Spark企业级大数据项目实战 第7课

DATAGURU专业数据分析社区

法律声明

【声明】本视频和幻灯片为炼数成金网络课程的教学资料，所有资料只能在课程内使用，不得在课程以外范围散播，违者将可能被追究法律和经济责任。

课程详情访问炼数成金培训网站

http://edu.dataguru.cn

炼数成金逆向收费式网络课程

- Dataguru（炼数成金）是专业数据分析网站，提供教育，媒体，内容，社区，出版，数据分析业务等服务。我们的课程采用新兴的互联网教育形式，独创地发展了逆向收费式网络培训课程模式。既继承传统教育重学习氛围，重竞争压力的特点，同时又发挥互联网的威力打破时空限制，把天南地北志同道合的朋友组织在一起交流学习，使到原先孤立的学习个体组合成有组织的探索力量。并且把原先动辄成千上万的学习成本，直线下降至百元范围，造福大众。我们的目标是：低成本传播高价值知识，构架中国第一的网上知识流转阵地。
- 关于逆向收费式网络的详情，请看我们的培训网站 http://edu.dataguru.cn

本课内容

- 离线项目问题分析
- 项目ETL流程分析
- Flume核心概念、使用、规划

1 离线项目问题分析

- 数据量非常大
  1个月数据量：50G*60*24*30 =2.1 PB
  通过增加磁盘或者主机实现HDFS容量扩容。

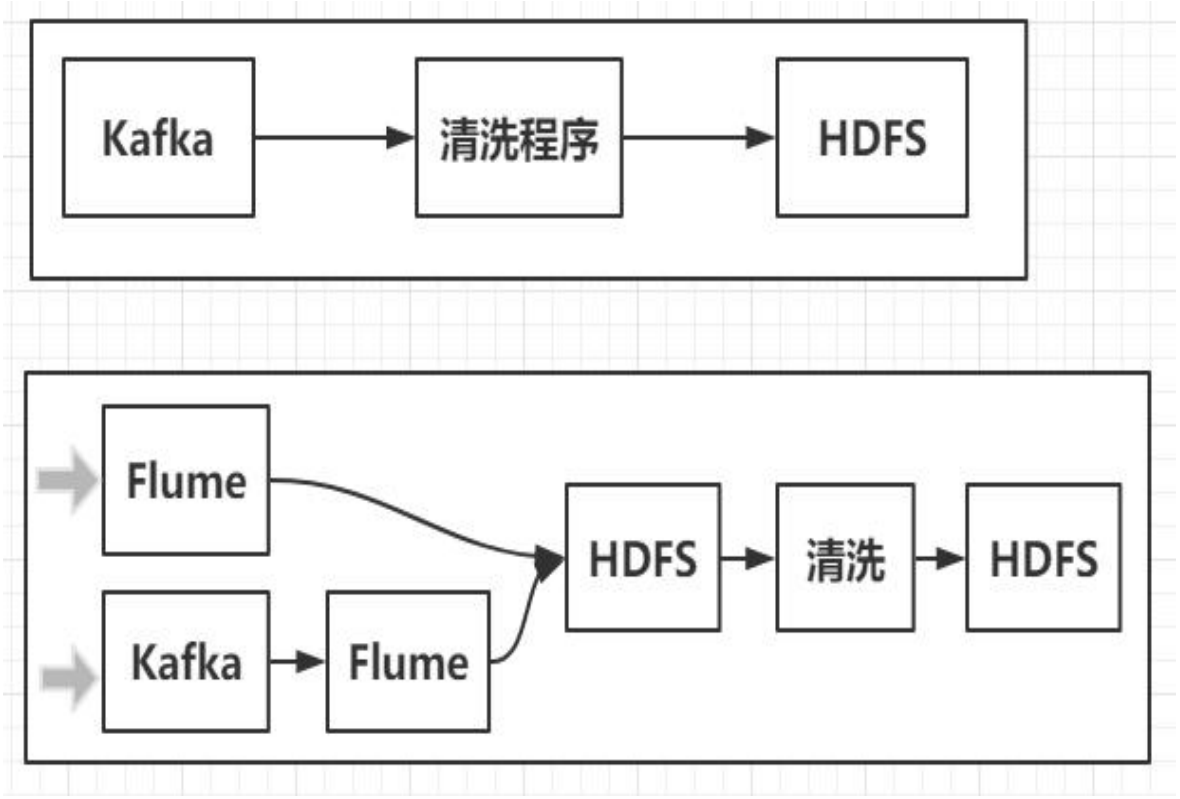
- 小文件
  小文件过多，影响NameNode性能。
  单个DataNode数据块过多也会影响DataNode的性能，CDH官方推荐每台DataNode不超过500, 000个数据块。

- 数据量的增加导致DataNode扩容
  集群整体的CPU和内存使用率不高， 造成浪费
  分散保存在集群各个节点。

- 数据处理效率
  MR或者Spark任务拉取数据计算，数据本地性差， 需要跨越集群的节点数量非常多。计算效率低。

- 数据准确性问题

2 项目ETL流程分析



Kafka数据

有如下几种方式清洗**Kafka**数据：

- 使用Flume将Kafka数据抽取到HDFS，再清洗一次入HDFS。
- 使用Spark Streaming读取Kafka数据， 清洗入HDFS。
- 使用Spark读取Kafka的离线API， 清洗入HDFS。

