**Apache Flume – Ingesting log data to HDFS**

As part of this session we will understand how we can use Apache Flume to ingest streaming real time data in detail.

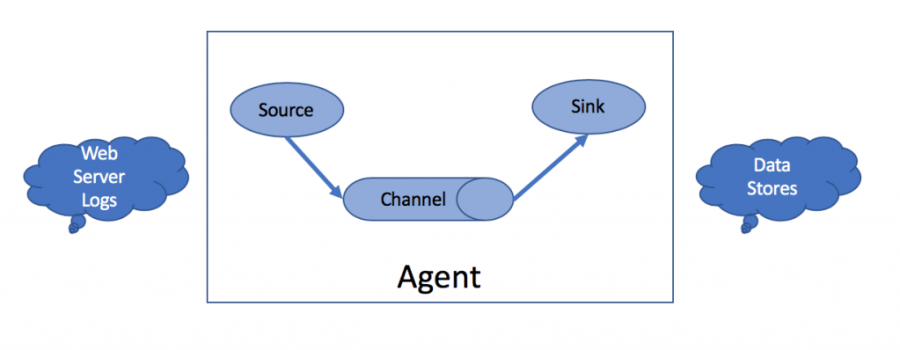
* Overview of Flume
* Setting up gen\_logs
* Develop first Flume Agent
* Understand Source, Sink and Channel
* Flume Multi Agent Flows
* Get data into HDFS using Flume
* Limitations and Conclusion

For this demo we will be using our Big Data developer labs. You need to have access to existing big data cluster or sign up to our labs.

**Overview of Flume**

Let us understand what Flume is all about.

* Flume is a distributed, reliable, and available service for efficiently collecting, aggregating, and moving large amounts of log data.
* It has a simple and flexible architecture based on streaming data flows.
* It is robust and fault tolerant with tunable reliability mechanisms and many failover and recovery mechanisms.
* It can be integrated with other technologies like Spark Streaming to build streaming analytic applications.
* We need to configure agent to get data from source like web server logs and target like data stores such as HDFS.
* Flume agent contains – Source, Sink and Channel
* Here is the [link](https://flume.apache.org/FlumeUserGuide.html) for latest flume documentation.



**Develop first Flume Agent**

Now let us go ahead and develop our first Flume agent.

* [Here](https://flume.apache.org/FlumeUserGuide.html#a-simple-example) is the complete configuration file for simple flume example.
* Agent Name: a1
* Source
  + Name: r1
  + Type: Netcat
  + Netcat is simple web server which runs on simple IP address and port number.
  + IP address is either gw02.test.com or gw03.test.com on which you want to run netcat service
  + Port Number: 44444
  + A web server will be started by flume agent automatically
  + This is primarily used to understand flume. We do not use this in actual implementations.
* Sink
  + Name: k1
  + Type: logger
  + Data will be sent back to the flume agent
* Channel
  + Name: c1
  + Type: memory
* Now we can start flume agent using flume-ng command.

|  |  |
| --- | --- |
|  | # Create a file by name example.conf |
|  | a1.sources = r1 |
|  | a1.sinks = k1 |
|  | a1.channels = c1 |
|  |  |
|  | a1.sources.r1.type = netcat |
|  | a1.sources.r1.bind = 0.0.0.0 |
|  | a1.sources.r1.port = 44444 |
|  |  |
|  | a1.sinks.k1.type = logger |
|  |  |
|  | 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 |

|  |  |
| --- | --- |
|  | flume-ng agent \ |
|  | --conf-file example.conf \ |
|  | --name a1 |

**Understand Source, Sink and Channel**

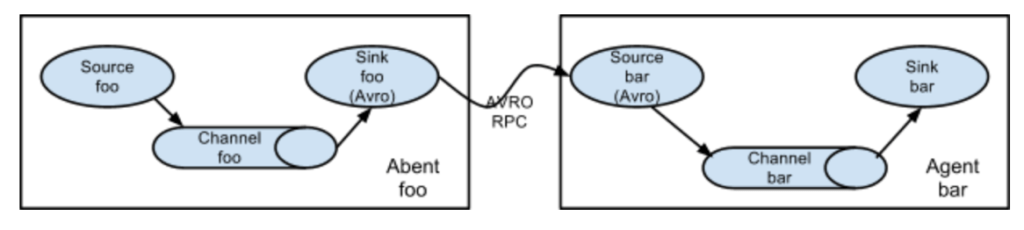
Each Flume agent have Source, Sink and Channel.

