xrzs 关注此人

关注(2) 粉丝(941) 积分(429)
1）Writing is thinking. 2）If you're interesting someone won't agree with what you said.
.发送私信 .请教问题

博客分类

- java(43)
- javame/android(1)
- python(57)
- perl(2)
- scala(1)
- awk/shell(24)
- sqlite(1)
- sql技巧(6)
- mysql(98)
- oracle(4)
- tcp/ip/http(11)
- JavaNet/Multithread(5)
- network(16)
- linux/unix(38)
- windows/office(5)
- php/nginx/apache(14)
- jsp/mvc/ssh(3)
- wsgi/Flask(7)
- js/ajax/extjs/jquery(12)
- hbase(12)
- pig(6)
- NoSQL(12)
- hadoop(39)
- hadoop生态圈(3)
- spark(5)
- hive(30)
- BI/DW/DP(5)
- DA/DM/ML(13)
- algorithm(7)
- data analysis theory (27)
- 日志采集与架构(5)
- 职场 · 杂谈(38)
- 管理之道(7)

- 工作笔记(19)
- resource(1)
- QA(2)
- 金融 · 理财(1)

阅读排行

1. 1. 关于 python ImportError: No module named 的问题
2. 2. 循序渐进Java Socket网络编程（多客户端、信息共享、文件传输）
3. 3. Flume NG 简介及配置实战
4. 4. Eclipse 下找不到或无法加载主类的解决办法
5. 5. 动态规划算法之：最长公共子序列 & 最长公共子串（LCS）
6. 6. 关于 HTTP GET/POST 请求参数长度最大值的一个理解误区
7. 7. storm 原理简介及单机版安装指南
8. 8. python 日志模块 logging 详解

最新评论

- @河边沉思：学习了 查看»
- @赛克蓝德：可以直接用工具Piwik等 查看»
- @PREPOET：后面部分完全一样 查看»
- @素人派surenpi：学习了 http://surenpi.com 查看»
- @Json_xu：枚举的不错 查看»
- @just4scala：写的真好 查看»
- @kyle_wang： 查看»
- @lyhabc：rsync报错protocol version mismatch is your s... 查看»
- @Leon_wy：2 查看»
- @进击的柯南：整理的很系统，多谢 查看»

访客统计

- 今日访问：644
- 昨日访问：1534
- 本周访问：5206
- 本月访问：3712
- 所有访问：742797

空间 » 博客 » hadoop生态圈

原 Flume NG 简介及配置实战

发表于2年前(2014-07-08 01:46) 阅读（28709）| 评论（9）60人收藏此文章, 我要收藏

赞20

目录[-]

- 1、Flume 的一些核心概念：
- 1.1 数据流模型
- 1.2 高可靠性

- 1.3 可恢复性
- 2、Flume 整体架构介绍
- 2.1 Exec source
- 2.2 Spooling Directory Source
- 3、常用架构、功能配置示例
- 3.1 先来个简单的：单节点 Flume 配置
- 3.2 单节点 Flume 直接写入 HDFS
- 3.3 来一个常见架构：多 agent 汇聚写入 HDFS
- 3.3.1 在各个webserv日志机上配置 Flume Client
- 3.3.2 在汇聚节点配置 Flume server
- 4、可能遇到的问题：
- 4.1 OOM 问题：
- 4.2 JDK 版本不兼容问题：
- 4.3 小文件写入 HDFS 延时的问题
- 4.4 数据重复写入、丢失问题
- 4.5 tail 断点续传的问题：
- 4.6 在 Flume 中如何修改、丢弃、按预定义规则分类存储数据？
- 5、Refer：

Flume 作为 cloudera 开发的实时日志收集系统，受到了业界的认可与广泛应用。Flume 初始的发行版本目前被统称为 Flume OG（original generation），属于 cloudera。但随着 FLume 功能的扩展，Flume OG 代码工程臃肿、核心组件设计不合理、核心配置不标准等缺点暴露出来，尤其是在 Flume OG 的最后一个发行版本 0.94.0 中，日志传输不稳定的现象尤为严重，为了解决这些问题，2011 年 10 月 22 号，cloudera 完成了 Flume-728，对 Flume 进行了里程碑式的改动：重构核心组件、核心配置以及代码架构，重构后的版本统称为 Flume NG（next generation）；改动的另一原因是将 Flume 纳入 apache 旗下，cloudera Flume 改名为 Apache Flume。IBM 的这篇文章：《Flume NG：Flume 发展史上的第一次革命》，从基本组件以及用户体验的角度阐述 Flume OG 到 Flume NG 发生的革命性变化。本文就不再赘述各种细枝末节了，不过这里还是简要提下 Flume NG （1.x.x）的主要变化：

