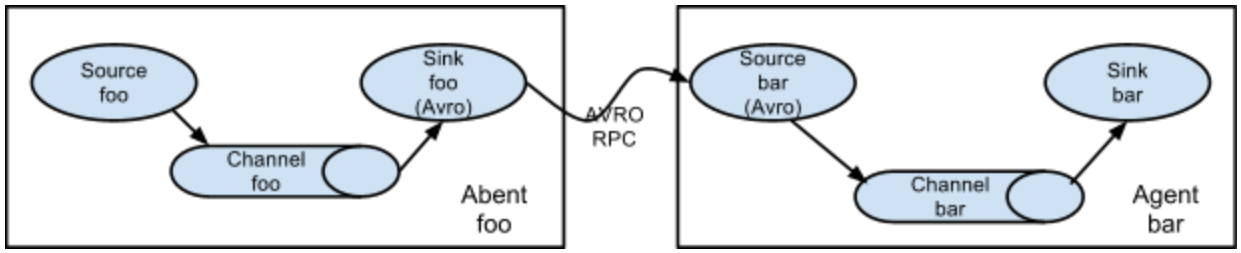
Un único flujo / agente

- Es el ejemplo que acabamos de mostrar
- Definimos el agente dentro del fichero “a1”
- Conectamos source y sink a través del channel
- Un source puede estar conectado a varios channels, pero un sink solo puede estár conectado a un channel.

Flume: opciones de configuración

Múltiples agentes

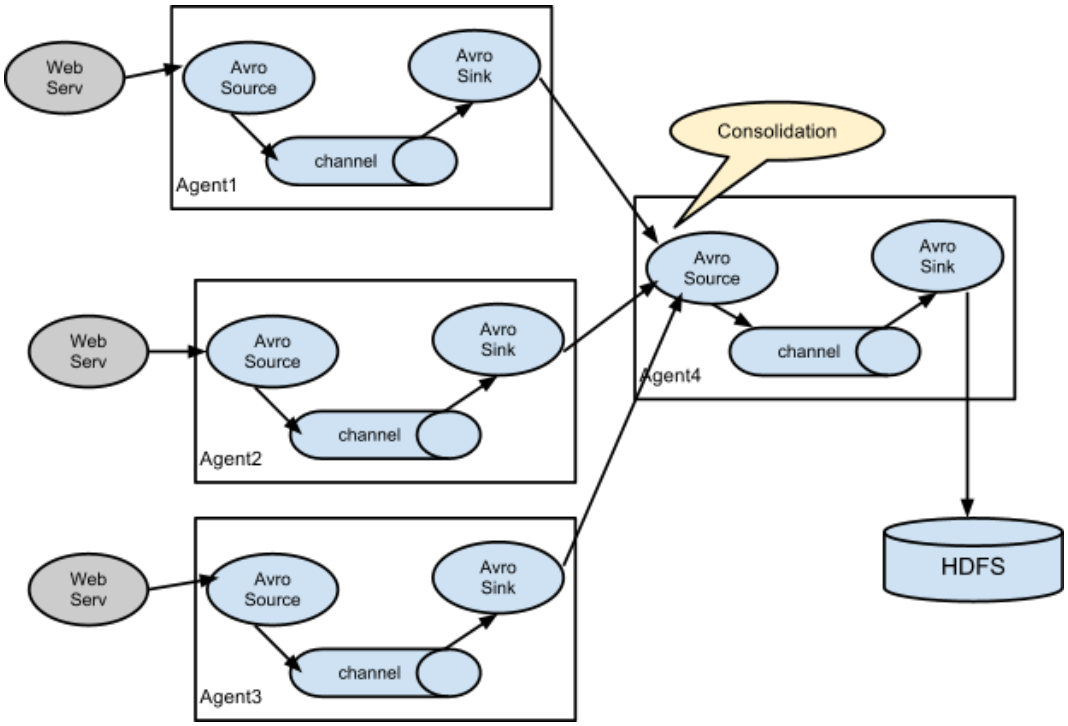
- Pare este tipo de configuración se necesita que el sink del agente previo y el source del actual tienen que ser de tipo AVRO (sistema de serialización de datos), con el sink apuntando al hostname y puerto del source



