# Lesson 1: Load Scenarios

We will look at four different load scenarios:

1. Data at rest
2. Data in motion
3. Data from a web server or a database log,
4. Data from a data warehouse

## Understand how to load data at rest, in motion

Data at rest is data that is already in a file in some directory. It is at rest, meaning that no additional updates are planned on this data and it can be transferred as is.

The transfer can be accomplished using standard HDFS shell commands, for example., cp or copyFromLocal or put, or using the BigInsights web console.

Q) The HDFS copyFromLocal command can be used to

1. capture streaming data that you want to store in Hadoop
2. ensure that log files which are actively being used to capture logging from a web server are moved into Hadoop
3. move data from a relational database or data warehouse into Hadoop
4. **None of the above**

What about when data is in motion?

This is data that is continuously being updated: New data might be added regularly to these data sources, Data might be appended to a file, or Discrete or different logs might be getting merged into one log.

You need to have the capability of merging the files before copying them into Hadoop. Examples of data in motion include:

* Data from a web server such as WebSphere Application
* Server or an Apache web server
* Data in database server logs
* Application logs

## Understand how to load data from common data sources e.g. RDBMS

When moving data from a data warehouse, or any RDBMS for that matter, we could export the data and then use Hadoop commands to import the data. There is a separate video on Sqoop in this course. If you are working with a Netezza system, then you can use the Jaql Netezza module to both read from and write to Netezza tables. Data can also be moved using BigSQL Load.

We also have Flume. You will see separate videos later dealing just with Flume. Flume is a three-tiered distributed service for data collection and possibly processing of the data that consists of logical nodes.

The first tier, or agent tier, has Flume agents installed at the sources of the data. These agents then send their data to the second tier, or collector tier. The collectors aggregate the data and in turn forward the data to the final storage tier such as HDFS. Each logical node has a source and a sink.

The source tells from where to collect data

The sink specifies to where the data is to be sent.

Interceptors (sometimes called Decorators or Annotators) can be optionally configured to allow for some simple data processing on data it is passed through.

Flume uses the concept of a physical node. A physical node corresponds to a single Java process running on one machine in a cluster as a single JVM. Here the concepts of physical machine and node are usually synonymous. But sometimes a physical node can host multiple logical nodes. We will see that Flume is a great tool for collecting data from a web server or from database logs. Another, alternate, approach here would be to use Java Management Extension (JMX) commands.

# Lesson 2: Using Sqoop

## Scoop overview

Apache Sqoop is a tool designed for efficiently transferring bulk data between Hadoop and structured data stores such as relational databases.

Q) What is the primary purpose of Sqoop in the Hadoop architecture?

1. To "catch" logging data as it is written to log files and move it into Hadoop
2. To schedule scripts that can be run periodically to collect data into Hadoop
3. **To import data from a relational database or data warehouse into Hadoop**
4. To move static files from the local file system into HDFS
5. To stream data into Hadoop

Q) Sqoop can be used to either import data from relational tables into Hadoop or export data from Hadoop to relational tables. True or false?

Sqoop has a command-line interface for transferring data. It supports incremental loads to/from a single database table, or a free-form SQL query, as well as scripts that can be run whenever needed to import updates made to a database since the last import.

Sqoop can be used also to populate tables in Hive or HBase. Sqoop provides a set of high-performance open-source connectors that can be customized for your specific external connections. Sqoop offers specific connector modules that are designed for different product types. In this video we do not intended to cover all aspects of Sqoop, but rather give you an idea of the capabilities of Sqoop and let you know where it fits in the Hadoop ecosystem.

Sqoop successfully graduated from incubator status in March of 2012 and is now a top-level Apache project

Sqoop is designed to transfer data between relational database systems and Hadoop. It uses JDBC to access the relational systems.

Q) A Sqoop JDBC connection string must include

1. the name of the database you wish to connect to
2. the hostname of the database server
3. the port that the database server is listening on
4. the name of the JDBC driver to use for the connection
5. **All of the above**

To use it with BigInsights, you must copy the JDBC driver JAR for the relational database to be accessed into the $SQOOP\_HOME/lib directory so that the driver can be used by the Sqoop software.