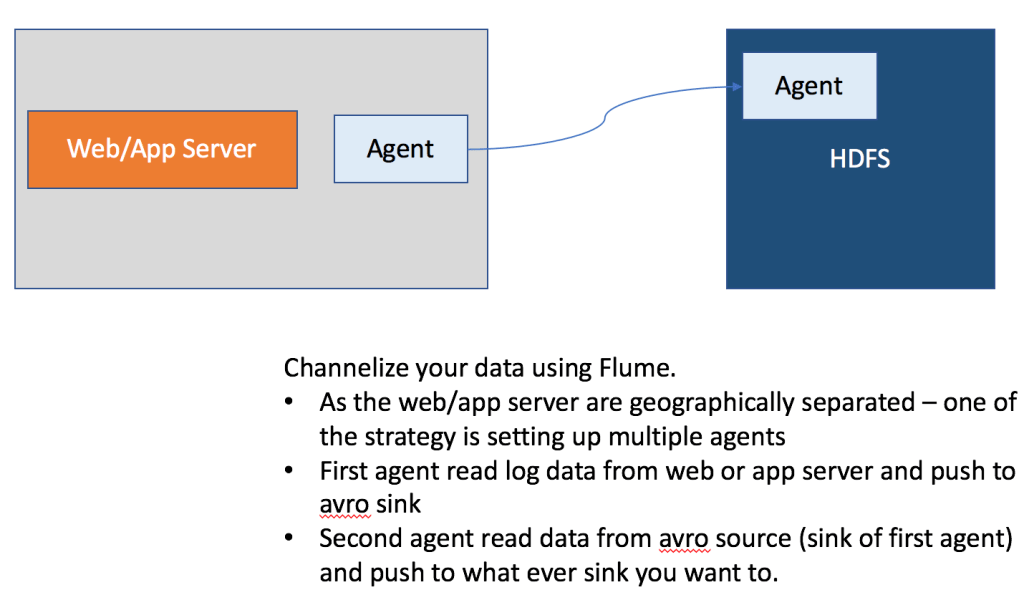
* Source is primarily to read data from web server logs. There are several types of sources.
  + netcat
  + exec
  + syslog
  + avro
  + and more.
* Sink is primarily to write data into data stores or other Flume agent sources (via avro). There are several types of sinks.
  + logger
  + HDFS
  + avro
  + and many more
* Channel is to channelize data between source and sink. There are different types of channels but most popular ones are memory, file and Kafka.
* Memory gives you good performance, but not reliable. File gives you reliability at the cost of performance.
* There can be one to many relationship between source and sink. But for each sink there need to be one channel.
* Properties are determined by the type chosen.
* It is easily extensible. For example we can integrate custom sources, sinks or channels with Flume.

**Flume Multi Agent Flows**

Flume can be configured with multiple agents for scalability, consolidation etc.

***Simple Multi Agent Flow***





* We have to define multiple agents and connect them to get data from source to target in complex environments
* If output of one agent have to be passed as input to other then we need to use avro sink and avro source
* Develop First agent on the host from where we want to ingest data into HDFS. For now we are using logger as sink.
  + Source: avro
  + Channel: memory
  + Sink: logger
* Develop Second agent on the host where we want to read data from web server logs
  + Source: log data from web/app server logs
  + Channel: memory
  + Sink: avro
* First start the agent where we have avro source. It will start the associated web server on ip address and port number.
* Start the second agent wherewe have avroas sink

|  |  |
| --- | --- |
|  | # File Name: first-agent.conf |
|  |  |
|  | fa.sources = r1 |
|  | fa.sinks = k1 |
|  | fa.channels = c1 |
|  |  |
|  | fa.sources.r1.type = avro |
|  | fa.sources.r1.bind = gw02.test.com |
|  | fa.sources.r1.port = 44444 |
|  |  |
|  | fa.sinks.k1.type = logger |
|  |  |
|  | fa.channels.c1.type = memory |
|  | fa.channels.c1.capacity = 1000 |
|  | fa.channels.c1.transactionCapacity = 100 |
|  |  |
|  | fa.sources.r1.channels = c1 |
|  | fa.sinks.k1.channel = c1 |

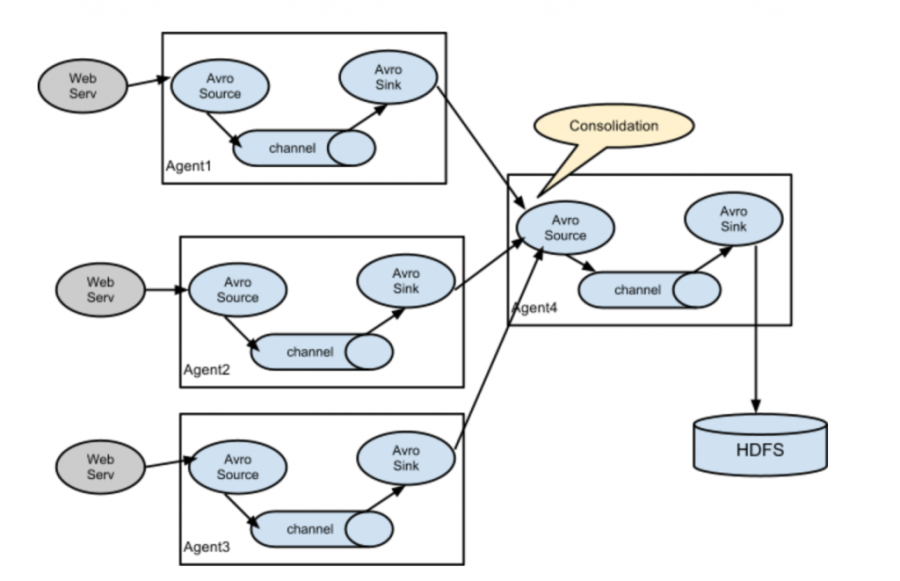
|  |  |
| --- | --- |
|  | # File Name: second-agent.conf |
|  |  |
|  | sa.sources = r1 |
|  | sa.sinks = k1 |
|  | sa.channels = c1 |
|  |  |
|  | sa.sources.r1.type = exec |
|  | sa.sources.r1.command = tail -F /opt/gen\_logs/logs/access.log |
|  |  |
|  | sa.sinks.k1.type = avro |
|  | sa.sinks.k1.hostname = gw02.test.com |
|  | sa.sinks.k1.port = 44444 |
|  |  |
|  | sa.channels.c1.type = memory |
|  | sa.channels.c1.capacity = 1000 |
|  | sa.channels.c1.transactionCapacity = 100 |
|  |  |
|  | sa.sources.r1.channels = c1 |
|  | sa.sinks.k1.channel = c1 |

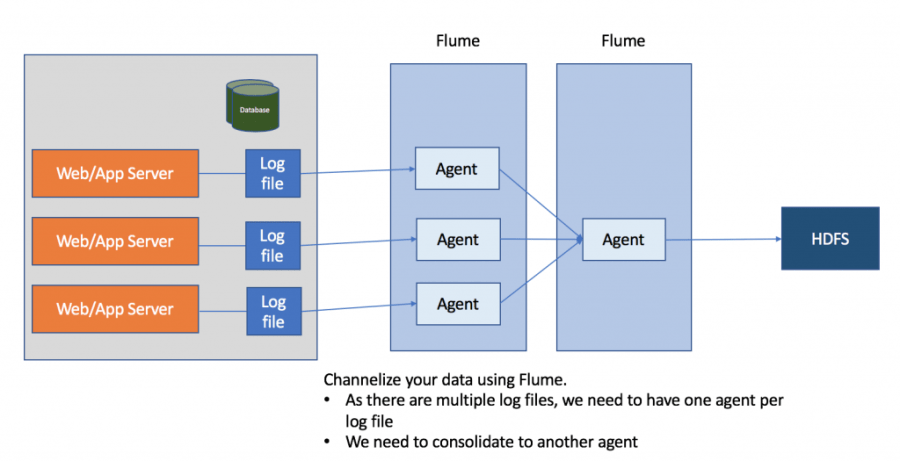
|  |  |
| --- | --- |
|  | flume-ng agent \ |
|  | --name fa |
|  | --conf-file first-agent.conf |

|  |  |
| --- | --- |
|  | flume-ng agent \ |
|  | --name sa |
|  | --conf-file second-agent.conf |

***Consolidation***

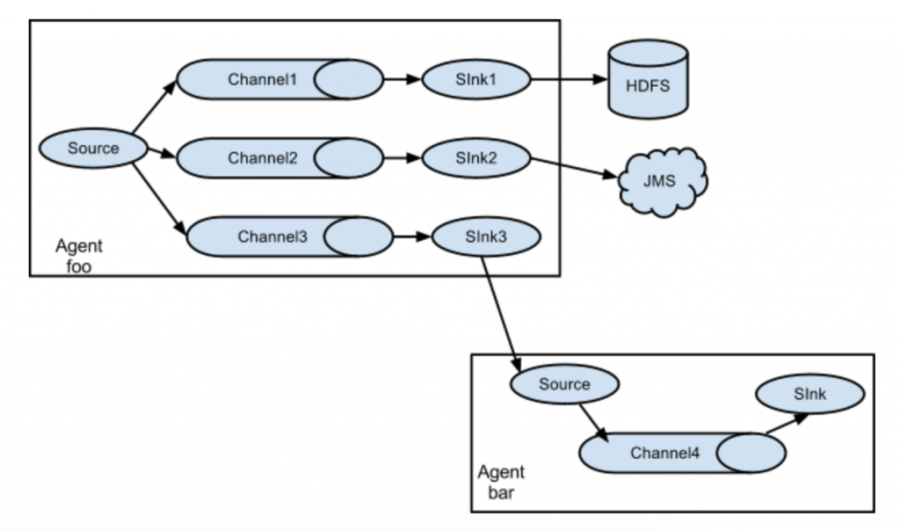
* Typically to read data related to same application which is deployed on multiple web/app servers
* We need to define one agent for each web/app server and each of those need to write to single avro sink
* Then we can have one agent to read data by configuring avro source and then to what ever target you want to write to.