Flume: opciones de configuración

Múltiples agentes

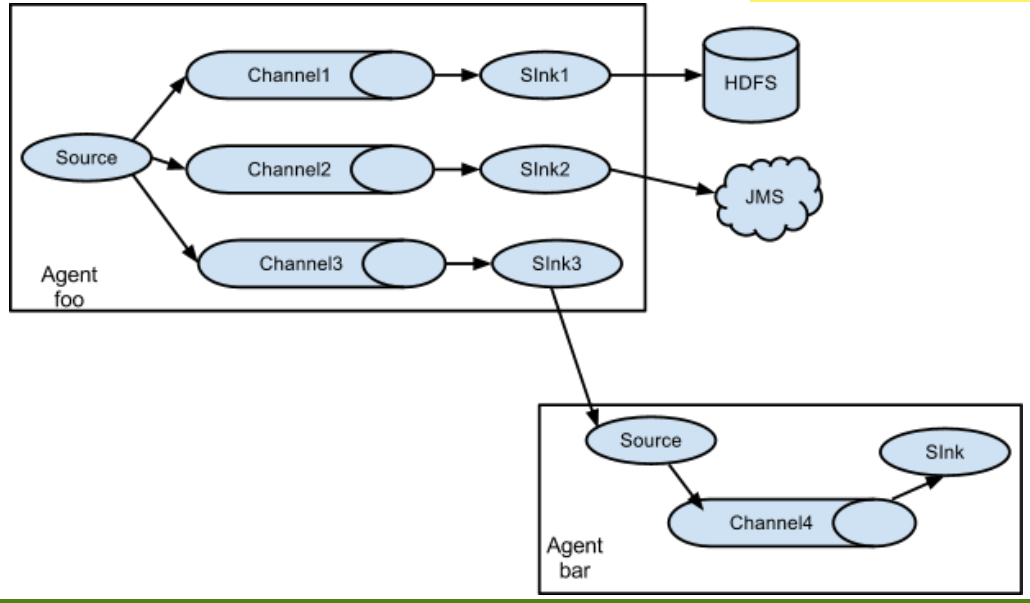
- Un caso típico de este tipo de configuración es la de multiples webs enviando logs a un sistema de almacenamiento como HDFS



Flume: opciones de configuración

Múltiples flows

- La lidea es multiplexar los eventos a uno o más destinatarios en función de una serie de condiciones indicadas en el fichero de configuración
- Existen dos opciones
  - Replica: donde todos los eventos se mandan sobre todos los canales
  - Multiplexación: donde los eventos se mandan sobre ciertos canales en función de las propiedades de selección



Flume: detalle configuración

Esquema de fichero de configuración

```
# properties for sources
<Agent>.sources.<Source>.<someProperty> = <someValue>
# properties for channels
<Agent>.channel.<Channel>.<someProperty> = <someValue>
# properties for sinks
<Agent>.sources.<Sink>.<someProperty> = <someValue>
```

```
# lista de sources, sinks and channels para el agente
agent_foo.sources = avro-appserver-src-1
agent_foo.sinks = hdfs-sink-1
agent_foo.channels = mem-channel-1

# configuración del canal para el source
agent_foo.sources.avro-appserver-src-1.channels = mem-channel-1

# configuración del canal para el sink
agent_foo.sinks.hdfs-sink-1.channel = mem-channel-1
```

Flume: detalle configuración

- Una de las propiedades más importantes que hay que configurar en Flume es la "**type**", con la cual indica el tipo de componente que es.

```
agent_foo.sources = avro-AppSrv-source
agent_foo.sinks = hdfs-Cluster1-sink
agent_foo.channels = mem-channel-1

# set channel for sources, sinks

# properties of avro-AppSrv-source
agent_foo.sources.avro-AppSrv-source.type = avro
agent_foo.sources.avro-AppSrv-source.bind = localhost
agent_foo.sources.avro-AppSrv-source.port = 10000

# properties of mem-channel-1
agent_foo.channels.mem-channel-1.type = memory
agent_foo.channels.mem-channel-1.capacity = 1000
agent_foo.channels.mem-channel-1.transactionCapacity = 100

# properties of hdfs-Cluster1-sink
agent_foo.sinks.hdfs-Cluster1-sink.type = hdfs
agent_foo.sinks.hdfs-Cluster1-sink.hdfs.path = hdfs://namenode/flume/webdata
```

Flume: detalle configuración

Ejemplo de dos sources conectados con dos sinks a través de dos channels

```
# sources, sinks and channels en el agente
agent_foo.sources = avro-AppSrv-source1 exec-tail-source2
agent_foo.sinks = hdfs-Cluster1-sink1 avro-forward-sink2
agent_foo.channels = mem-channel-1 file-channel-2

# configuracion flow #1
agent_foo.sources.avro-AppSrv-source1.channels = mem-channel-1
agent_foo.sinks.hdfs-Cluster1-sink1.channel = mem-channel-1

# configuracion flow #2
agent_foo.sources.exec-tail-source2.channels = file-channel-2
agent_foo.sinks.avro-forward-sink2.channel = file-channel-2
```

