

# Flume Integration with Kafka

## 概述：

(1) kafka 和 flume 都是日志系统。kafka 是分布式消息中间件，自带存储，提供 push 和 pull 存取数据功能。flume 分为 agent（数据采集器）,collector（数据简单处理和写入）,storage（存储器）三部分，每一部分都是可以定制的。比如 agent 采用 RPC（Thrift-RPC）、text（文件）等，storage 指定用 hdfs 做。

(2) kafka 做日志缓存应该是更为合适的，但是 flume 的数据采集部分做的很好，可以定制很多数据源，减少开发量。所以比较流行 flume+kafka 模式，如果为了利用 flume 写 hdfs 的能力，也可以采用 kafka+flume 的方式。

## 1. Flume 介绍

Flume 是一个分布式、可靠、和高可用的海量日志聚合的系统，支持在系统中定制各类数据发送方，用于收集数据；同时，Flume 提供对数据进行简单处理，并写到各种数据接受方（可定制）的能力。

### 设计目标：

可靠性: 当节点出现故障时，日志能够被传送到其他节点上而不会丢失。Flume 提供了三种级别的可靠性保障，从强到弱依次分别为：**end-to-end**（收到数据 agent 首先将 event 写到磁盘上，当数据传送成功后，再删除；如果数据发送失败，可以重新发送。），**Store on failure**（这也是 scribe 采用的策略，当数据接收方 crash 时，将数据写到本地，待恢复后，继续发送），**Best effort**（数据发送到接收方后，不会进行确认）。

可扩展性: Flume 采用了三层架构，分别为 agent, collector 和 storage，每一层均可以水平扩展。其中，所有 agent 和 collector 由 master 统一管理，这使得系统容易监控和维护，且 master 允许有多个（使用 ZooKeeper 进行管理和负载均衡），这就避免了单点故障问题。

可管理性: 所有 agent 和 collector 由 master 统一管理，这使得系统便于维护。多 master 情况，Flume 利用 ZooKeeper 和 gossip，保证动态配置数据的一致性。用户可以在 master 上查看各个数据源或者数据流执行情况，且可以对各个数据源配置和动态加载。Flume 提供了 web 和 shell script command 两种形式对数据流进行管理。

功能可扩展性用户可以根据需要添加自己的 agent, collector 或者 storage。此外，Flume 自带了很多组件，包括各种 agent（file, syslog 等），collector 和 storage（file, HDFS 等）。

**数据采集：**负责从各节点上实时采集数据，选用 Cloudera 的 flume 来实现

**数据接入：**由于采集数据的速度和数据处理的速度不一定同步，因此添加一个消息中间件来作为缓冲，选用 apache 的 kafka

**流式计算：**对采集到的数据进行实时分析，选用 apache 的 storm

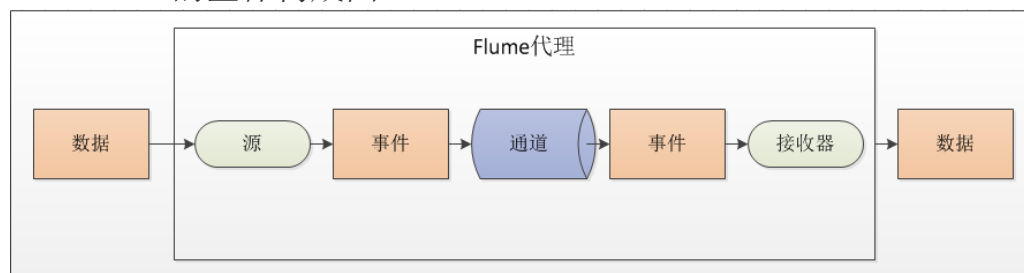
**数据输出：**对分析后的结果持久化，暂定用 mysql，另一方面是模块化之后，假如当 Storm 挂掉了之后，数据采集和数据接入还是继续在跑着，数据不会丢失，storm 起来之后可以继续流式计算；

## 2. Flume 的一些核心概念：

组件名称	功能介绍
Agent 代理	使用 JVM 运行 Flume。每台机器运行一个 agent，但是可以在一个 agent 中包含多个 sources 和 sinks。
Client 客户端	生产数据，运行在一个独立的线程。
Source 源	从 Client 收集数据，传递给 Channel。
Sink 接收器	从 Channel 收集数据，进行相关操作，运行在一个独立线程。

组件名称	功能介绍
Channel 通道	连接 sources 和 sinks ， 这个有点像一个队列。
Events 事件	传输的基本数据负载。

### 3. Flume 的整体构成图



### 4. Flume 配置

Flume 完全可以满足我们这些要求，在 Flume 中，Spooling 模式可以扫描一个文件目录下所有的文件，并将新增的文件发送给 HDFS；根据我们的架构设计要求，实时数据发给 kafka，历史数据发给 HDFS；同时其 TAIL DIR 模式中，可以扫描一个（或者多个）文件，不断 tail 其最新追加的信息，然后发送给 kafka。基本概念：

1、source：源文件、源数据端，指定 Flume 从何处采集数据（流）。Flume 支持多种 source，比如“Avro source”（类似 RPC 模式，接收远端 Avro 客户端发送的数据 Entity）、“Thrift Source”（Thrift 客户端发送的数据）、“Exec Source”（linux 指令返回的数据条目）、“Kafka Source”、“Syslog Source”、“Http Source”等等。

本文主要涉及到 Spooling 和 Taildir 两种，Taildir 是 1.7 新增的特性，在此之前，如果想实现 tail 特性，需要使用“Exec Source”来模拟，或者自己开发代码。

2、channel：通道，简单而言就是数据流的缓冲池，多个 source 的数据可以发送给一个 channel，在 channel 内部可以对数据进行 cache、溢出暂存、流量整形等。目前 Flume 支持“Memory Channel”（数据保存在有限空间的内存中）、“JDBC Channel”（数据暂存在数据库中，保障恢复）、“Kafka Channel”（暂存在 kafka 中）、“File Channel”（暂存在本地文件中）；除 Memory 之外，其他的 channel 都支持持久化，可以在故障恢复、sink 离线或者无 sink 等场景下提供有效的担保机制，避免消息丢失和流量抗击。