- sources和sinks 使用channels 进行链接
- 两个主要channel 。1， in-memory channel 非持久性支持，速度快。2 ， JDBC-based channel 持久性支持。
- 不再区分逻辑和物理node，所有物理节点统称为 “agents”,每个agents 都能运行0个或多个 sources 和sinks
- 不再需要master节点和对zookeeper的依赖，配置文件简单化。
- 插件化，一部分面对用户，工具或系统开发人员。
- 使用Thrift、Avro Flume sources 可以从flume0.9.4 发送 events 到flume 1.x

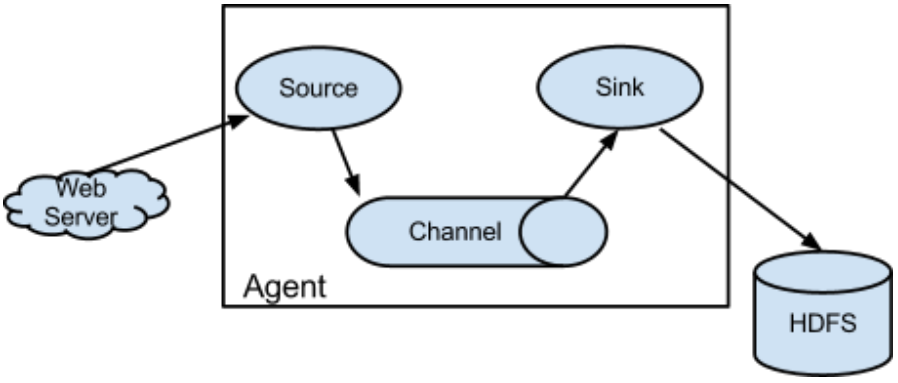
注：本文所使用的 Flume 版本为 flume-1.4.0-cdh4.7.0，不需要额外的安装过程，解压缩即可用。

1、Flume 的一些核心概念：

组件	功能
Agent	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	生产数据，运行在一个独立的线程。
Source	从Client收集数据，传递给Channel。
Sink	从Channel收集数据，运行在一个独立线程。
Channel	连接 sources 和 sinks ，这个有点像一个队列。
Events	可以是日志记录、 avro 对象等。

