



## Capturing Data with Apache Flume

### Chapter 9

201509



**Goal:** This chapter provides a basic introduction to Flume and shows the student how to configure an agent, as well as several common sources and sinks.

## Course Chapters

1	Introduction	Course Introduction
2	Introduction to Hadoop and the Hadoop Ecosystem	Introduction to Hadoop
3	Hadoop Architecture and HDFS	
4	Importing Relational Data with Apache Sqoop	Importing and Modeling Structured Data
5	Introduction to Impala and Hive	
6	Modeling and Managing Data with Impala and Hive	
7	Data Formats	
8	Data File Partitioning	Ingesting Streaming Data
9	<b>Capturing Data with Apache Flume</b>	
10	Spark Basics	Distributed Data Processing with Spark
11	Working with RDDs in Spark	
12	Aggregating Data with Pair RDDs	
13	Writing and Deploying Spark Applications	
14	Parallel Processing in Spark	
15	Spark RDD Persistence	
16	Common Patterns in Spark Data Processing	
17	Spark SQL and DataFrames	
18	Conclusion	Course Conclusion

## Capturing Data with Apache Flume

---

**In this chapter you will learn**

- **What are the main architectural components of Flume**
- **How these components are configured**
- **How to launch a Flume agent**
- **How to configure a standard Java application to log data using Flume**

## Chapter Topics

### Capturing Data with Apache Flume

### Introduction to Flume

- **What is Apache Flume?**
- Basic Flume Architecture
- Flume Sources
- Flume Sinks
- Flume Channels
- Flume Configuration
- Conclusion
- Homework: Collect Web Server Logs with Flume

## What Is Apache Flume?

- **Apache Flume is a high-performance system for data collection**
  - Name derives from original use case of near-real time log data ingestion
  - Now widely used for collection of any streaming event data
  - Supports aggregating data from many sources into HDFS
- **Originally developed by Cloudera**
  - Donated to Apache Software Foundation in 2011
  - Became a top-level Apache project in 2012
  - Flume OG gave way to Flume NG (Next Generation)
- **Benefits of Flume**
  - Horizontally-scalable
  - Extensible
  - Reliable



cloudera

© Copyright 2010-2015 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera. 9-5

Early history of Flume [<http://www.slideshare.net/cloudera/chicago-data-summit-flume-an-introduction>] (note that this was for Flume OG, and is otherwise obsolete with the advent of Flume NG). Flume is now part of every major Hadoop distribution (CDH, MapR, Hortonworks, Intel, Pivotal).

Main point here is that instead of waiting for data to be collected into a big file so you can add it to your cluster (e.g., adding day-old server logs at 1:00 AM after the nightly log rotation runs) you can add the data to your cluster as it is produced. The importance of this is an underlying theme in the class and the fact that Flume has achieved such widespread adoption and built a huge community serves as evidence that this is important. As Hadoop moves from a batch-oriented system (i.e. MapReduce and Hive) to a more interactive system (i.e. Impala and Search), this is going to be essential. That is the future we're preparing the students for in this class.

Again, it's worth emphasizing that Flume is mainly concerned with the ingestion of data, not processing of the data.

Dictionary definition: Flume (noun):

- a deep narrow channel or ravine with a stream running through it.
- **an artificial channel conveying water, typically used for transporting logs or timber.**
- a water-chute ride at an amusement park.

## Flume's Design Goals: Reliability

---

- **Channels provide Flume's reliability**
- **Memory Channel**
  - Data will be lost if power is lost
- **Disk-based Channel**
  - Disk-based queue guarantees durability of data in face of a power loss
- **Data transfer between Agents and Channels is transactional**
  - A failed data transfer to a downstream agent rolls back and retries
- **Can configure multiple Agents with the same task**
  - For example, 2 Agents doing the job of 1 'collector' – if one agent fails then upstream agents would fail over

## Flume's Design Goals: Scalability

---

- **Scalability**

- The ability to increase system performance linearly – or better – by adding more resources to the system
- Flume scales horizontally
  - As load increases, more machines can be added to the configuration

