Flume

刘磊

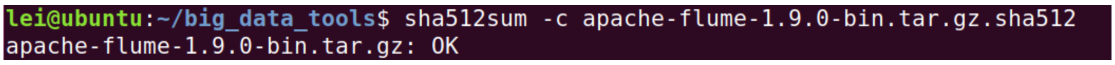
2020 年 11 月

目录

1. Flume 安装

将 apache-flume-1.9.0-bin.tar.gz 和 apache-flume-1.9.0-bin.tar.gz.sha512 两个文件放到~/big_data_tools/下，检验安装包 apache-flume-1.9.0-bin.tar.gz 的完整性

```
cd ~/big_data_tools
sha512sum -c apache-flume-1.9.0-bin.tar.gz.sha512
```



```
lei@ubuntu:~/big_data_tools$ sha512sum -c apache-flume-1.9.0-bin.tar.gz.sha512
apache-flume-1.9.0-bin.tar.gz: OK
```

复制 apache-flume-1.9.0-bin.tar.gz 到/apps 目录

```
cp ~/big_data_tools/apache-flume-1.9.0-bin.tar.gz /apps
```

解压并重命名

```
tar zxvf apache-flume-1.9.0-bin.tar.gz
mv apache-flume-1.9.0-bin flume
```

删除安装包

```
rm apache-flume-1.9.0-bin.tar.gz
```

设置环境变量

```
vim ~/.bashrc
```

将 Flume 的 bin 目录，添加到用户环境变量 PATH 中

```
# Flume
export FLUME_HOME=/apps/flume
export PATH=$FLUME_HOME/bin:$PATH
```

```
# Flume
export FLUME_HOME=/apps/flume
export PATH=$FLUME_HOME/bin:$PATH
```

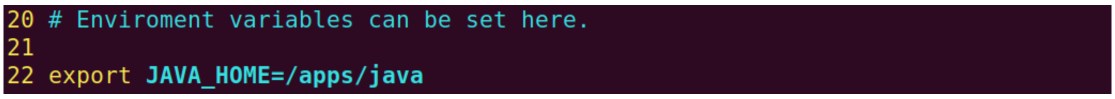
执行 source 命令，使 Flume 环境变量生效。

```
source ~/.bashrc
```

修改配置文件

```
cd /apps/flume/conf
cp flume-env.sh.template flume-env.sh
```

打开 flume-env.sh，去掉 22 行的注释并修改为 Java 的安装路径



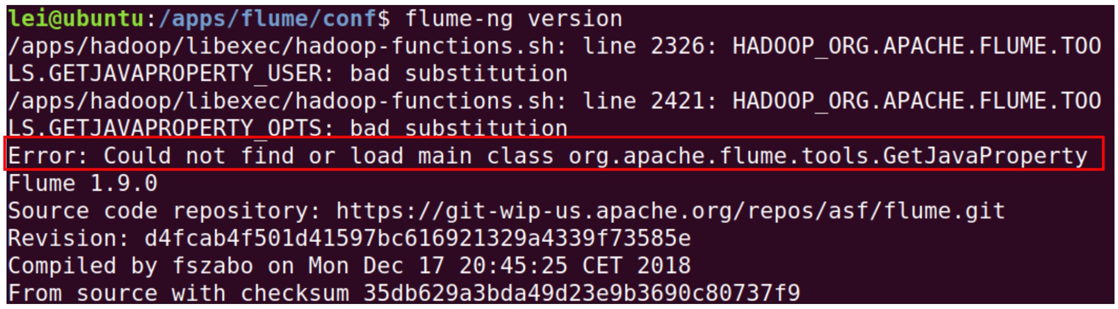
```
# Enviroment variables can be set here.

export JAVA_HOME=/apps/java
```

在终端中输入以下命令进行测试

```
flume-ng version
```

报错：Error: Could not find or load main class org.apache.flume.tools.

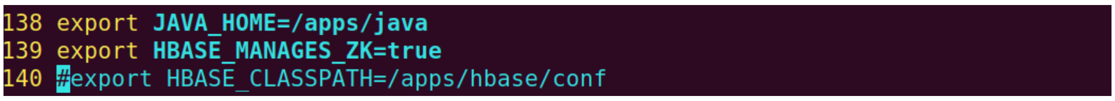
GetJavaProperty



```
lei@ubuntu:/apps/flume/conf$ flume-ng version
/apps/hadoop/libexec/hadoop-functions.sh: line 2326: HADOOP_ORG.APACHE.FLUME.TOO
LS.GETJAVAPROPERTY_USER: bad substitution
/apps/hadoop/libexec/hadoop-functions.sh: line 2421: HADOOP_ORG.APACHE.FLUME.TOO
LS.GETJAVAPROPERTY_OPTS: bad substitution
Error: Could not find or load main class org.apache.flume.tools.GetJavaProperty
Flume 1.9.0
Source code repository: https://git-wip-us.apache.org/repos/asf/flume.git
Revision: d4fcab4f501d41597bc616921329a4339f73585e
Compiled by fszabo on Mon Dec 17 20:45:25 CET 2018
From source with checksum 35db629a3bda49d23e9b3690c80737f9
```

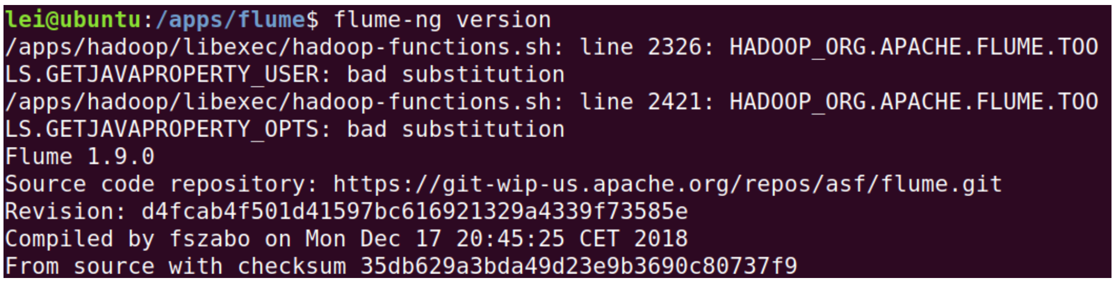
解决方法：将 /apps/hbase/conf/hbase-env.sh 文件中对环境变量 HBASE_CLASSPATH

的设置注释掉。



```
export JAVA_HOME=/apps/java
export HBASE_MANAGES_ZK=true
#export HBASE_CLASSPATH=/apps/hbase/conf
```