Sqoop accesses the database so that it can understand the schema of the data involved in a transfer. It then generates a MapReduce application to import or export the data from/to the database/. When you use Sqoop to import data into Hadoop, Sqoop generates a Java class that encapsulates one row of the imported table. You have access to the actual source code for the generated Java class. This can allow you to quickly develop other MapReduce applications that use the records that Sqoop stored into HDFS.

## Import data from a relational database table into HDFS

## Use Sqoop import and export command

Sqoop import/export –connect jdbc:db2://your.db2.com:50000/yourDB \

–username db2user –password yourpassword --table db2table \

--target-dir sqoopdata…

The connection information is the same whether you are doing an import or an export. You specify a JDBC connection string, the username, and the password.

In the example on the slide, you use the keyword import or the keyword export — just one, not at the same time — depending on the action you want to perform.

In this example you are connecting to a DB2 system that listens on port 50000 and is running on a system with a hostname of your.db2.com. The connection is made with a userid of db2user and a password of db2password.

Note here that you follow the usual UNIX/Linux convention that single letter parameters use a single-dash, or, as in this case, double-dash because all the parameters are word parameters (for example, dash dash connect or dash dash username). Additional required and optional arguments are not shown.

The Sqoop import command is used to extract data from a relational table and load it into Hadoop. Each row in HDFS comes from a row in the corresponding table. The resulting data in HDFS can be stored as text files or binary files, as well as imported directly into HBase or Hive. By default, all columns of all rows are imported, however, there are arguments that allow you to specify columns or specify a WHERE clause to limit the rows. You can even specify your own query to access the relational data. If you want to specify the location of the imported data, use the --target-dir argument. Otherwise, the target directory name will be the same as the table name.

Q) When importing data via Sqoop, the imported data can include

1. a collection of data from multiple tables via a join operation, as specified by a SQL query
2. specific rows and columns from a specific table
3. all of the data from a specific table
4. **All of the Above**

Q) Sqoop uses MapReduce jobs to import and export data, and you can configure the number of Mappers used. True or false?

Sqoop imports the data in parallel. You can override the number of mappers that Sqoop is to use. The default is 4. To split the data across multiple mappers, by default, Sqoop uses the primary key of the table. It determines the minimum and maximum values for the key and then assumes an even distribution of values.

* --split-by
  + argument to have the distribution work with a different column.
  + If the table does not have an index column, or has a multi-column key, then you must specify the split-by column parameter,
* --columns
  + only import data from a subset of columns
  + the column names are comma separated.
* --where
  + Limits the rows
  + supply your own query that returns the rows to be imported
* --query “Select \* From Table Join OtherTable On Keys”
  + This allows for greater flexibility, for example allowing you to get data by joining tables.

By default the imported data is in

* delimited text format (--as-textfile).
  + Optional parameters:
    - allow you to import in binary format (--as-sequencefile)
    - as an Avro data file ( as avrodatafile).

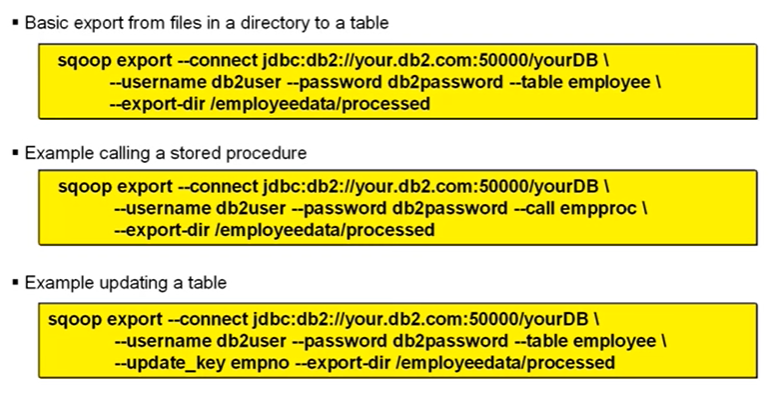
Q) When importing data via Sqoop, the incoming data can be stored as

1. Serialized Objects
2. JSON
3. XML
4. **None of the Above**

Also, you can override the default in order to have the data compressed.

