Flume

Introduction

- Distributed, reliable system for efficient collection, aggregation and movement of streaming data
- Common uses

  - moving log data

  - event data
    - Social media feeds
    - Message event queues
    - Network traffic data

- Flume sources

  consume events from external sources

  forward it to channels

- External sources

  any system that generates events

  - Social media feed
    - Twitter
    - Machine logs
    - Message queues

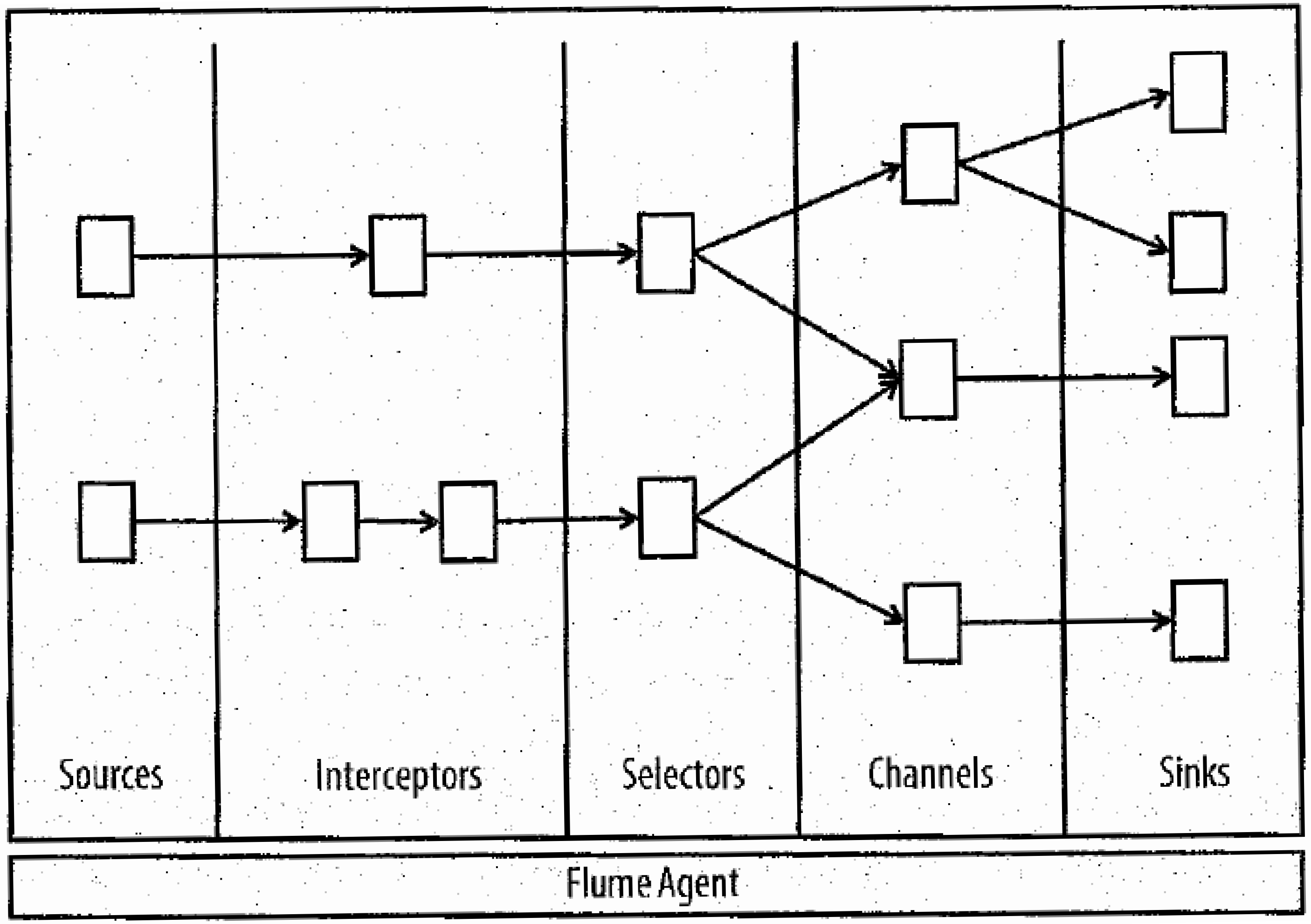
- Flume interceptors

  allow events to be intercepted and modified in flight

  - Transforming the event
  - Enriching the event
  - Anything that can be implemented in a java class

  examples

  - Formatting
  - Partitioning
  - Filtering
  - Splitting
  - Validating
  - …


Sources
Interceptors
Selectors
Channels
Sinks
Flume Agent

- Selectors

  provide routing for events

  to send events down 0 or more paths

  - To fork to multiple channels
  - Send to a specific channel based on event

- Channels

  store events until they are consumed by a sink

  common channels

  - Memory channel
  - File or disk channel

  Memory channel

  - Stores events in memory
  - Best performance
  - Least reliable as events will be lost if process or host goes down