Flume: detalle configuración

Ejemplo de multiplexación de flows

```
# lista de sources, sinks and channels en el agente
agent_foo.sources = avro-AppSrv-source1
agent_foo.sinks = hdfs-Cluster1-sink1 avro-forward-sink2
agent_foo.channels = mem-channel-1 file-channel-2

# set el channel para el source
agent_foo.sources.avro-AppSrv-source1.channels = mem-channel-1 file-channel-2

# set channel para los sinks
agent_foo.sinks.hdfs-Cluster1-sink1.channel = mem-channel-1
agent_foo.sinks.avro-forward-sink2.channel = file-channel-2

# configuración del selector del channel
agent_foo.sources.avro-AppSrv-source1.selector.type = multiplexing
agent_foo.sources.avro-AppSrv-source1.selector.header = State
agent_foo.sources.avro-AppSrv-source1.selector.mapping.CA = mem-channel-1
agent_foo.sources.avro-AppSrv-source1.selector.mapping.AZ = file-channel-2
agent_foo.sources.avro-AppSrv-source1.selector.mapping.NY = mem-channel-1 file-channel-2
agent_foo.sources.avro-AppSrv-source1.selector.default = mem-channel-1
```

Flume: Sources

- Existen numerosos tipos de fuentes de datos soportados por Flume.
- Cada uno de ellos se puede parametrizar.
- Desde el punto de vista funcional existen dos tipos.
  - **EvenDriverSource**: son activos. Controlan cómo y cuándo los eventos son añadidos al channel. Responden al comportamiento “push”. El ejemplo más claro es el Avro source. Cuando recibe una RPC (Remote Procedure Call) con uno o mas eventos, inmediatamente los añade al channel.
  - **PollableSource**: son pasivos. No hay forma de saber cuándo llega un nuevo dato que ser enviado a través del flow, por lo que Flume tiene que “tirar de ellos: pull” cada cierto tiempo. Son fáciles de configurar pero proveen menos control.

Flume: Sources predefinidos

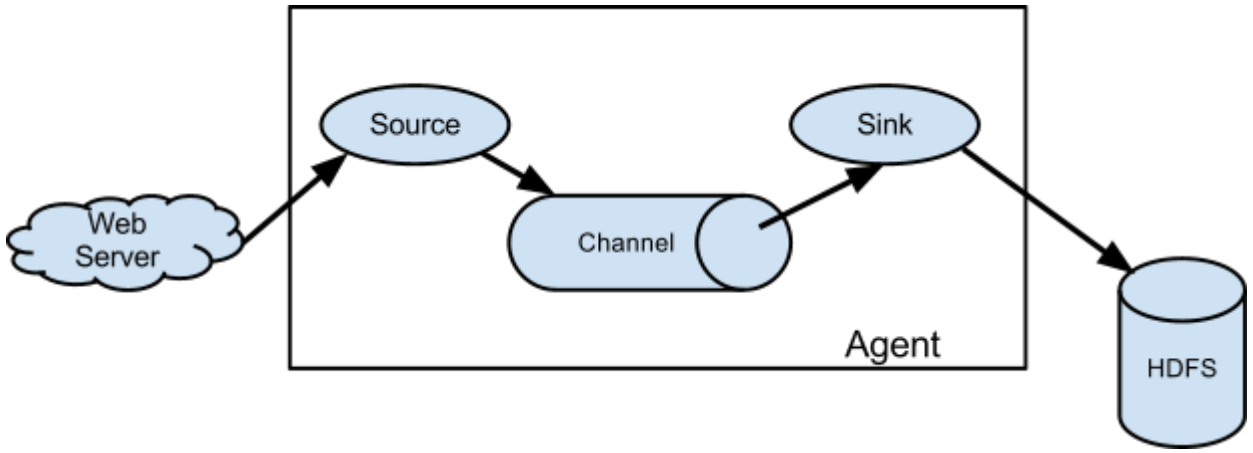
- **Avro**: escucha de un puerto Avro y recibe eventos desde streams de clientes externos Avro.
- **Thrift** (framework desarrollo servicios inter-lenguaje): escucha en un puerto Thrift y recibe eventos desde streams en clientes Thrift externos. Puede ser autenticado utilizando Kerberos
- **Exec**: ejecuta comandos Unix al inicializar la fuente. Si el comando continuo (cat, tail) se irán recogiendo eventos según un threshold (número de líneas, tiempo, …). Si el comando es concreto (date, ls) únicamente se recogerá un evento.
- **JMS**: se leen mensajes de una cola / topic. Flume indica que debería funcionar con cualquier proveedor pero que solo ha sido probado con ActiveMQ.
- **Spooling directory**: el source lee desde ficheros que han sido movidos a un directorio (spool directory). El source se va a encargar de ir leyendo el fichero e ir enviando el contenido / evento de dicho fichero al channel asociado. Una vez enviado, el fichero se puede marcar como “ya leído”. Se trata de una fuente confiable.