### Export Modes

* Insert mode
  + The Sqoop export command reads data in Hadoop and places it into relational tables (you export from HDFS into a database). The target table must already exist and you can specify your own parsing specifications. By default, Sqoop inserts rows into a relational table. This is primarily intended for loading data into a new table. If there are any errors when doing the insert, the export process will fail.
* The update mode
  + causes Sqoop to generate update statements. To do updates, you must specify the --update-key argument. Here you tell Sqoop which table column (or comma-separated columns) to use in the WHERE clause of the update statement. If the update does not modify a row, it is not considered to be an error. The condition just goes undetected.
  + --update-mode allowinsert to be specified — these are databases that have an UPSERT command (UPSERT does an update if the row exists, but otherwise inserts a new row).
* Call mode.
  + Sqoop calls and passes the record to a stored procedure.
* --export-dir parameter defines the location of the files in HDFS that are to be exported from HDFS to put records into the database.



Now for a couple of additional pieces of information.

### Parsing Data

By default, Sqoop assumes that it is working with comma-separated fields and that each record is terminated by a newline. Both the import and export commands have the facility to allow you to override this behavior.

Remember, that when data is imported into Hadoop, you are given access to the Java source for the Java class what was generated. If your data was not in the default format and, if the data that you are exporting is in that same format, then you can use parts of that same code to read the data.

### Transactions

What about committing data to your database? Transactions? When Sqoop is inserting rows into a table, it generates a multi-row insert. Each multi-row insert handles up to 100 rows. The mapper then does a commit after 100 statements are executed. This means that 10,000 rows are inserted before being committed. Also, each export mapper in the generated MapReduce program commits with separate transactions.

# Lesson 3: Flume Overview

## Describe Flume and its uses

You should note the origin of the word Flume. It comes from Latin, via French. The Latin word flumen / flumen is is a river.

A flume is an open artificial water channel in the form of a gravity chute that leads water from a diversion dam or weir using a natural flow downhill.

Flumes are used in the transportation of logs in the logging industry. The use of the word Flume here — with Flume software — is thus metaphorical.

Flume — in the world of Hadoop — is used typically to transport web log records or database log records, and not wooden logs, from the log file where records were deposited to a central repository in the Hadoop distributed file system (HDFS) for analysis by MapReduce programs.

Q) What is the primary purpose of Flume in the Hadoop architecture?

1. **To "catch" logging data as it is written to log files and move it into Hadoop**
2. To schedule scripts that can be run periodically to collect data into Hadoop
3. To import data from a relational database or data warehouse into Hadoop
4. To move static files from the local file system into HDFS
5. To stream data into Hadoop

Flume is built on the concept of flows. The various sources of the data sent through Flume may have different batching or reliability setup. Often logs are continually being added to and what we want are the new records as they are added.

## How Flume works

Any number of Flume agents can be chained together.

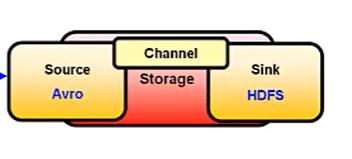
You begin by starting a Flume agent where the data originates.

This data then flows through a series of nodes that are chained together. Every Flume agent works with both a source and a sink. (Actually, a single agent can work with multiple sources and multiple sinks.)

Q) Flume agents can run on multiple servers in the enterprise, and they can communicate with each other over the network to move data. True or false?

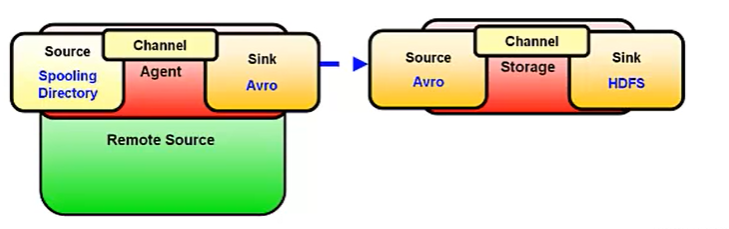
Sources and sinks are wired together via a channel.