直接对接Flume的数据

- 使用Flume将数据读取到HDFS， 再使用Spark 清洗HDFS数据。

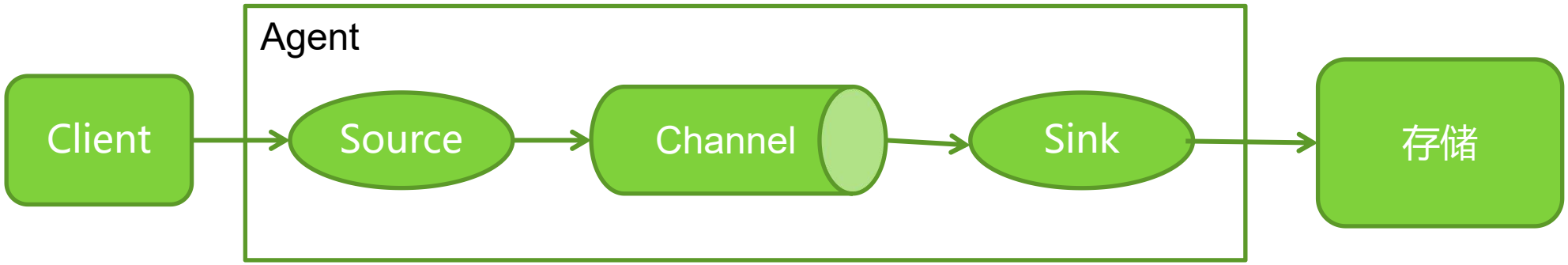
清洗关键点： 文件格式、根据数据的实际时间分区

3 Flume 简介

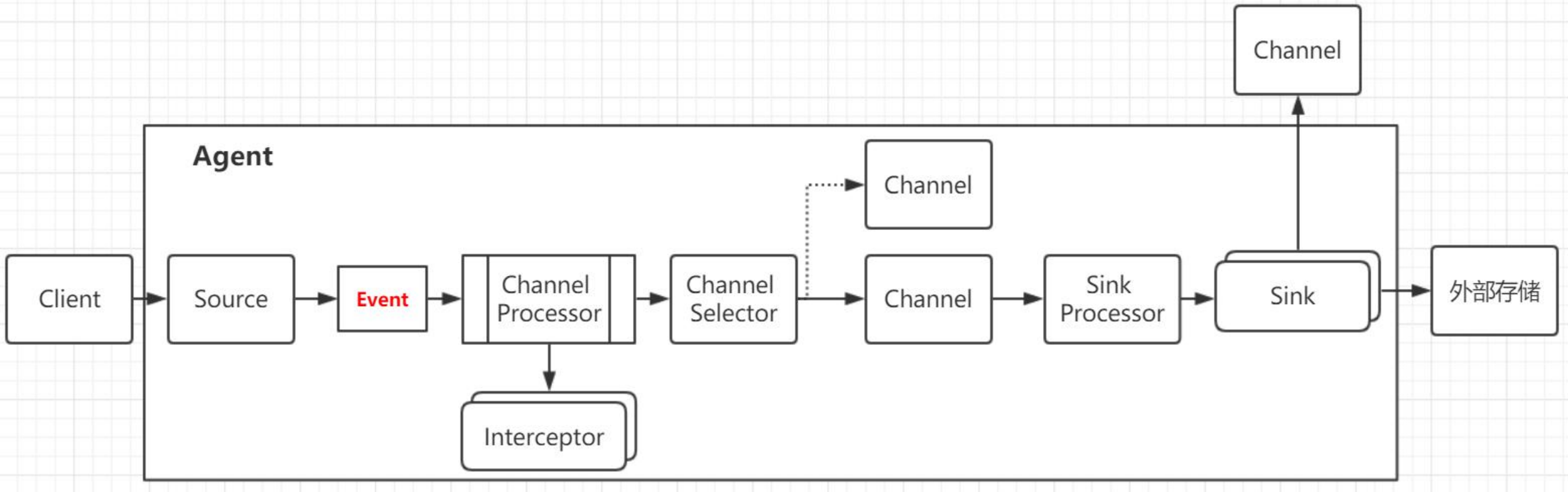
- Flume是一个分布式、高可用、可靠的海量日志采集、聚合和传输系统。
  - 故障转移、恢复机制。
  - 可靠性机制。
  - 可扩展数据模型。

- 数据传输流程
  - 将数据（events）从clients传输到sinks。

注意： 这里的数据指是event（即每条数据）， 而不是文件。



3 Flume总体架构图



组件：Client、Source、Agent、Event、Interceptor、Channel Selector、Channel、Sink Processor、Sink

3 Flume一个简单的例子

```
# 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
```

Flume启动脚本：

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

使用telnet发送event：

```
$ telnet localhost 44444
```

3.1 核心概念

组件	功能
Client	生成事件（event）并将它们发送给一个或多个agent的实体
Agent	用来托管Sources、Channels、Sinks等组件的JVM容器。 将event从source传送到sink。
Source	从特定位置或机制接收事件并将其置于一个或多个Channel上的活动组件。
Event	event是flume中处理消息的基本单元，由零个或者多个header和正文body组成。
Interceptor	根据需要将拦截器应用于Source，装饰和过滤event，甚至删除event。支持多个拦截器的链接。
Channel Selector	通道选择器可以根据当前的标准从所有配置的通道中选择一个或多个Channel
Channel	用于缓冲传入event，直到它们被sink消费。
Sink Processor	负责从一个指定sink组中调用一个sink。
Sink	活动组件，用于从Channel消费事件（event）并将它们传输到其下一个跃点目标（外部存储或下游Agent）

核心概念-Client

生成事件（**event**）并将它们发送给一个或多个**agent**的实体

- **Client**例子：
  - Log4j、Flume Client SDK
- 不是必须的组件。
- 将**Flume**从生成**event**数据的系统分离

核心概念-Agent

托管Sources、Channels、Sinks等组件的JVM容器。 将event从source传送到sink。

- **Flume流基本组成部分**
- **为托管组件提供配置、生命周期管理和监控支持**
- **轻量级配置**

核心概念-Source

从特定位置或机制接收事件并将其置于一个或多个Channel上的活动组件。

- **Flume的Source消费从外部数据源传递过来的events， 比如web server。**
- 常用的**Source**
  - Syslog、Netcat、Avro、TAILDIR、spooldir、kafka、JMS
- **一个Source可以将event发送给1-N个channel**

核心概念-Channel

用于缓冲传入event，直到它们被sink消费。可以把Channel理解为一个消息队列。

- **常用Channel类型**
  - Memory、JDBC、File、Spillable Memory、Custom、Kafka
- **Channel 使用事务机制保证数据的可靠性**
- **Sink端的写入速度应当大于Source端摄入数据的速度**

核心概念-Sink

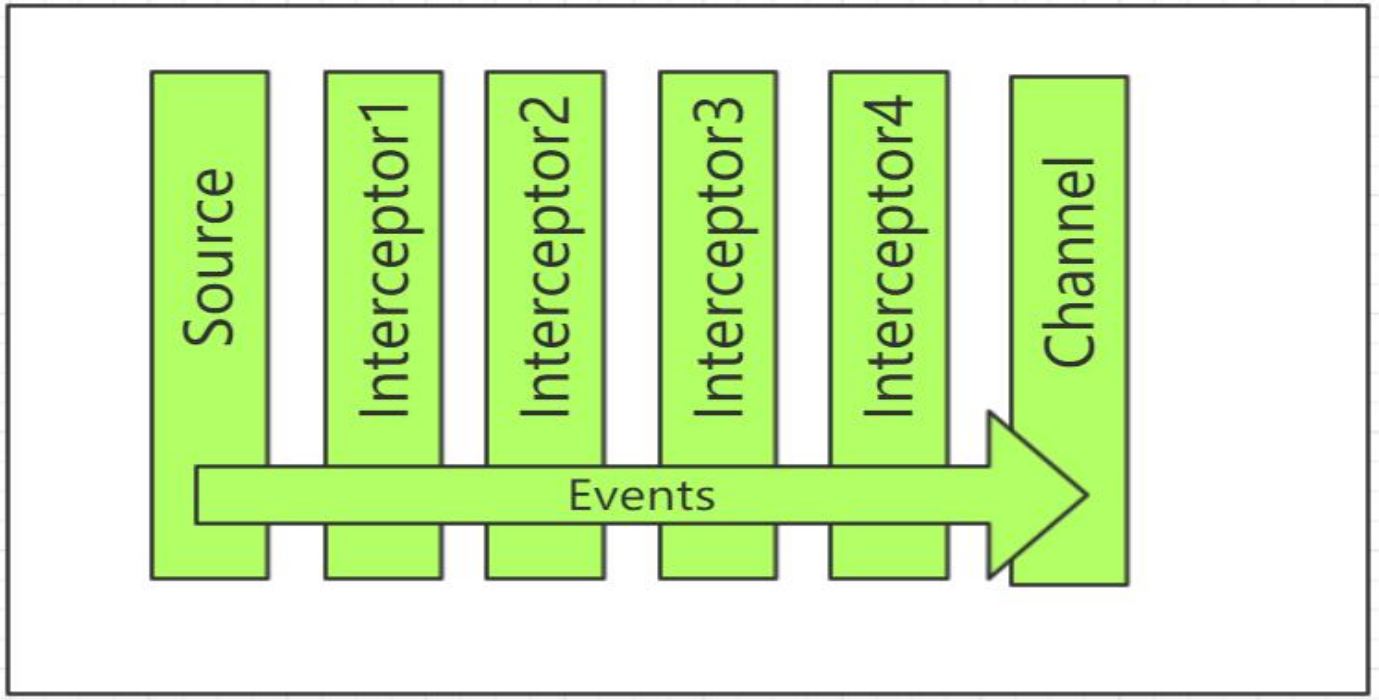
用于从Channel消费事件（event）并将它们传输到其下一个跃点目标（外部存储或下游Agent）

