Day 06 Flume

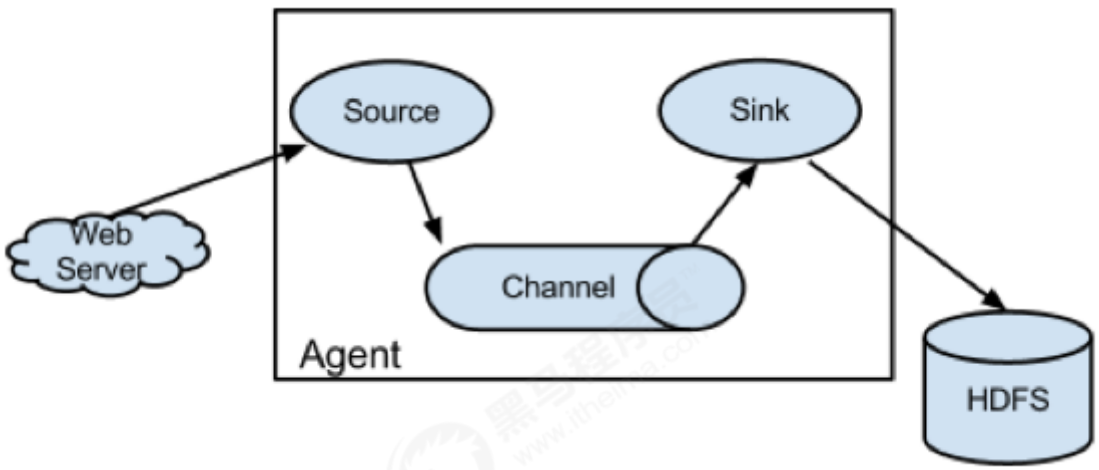
1. Flume 介绍

1.1. 概述

- Flume是一个分布式、可靠、和高可用的海量日志采集、聚合和传输的系统。
- Flume可以采集文件，socket数据包、文件、文件夹、kafka等各种形式源数据，又可以将采集到的数据(下沉sink)输出到HDFS、hbase、hive、kafka等众多外部存储系统中
- 一般的采集需求，通过对flume的简单配置即可实现
- Flume针对特殊场景也具备良好的自定义扩展能力，
  因此，flume可以适用于大部分的日常数据采集场景

1.2. 运行机制

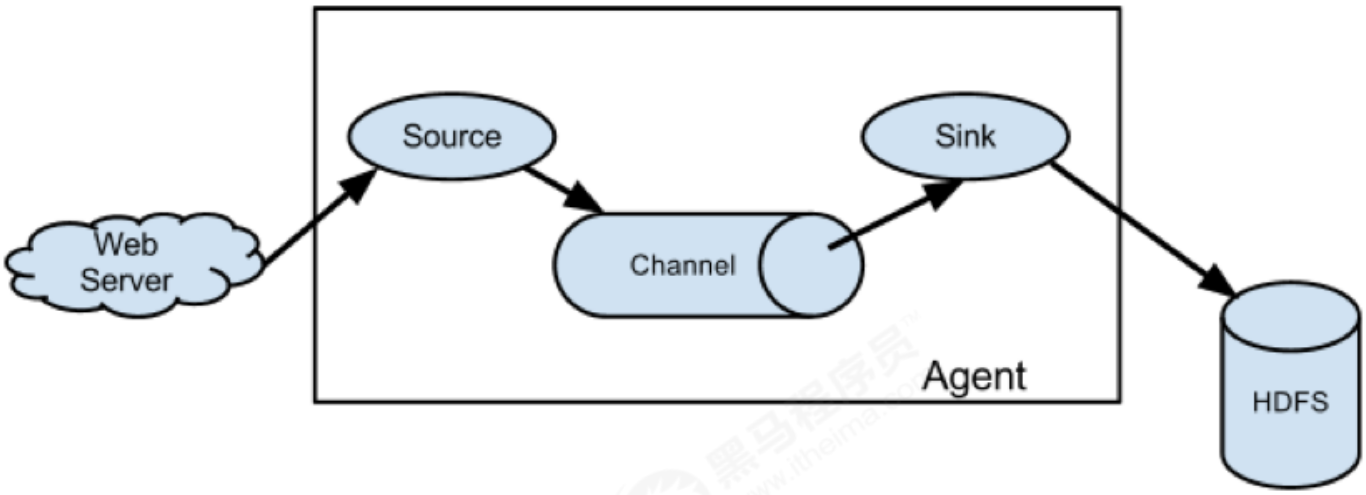
1. Flume分布式系统中最核心的角色是agent，flume采集系统就是由一个个agent所连接起来形成
2. 每一个agent相当于一个数据传递员，内部有三个组件：
    1. Source：采集组件，用于跟数据源对接，以获取数据
    2. Sink：下沉组件，用于往下一级agent传递数据或者往最终存储系统传递数据
    3. Channel：传输通道组件，用于从source将数据传递到sink



