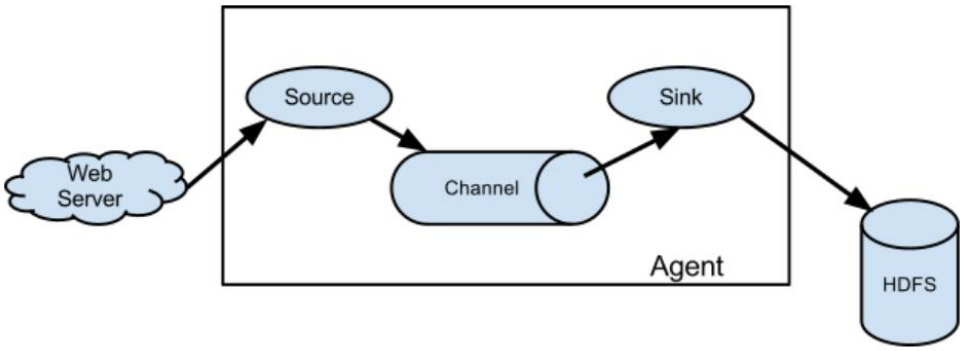
Chapter -2

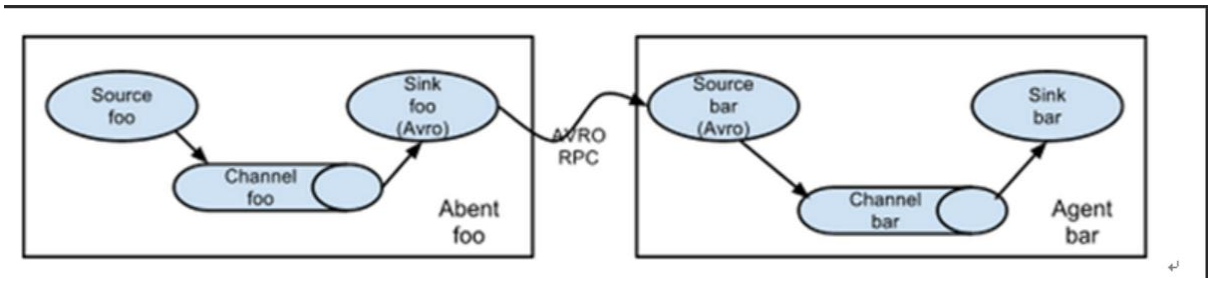
Data Flow: -

- Web server is a data generator. It generates data.
- Source consumes data/events from web server.
- Source transfer data/events to channel.
- Channel receives data from source.
- Channel will store data in memory or file.(The channel is a passive store that keeps the event until it's consumed by a Flume sink)
- The sink removes the event from the channel and puts it into an external repository like HDFS.