- **常见Sink**
  - HDFS、Kafka、Hbase、Avro
- **一个sink只能从一个Channel获取Event**

核心概念-Interceptors

根据需要将拦截器应用于Source，装饰和过滤event，甚至删除event。

- **内置拦截器允许添加headers信息：时间戳、主机名、静态标记**
- **支持多个拦截器的链接**



核心概念-Channel Selectors

可以根据当前的标准从所有配置的通道中选择一个或多个Channel

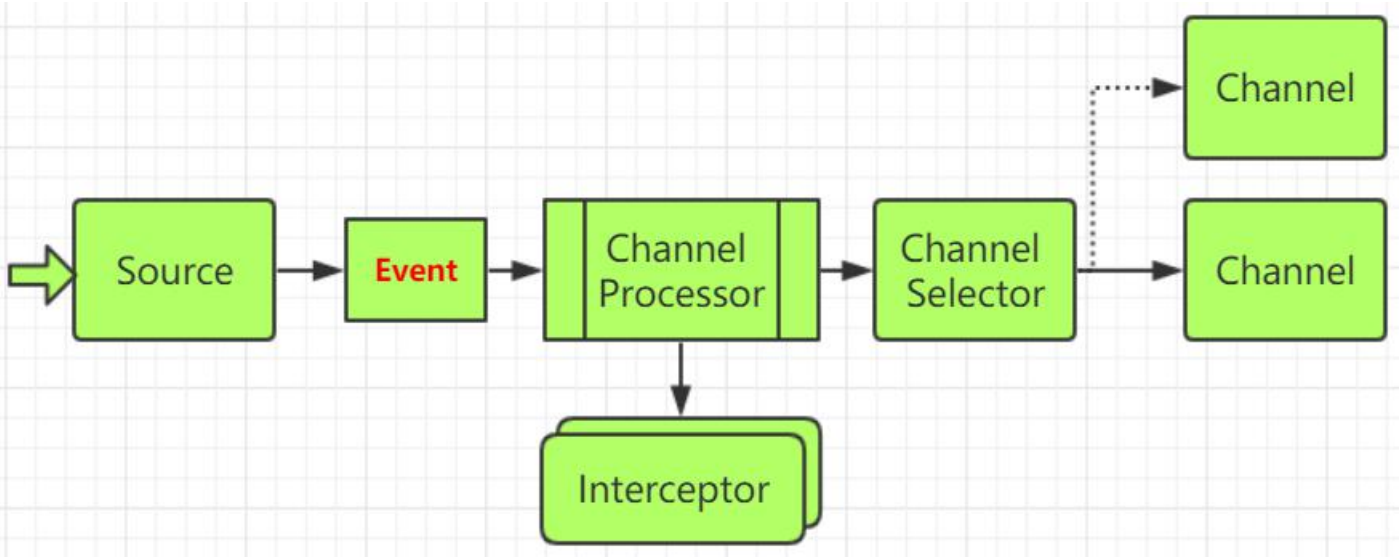
- **内置Channel Selectors**
  - **Replicating**：Source以复制的方式将event写入到一个或多个Channel中
  - **Multiplexing**：根据Source中Event的header不同的键值路由到相应的Channel中。
- **定制Channel Selectors**

核心概念-Sink Processor

负责协调从sink组中调用一个sink。

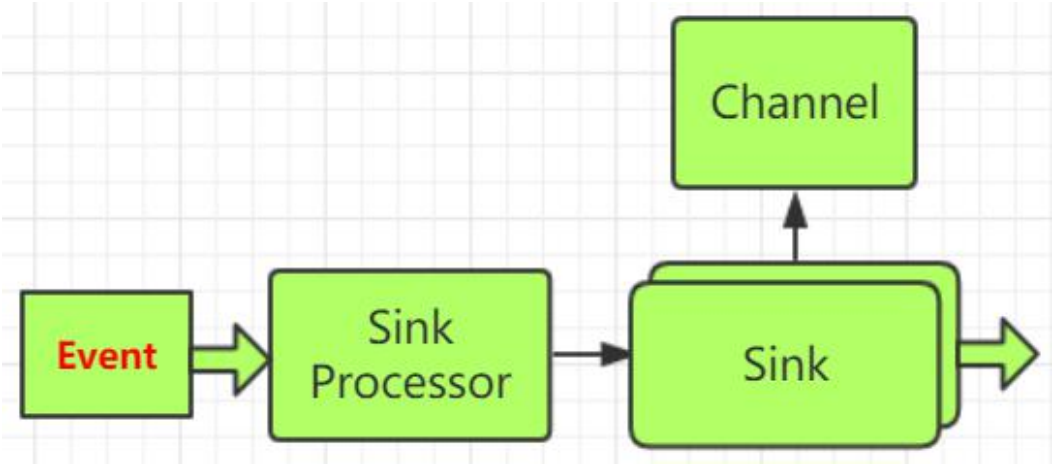
- **Sink Group： 将多个sink放到一个组中。**
- **内置Sink Processor**
  - **Load Balancing：**负载均衡处理器，使用Random、Round_robin或者定制算法
  - **Failover：**容错处理器，根据配置的Sink优先级选择Sink， 值越大， 优先级越高， 运行时只有一个Sink工作。
  - **Default:** 默认处理器， 不需要创建**Sink Processor**

3.1 Agent数据摄取



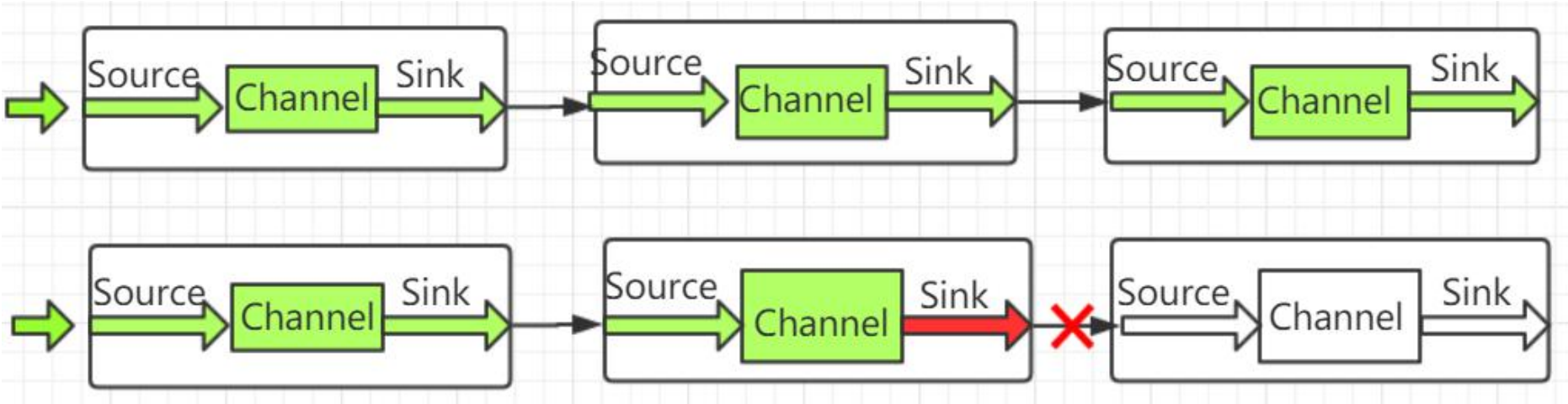
1. Client 向Agent发送Event
2. Source通过与Agent协作接收Event
3. 如果配置了Interceptor，Source将Event发送给Interceptor
4. 根据配置的Channel Selector， Event被放置到Channel中