1.3. Flume 结构图

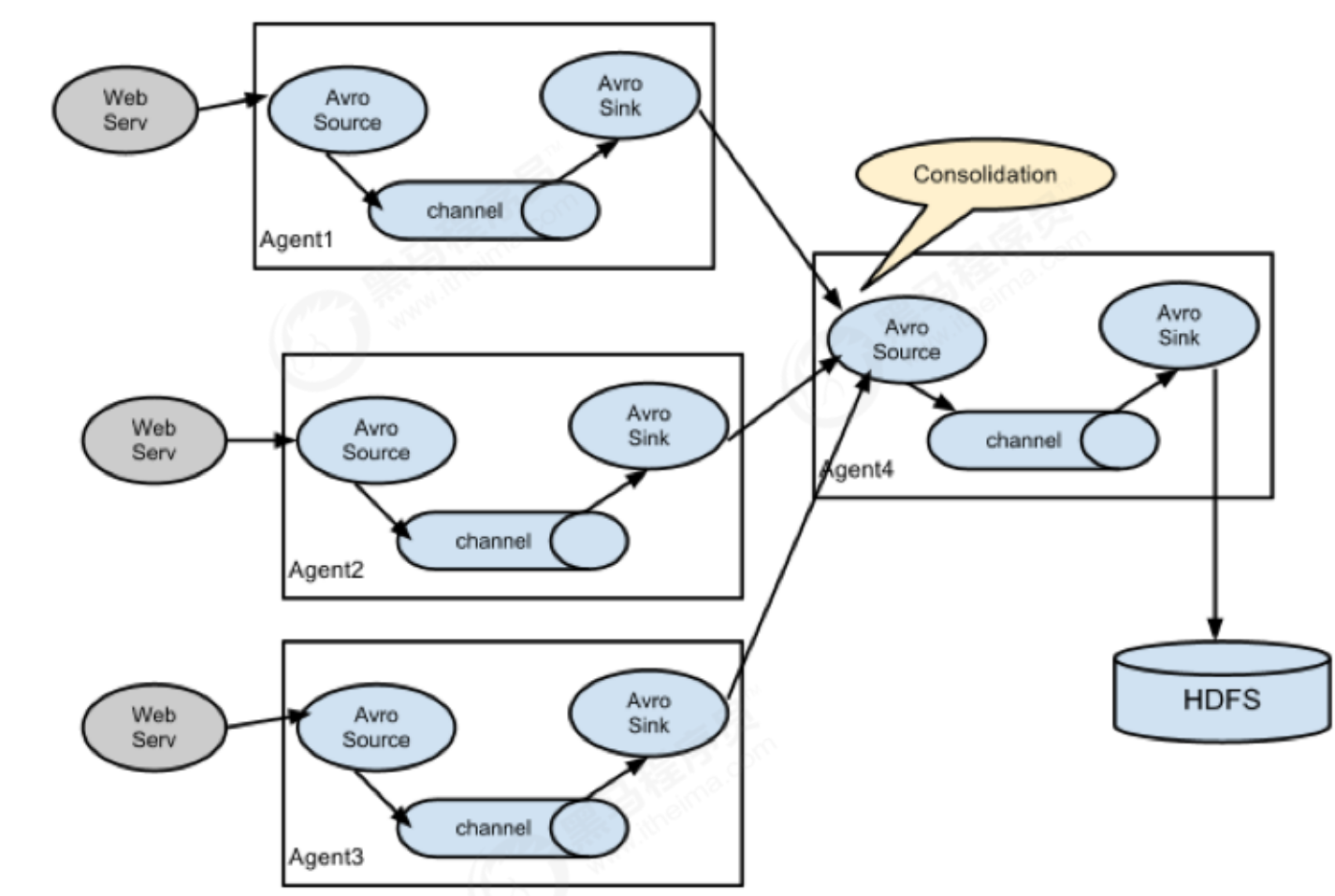
简单结构

单个 Agent 采集数据



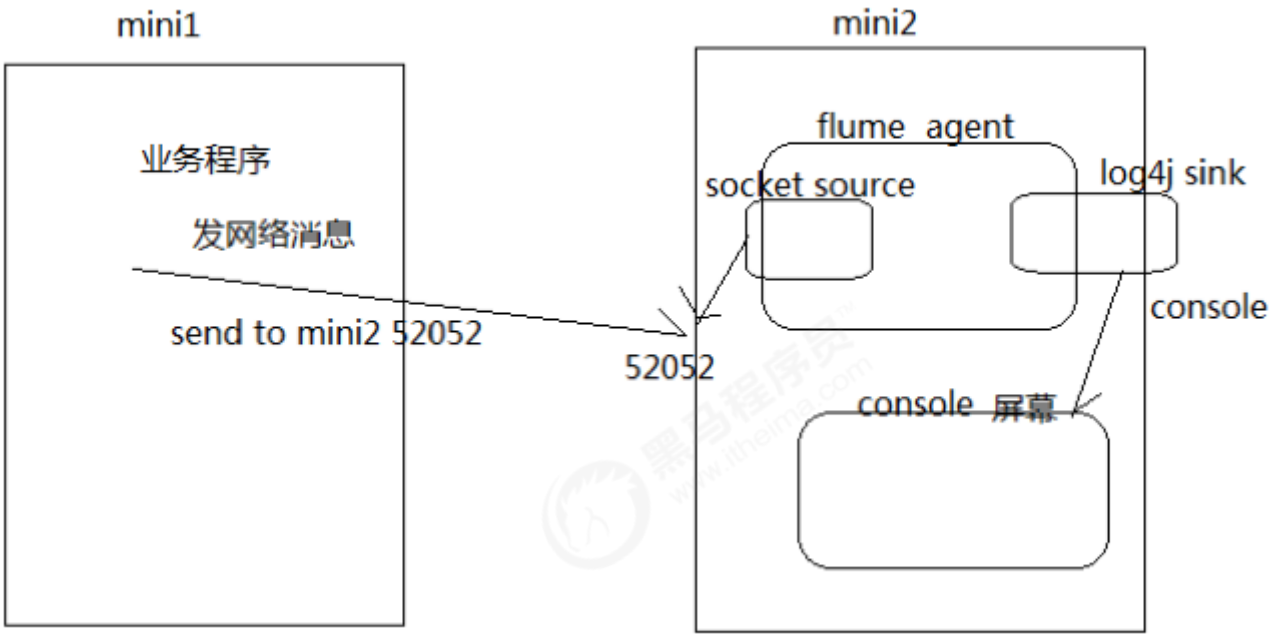
复杂结构

多级 Agent 之间串联



2. Flume 实战案例

> 案例：使用网络telent命令向一台机器发送一些网络数据，然后通过flume采集网络端口数据



2.1. Flume 的安装部署

Step 1: 下载解压修改配置文件

下载地址：

http://archive.apache.org/dist/flume/1.8.0/apache-flume-1.8.0-bin.tar.gz

Flume的安装非常简单，只需要解压即可，当然，前提是已有hadoop环境

上传安装包到数据源所在节点上

这里我们采用在第三台机器来进行安装

```
cd /export/softwares/
tar -zxvf apache-flume-1.8.0-bin.tar.gz -C ../servers/
cd /export/servers/apache-flume-1.8.0-bin/conf
cp  flume-env.sh.template flume-env.sh
vim flume-env.sh
export JAVA_HOME=/export/servers/jdk1.8.0_141
```

Step 2: 开发配置文件

根据数据采集的需求配置采集方案，描述在配置文件中(文件名可任意自定义)

配置我们的网络收集的配置文件
在flume的conf目录下新建一个配置文件（采集方案）

```
vim   /export/servers/apache-flume-1.8.0-bin/conf/netcat-logger.conf
```

```
# 定义这个agent中各组件的名字
a1.sources = r1
a1.sinks = k1
a1.channels = c1

# 描述和配置source组件：r1
a1.sources.r1.type = netcat
a1.sources.r1.bind = 192.168.174.
a1.sources.r1.port = 44444

# 描述和配置sink组件：k1
a1.sinks.k1.type = logger

# 描述和配置channel组件，此处使用是内存缓存的方式
a1.channels.c1.type = memory
a1.channels.c1.capacity = 1000
a1.channels.c1.transactionCapacity = 100

# 描述和配置source  channel   sink之间的连接关系
a1.sources.r1.channels = c1
a1.sinks.k1.channel = c1
```

Step 3: 启动配置文件

指定采集方案配置文件，在相应的节点上启动flume agent

先用一个最简单的例子来测试一下程序环境是否正常
启动agent去采集数据

```
bin/flume-ng agent -c conf -f conf/netcat-logger.conf -n a1  -Dflume.root.logger=INFO,c
```

- `-c conf` 指定flume自身的配置文件所在目录
- `-f conf/netcat-logger.con` 指定我们所描述的采集方案
- `-n a1` 指定我们这个agent的名字