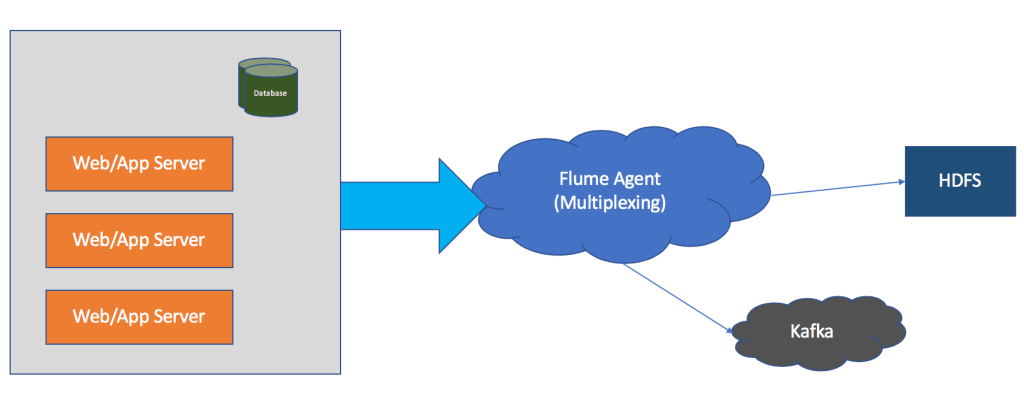




***Multiplexing and Replicating***

* Typically reading is faster than writing, which means in a given Flume Agent source will be able to read data much faster than sink processing those.
* If we have too many messages read by source, sink might not be able to process all the data.
* We can use multiplexing and have multiple channels and sinks to distribute the writes.
* Also we can push data to multiple targets using replicating – which means data captured by source will be passed on to sinks as multiple copies.
* All sources have properties such as selectors and interceptors.
* Replicating is very straight forward to implement. Choose selector as replicating (default) and define one channel and one sink for each copy of data to be processed.
* Implementing multiplexing is a bit complicated
  + You need to set selector as multiplexing
  + Define interceptors to apply rules for multiplexing as data is read by the source





**Get data into HDFS using Flume**

Now let us understand how to get data into HDFS using Flume.

* When we get data from web logs into HDFS, we need to consolidate multiple messages before writing into files.

***Source – Determine appropriate source***

* First we need to understand whether to get data locally where logs are generated or over the network
* exec can be used to get data from local logs in asynchronous fashion. But it have limitations, there can be data loss.
* There are other reliable options such as Spooling Directory. But for this demonstration we will use exec to keep it simple.

|  |  |
| --- | --- |
|  | lm.sources = r1 |
|  | lm.sinks = k1 |
|  | lm.channels = c1 |
|  |  |
|  | lm.sources.r1.type = exec |
|  | lm.sources.r1.command = tail -F /opt/gen\_logs/logs/access.log |
|  |  |
|  | lm.sinks.k1.type = logger |
|  |  |
|  | lm.channels.c1.type = memory |
|  | lm.channels.c1.capacity = 1000 |
|  | lm.channels.c1.transactionCapacity = 100 |
|  |  |
|  | lm.sources.r1.channels = c1 |
|  | lm.sinks.k1.channel = c1 |

***Sink – HDFS***

* There are several important properties we need to be aware of.
* rollCount, rollIntervaland rollSize – to consolidate messages based on either count or interval or size.
* Specifying prefix and suffix to the files which will have messages
* Using timestamp so that we can manage files into directories based on date or month or year.
* Channel Type – Memory
* We will explore all the options from official documentation and then use it as part of agent definition.