Each node receives data as "source," stores it in a channel, and sends it via a "sink."



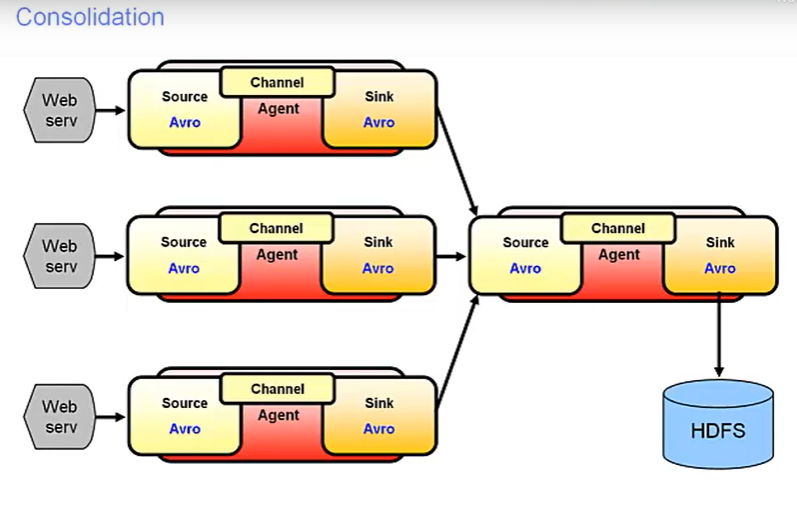
An agent running on one node can pass the data to an agent on a different node. The agent on the second node also works with a source and a sink.

Q) To pass data from a Flume agent on one node to another, you can configure an Avro sink on the first node and an Avro source on the second. **True** or false?

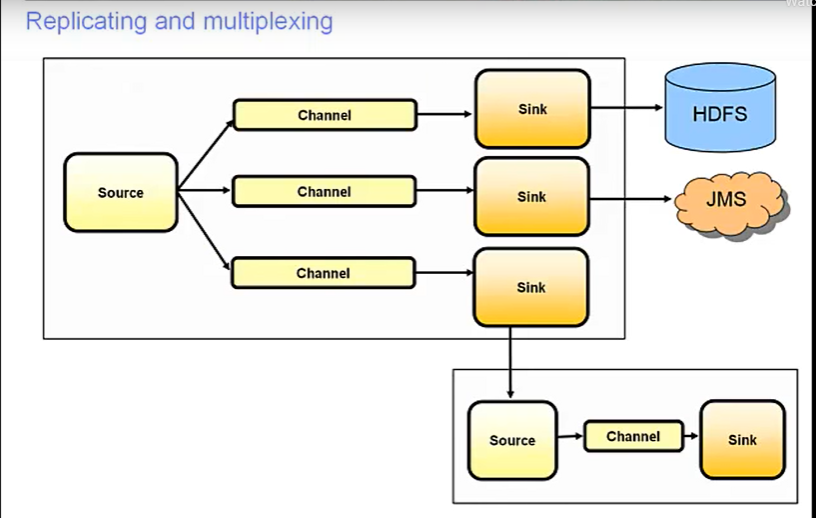


There are several different types of sources and sinks supplied with the product. For data to be passed from one agent to another, we can have an Avro sink on the first communicating with an Avro source on the second.

Avro is a remote procedure call and serialization framework that is a separate Apache project. It uses JSON for defining data types and protocols and it serializes data in a compact binary format.



Flume supports more than just a multi-tiered topology. This is an example of a consolidation topology. Here a single Avro source receives data from multiple Avro sinks.



Both a replication and a multiplexing topology are also supported. In a replicating topology, log events from the source are passed to all channels connected to that source. In a multiplexing topology, data in the header area of a log event can be queried and used to distribute the event to one or more channels.

# Lesson 4: Using Flume

## List the Flume configuration components

Flume components are defined in a configuration file.