1.1 数据流模型

Flume以agent为最小的独立运行单位。一个agent就是一个JVM。单agent由Source、Sink和Channel三大组件构成，如下图：

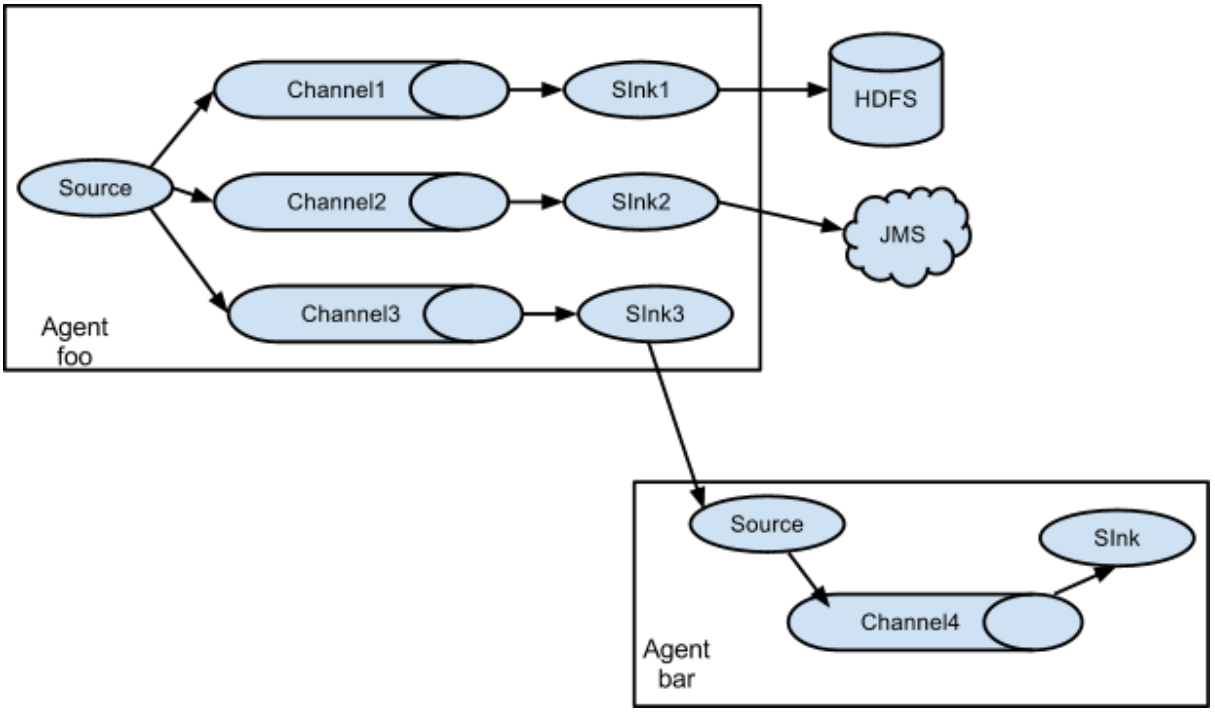


图一

Flume的数据流由事件(Event)贯穿始终。事件是Flume的基本数据单位，它携带日志数据(字节数组形式)并且携带有头信息，这些Event由Agent外部的Source，比如上图中的Web Server生成。当Source捕获事件后会进行特定的格式化，然后Source会把事件推入(单个或多个)Channel中。你可以把Channel看作是一个缓冲区，它将保存事件直到Sink处理完该事件。Sink负责持久化日志或者把事件推向另一个Source。

很直白的设计，其中值得注意的是，Flume提供了大量内置的Source、Channel和Sink类型。不同类型的Source,Channel和Sink可以自由组合。组合方式基于用户设置的配置文件，非常灵活。比如：Channel可以把事件暂存在内存里，也可以持久化到本地硬盘上。Sink可以把日志写入HDFS, HBase，甚至是另外一个Source等等。

如果你以为Flume就这些能耐那就大错特错了。Flume支持用户建立多级流，也就是说，多个agent可以协同工作，并且支持Fan-in、Fan-out、Contextual Routing、Backup Routes。如下图所示：



1.2 高可靠性

作为生产环境运行的软件，高可靠性是必须的。

从单agent来看，Flume使用基于事务的数据传递方式来保证事件传递的可靠性。Source和Sink被封装进一个事务。事件被存放在Channel中直到该事件被处理，Channel中的事件才会被移除。这是Flume提供的点到点的可靠机制。
从多级流来看，前一个agent的sink和后一个agent的source同样有它们的事务来保障数据的可靠性。

1.3 可恢复性

还是靠Channel。推荐使用FileChannel，事件持久化在本地文件系统里(性能较差)。

2、Flume 整体架构介绍

Flume架构整体上看就是 source-->channel-->sink 的三层架构（参见最上面的 图一），类似生成者和消费者的架构，他们之间通过queue（channel）传输，解耦。

Source:完成对日志数据的收集，分成 transtion 和 event 打入到channel之中。
Channel:主要提供一个队列的功能，对source提供中的数据进行简单的缓存。
Sink:取出Channel中的数据，进行相应的存储文件系统，数据库，或者提交到远程服务器。
对现有程序改动最小的使用方式是使用是直接读取程序原来记录的日志文件，基本可以实现无缝接入，不需要对现有程序进行任何改动。
对于直接读取文件Source, 主要有两种方式：

2.1 Exec source

可通过写Unix command的方式组织数据，最常用的就是tail -F [file]。
可以实现实时传输，但在flume不运行和脚本错误时，会丢数据，也不支持断点续传功能。因为没有记录上次文件读到的位置，从而没办法知道，下次再读时，从什么地方开始读。特别是在日志文件一直在增加的时候。flume的source挂了。等flume的source再次开启的这段时间内，增加的日志内容，就没办法被source读取到了。不过flume有一个execStream的扩展，可以自己写一个监控日志增加情况，把增加的日志，通过自己写的工具把增加的内容，传送给flume的node。再传送给sink的node。要是能在tail类的source中能支持，在node挂掉这段时间的内容，等下次node开启后在继续传送，那就更完美了。

2.2 Spooling Directory Source

SpoolSource:是监测配置的目录下新增的文件，并将文件中的数据读取出来，可实现准实时。需要注意两点：1、拷贝到spool目录下的文件不可以再打开编辑。2、spool目录下不可包含相应的子目录。在实际使用的过程中，可以结合log4j使用，使用log4j的时候，将log4j的文件分割机制设为1分钟一次，将文件拷贝到spool的监控目录。log4j有一个TimeRolling的插件，可以把log4j分割的文件到spool目录。基本实现了实时的监控。Flume在传完文件之后，将会修改文件的后缀，变为.COMPLETED（后缀也可以在配置文件中灵活指定）
ExecSource，SpoolSource对比：ExecSource可以实现对日志的实时收集，但是存在Flume不运行或者指令执行出错时，将无法收集到日志数据，无法何证日志数据的完整性。SpoolSource虽然无法实现实时的收集数据，但是可以使用以分钟的方式分割文件，趋近于实时。如果应用无法实现以分钟切割日志文件的话，可以两种收集方式结合使用。
Channel有多种方式：有MemoryChannel, JDBC Channel, MemoryRecoverChannel, FileChannel。MemoryChannel可以实现高速的吞吐，但是无法保证数据的完整性。

