 nginx的日志文件没有rotate功能。如果你不处理，日志文件将变得越来越大，还好我们可以写一个nginx日志切割脚本来自动切割日志文件。  
第一步就是重命名日志文件，不用担心重命名后nginx找不到日志文件而丢失日志。在你未重新打开原名字的日志文件前，nginx还是会向你重命名的文件写日志，linux是靠文件描述符而不是文件名定位文件。  
第二步向nginx主进程发送USR1信号。  
nginx主进程接到信号后会从配置文件中读取日志文件名称，重新打开日志文件(以配置文件中的日志名称命名)，并以工作进程的用户作为日志文件的所有者。  
重新打开日志文件后，nginx主进程会关闭重名的日志文件并通知工作进程使用新打开的日志文件。  
工作进程立刻打开新的日志文件并关闭重名名的日志文件。  
然后你就可以处理旧的日志文件了

**Nginx\_log\_division.sh代码：**

 Shell代码

#!/bin/bash

#设置日志文件存放目录

logs\_path="/usr/local/nginx/nginxlog/"

#设置pid文件

pid\_path="/usr/local/nginx/nginx-1.7.3/logs/nginx.pid"

#日志文件

filepath=${logs\_path}"access.log"

# Source function library.

#重命名日志文件

mv ${logs\_path}access.log ${logs\_path}access\_$(date -d '-1 day' '+%Y-%m-%d').log

#向nginx主进程发信号重新打开日志

kill -USR1 `cat ${pid\_path}`

**Oracle\_log\_division.sh代码：**

 Shell代码

#!/bin/bash

#设置日志文件存放目录

logs\_path="/usr/local/nginx/nginxlog/"

#设置pid文件

pid\_path="/usr/local/nginx/nginx-1.7.3/logs/nginx.pid"

#日志文件

filepath=${logs\_path}"access.log"

# Source function library.

#重命名日志文件

mv ${logs\_path}access.log ${logs\_path}access\_$(date -d '-1 day' '+%Y-%m-%d').log

#向nginx主进程发信号重新打开日志

kill -USR1 `cat ${pid\_path}`

**flumeng配置：**

Xml代码  [IMG_256](https://www.cnblogs.com/tonychai/p/javascript:void())

# A single-node Flume configuration

# Name the components on this agent

agent1.sources = source1

agent1.sinks = sink1

agent1.channels = channel1

# Describe/configure source1

agent1.sources.source1.type = exec

agent1.sources.source1.command = tail -n +0 -F /logs/access.log

agent1.sources.source1.channels = channel1

# Describe sink1

agent1.sinks.sink1.type = file\_roll

agent1.sinks.sink1.sink.directory=/var/log/data

# Use a channel which buffers events in memory

agent1.channels.channel1.type = file

agent1.channels.channel1.checkpointDir=/var/checkpoint

agent1.channels.channel1.dataDirs=/var/tmp

agent1.channels.channel1.capacity = 1000

agent1.channels.channel1.transactionCapactiy = 100

# Bind the source and sink to the channel

agent1.sources.source1.channels = channel1

agent1.sinks.sink1.channel = channel1

flume-ng agent -n a1 -c conf -f conf/example.file -Dflume.root.logger=DEBUG,console

参数说明：

-n 指定agent名称(与配置文件中代理的名字相同)

-c 指定flume中配置文件的目录

-f 指定配置文件

-Dflume.root.logger=DEBUG,console 设置日志等级

agent1.channels.channel1.transactionCapactiy = 100 sink端收集到100条数据后提交事务

Sink.batchsize sink端一次性从channel取的数据量(sink和channel使用一个双向队列维护事务，进行回滚。LinkedBlockingQuene)

og与ng的区别：

og是单线程  主从结构，原本由zookeeper管理

ng是双线程    已经取消master管理机制和zookeeper管理机制，变成了纯粹的传输工具

* source：负责日志的流入，eg：从文件、网络、kafka等数据源流入数据。   
  数据的流入有两种方式：
  + 轮询拉去
  + 事件驱动
* channel:数据聚合/暂存，暂存到内存、本地文件、数据库、kafka等，日志数据不会在管道停留很长的时间，很快被sink消费掉
* sink：负责数据转移存储，从channel中拿到日志，直接存储到HDFS、Hbase、kafka、ElasticSearch等

source和sink是异步的。

# flumeSource

Source支持Avro、Exec、Http、Spooling Director、Kafka、Syslog

* Exec Source 以运行linux命令的方式，持续的输出最新的数据
* Spool Source 支持监视一个目录或文件，解析其中新生成的事件，并将文件中的数据读取出来   
  拷贝到spool目录下的文件不可以再打开编辑，spool目录不可包含相应的子目录