注释后重新执行 flume-ng version，可以看到不再报错，显示出 Flume 版本号。



```
lei@ubuntu:/apps/flume$ flume-ng version
/apps/hadoop/libexec/hadoop-functions.sh: line 2326: HADOOP_ORG.APACHE.FLUME.TOO
LS.GETJAVAPROPERTY_USER: bad substitution
/apps/hadoop/libexec/hadoop-functions.sh: line 2421: HADOOP_ORG.APACHE.FLUME.TOO
LS.GETJAVAPROPERTY_OPTS: bad substitution
Flume 1.9.0
Source code repository: https://git-wip-us.apache.org/repos/asf/flume.git
Revision: d4fcab4f501d41597bc616921329a4339f73585e
Compiled by fszabo on Mon Dec 17 20:45:25 CET 2018
From source with checksum 35db629a3bda49d23e9b3690c80737f9
```

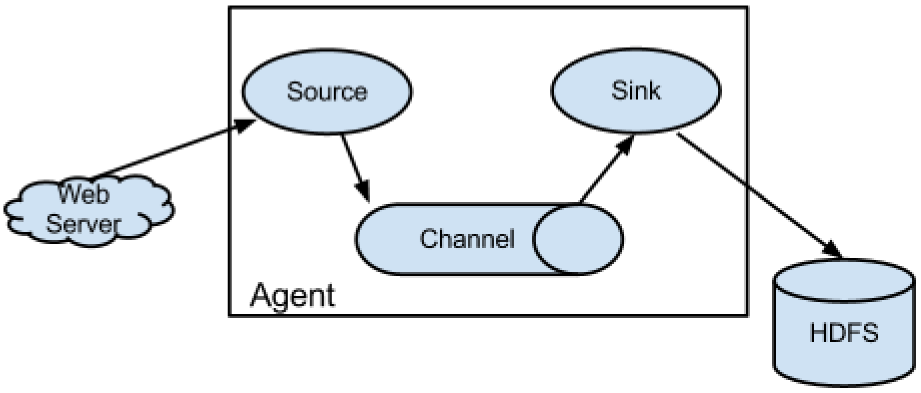
2. Flume 使用

2.1 节介绍如何写一个配置文件，后面的小节介绍如何使用不同的 source 和 sink 类型

并给出实际的例子。

2.1 如何写配置文件

定义一个流程



流程的是定义在配置文件中的。在配置文件中需要把 source 和 sink 用 channel 连接起来。 首先列出一个 agent 所有的 source, sink 和 channel，然后为 source 和 sink 指定 channel。一个 source 可以指定多个 channel，但是一个 sink 只能指定一个 channel。格式如下：

```
# list the sources, sinks and channels for the agent
<Agent>.sources = <Source>
<Agent>.sinks = <Sink>
<Agent>.channels = <Channel1> <Channel2>

# set channel for source
<Agent>.sources.<Source>.channels = <Channel1> <Channel2> ...

# set channel for sink
<Agent>.sinks.<Sink>.channel = <Channel1>
```

设置组件

流程定义好以后需要为每个 source,sink 和 channel 设置属性。

```
# properties for sources
<Agent>.sources.<Source>.<someProperty> = <someValue>

# properties for channels
<Agent>.channel.<Channel>.<someProperty> = <someValue>
```

```
# properties for sinks
<Agent>.sources.<Sink>.<someProperty> = <someValue>
```

2.2 Avro Source

监听 Avro 端口来接受来自外部 Avro 客户端的 event 流。当与另一个 Flume agent 内置的 Avro Sink 配对时，它可以创建分层收集结构。

Avro 是一个数据序列化系统，设计用于支持大批量数据交换的应用。它的主要特点是支持二进制序列化方式，可以便捷，快速地处理大量数据；动态语言友好，Avro 提供的机制使动态语言可以方便地处理 Avro 数据。

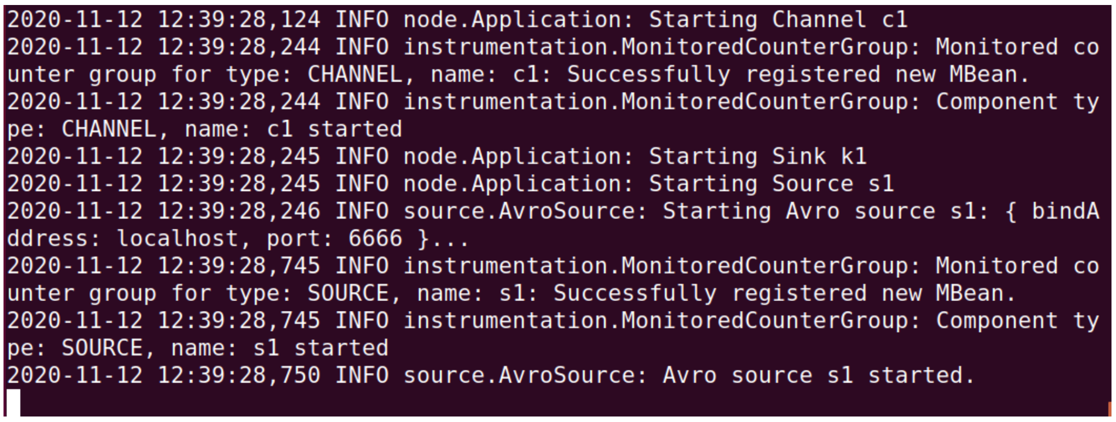
在/apps/flume/conf 目录下新建文件 single_avro.conf，写入如下内容

```
#配置一个 agent，agent 的名称可以自定义（如 a1）
#指定 agent 的 sources（如 s1）、sinks（如 k1）、channels（如 c1）
#分别指定 agent 的 sources，sinks,channels 的名称 名称可以自定义
a1.sources = s1
a1.sinks = k1
a1.channels = c1

#为 sources 和 sinks 绑定 channels
a1.sources.s1.channels = c1
a1.sinks.k1.channel = c1

#配置 source
a1.sources.s1.channels = c1
a1.sources.s1.type = avro
a1.sources.s1.bind = localhost
a1.sources.s1.port = 6666

#配置 channels
a1.channels.c1.type = memory

#配置 sinks
a1.sinks.k1.channel = c1
a1.sinks.k1.type = logger
```

通过下面的命令启动 flume

```
flume-ng agent --conf conf --conf-file \
/apps/flume/conf/single_avro.conf --name a1 \
-Dflume.root.logger=DEBUG,console \
-Dorg.apache.flume.log.printconfig=true \
-Dorg.apache.flume.log.rawdata=true
```