Where multiple agents running on the same node, they can all be defined in the one configuration file. For each agent, you define the components:

* The source(s)
* The sink(s)
* The channel(s)

For an agent to run, it must have a source from which to get data, a sink that writes to a target, and a channel which specifies where the data is to be temporarily held until it is written by the sink.

This information is supplied to the agent via the configuration file that the agent loads at startup.

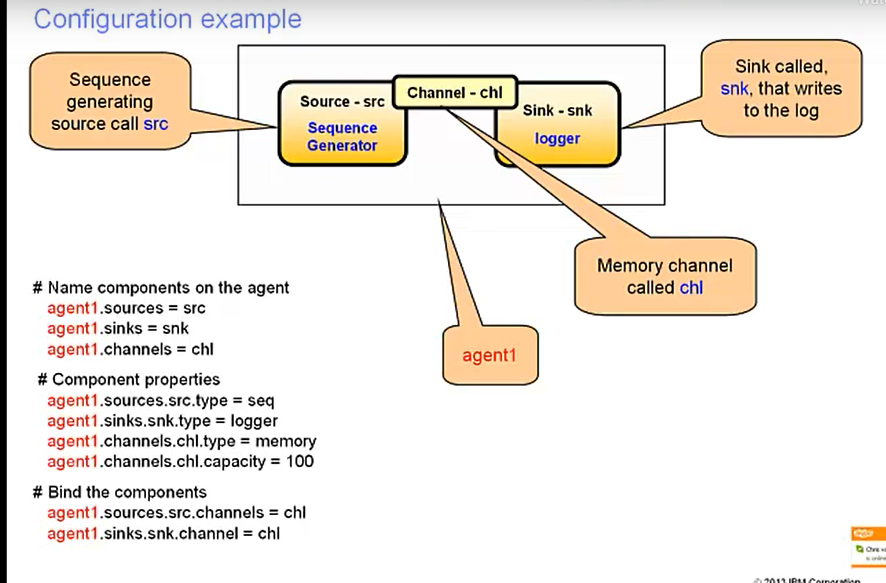
Within the configuration file each source, sink, and channel are given names. Properties are assigned to each component and you define the relationships between the components.

The configuration file is like a Java properties file. Let’s start out by looking at a configuration example. The intention here is not to go into a lot of detail but to begin to lay a foundation so that the following information makes a little sense.

Q) Flume agent configuration is specified using

1. CSV
2. **a text file, similar to the Java.properties format**
3. JSON
4. XML, similar to Sqoop configuration

By the way, there is no required order in defining components, specifying properties, and defining relationships.



The definitions file is declarative in nature and not procedural. In this visual, the name of the agent to run on this system is agent1. (More than one agent can run on a system. This example, for ease of learning, only has one.)

Q) When using Flume, a Source and a Sink are "wired together" using an Interceptor. True or false? (channel)

* The source for agent1 is going to be a sequence generator with a name of src.
* (A sequence generator continuously generates events and is used for testing.)
* The sink, in this case, is a logger sink with a name of sink.
* (A logger sink logs events at the INFO level and is also used for testing.)
* The source and the sink are wired together via a channel named chl.
* Next look at the configuration file definitions.
* Note that all statements begin with agent1. (agent1 dot).
* These are attributes of agent agent1. This is how an agent detects applicable configuration statements. As stated before, it is possible to run multiple agents on a system, and all those agents can be configured the same configuration file.
* Agents only pay attention to configuration statements that are prefixed by their name.

The three sections:

Name the components on the agent

Provide component properties Bind the components

The third of these — “bind the components” — is the set of statements that wire the source and the sink together. Since both src and snk are connected to the same channel, they are wired together.