3.1 Agent数据流出



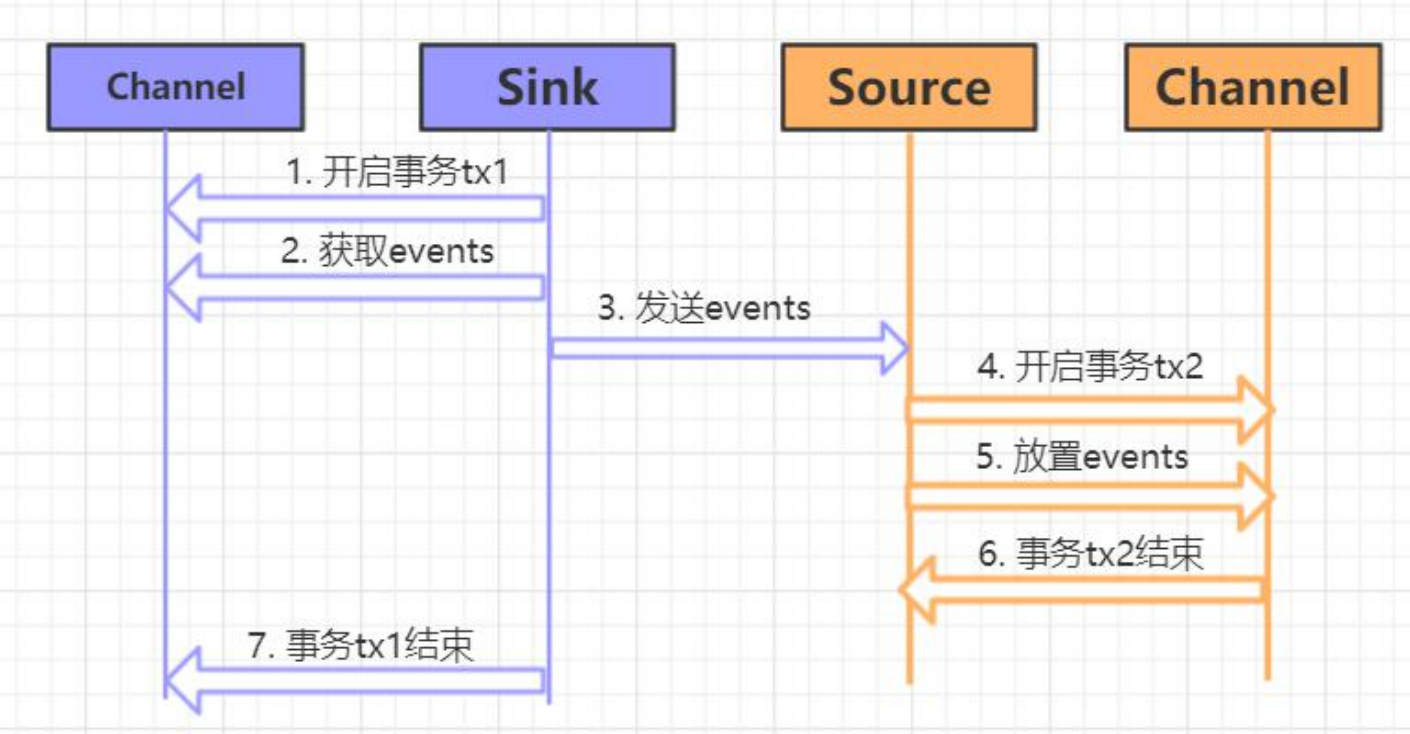
1. Sink Processor从配置组识别一个Sink并调用。
2. 调用的Sink从配置的Channel中获取数据并发送到下一跳目标地。
3. Event从Channel移除是事务性的。

3.1 Agent 管道FLow



1. Agents传输Event到下一跳的Agent
2. 如果传输失败， 数据将被保存在第二个Agent的Channel中。

3.1 数据传输保证



1. Agents使用事务交换保证数据跨Agen传递。
2. 持久化的Channel保证数据在传输过程中不丢失

3.2 FLume使用-Avro Source

Avro Source是FLume RPC Source， 支持Avro协议的RPC服务端。

侦听Avro端口并从外部Avro客户端流接收事件。 当与另一个（上一跳）Flume代理上的内置Avro Sink配对时，它可以创建分层集合拓扑。

http://flume.apache.org/FlumeUserGuide.html#avro-source

重要参数说明：

type： 类型名， 设置为avro

bind：绑定的ip地址或主机名。使用0.0.0.0表示绑定机器所有的接口

port： 绑定的端口

threads： 接收从客户端或Avro Sink传入的数据的最大工作线程的数量。

compression-type： 用于解压缩传入数据的压缩格式。可以设置为none 或 deflate,默认是none. 注意： 上游的avro sink也需要设置压缩格式。

3.2 FLume使用-Spooling Directory Source

Spooling Directory Source监控文件目录。
一个文件被完全读入Channel后， 文件会被重命名，默认给文件添加后缀名.COMPLETED。
相比EXEC Source， 要可靠。支持正则匹配文件名和忽略文件名。
不支持目录嵌套。
文件的内容不允许动态追加，Flume抛出异常并终止。
不同出现同名覆盖文件，Flume抛出异常并终止。

重要参数说明：

type： 类型名， 设置为spooldir
spoolDir：监控的目录路径
fileSuffix： 文件传输完成后添加的后缀名， 默认是.COMPLETED
deletePolicy：是否删除传输完成后的文件，never（默认） 、 immediate
fileHeader： 是否添加文件的绝对路径到event的header
fileHeaderKey： 文件绝对路径的key值， 默认file

http://flume.apache.org/FlumeUserGuide.html#spooling-directory-source

3.2 FLume使用-Taildir Source

Taildir Source： 实时监测指定文件新追的内容。可以通过正则匹配监控目录下文件。 支持断点续传， 支持目录嵌套。

http://flume.apache.org/FlumeUserGuide.html#taildir-source

重要参数说明：

type： 类型名， 设置为TAILDIR
filegroups：空格分隔的文件组列表。 每个文件组指示一组要加尾的文件
filegroups.<filegroupName>：文件组的绝对路径。 正则表达式只能用于文件名。
positionFile： 以JSON格式文件记录每个tailing文件的inode、绝对路径和最后位置。
idleTimeout： 时间（毫秒）关闭不活动的文件。 如果关闭的文件附加了新行，该源将自动重新打开它。
writePosInterval：向positionFile写入每个文件的最后位置的间隔时间（毫秒），默认3000
maxBackoffSleep：每次重新尝试轮询新数据时，最后一次尝试未找到任何新数据时的最大时间延迟。

3.2 FLume使用-Taildir Source例子

```
a1.sources.r1.type = TAILDIR
a1.sources.r1.channels = c1
a1.sources.r1.positionFile = /tmp/log/flume/taildir_position.json
a1.sources.r1.filegroups = f1 f2
a1.sources.r1.filegroups.f1 = /tmp/log/test1/example.log
a1.sources.r1.headers.f1.headerKey1 = value1
a1.sources.r1.filegroups.f2 = /tmp/log/test2/.*log.*
a1.sources.r1.headers.f2.headerKey1 = value2
a1.sources.r1.headers.f2.headerKey2 = value2-2
a1.sources.r1.fileHeader = true
```

文件绝对路径、记录的最后位置保存在文件：/tmp/log/flume/taildir_position.json
监控/tmp/log/test1/example.log和/tmp/log/test2/.*log.*
f1组的文件绝对路径key为value2， f2组的文件绝对路径key为value2-2

3.2 FLume使用-Kafka Source

Kafka Source是一个从Kafka的 Topic中读取消息的Apache Kafka消费者。 如果有多个Kafka source运行，您可以使用相同的Consumer Group配置它们，因此每个将读取topic中一组唯一的分区。