可以看到各组件已经启动



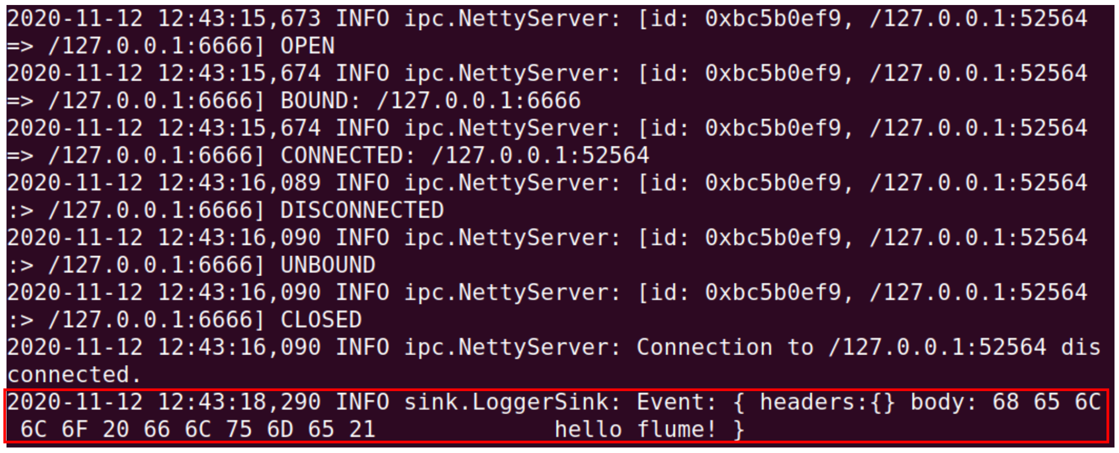
```
2020-11-12 12:39:28,124 INFO node.Application: Starting Channel c1
2020-11-12 12:39:28,244 INFO instrumentation.MonitoredCounterGroup: Monitored co
unter group for type: CHANNEL, name: c1: Successfully registered new MBean.
2020-11-12 12:39:28,244 INFO instrumentation.MonitoredCounterGroup: Component ty
pe: CHANNEL, name: c1 started
2020-11-12 12:39:28,245 INFO node.Application: Starting Sink k1
2020-11-12 12:39:28,245 INFO node.Application: Starting Source s1
2020-11-12 12:39:28,246 INFO source.AvroSource: Starting Avro source s1: { bindA
ddress: localhost, port: 6666 }...
2020-11-12 12:39:28,745 INFO instrumentation.MonitoredCounterGroup: Monitored co
unter group for type: SOURCE, name: s1: Successfully registered new MBean.
2020-11-12 12:39:28,745 INFO instrumentation.MonitoredCounterGroup: Component ty
pe: SOURCE, name: s1 started
2020-11-12 12:39:28,750 INFO source.AvroSource: Avro source s1 started.
```

另外打开一个终端，通过 Flume 提供的 avro 客户端向指定机器指定端口发送日志信息：

```
flume-ng avro-client -c ~/apps/flume/conf -H localhost -p 6666 -F
/data/hello.txt
```

这里是将 hello.txt 的内容发送给 Flume，hello.txt 需要提前创建。在 Flume 的启动窗口

可以看到 Event 中的信息就是 hello.txt 的内容 hello flume!



```
2020-11-12 12:43:15,673 INFO ipc.NettyServer: [id: 0xbc5b0ef9, /127.0.0.1:52564
=> /127.0.0.1:6666] OPEN
2020-11-12 12:43:15,674 INFO ipc.NettyServer: [id: 0xbc5b0ef9, /127.0.0.1:52564
=> /127.0.0.1:6666] BOUND: /127.0.0.1:6666
2020-11-12 12:43:15,674 INFO ipc.NettyServer: [id: 0xbc5b0ef9, /127.0.0.1:52564
=> /127.0.0.1:6666] CONNECTED: /127.0.0.1:52564
2020-11-12 12:43:16,089 INFO ipc.NettyServer: [id: 0xbc5b0ef9, /127.0.0.1:52564
:> /127.0.0.1:6666] DISCONNECTED
2020-11-12 12:43:16,090 INFO ipc.NettyServer: [id: 0xbc5b0ef9, /127.0.0.1:52564
:> /127.0.0.1:6666] UNBOUND
2020-11-12 12:43:16,090 INFO ipc.NettyServer: [id: 0xbc5b0ef9, /127.0.0.1:52564
:> /127.0.0.1:6666] CLOSED
2020-11-12 12:43:16,090 INFO ipc.NettyServer: Connection to /127.0.0.1:52564 dis
connected.
2020-11-12 12:43:18,290 INFO sink.LoggerSink: Event: { headers:{} body: 68 65 6C
 6C 6F 20 66 6C 75 6D 65 21             hello flume! }
```

2.3 Exec Source

ExecSource 的配置就是设定一个 Linux 命令，然后通过这个命令不断输出数据。如果进程退出，ExecSource 也一起退出，不会产生进一步的数据。

保存以下内容到/apps/flume/conf/case_exec.conf

```
#Name the components on this agent
a1.sources= s1
a1.sinks= k1
a1.channels= c1

# Bind the source and sink to the channel
a1.sources.s1.channels = c1
a1.sinks.k1.channel = c1

#配置 sources
a1.sources.s1.type = exec
a1.sources.s1.command = tail -F /home/lei/logs/test.log

#配置 sinks
a1.sinks.k1.type= logger

#配置 channel
a1.channels.c1.type= memory
a1.channels.c1.capacity = 1000
a1.channels.c1.transactionCapacity = 100
```

这里我们用 tail –F 命令一直读日志的尾部。

注:

a1.channels.c1.capacity：channel 中能够存储的 event 的最大个数。

a1.channels.c1.transactionCapacity：chanel 每次从 source 取或每次传给 sink 的 event 的最大个数。

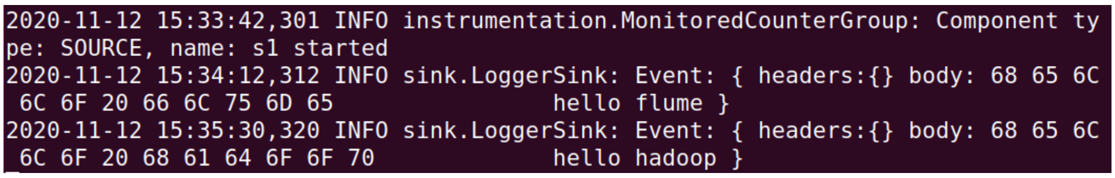
启动 flume

```
flume-ng agent --conf conf --conf-file \
/apps/flume/conf/case_exec.conf --name a1 \
-Dflume.root.logger=DEBUG,console \
-Dorg.apache.flume.log.printconfig=true \
-Dorg.apache.flume.log.rawdata=true
```