MemoryRecoverChannel在官方文档的建议上已经建义使用FileChannel来替换。FileChannel保证数据的完整性与一致性。在具体配置FileChannel时，建议FileChannel设置的目录和程序日志文件保存的目录设成不同的磁盘，以便提高效率。
Sink在设置存储数据时，可以向文件系统中，数据库中，hadoop中储数据，在日志数据较少时，可以将数据存储在文件系中，并且设定一定的时间间隔保存数据。在日志数据较多时，可以将相应的日志数据存储到Hadoop中，便于日后进行相应的数据分析。

3、常用架构、功能配置示例

3.1 先来个简单的：单节点 Flume 配置

```
# example.conf: A single-node Flume configuration

# Name the components on this agent
a1.sources = r1
a1.sinks = k1
a1.channels = c1

# Describe/configure the source
a1.sources.r1.type = netcat
a1.sources.r1.bind = localhost
a1.sources.r1.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

# Bind the source and sink to the channel
a1.sources.r1.channels = c1
a1.sinks.k1.channel = c1
```

将上述配置存为：example.conf

然后我们就可以启动 Flume 了：

```
bin/flume-ng agent --conf conf --conf-file example.conf --name a1 -Dflume.r
```

PS：-Dflume.root.logger=INFO,console 仅为 debug 使用，请勿生产环境生搬硬套，否则大量的日志会返回到终端。。。

-c/--conf 后跟配置目录，-f/--conf-file 后跟具体的配置文件，-n/--name 指定agent的名称

然后我们再开一个 shell 终端窗口，telnet 上配置中侦听的端口，就可以发消息看到效果了：

```
$ telnet localhost 44444
Trying 127.0.0.1...
Connected to localhost.localdomain (127.0.0.1).
Escape character is '^]'.
Hello world! <ENTER>
OK
```

Flume 终端窗口此时会打印出如下信息，就表示成功了：

```
12/06/19 15:32:19 INFO source.NetcatSource: Source starting
12/06/19 15:32:19 INFO source.NetcatSource: Created serverSocket:sun.nio.ch
12/06/19 15:32:34 INFO sink.LoggerSink: Event: { headers:{} body: 48 65 6C
```

至此，咱们的第一个 Flume Agent 算是部署成功了！

3.2 单节点 Flume 直接写入 HDFS

```
# Define a memory channel called ch1 on agent1
agent1.channels.ch1.type = memory
agent1.channels.ch1.capacity = 100000
agent1.channels.ch1.transactionCapacity = 100000
agent1.channels.ch1.keep-alive = 30

# Define an Avro source called avro-source1 on agent1 and tell it
# to bind to 0.0.0.0:41414. Connect it to channel ch1.
#agent1.sources.avro-source1.channels = ch1
#agent1.sources.avro-source1.type = avro
#agent1.sources.avro-source1.bind = 0.0.0.0
#agent1.sources.avro-source1.port = 41414
#agent1.sources.avro-source1.threads = 5

#define source monitor a file
agent1.sources.avro-source1.type = exec
agent1.sources.avro-source1.shell = /bin/bash -c
agent1.sources.avro-source1.command = tail -n +0 -F /home/storm/tmp/id.txt
agent1.sources.avro-source1.channels = ch1
agent1.sources.avro-source1.threads = 5

# Define a logger sink that simply logs all events it receives
# and connect it to the other end of the same channel.
agent1.sinks.log-sink1.channel = ch1
agent1.sinks.log-sink1.type = hdfs
agent1.sinks.log-sink1.hdfs.path = hdfs://192.168.1.111:8020/flumeTest
agent1.sinks.log-sink1.hdfs.writeFormat = Text
agent1.sinks.log-sink1.hdfs.fileType = DataStream
agent1.sinks.log-sink1.hdfs.rollInterval = 0
agent1.sinks.log-sink1.hdfs.rollSize = 1000000
agent1.sinks.log-sink1.hdfs.rollCount = 0
agent1.sinks.log-sink1.hdfs.batchSize = 1000
agent1.sinks.log-sink1.hdfs.txnEventMax = 1000
agent1.sinks.log-sink1.hdfs.callTimeout = 60000
agent1.sinks.log-sink1.hdfs.appendTimeout = 60000

# Finally, now that we've defined all of our components, tell
# agent1 which ones we want to activate.
agent1.channels = ch1
agent1.sources = avro-source1
agent1.sinks = log-sink1
```