Exec source和Spool Source的比较：

1. Exec source可以实现对日志的实时收集，但是存在Flume不运行或指令出错时将无法手机到日志数据，无法保证日志数据的完整性。
2. Spool Source 虽然无法实现实时的收集数据，但是可以以分钟的方式切割文件，趋近于实时。
3. 如果应用无法以分钟分割日志文件，可以两种收集方式同时使用。

Flume 过滤器（拦截器）：在App（应用程序日志）和source之间，对App日志进行拦截处理，官方已有的拦截器有：

* Timestamp Interceptor：在event的header中添加一个key作为timestamp，value为当前的事件戳。
* Host Interceprot：在event的header中添加一个的key为host，value为当前机器的hostname或IP
* Static Interceptor：在event的header中添加自动以的key和value
* Regex Filtering Interceptor：通过正则来清晰或包含匹配的events
* Regex Extrator Inteneptor：通过正则来在header中添加指定的key，value则为正则匹配的部分。

# Flume Channel

缓存从Source 到Sink的中间数据，可用下列配置作为channel：

* 内存
* 文件
* JDBC
* kafka

Flume Channel选择器：数据通过不同的Channel路由到不同的Sink中

* Replication Channel Selectot：将source过来的events发往所有的Channel
* Multiplexing Channel Selectot：可以选择将数据发往哪些Channel

# Flume Sink

负责从Channel中读取数据并发送给下一个Agent或最终目的地。   
目的地：HDFS、Hbase、Avro、ElasticSearch、File、Logger

Flume 序列化器

* Body Text Serializer：忽略event header ，只将event value写到输出流。
* Avro Event Serializer：数据以Avro序列化，然后写到输出流。

fileHeader = true

# 或者

basenameHeader = true会将文件名加在数据之前

# kafka Sink 配置坑

AGENT\_NAME.sinks.malwrK.kafka.bootstrap.servers= serv1:PORT[,serv2:PORT,serv3:PORT...]

这项配置中，如果只指定单个的 Kafka 服务器地址，Flume Agent 会正常运行，但如果把 Kafka集群里的多个服务器地址都写上，并且Kafka的默认配置不修改，则会报以下错误：

WARN Error while fetching metadata with correlation id 39 : {4-3-16-topic1=LEADER\_NOT\_AVAILABLE}

并且数据无法正常发送到 Kafka 集群。该错误的意思是集群中的服务器，没有一个可用的 LEADER，导致数据无法正常 Fetch。那么解决方法也很简单，在 Kafka 集群的每个服务器 server.properties 配置文件中，开启（取消注释）以下配置，并把配置项的值改成统一指定的一台 Kafka 服务器做 LEADER：

|  |  |
| --- | --- |
| 1 | advertised.listeners=PLAINTEXT://your.host.name:9092 |

# http json source 配置坑

Flume 的 Event 只有两部分组成：headers 和 body，所以传输的 JSON Array 里的 Flume Event，每个 Event 也只可以包括 headers 和 body 两部分，而且 headers的值是 JSON 对象（object），而body的值只能是 JSON 字符串（string），不能是其他 JSON 值类型，比如下图所示的 JSON 形式，对 Flume 来说算是合法的

IMG_256

最后总结一下，传给 flume HTTP Source 的 JSON 数据：

必须是 Array 形式（用方括号括起来），即使 Array 里只有 1 个元素（每个元素到了 Flume 那儿就代表一个 Event），在 Python 里就要用 dict 外面嵌套 list 或者 tuple 生成 JSON Array；

JSON Array 里每个元素必须由 headers 和 body 两个 field 构成;

JSON Array 中元素的 headers 的值是 JSON Object(具体可以自定义)，body 的值必须是字串（JSON String），而不能是其他形式（JSON Object/JSON Array等，对应 Python 中的 dict/list/tuple 等）