**Quick\_Ref:Flume**

|  |  |  |
| --- | --- | --- |
| **S.No** | **Topic** | **Reference** |
|  | **WebReference** |  |
|  | Flume Introduction | <https://www.youtube.com/watch?v=ZOQDdtU456Q&index=19&list=PLf0swTFhTI8rJvGpOp-LujOcpk-Rlz-yE> |
|  | User Guide | <https://flume.apache.org/FlumeUserGuide.html> |
|  | **Cloudera Flume Exercise** | <http://www.cloudera.com/developers/get-started-with-hadoop-tutorial/exercise-4.html> |
|  | Cloudera Flume Install & Configuration | <http://www.cloudera.com/documentation/archive/cdh/4-x/4-3-0/CDH4-Installation-Guide/cdh4ig_topic_12_3.html> |
|  | Stream Twitter data into flume | <https://www.youtube.com/watch?v=YMvNUNclW_E> |
|  | The above twitter demo file | <https://github.com/prasadram/ApacheFlumeDemo/blob/master/twitter.conf> |
|  | Supported Sources, Sinks, and Channels | <http://www.cloudera.com/documentation/archive/cdh/4-x/4-3-0/CDH4-Installation-Guide/cdh4ig_topic_12_9.html> |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  | **General Info** |  |
| 1 | About Flume | Apache Flume is a service for streaming logs into Hadoop  Apache Flume is a distributed, reliable, and available system for efficiently collecting, aggregating and moving large amounts of log data from many different sources into the Hadoop Distributed File System (HDFS)  It has a simple and flexible architecture based on streaming data flows; and is robust and fault tolerant with tunable reliability mechanisms for failover and recovery.  Apache Flume is a top level project at the Apache Software Foundation. |
| 2 | A real time example | **Question:** Consider a scenario You want to understand more about how users browse your public website. you want to know which pages they visit prior to placing an order. You have a server farm of 200 web servers hosting your website. Which is the most efficient process to gather these web servers across logs into your Hadoop cluster analysis?  **Ans:** Ingest the server web logs into HDFS using Flume |
| 3 | Purpose of Flume | # Consider a scenario you want to move data from RDBMS and Some other open systems (ex: weblog) to HDFS   1. RDBMS -> HDFS # We can use Sqoop for this 2. Open system ( WebLog) -> HDFS # Sqoop will not support this, so use Flume, Kafka, storm, etc   # Flume is primarily used to pull the data from open systems via weblog and integrate into HDFS   1. Open system ( WebLog – Place flume agent here) -> flume agent will channel the data collected from open source and push to -> HDFS 2. Agent will be in Open System and HDFS gateway |
| 4 | **Flume Architecture**  Three architectures   1. Multiple agent flow 2. Consolidation flow 3. Multiplexing the flow | # The below box is the flume agent between source and HDFS and there can be multiple flume agent like this… Get detailed ‘three’ different architecture diagram from below link  <https://flume.apache.org/FlumeUserGuide.html> |
| 4.1 | Multiple agent flow | # 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 |
| 4.2 | Consolidation Flow | # A very common scenario in log collection is a large number of log producing clients sending data to a few consumer agents that are attached to the storage subsystem. For example, logs collected from hundreds of web servers sent to a dozen of agents that write to HDFS cluster |
| 4.3 | Multiplexing Flow  The output of one channel will goes to HDFS and other to JMS and other to an another agent | # 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. |
| 5 | System Requirement | 1. Java Runtime Environment - Java 1.6 or later (Java 1.7 Recommended) 2. Memory - Sufficient memory for configurations used by sources, channels or sinks 3. Disk Space - Sufficient disk space for configurations used by channels or sinks 4. Directory Permissions - Read/Write permissions for directories used by agent |
| 6 | Network streams | # Flume supports the following mechanisms to read data from popular log stream types, such as:   1. Avro 2. Thrift 3. Syslog 4. Netcat |
| 7 | Morphlines | Morphlines is the tool provided by cloudera to generate test data simulating the weblog |
| **8** | **Supported Sources** | **Supported Sources, Sinks, and Channels** |
| 8.1 | Avro | # Avro Netty RPC event source. Listens on Avro port and receives events from external Avro streams.  Implementation Class: AvroSource |
| 8.2 | Netcat | # Netcat style TCP event source. Listens on a given port and turns each line of text into an event.  Implementation Class: NetcatSource |