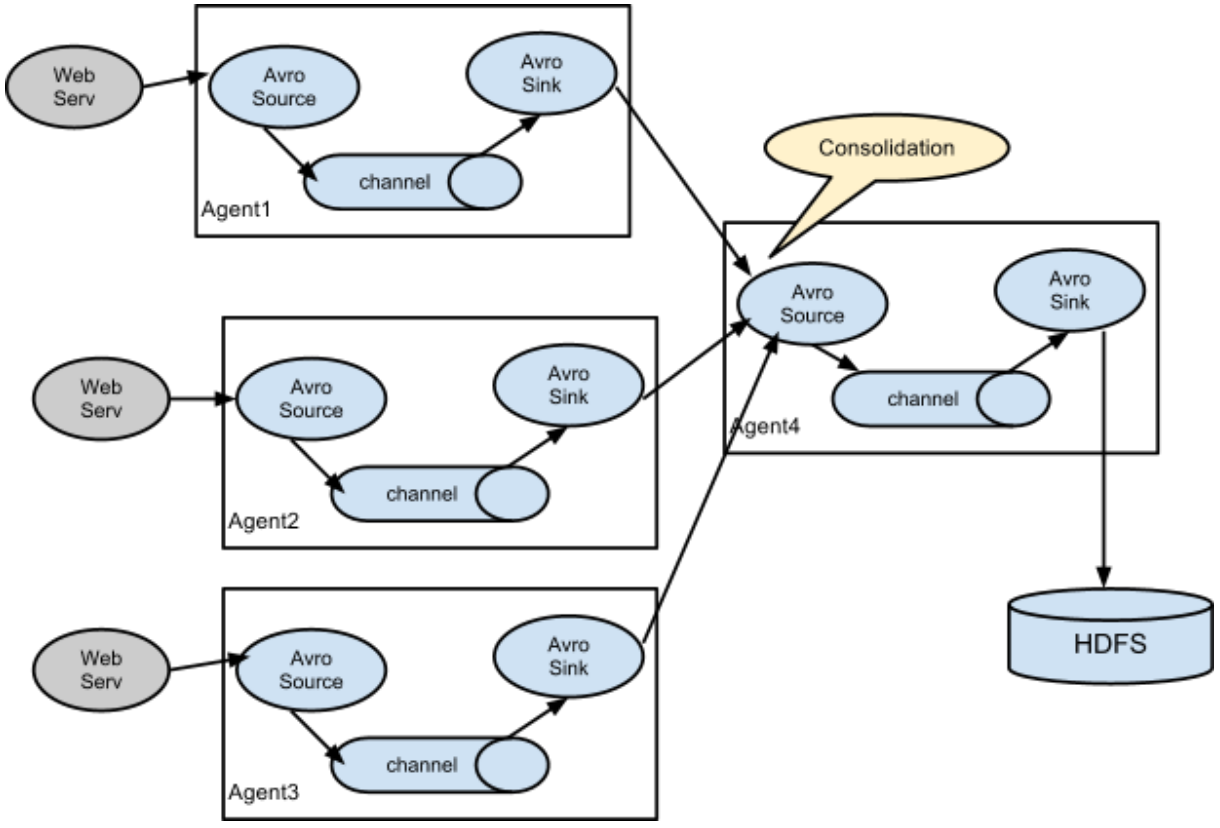
启动如下命令，就可以在 hdfs 上看到效果了。

../bin/flume-ng agent --conf ../conf/ -f flume_directHDFS.conf -n agent1 -Dflume.root.logger=INFO,console

PS：实际环境中有这样的需求，通过在多个agent端tail日志，发送给collector，collector再把数据收集，统一发送给HDFS存储起来，当HDFS文件大小超过一定的大小或者超过在规定的时间间隔会生成一个文件。

Flume 实现了两个Trigger，分别为SizeTriger（在调用HDFS输出流写的同时，count该流已经写入的大小总和，若超过一定大小，则创建新的文件和输出流，写入操作指向新的输出流，同时close以前的输出流）和TimeTriger（开启定时器，当到达该点时，自动创建新的文件和输出流，新的写入重定向到该流中，同时close以前的输出流）。