Kafka Source 保证至少一次消息检索策略

主要的几个属性：

type:　source的类型，org.apache.flume.source.kafka.KafkaSource

kafka.bootstrap.servers：Kafka Broker

kafka.consumer.group.id ：消费者组的唯一标识。 在多个来源或代理中设置相同的ID表示它们是同一个使用者组的一部分

kafka.topics ： 使用逗号隔开的kafka 消费者从那些topics获取的消息

kafka.topics.regex： 使用正则表达是定义该Source订阅的topic的集合，此属性的优先级高于kafka.topics， 并覆盖kafka.topics（如果存在）。

batchSize： 一个批次中写入通道的最大消息数

3.2 FLume使用-Kafka Source例子

topic名称使用逗号分隔：

```
tier1.sources.source1.type = org.apache.flume.source.kafka.KafkaSource
tier1.sources.source1.channels = channel1
tier1.sources.source1.batchSize = 5000
tier1.sources.source1.batchDurationMillis = 2000
tier1.sources.source1.kafka.bootstrap.servers = localhost:9092
tier1.sources.source1.kafka.topics = test1, test2
tier1.sources.source1.kafka.consumer.group.id = custom.g.id
```

topic名称使用正则匹配：

```
tier1.sources.source1.type = org.apache.flume.source.kafka.KafkaSource
tier1.sources.source1.channels = channel1
tier1.sources.source1.kafka.bootstrap.servers = localhost:9092
tier1.sources.source1.kafka.topics.regex = ^topic[0-9]$
# the default kafka.consumer.group.id=flume is used
```

3.2 FLume使用-Memory Channel

Memory Channel把Event保存在内存队列中，该队列能保存的Event数量有最大值上限。由于Event数据都保存在内存中，Memory Channel有最好的性能，不过也有数据可能会丢失的风险，如果Flume崩溃或者重启，那么保存在Channel中的Event都会丢失。同时由于内存容量有限，当Event数量达到最大值或者内存达到容量上限，Memory Channel会有数据丢失。

主要的几个属性：

type：值为memory

capacity：channel中保存event的最大容量。

transactionCapacity：Channel每次提交的Event数量， 默认100

3.2 FLume使用-File Channel

File Channel把Event保存在本地硬盘中，比Memory Channel提供更好的可靠性和可恢复性，但是性能要差。

属性	默认值	描述
type	file	名称
checkpointDir	~/.flume/file-channel/checkpoint	检测点文件所存储的目录
useDualCheckpoints	FALSE	备份检测点如果设置为true，backupChec kpointDir必须设置
backupCheckpointDir	-	备份检测点的备份到所在的目录，不要与数据检测点或者目录重复
dataDirs	~/.flume/file-channel/data	数据存储所在的目录设置
transactionCapacity	1000	事务容量的最大值设置
checkpointInterval	30000	检测点之间的时间值设置（单位微秒）
maxFileSize	2146435071	一个单一日志的最大值设置（以字节为单位）
minimumRequiredSpace	524288000	最小的请求闲置空间（以字节为单位）
capacity	1000000	隧道的最大容量
keep-alive	3	一个存放操作的等待时间值（秒）设置
write-timeout	3	一个写操作的等待时间值（秒）设置

3.2 FLume使用-File Channel例子

```
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

# Use a channel which buffers events in File
a1.channels = c1
a1.channels.c1.type = file
a1.channels.c1.checkpointDir = /tmp/flume/checkpoint
a1.channels.c1.dataDirs = /tmp/flume/data

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

3.2 FLume使用-Kafka Channel

以kafka的一个topic为channel，相比其他channel类型， Kafka Channel兼并了快和安全的要求

主要的几个属性：

type:　source的类型，org.apache.flume.channel.kafka.KafkaChannel
kafka.bootstrap.servers：Kafka Broker
kafka.consumer.group.id ：消费者组的唯一标识。
kafka.topic： 指定kafka的topic保存channel， 默认flume-channel
kafka.consumer.auto.offset.reset： 指定最早或者最新的offset拉取数据。

3.2 FLume使用-Kafka Channel例子

```
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

# Use a channel which buffers events in File
a1.channels = c1
a1.channels.c1.type = org.apache.flume.channel.kafka.KafkaChannel
a1.channels.c1.kafka.bootstrap.servers = localhost:9092,localhost:19092
a1.channels.c1.kafka.topic = test1
a1.channels.c1.kafka.consumer.auto.offset.reset = latest

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

3.2 FLume使用-Avro Sink

Avro Sink用于对接下游的avro source。

主要的几个属性：

type:　Sink的类型，avro
hostname：要绑定的主机名或IP地址
port ：要监听的端口号
batch-size：　批量发送的数量

```
a1.channels = c1
a1.sinks = k1
a1.sinks.k1.type = avro
a1.sinks.k1.channel = c1
a1.sinks.k1.hostname = 10.10.10.10
a1.sinks.k1.port = 4545
```

3.2 FLume使用-HDFS Sink

将event保存到hdfs， 可以基于时间戳将数据保存到hdfs的时间目录中。

主要的几个属性：

hdfs.path: hdfs目录路径
hdfs.filePrefix: 文件前缀。默认值FlumeData
hdfs.fileSuffix: 文件后缀
hdfs.rollInterval: 多久时间后close hdfs文件。单位是秒，默认30秒。设置为0的话表示不根据时间close hdfs文件
hdfs.rollSize: 文件大小超过一定值后，close文件。默认值1024，单位是字节。设置为0的话表示不基于文件大小
hdfs.rollCount: 写入了多少个事件后close文件。默认值是10个。设置为0的话表示不基于事件个数
hdfs.fileType: 文件格式， 有3种格式可选择：SequenceFile, DataStream or CompressedStream
hdfs.batchSize: 批次数，HDFS Sink每次从Channel中拿的事件个数。默认值100
hdfs.minBlockReplicas: HDFS每个块最小的replicas数字，不设置的话会取hadoop中的配置
hdfs.maxOpenFiles: 允许最多打开的文件数，默认是5000。如果超过了这个值，越早的文件会被关闭
serializer: HDFS Sink写文件的时候会进行序列化操作。会调用对应的Serializer借口，可以自定义符合需求的Serializer
hdfs.retryInterval: 关闭HDFS文件失败后重新尝试关闭的延迟数，单位是秒
hdfs.callTimeout: HDFS操作允许的时间，比如hdfs文件的open，write，flush，close操作。单位是毫秒，默认值是10000

3.2 FLume使用-HDFS Sink例子

```
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
a1.sources.r1.interceptors=i1
a1.sources.r1.interceptors.i1.type=timestamp
a1.sources.r1.interceptors.i1.preserveExisting=false
# Describe the sink
a1.sinks.k1.type = hdfs
a1.sinks.k1.hdfs.path = /flume/%Y%m%d
a1.sinks.k1.hdfs.filePrefix = hdfsflume
a1.sinks.k1.hdfs.fileType = DataStream
a1.sinks.k1.hdfs.writeFormat = Text
a1.sinks.k1.hdfs.round = true
a1.sinks.k1.hdfs.roundValue = 1
a1.sinks.k1.hdfs.roundUnit = minute
a1.sinks.k1.hdfs.callTimeout=6000
# 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
```