## Describe how to start and configure a Flume agent

Q) When you create the configuration file for a Flume agent, you must configure

1. an Interceptor
2. a Sink
3. a Channel
4. a Source
5. **All of the above**

### Sources

* Avro source
  + Listens on avro port
  + Receives events from external avro client steam
* Exec
  + Runs a specified unix command on start up
* Spooling directory
  + Reads data from files in a spooling directory
* Netcat
  + Listens on a given port and turns each line of text into an event
* Sequence generator
  + Continuously generates events with a counter
* Syslog source
  + Reads syslog data
* http
  + accepts events by http post and get
* jsonhandler
  + handles events in json format
* Legacy
  + Receives events from a flume 0.9.4 agent
* Custom
  + Your own source
* Scribble
  + Accepts events from scribe

Q) Flume provides a number of source types including

1. Elastic Search
2. HBase
3. Hive
4. HDFS
5. None of the Above

You see here a list of the various possible Flume sources. Each source has a type property and depending on the type of source, there will be other appropriate properties. Some of these other properties are required. You should consult the Flume User Guide to get detailed information about a particular source. We will look at just one of these: Avro Source.

Avro source: Listens on an Avro port and receives events from external Avro client streams.

When paired with an Avro sink from another agent, allows data to flow from one agent to another. This source requires a bind property which is either a hostname or an IP address and in both cases a port property.

### Interceptors

One of the optional properties for a source is an Interceptor. Interceptors provide the ability to either modify or drop an in-flight event. There are several predefined interceptors that come with Flume. In addition, tools are provided for you to code your own.

Each Flume event is composed of a header and a body. When data is processed by Flume, it is placed into the body of the event and by default there is nothing in the header. Interceptors allow you to update the header of an event.

The data in a header can then be used by channels in order to implement a multiplexing topology. Or some sinks, for example, the hdfs sink, are able to access data in the header. The hdfs source can use portions of the header data as part of the directory naming convention.

The supplied interceptors are:

* Timestamp:
  + This inserts the time at which the event was processed into the event header.
* HostType:
  + This inserts into the event header the hostname, or IP address of the host, on which the agent is running.
* Static:
  + This inserts the same fixed value into the header of each processed event.
* Regex filtering:
  + This applies a supplied regex expression against the body of the event. If there is a match the interceptor can be configured to either include or exclude the event.
* Regex extractor:
  + This applies a regex expression against the event body and extracts matched groups. The extracted group values are then appended to the eventís header.
* In addition, you can build a custom interceptor that allows for your own implementation of an interceptor. If there are multiple interceptors defined, they are processed in the order of their definition.

Sometimes you will see the terms decorator or annotator used for interceptors.

### Sinks

Sinks externalize data. As with sources, this page lists a number of sinks that are supplied with Flume. Here we have a list of available sinks for Flume, including the custom sink, allowing you to build your own.

* HDFS sink
  + Writes events to hdfs
* Logger sink
  + Logs events at INFO level
* Avro sink
  + Used with Flumes tiered collection support
* IRC sink
  + Sends messages to configured IRC destinations
* File Roll sink
  + Stores events on the local file system
* Null sink
  + Discards events
* HBaseSinks
  + Writes data to Hbase
* ElasticSearchSink
  + Writes data to Elastic Search
* Custom sink
  + Your own

Each sink has a specific type property and then depending on the sink type, various other properties. Some of the properties might be required, depending on the sink type.

Again, let us look at just one of them:

ElasticSearchSink. The ElasticSearchSink is an interface to the open-source Apache Lucene software. Lucene is an indexing mechanism. ElasticSearch can be used to search all kinds of documents with a scalable search solution that has near real-time search capability. Information on all the sources and sinks are available in the Flume documentation

### Channels

Channels are where the data is staged after having been read in by a source and not yet written out by a sink. There are three supplied channels and plus a fourth, the option of writing your own. These four channels provide for intermediate storage in memory, in a database with JDBC (but only the Derby database is currently supported), in a file, or in your own custom mechanism.