3、sink：流输出端，每个 channel 都可以对应一个 sink，每个 sink 可以指定一种类型的存储方式，目前 Flume 支持的 sink 类型比较常用的有“HDFS Sink”（将数据保存在 hdfs 中）、“Hive Sink”、“Logger Sink”（特殊场景，将数据以 INFO 级别输出到控制台，通常用于测试）、“Avro Sink”、“Thrift Sink”、“File Roll Sink”（转存到本地文件系统中）等等。

source、channel、sink 这种模型就是 pipeline，一个 source 的数据可以“复制”到多个 channels（扇出），当然多个 source 也可以聚集到一个 channel 中，每个 channel 对应一个 sink。每种类型的 source、channel、sink 都有各自的配置属性，用于更好的控制数据流。

Flume 是 java 语言开发，所以我们在启动 Flume 之前，需要设定 JVM 的堆栈大小等选参，以免 Flume 对宿主机上的其他 application 带来负面影响。在 conf 目录下，修改 flume-env.sh：

```
export JAVA_OPTS="-Dcom.sun.management.jmxremote -verbose:gc -server -Xms1g -Xmx1g  
-XX:NewRatio=3 -XX:SurvivorRatio=8 -XX:MaxMetaspaceSize=128M -XX:+UseConcMarkSweepGC  
-XX:CompressedClassSpaceSize=128M -XX:MaxTenuringThreshold=5  
-XX:CMSInitiatingOccupancyFraction=70 -XX:+PrintGCDetails -XX:+PrintGCDateStamps  
-Xloggc:/opt/flume/logs/server-gc.log.$(date +%F) -XX:+UseGCLogFileRotation  
-XX:NumberOfGCLogFiles=1 -XX:GCLogFileSize=64M"
```

除此之外，就是 **flume** 的启动配置文件了（**flume-conf.properties**），

## ● Nginx:

如下配置我们模拟一个收集 **nginx** 日志的场景：

```
##main  
nginx.channels=ch-spooling ch-tail  
nginx.sources=spooling-source tail-source  
nginx.sinks=hdfs-spooling kafka-tail  
  
##channel  
nginx.channels.ch-spooling.type=file  
nginx.channels.ch-spooling.checkpointDir=/data/flume/.flume/file-channel/ch-spooling/ch  
eckpoint  
nginx.channels.ch-spooling.dataDirs=/data/flume/.flume/file-channel/ch-spooling/data  
nginx.channels.ch-spooling.capacity=1000  
nginx.channels.ch-spooling.transactionCapacity=100  
nginx.channels.ch-spooling.capacity=100000  
  
nginx.channels.ch-tail.type=file  
nginx.channels.ch-tail.checkpointDir=/data/flume/.flume/file-channel/ch-tail/checkpoint  
nginx.channels.ch-tail.dataDirs=/data/flume/.flume/file-channel/ch-tail/data  
nginx.channels.ch-tail.capacity=1000  
nginx.channels.ch-tail.transactionCapacity=100  
nginx.channels.ch-tail.capacity=100000  
  
##source, 历史数据  
nginx.sources.spooling-source.type=spooldir  
nginx.sources.spooling-source.channels=ch-spooling  
##指定 logs 目录  
nginx.sources.spooling-source.spoolDir=/data/logs/nginx  
##开启 header, 此后 event 将携带此 header  
nginx.sources.spooling-source.fileHeader=true  
nginx.sources.spooling-source.fileHeaderKey=file  
##header 中增加文件名  
nginx.sources.spooling-source.basenameHeader=true  
nginx.sources.spooling-source.basenameHeaderKey=basename  
##日志发送完毕后，是否删除此源文件，  
# “immediate” 表示发送完毕后立即删除，可以节约磁盘空间  
nginx.sources.spooling-source.deletePolicy=never
```

```
##包含的文件的列表，我们约定所有的日志每天 rotate，
##格式为 “<filename>.log-<yyyyMMdd>”
##当前的日志，不会被包含进来。
nginx.sources.spooling-source.includePattern=^.*\.log-.$
nginx.sources.spooling-source.consumeOrder=oldest
nginx.sources.spooling-source.recursiveDirectorySearch=false
nginx.sources.spooling-source.batchSize=100
nginx.sources.spooling-source.inputCharset=UTF-8
##如果编解码失败，忽略相应的字符。
nginx.sources.spooling-source.decodeErrorPolicy=IGNORE
nginx.sources.spooling-source.selector.type=replicating
nginx.sources.spooling-source.interceptors=i1 i2
##使用 timestamp 拦截器，将会在 event header 中增加时间戳字段
nginx.sources.spooling-source.interceptors.i1.type=timestamp
##使用 host 拦截器，将会在 event header 中增加“host”字段，值为 ip
nginx.sources.spooling-source.interceptors.i2.type=host
nginx.sources.spooling-source.interceptors.i2.useIP=true
nginx.sources.spooling-source.interceptors.i2.hostHeader=host

nginx.sources.tail-source.type=TAILDIR
nginx.sources.tail-source.channels=ch-tail
##本人不想写 flume 的扩展代码，所以就为每个 tail 的文件指定一个 group
nginx.sources.tail-source.filegroups=www error
nginx.sources.tail-source.filegroups.www=/data/logs/nginx/www.log
nginx.sources.tail-source.filegroups.error=/data/logs/nginx/error.log
##对于 taildir，需要间歇性的保存 tail 文件的位置，以便中断后可以继续
##json 格式文件
nginx.sources.tail-source.positionFile=/data/flume/.flume/ch-tail/taildir_position.json
##每个 tail 的文件，创建一个 kafka topic
nginx.sources.tail-source.headers.www.topic=nginx-www
nginx.sources.tail-source.headers.error.topic=nginx-error
nginx.sources.tail-source.skipToEnd=true
nginx.sources.tail-source.interceptors=i1 i2
nginx.sources.tail-source.interceptors.i1.type=timestamp
nginx.sources.tail-source.interceptors.i2.type=host
nginx.sources.tail-source.interceptors.i2.useIP=true
nginx.sources.tail-source.interceptors.i2.hostHeader=host

##spooling 历史数据
nginx.sinks.hdfs-spooling.channel=ch-spooling
nginx.sinks.hdfs-spooling.type=hdfs
nginx.sinks.hdfs-spooling.hdfs.fileType=DataStream
nginx.sinks.hdfs-spooling.hdfs.writeFormat=Text
##保存在 hdfs 中，路径表达了日志分类信息，第一级为<project>
##第二级为<date>
##即同一个项目的日子，按照日期汇总。
nginx.sinks.hdfs-spooling.hdfs.path=hdfs://hadoop-ha/logs/nginx/%Y-%m-%d
##hdfs 的文件名中包括此源文件所在的 host 地址，便于数据分拣
nginx.sinks.hdfs-spooling.hdfs.filePrefix=%{basename}.[%{host}]
```

```

##对于 spooling 的文件，文件名尽可能接近原始名称，所以 suffix 值为空
nginx.sinks.hdfs-spooling.hdfs.fileSuffix=
##文件在同步过程中，后缀为.tmp
nginx.sinks.hdfs-spooling.hdfs.inUseSuffix=.tmp
##不按照时间间隔滚动生成新文件
nginx.sinks.hdfs-spooling.hdfs.rollInterval=0
##1G，当文件大小达到 1G 后，滚动生成新文件
nginx.sinks.hdfs-spooling.hdfs.rollSize=1073741824
##不按照 event 条数滚动生成新文件
nginx.sinks.hdfs-spooling.hdfs.rollCount=0
##IO 通道空闲 60S 秒后，关闭
nginx.sinks.hdfs-spooling.hdfs.idleTimeout=60

##tail 实时数据
nginx.sinks.kafka-tail.channel=ch-tail
nginx.sinks.kafka-tail.type=org.apache.flume.sink.kafka.KafkaSink
##kafka 集群地址，可以为其子集
nginx.sinks.kafka-tail.kafka.bootstrap.servers=10.0.3.78:9092,10.0.4.78:9092,10.0.4.79:
9092,10.0.3.77:9092
##注意，topic 中不支持参数化
##但是为了提高扩展性，我们把 topic 信息通过 header 方式控制
#nginx.sinks.kafka-tail.kafka.topic=nginx-%{filename}
##default 100，值越大，网络效率越高，但是延迟越高，准实时
nginx.sinks.kafka-tail.flumeBatchSize=32
nginx.sinks.kafka-tail.kafka.producer.acks=1
##use Avro-event format,will contain flume-headers
##default : false
nginx.sinks.kafka-tail.useFlumeEventFormat=false