## Flume's Design Goals: Extensibility

---

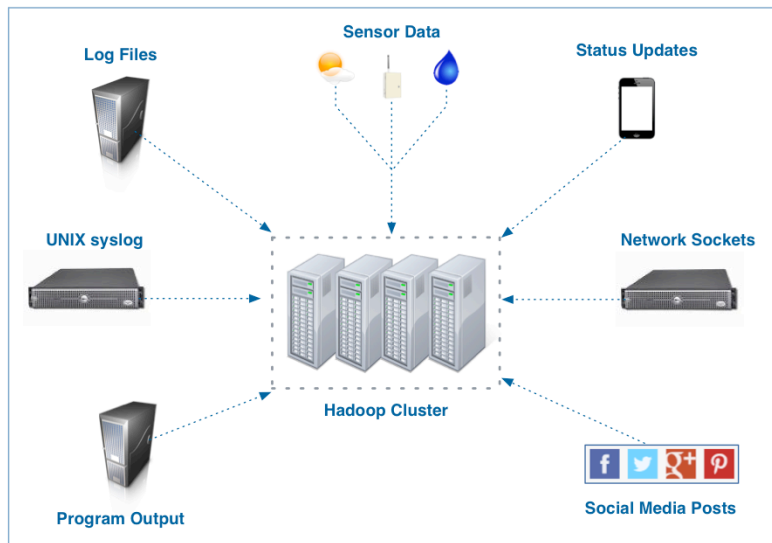
- **Extensibility**
  - The ability to add new functionality to a system
- **Flume can be extended by adding Sources and Sinks to existing storage layers or data platforms**
  - General Sources include data from files, syslog, and standard output from any Linux process
  - General Sinks include files on the local filesystem or HDFS
  - Developers can write their own Sources or Sinks

Reading data from Twitter streams may seem silly at first, but it's widely used by marketers, financial analysts and political scientists for "sentiment analysis" and to determine trending topics.

You might write your own connector to connect Flume up to some legacy system inside your company.



## Common Flume Data Sources

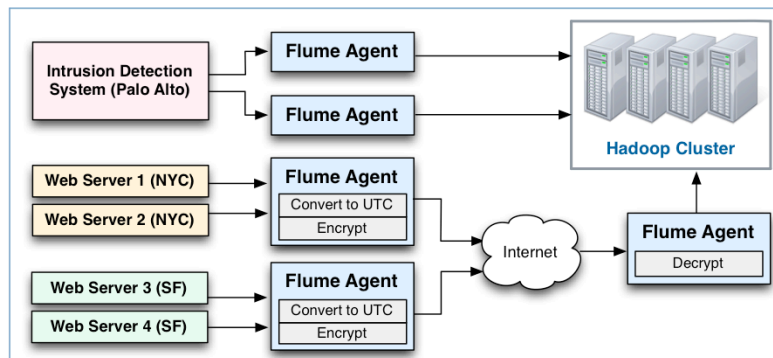


Here are some of the systems that produce data commonly ingested into the Hadoop cluster using Flume. We're just trying to show the variety on this slide, but the one that follows shows a more in-depth look at the architecture for a particular source.

## Large-Scale Deployment Example

- **Flume collects data using configurable “agents”**

- Agents can receive data from many sources, including other agents
- Large-scale deployments use multiple tiers for scalability and reliability
- Flume supports inspection and modification of in-flight data



cloudera

© Copyright 2010-2015 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera. 9-10

In this example, we have two distinct producers of data. The first is an IDS (i.e. a system that detects attempts to infiltrate important computer systems). The alerts that it generates are really important, so we send each message to two Flume agents so we're sure to get everything even if one fails. Those agents are in our local Palo Alto data center, so we just write the data into the cluster.

We also have a Web server farm split across two co-located data centers (New York City and San Francisco). We need to collect the log data that the servers produce, but since missing a record in a Web server log file isn't catastrophic, we decided that there is no need for failover here (if the agent fails, the logs will still be on the local filesystem for us to get later). The two data centers are in different timezones, so we convert the log timestamps to UTC for consistency, and then encrypt the data before we send it across the internet to a second tier Flume agent that decrypts the data and writes it to the cluster. This illustrates how we can inspect and modify data as it passes through the agent (i.e. using an interceptor). Interceptors are not covered in this course, but are in the Dev II course.