Q) Possible Flume channels include

1. The implementation of your own channel
2. File Storage
3. Database Storage
4. In Memory
5. **All of the Above**

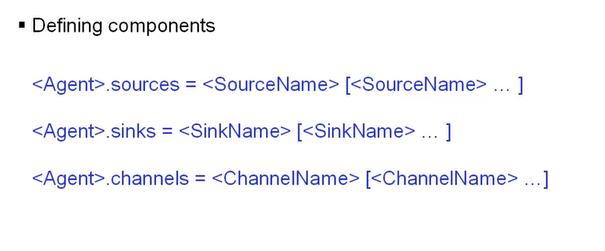
* Memory channel
  + Events are stored in an in-memory queue
* JDBC
  + Events are stored in a database
  + Currently supports imbedded derby
* File
  + Stores in files
* Custom
  + Implement your own

### Selectors

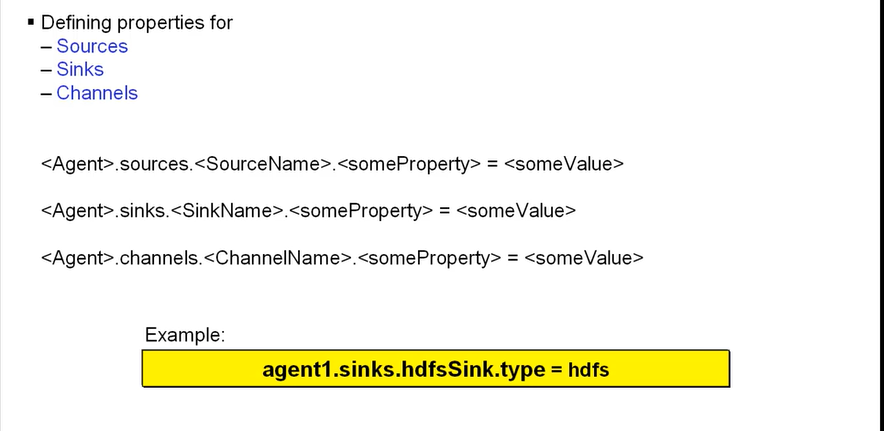
As with the other Flume components, each type of channel has its own properties. One of the properties for sources is selector.type which in most cases defaults to replicating. This option sends an event to each channel connect to the source. A second supplied selector.type is multiplexing. This allows for some basic logic to decide to which channel or channels the source should send the event. You also have the option to implement your own channel selector mechanism.

* Replicating channel selector
  + Default
  + Event is written to each sink
* Multiplexing channel selector
  + Event can be delivered to a subset of sinks
* Custom channel selector
  + Allows you implement your own

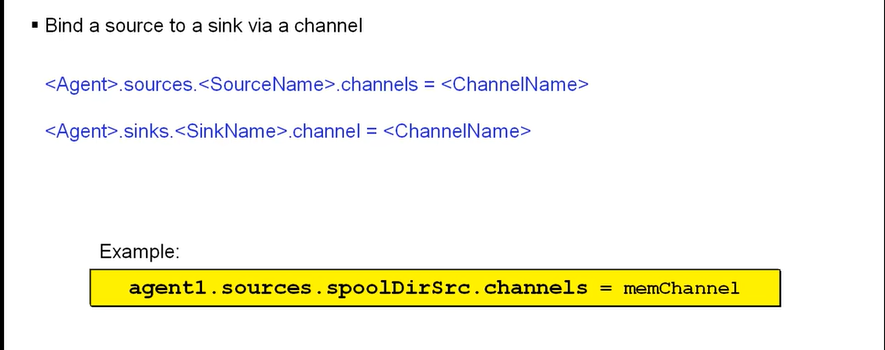
### Code



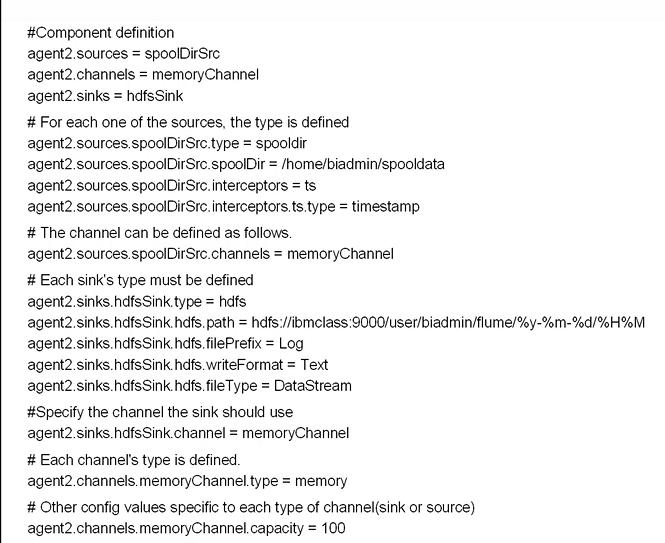
So far you have seen an example of some statements in a Flume configuration file. And you have been introduced to the types of sources, sinks, and channels. Now let’s make sure that you understand how to code configuration statements. Each statement in the configuration files begins with the name of an agent (thus, agent1 dot). As we noted previously, a single configuration file can be used with multiple agents. An agent only processes those statements that are prefixed with that agent’s name. So <Agent>.sources defines one or more sources. Multiple source names can be coded on the right side of the equal sign if they are separated by spaces. Both sinks and channels are defined using basically the same format.