另外打开一个终端，向日志文件追加一些内容

```
mkdir ~/logs
echo "hello flume" >> ~/logs/test.log
echo "hello hadoop" >> ~/logs/test.log
```

在 Flume 运行界面可以看到，已经接收到添加的文本



```
2020-11-12 15:33:42,301 INFO instrumentation.MonitoredCounterGroup: Component ty
pe: SOURCE, name: s1 started
2020-11-12 15:34:12,312 INFO sink.LoggerSink: Event: { headers:{} body: 68 65 6C
 6C 6F 20 66 6C 75 6D 65                hello flume }
2020-11-12 15:35:30,320 INFO sink.LoggerSink: Event: { headers:{} body: 68 65 6C
 6C 6F 20 68 61 64 6F 6F 70             hello hadoop }
```

2.4 Spooling Directory Source

Spooling Directory Source 监视设置的目录，并将解析新文件的出现。Spool Source 有 2 个注意地方，第一个是拷贝到 spool 目录下的文件不可以再打开编辑，第二个是 spool 目录下不可包含相应的子目录。这个主要用途作为对日志的准实时监控。

将下面的内容添加到文件/apps/flume/conf/case_spool.conf

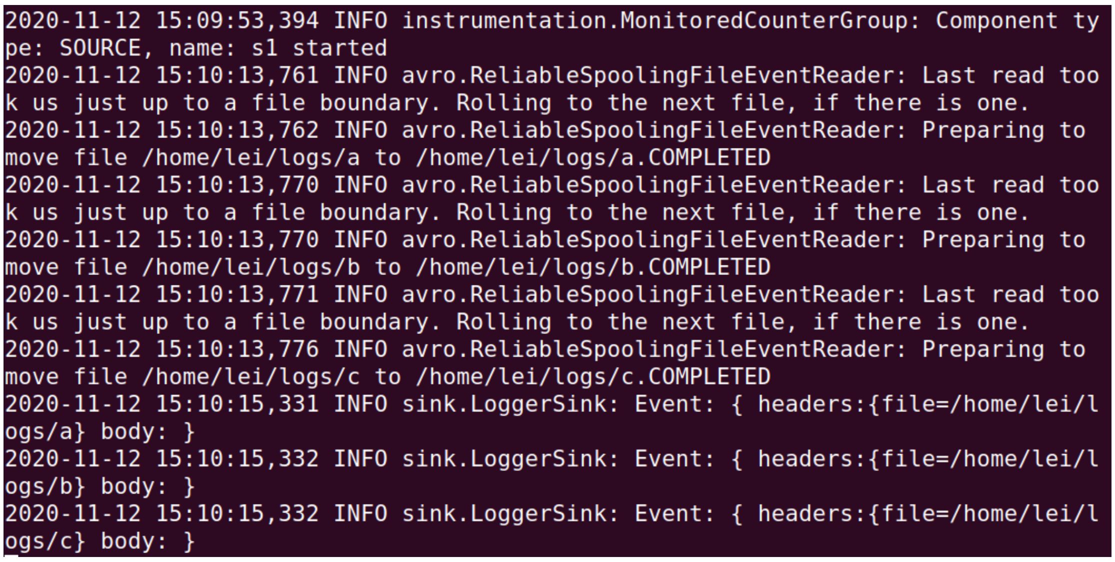
```
# Name the components on this agent
a1.sources = s1
a1.sinks = k1
a1.channels = c1

# Bind the source and sink to the channel
a1.sources.s1.channels =c1
a1.sinks.k1.channel = c1

# Describe/configure the source
a1.sources.s1.type =spooldir
```

```
a1.sources.s1.spoolDir =/home/lei/logs
a1.sources.s1.fileHeader= true

# Describe the sink
a1.sinks.k1.type = logger

# Use a channel which buffers events in memory
a1.channels.c1.type = memory
```

启动 Flume

```
flume-ng agent --conf conf --conf-file \
/apps/flume/conf/case_spool.conf --name a1 \
-Dflume.root.logger=INFO,console
```

另外打开一个终端，在监控目录/home/lei/logs 中新建三个文件 a,b,c，在 Flume 运行界面会显示如下信息



```
2020-11-12 15:09:53,394 INFO instrumentation.MonitoredCounterGroup: Component ty
pe: SOURCE, name: s1 started
2020-11-12 15:10:13,761 INFO avro.ReliableSpoolingFileEventReader: Last read too
k us just up to a file boundary. Rolling to the next file, if there is one.
2020-11-12 15:10:13,762 INFO avro.ReliableSpoolingFileEventReader: Preparing to
move file /home/lei/logs/a to /home/lei/logs/a.COMPLETED
2020-11-12 15:10:13,770 INFO avro.ReliableSpoolingFileEventReader: Last read too
k us just up to a file boundary. Rolling to the next file, if there is one.
2020-11-12 15:10:13,770 INFO avro.ReliableSpoolingFileEventReader: Preparing to
move file /home/lei/logs/b to /home/lei/logs/b.COMPLETED
2020-11-12 15:10:13,771 INFO avro.ReliableSpoolingFileEventReader: Last read too
k us just up to a file boundary. Rolling to the next file, if there is one.
2020-11-12 15:10:13,776 INFO avro.ReliableSpoolingFileEventReader: Preparing to
move file /home/lei/logs/c to /home/lei/logs/c.COMPLETED
2020-11-12 15:10:15,331 INFO sink.LoggerSink: Event: { headers:{file=/home/lei/l
ogs/a} body: }
2020-11-12 15:10:15,332 INFO sink.LoggerSink: Event: { headers:{file=/home/lei/l
ogs/b} body: }
2020-11-12 15:10:15,332 INFO sink.LoggerSink: Event: { headers:{file=/home/lei/l
ogs/c} body: }
```

2.5 NetCat Source

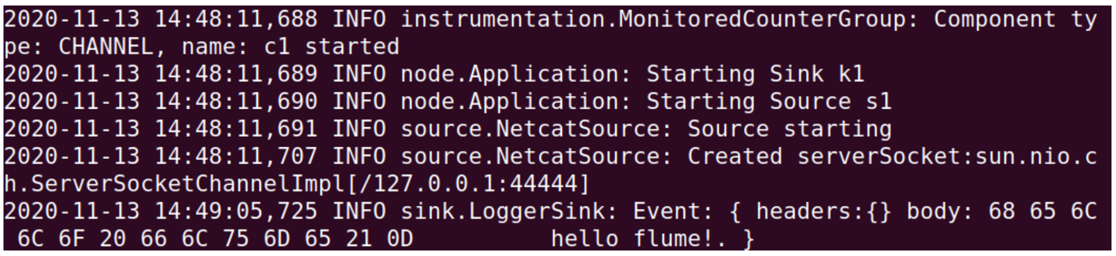
Netcat source 在某一端口上进行监听，并将接收到的数据每一行文字作为一个 event，也就是数据是基于换行符分隔。并通过连接通道发送。