This is a somewhat complex (but common) example of how Flume is often deployed, included here to emphasize the scalability, reliability, and extensibility of Flume. However, the details of how to set up something of this complexity is an administration topic well outside the scope of this course. Our further examples are

## Chapter Topics

### Capturing Data with Apache Flume

### Introduction to Flume

- What is Apache Flume?
- **Basic Flume Architecture**
- Flume Sources
- Flume Sinks
- Flume Channels
- Flume Configuration
- Conclusion
- Homework: Collect Web Server Logs with Flume



© Copyright 2010-2015 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera. 9-11

Note on prev slide: "Flume supports inspection and modification of in-flight data."  
This is achieved through Interceptors.

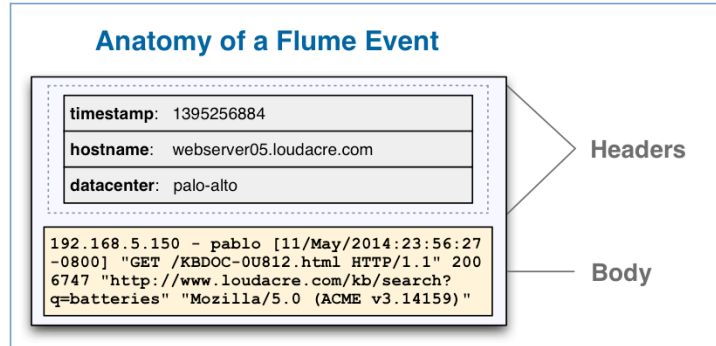
From the Flume User Guide: <http://flume.apache.org/FlumeUserGuide.html#flume-interceptors>

Flume has the capability to modify/drop events in-flight. This is done with the help of interceptors. Interceptors are classes that implement `org.apache.flume.interceptor.Interceptor` interface. An interceptor can modify or even drop events based on any criteria chosen by the developer of the interceptor. Flume supports chaining of interceptors.

```
public interface Interceptor {  
    public void initialize();  
    public Event intercept(Event event);  
    public List<Event> intercept(List<Event> list);  
    public void close();  
    public static interface Builder extends Configurable {  
        Interceptor build();  
    }  
}
```

## Flume Events

- An *event* is the fundamental unit of data in Flume
  - Consists of a body (payload) and a collection of headers (metadata)
- Headers consist of name-value pairs
  - Headers are mainly used for directing output



© Copyright 2010-2015 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera. 9-12

A flume event represents a single event (e.g., a page request to a Web server) and is similar to an e-mail message in that it has one body and some number of headers. Headers are metadata; they are an unordered set of key/value pairs (which might include things like the timestamp or hostname) and are often used for routing traffic in a large collection workflow (well beyond the scope of this class). Flume can add and modify headers on the fly, so you might add headers that specify details about the origin of data and use that to determine where in HDFS that data is written. The body is the payload and is simply an array of bytes, so it can contain any type of content.

Headers are part of the “control plane” – their values aren’t usually what you’re ultimately interested in storing but might help you in figuring out *where* to store the data (message body) that you care about. For example, you might have a header containing the timestamp at which a message was generated or the hostname from which a log file originated. You could use these values when you specify an output path in HDFS (e.g., setting `agent1.sinks.k1.hdfs.path = /flume/events/{host}/{%y-%m-%d}` in your configuration file would store the data in a directory based on the host header, and below that, within subdirectories based on the date portion of the timestamp). We show a basic example of this later in the chapter. While the body is a byte array, both header names and header values are Strings in the Flume API.