3.3 来一个常见架构：多 agent 汇聚写入 HDFS



3.3.1 在各个webserv日志机上配置 Flume Client

```
# clientMainAgent
clientMainAgent.channels = c1
clientMainAgent.sources  = s1
clientMainAgent.sinks    = k1 k2
# clientMainAgent sinks group
clientMainAgent.sinkgroups = g1
# clientMainAgent Spooling Directory Source
clientMainAgent.sources.s1.type = spooldir
clientMainAgent.sources.s1.spoolDir  =/dsap/rawdata/
clientMainAgent.sources.s1.fileHeader = true
clientMainAgent.sources.s1.deletePolicy =immediate
clientMainAgent.sources.s1.batchSize =1000
clientMainAgent.sources.s1.channels =c1
clientMainAgent.sources.s1.deserializer.maxLineLength =1048576
# clientMainAgent FileChannel
clientMainAgent.channels.c1.type = file
clientMainAgent.channels.c1.checkpointDir = /var/flume/fchannel/spool/che
clientMainAgent.channels.c1.dataDirs = /var/flume/fchannel/spool/data
clientMainAgent.channels.c1.capacity = 200000000
clientMainAgent.channels.c1.keep-alive = 30
clientMainAgent.channels.c1.write-timeout = 30
clientMainAgent.channels.c1.checkpoint-timeout=600
# clientMainAgent Sinks
```

```
# k1 sink
clientMainAgent.sinks.k1.channel = c1
clientMainAgent.sinks.k1.type = avro
# connect to CollectorMainAgent
clientMainAgent.sinks.k1.hostname = flume115
clientMainAgent.sinks.k1.port = 41415
# k2 sink
clientMainAgent.sinks.k2.channel = c1
clientMainAgent.sinks.k2.type = avro
# connect to CollectorBackupAgent
clientMainAgent.sinks.k2.hostname = flume116
clientMainAgent.sinks.k2.port = 41415
# clientMainAgent sinks group
clientMainAgent.sinkgroups.g1.sinks = k1 k2
# load_balance type
clientMainAgent.sinkgroups.g1.processor.type = load_balance
clientMainAgent.sinkgroups.g1.processor.backoff   = true
clientMainAgent.sinkgroups.g1.processor.selector  = random
```

../bin/flume-ng agent --conf ../conf/ -f flume_Consolidation.conf -n clientMainAgent -Dflume.root.logger=DEBUG,console

3.3.2 在汇聚节点配置 Flume server

```
# collectorMainAgent
collectorMainAgent.channels = c2
collectorMainAgent.sources  = s2
collectorMainAgent.sinks    =k1 k2
# collectorMainAgent AvroSource
#
collectorMainAgent.sources.s2.type = avro
collectorMainAgent.sources.s2.bind = flume115
collectorMainAgent.sources.s2.port = 41415
collectorMainAgent.sources.s2.channels = c2

# collectorMainAgent FileChannel
#
collectorMainAgent.channels.c2.type = file
collectorMainAgent.channels.c2.checkpointDir =/opt/var/flume/fchannel/spool
collectorMainAgent.channels.c2.dataDirs = /opt/var/flume/fchannel/spool/dat
collectorMainAgent.channels.c2.capacity = 200000000
collectorMainAgent.channels.c2.transactionCapacity=6000
collectorMainAgent.channels.c2.checkpointInterval=60000
# collectorMainAgent hdfsSink
collectorMainAgent.sinks.k2.type = hdfs
collectorMainAgent.sinks.k2.channel = c2
collectorMainAgent.sinks.k2.hdfs.path = hdfs://db-cdh-cluster/flume%{dir}
collectorMainAgent.sinks.k2.hdfs.filePrefix =k2_%{file}
collectorMainAgent.sinks.k2.hdfs.inUsePrefix =_
collectorMainAgent.sinks.k2.hdfs.inUseSuffix =.tmp
collectorMainAgent.sinks.k2.hdfs.rollSize = 0
collectorMainAgent.sinks.k2.hdfs.rollCount = 0
collectorMainAgent.sinks.k2.hdfs.rollInterval = 240
collectorMainAgent.sinks.k2.hdfs.writeFormat = Text
collectorMainAgent.sinks.k2.hdfs.fileType = DataStream
collectorMainAgent.sinks.k2.hdfs.batchSize = 6000
collectorMainAgent.sinks.k2.hdfs.callTimeout = 60000
collectorMainAgent.sinks.k1.type = hdfs
collectorMainAgent.sinks.k1.channel = c2
collectorMainAgent.sinks.k1.hdfs.path = hdfs://db-cdh-cluster/flume%{dir}
collectorMainAgent.sinks.k1.hdfs.filePrefix =k1_%{file}
```

```
collectorMainAgent.sinks.k1.hdfs.inUsePrefix =_
collectorMainAgent.sinks.k1.hdfs.inUseSuffix =.tmp
collectorMainAgent.sinks.k1.hdfs.rollSize = 0
collectorMainAgent.sinks.k1.hdfs.rollCount = 0
collectorMainAgent.sinks.k1.hdfs.rollInterval = 240
collectorMainAgent.sinks.k1.hdfs.writeFormat = Text
collectorMainAgent.sinks.k1.hdfs.fileType = DataStream
collectorMainAgent.sinks.k1.hdfs.batchSize = 6000
collectorMainAgent.sinks.k1.hdfs.callTimeout = 60000
```