```

这是一个很长的配置文件，各个配置项的含义大家可以去官网查阅，我们需要注意几个地方：

- 1) **checkpoint**、**data** 目录，最好指定，这对以后排查问题很有帮助
- 2) **channel**，我们需要显示声明其类型，通常我们使用 **file**，对流量抗击有些帮助，前提是指定的目录所在磁盘空间应该相对充裕和高速。
- 3) **header** 并不会真的会写入 **sink**，**header** 信息只是在 **source**、**channel**、**sink** 交互期间有效；我们可以通过 **header** 标记一个 **event** 流动的特性。
- 4) 对于 **spooling source**，建议开启 **basename**，即文件的实际名称，我们可以将此 **header** 传递到 **sink** 阶段。
- 5) 所有涉及到 **batchSize** 的特性，都是需要权衡的：在发送效率和延迟中做出合理的决策。
- 6) **interceptor** 是 **Flume** 很重要的特性，可以帮助我们在 **source** 生命周期之后做一些自定义的操作，比如增加 **header**、内容修正等；此时我们需要关注一些性能问题。

7) 对于 **taildir**, **filegroups** 中可以指定多个值, 我的设计原则是一个 **tail** 文件对应一个 **group** 名称, 目前还没有特别好的办法来通配 **tail** 文件, 只能逐个声明。

8) 对于 **kafka sink**, **topic** 信息可以通过“**kafka.topic**”指定, 也可以在通过 **header** 指定 (**headers.www.topic**, “**www**”对应 **group** 名称, “**topic**”是 **header** 的 **key** 名称)。为了灵活性, 我更倾向于在 **headers** 中指定 **topic**。

9) **hdfs sink** 需要注意其 **roll** 的时机, 目前影响 **roll** 时机的几个参数“**minBlockReplicas**”、“**rollInterval**” (根据时间间隔)、“**rollSize**”(根据文件尺寸)、“**rollCount**” (根据 **event** 条数); 此外“**round**”相关的选项也可以干预滚动生成新文件的时机。

关于 **hdfs sink**, **flume** 每次 **flush** 都将生成一个新的 **hdfs** 文件, 最终导致生成很多小文件, 希望一个 **tail** 的文件最终在 **hdfs** 中也是一个文件; 后来经过考虑, 使用基于 **rollSize** 来滚动生成文件, 通常的 **nginx** 日志文件不超过 **1G**, 那么我就让 **rollSize** 设置为 **1G**, 这样就可以确保不会 **roll**。此外, **hdfs** 每个文件都会有一个“数字”后缀, 这个数字是一个内部的 **counter**, 目前没有办法通过配置的方式来“消除”, 我们先暂且接受吧。

如下为 **nginx** 中 **log\_format** 样例, 我们在每条日志的首个位置, 设置了 **\$hostname** 用于标记此文件的来源机器, 便于 **kafka** 消息消费者分拣数据。

```
log_format main '$hostname|$remote_addr|$remote_user|$time_local|$request|'
                '$status|$body_bytes_sent|$http_referer|$request_id|'
                '$http_user_agent|$http_x_forwarded_for|$request_time|$upstream_response_time|$upstream
_addr|$upstream_connect_time';
```

对于 **flume** 的配置, 我们可以通过 **zookeeper** 来保存, 这是 **1.7** 版本新增的特性, 配置中心化, 这种方式大家可以参考。不过考虑到配置的可见性, 并没有将配置放在 **zookeeper** 中, 而是放在了一台配置中控机上, 通过 **jenkins** 来部署 **flume**, 每个 **project** 分布式部署, 每个节点一个 **flume** 实例, 它们使用同一个配置文件, 在部署 **flume** 时从中控机上 **scp** 新配置即可。(这需要先有一个自动化部署平台)

我们看到配置文件中的配置项都以“**nginx**”开头, 这个前缀表示 **agent** 的名称, 我们可以根据实际业务来命名即可, 但是在启动 **flume** 时必须制定, 原则上一个 **flume-conf.properties** 文件中可以声明多个 **agent** 的配置项, 不过我们通常不建议这么用。

我们把 **flume** 部署在 **nginx** 所在机器上, 调整好配置文件, 即可启动, **flume** 启动脚本:

```
nohup bin/flume-ng agent
--conf conf
--conf-file flume-conf.properties
--name nginx
-Dflume.root.logger=INFO,CONSOLE
-Dorg.apache.flume.log.printconfig=true
-Dorg.apache.flume.log.rawdata=true
```

上述启动指令中，`--config-file` 就是指定配置文件的路径和名称，`--name` 指定 agent 名称（与配置文件中的配置项前缀保持一致），`logger` 信息我们在线上为 `INFO`，在测试期间可以指定为 `"DEBUG,LOGFILE"` 便于我们排查问题。