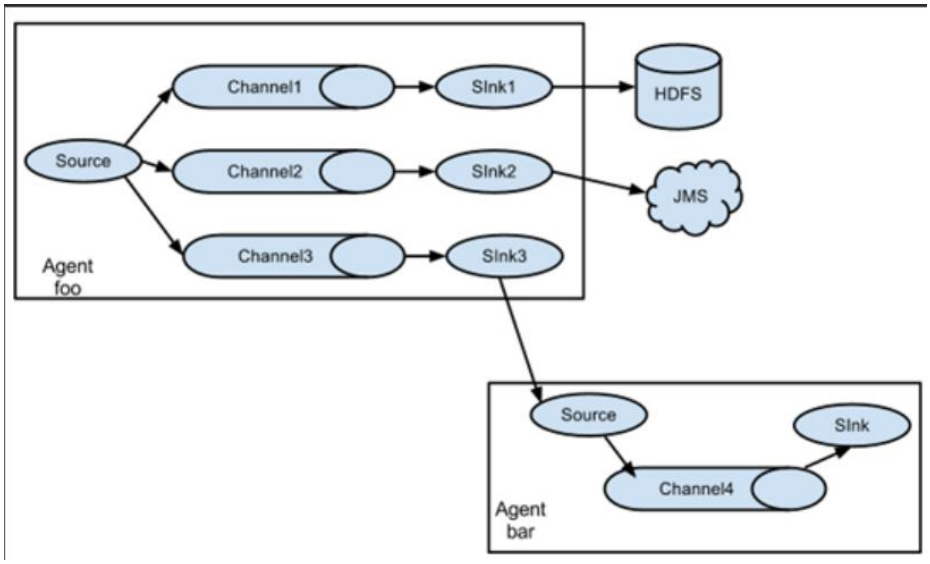


Complex Data Flow

Flume allows a user to build multi-hop flows where events travel through multiple agents before reaching the final destination. It also allows fan-in and fan-out flows, contextual routing and backup routes (fail-over) for failed hops.



Multiplexing the Flow



Flume Supports multiplexing the event flow to one or more destinations.

In this example,

source from agent “foo” fanning out the flow to three different channels. This fan out can be replicating or multiplexing.

Replicating

In case of replicating flow, each event is sent to all three channels.

Same data/event sends all multiple channels and channels will transfer to sink and then different destination.

Multiplexing

An event is delivered to a subset of available channels when an event’s attribute matches a preconfigured value.

Example:-

- we have a data set.
- In the data set we have country
- In the country we have CN, USA, UK, IND.
- We can set CN data will transfer from channel 1.
- USA will transfer from channel 2.
- UK will transfer from channel 3.
- IND will transfer from all.

Data Flow with the help of Netcat.

A netcat-like source that listens on a given port and turns each line of text into an event.

With the help of netcat we are transferring data. Netcat is data source and flume source will take data and transfer into hdfs or console.

Task: -In the example data will flow with the help of netcat. Flume source connects with netcat and sink will connect with console.

- type – Type should be correct. We cannot write anything on type. Type means we are using netcat source.
- bind, we use same computer ip for listening.
- Without port it will not work.
- logger, information will display on terminal only.

```
#agent Name/source name/channel name/sink name

agent.sources = NetCat
agent.channels = memoryChannel
agent.sinks = loggerSink

# Source Information
agent.sources.NetCat.type = netcat
agent.sources.NetCat.bind = localhost
agent.sources.NetCat.port = 9999


# Sink Information
agent.sinks.loggerSink.type = logger



# Channel information.
Agent.channels.memoryChannel.type = memory

#capacity
agent.channels.memoryChannel.capacity = 1000
agent.channels.memoryChannel.transactionCapacity = 100

# Connect Source with Channel
agent.sources.NetCat.channels = memoryChannel
agent.sinks.loggerSink.channel = memoryChannel
```

Task:- In the example data will flow with the help of netcat. Flume source connects with netcat and sink will connect with HDFS.

```
#agent Name/source name/channel name/sink name
```

```
agent.sources = NetCat
agent.channels = memoryChannel
agent.sinks = HDFS
# Source Information

agent.sources.NetCat.type = netcat
agent.sources.NetCat.bind = localhost
agent.sources.NetCat.port = 22222


# Sink Information
agent.sinks.HDFS.type = hdfs
agent.sinks.HDFS.hdfs.path = /vipin/data/
agent.sinks.HDFS.hdfs.filePrefix = vip
agent.sinks.HDFS.hdfs.rollInterval = 30
agent.sinks.HDFS.hdfs.fileType = DataStream
agent.sinks.HDFS.hdfs.hdfs.writeFormat = Text


# Channel information.

Agent.channels.memoryChannel.type = memory
agent.channels.memoryChannel.capacity = 1000


# Connect Source with Channel

agent.sources.NetCat.channels = memoryChannel
agent.sinks.HDFS.channel = memoryChannel
```

Data Flow with the help of Exec.

Exec source runs a given Unix command on start-up and expects that process to continuously produce data on standard out. If the process exits for any reason, the source also exits and will produce no further data.

We have 1 machine.

In this machine we have a web server/web application. On the same machine, we have flume and HDFS.

We have 1 File, web server generating continuous logs. So we will use Linux command to read file and transfer file data into hdfs/console.