../bin/flume-ng agent --conf ../conf/ -f flume_Consolidation.conf -n collectorMainAgent -Dflume.root.logger=DEBUG,console

上面采用的就是类似 cs 架构，各个 flume agent 节点先将各台机器的日志汇总到 Consolidation 节点，然后再由这些节点统一写入 HDFS，并且采用了负载均衡的方式，你还可以配置高可用的模式等等。

4、可能遇到的问题：

4.1 OOM 问题：

```
flume 报错:
java.lang.OutOfMemoryError: GC overhead limit exceeded
或者:
java.lang.OutOfMemoryError: Java heap space
Exception in thread "SinkRunner-PollingRunner-DefaultSinkProcessor" java.la
```

Flume 启动时的最大堆内存大小默认是 20M，线上环境很容易 OOM，因此需要你在 flume-env.sh 中添加 JVM 启动参数:

```
JAVA_OPTS="-Xms8192m -Xmx8192m -Xss256k -Xmn2g -XX:+UseParNewGC -XX:+UseC
```

然后在启动 agent 的时候一定要带上 -c conf 选项，否则 flume-env.sh 里配置的环境变量不会被加载生效。

具体参见：

http://stackoverflow.com/questions/1393486/error-java-lang-outofmemoryerror-gc-overhead-limit-exceeded

http://marc.info/?l=flume-user&m=138933303305433&w=2

4.2 JDK 版本不兼容问题：

```
2014-07-07 14:44:17,902 (agent-shutdown-hook) [WARN - org.apache.flume.sink
java.lang.UnsupportedOperationException: This is supposed to be overridden by
        at com.google.protobuf.GeneratedMessage.getUnknownFields(GeneratedMes
        at org.apache.hadoop.hdfs.protocol.proto.ClientNamenodeProtocolProtos
        at com.google.protobuf.AbstractMessageLite.toByteString(AbstractMessa
        at org.apache.hadoop.ipc.ProtobufRpcEngine$Invoker.constructRpcReques
        at org.apache.hadoop.ipc.ProtobufRpcEngine$Invoker.invoke(ProtobufRpc
```

把你的 jdk7 换成 jdk6 试试。

4.3 小文件写入 HDFS 延时的问题

其实上面 3.2 中已有说明，flume 的 sink 已经实现了几种最主要的持久化触发器：

比如按大小、按间隔时间、按消息条数等等，针对你的文件过小迟迟没法写入 HDFS 持久化的问题，

那是因为你此时还没有满足持久化的条件，比如你的行数还没有达到配置的阈值或者大小还没达到等等，

可以针对上面 3.2 小节的配置微调下，例如：

```
agent1.sinks.log-sink1.hdfs.rollInterval = 20
```

当迟迟没有新日志生成的时候，如果你想很快的 flush，那么让它每隔 20s flush 持久化一下，agent 会根据多个条件，优先执行满足条件的触发器。

下面贴一些常见的持久化触发器：

```
# Number of seconds to wait before rolling current file (in 600 seconds)
agent.sinks.sink.hdfs.rollInterval=600

# File size to trigger roll, in bytes (256Mb)
agent.sinks.sink.hdfs.rollSize = 268435456

# never roll based on number of events
agent.sinks.sink.hdfs.rollCount = 0

# Timeout after which inactive files get closed (in seconds)
agent.sinks.sink.hdfs.idleTimeout = 3600

agent.sinks.HDFS.hdfs.batchSize = 1000
```

更多关于 sink 的触发机制与参数配置请参见：
http://flume.apache.org/FlumeUserGuide.html#hdfs-sink

http://stackoverflow.com/questions/20638498/flume-not-writing-to-hdfs-unless-killed

注意：对于 HDFS 来说应当竭力避免小文件问题，所以请慎重对待你配置的持久化触发机制。

4.4 数据重复写入、丢失问题

Flume的HDFSsink在数据写入/读出Channel时，都有Transcation的保证。当Transaction失败时，会回滚，然后重试。但由于HDFS不可修改文件的内容，假设有1万行数据要写入HDFS，而在写入5000行时，网络出现问题导致写入失败，Transaction回滚，然后重写这10000条记录成功，就会导致第一次写入的5000行重复。这些问题是 HDFS 文件系统设计上的特性缺陷，并不能通过简单的Bugfix来解决。我们只能关闭批量写入，单条事务保证，或者启用监控策略，两端对数。