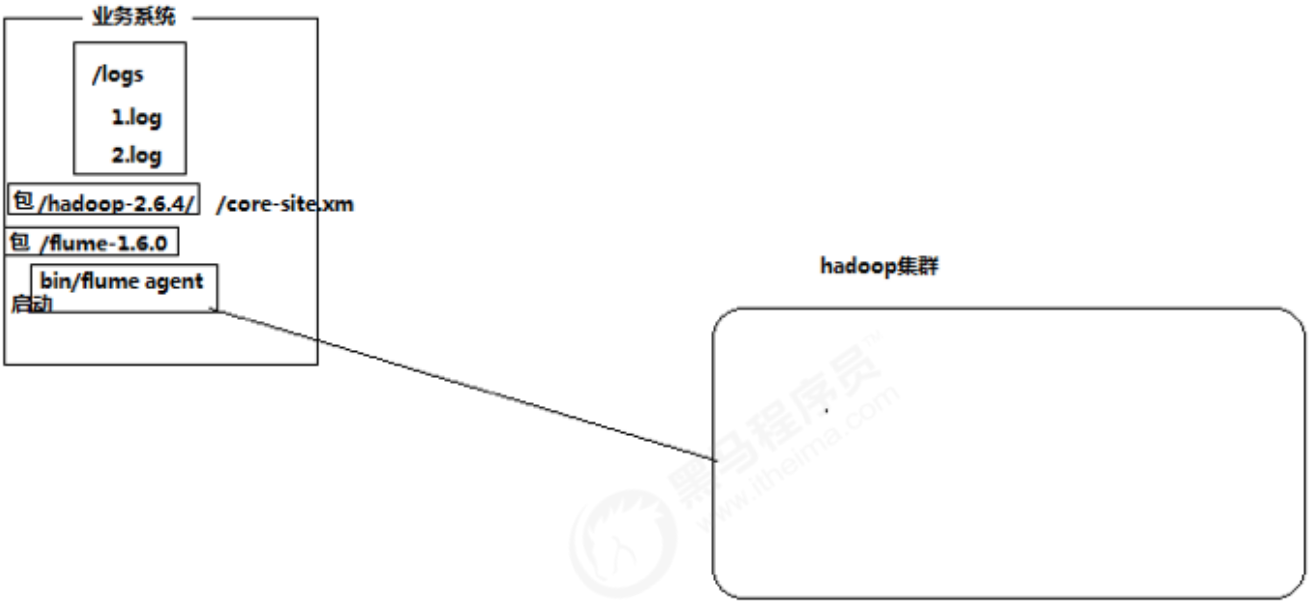
Step 4: 安装 Telnet 准备测试

在node02机器上面安装telnet客户端，用于模拟数据的发送

```
yum -y install telnet
telnet  node03  44444   # 使用telnet模拟数据发送
```

2.2. 采集案例

2.2.3. 采集目录到 HDFS



需求

某服务器的某特定目录下，会不断产生新的文件，每当有新文件出现，就需要把文件采集到HDFS中去

思路

根据需求，首先定义以下3大要素

1. 数据源组件，即source ——监控文件目录 : spooldir
    1. 监视一个目录，只要目录中出现新文件，就会采集文件中的内容
    2. 采集完成的文件，会被agent自动添加一个后缀：COMPLETED
    3. 所监视的目录中不允许重复出现相同文件名的文件
2. 下沉组件，即sink——HDFS文件系统 : hdfs sink
3. 通道组件，即channel——可用file channel 也可以用内存channel

Step 1: Flume 配置文件

```
cd  /export/servers/apache-flume-1.8.0-bin/conf
mkdir -p /export/servers/dirfile
vim spooldir.conf
```

```
# Name the components on this agent
a1.sources = r1
a1.sinks = k1
a1.channels = c1
# Describe/configure the source
##注意：不能往监控目中重复丢同名文件
a1.sources.r1.type = spooldir
a1.sources.r1.spoolDir = /export/servers/dirfile
a1.sources.r1.fileHeader = true
# Describe the sink
a1.sinks.k1.type = hdfs
a1.sinks.k1.channel = c1
a1.sinks.k1.hdfs.path = hdfs://node01:8020/spooldir/files/%y-%m-%d/%H%M/
a1.sinks.k1.hdfs.filePrefix = events-
a1.sinks.k1.hdfs.round = true
a1.sinks.k1.hdfs.roundValue = 10
a1.sinks.k1.hdfs.roundUnit = minute
a1.sinks.k1.hdfs.rollInterval = 3
a1.sinks.k1.hdfs.rollSize = 20
a1.sinks.k1.hdfs.rollCount = 5
a1.sinks.k1.hdfs.batchSize = 1
a1.sinks.k1.hdfs.useLocalTimeStamp = true
#生成的文件类型，默认是Sequencefile，可用DataStream，则为普通文本
a1.sinks.k1.hdfs.fileType = DataStream
# 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
```

Channel参数解释

capacity：默认该通道中最大的可以存储的event数量
trasactionCapacity：每次最大可以从source中拿到或者送到sink中的event数量
keep-alive：event添加到通道中或者移出的允许时间

Step 2: 启动 Flume

```
bin/flume-ng agent -c ./conf -f ./conf/spooldir.conf -n a1 -Dflume.root.logger=INFO,cor
```

Step 3: 上传文件到指定目录

将不同的文件上传到下面目录里面去，注意文件不能重名

```
cd /export/servers/dirfile
```

2.2.4. 采集文件到 HDFS

需求

比如业务系统使用log4j生成的日志，日志内容不断增加，需要把追加到日志文件中的数据实时采集到hdfs

分析

根据需求，首先定义以下3大要素

- 采集源，即source——监控文件内容更新 : exec ‘tail -F file’
- 下沉目标，即sink——HDFS文件系统 : hdfs sink
- Source和sink之间的传递通道——channel，可用file channel 也可以用 内存channel

Step 1: 定义 Flume 配置文件

```
cd /export/servers/apache-flume-1.8.0-bin/conf
vim tail-file.conf
```

```
agent1.sources = source1
agent1.sinks = sink1
agent1.channels = channel1

# Describe/configure tail -F source1
agent1.sources.source1.type = exec
agent1.sources.source1.command = tail -F /export/servers/taillogs/access_log
agent1.sources.source1.channels = channel1


# Describe sink1
agent1.sinks.sink1.type = hdfs
#a1.sinks.k1.channel = c1
agent1.sinks.sink1.hdfs.path = hdfs://node01:8020/weblog/flume-collection/%y-%m-%d/%H-%
agent1.sinks.sink1.hdfs.filePrefix = access_log
agent1.sinks.sink1.hdfs.maxOpenFiles = 5000
agent1.sinks.sink1.hdfs.batchSize= 100
agent1.sinks.sink1.hdfs.fileType = DataStream
agent1.sinks.sink1.hdfs.writeFormat =Text

agent1.sinks.sink1.hdfs.round = true
agent1.sinks.sink1.hdfs.roundValue = 10
agent1.sinks.sink1.hdfs.roundUnit = minute
agent1.sinks.sink1.hdfs.useLocalTimeStamp = true

# Use a channel which buffers events in memory
agent1.channels.channel1.type = memory
agent1.channels.channel1.keep-alive = 120
agent1.channels.channel1.capacity = 500000
agent1.channels.channel1.transactionCapacity = 600

# Bind the source and sink to the channel
agent1.sources.source1.channels = channel1
agent1.sinks.sink1.channel = channel1
```

Step 2: 启动 Flume

```
cd  /export/servers/apache-flume-1.6.0-cdh5.14.0-bin
bin/flume-ng agent -c conf -f conf/tail-file.conf -n agent1  -Dflume.root.logger=INFO,c
```