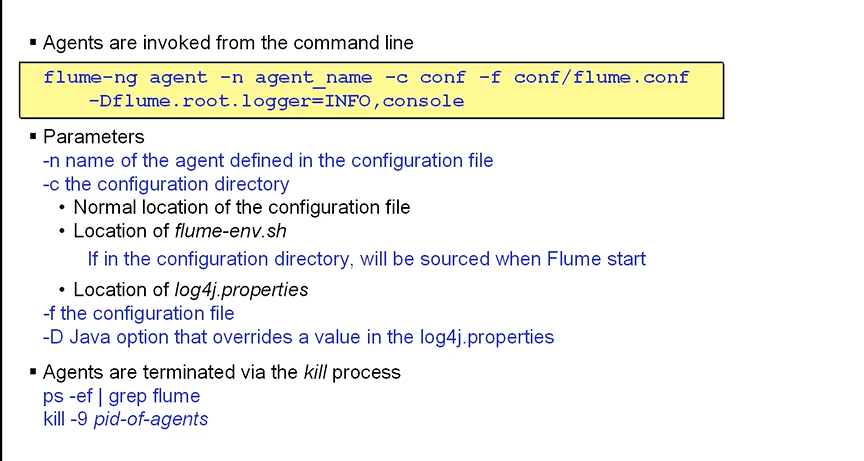
Defining properties for sources, sinks, and channels is a little bit more interesting. Once again, each statement is prefixed by the agent’s name (for example, agent1 doc). However, when defining a property for a component, the name of the component becomes part of the property definition. Assume that you have a sink called hdfsSink and you want to define the type of this sink as being an HDFS sink. Also, the agent that is to process the statement is agent1. You would code this example, as shown in the yellow box, as: agent1.sinks.hdfsSink.type = hdfs So each property definition consists of two “variable” names, the agent’s name and the component’s name. And two keywords, the type of component, (sources, sinks, or channels), and the property name. (Some property names may be compound names.) Sources and channels property definitions follow this same format.



After you have given your components their names and specified their properties, you need to connect them together. Essentially you are setting your sources and sinks equal to the channel that they are to use. When a source is defined to use the same channel as a sink, then the events processed by that source are passed to the sink and that sink then writes the event. So the following example, again shown in a yellow box, agent1.sources.spoolDirSrc.channels = memChannel defines a configuration statement that is to be processed by an agent with a name of agent1. The keyword sources indicates that this statement is for a source. The name of the source is spoolDirSrc. And the keyword channels indicates that this statement is intended to associate a source with at least one channel. The name of the channel appears to the right of the equal sign.



This is a more detailed example of the statements in a configuration file. But, by now, you should have a better idea as to the meaning of each statement. All of these statements are processed by an agent that has the name of agent2. There are three component definitions. These that define a source called spoolDirSrc, a sink called hdfsSink, and a channel called memoryChannel of type memory.



Agents are started from the command line and are generally started at host startup time. It is at this start up time that the configuration file is read and that the agent is given a name. The keyword agent, in the example shown, indicates that a Flume agent is to be started. Another option would perhaps be to start an Avro agent.

Once a Flume agent is started, the only way to terminate it is via the Linux kill statement.

# Lesson 5: Using Data Click

## Describe Data Click for BigInsights

## List the major components of Data Click