Task :- we want to transfer single log file data into console.

- type – we are using exec source.
- command – linux command and file location with name(cat and tail -F)

In this Example we will execute linux command on file and transfer data on console.

```
#agent Name/source name/channel name/sink name

agent.sources = source1
agent.channels = memoryChannel
agent.sinks = loggerSink
```

```
# Source Information
agent.sources.source1.type = exec
agent.sources.source1.command = cat /home/hadoop-SRV1/Desktop/file4.txt

# Sink Information


agent.sinks.loggerSink.type = logger

# Channel information.

agent.channels.memoryChannel.type = memory
agent.channels.memoryChannel.capacity = 1000
```

Task :- In this Example we will execute linux command on file and transfer data on HDFS.

```
#agent Name/source name/channel name/sink name

agent.sources = source1
agent.channels = memoryChannel
agent.sinks = loggerSink

# Source Information

agent.sources.source1.type = exec
agent.sources.source1.command = cat /home/hadoop-SRV1/Desktop/file4.txt
or
agent.sources.source1.command = tail -f /home/hadoop-SRV1/Desktop/file4.txt


# Sink Information

agent.sinks.HDFS.type = hdfs
agent.sinks.HDFS.hdfs.path = /vipin/data/
agent.sinks.HDFS.hdfs.filePrefix = vip
agent.sinks.HDFS.hdfs.rollInterval = 30
agent.sinks.HDFS.hdfs.fileType = DataStream
agent.sinks.HDFS.hdfs.hdfs.writeFormat = Text

# Channel information.

agent.channels.memoryChannel.type = memory
agent.channels.memoryChannel.capacity = 1000
agent.channels.memoryChannel.transactionCapacity = 1000


# Connect Source with Channel

agent.sources.source1.channels = memoryChannel
agent.sinks.loggerSink.channel = memoryChannel
```

Task :- Use tail –F command to transfer data countinously.
If new record comes in the file, tail –F command will read and send to source.

```
agent.sources.source1.type = exec
agent.sources.source1.command = tail -F /home/hadoop-SRV1/Desktop/file4.txt
agent.sources.source1.bind = localhost
```

```
agent.sources.source1.port = 22222


agent.channels.memoryChannel.type = memory
agent.channels.memoryChannel.capacity = 1000



agent.sinks.HDFS.type = hdfs
agent.sinks.HDFS.hdfs.path = /vipin/data/
agent.sinks.HDFS.hdfs.filePrefix = vs
agent.sinks.HDFS.hdfs.rollInterval = 30
agent.sinks.HDFS.hdfs.fileType = DataStream

agent.sources.source1.channels = memoryChannel
agent.sinks.HDFS.channel = memoryChannel
```

Data Flow with the help of SpoolDir.

This source lets you ingest data by placing files to be ingested into a “spooling” directory on disk.

This source will watch the specified directory for new files, and will parse events out of new files as they appear. The event parsing logic is pluggable. After a given file has been fully read into the channel, completion by default is indicated by renaming the file or it can be deleted or the trackerDir is used to keep track of processed files.

We have 1 machine.

In this machine we have a web server/web application. On the same machine we have flume and hdfs.

Task :- We have 1 directory and in this directory, web server generating multiple log file in 1 directory. So spooldir will transfer directory files in hbase.

- **First, we start hdfs and hbase server(start-hbase.sh). then open hbase shell (hbase shell).**
- **Create table in hbase (create ‘tablename’,’columnfamily’)**
- **List to check table is create or not.**
- **Scan to check data in table (scan ‘tablename’)**

(every file will transfer from the directory. If flume marked .COMPLETED, no more data will transfer from the file.)