Step 3: 开发 Shell 脚本定时追加文件内容

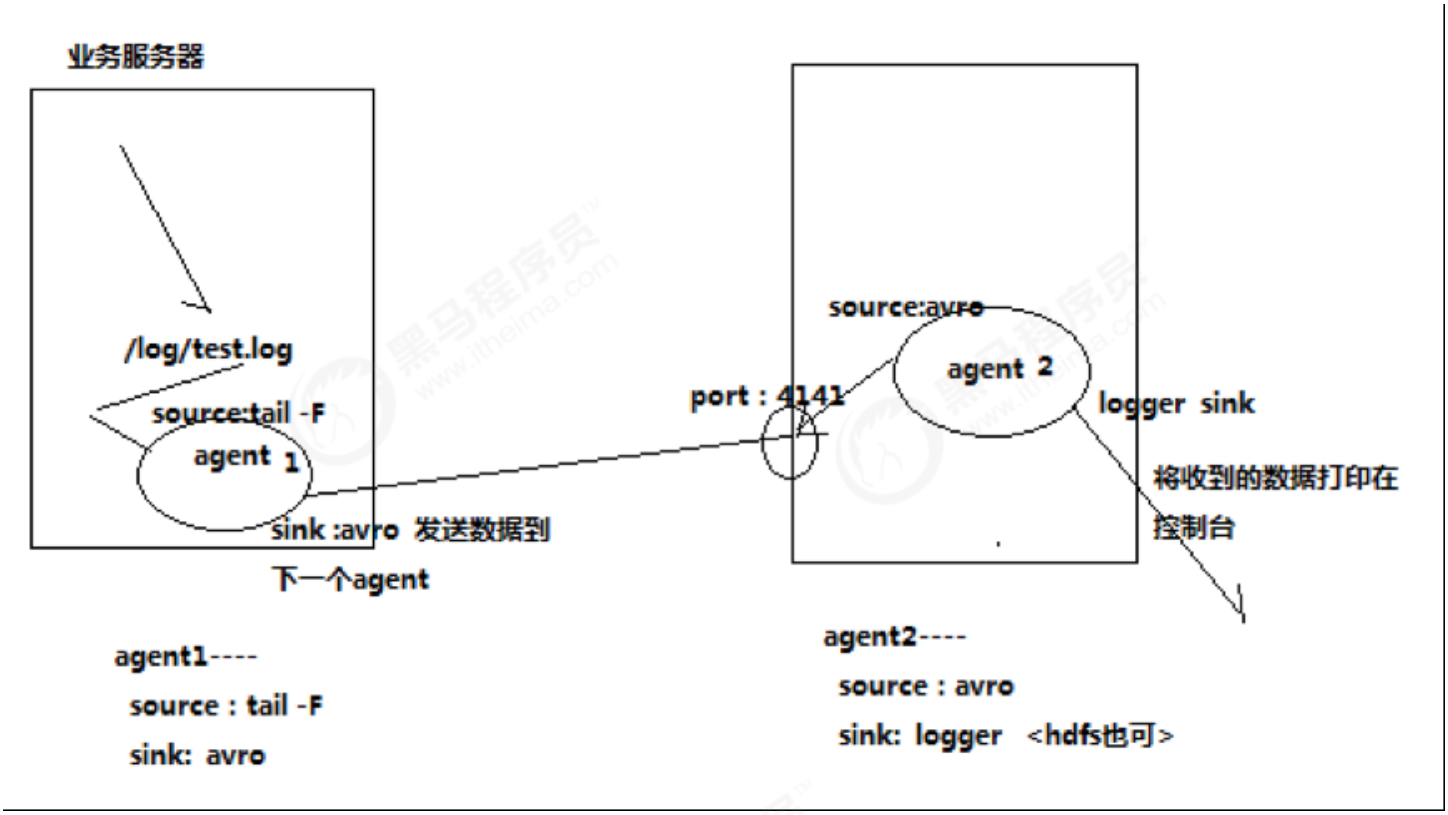
```
mkdir -p /export/servers/shells/
cd  /export/servers/shells/
vim tail-file.sh
```

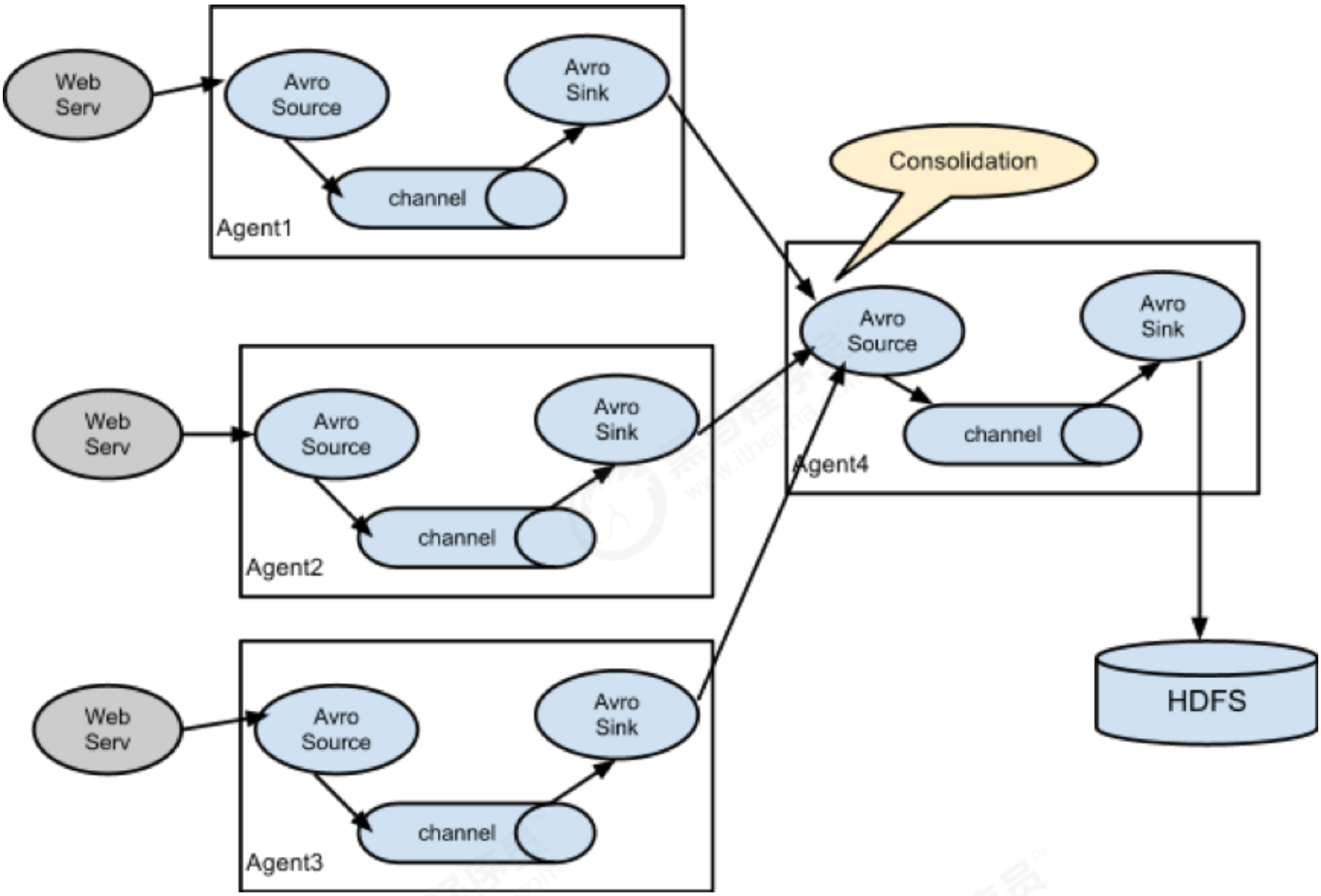
```
#!/bin/bash
while true
do
 date >> /export/servers/taillogs/access_log;
  sleep 0.5;
done
```