## Components in Flume's Architecture

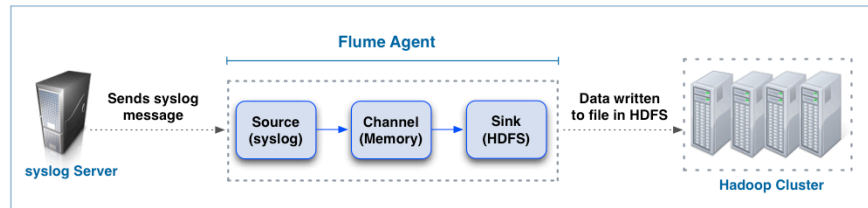
---

- **Source**
  - Receives events from the external actor that generates them
- **Sink**
  - Sends an event to its destination
- **Channel**
  - Buffers events from the source until they are drained by the sink
- **Agent**
  - Java process that configures and hosts the source, channel, and sink

## Flume Data Flow

■ This diagram illustrates how syslog data might be captured to HDFS

1. Message is logged on a server running a syslog daemon
2. Flume agent configured with syslog source receives event
3. Source pushes event to the channel, where it is buffered in memory
4. Sink pulls data from the channel and writes it to HDFS



Although this makes it seem like there is only one kind of sink (HDFS), as the student will soon see, there are many other sinks to which data could be written.

Also, be sure to point out that the Flume agent typically runs on the same machine that's generating the data (i.e. a server running syslog, in this case), but this machine is typically not part of the Hadoop cluster.

## Chapter Topics

### Capturing Data with Apache Flume

### Introduction to Flume

- What is Apache Flume?
- Basic Flume Architecture
- **Flume Sources**
- Flume Sinks
- Flume Channels
- Flume Configuration
- Conclusion
- Homework: Collect Web Server Logs with Flume

## Notable Built-in Flume Sources

- **Syslog**
  - Captures messages from UNIX syslog daemon over the network
- **Netcat**
  - Captures any data written to a socket on an arbitrary TCP port
- **Exec**
  - Executes a UNIX program and reads events from standard output \*
- **Spooldir**
  - Extracts events from files appearing in a specified (local) directory
- **HTTP Source**
  - Receives events from HTTP requests

\* Asynchronous sources do not guarantee that events will be delivered



© Copyright 2010-2015 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera. 9-16

This is by no means a complete list of built-in sources (see the Flume docs for that [<http://flume.apache.org/FlumeUserGuide.html>]). The syslog source can capture data over TCP or UDP. The netcat source only supports TCP. There is also an Avro source, but this is primarily used for sending data to another Flume agent (running an Avro sink) in a multi-tiered setup, so we don't go into it here.

The exec source can run the program on a recurring basis (param:restart=true) with a delay specified in milliseconds (param: restartThrottle). This is helpful in case you want to gather machine stats (e.g., by running ps, vmstat, df, or similar programs) and is also an easy way of integrating with legacy programs (i.e. write a shell script that does whatever you need). The docs explain why exec is unreliable: "The problem with ExecSource and other asynchronous sources is that the source can not guarantee that if there is a failure to put the event into the Channel the client knows about it. In such cases, the data will be lost." It goes on to explain that, essentially, messages from asynchronous sources are ephemeral and Flume cannot communicate with an arbitrary program to tell it that, for example, the channel is full and cannot accept any more events. Hari's presentation [<http://www.slideshare.net/ydn/flume-hug>] says on slide 20 that syslog is also asynchronous and unreliable, but since the Flume user guide does not state this, neither do I.

The HTTP source can accept both GET or POST requests, but GET support is considered experimental.



## Chapter Topics

### Capturing Data with Apache Flume

### Introduction to Flume

- What is Apache Flume?
- Basic Flume Architecture
- Flume Sources
- **Flume Sinks**
- Flume Channels
- Flume Configuration
- Conclusion
- Homework: Collect Web Server Logs with Flume

## Interesting Built-in Flume Sinks

---

- **Null**
  - Discards all events (Flume equivalent of `/dev/null`)
- **Logger**
  - Logs event to INFO level using SLF4J
- **IRC**
  - Sends event to a specified Internet Relay Chat channel
- **HDFS**
  - Writes event to a file in the specified directory in HDFS
- **HBaseSink**
  - Stores event in HBase

SLF4J: Simple Logging Façade for Java

This is likewise not a complete list of all built-in Flume sinks.

The Null and Logger sinks are mainly used for testing purposes, though in a large/complex flow, you might set up multiplexing and conditional routing such that valuable messages (like records from a firewall log related to a possible intrusion) go to HDFS (or IRC!) while less valuable messages (like bootup messages from the firewall) go to the null sink. The logger sink logs messages using SLF4J (and ultimately to log4j or whatever it's configured to use), not to the system log or to the Hadoop logs.