将下面的内容添加到文件/apps/flume/conf/case_netcat.conf

```
# Name the components on this agent
a1.sources = s1
a1.sinks = k1
a1.channels = c1

# Bind the source and sink to the channel
a1.sources.s1.channels = c1
a1.sinks.k1.channel = c1

# Describe/configure the source
a1.sources.s1.type = netcat
a1.sources.s1.bind = localhost
a1.sources.s1.port = 44444

# Describe the sink
a1.sinks.k1.type = logger

# Use a channel which buffers events in memory
a1.channels.c1.type = memory
a1.channels.c1.capacity = 1000
a1.channels.c1.transactionCapacity = 100
```

启动 Flume

```
flume-ng agent --conf conf --conf-file \
/apps/flume/conf/case_netcat.conf --name a1 \
-Dflume.root.logger=INFO,console
```



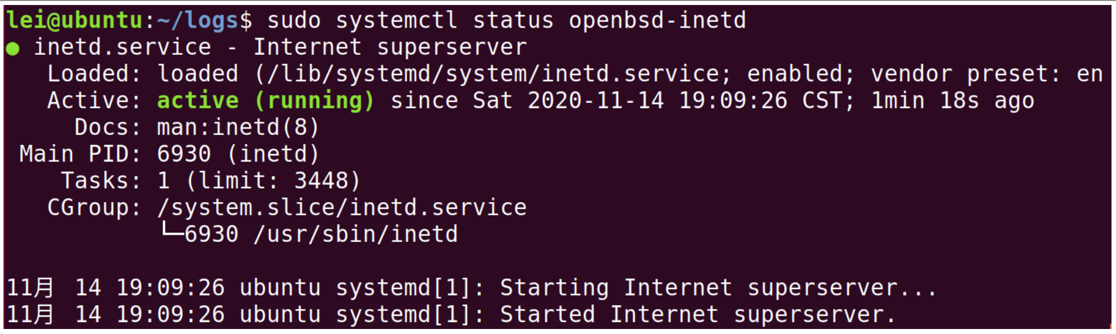
```
2020-11-13 14:48:11,688 INFO instrumentation.MonitoredCounterGroup: Component ty
pe: CHANNEL, name: c1 started
2020-11-13 14:48:11,689 INFO node.Application: Starting Sink k1
2020-11-13 14:48:11,690 INFO node.Application: Starting Source s1
2020-11-13 14:48:11,691 INFO source.NetcatSource: Source starting
2020-11-13 14:48:11,707 INFO source.NetcatSource: Created serverSocket:sun.nio.c
h.ServerSocketChannelImpl[/127.0.0.1:44444]
2020-11-13 14:49:05,725 INFO sink.LoggerSink: Event: { headers:{} body: 68 65 6C
 6C 6F 20 66 6C 75 6D 65 21 0D          hello flume!. }
```

下面使用 telnet 向端口发送数据，先安装 telent

```
sudo apt-get install openbsd-inetd
sudo apt-get install telnetd
```

查看运行状态

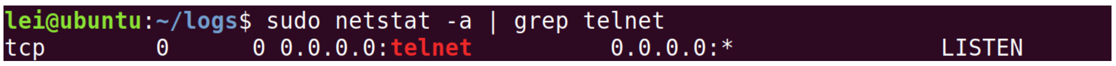
```
sudo systemctl status openbsd-inetd
```



```
lei@ubuntu:~/logs$ sudo systemctl status openbsd-inetd
● inetd.service - Internet superserver
   Loaded: loaded (/lib/systemd/system/inetd.service; enabled; vendor preset: en
   Active: active (running) since Sat 2020-11-14 19:09:26 CST; 1min 18s ago
     Docs: man:inetd(8)
 Main PID: 6930 (inetd)
    Tasks: 1 (limit: 3448)
   CGroup: /system.slice/inetd.service
           └─6930 /usr/sbin/inetd

11月 14 19:09:26 ubuntu systemd[1]: Starting Internet superserver...
11月 14 19:09:26 ubuntu systemd[1]: Started Internet superserver.
```

查看 telnet 服务是否开启

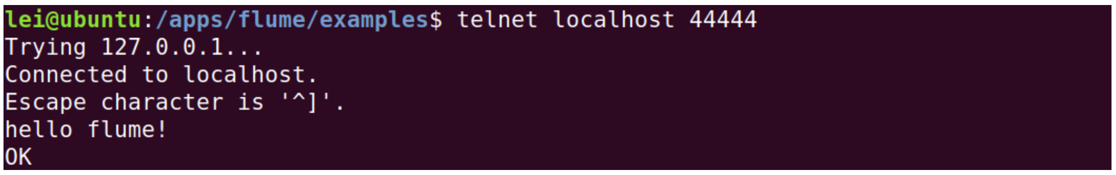
```
sudo apt install net-tools
sudo netstat -a | grep telnet
```



```
lei@ubuntu:~/logs$ sudo netstat -a | grep telnet
tcp        0      0 0.0.0.0:telnet          0.0.0.0:*               LISTEN
```

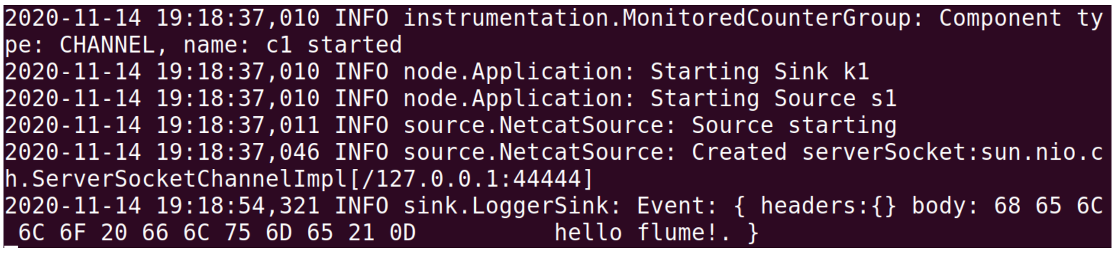
在另一终端中发送数据

```
telnet localhost 44444
```



```
lei@ubuntu:/apps/flume/examples$ telnet localhost 44444
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
hello flume!
OK
```

