HDFS Sink configuration:



If you want sink as **binary** file -🡪 hdfs.fileType is **sequenceFile**

If you want sink to be written as **text** file -🡪 hdfs.fileType is **dataStream**

If you want sink as **compressed** file -🡪 hdfs.fileType is **compressedStream**

hdfs.fileType = DataStream denotes that in HDFS sink, output files should be created with text File format. (DataStream means “copy data to the destination exactly same as in source)

Also we have specified avro\_event  serializer with snappy compression codec to store the output files with avro serialized format and compressing the sequence files with snappy Compression Codec.

Example:

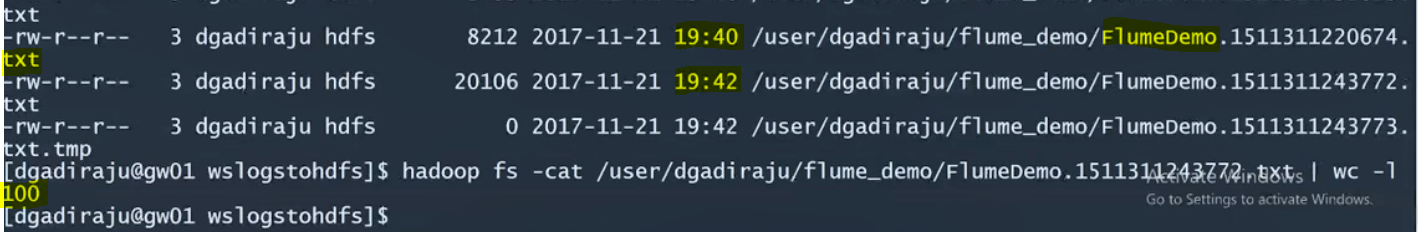


There are three threshold limit of the .tmp file, reaching to that point (any of the three) it stops writing in .tmp file (at this point .tmp converted in to permanent output file with filePrefix and fileSuffix in their name) and starts to write in new .tmp file.

1. rollInterval (in seconds)
2. rollsize (in bytes)
3. rollCount (number of lines)

Run the flume agent and check the output files:





**Memory channel configuration:**

Because memory is transient, if we use memory-channel, if the agent fails, there is a chance that it will lose the data. You have to understand this limitation. To limit the agent failures especially due to the channel, you have to understand these two properties:

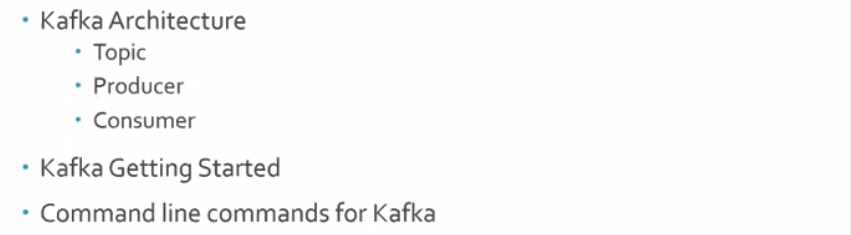
1. Capacity – maximum number of messages stored in channel. If all channel capacity are not consumed by sink. Flume will through error.
2. transactionCapacity

This is the maximum number of event which a channel can take from a source or give to a sink per transaction. In general capacity is higher than transactionCapacity.

It means, by the time transaction capacity is reached max from the source, data should be processed by the channel (same is with sink unless the tC of sink is being recycled by the time it reaches max (capacity))

Typically the data which are read by source will be channeled to sink as early as possible.

**Getting started with Kafka:**



Event Serializers:

Flume supports serialization of events with File\_Roll Sink and HDFS Sink only. But we can write our custom event serializers by implementing EventSerializer interface.

As of now, flume provides two serializers shown below.

**Body Text Serializer:**

By default, events are Text serialized : This serializer writes the body of the event to an output stream without any transformation or modification. The event headers are ignored.

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Default** | **Description** |
| appendNewline | true | Whether a newline will be appended to each event at write time. The default of true assumes that events do not contain newlines, for legacy reasons. |

Example for agent named a1:

a1.sinks.k1.type = file\_roll  
a1.sinks.k1.sink.directory = /var/log/flume  
a1.sinks.k1.sink.serializer = text  
a1.sinks.k1.sink.serializer.appendNewline = false

**Avro Event Serializer:**

avro\_event: This serializer serializes Flume events into an Avro container file. The schema used is the same schema used for Flume events in the Avro RPC mechanism. This serializers inherits from the *AbstractAvroEventSerializer* class.

Configuration options are as follows:

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Default** | **Description** |
| syncIntervalBytes | 2048000 | Avro sync interval, in approximate bytes. |
| compressionCodec | null | Avro compression codec. E.g snappy |

**Ref:**

<http://hadooptutorial.info/flume-data-collection-into-hdfs-avro-serialization/>