Flume: Sources predefinidos

- **Twitter**: conecta a la API de streaming de Twitter con las credenciales de tu usuario. Se accede al Firehose de Twitter (1% de los tweets)
- **Kafka:** se leen topics de mensajes almacenados en Kafka.
- **Netcat:** se lee de un puerto. Cada línea de texto que se escribe se transforma en un evento.
- **Sequence Generator:** generador secuencial de eventos.
- **Syslog:** tienen como origen de datos los syslog de la máquina (UDP/TCP).
- **Http:** se aceptan eventos desde una petición HTTP Get o Post. Se puede definir un handler que trate el payload de la petición.
- **Stress:** simula un test de carga (eventos exitosos/fallidos).
- **Legacy:** permite recibir eventos de agentes Flume más antiguos.
- **Custom**: implementación ad-hoc. Se tiene que implementar una clase Java propia implementando las interfaces base.
- **Scribe:** sistema de ingesta propia que se puede querer utilizar junto a Flume.

Flume: Sinks

Se encargan de extraer evento de un channel y almacenarlos en un sistema externo o enviarlos al siguiente agente del flow.



Flume: Sinks predefinidos

- **HDFS**: almacena eventos en el sistema de ficheros de Hadoop. Permite almacenar ficheros en formato text y sequenceFile. Permite compresión.
- **Hive**: almacena eventos con formato texto o Json en tablas o particiones de Hive. Utiliza la transaccionalidad de Hive. (**Not production ready**)
- **Logger**: utiliza el nivel de log INFO para guardar los eventos
- **Avro**: almacenados sobre un host/port de Avro
- **Thrift**: almacenados sobre un host/port de Thrift (framework multilenguaje)
- **IRC**: utiliza el sistema de IRC channels
- **FileRoll**: almacena eventos en el sistema de ficheros local.
- **Null**: descarta los eventos.
- **Hbase**: almacena eventos en una base de datos Hbase. Se necesita utilizar un serializer de Hbase específico. Se puede tener autenticación mediante Kerberos.
- **MorphlineSolr**: transforma los eventos a partir de una configuración y los almacena en un motor de búsqueda Solr
- **ElasticSearch**: almacena los eventos en ElasticSearch
- **Kite Dataset**: permite almacenar eventos en Kite (hadoop layer for speedup dev.)
- **Kafka**: se puede publicar los eventos en un topic de Kafka
- **Custom**: se pueden construir sumideros específicos.

Flume: Channels predefinidos

- **Memoria**: Los eventos se almacenan en una cola de memoria de tamaño predefinido. Útil cuando se queremos un throughput alto y podemos recuperar los eventos.
- **JDBC**. Los eventos son persistidos sobre una base de datos. Se necesita definir el driver, la url de conexión, …
- **Kafka**: los eventos son persistidos en un cluster de Kafka. Nos proporciona alta disponibilidad y replicación.
- **File**: los eventos son guardados en un fichero en el local system.
- **Spillable Memory**: Los eventos se guardan por defecto en una cola en memoria. Si el número de eventos almacenados sobrecarga la cola, éstos se pueden guardar en disco.
- **Pseudo Transaction**: utilizado para testing.
- **Custom Channel**: Implementación propia.

Flume: Interceptores

- Flume permite modificar eventos en caliente mediante interceptores.
- Podríamos decir que los interceptores actúan como si fuera una pequeña **ETL** con una serie de funciones predefinidas.
- Implementan la interfaz org.apache.flume.interceptor.Interceptor.
- Si hay varios interceptores, estos se ejecutan en el orden en que se hayan definido
- Los eventos irán pasando de uno a otro de manera encadenada.

Flume: Interceptores predefinidos

- **Timestamp**: inserta un timestamp en las cabecera de los eventos.
- **Host**: añade el host o la IP al evento.
- **Static**: añade una cabecera ‘fija’ a los eventos.
- **UUID**: añade un identificador único a la cabecera.
- **Morphline**: permite hacer un transformación predefinida en un fichero de configuración
- **Search&Replace**: busca una cadena en el evento y la reemplaza por otra cadena.
- **Regex**: utiliza una expresión regular sobre la que matchear.

Flume: ejercicios

Ver hoja de ejercicios.