在 flume 的启动界面就会输出接收到的数据



```
2020-11-14 19:18:37,010 INFO instrumentation.MonitoredCounterGroup: Component ty
pe: CHANNEL, name: c1 started
2020-11-14 19:18:37,010 INFO node.Application: Starting Sink k1
2020-11-14 19:18:37,010 INFO node.Application: Starting Source s1
2020-11-14 19:18:37,011 INFO source.NetcatSource: Source starting
2020-11-14 19:18:37,046 INFO source.NetcatSource: Created serverSocket:sun.nio.c
h.ServerSocketChannelImpl[/127.0.0.1:44444]
2020-11-14 19:18:54,321 INFO sink.LoggerSink: Event: { headers:{} body: 68 65 6C
 6C 6F 20 66 6C 75 6D 65 21 0D          hello flume!. }
```

2.6 Syslogtcp Source

Syslogtcp source 接收 tcp 协议发过来的数据。

将下面的内容保存为/apps/flume/conf/case_syslogtcp.conf

```
# Name the components on this agent
a1.sources = s1
a1.sinks = k1
a1.channels = c1

# Bind the source and sink to the channel
a1.sources.s1.channels = c1
a1.sinks.k1.channel = c1

# Describe/configure the source
a1.sources.s1.type = syslogtcp
a1.sources.s1.port = 9999
a1.sources.s1.host = localhost

# Describe the sink
a1.sinks.k1.type = logger

# Use a channel which buffers events inmemory
a1.channels.c1.type = memory
```

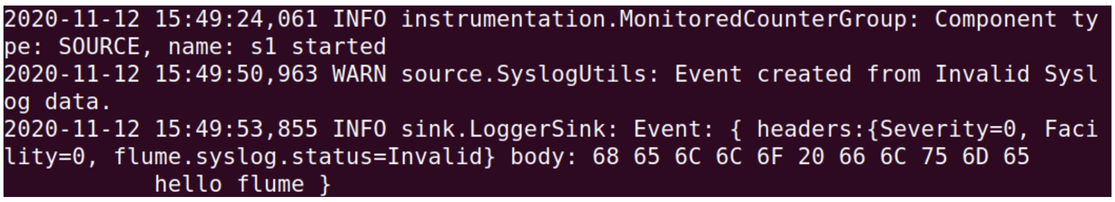
启动 Flume

```
flume-ng agent --conf conf --conf-file \
/apps/flume/conf/case_syslogtcp.conf --name a1 \
-Dflume.root.logger=INFO,console
```

另外打开一个终端，向端口发送内容

```
echo "hello flume" | nc localhost 9999
```

在 flume 的启动界面就会输出接收到的数据



```
2020-11-12 15:49:24,061 INFO instrumentation.MonitoredCounterGroup: Component ty
pe: SOURCE, name: s1 started
2020-11-12 15:49:50,963 WARN source.SyslogUtils: Event created from Invalid Sysl
og data.
2020-11-12 15:49:53,855 INFO sink.LoggerSink: Event: { headers:{Severity=0, Faci
lity=0, flume.syslog.status=Invalid} body: 68 65 6C 6C 6F 20 66 6C 75 6D 65
           hello flume }
```

2.7 HDFS Sink

将下面的内容保存为/apps/flume/conf/case_syslogtcp_hdfs.conf

```
# Name the components on this agent
a1.sources = s1
a1.sinks = k1
a1.channels = c1

# Bind the source and sink to the channel
a1.sources.s1.channels = c1
a1.sinks.k1.channel = c1

# Describe/configure the source
a1.sources.s1.type = syslogtcp
a1.sources.s1.port = 9999
a1.sources.s1.host = localhost

# Describe the sink
a1.sinks.k1.type=hdfs
a1.sinks.k1.hdfs.path=hdfs://localhost:9000/flume
a1.sinks.k1.hdfs.fileType=DataStream

# Use a channel which buffers events inmemory
a1.channels.c1.type = memory
```

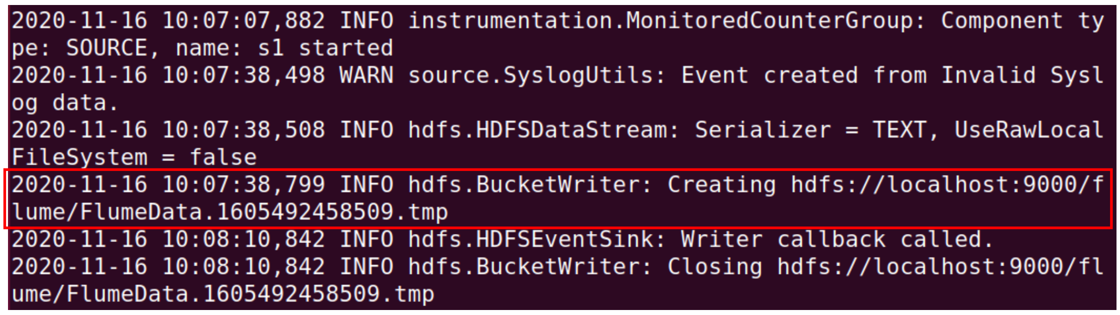
启动 Flume

```
flume-ng agent --conf conf --conf-file \
/apps/flume/conf/ case_syslogtcp_hdfs.conf --name a1 \
-Dflume.root.logger=INFO,console
```

另外打开一个终端，向端口发送内容

```
echo "hello flume" | nc localhost 9999
```

在 flume 的启动界面可以看到在 HDFS 上创建了一个文件

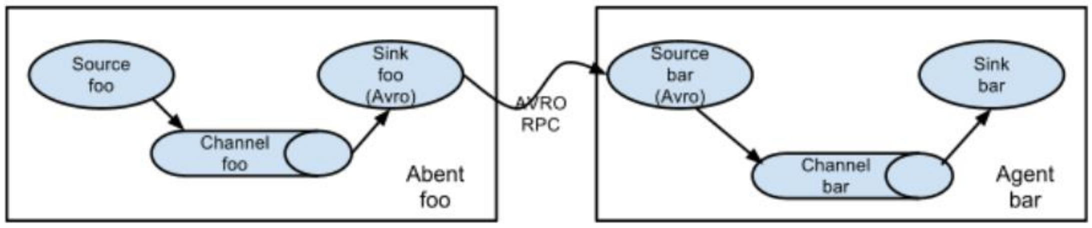


```
2020-11-16 10:07:07,882 INFO instrumentation.MonitoredCounterGroup: Component ty
pe: SOURCE, name: s1 started
2020-11-16 10:07:38,498 WARN source.SyslogUtils: Event created from Invalid Sysl
og data.
2020-11-16 10:07:38,508 INFO hdfs.HDFSDataStream: Serializer = TEXT, UseRawLocal
FileSystem = false
2020-11-16 10:07:38,799 INFO hdfs.BucketWriter: Creating hdfs://localhost:9000/f
lume/FlumeData.1605492458509.tmp
2020-11-16 10:08:10,842 INFO hdfs.HDFSEventSink: Writer callback called.
2020-11-16 10:08:10,842 INFO hdfs.BucketWriter: Closing hdfs://localhost:9000/fl
ume/FlumeData.1605492458509.tmp
```