3.2 FLume使用-Kafka Sink

可以将数据发布到Kafka的Topic。 其中一个目标是将Flume与Kafka集成，以便基于拉式的处理系统可以处理来自各种Flume源的数据。

主要的几个属性：

type: source的类型，org.apache.flume.channel.kafka.KafkaChannel

kafka.bootstrap.servers：Kafka Broker

kafka.topic： Kafka的topic名称

kafka.producer.acks：ack机制：broker表示发来的数据已确认接收无误，表示数据已经保存到磁盘。

0：不等待broker返回确认消息

1：等待topic中某个partition leader保存成功的状态反馈

-1：等待topic中某个partition 所有副本都保存成功的状态反馈

3.2 FLume使用-Kafka Sink例子

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

a1.sources.r1.type = netcat
a1.sources.r1.bind = localhost
a1.sources.r1.port = 44444

a1.sinks.k1.channel = c1
a1.sinks.k1.type = org.apache.flume.sink.kafka.KafkaSink
a1.sinks.k1.kafka.topic = mytopic
a1.sinks.k1.kafka.bootstrap.servers = localhost:9092
a1.sinks.k1.kafka.flumeBatchSize = 20
a1.sinks.k1.kafka.producer.acks = 1
a1.sinks.k1.kafka.producer.linger.ms = 1
a1.sinks.k1.kafka.producer.compression.type = snappy

a1.channels.c1.type = memory
a1.channels.c1.capacity = 1000
a1.channels.c1.transactionCapacity = 100

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

3.2 FLume使用-interceptor

Timestamp Interceptor :在event的header中添加一个key叫：timestamp,value为当前的时间戳。这个拦截器在sink为hdfs 时很有用。

Host Interceptor：在event的header中添加一个key叫：host,value为当前机器的hostname或者ip。

Static Interceptor:可以在event的header中添加自定义的key和value。

Regex Filtering Interceptor:通过正则来清洗或包含匹配的events。

Regex Extractor Interceptor：通过正则表达式来在header中添加指定的key,value则为正则匹配的部分

3.2 FLume使用-interceptor例子

1. 数据入HDFS使用了Timestamp拦截器。见hdfs sink例子。

2. 静态拦截器
   见后面多路复用的例子

3.2 FLume使用-Selector选择器

Replicating Channel Selector (default)：

- **Replicating**：Source以复制的方式将event写入到一个或多个Channel中

主要属性：
selector.type： replicating
selector.optional： 标记为可选的channel

```
a1.sources = r1
a1.channels = c1 c2 c3
a1.sources.r1.selector.type = replicating
a1.sources.r1.channels = c1 c2 c3
a1.sources.r1.selector.optional = c3
```

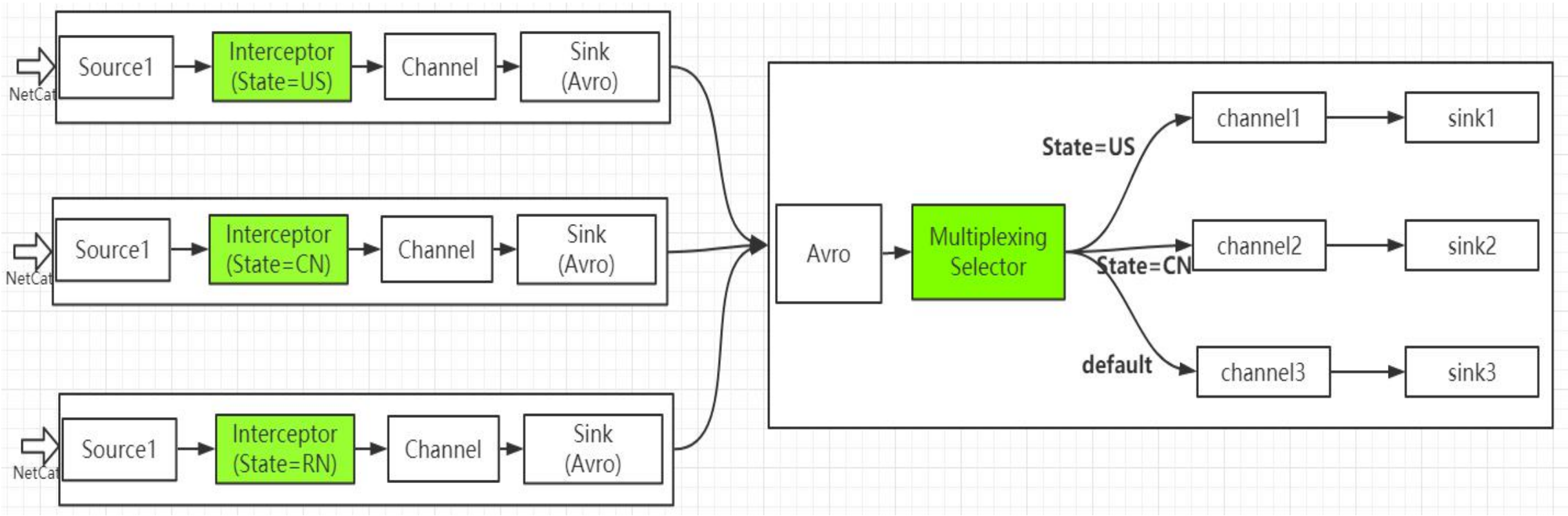
3.2 FLume使用-Selector选择器

Multiplexing：根据Source中Event的header不同的键值路由到相应的Channel中。

主要属性：
selector.type： multiplexing
selector.header： header的key
selector.mapping.* 根据header不同的值， 映射到不同的channel

```
a1.sources = r1
a1.channels = c1 c2 c3 c4
a1.sources.r1.selector.type = multiplexing
a1.sources.r1.selector.header = state
a1.sources.r1.selector.mapping.CZ = c1
a1.sources.r1.selector.mapping.US = c2 c3
a1.sources.r1.selector.default = c4
```

3.2 FLume使用-Selector选择器例子



3.2 FLume使用-Sink Processor

Load balancing Sink Processor：负载均衡处理器，使用Random、Round_robin或者定制算法

主要属性：
sinks： 参与该组的以空格分隔的sink列表
processor.type：load_balance
processor.priority.<sinkName>: 根据配置的Sink优先级选择Sink， 值越大， 优先级越高
processor.selector: 轮询苏算法， Random、Round_robin
processor.backoff： 将失败的sink放入黑名单, 一定时间之后再从黑名单中移除，继续被尝试

```
a1.sinkgroups = g1
a1.sinkgroups.g1.sinks = k1 k2
a1.sinkgroups.g1.processor.type = load_balance
a1.sinkgroups.g1.processor.backoff = true
a1.sinkgroups.g1.processor.selector = random
```