  Disk channel

  - Durable storage of events by persisting to disk

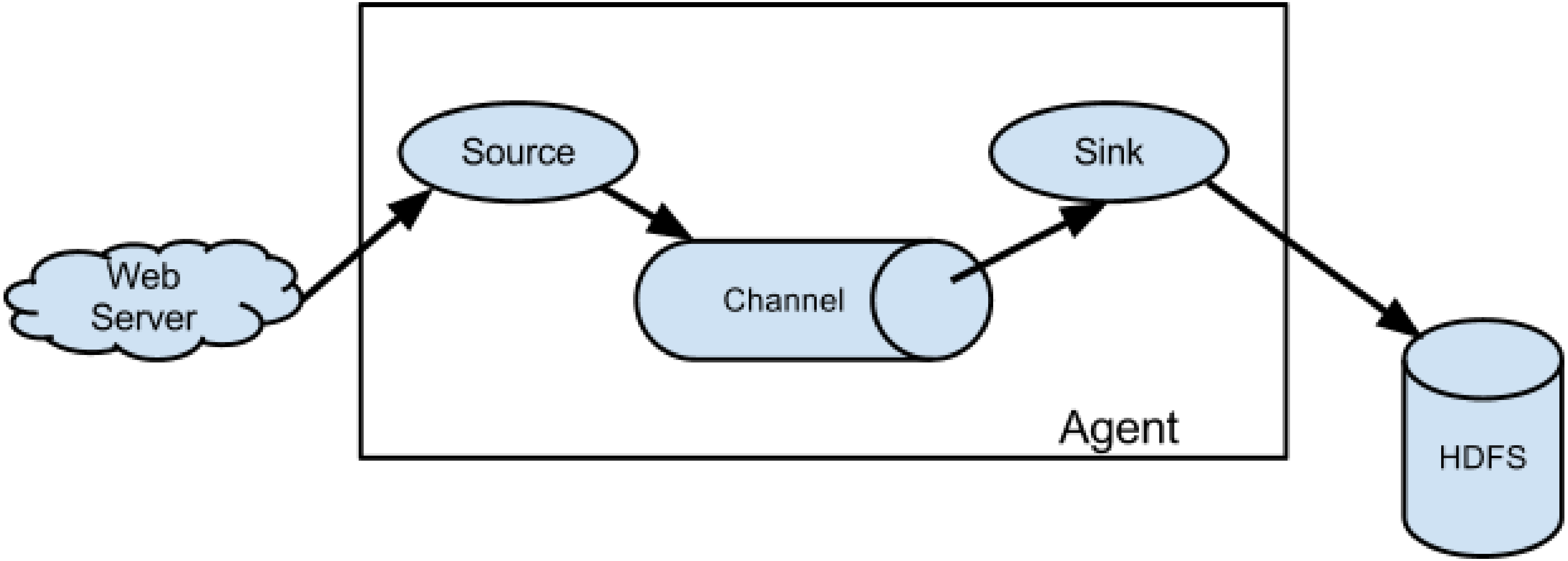
- Sinks

  remove events from a channel and deliver it to a destination destination

  - Final target for events
  - Feed into further Flume processing

  Example

  - HDFS sink – writes events into HDFS files

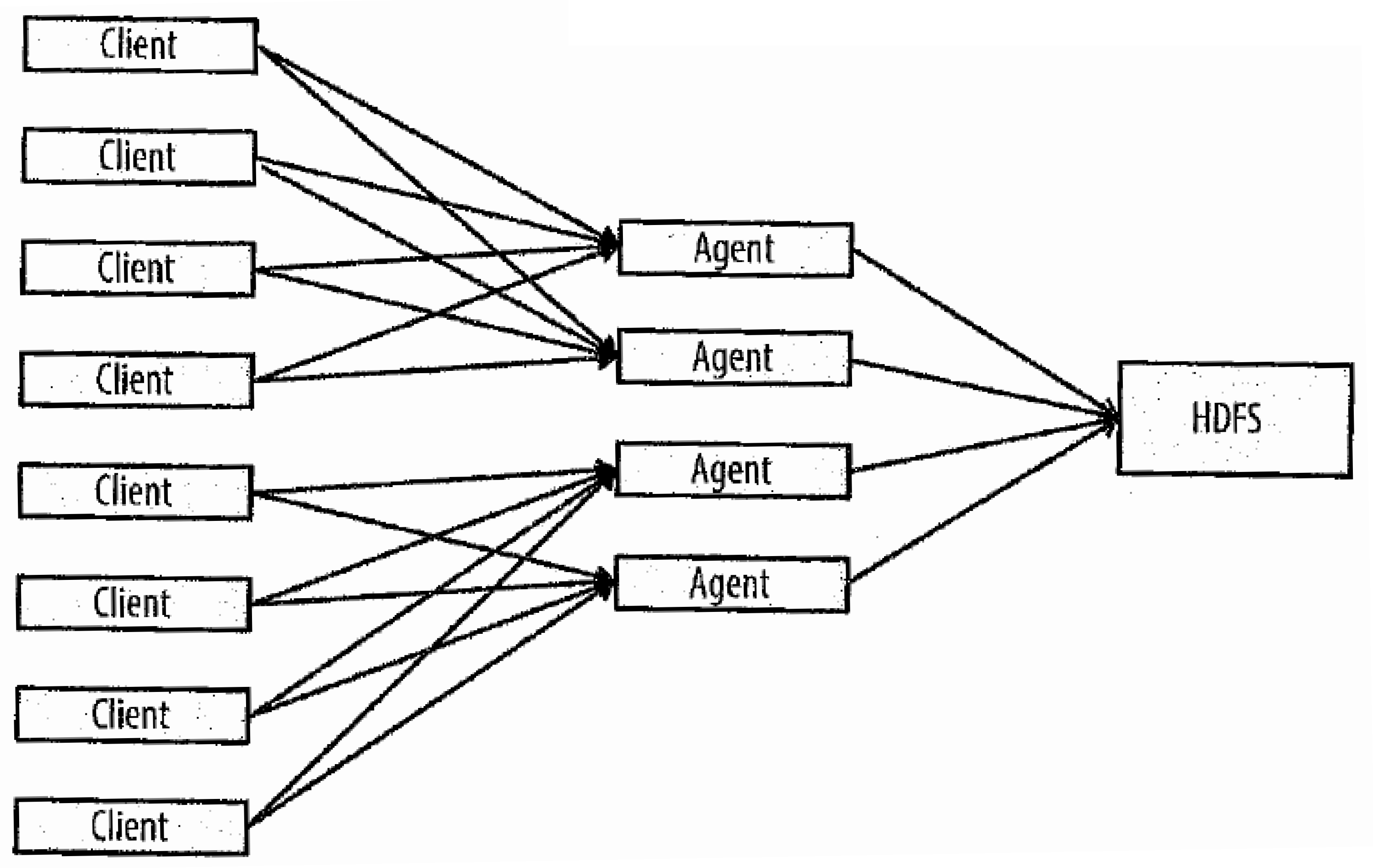

Web Server
Source
Channel
Sink
Agent
HDFS

- Agent

  container

  JVM process hosting of a set of Flume:

  - Sources
  - Sinks
  - Channels
  - …


Client
Client
Client
Client
Client
Client
Client
Client
Agent
Agent
Agent
Agent
HDFS

Flume features and patterns

- Reliability

  events stored in channel until delivered to next stage
- Recoverable

  events persisted to disk and recovered in event of failure
- Declarative

  no coding
  - Configuration specifies how components are wired together
- Highly customizable

  pluggable architecture

- Fan-in

  flume agent on each source system (say web servers)
  - agent sends events to agents on Hadoop edge nodes
- Edge nodes

  nodes on Hadoop cluster
- Features

  multiple edge nodes
  - Reliability – if one edge node goes down, events not lost