Memory和exec的方式可能会有数据丢失，file 是 end to end 的可靠性保证的，但是性能较前两者要差。

end to end、store on failure 方式 ACK 确认时间设置过短（特别是高峰时间）也有可能引发数据的重复写入。

4.5 tail 断点续传的问题：

可以在 tail 传的时候记录行号，下次再传的时候，取上次记录的位置开始传输，类似：

```
1 agent1.sources.avro-source1.command = /usr/local/bin/tail  -n +$(tail -n1
```

需要注意如下几点：

（1）文件被 rotation 的时候，需要同步更新你的断点记录“指针”，

（2）需要按文件名来追踪文件，

（3）flume 挂掉后需要累加断点续传“指针”

（4）flume 挂掉后，如果恰好文件被 rotation，那么会有丢数据的风险，

只能监控尽快拉起或者加逻辑判断文件大小重置指针。

（5）tail 注意你的版本，请更新 coreutils 包到最新。

4.6 在 Flume 中如何修改、丢弃、按预定义规则分类存储数据？

这里你需要利用 Flume 提供的拦截器（Interceptor）机制来满足上述的需求了，具体请参考下面几个链接：

（1）Flume-NG源码阅读之Interceptor(原创)

http://www.cnblogs.com/lxf20061900/p/3664602.html

（2）Flume-NG自定义拦截器

http://sep10.com/posts/2014/04/15/flume-interceptor/

（3）Flume-ng生产环境实践（四）实现log格式化interceptor

http://blog.csdn.net/rjhym/article/details/8450728

（4）flume-ng如何根据源文件名输出到HDFS文件名

http://abloz.com/2013/02/19/flume-ng-output-according-to-the-source-file-name-to-the-hdfs-file-name.html

5、Refer：

（1）scribe、chukwa、kafka、flume日志系统对比

http://www.ttlsa.com/log-system/scribe-chukwa-kafka-flume-log-system-contrast/

（2）关于Flume-ng那些事 http://www.ttlsa.com/?s=flume

关于Flume-ng那些事（三）：常见架构测试 http://www.ttlsa.com/log-system/about-flume-ng-3/

（3）Flume 1.4.0 User Guide

http://archive.cloudera.com/cdh4/cdh/4/flume-ng-1.4.0-cdh4.7.0/FlumeUserGuide.html

（4）flume日志采集 http://blog.csdn.net/sunmeng_007/article/details/9762507

（5）Flume-NG + HDFS + HIVE 日志收集分析

http://eyelublog.wordpress.com/2013/01/13/flume-ng-hdfs-hive-%E6%97%A5%E5%BF%97%E6%94%B6%E9%9B%86%E5%88%86%E6%9E%90/

（6）【Twitter Storm系列】flume-ng+Kafka+Storm+HDFS 实时系统搭建

http://blog.csdn.net/weijonathan/article/details/18301321

（7）Flume-NG + HDFS + PIG 日志收集分析

http://hi.baidu.com/life_to_you/item/a98e2ec3367486dbef183b5e

flume 示例一收集tomcat日志 http://my.oschina.net/88sys/blog/71529

flume-ng 多节点集群示例 http://my.oschina.net/u/1401580/blog/204052

试用flume-ng 1.1 http://heipark.iteye.com/blog/1617995

（8）Flafka: Apache Flume Meets Apache Kafka for Event Processing

http://blog.cloudera.com/blog/2014/11/flafka-apache-flume-meets-apache-kafka-for-event-processing/

（9）Flume-ng的原理和使用

http://segmentfault.com/blog/javachen/1190000002532284

（10）基于Flume的美团日志收集系统(一)架构和设计

http://tech.meituan.com/mt-log-system-arch.html

（11）基于Flume的美团日志收集系统(二)改进和优化

http://tech.meituan.com/mt-log-system-optimization.html

（12）How-to: Do Real-Time Log Analytics with Apache Kafka, Cloudera Search, and Hue

http://blog.cloudera.com/blog/2015/02/how-to-do-real-time-log-analytics-with-apache-kafka-cloudera-search-and-hue/

（13）Real-time analytics in Apache Flume - Part 1

http://jameskinley.tumblr.com/post/57704266739/real-time-analytics-in-apache-flume-part-1

- « 上一篇
- 下一篇 »