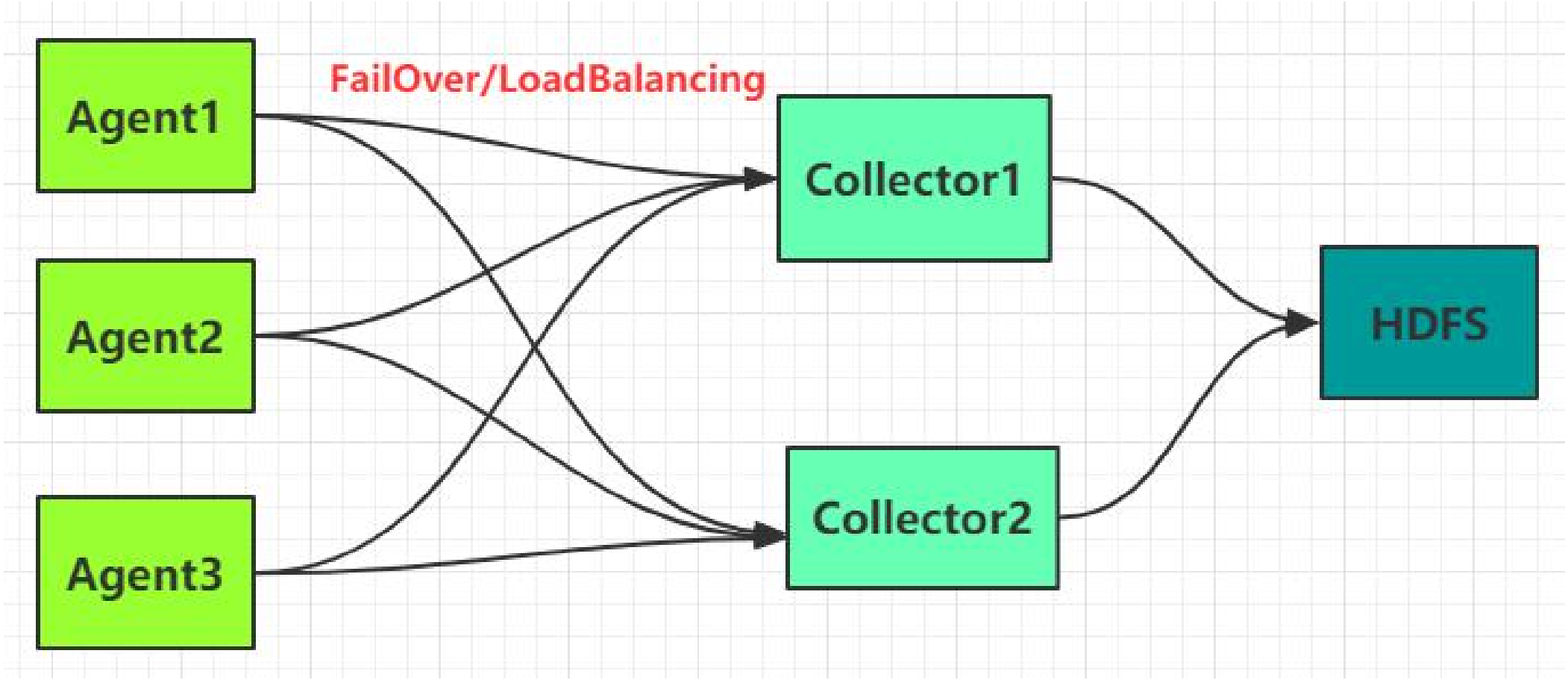
3.2 FLume使用-Sink Processor

Failover Sink Processor：容错处理器，根据配置的Sink优先级选择Sink， 值越大， 优先级越高， 运行时只有一个Sink工作

主要属性：
sinks： 参与该组的以空格分隔的sink列表
processor.type：failover
processor.priority.<sinkName>: 根据配置的Sink优先级选择Sink， 值越大， 优先级越高
processor.maxpenalty: 等待失败sink恢复的最长时间

```
a1.sinkgroups = g1
a1.sinkgroups.g1.sinks = k1 k2
a1.sinkgroups.g1.processor.type = failover
a1.sinkgroups.g1.processor.priority.k1 = 5
a1.sinkgroups.g1.processor.priority.k2 = 10
a1.sinkgroups.g1.processor.maxpenalty = 10000
```