查看其中的内容，确认就是我们发送的字符串。

```
hadoop fs -cat /flume/*
```

2.8 配置多代理流程



设置一个多层的流程,第一个 agent 需要有一个 Avro sink 指向第二个 agent 的 Avro 源。

第一个 agent 将转发 Event 到下一个 agent。

创建文件/apps/flume/conf/case_avro_sink.conf

```
# Name the components on this agent
a2.sources = s1
a2.sinks = k1
a2.channels = c1

# Bind the source and sink to the channel
a2.sources.s1.channels = c1
a2.sinks.k1.channel = c1

# Describe/configure the source
a2.sources.s1.type = syslogtcp
a2.sources.s1.host = localhost
a2.sources.s1.port = 33333

# Describe the channel
a2.channels.c1.type = memory
```

```
a2.channels.c1.capacity = 1000
a2.channels.c1.transactionCapacity = 100

# Describe the sink
a2.sinks.k1.type = avro
a2.sinks.k1.hostname = localhost
a2.sinks.k1.port = 22222
```

创建文件 /apps/flume/conf/case_avro.conf

```
# Name the components on this agent
a1.sources = s1
a1.sinks = k1
a1.channels = c1

# Bind the source and sink to the channel
a1.sources.s1.channels = c1
a1.sinks.k1.channel = c1

# Describe/configure the source
a1.sources.s1.type = avro
a1.sources.s1.bind = localhost
a1.sources.s1.port = 22222

# Describe the channel
a1.channels.c1.type = memory
a1.channels.c1.capacity = 1000
a1.channels.c1.transactionCapacity = 100

# Describe the sink
a1.sinks.k1.type = logger
```

注意 case_avro_sink.conf 是前面的 Agent，case_avro.conf 是后面的 Agent.

先在一个终端中启动 Avro 的 Source,监听端口

```
flume-ng agent --conf conf --conf-file \
/apps/flume/conf/case_avro.conf --name a1 \
-Dflume.root.logger=DEBUG,console \
-Dorg.apache.flume.log.printconfig=true \
-Dorg.apache.flume.log.rawdata=true
```

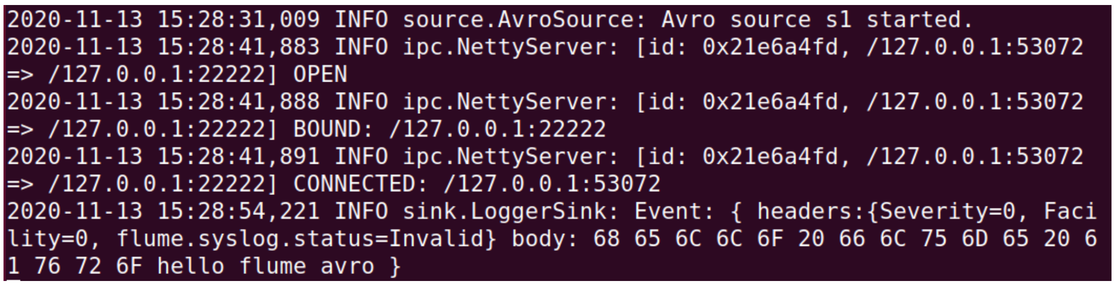
再在另一个终端中启动 Avro 的 Sink

```
flume-ng agent --conf conf --conf-file \
/apps/flume/conf/case_avro_sink.conf --name a2 \
-Dflume.root.logger=DEBUG,console \
-Dorg.apache.flume.log.printconfig=true \
-Dorg.apache.flume.log.rawdata=true
```

再开一个终端中发送数据

```
echo "hello flume avro sink" | nc localhost 33333
```

可以看到已经建立连接，第二次 agent 成功接收到数据



```
2020-11-13 15:28:31,009 INFO source.AvroSource: Avro source s1 started.
2020-11-13 15:28:41,883 INFO ipc.NettyServer: [id: 0x21e6a4fd, /127.0.0.1:53072
=> /127.0.0.1:22222] OPEN
2020-11-13 15:28:41,888 INFO ipc.NettyServer: [id: 0x21e6a4fd, /127.0.0.1:53072
=> /127.0.0.1:22222] BOUND: /127.0.0.1:22222
2020-11-13 15:28:41,891 INFO ipc.NettyServer: [id: 0x21e6a4fd, /127.0.0.1:53072
=> /127.0.0.1:22222] CONNECTED: /127.0.0.1:53072
2020-11-13 15:28:54,221 INFO sink.LoggerSink: Event: { headers:{Severity=0, Faci
lity=0, flume.syslog.status=Invalid} body: 68 65 6C 6C 6F 20 66 6C 75 6D 65 20 6
1 76 72 6F hello flume avro }
```

3. Flume AVRO Client 开发

在实际工作中，数据的生产方式极具多样性，Flume 虽然包含了一些内置的机制来采集数据，但是更多的时候用户希望能将应用程序和 Flume 直接相连。下面学习在应用程序中，通过 IPC 或 RPC 连接 Flume 并往 Flume 发送数据。

Flume 的 RpcClient 实现了 Flume 的 RPC 机制。用户的应用程序可以很简单的调用 Flume Client SDK 的 append(Event) 或者 appendBatch(List<Event>) 方法发送数据。用户可以通过直接实现 Event 接口提供所需的 event，例如可以使用 EventBuilder 的 writeBody()静态辅助方法。

自 Flume 1.4.0 起，Avro 是默认的 RPC 协议。NettyAvroRpcClient 和 ThriftRpcClient 实现了 RpcClient 接口。实现中我们需要知道我们将要连接的目标

agent 的 host 和 port 用于创建 client 实例，然后使用 RpcClient 发送数据到 Flume agent。

进程间通信（IPC）是在多任务操作系统或联网的计算机之间运行的程序和进程所用的通信技术。有两种类型的进程间通信：

- **本地过程调用（LPC）**：LPC 用在多任务操作系统中，使得同时运行的任务能互相会话。这些任务共享内存空间使任务同步和互相发送信息。
- **远程过程调用（RPC）**：RPC 类似于 LPC，只是在网上工作 RPC 开始是出现在 Sun 微系统公司和 HP 公司运行 UNIX 操作系统的计算机中。