- spoolDir = location of directory (every file will transfer from the directory. If flume marked .COMPLETED, no more data will transfer from the file.)

```
agent.sources = source1
```

```
agent.channels = memoryChannel
agent.sinks = HDFS

agent.sources.source1.type = spooldir
agent.sources.source1.spoolDir = /home/hadoop-SRV1/datafiles

agent.sinks.HDFS.type = hbase2
agent.sinks.HDFS.table = flume
agent.sinks.HDFS.columnFamily=data
agent.sinks.HDFS.serializer=org.apache.flume.sink.hbase2.RegexHBase2EventSerializer

agent.channels.memoryChannel.type = memory
agent.channels.memoryChannel.capacity = 1000

agent.sources.source1.channels = memoryChannel
agent.sinks.HDFS.channel = memoryChannel
```

Data Flow with the help of Taildir.

Watch the specified files, and tail them in nearly real-time once detected new lines appended to the each files. If the new lines are being written, this source will retry reading them in wait for the completion of the write.

This source is reliable and will not miss data even when the tailing files rotate. It periodically writes the last read position of each files on the given position file in JSON format. If Flume is stopped or down for some reason, it can restart tailing from the position written on the existing position file.

Task :- we have multiple files and data comes real-time in these files(data coming every sec/min/hour).

We want if new records comes in these file, data will transfer automatically in the hdfs.

(flume will run always.)

- filegroups = number of files(how many files we want to target)
- filegroups.f1 = location of file 1.
- filegroups.f2 = location of file 2.
- positionFile = Json file location

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

agent1.sources.source1.type = TAILDIR
agent1.sources.source1.filegroups = f1 f2
agent1.sources.source1.filegroups.f1 = /home/hadoop-SRV1/datafiles/log.txt
agent1.sources.source1.filegroups.f2 = /home/hadoop-SRV1/datafiles/.*log.*
agent1.sources.source1.positionFile = /home/hadoop-SRV1/jsonfile/

agent1.channels.channel1.type = memory
```

```
agent1.sinks.sink1.type = logger

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

HTTP SOURCE

A source which accepts Flume Events by HTTP POST and GET. HTTP requests are converted into flume events by a pluggable “handler” which must implement the HTTPSourceHandler interface. This handler takes a HttpServletRequest and returns a list of flume events.

Task:- we will transfer http request in hdfs.

In windows, we will use postman software to generate http request.

In linux, we will run flume and http flume source will convert request into events.

- bind = self ip
- port = port number

```
agent.sources = source1

agent.channels = memoryChannel

agent.sinks = HDFS


agent.sources.source1.type = http

agent.sources.source1.port = 22222

agent.sources.source1.bind = 192.168.0.4


agent.sinks.HDFS.type = hdfs

agent.sinks.HDFS.hdfs.path = /vipin/data/

agent.sinks.HDFS.hdfs.filePrefix = httpdata

agent.sinks.HDFS.hdfs.rollInterval = 30

agent.sinks.HDFS.hdfs.fileType = DataStream


agent.channels.memoryChannel.type = memory

agent.channels.memoryChannel.capacity = 1000


agent.sources.source1.channels = memoryChannel

agent.sinks.HDFS.channel = memoryChannel
```

KAFKA SOURCE

Kafka Source is an Apache Kafka consumer that reads messages from Kafka topics. You can configure them with the same Consumer Group so each will read a unique set of partitions for the topics.

Task:- Transfer Kafka messages into hdfs.

We will start Kafka and produce some messages.

Then we will start flume with Kafka source and transfer Kafka messages in hdfs.

Start Kafka, kafka-server-start.sh config/server.properties

Create topics, kafka-topics.sh --bootstrap-server localhost:9092 --create --topic myTopic --partitions 1 --replication-factor 1

Produce message, kafka-console-producer.sh --bootstrap-server localhost:9092 --topic myTopic

Check consumer group, kafka-consumer-groups.sh --bootstrap-server localhost:9092 --list