Step 4: 启动脚本

```
# 创建文件夹
mkdir -p /export/servers/taillogs
# 启动脚本
sh /export/servers/shells/tail-file.sh
```

2.2.5. Agent 级联





分析

第一个agent负责收集文件当中的数据，通过网络发送到第二个agent当中去
第二个agent负责接收第一个agent发送的数据，并将数据保存到hdfs上面去

Step 1: Node02 安装 Flume

将node03机器上面解压后的flume文件夹拷贝到node02机器上面去

```
cd  /export/servers
scp -r apache-flume-1.8.0-bin/ node02:$PWD
```

Step 2: Node02 配置 Flume

在node02机器配置我们的flume

```
cd /export/servers/ apache-flume-1.8.0-bin/conf
vim tail-avro-avro-logger.conf
```

```
##################
# Name the components on this agent
a1.sources = r1
a1.sinks = k1
a1.channels = c1
# Describe/configure the source
a1.sources.r1.type = exec
a1.sources.r1.command = tail -F /export/servers/taillogs/access_log
a1.sources.r1.channels = c1
# Describe the sink
##sink端的avro是一个数据发送者
a1.sinks = k1
a1.sinks.k1.type = avro
a1.sinks.k1.channel = c1
a1.sinks.k1.hostname = 192.168.174.120
a1.sinks.k1.port = 4141
a1.sinks.k1.batch-size = 10
# 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
```

Step 3: 开发脚本向文件中写入数据

直接将node03下面的脚本和数据拷贝到node02即可，node03机器上执行以下命令

```
cd  /export/servers
scp -r shells/ taillogs/ node02:$PWD
```

Step 4: Node03 Flume 配置文件

在node03机器上开发flume的配置文件

```
cd /export/servers/apache-flume-1.8.0-bin/conf
vim avro-hdfs.conf
```

```
# Name the components on this agent
a1.sources = r1
a1.sinks = k1
a1.channels = c1
# Describe/configure the source
##source中的avro组件是一个接收者服务
a1.sources.r1.type = avro
a1.sources.r1.channels = c1
a1.sources.r1.bind = 192.168.174.120
a1.sources.r1.port = 4141
# Describe the sink
a1.sinks.k1.type = hdfs
a1.sinks.k1.hdfs.path = hdfs://node01:8020/av /%y-%m-%d/%H%M/
a1.sinks.k1.hdfs.filePrefix = events-
a1.sinks.k1.hdfs.round = true
a1.sinks.k1.hdfs.roundValue = 10
a1.sinks.k1.hdfs.roundUnit = minute
a1.sinks.k1.hdfs.rollInterval = 3
a1.sinks.k1.hdfs.rollSize = 20
a1.sinks.k1.hdfs.rollCount = 5
a1.sinks.k1.hdfs.batchSize = 1
a1.sinks.k1.hdfs.useLocalTimeStamp = true
#生成的文件类型，默认是Sequencefile，可用DataStream，则为普通文本
a1.sinks.k1.hdfs.fileType = DataStream
# 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
```

Step 5: 顺序启动

node03机器启动flume进程

```
cd /export/servers/apache-flume-1.8.0-bin
bin/flume-ng agent -c conf -f conf/avro-hdfs.conf -n a1  -Dflume.root.logger=INFO,consc
```

node02机器启动flume进程

```
cd /export/servers/apache-flume-1.8.0-bin/
bin/flume-ng agent -c conf -f conf/tail-avro-avro-logger.conf -n a1  -Dflume.root.logge
```

node02机器启shell脚本生成文件

```
cd  /export/servers/shells
sh tail-file.sh
```

3. 高可用方案

在完成单点的Flume NG搭建后，下面我们搭建一个高可用的Flume NG集群，架构图如下所示：

3.1. 角色分配

Flume的Agent和Collector分布如下表所示：

名称	HOST	角色
Agent1	node01	Web Server
Collector1	node02	AgentMstr1
Collector2	node03	AgentMstr2

图中所示，Agent1数据分别流入到Collector1和Collector2，Flume NG本身提供了Failover机制，可以自动切换和恢复。在上图中，有3个产生日志服务器分布在不同的机房，要把所有的日志都收集到一个集群中存储。下 面我们开发配置Flume NG集群

3.2. Node01 安装和配置

将node03机器上面的flume安装包以及文件生产的两个目录拷贝到node01机器上面去

node03机器执行以下命令

```
cd /export/servers
scp -r apache-flume-1.8.0-bin/ node01:$PWD
scp -r shells/ taillogs/ node01:$PWD
```

node01机器配置agent的配置文件

```
cd /export/servers/apache-flume-1.8.0-bin/conf
vim agent.conf
```

```
#agent1 name
agent1.channels = c1
agent1.sources = r1
agent1.sinks = k1 k2
#
##set gruop
agent1.sinkgroups = g1
#

agent1.sources.r1.channels = c1
agent1.sources.r1.type = exec
agent1.sources.r1.command = tail -F /export/servers/taillogs/access_log
#
##set channel
agent1.channels.c1.type = memory
agent1.channels.c1.capacity = 1000
agent1.channels.c1.transactionCapacity = 100
#
## set sink1
agent1.sinks.k1.channel = c1
agent1.sinks.k1.type = avro
agent1.sinks.k1.hostname = node02
agent1.sinks.k1.port = 52020
#
## set sink2
agent1.sinks.k2.channel = c1
agent1.sinks.k2.type = avro
agent1.sinks.k2.hostname = node03
agent1.sinks.k2.port = 52020
#
##set sink group
agent1.sinkgroups.g1.sinks = k1 k2
#
##set failover
agent1.sinkgroups.g1.processor.type = failover
agent1.sinkgroups.g1.processor.priority.k1 = 10
agent1.sinkgroups.g1.processor.priority.k2 = 1
agent1.sinkgroups.g1.processor.maxpenalty = 10000
```

3.3. Node02 与 Node03 配置 FlumeCollection

node02机器修改配置文件

```
cd /export/servers/apache-flume-1.8.0-bin/conf
vim collector.conf
```

```
#set Agent name
a1.sources = r1
a1.channels = c1
a1.sinks = k1
#
##set channel
a1.channels.c1.type = memory
a1.channels.c1.capacity = 1000
a1.channels.c1.transactionCapacity = 100
#
## other node,nna to nns
a1.sources.r1.type = avro
a1.sources.r1.bind = node02
a1.sources.r1.port = 52020
a1.sources.r1.channels = c1
#
##set sink to hdfs
a1.sinks.k1.type=hdfs
a1.sinks.k1.hdfs.path= hdfs://node01:8020/flume/failover/
a1.sinks.k1.hdfs.fileType=DataStream
a1.sinks.k1.hdfs.writeFormat=TEXT
a1.sinks.k1.hdfs.rollInterval=10
a1.sinks.k1.channel=c1
a1.sinks.k1.hdfs.filePrefix=%Y-%m-%d
#
```

node03机器修改配置文件

```
cd  /export/servers/apache-flume-1.8.0-bin/conf
vim collector.conf
```

```
#set Agent name
a1.sources = r1
a1.channels = c1
a1.sinks = k1
#
##set channel
a1.channels.c1.type = memory
a1.channels.c1.capacity = 1000
a1.channels.c1.transactionCapacity = 100
#
## other node,nna to nns
a1.sources.r1.type = avro
a1.sources.r1.bind = node03
a1.sources.r1.port = 52020
a1.sources.r1.channels = c1
#
##set sink to hdfs
a1.sinks.k1.type=hdfs
a1.sinks.k1.hdfs.path= hdfs://node01:8020/flume/failover/
a1.sinks.k1.hdfs.fileType=DataStream
a1.sinks.k1.hdfs.writeFormat=TEXT
a1.sinks.k1.hdfs.rollInterval=10
a1.sinks.k1.channel=c1
a1.sinks.k1.hdfs.filePrefix=%Y-%m-%d
```

3.4. 顺序启动

node03机器上面启动flume

```
cd /export/servers/apache-flume-1.8.0-bin
bin/flume-ng agent -n a1 -c conf -f conf/collector.conf -Dflume.root.logger=DEBUG,consc
```

node02机器上面启动flume

```
cd /export/servers/apache-flume-1.8.0-bin
bin/flume-ng agent -n a1 -c conf -f conf/collector.conf -Dflume.root.logger=DEBUG,consc
```

node01机器上面启动flume

```
cd /export/servers/apache-flume-1.8.0-bin
bin/flume-ng agent -n agent1 -c conf -f conf/agent.conf -Dflume.root.logger=DEBUG,consc
```

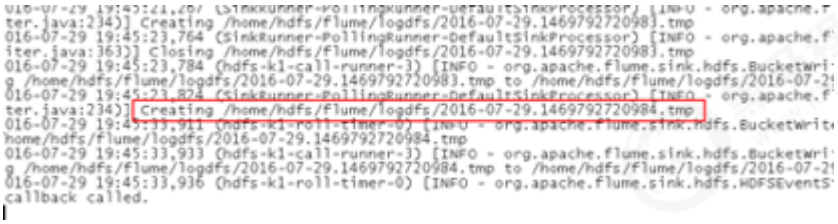
node01机器启动文件产生脚本

```
cd  /export/servers/shells
sh tail-file.sh
```

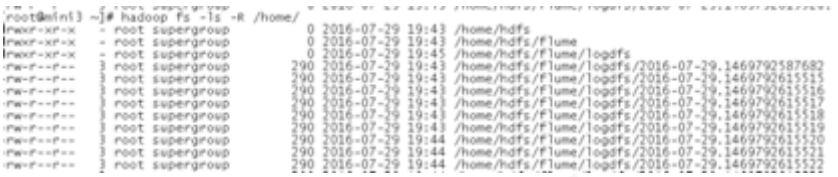
3.5. Failover 测试

下面我们来测试下Flume NG集群的高可用（故障转移）。场景如下：我们在Agent1节点上传文件，由于我们配置Collector1的权重比Collector2大，所以 Collector1优先采集并上传到存储系统。然后我们kill掉Collector1，此时有Collector2负责日志的采集上传工作，之后，我 们手动恢复Collector1节点的Flume服务，再次在Agent1上次文件，发现Collector1恢复优先级别的采集工作。具体截图如下所 示：

Collector1优先上传



HDFS集群中上传的log内容预览

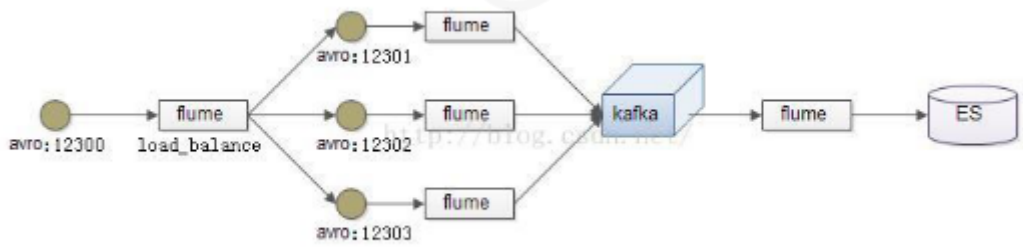


Collector1宕机，Collector2获取优先上传权限

重启Collector1服务，Collector1重新获得优先上传的权限

4. Flume 的负载均衡

负载均衡是用于解决一台机器(一个进程)无法解决所有请求而产生的一种算法。Load balancing Sink Processor 能够实现 load balance 功能，如下图Agent1 是一个路由节点，负责将 Channel 暂存的 Event 均衡到对应的多个 Sink组件上，而每个 Sink 组件分别连接到一个独立的 Agent 上，示例配置，如下所示：



在此处我们通过三台机器来进行模拟flume的负载均衡

三台机器规划如下：

node01：采集数据，发送到node02和node03机器上去

node02：接收node01的部分数据

node03：接收node01的部分数据

第一步：开发node01服务器的flume配置

node01服务器配置：

```
cd /export/servers/apache-flume-1.8.0-bin/conf
vim load_banlancer_client.conf
```

```
# agent name
<p class="mume-header " id="agent-name"></p>


a1.channels = c1

a1.sources = r1

a1.sinks = k1 k2



# set gruop
<p class="mume-header " id="set-gruop"></p>


a1.sinkgroups = g1

# set channel
<p class="mume-header " id="set-channel"></p>


a1.channels.c1.type = memory

a1.channels.c1.capacity = 1000

a1.channels.c1.transactionCapacity = 100

a1.sources.r1.channels = c1

a1.sources.r1.type = exec

a1.sources.r1.command = tail -F /export/servers/taillogs/access_log

# set sink1
<p class="mume-header " id="set-sink1"></p>


a1.sinks.k1.channel = c1

a1.sinks.k1.type = avro

a1.sinks.k1.hostname = node02

a1.sinks.k1.port = 52020

# set sink2
<p class="mume-header " id="set-sink2"></p>


a1.sinks.k2.channel = c1

a1.sinks.k2.type = avro
```

```
a1.sinks.k2.hostname = node03

a1.sinks.k2.port = 52020

# set sink group
<p class="mume-header " id="set-sink-group"></p>


a1.sinkgroups.g1.sinks = k1 k2

# set failover
<p class="mume-header " id="set-failover"></p>


a1.sinkgroups.g1.processor.type = load_balance

a1.sinkgroups.g1.processor.backoff = true

a1.sinkgroups.g1.processor.selector = round_robin

a1.sinkgroups.g1.processor.selector.maxTimeOut=10000
```