## Chapter Topics

### Capturing Data with Apache Flume

### Introduction to Flume

- What is Apache Flume?
- Basic Flume Architecture
- Flume Sources
- Flume Sinks
- **Flume Channels**
- Flume Configuration
- Conclusion
- Homework: Collect Web Server Logs with Flume

## Built-In Flume Channels

---

- **Memory**
  - Stores events in the machine's RAM
  - Extremely fast, but not reliable (memory is volatile)
- **File**
  - Stores events on the machine's local disk
  - Slower than RAM, but more reliable (data is written to disk)
- **JDBC**
  - Stores events in a database table using JDBC
  - Slower than file channel

The JDBC channel is not recommended: [[https://mail-archives.apache.org/mod\\_mbox/flume-user/201210.mbox/%3CCAFukC=7Eo9FDozu8bW\\_BP\\_mWA62ADvg266b8K5qbbETrFgT=tQ@mail.gmail.com%3E](https://mail-archives.apache.org/mod_mbox/flume-user/201210.mbox/%3CCAFukC=7Eo9FDozu8bW_BP_mWA62ADvg266b8K5qbbETrFgT=tQ@mail.gmail.com%3E)]

Other channels:

Kafka Channel

Spillable Memory Channel

Pseudo Transaction Channel (NOT for production)

Custom Channel (you implement the Channel interface)

## Chapter Topics

### Capturing Data with Apache Flume

### Introduction to Flume

- What is Apache Flume?
- Basic Flume Architecture
- Flume Sources
- Flume Sinks
- Flume Channels
- **Flume Configuration**
- Conclusion
- Homework: Collect Web Server Logs with Flume



© Copyright 2010-2015 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera. 9-21

NOTE: If Flume is started from an init script (a typical setup in production) rather than the command line as we do in class, then it likely runs as a username like "flume" or "flume-ng" but will often need to impersonate some other user account so that file permissions and ownership are correct. Although it's delving a bit too far into Admin territory, this can be achieved through "secure user impersonation," which is somewhat like sudo for Hadoop services. It provides the ability for a specified user to impersonate one or more user accounts (an asterisk is a wildcard that will allow any account) on one or more hosts (a wildcard is allowed here too). See the "Writing as different users across multiple HDFS sinks in a single Flume agent" section of the CDH Security Guide [[http://www.cloudera.com/content/cloudera/en/documentation/core/latest/topics/cdh\\_sg\\_flume\\_security\\_props.html](http://www.cloudera.com/content/cloudera/en/documentation/core/latest/topics/cdh_sg_flume_security_props.html)] to read about secure impersonation.

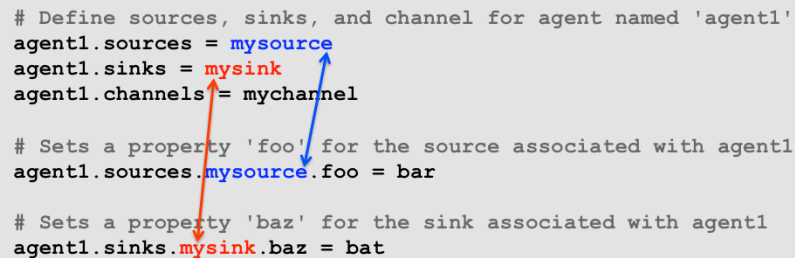
## Flume Agent Configuration File

- **Flume agent is configured through a Java properties file**
  - Multiple agents can be configured in a single file
- **The configuration file uses hierarchical references**
  - Each component is assigned a user-defined ID
  - That ID is used in the names of additional properties

```
# Define sources, sinks, and channel for agent named 'agent1'
agent1.sources = mysource
agent1.sinks = mysink
agent1.channels = mychannel

# Sets a property 'foo' for the source associated with agent1
agent1.sources.mysource.foo = bar

# Sets a property 'baz' for the sink associated with agent1
agent1.sinks.mysink.baz = bat
```