创建工程【flume_examples】,包【sds.flume】,类【AVRO_Client】，将下面的代码复制其中。

```
package sds.flume;

import org.apache.flume.Event;
import org.apache.flume.EventDeliveryException;
import org.apache.flume.api.RpcClient;
import org.apache.flume.api.RpcClientFactory;
import org.apache.flume.event.EventBuilder;
import java.nio.charset.Charset;

public class AVRO_Client {
  public static void main(String[] args) {
    MyRpcClientFacade client = new MyRpcClientFacade();
    // Initialize client with the remote Flume agent's host and port
client.init("localhost",42424);

   // Send 10events to the remote Flume agent.
   // configured tolisten with an AvroSource.
   String sampleData = "Hello Flume!";
   for (int i =0; i < 10; i++) {
     client.sendDataToFlume(sampleData);
   }
   client.cleanUp();
  }
}

class MyRpcClientFacade {
  private RpcClient client;
  private String hostname;
  private int port;

  public void init(String hostname, int port) {
   // Setup the RPCconnection
   this.hostname = hostname;
   this.port = port;
   this.client = RpcClientFactory.getDefaultInstance(hostname, port);
  }

  public void sendDataToFlume(String data) {
   // Create a Flume Event object that encapsulates the sample data
   Event event = EventBuilder.withBody(data, Charset.forName("UTF-
8"));
  // Send the event
   try {
     client.append(event);
   } catch (EventDeliveryException e) {
     // clean up and recreate the client
     client.close();
     client = null;
     client = RpcClientFactory.getDefaultInstance(hostname, port);
   }
  }

  public void cleanUp() {
   // Close the RPCconnection
   client.close();
  }
}
```

将/apps/flume/lib 中的 jar 包全部导入工程中。

在/apps/flume/conf 中创建文件 avro_client.conf

```
# Name the components on this agent
a1.sources = s1
a1.sinks = k1
a1.channels = c1

```

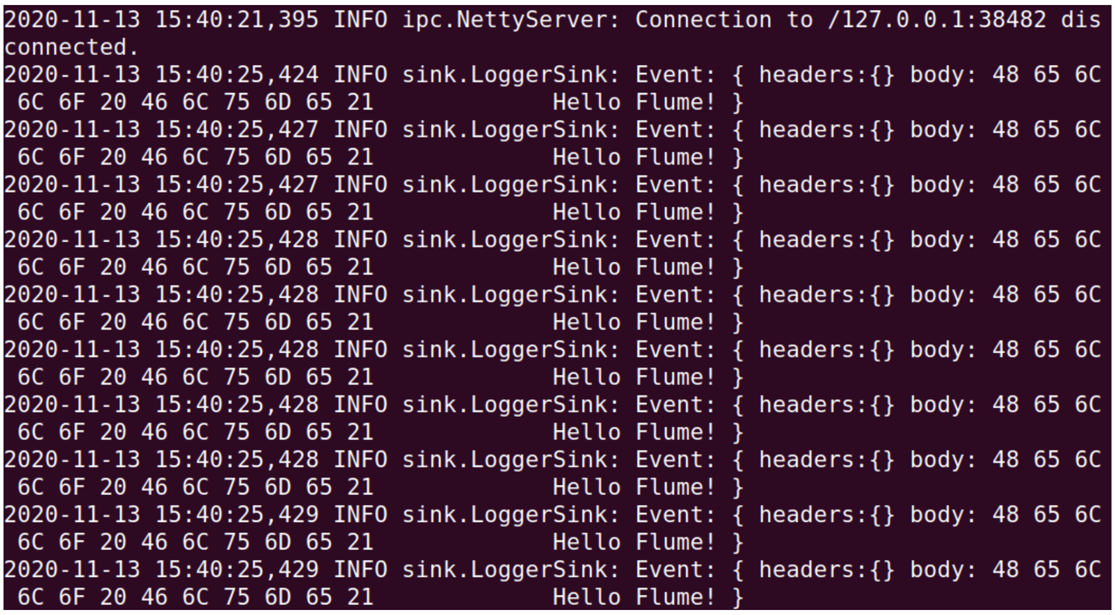
```
# Describe/configure the source
a1.sources.s1.type = avro
a1.sources.s1.port = 42424
a1.sources.s1.bind = localhost
a1.sources.s1.channels = c1

# Describe the sink
a1.sinks.k1.channel = c1
a1.sinks.k1.type = logger

# Use a channel which buffers events inmemory
a1.channels.c1.type = memory
#a1.channels.c1.capacity = 1000
#a1.channels.c1.transactionCapacity = 100
```

在终端中启动 Flume

```
flume-ng agent -conf conf --conf-file \
/apps/flume/conf/avro_client.conf --name a1 \
-Dflume.root.logger=INFO,console
```

启动成功后，在 eclipse 中执行 AVRO_client 类，在终端 Flume 启动界面可以看到输出了

10 遍 Hello Flume!



```
2020-11-13 15:40:21,395 INFO ipc.NettyServer: Connection to /127.0.0.1:38482 dis
connected.
2020-11-13 15:40:25,424 INFO sink.LoggerSink: Event: { headers:{} body: 48 65 6C
 6C 6F 20 46 6C 75 6D 65 21                Hello Flume! }
2020-11-13 15:40:25,427 INFO sink.LoggerSink: Event: { headers:{} body: 48 65 6C
 6C 6F 20 46 6C 75 6D 65 21                Hello Flume! }
2020-11-13 15:40:25,427 INFO sink.LoggerSink: Event: { headers:{} body: 48 65 6C
 6C 6F 20 46 6C 75 6D 65 21                Hello Flume! }
2020-11-13 15:40:25,428 INFO sink.LoggerSink: Event: { headers:{} body: 48 65 6C
 6C 6F 20 46 6C 75 6D 65 21                Hello Flume! }
2020-11-13 15:40:25,428 INFO sink.LoggerSink: Event: { headers:{} body: 48 65 6C
 6C 6F 20 46 6C 75 6D 65 21                Hello Flume! }
2020-11-13 15:40:25,428 INFO sink.LoggerSink: Event: { headers:{} body: 48 65 6C
 6C 6F 20 46 6C 75 6D 65 21                Hello Flume! }
2020-11-13 15:40:25,428 INFO sink.LoggerSink: Event: { headers:{} body: 48 65 6C
 6C 6F 20 46 6C 75 6D 65 21                Hello Flume! }
2020-11-13 15:40:25,428 INFO sink.LoggerSink: Event: { headers:{} body: 48 65 6C
 6C 6F 20 46 6C 75 6D 65 21                Hello Flume! }
2020-11-13 15:40:25,429 INFO sink.LoggerSink: Event: { headers:{} body: 48 65 6C
 6C 6F 20 46 6C 75 6D 65 21                Hello Flume! }
2020-11-13 15:40:25,429 INFO sink.LoggerSink: Event: { headers:{} body: 48 65 6C
 6C 6F 20 46 6C 75 6D 65 21                Hello Flume! }
```