  compress events to reduce traffic

  SSL to encrypt data

- Splitting data on ingest

  split events for ingestion into multiple targets

  example

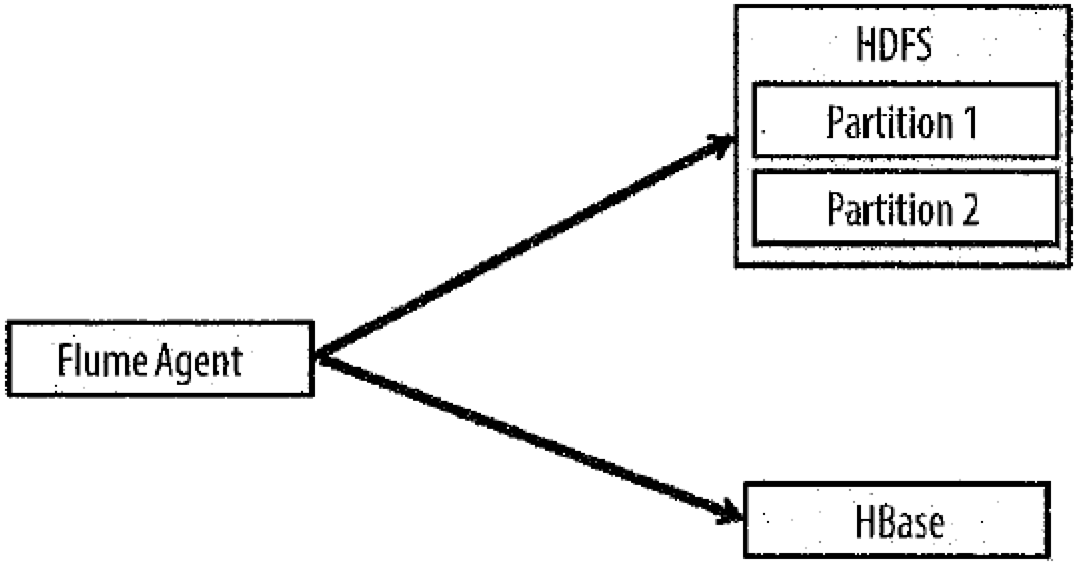
  - Send events into a primary cluster and a backup cluster

- Partitioning data on ingest

  parititition data as it is ingested

  example

  - Partition events by timestamp


HDFS
Partition 1
Partition 2
Flume Agent
HBase

- Flume source consumes events delivered to it by an external source like a web server.
- External source sends events to Flume in a format that is recognized by the target Flume source.
    - Example - Avro Flume source can receive Avro events from Avro clients or
    - other Flume agents in the flow that send events from an Avro sink.
- When Flume source receives an event, it stores it into one or more channels.
- Channel - passive store that keeps the event until it’s consumed by a Flume sink.
    - Example - file channel – backed by the local filesystem.

- Sink removes event from the channel
  - puts it into an external repository like HDFS (via Flume HDFS sink) or
  - forwards it to the Flume source of the next Flume agent (next hop) in the flow.
- Source and sink within agent run asynchronously with events staged in the channel.

- Splitting events for streaming analytics

sending to a streaming analytics engine such as storm or spark streaming

  - Real-time counts
  - Windowing
  - Summaries

spark streaming implement interface for Flume Avro source

  - Point a Flume Avro sink to Spark Streaming's Flume Stream

File Formats

- Text files

  not optimal for HDFS

  better to save to SequenceFiles (default for HDFS sink)

  or save to Avro
- Columnar formats

  RCFile, ORC, Parquet

Best Practices – Flume sources

- Batch size

  Example

  Client sends batch of events to Avro source

  Client must wait until those events are in the channel and Avro source has responded back to client with an acknowledgment of success

  - Latency can be an issue
  - Select appropriate batch size

- Threads

  Avro source

  Can have many connections to it at the same time

  to make Avro source multi-threaded

  - Add more clients or client threads

  JVM source

  pull source

  to get more threads

  - Configure more sources in Flume agent

Best Practices – Flume sinks

- Number of sinks

  sink can fetch data from a single channel

  many sinks can fetch data from that same channel

  sink runs in a single thread

  - Example

    HDFS gives 30Mbps to a single sink

    therefore 30Mbps throughput only

    more sinks consuming from the same channel will resolve this bottleneck

- Batch sizes

  if sink is getting small batches, lot of time lost to executing system calls such as fsink

Best Practices – Channels

- Memory channels

  limit number of memory channels on a single node

  More memory channels on a single node – less memory available to each of these channels

  memory channel can be fed by multiple sources and be fetched from by multiple sinks

- File channels

  multiple file channels write to multiple disks

Flume Configuration File

- Name the components of the current agent.
- Describe/Configure the source.
- Describe/Configure the sink.
- Describe/Configure the channel.
- Bind the source and the sink to the channel.
- Can have multiple agents in Flume.
    - Differentiate each agent by using a unique name.
    - Using this name, we have to configure each agent.