3.2 FLume使用-Sink Processor

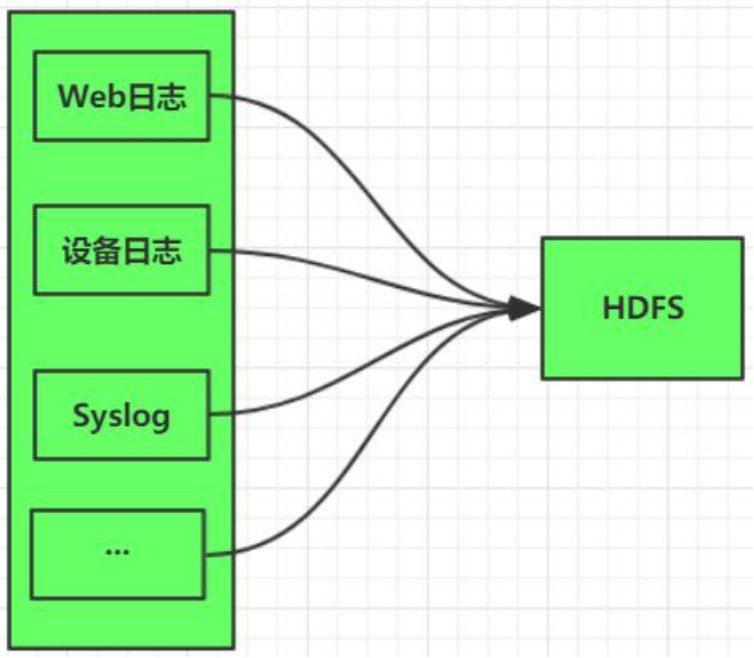


基于源头的数据大小、数据采集的目的地规划Flume拓扑架构

具备缓冲数据峰值的能力

规划满足处理瞬时故障所需的容量

3.3 规划-单层架构



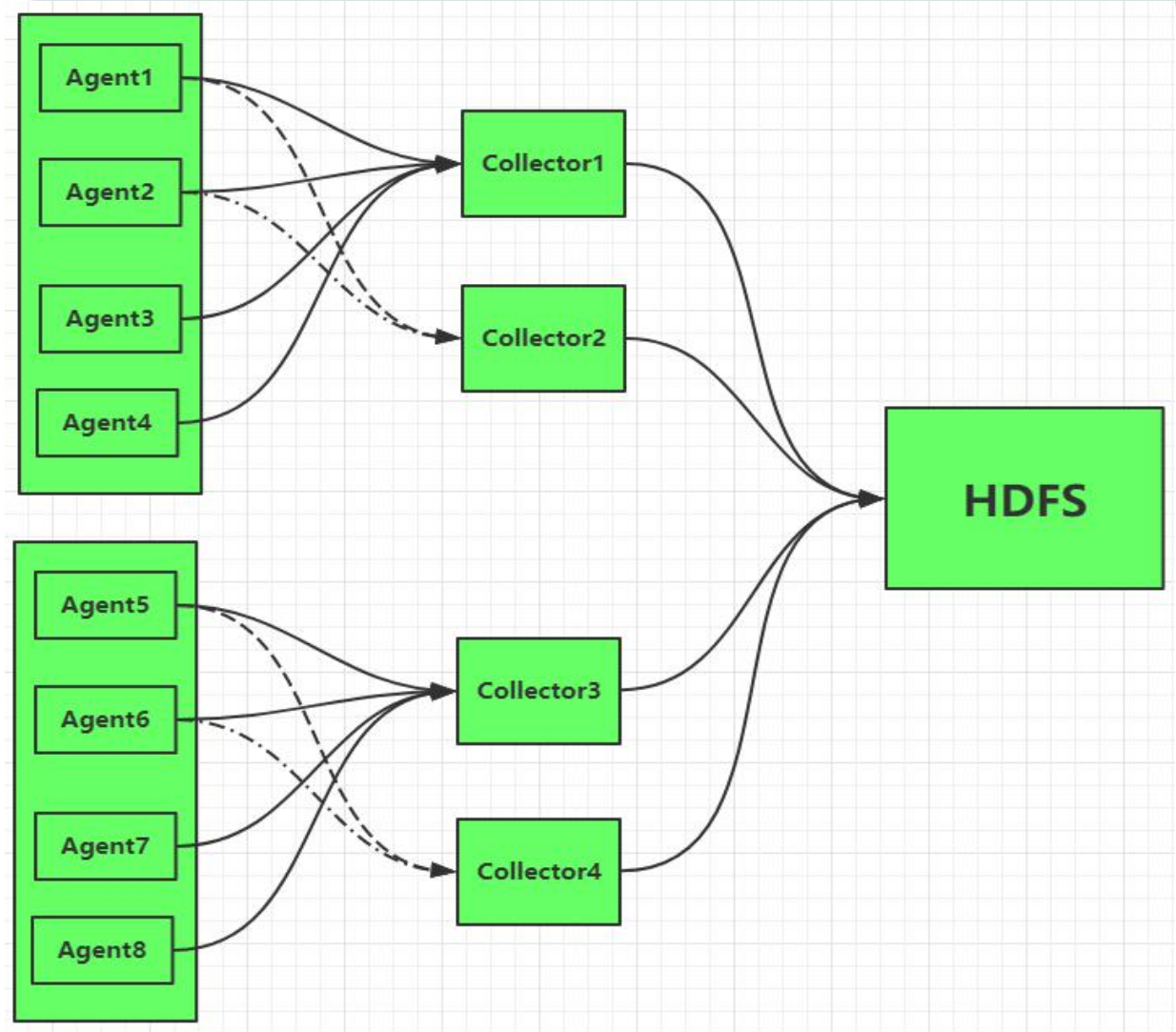
1. 架构简单

2. 配置管理复杂， 维护难度大

3. HDFS频繁写， 小文件多， HDFS压力大

4. 安全性差

3.3 规划-分层架构



1. 安全性
   汇聚层的Agent（Collector1与Collector2、Collector3与Collector4）做负载均衡、高可用

2. HDFS小文件
   汇聚层汇聚第一层多个Agent的event。

3. 扩展性、可用性

4. 升级维护
   HDFS升级维护等， 只需要维护汇聚层的Agent

3.3 规划-规划每层节点数量

经验法则：
每4-16台agent做一层聚合Agent。

- 根据每个聚合Agent的数据摄取能力。
- 理论上：最大的agent数量基于将网络带宽跑满。
- 最外层agent比可高达100:1
- 内层大幅减少agent数量
- 需要考虑Load Balancing、Failover

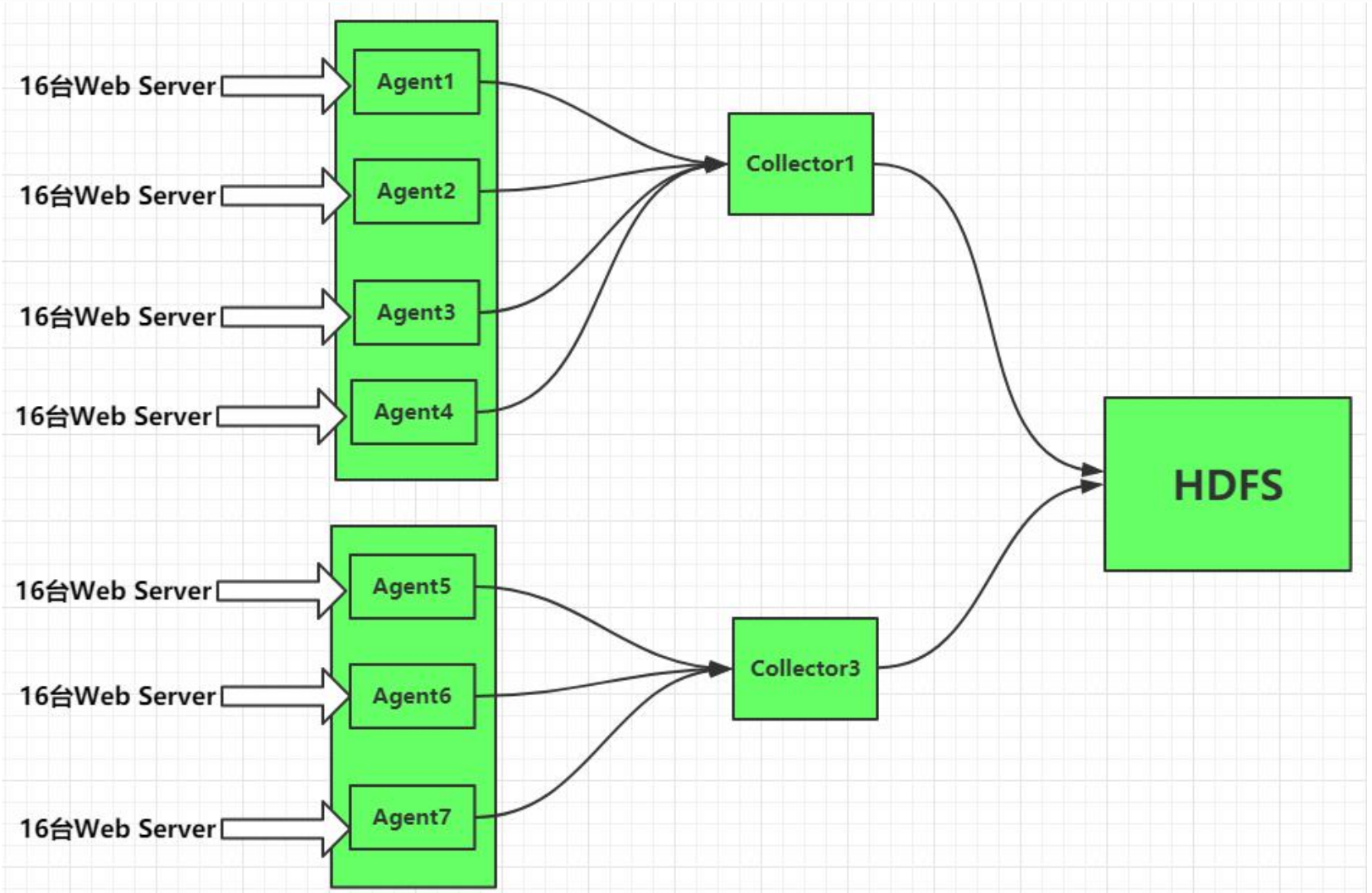
案例： 从100台Web Server 收集日志

1. 第一层：按照1:16规划agent数量， 不考虑load balancing和failover。
第一层的agent数量： 100/16 =7.

2. 中间层：按照1:4规划agent数量， 不考虑load balancing和failover。
第二层的agent数量：7/4 =2.

合计：两层， 9个Agent。

3.3 规划-规划每层节点数量



3.3 规划- Sink Batch Size

基于前面的拓扑， 以Agent1为例：
考虑到稳定性， agent1能处理的最大event数量为1600个。

- Agent1
  1. 在一个周期内， 每台web server发送了100个event。
  2. Agent1的接收的event数量：16*100=1600个，Agent1的出口event数量
  3. 如果web Server的event变大， agent1的 出口event数量增加到2500个， 那么这个时候使用mutilple sinks。

- Collector1
  1. 从上游4个Agent接收数据， 每批次数据量： 4 * 1600 = 6400个。
  2. 将 6400个分成3个sink， 每个sink的batch size为2150.

sink的batch size越大， 数据重复的风险就越大， 因为Flume只能保证每个event被推送到至少一个sink。

3.3 规划-Channel Capacity

Channel容量规划

- 根据故障处理要求规划，下游故障， 导致channel驻留大量的event
- Channel选择
  - Memory: 传输速度快， 可靠性差
  - File: 数据持久化到磁盘， 数据不会丢失。重启断点续传。
  - Kafka Channel： 传输速度快，安全可靠。
- File Channel Capacity规划

  假设能够容忍的下游故障处理时间为1个小时。

  1个agent的传输event速率为100个/秒。

  1个小时的event数为： 1*60*60*100 = 360000

  File的磁盘容量需要满足360000个event的存储， 从安全性考虑， 设计File的磁盘容量为360000*1.5=540000个event的容量。

- Kafka Channel的规划

  假设能够容忍的下游故障处理时间为1个小时

  Kafka的segment配置保留时间大于1个小时。 从安全性考虑，适当增大。

  Kafka的segment配置保留字节大小大于540000个event的容量。

3.3 规划-硬件

CPU核心数：（Source数量 + Sink数量）/ 2

如果使用Memory Channel， 尽量配置比较大的内存

如果使用File Channel， 磁盘越多， 吞吐量越大

Thanks

FAQ时间