| 8.3 | Seq | # Monotonically incrementing sequence generator event source  Implementation class: SequenceGeneratorSource |
| 8.4 | Exec | # Execute a long-lived Unix process and read from stdout.  Implementation class: ExecSource |
| 8.5 | http | # Accepts Flume events by HTTP POST and GET. GET should be used for experimentation only.  Implementation Class: HTTPSource |
| 8.6 | org.apache.flume.source.jms.JMSSource | # Reads messages from a JMS destination such as a queue or topic.  Implementatoin Class: JMSSource |
| 8.7 | Other (custom) | You need to specify the fully-qualified name of the custom source, and provide that class (and its dependent code) in Flume's classpath. You can do this by creating a JAR file to hold the custom code, and placing the JAR in Flume's lib directory. |
| **9** | **Supported Sinks** | **Supported Sources, Sinks, and Channels** |
| 9.1 | Null | /dev/null for Flume - blackhole all events received  Implementation Class: NullSink |
| 9.2 | Logger | Log events at INFO level via configured logging subsystem (log4j by default)  Implementation Class: LoggerSink |
| 9.3 | Avro | Sink that invokes a pre-defined Avro protocol method for all events it receives (when paired with an avro source, forms tiered collection)  Implementation Class: AvroSink |
| 9.4 | Hdfs | Writes all events received to HDFS (with support for rolling, bucketing, HDFS-200 append, and more)  Implementation Class: HDFSEventSink |
| 9.5 | file\_roll | Writes all events received to one or more files.  Implemnetation Classs: RollingFileSink |
| 9.6 | Other (custom) | You need to specify the fully-qualified name of the custom sink, and provide that class (and its dependent code) in Flume's classpath. You can do this by creating a JAR file to hold the custom code, and placing the JAR in Flume's lib directory. |
| **10** | **Supported Channels** | **Supported Sources, Sinks, and Channels** |
| 10.1 | Memory | In-memory, fast, non-durable event transport  Implementation Class: MemoryChannel |
| 10.2 | Jdbc | JDBC-based, durable event transport (Derby-based)  Implementation Class: JDBCChannel |
| 10.3 | File | File-based, durable event transport  Implementation Class: FileChannel |
| 10.4 | Other (Custom) | You need to specify the fully-qualified name of the custom channel, and provide that class (and its dependent code) in Flume's classpath. You can do this by creating a JAR file to hold the custom code, and placing the JAR in Flume's lib directory. |
| 11 | Sqoop vs Flume vs Spark  *<An example for Data Ingest>* | The skills to transfer data between external systems and your cluster. This includes the following:   1. Import data from a MySQL database into HDFS using Sqoop 2. Export data to a MySQL database from HDFS using Sqoop 3. Change the delimiter and file format of data during import using Sqoop 4. Ingest real-time and near-real time (NRT) streaming data into HDFS using Flume 5. Load data into and out of HDFS using the Hadoop File System (FS) commands |
| 12 | Sqoop vs Flume vs Spark  *<An example for Transform, Stage, Store>* | Convert a set of data values in a given format stored in HDFS into new data values and/or a new data format and write them into HDFS. This includes writing Spark applications in both Scala and Python (see note above on exam question format for more information on using either Scale or Python):   1. Load data from HDFS and store results back to HDFS using Spark 2. Join disparate datasets together using Spark 3. Calculate aggregate statistics (e.g., average or sum) using Spark 4. Filter data into a smaller dataset using Spark 5. Write a query that produces ranked or sorted data using Spark |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  | **Flume Admin Commands** |  |
| 1 | To install the Flume tarball on Linux-based systems | # Run the following commands:  $ cd /usr/local/lib  $ sudo tar -zxvf <path\_to\_flume-ng-1.3.0-cdh4.2.0.tar.gz>  $ sudo mv flume-ng-1.3.0-cdh4.2.0.tar.gz flume-ng |
| 2 | Configuration | # To complete the configuration of a tarball installation, you must set your PATH variable to include the bin/ subdirectory of the directory where you installed Flume. For example:  $ export PATH=/usr/local/lib/flume-ng/bin:$PATH |
| 3 | Flume-ng | # Flume RPM and Debian packages consist of three packages. Note: All Flume installations require the common code provided by flume-ng.   1. flume-ng — Everything you need to run Flume 2. flume-ng-agent — Handles starting and stopping the Flume agent as a service 3. flume-ng-doc — Flume documentation |