## ● tomcat 业务日志收集

关于 Flume 收集 tomcat 业务日志，需要调整的点比较多；设计初衷是：

- 1) HDFS 中收集所有的历史日志，包括 `catalina`、`access_log`、业务日志等。
- 2) kafka 只实时收集 `access_log` 和指定的业务日志；我们可以用这些数据做业务监控等。

### 1、tomcat 日志格式

我们首先调整 tomcat 中的 `logging.properties`：

```
1catalina.org.apache.juli.AsyncFileHandler.level = FINE
1catalina.org.apache.juli.AsyncFileHandler.directory = ${catalina.base}/logs
##here
1catalina.org.apache.juli.AsyncFileHandler.prefix = catalina.log.
1catalina.org.apache.juli.AsyncFileHandler.suffix =

2localhost.org.apache.juli.AsyncFileHandler.level = FINE
2localhost.org.apache.juli.AsyncFileHandler.directory = ${catalina.base}/logs
2localhost.org.apache.juli.AsyncFileHandler.prefix = localhost.log.
2localhost.org.apache.juli.AsyncFileHandler.suffix =

3manager.org.apache.juli.AsyncFileHandler.level = FINE
3manager.org.apache.juli.AsyncFileHandler.directory = ${catalina.base}/logs
3manager.org.apache.juli.AsyncFileHandler.prefix = manager.log.
3manager.org.apache.juli.AsyncFileHandler.suffix =

4host-manager.org.apache.juli.AsyncFileHandler.level = FINE
4host-manager.org.apache.juli.AsyncFileHandler.directory = ${catalina.base}/logs
4host-manager.org.apache.juli.AsyncFileHandler.prefix = host-manager.log.
4host-manager.org.apache.juli.AsyncFileHandler.suffix =
```

因为 tomcat 日志文件滚动格式默认为 `"catalina.<yyyy-MM-dd>.log"`，我们应该把它调整为 `"catalina.log.<yyyy-MM-dd>"`，我们可以通过上述配置方式来达成，最终我们希望无论是 tomcat 自己的日志、application 的业务日志，滚动生成的文件名格式都统一为 `"<filename>.log.<yyyy-MM-dd>"`，这样便于我们在 flume 中配置正则表达式来 spooling 这些历史文件。

Flume 的配置文件与 nginx 基本类似，此处不再赘言。

## ● 业务日志

我们约定 **application** 的业务日志也打印在 `${tomcat_home}/logs` 目录下, 即与 `catalina.out` 在一个目录, 每个业务日志每天滚动生成新的历史文件, 文件后缀以“.yyyy-MM-dd”结尾, 这类文件称为历史文件, 被同步到 **HDFS** 中。对于实时的日志信息, 我们仍然发送给 **kafka**, **kafka topic** 的设计思路跟 **nginx** 一样, 每个 **project** 一种文件对应一个 **topic**, 每种文件的日志来自多个 **application** 实例, 它们混淆在 **kafka topic** 中, 为了便于日志分拣, 我们需要在每条日志中增加一个 **IP** 标志项。通过整理发现, 在 **logback** 中打印 **local ip** 默认是不支持的, 所以我们需要变通一下, 我们在 **tomcat** 的启动脚本中定义一个 **LOCAL\_IP** 这个环境变量, 然后再 **logback.xml** 中引入即可解决。

```
##catalina.sh
##add
export LOCAL_IP=`hostname -I`
```

在项目中的 **logback.xml** 中即可通过 `${LOCAL_IP}` 变量声明即可

```
<appender name="FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
  <file>${LOG_HOME}/order_center.log</file>
  <Append>true</Append>
  <prudent>false</prudent>
  <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
    <pattern>${LOCAL_IP} %d{yyyy-MM-dd HH:mm:ss.SSS}
[%thread] %-5level %logger{50} - %msg%n</pattern>
  </encoder>
  <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
    <FileNamePattern>${LOG_HOME}/order_center.log.%d{yyyy-MM-dd}</FileNamePattern>
    <maxHistory>72</maxHistory>
  </rollingPolicy>
</appender>
```

## ● access\_log 日志

**tomcat** 的 **access\_log** 非常重要, 可以打印很多信息来帮助我们分析业务问题, 所以我们需要将 **access\_log** 日志整理规范; 我们在 **server.xml** 中通过修改如下内容即可:

```
<Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"
  prefix="localhost_access_log" suffix=".log" renameOnRotate="true"
  pattern="%A|%h|%m|%t|%D|"%r"|"%{Referer}i"|"%{User-Agent}i"|
t;|s|S|b|X-Request-ID|i|%{begin:msec}t|%{end:msec}t" />
```

“**renameOnRotate**”表示是否在 **rotate** 时机重命名 **access\_log**, 我们设定为 **true**, 这样 **access\_log** 文件名默认不带日期格式, 时间格式在 **rotate** 期间才添加进去。“**%A**”表示本机的 **local ip** 地址, 也是



用于 **kafka** 分拣日志的标记，**X-Request-ID** 是 **nginx** 层自定义的一个 **trace-ID** 用于跟踪请求的，如果你没有设定，则可以去掉。

到此为止，我们基本上可以完成这一套日志采集系统了，也为 **kafka** 分拣日志信息做好了铺垫，后续接入 **ELK**、**storm** 实时数据分析等也将相对比较容易。

## ● 问题总结：

### 1、flume + hdfs:

1) 我们首先将 **hdfs-site.xml**, **core-site.xml** 复制到 **\$(flume\_home)/conf** 目录下。且 **flume** 机器能够与 **hdfs** 所有节点通信（网络隔离、防火墙都可能导致它们无法正常通信）。

2) 在 **Flume** 根目录下，创建一个 **plugins.d/hadoop** 目录，创建 **lib**、**libext**、**native** 子目录；并将 **hadoop** 的相关依赖包复制到 **libext** 目录中：

```
commons-configuration-1.6.jar
hadoop-annotations-2.6.5.jar
hadoop-auth-2.6.5.jar
hadoop-common-2.6.5.jar
hadoop-hdfs-2.6.5.jar
htrace-core-3.0.4.jar
```

同时将如下文件复制到 **native** 目录中：

```
libhadoop.a
libhadooppipes.a
libhadoop.so.1.0.0
libhadooputils.a
libhdfs.a
libhdfs.so.0.0.0
```

这些依赖包，都可以在 **hadoop** 的部署包中找到。

### 2、启动异常：

```
2016-11-21 12:17:51,419 (SinkRunner-PollingRunner-DefaultSinkProcessor) [ERROR -
org.apache.flume.SinkRunner$PollingRunner.run(SinkRunner.java:158)] Unable to deliver
event. Exception follows.
java.lang.IllegalStateException: Channel closed [channel=ch-tail]. Due to
java.io.IOException: Cannot lock /root/.flume/file-channel/checkpoint. The directory is
already locked. [channel=ch-tail]
```

错误描述为：文件已经被 **lock**，无法继续加锁。解决办法：如果一个 **flume** 中有多个 **channel** 为 **file** 类型，它们应该使用不同的数据目录，通过修改默认配置即可。

### 3、hdfs sink:

`hdfs.fileSuffix` 的值不支持参数化，如果希望在 `fileSuffix` 中使用 `header`，比如 `hdfs.fileSuffix=%{filename}`，后来多次尝试发现 **Flume** 暂时不支持。

4、在 **Spooling** 模式中，已经收集的日志文件，将会被重名为“.COMPLATED”后缀，如果认为的再此创建同名的文件，此时 **Flume** 将会报错且停止采集数据。

### 5、运行时异常:

```
Nov 2016 17:15:04,737 WARN [kafka-producer-network-thread | producer-1]
(org.apache.kafka.clients.NetworkClient$DefaultMetadataUpdater.handleResponse:582) -
Error while fetching metadata with correlation id 96 : {nginx-www=UNKNOWN}
```

出现这种错误的问题，就是 **flume** 无法与 **kafka** 集群建立连接，无法获取 **meta** 信息导致的；通常情况下，你需要修改 **kafka** 中的 `server.properties` 文件，调整“`listeners`”、“`host.name`”配置项即可；其中“`listeners`”中明确指定绑定到本机的内网 IP，“`host.name`”保持默认或者不声明。

## 5. KAFKA

### 基本概念

kafka 是一个分布式的消息缓存系统

kafka 集群中的服务器叫做 broker

kafka 有两种客户端，**producer**（消息生产者），**consumer**（消息消费者），客户端（两种）与 kafka 服务器之间使用 **tcp** 通信

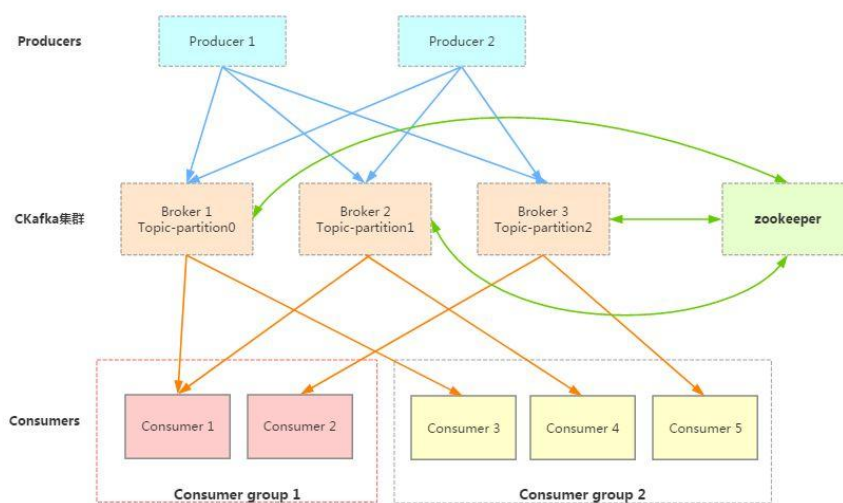
kafka 中不同业务系统的消息可以通过 **topic** 进行区分，而且每一个消息 **topic** 都会被分区，以分担消息读写的负载

每一个分区可以有多个副本，防止数据的丢失

如果某个分区中的数据需要更新，必须通过该分区所有副本中的 **leader** 来更新

消费者可以分组，比如有两个消费者 **AB**，共同消费一个 **topic:testTopic**，**AB** 所消费的消息不会重复，比如 **testTopic** 中有 100 个消息，编号为 0-99，如果 A 消费 0-49，那么 B 就消费 50-99。消费者在消费时可以指定消息的起始偏移量

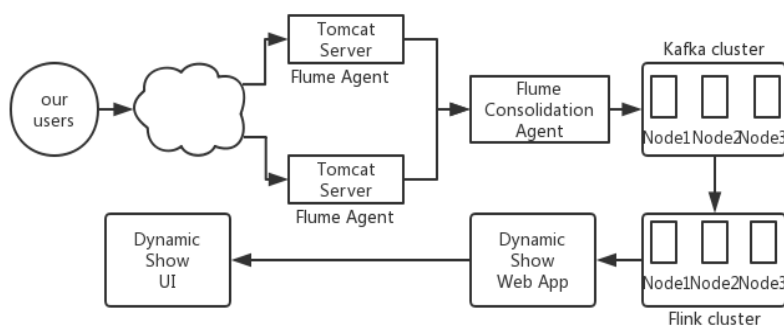
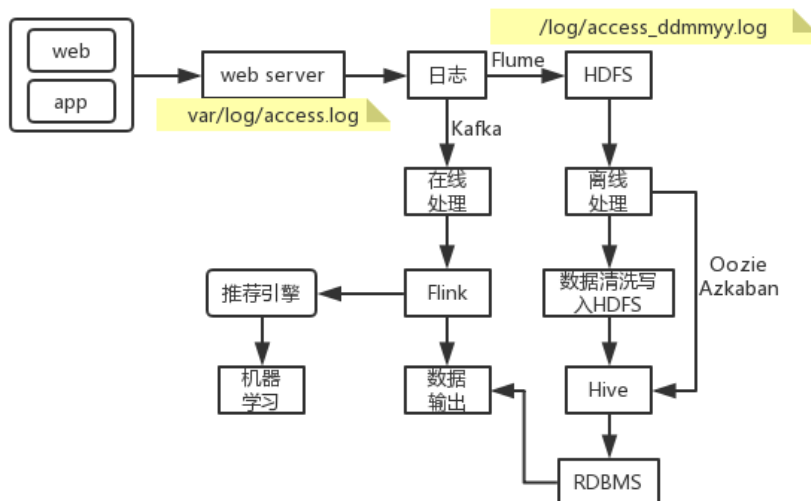
kafka 架构图：



producer 是数据源，比如 flume 架构，consumer 是数据的输出，例如 storm 架构。

Kafka 服务器支持消息的分主题、分区。不同的子系统可以使用不同的主题。分区的意义在于负载均衡

一般性的大数据流处理，还可以包括 **FLINK** 以及 **NOSQL**，如下二图，这里介绍如何集成 **Flume** 和 **Kafka**。 (**Flink** 和 **Redis** 等不在讨论之列，<https://yq.aliyun.com/articles/635327>)



## 6. Kafka 的配置样例 (server.properties) :

broker.id=1

listeners=PLAINTEXT://192.168.1.41:9092

#port=9092

#host.name=192.168.1.41

num.network.threads=3

num.io.threads=8

num.io.threads=8

num.network.threads=8

num.partitions=1

socket.send.buffer.bytes=102400

socket.receive.buffer.bytes=102400

socket.request.max.bytes=104857600

log.dirs=/data/kafka

num.partitions=1

num.recovery.threads.per.data.dir=1

default.replication.factor=2

log.flush.interval.messages=10000

log.flush.interval.ms=1000

```
log.retention.hours=168
log.segment.bytes=1073741824
log.retention.check.interval.ms=300000
```

```
zookeeper.connect=192.168.1.45:2181,192.168.1.44:2181,192.168.1.43:2181,192.168.1.42:2181,192.168.1.41:2181/kafka
zookeeper.connection.timeout.ms=6000
delete.topic.enable=true
min.insync.replicas=1
zookeeper.session.timeout.ms=6000
```

上述配置中,有 2 个地方需要特别注意: **listeners** 和 **host.name**, 我们在 **listeners** 中指定 **kafka** 绑定的地址和端口, 通常为本机的内网 IP, 将 **host.name** 设置为空, 此处如果设置不当, 会导致 **Flume** 无法找到 **kafka** 地址 (**address resolve** 失败); 第二点就是 **zookeeper.connect** 地址, 我们在地址后面增加了 **root path**, 此后 **Flume** 作为 **producer** 端发送消息时, 指定的 **zookeeper** 地址也要带上此 **root path**。此外, 还有一些重要的参数, 比如 **replicas**、**partitions** 等。

## 启动

```
cd ${KAFKA_HOME}
./bin/kafka-server-start.sh -daemon ./config/server.properties > /root/kafka/kafka-logs/logs &
./bin/kafka-server-start.sh ./config/server.properties
```

## kafka shell

创建话题

```
bin/kafka-topics.sh --create --zookeeper master:2181 --replication-factor 3 --partitions 1 --topic mytopics
```

创建的话题名称是有要求的 Due to limitations in metric names, topics with a period ('.') or underscore ('\_') could collide. To avoid issues it is best to use either, but not both.

如果在 **zookeeper** 里指定了 **kafka** 的目录, 例如 **/kafka**, 那么在用 **shell** 进行 **topic** 操作的时候, 需要指定被操作的 **topic** 所属的 **zookeeper** 目录, 例如 **bin/kafka-topics.sh - create - zookeeper master:2181 /kafka - replication-factor 3 - partitions 1 - topic mytopics**。(因为 **kafka** 的集群化是归 **zookeeper** 管的)

列出当前话题

```
bin/ kafka-topics.sh --list --zookeeper master:2181,slave1:2181,slave2:2181
```

删除话题

```
kafka-topics.sh --delete --zookeeper master:2181 --topic mytopics
```

注意由控制台的提示: **Note: This will have no impact if delete.topic.enable is not set to true.** 可知, 需要修改一下 **server.properties** 文件, 在最后一行加上 **delete.topic.enable=true**

创建一个生产者

```
kafka-console-producer.sh --broker-list master:9092 --topic t_test
```

创建一个消费者

```
kafka-console-consumer.sh --zookeeper master:2181 --from-beginning --topic t_test
```

查看话题状态信息

```
kafka-topics.sh --describe --zookeeper master:2181 --topic t_test
```

## 7. ZOOKEEPER

```
/opt/diyu/hadoop-system/zookeeper-3.4.10/conf/zoo.cfg
```

Zookeeper 运行在独立模式下可以很方便的进行测试，评估，研发，但在实际应用中 Zookeeper 运行在所谓的复制模式下，我们把提供相同应用的服务器组称之为一个 quorum，quorum 中的所有机器都有相同的配置文件

```
dataDir=/opt/ diyu /hadoop-system/zookeeper-3.4.10/zk-data
dataLogDir=/opt/ diyu /hadoop-system/zookeeper-3.4.10/zk-logs
server.41=192.168.1.41:2888:3888
server.42=192.168.1.42:2888:3888
server.43=192.168.1.43:2888:3888
server.44=192.168.1.44:2888:3888
server.45=192.168.1.45:2888:3888
~
```

### myid 配置

myid 配置，五台服务器分别为（对应上面的 server.4\*的后面数字）

在 dataDir 所定义的目录下新建 myid 文件，本例中在/opt/diyu/hadoop-system/zookeeper-3.4.10/zk-data 下新建 myid 文件[b][color]，填入各主机之 ID。如 192.168.1.41 机器的 myid 文件内容为 41。

dataDir=\$Hadoop\_home/hadoop/zookeeper-3.4.3/data 后面不能有空格其次是 后面不能跟注释 不然创建的文件夹名字是 data#注释

配置说明：

tickTime: 这个时间是作为 Zookeeper 服务器之间或客户端与服务器之间维持心跳的时间间隔，也就是每个 tickTime 时间就会发送一个心跳。

dataDir: 顾名思义就是 Zookeeper 保存数据的目录，默认情况下，Zookeeper 将写数据的日志文件也保存在这个目录里。

clientPort: 这个端口就是客户端连接 Zookeeper 服务器的端口，Zookeeper 会监听这个端口，接受客户端的访问请求。

### 启动 zookeeper

```
netstat -at|grep 2181      #查看 zookeeper 端口
netstat -nat               #查看端口信息
./zkServer.sh start        #启动
bin/zkCli.sh -server 127.0.0.1:2181 （查看每个机器里面的端口号）
```

```
[root@testing1.r1.test43 zookeeper-3.4.10]$bin/zkCli.sh -server localhost:2183
```