- Source1.type = source name
- Kafka.bootstrap.server = kafka server ip with portnumber
- kafka.topics = topic name(topic hold messages)
- consumer.group.id = every consumer is a part of consumer group
- batchSize = Maximum number of messages written to Channel in one batch

```
agent.sources = source1

agent.channels = memoryChannel

agent.sinks = loggerSink


agent.sources.source1.type = org.apache.flume.source.kafka.KafkaSource

agent.sources.source1.kafka.bootstrap.servers = localhost:9092

agent.sources.source1.kafka.topics = myTopic

agent.sources.source1.kafka.consumer.group.id = console-consumer-57888

agent.sources.source1.batchSize = 2500


agent.sinks.loggerSink.type = hdfs

agent.sinks.loggerSink.hdfs.path = /vipin/data/
```

```
agent.sinks.loggerSink.hdfs.filePrefix = kafkamessage

agent.sinks.loggerSink.hdfs.rollInterval = 30

agent.sinks.loggerSink.hdfs.rollCount = 0

agent.sinks.loggerSink.hdfs.rollSize = 0

agent.sinks.loggerSink.hdfs.fileType = DataStream


agent.channels.memoryChannel.type = memory

agent.channels.memoryChannel.capacity = 2500

agent.channels.memoryChannel.transactionCapacity = 2500


agent.sources.source1.channels = memoryChannel

agent.sinks.loggerSink.channel = memoryChannel
```

Multiplexing the Flow

Flume supports multiplexing the event flow to one or more destinations. This is achieved by defining a flow multiplexer that can replicate or selectively route an event to one or more channels.

Task:- we have some data and we want to transfer same data in multiple location(hdfs,hbase).

So we have to setup 2 channel and 2 sinks.

1st channel for sink1(hdfs) and 2nd for sink2(hbase).

REPLICATION

```
agent.sources = source1

agent.channels = channel1 channel2

agent.sinks = HDFS sink2


#define sources


agent.sources.source1.type = spooldir

agent.sources.source1.spoolDir = /home/hadoop-SRV1/datafiles

agent.sources.source1.batchSize = 500

agent.sources.source1.selector.type = replicating
```

```
#define channel1

agent.channels.channel1.type = memory

agent.channels.channel1.capacity = 10000

agent.channels.channel1.transactionCapacity = 10000

#define channel2

agent.channels.channel2.type = memory

agent.channels.channel2.capacity = 10000

agent.channels.channel2.transactionCapacity = 10000

# define sink 1

agent.sinks.HDFS.type = hdfs

agent.sinks.HDFS.hdfs.path = /vipin/data/datacheck/

agent.sinks.HDFS.hdfs.filePrefix = dataspool

agent.sinks.HDFS.hdfs.fileType = DataStream

agent.sinks.HDFS.hdfs.hdfs.writeFormat = Text

agent.sinks.HDFS.hdfs.rollInterval = 40

agent.sinks.HDFS.hdfs.rollCount = 0

agent.sinks.HDFS.hdfs.rollSize = 0

#define sink2

agent.sinks.sink2.type=hbase2

agent.sinks.sink2.table = vipin2

agent.sinks.sink2.columnFamily = data

#connect source with channels
```

```
agent.sources.source1.channels = channel1 channel2


#connect sink with channel
agent.sinks.HDFS.channel = channel1
```

MULTIAGENT with avro

In order to flow the data across multiple agents or hops, the sink of the previous agent and source of the current hop need to be avro type with the sink pointing to the hostname (or IP address) and port of the source.

Avro source -Flume events sent to this sink are turned into Avro events and sent to the configured hostname / port pair.

Avro sink -When paired with the built-in Avro Sink on another (previous hop) Flume agent, it can create tiered collection topologies.

Task:- we have 2 computers.

In computer1, we have data and flume but don't have hdfs.

Computer 2, we have flume and hdfs.

We want to transfer data from computer 1 to computer 2 hdfs.

So we have to run 2 flume agent. 1st for computer 1 and 2nd for computer 2.