第二步：开发node02服务器的flume配置

```
cd /export/servers/apache-flume-1.8.0-bin/conf

vim load_banlancer_server.conf
```

```
# Name the components on this agent
<p class="mume-header " id="name-the-components-on-this-agent"></p>


a1.sources = r1

a1.sinks = k1

a1.channels = c1



# Describe/configure the source
<p class="mume-header " id="describeconfigure-the-source"></p>


a1.sources.r1.type = avro

a1.sources.r1.channels = c1

a1.sources.r1.bind = node02

a1.sources.r1.port = 52020



# Describe the sink
<p class="mume-header " id="describe-the-sink"></p>


a1.sinks.k1.type = logger



# Use a channel which buffers events in memory
<p class="mume-header " id="use-a-channel-which-buffers-events-in-memory"></p>


a1.channels.c1.type = memory

a1.channels.c1.capacity = 1000

a1.channels.c1.transactionCapacity = 100



# Bind the source and sink to the channel
<p class="mume-header " id="bind-the-source-and-sink-to-the-channel"></p>


a1.sources.r1.channels = c1

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

第三步：开发node03服务器flume配置

node03服务器配置

```
cd /export/servers/apache-flume-1.8.0-bin/conf
vim load_banlancer_server.conf
```

```
# Name the components on this agent
<p class="mume-header " id="name-the-components-on-this-agent-1"></p>


a1.sources = r1

a1.sinks = k1

a1.channels = c1



# Describe/configure the source
<p class="mume-header " id="describeconfigure-the-source-1"></p>


a1.sources.r1.type = avro

a1.sources.r1.channels = c1

a1.sources.r1.bind = node03

a1.sources.r1.port = 52020



# Describe the sink
<p class="mume-header " id="describe-the-sink-1"></p>


a1.sinks.k1.type = logger



# Use a channel which buffers events in memory
<p class="mume-header " id="use-a-channel-which-buffers-events-in-memory-1"></p>


a1.channels.c1.type = memory

a1.channels.c1.capacity = 1000

a1.channels.c1.transactionCapacity = 100



# Bind the source and sink to the channel
<p class="mume-header " id="bind-the-source-and-sink-to-the-channel-1"></p>


a1.sources.r1.channels = c1

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

第四步：准备启动flume服务

启动node03的flume服务

```
cd /export/servers/apache-flume-1.8.0-bin
bin/flume-ng agent -n a1 -c conf -f conf/load_banlancer_server.conf -Dflume.root.logger
```

启动node02的flume服务

```
cd /export/servers/apache-flume-1.8.0-bin
bin/flume-ng agent -n a1 -c conf -f conf/load_banlancer_server.conf -Dflume.root.logger
```

启动node01的flume服务

```
cd /export/servers/apache-flume-1.8.0-bin

bin/flume-ng agent -n a1 -c conf -f conf/load_banlancer_client.conf -Dflume.root.logger
```

第五步：node01服务器运行脚本产生数据

cd /export/servers/shells

sh tail-file.sh

5. Flume 案例一

1. 案例场景

A、B两台日志服务机器实时生产日志主要类型为access.log、nginx.log、web.log

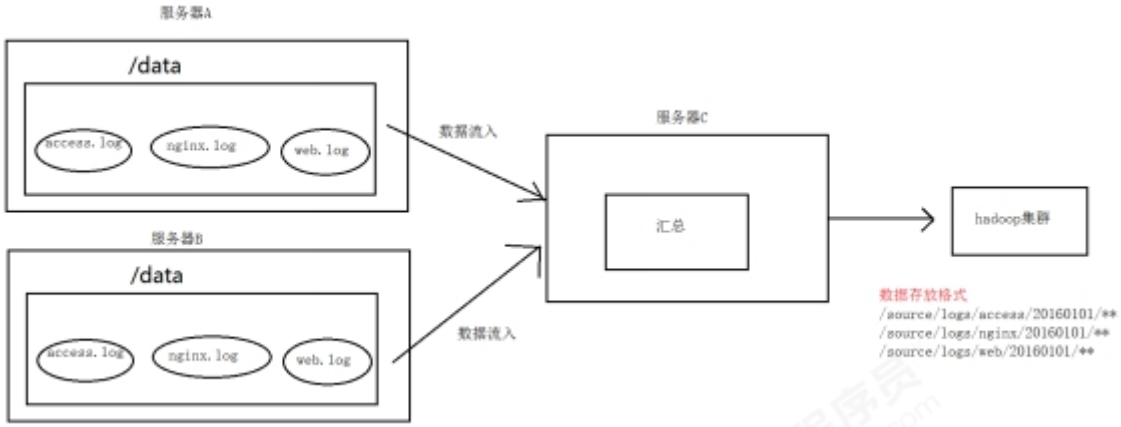
现在要求：

把A、B 机器中的access.log、nginx.log、web.log 采集汇总到C机器上然后统一收集到hdfs中。

但是在hdfs中要求的目录为：

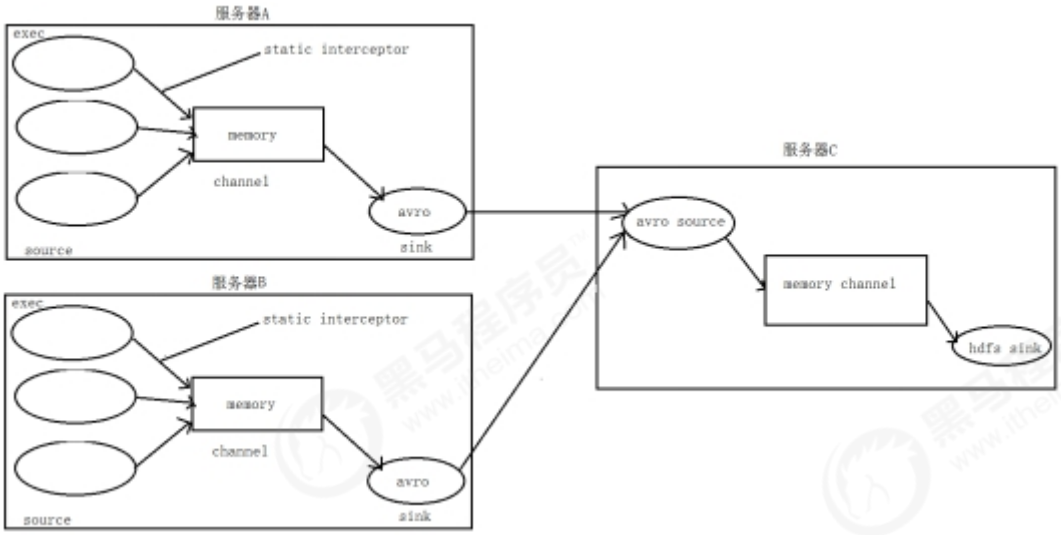
```
/source/logs/access/20180101/**
/source/logs/nginx/20180101/**
/source/logs/web/20180101/**
```