Welcome to ZooKeeper!

```
2019-06-13 12:26:52,425 [myid:] - INFO
```

```
[main-SendThread(localhost:2183):ClientCnxn$SendThread@1032] - Opening socket connection to
server localhost/127.0.0.1:2183. Will not attempt to authenticate using SASL (unknown error)
```

```
JLine support is enabled
```

```
2019-06-13 12:26:52,502 [myid:] - INFO
```

```
[main-SendThread(localhost:2183):ClientCnxn$SendThread@876] - Socket connection established
to localhost/127.0.0.1:2183, initiating session
```

```
2019-06-13 12:26:52,532 [myid:] - INFO
[main-SendThread(localhost:2183):ClientCnxn$SendThread@1299] - Session establishment
complete on server localhost/127.0.0.1:2183, sessionId = 0x2b6b4f1080670000, negotiated timeout
= 30000
```

WATCHER::

```
WatchedEvent state:SyncConnected type:None path:null
```

```
[zk: localhost:2183(CONNECTED) 0]
```

每个机器分别启动，查看：

```
jps #查看启动的服务名称
```

```
./zkServer.sh stop #关闭
```

Disable IPV6

编辑文件/etc/sysctl.conf

```
vi /etc/sysctl.conf
```

添加下面的行：

```
net.ipv6.conf.all.disable_ipv6 = 1
```

```
net.ipv6.conf.default.disable_ipv6 = 1
```

如果你想要为特定的网卡禁止 IPv6，比如，对于 enp0s3，添加下面的行。

```
net.ipv6.conf.enp0s3.disable_ipv6 = 1
```

保存并退出文件。

执行下面的命令来使设置生效。

```
sysctl -p
```

```
368 sysctl -w net.ipv6.conf.all.disable_ipv6=1
```

```
369 sysctl -w net.ipv6.conf.default.disable_ipv6=1
```

```
370 vi /etc/sysctl.conf
```

```
371 sysctl -p
```

扫描指定文件的对接例子：

```
agent.sources.s1.type=exec
```

```
agent.sources.s1.command=tail -F /Users/it-od-m/Downloads/abc.log agent.sources.s1.channels=c1
```

```
agent.channels.c1.type=memory
```

```
agent.channels.c1.capacity=10000
```

```
agent.channels.c1.transactionCapacity=100
```

#设置 Kafka 接收器

```
agent.sinks.k1.type= org.apache.flume.sink.kafka.KafkaSink
```

#设置 Kafka 的 broker 地址和端口号

```
agent.sinks.k1.brokerList=127.0.0.1:9092
```

```
#设置 Kafka 的 Topic
agent.sinks.k1.topic=testKJ1
#设置序列化方式
agent.sinks.k1.serializer.class=kafka.serializer.StringEncoder
agent.sinks.k1.channel=c1
```

使用如下命令启动 Flume:

```
./bin/flume-ng agent -n agent -c conf -f conf/hw.conf -Dflume.root.logger=INFO,console
```

最后一行显示 Component type:SINK,name:k1 started 表示启动成功。

在启动 Flume 之前, Zookeeper 和 Kafka 要先启动成功, 不然启动 Flume 会报连不上 Kafka 的错误。

、使用 `./zkServer.sh start` 启动 zookeeper。

2、使用如下命令启动 Kafka, 更详细的 Kafka 命令请参照我之前总结的 <http://www.jianshu.com/p/cfedb7122e38> (Kafka 常用命令行总结)

```
./kafka-server-start.sh -daemon ../config/server.properties
```

3、使用 Kafka 默认提供的 Consumer 来接收消息

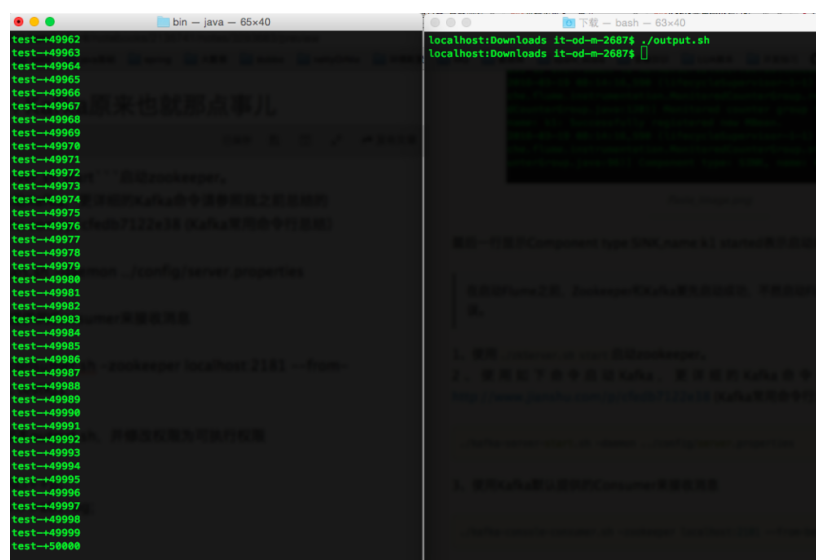
```
./kafka-console-consumer.sh -zookeeper localhost:2181 --from-beginning --topic testKJ1
```

4、编写简单 Shell 脚本 output.sh, 并修改权限为可执行权限

```
for((i=0;i<=50000;i++));
do echo "test-"+$i>>abc.log;
done
```

循环向 abc.log 文件插入 test 文本消息。

5、执行 output.sh。

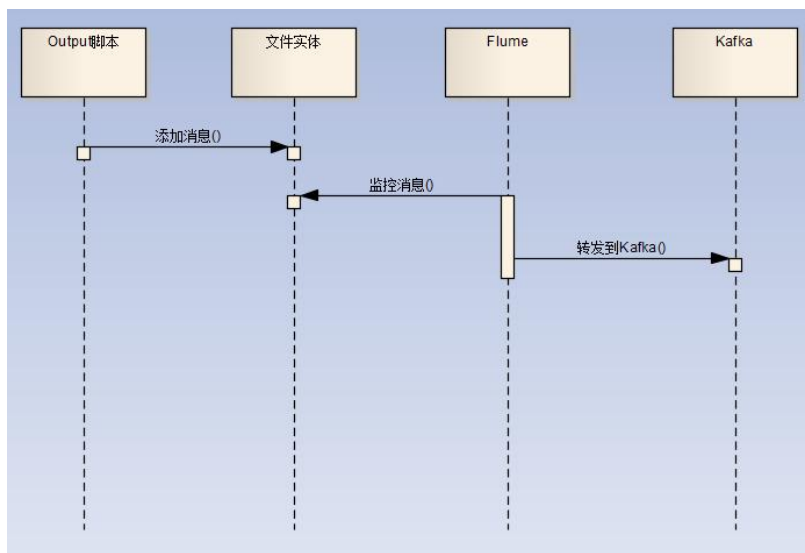


```
test--49982
test--49983
test--49984
test--49985
test--49986
test--49987
test--49988
test--49989
test--49990
test--49991
test--49992
test--49993
test--49994
test--49995
test--49996
test--49997
test--49998
test--49999
test--50000

localhost:Downloads it-od-m-2687$ ./output.sh
localhost:Downloads it-od-m-2687$
```