Computer 1 configuration

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


#define source


agent1.sources.source1.type = spooldir
agent1.sources.source1.spoolDir = D:/spooldir/


#define channels


agent1.channels.channel1.type = memory
```

agent1.channels.channel1.capacity = 1000

define sink

agent1.sinks.sink1.type = avro

agent1.sinks.sink1.hostname = 192.168.56.107

agent1.sinks.sink1.port = 22222

#connect source with channels and sink with channels

agent1.sources.source1.channels = channel1

agent1.sinks.sink1.channel = channel1

Computer 2 configuration

agent2.sources = source2

agent2.channels = channel2

agent2.sinks = sink2

agent2.sources.source2.type = avro

agent2.sources.source2.bind = 192.168.56.107

agent2.sources.source2.port = 22222

agent2.sinks.sink2.type = hdfs

agent2. sinks.sink2.type = hdfs

agent2. sinks.sink2.hdfs.path = /vipin/data/

agent2. sinks.sink2.hdfs.filePrefix = kafkamessage

agent2. sinks.sink2.hdfs.rollInterval = 30

agent2. sinks.sink2.hdfs.rollCount = 0

agent2. sinks.sink2.hdfs.rollSize = 0

agent2. sinks.sink2.hdfs.fileType = DataStream

```
agent2.channels.channel2.type = memory

agent2.channels.channel2.capacity = 1000


agent2.sources.source2.channels = channel2

agent2.sinks.sink2.channel=channel2
```

HDFS SINK

Sink writes events into the Hadoop Distributed File System (HDFS). It currently supports creating text and sequence files. It supports compression in both file types.

- loggerSink.type = hdfs (want to transfer data into hdfs).
- hdfs.path = location of hdfs(where I want to save data in hdfs).
- hdfs.filePrefix= starting name of file
- hdfs.rollInterval = Maximum waiting time for data(30 sec default).
- hdfs.rollICount = Maximum lines in files(10 lines default).
- hdfs.rollISize = size of file (1024 byte default).
- hdfs.writeFormat = Set to Text before creating data files with Flume, otherwise those files cannot be read by either Apache Impala (incubating) or Apache Hive.
- hdfs.fileType = File format
  currently SequenceFile, DataStream or CompressedStream (1)DataStream will not compress output file and please don't set codeC (2)CompressedStream requires set hdfs.codeC with an available codec

```
agent.sinks.loggerSink.type = hdfs

agent.sinks.loggerSink.hdfs.path = /vipin/data/

agent.sinks.loggerSink.hdfs.filePrefix = kafkamessage

agent.sinks.loggerSink.hdfs.rollInterval = 30

agent.sinks.loggerSink.hdfs.rollCount = 0

agent.sinks.loggerSink.hdfs.rollSize = 0

agent.sinks.loggerSink.hdfs.fileType = DataStream
```

HBASE SINK

This sink writes data to HBase. A class implementing HbaseEventSerializer which is specified by the configuration is used to convert the events into HBase puts and/or increments. These puts and increments are then written to HBase.

- table = hbase table name.
- columnFamily = columnfamily name.

```
agent.sinks.sink1.type = hbase2

agent.sinks.sink1.table = flume

agent.sinks.sink1.columnFamily=data

agent.sinks.sink1.serializer=org.apache.flume.sink.hbase2.RegexHBase2EventSerializer


agent.sinks.sink2.channel = channel2
```

Channels

Channels are the repositories where the events are staged on a agent. Source adds the events and Sink removes it.

Memory Channel

The events are stored in an in-memory queue with configurable max size.

- Capacity = The maximum number of events stored in the channel
- transactionCapacity = The maximum number of events the channel will take from a source or give to a sink per transaction.

```
agent2.channels.channel2.type = memory

agent2.channels.channel2.capacity = 1000

agent2.channels.channel2.transactionCapacity = 1000
```

File Channel

File channel writes out all the flume events to the disk. It does not lose data even when the process or machine shuts down or crashes. This channel ensures that any events which are committed into the channel are removed from it only when a sink consumes the events and commits the transaction.

- Checkpointdir = directory where checkpoint file will be stored.
- dataDirs = directories for storing log files. Using multiple directories on separate disks can improve file channel performance.

```
agent2.channels.channel2.type = file

agent2.channels.channel2. checkpointDir= /home/hadoop-SRV1/filechannel/checkdir

agent2.channels.channel2. dataDirs = /home/hadoop-SRV1/filechannel/datadir
```