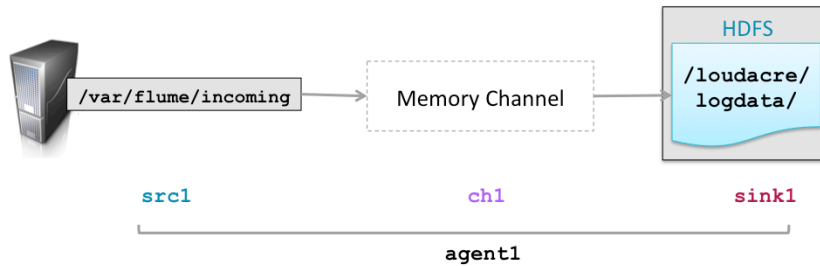


Although you can define multiple Flume agents in a single configuration file, we do not do so in the examples shown in class. Be sure to emphasize on this slide that this example defines an agent named 'agent1' and that the agent name is arbitrary (we might instead call it something more descriptive, like "syslog-hdfs-agent").

This example shows an example of the general pattern for a Flume agent configuration file. The top section is where the components for an agent are defined and given an ID, while the remaining sections are where those components are configured. A complete working example is shown on the next slide.

## Example: Configuring Flume Components (1)

- Example: Configure a Flume Agent to collect data from remote spool directories and save to HDFS



This diagram shows the example that will be implemented by the configuration file on the next slide.

## Example: Configuring Flume Components (2)

```
agent1.sources = src1
agent1.sinks = sink1
agent1.channels = ch1

agent1.channels.ch1.type = memory

agent1.sources.src1.type = spooldir
agent1.sources.src1.spoolDir = /var/flume/incoming
agent1.sources.src1.channels = ch1

agent1.sinks.sink1.type = hdfs
agent1.sinks.sink1.hdfs.path = /loudacre/logdata
agent1.sinks.sink1.channel = ch1
```

Connects source  
and channel

Connects sink  
and channel

- Properties vary by component type (source, channel, and sink)
  - Properties also vary by subtype (e.g., netcat source vs. syslog source)
  - See the Flume user guide for full details on configuration

This shows a basic, but complete, Flume agent configuration file similar to what the student will create in the upcoming homework. It does not matter whether you define the channel, source, or sink first. For readability, though, the named components like source, sink, and channel(s) are usually defined at the top of the file.

At the top, we define an agent (named `agent1` by virtue of the name used in the first three properties), a source (named `src1`), a sink (named `sink1`), and a channel (named `ch1`). The configuration file can hold the configuration for multiple agents, though this example shows only one. After defining the active components for this agent, we configure the channel as a memory-based channel (since we didn't set any additional properties, it uses the default values and will be capable of only storing 100 events before it fills up). After that we declare that the source will look for data in the (local) `/var/flume/incoming` directory. We then bind this to the memory channel. Finally, we set the sink to write data to a directory in HDFS and bind that sink to the channel.

There are other types of channels (file-based and JDBC-based, for example). These have different advantages as discussed in the Flume documentation, but are out of scope for this class. It's sufficient to understand that the memory channel is very fast but has limited capacity compared to the others.



## Aside: HDFS Sink Configuration

- Path may contain patterns based on event headers, such as timestamp
- The HDFS sink writes uncompressed SequenceFiles by default
  - Specifying a codec will enable compression

```
agent1.sinks.sink1.type = hdfs
agent1.sinks.sink1.hdfs.path = /loudacre/logdata/%y-%m-%d
agent1.sinks.sink1.hdfs.codec = snappy
agent1.sinks.sink1.channel = chl
```

- Setting fileType parameter to DataStream writes raw data
  - Can also specify a file extension, if desired

```
agent1.sinks.sink1.type = hdfs
agent1.sinks.sink1.hdfs.path = /loudacre/logdata/%y-%m-%d
agent1.sinks.sink1.hdfs.fileType = DataStream
agent1.sinks.sink1.hdfs.fileSuffix = .txt
agent1.sinks.sink1.channel = chl
```



"Specifying a codec will enable compression" – the weird uppercase 'C' in 'codeC' is how it appears in the documentation.