|  |  |
| --- | --- |
|  | lm.sources = r1 |
|  | lm.sinks = k1 |
|  | lm.channels = c1 |
|  |  |
|  | lm.sources.r1.type = exec |
|  | lm.sources.r1.command = tail -F /opt/gen\_logs/logs/access.log |
|  |  |
|  | lm.sinks.k1.type = hdfs |
|  | lm.sinks.k1.hdfs.path = hdfs://nn01.test.com:8020/user/test/logstohdfs |
|  | lm.sinks.k1.hdfs.filePrefix = retail |
|  | lm.sinks.k1.hdfs.fileSuffix = .txt |
|  | lm.sinks.k1.hdfs.rollInterval = 60 |
|  | lm.sinks.k1.hdfs.rollSize = 0 |
|  | lm.sinks.k1.hdfs.rollCount = 100 |
|  | lm.sinks.k1.hdfs.fileType = DataStream |
|  |  |
|  | lm.channels.c1.type = memory |
|  | lm.channels.c1.capacity = 1000 |
|  | lm.channels.c1.transactionCapacity = 100 |
|  |  |
|  | lm.sources.r1.channels = c1 |
|  | lm.sinks.k1.channel = c1 |

**Get data into Kafka topic using Flume**

Let us see live example of multiplexing with replicating by adding another sink of type Kafka.

* Source – Same source will be multiplexing into another channel and sink

***Sink – Kafka***

* Get get details about Kafka broker and topic
* Add relevant properties while defining the sink
* We will validate by using second channel of type memory

|  |  |
| --- | --- |
|  | lm.sources = r1 |
|  | lm.sinks = k1 k2 |
|  | lm.channels = c1 c2 |
|  |  |
|  | lm.sources.r1.type = exec |
|  | lm.sources.r1.command = tail -F /opt/gen\_logs/logs/access.log |
|  |  |
|  | lm.sinks.k1.type = hdfs |
|  | lm.sinks.k1.hdfs.path = hdfs://nn01.test.com:8020/user/test/logstohdfs\_%Y-%m-%d |
|  | lm.sinks.k1.hdfs.filePrefix = retail |
|  | lm.sinks.k1.hdfs.fileSuffix = .txt |
|  | lm.sinks.k1.hdfs.rollInterval = 60 |
|  | lm.sinks.k1.hdfs.rollSize = 0 |
|  | lm.sinks.k1.hdfs.rollCount = 100 |
|  | lm.sinks.k1.hdfs.fileType = DataStream |
|  | lm.sinks.k1.hdfs.useLocalTimeStamp = true |
|  |  |
|  | lm.sinks.k2.type = org.apache.flume.sink.kafka.KafkaSink |
|  | lm.sinks.k2.kafka.bootstrap.servers = wn01.test.com:6667,wn02.test.com:6667 |
|  | lm.sinks.k2.topic = logstokafka |
|  |  |
|  | lm.channels.c1.type = memory |
|  | lm.channels.c1.capacity = 1000 |
|  | lm.channels.c1.transactionCapacity = 100 |
|  |  |
|  | lm.channels.c2.type = memory |
|  | lm.channels.c2.capacity = 1000 |
|  | lm.channels.c2.transactionCapacity = 100 |
|  |  |
|  | lm.sources.r1.channels = c1 c2 |
|  | lm.sinks.k1.channel = c1 |
|  | lm.sinks.k2.channel = c2 |

***Channel Type – File***

* Once we get the data into Kafka broker using memory channel, we will explore how we can change it to file while understanding the properties related to file.
* We will explore all the options from official documentation and then use it as part of agent definition.

**Limitations and Conclusion**

* Flume is primarily data ingestion tool.
* We can apply row level transformations as well as filter junk data using interceptors
* If we want to perform aggregations on inflight data, we need to integrate flume output with technologies like Spark Streaming, Storm, Flinketc.
* Use Cases
  + Get raw data into HDFS for compliance or batch processing
  + Perform streaming analytics using Flume and Spark Streaming or Storm or Flink
  + Ingest data from legacy application’s web server logs into Kafka topic
* Limitations
  + Manageability – Over a period of time there can be many flume agents and there is no tools to manage them (unlike Kafka)
  + Flume is not designed for IoT applications, IoT is one of the prominent use case with respect to streaming analytics
  + Scalability – Scaling up is not straight forward, you need to use interceptors and they are a bit complicated
  + Reliability – Reliability might be possible but it is complicated
  + Extensability- You can develop plugins for Flume but not very straight forward.
* Overall Kafka eco system is much more modern, scalable, reliable, light weight than Flume. Kafka have modules such as connect, kstreams, ksqletc around the core component Kafka topic which can help you to build streaming pipelines and data integration frameworks for modern applications.