2. 场景分析



图一

3. 数据流程处理分析



4、实现

服务器A对应的IP为 192.168.174.100

服务器B对应的IP为 192.168.174.110

服务器C对应的IP为 192.168.174.120

采集端配置文件开发

node01与node02服务器开发flume的配置文件

```
cd /export/servers/apache-flume-1.6.0-cdh5.14.0-bin/conf
vim exec_source_avro_sink.conf
```

```
# Name the components on this agent
<p class="mume-header " id="name-the-components-on-this-agent-2"></p>

a1.sources = r1 r2 r3
a1.sinks = k1
a1.channels = c1
# Describe/configure the source
<p class="mume-header " id="describeconfigure-the-source-2"></p>

a1.sources.r1.type = exec
a1.sources.r1.command = tail -F /export/servers/taillogs/access.log
a1.sources.r1.interceptors = i1
a1.sources.r1.interceptors.i1.type = static
## static拦截器的功能就是往采集到的数据的header中插入自己定## 义的key-value对
<p class="mume-header " id="static拦截器的功能就是往采集到的数据的header中插入自己定-义的key-va

a1.sources.r1.interceptors.i1.key = type
a1.sources.r1.interceptors.i1.value = access

a1.sources.r2.type = exec
a1.sources.r2.command = tail -F /export/servers/taillogs/nginx.log
a1.sources.r2.interceptors = i2
a1.sources.r2.interceptors.i2.type = static
a1.sources.r2.interceptors.i2.key = type
a1.sources.r2.interceptors.i2.value = nginx

a1.sources.r3.type = exec
a1.sources.r3.command = tail -F /export/servers/taillogs/web.log
a1.sources.r3.interceptors = i3
a1.sources.r3.interceptors.i3.type = static
a1.sources.r3.interceptors.i3.key = type
a1.sources.r3.interceptors.i3.value = web

# Describe the sink
<p class="mume-header " id="describe-the-sink-2"></p>

a1.sinks.k1.type = avro
a1.sinks.k1.hostname = node03
a1.sinks.k1.port = 41414
# Use a channel which buffers events in memory
<p class="mume-header " id="use-a-channel-which-buffers-events-in-memory-2"></p>

a1.channels.c1.type = memory
a1.channels.c1.capacity = 20000
a1.channels.c1.transactionCapacity = 10000

# Bind the source and sink to the channel
<p class="mume-header " id="bind-the-source-and-sink-to-the-channel-2"></p>

a1.sources.r1.channels = c1
a1.sources.r2.channels = c1
a1.sources.r3.channels = c1
a1.sinks.k1.channel = c1
```

服务端配置文件开发

在node03上面开发flume配置文件

```
cd /export/servers/apache-flume-1.6.0-cdh5.14.0-bin/conf
vim avro_source_hdfs_sink.conf
```

```
a1.sources = r1
a1.sinks = k1
a1.channels = c1
# 定义source
<p class="mume-header " id="定义source"></p>

a1.sources.r1.type = avro
a1.sources.r1.bind = 192.168.174.120
a1.sources.r1.port =41414

# 添加时间拦截器
<p class="mume-header " id="添加时间拦截器"></p>

a1.sources.r1.interceptors = i1
a1.sources.r1.interceptors.i1.type = org.apache.flume.interceptor.TimestampInterceptor$
# 定义channels
<p class="mume-header " id="定义channels"></p>

a1.channels.c1.type = memory
a1.channels.c1.capacity = 20000
a1.channels.c1.transactionCapacity = 10000
# 定义sink
<p class="mume-header " id="定义sink"></p>

a1.sinks.k1.type = hdfs
a1.sinks.k1.hdfs.path=hdfs://192.168.174.100:8020/source/logs/%{type}/%Y%m%d
a1.sinks.k1.hdfs.filePrefix =events
a1.sinks.k1.hdfs.fileType = DataStream
a1.sinks.k1.hdfs.writeFormat = Text
# 时间类型
<p class="mume-header " id="时间类型"></p>

a1.sinks.k1.hdfs.useLocalTimeStamp = true
# 生成的文件不按条数生成
<p class="mume-header " id="生成的文件不按条数生成"></p>

a1.sinks.k1.hdfs.rollCount = 0
# 生成的文件按时间生成
<p class="mume-header " id="生成的文件按时间生成"></p>

a1.sinks.k1.hdfs.rollInterval = 30
# 生成的文件按大小生成
<p class="mume-header " id="生成的文件按大小生成"></p>

a1.sinks.k1.hdfs.rollSize  = 10485760
# 批量写入hdfs的个数
<p class="mume-header " id="批量写入hdfs的个数"></p>

a1.sinks.k1.hdfs.batchSize = 10000
# flume操作hdfs的线程数（包括新建，写入等）
<p class="mume-header " id="flume操作hdfs的线程数包括新建写入等"></p>

a1.sinks.k1.hdfs.threadsPoolSize=10
# 操作hdfs超时时间
```

```
<p class="mume-header " id="操作hdfs超时时间"></p>

a1.sinks.k1.hdfs.callTimeout=30000
# 组装source、channel、sink
<p class="mume-header " id="组装source-channel-sink"></p>

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

采集端文件生成脚本

在node01与node02上面开发shell脚本，模拟数据生成

```
cd /export/servers/shells
vim server.sh


# !/bin/bash
<p class="mume-header " id="binbash"></p>


while true

do

 date >> /export/servers/taillogs/access.log;

 date >> /export/servers/taillogs/web.log;

 date >> /export/servers/taillogs/nginx.log;

  sleep 0.5;

done
```

顺序启动服务

node03启动flume实现数据收集

```
cd /export/servers/apache-flume-1.6.0-cdh5.14.0-bin

bin/flume-ng agent -c conf -f conf/avro_source_hdfs_sink.conf -name a1 -Dflume.root.log
```

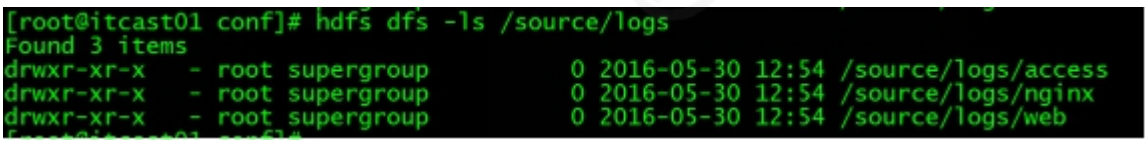
node01与node02启动flume实现数据监控

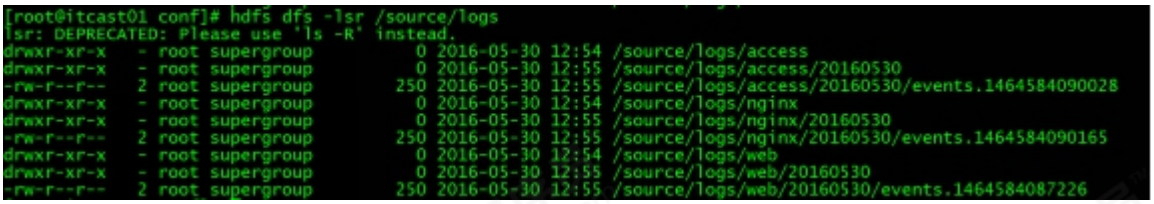
```
cd /export/servers/apache-flume-1.6.0-cdh5.14.0-bin

bin/flume-ng agent -c conf -f conf/exec_source_avro_sink.conf -name a1 -Dflume.root.log
```

node01与node02启动生成文件脚本

```
cd /export/servers/shells
sh server.sh
```

5、项目实现截图





6. Flume 案例二