"Setting fileType parameter to DataStream writes raw data" – although this parameter is widely-used, no description of what it actually does currently appears in the Flume documentation. From reading the code for Flume 1.4.0, the `HDFSWriterFactory` class uses the value of the `fileType` parameter to select which class to use when writing the data to HDFS. If the value is set to 'SequenceFile' (or omitted, since that is the default), then the `org.apache.flume.sink.hdfs.HDFSSequenceFile` class (which encapsulates an output stream and writes data to a sequence file) is selected, enabling block-level compression (rather than the less-efficient record-level compression) if a codec is specified. If the value is 'DataStream' then the `org.apache.flume.sink.hdfs.HDFSDataStream` class is used (or `org.apache.flume.sink.hdfs.HDFSCompressedDataStream`, if compression is enabled), and data is written as raw bytes to the HDFS output stream. Thus, if the data is serialized as text, then you'll get text files in HDFS. If the data is serialized as Avro, then you'll get Avro files in HDFS. Although you could specify a file extension like ".csv" be aware that this does not magically make your data comma-delimited. Flume just writes the data you give it, so any formatting must be done in advance (or could also be done on the fly using an interceptor and/or custom serializer, as we'll see later).

## Starting a Flume Agent

### ▪ Typical command line invocation

- The `--name` argument must match the agent's name in the configuration file
- Setting root logger as shown will display log messages in the terminal

```
$ flume-ng agent \  
  --conf /etc/flume-ng/conf \  
  --conf-file /path/to/flume.conf \  
  --name agent1 \  
  -Dflume.root.logger=INFO,console
```

\* ng = Next Generation (prior version now referred to as og)

**cloudera** © Copyright 2010-2015 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera. 9-26

In CDH, the `/etc/flume-ng/conf/` directory contains additional configuration files, including the `flume-env.sh` (used to set classpath and JVM settings) and the `log4j.properties` file (used to control logging from the Flume agent). The `flume.conf` file in that directory is the one used when running an agent from the init scripts (the name and other details of the agent started by default from the init script are configured in `/etc/default/flume-ng-agent`). In a CM-managed environment, these files are not necessarily used since configuration is largely done from within CM itself.

Configuration in the file for any agents other than the one specified in the `--name` argument will be ignored, as will any misconfigured components.

Adding `-Dflume.root.logger=INFO,console` to the invocation when you start Flume can be handy for debugging (assuming you haven't changed the `log4j.properties` file), since it will show additional messages to the terminal window in which you started the agent.

## Chapter Topics

### Capturing Data with Apache Flume

### Introduction to Flume

- What is Apache Flume?
- Basic Flume Architecture
- Flume Sources
- Flume Sinks
- Flume Channels
- Flume Configuration
- **Conclusion**
- Homework: Collect Web Server Logs with Flume

## Essential Points

- **Apache Flume is a high-performance system for data collection**
  - Scalable, extensible, and reliable
- **A Flume agent manages the source, channels, and sink**
  - Source receives event data from its origin
  - Sink sends the event to its destination
  - Channel buffers events between the source and sink
- **The Flume agent is configured using a properties file**
  - Each component is given a user-defined ID
  - This ID is used to define properties of that component

**REVIEW QUESTIONS** (optional, but will help the instructor gauge whether the class is absorbing the material, and to provide some elasticity if you're running ahead of schedule):

- **Q1:** How would you describe the structure of a Flume event? (Answer: a map of name-value string pairs as the header and a byte array as the body)
- **Q2:** Which of the three channels we described has the fastest performance? (Answer: memory channel)
- **Q3:** What is the default format of data written by the HDFS sink? (Answer: uncompressed sequence files)

## Bibliography

---

The following offer more information on topics discussed in this chapter

- **Flume User Guide**

- <http://flume.apache.org/FlumeUserGuide.html>

## Chapter Topics

### Capturing Data with Apache Flume

### Introduction to Flume

- What is Apache Flume?
- Basic Flume Architecture
- Flume Sources
- Flume Sinks
- Flume Channels
- Flume Configuration
- Conclusion
- **Homework: Collect Web Server Logs with Flume**

## Homework: Collect Web Server Logs with Flume

---

- **In this homework assignment you will**
  - Configure Flume to ingest web server log data to HDFS
- **Please refer to your Homework description**