- Name/list components such as sources, sinks, and the channels of the agent

```
agent_name.sources = source_name
agent_name.sinks = sink_name
agent_name.channels = channel_name
```

```
TwitterAgent.sources = Twitter
TwitterAgent.channels = MemChannel
TwitterAgent.sinks = HDFS
```

Transferring Twitter data using Twitter source through a memory channel to an HDFS sink, and the agent name id TwitterAgent

Source

- Each source will have a separate list of properties.
- Property named “type” is common to every source, and it is used to specify the type of the source we are using.
- Provide the values of all the **required** properties of a particular source to configure it

```
agent_name.sources. source_name.type = value
agent_name.sources. source_name.property2 = value
agent_name.sources. source_name.property3 = value
```

```
TwitterAgent.sources.Twitter.type = Twitter (type name)
TwitterAgent.sources.Twitter.consumerKey =
TwitterAgent.sources.Twitter.consumerSecret =
TwitterAgent.sources.Twitter.accessToken =
TwitterAgent.sources.Twitter.accessTokenSecret =
```

Sink

- Each sink will have a separate list of properties.
- Property named “type” - specifies the type of the sink
- Provide values to all the **required** properties of a particular sink

```
agent_name.sinks. sink_name.type = value
agent_name.sinks. sink_name.property2 = value
agent_name.sinks. sink_name.property3 = value
```

HDFS sink

TwitterAgent.sinks.HDFS.type = hdfs (type name)

TwitterAgent.sinks.HDFS.hdfs.path = HDFS directory’s Path to store the data

Channels

- Channels to transfer data between sources and sinks.
- Need to describe the channel used in the agent.

- To describe each channel - set the required properties

```
agent_name.channels.channel_name.type = value
agent_name.channels.channel_name. property2 = value
agent_name.channels.channel_name. property3 = value
```

Memory channel

`TwitterAgent.channels.MemChannel.type = memory` (type name)

Binding source and sink to channel

- Since the channels connect the sources and sinks, it is required to bind both of them to the channel

```
agent_name.sources.source_name.channels = channel_name
agent_name.sinks.sink_name.channels = channel_name
```

twitter source, memory channel, HDFS sink.

```
TwitterAgent.sources.Twitter.channels = MemChannel
TwitterAgent.sinks.HDFS.channels = MemChannel
```

A sample configuration file with file extension .conf is shown below. It shows all the keys and keywords to be used to collect the twitter data.

```
TwitterAgent.sources = Twitter
TwitterAgent.channels = MemChannel
TwitterAgent.sinks = HDFS
TwitterAgent.sources.Twitter.type = com.cloudera.flume.source.TwitterSource
TwitterAgent.sources.Twitter.channels = MemChannel
TwitterAgent.sources.Twitter.consumerKey =
TwitterAgent.sources.Twitter.consumerSecret =
TwitterAgent.sources.Twitter.accessToken =
TwitterAgent.sources.Twitter.accessTokenSecret =
TwitterAgent.sources.Twitter.keywords = Keywords to be specified here
TwitterAgent.sinks.HDFS.channel = MemChannel
TwitterAgent.sinks.HDFS.type = hdfs
TwitterAgent.sinks.HDFS.hdfs.path =   Configuration File Path to store the data.  # hdfs://hadoop1:9000/rramine/Food_data/
TwitterAgent.sinks.HDFS.hdfs.fileType = DataStream
TwitterAgent.sinks.HDFS.hdfs.writeFormat = Text
TwitterAgent.sinks.HDFS.hdfs.batchSize = 100
TwitterAgent.sinks.HDFS.hdfs.rollSize = 0
TwtterAgent.sinks.HDFS.hdfs.rollCount = 0
TwitterAgent.channels.MemChannel.type = memory
TwitterAgent.channels.MemChannel.capacity = 10000
TwitterAgent.channels.MemChannel.transactionCapacity = 10000
```