整个流程如下:





## 1. 配置 flume , 在 flume 的 conf 目录下新建文件 ( flume\_kafka\*.conf ) 并配置。

```
#####
##主要作用是监听目录中的新增数据，采集到数据之后，输出到 kafka
## 注意：Flume agent 的运行，主要就是配置 source channel sink
## 下面的 a1 就是 agent 的代号，source 叫 r1 channel 叫 c1 sink 叫 k1
#####
```

```
a1.sources = r1
a1.sinks = k1
a1.channels = c1
```

#具体定义 source

```
a1.sources.r1.type = spooldir -----》有几种方式
```

#先创建此目录，保证里面空的

```
a1.sources.r1.spoolDir = /tmp/log/spooldir/hadoop.log
```

#sink 到 kafka 里面

```
a1.sinks.k1.channel = c1
```

```
a1.sinks.k1.type = org.apache.flume.sink.kafka.KafkaSink
```

#设置 Kafka 的 Topic

```
a1.sinks.k1.kafka.topic = test3
```

#设置 Kafka 的 broker 地址和端口号

```
a1.sinks.k1.kafka.bootstrap.servers
```

```
192.168.1.41:9092,192.168.1.42:9092,192.168.1.43:9092,192.168.1.44:9092,192.168.1.45:9092,
```

#配置批量提交的数量

```
a1.sinks.k1.kafka.flumeBatchSize = 20
```

```
a1.sinks.k1.kafka.producer.acks = 1
```

```
a1.sinks.k1.kafka.producer.linger.ms = 1
```

```
a1.sinks.ki.kafka.producer.compression.type= snappy
```

#对于 channel 的配置描述 使用文件做数据的临时缓存 这种的安全性要高

```
a1.channels.c1.type = file
a1.channels.c1.checkpointDir = /tmp/uplooking/data/flume/checkpoint
a1.channels.c1.dataDirs = /tmp/uplooking/data/flume/data
```

```
#通过 channel c1 将 source r1 和 sink k1 关联起来
a1.sources.r1.channels = c1
a1.sinks.k1.channel = c1
```

## 2. 启动

### 2.1.启动 Zookeeper(5 台)

```
[root@master bin]# ./zkServer.sh start
```

### 2.2.再启动 kafka(5 台)

```
cd /opt/diyu/hadoop-system/kafka_2.11-1.0.1/
./bin/kafka-server-start.sh -daemon config/server.properties &
```

### 2.3.如果没有主题创建主题

```
cd /opt/diyu/hadoop-system/kafka_2.11-1.0.1/
./bin/kafka-topics.sh --create --zookeeper 192.168.1.41:2181, 192.168.1.42:2181, 192.168.1.43:2181
192.168.1.44:2181, 192.168.1.45:2181, --replication-factor 3 --partitions 3 --topic test3
```

### 2.4.启动一个该主题的消费者

```
cd /opt/diyu/hadoop-system/kafka_2.11-1.0.1/
./bin/kafka-console-consumer.sh --bootstrap-server 192.168.1.41:9092, 192.168.1.42:9092, 192.168.1.43:9092
192.168.1.44:9092 192.168.1.45:9092 --from-beginning --topic test3
```

### 2.5.启动 flume

```
cd /opt/diyu/hadoop-system/apache-flume-1.8.0-bin/
./bin/flume-ng agent -n a1 -c conf -f conf/flume-kafka1.conf -Dflume.root.logger=INFO,console
```

### 2.6. 向 flume 监听目录里面添加内容，观察消费者

flume-kafka2.conf ， flume 里用了 EXEC 命令的方式, tail -f Hadoop.log。

发送消息：

```
/kafka-console-producer.sh --broker-list localhost:9092 --topic
test3
```

这里不用，因为消息是从 FLUME 过来的。

查看 Topic:

```
[root@diyu41 kafka_2.11-1.0.1]$bin/kafka-topics.sh --list --zookeeper localhost:2181
test3
[root@diyu41 kafka_2.11-1.0.1]$
```

```
[root@diyu41 kafka_2.11-1.0.1]$bin/kafka-console-consumer.sh --zookeeper localhost:2181 --topic test3 --from-beginning
2019-06-13 12:07:15 267090 DEBUG hadoop--default--IPC Server handler 47 on 8031: org.apache.hadoop.yarn.server.api.ResourceTrackerPB.nodeHeartbeat from 192.168.1.43:52756 Call#250052
Retry#0 for RpcKind RPC_PROTOCOL_BUFFER
2019-06-13 12:07:15 267191 DEBUG hadoop--default--Served: nodeHeartbeat queueTime= 1 procesingTime= 0
2019-06-13 12:07:15 517382 DEBUG hadoop--default-- got #249921
2019-06-13 12:07:15 517538 DEBUG hadoop--default--Served: nodeHeartbeat queueTime= 0 procesingTime= 0
2019-06-13 12:07:15 530514 DEBUG hadoop--default--IPC Client (112200409) connection to /192.168.1.41:8031 from root sending #250106
2019-06-13 12:07:15 530662 DEBUG hadoop--default--PrivilegedAction as:root (auth:SIMPLE) from:org.apache.hadoop.ipc.Server$Handler.run(Server.java:2045)
2019-06-13 12:07:15 530944 DEBUG hadoop--default--IPC Client (112200409) connection to /192.168.1.41:8031 from root got value #250106
Processed a total of 12230 messages
```

Another Simple example:

```
$ bin/flume-ng agent --conf conf --conf-file conf/example.conf --name a1
-Dflume.root.logger=INFO,console
```

```
$ telnet localhost 44444
```

```
Trying 127.0.0.1...
Connected to localhost.localdomain (127.0.0.1).
Escape character is '^]'.
Hello world! <ENTER>
OK
```

```
2019-06-14      14:25:15,012      (SinkRunner-PollingRunner-DefaultSinkProcessor)      [INFO      -
org.apache.flume.sink.LoggerSink.process(LoggerSink.java:95)] Event: { headers:{} body: 48 65 6C 6C 6F 20 57 6F 72 6C 64
21 0D      Hello World!. }
```

More examples of flume:

<https://www.cnblogs.com/ciade/p/5495218.html>