| 4 | **Configuration files:** | # Flume 1.x provides a template configuration file for flume.conf called conf/flume-conf.properties.template and a template for flume-env.sh called conf/flume-env.sh.template.   1. flume.conf 2. flume-env.sh |
| 4.1 | flume.conf | # Copy the Flume template property file conf/flume-conf.properties.template to conf/flume.conf, then edit it as appropriate.  $ sudo cp conf/flume-conf.properties.template conf/flume.conf  # This is where you define your sources, sinks, and channels, and the flow within an agent |
| 4.2 | Flume-env.sh | # Optionally, copy the template flume-env.sh file conf/flume-env.sh.template to conf/flume-env.sh.  $ sudo cp conf/flume-env.sh.template conf/flume-env.sh  The flume-ng executable looks for a file named flume-env.sh in the conf directory, and sources it if it finds it. Some use cases for using flume-env.sh |
| 5 | Verifying the Installation | # At this point, you should have everything necessary to run Flume, and the flume-ng command should be in your $PATH. You can test this by running:  $ flume-ng help |
| 6 | Running Flume | # If Flume is installed via an RPM or Debian package, you can use the following commands to start, stop, and restart the Flume agent via init scripts:  $ sudo service flume-ng-agent <start | stop | restart>  You can also run the agent in the foreground directly by using the flume-ng agent command:  $ /usr/bin/flume-ng agent -c <config-dir> -f <config-file> -n <agent-name>  For example:  $ /usr/bin/flume-ng agent -c /etc/flume-ng/conf -f /etc/flume-ng/conf/flume.conf -n agent |
| **7** | **Files Installed by the Flume RPM and Debian Packages** |  |
| 7.1 | Config Directory | /etc/flume-ng/conf |
| 7.2 | Config File | # This config will be picked-up by the flume agent startup script.  /etc/flume-ng/conf/flume.conf |
| 7.3 | Template of User Customizable Config File | # Contains a sample config. To use this config you should copy this file onto /etc/flume-ng/conf/flume.conf and then modify as appropriate  /etc/flume-ng/conf/flume-conf.properties.template |
| 7.4 | Template of User Customizable environment file | # If you want modify this file, copy it first and modify the copy  /etc/flume-ng/conf/flume-env.sh.template |
| 7.5 | Daemon Log Directory | # Contains log files generated by flume agent  /var/log/flume-ng |
| 7.6 | Default Flume Home | # Provided by RPMS and DEBS  /usr/lib/flume-ng |
| 7.7 | Flume Agent startup script | # Provided by RPMS and DEBS  /etc/init.d/flume-ng-agent |
| 7.8 | Recommended tar.gz Flume Home | # Recommended but installation dependent  /usr/local/lib/flume-ng |
| 7.9 | Flume Wrapper Script | # Called by the Flume Agent startup script  /usr/bin/flume-ng |
| 7.10 | Flume Agent configuration file | # Allows you to specify non-default values for the agent name and for the configuration file location  /etc/default/flume-ng-agent |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  | **General Commands** |  |
| 1 | Starting an Agent | An agent is started using a shell script called flume-ng which is located in the bin directory of the Flume distribution. You need to specify the agent name, the config directory, and the config file on the command line:  $ bin/flume-ng agent -n $agent\_name -c conf -f conf/flume-conf.properties.template  Now the agent will start running source and sinks configured in the given properties file. |
| 2 | Version | # Version 1.5.0  >flume-ng version |
| 3 | Flume files | # To check how the flume configuration files looks like  >cd /opt/examples/flume/conf |
| 4 | Property files | Flume-env.sh  Flume.conf |
| 4.1 | Flume.conf | # sample configuration data in Flume.conf. Here agent1 is agent name…  # Describe/Configure source1  agent1.sources.source1.type=exec  agent1.sources.source1.command = tail -F /opt/gen\_logs/logs/access.logs  *<The above highlighted command says to get last 10 lines (using tail) from access.logs. it will periodically run this command to get latest log>*  # Describe solrSink  agent1.sinks.solrSink.type=org.apache.flume.sink.solr.morphline  agent1.sinks.solrSink.batchSize=1000  etc...  #The maximum size of the transaction supported by channel  agent1.channels.channel1.capacity=20000  agent1.channels.channel1.transactionCapacity=1000 |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |