TAMPERE UNIVERSITY OF TECHNOLOGY

Apache Flume
Getting data into Hadoop

Problem

- Getting data into HDFS is not difficult:
  - **% hadoop fs --put data.csv .**
  - works great when data is neatly packaged and ready to upload
- Unfortunately, e.g. a webserver is creating data all the time
  - How often should a batch load data to HDFS happen? Daily? Hourly?
- The real need is a solution that can deal with streaming logs/data

Solution: Apache Flume

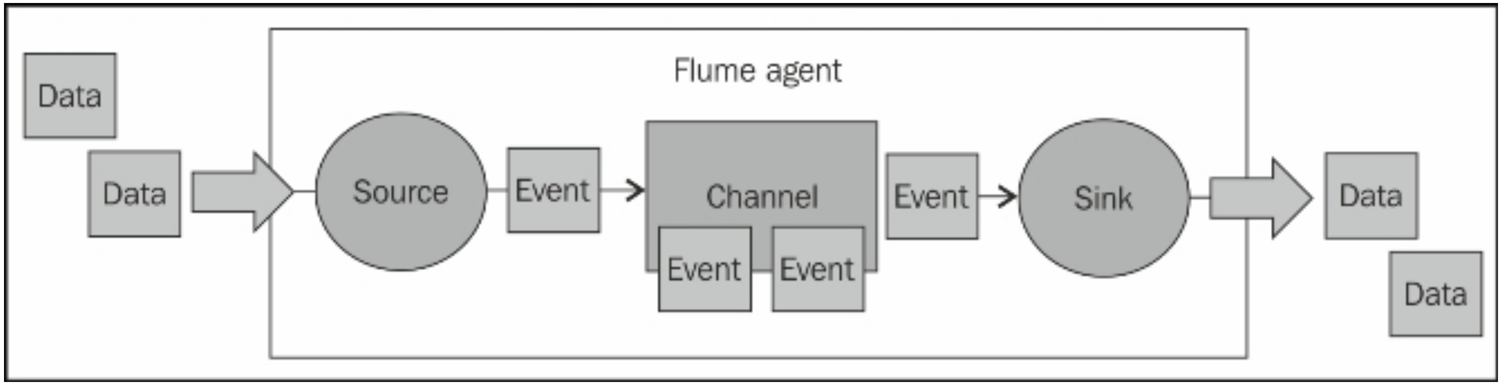
- Introduced in Cloudera's **CDH3** distribution
- versions 0.x: flume, 1.x: flume-ng

Overview

- Stream data (events, not files) from clients to sinks
- Clients: files, syslog, avro, …
- Sinks: HDFS files, HBase, …
- Configurable reliability levels
  - Best effort: “Fast and loose”
  - Guaranteed delivery: “Deliver no matter what”
- Configurable routing / topology

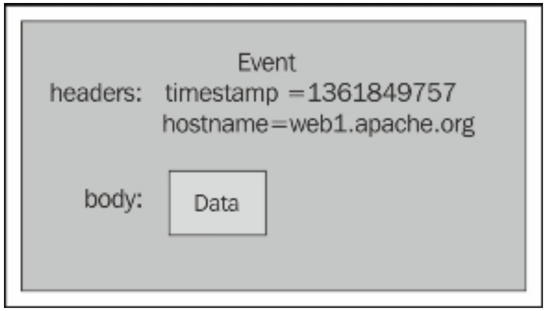
Architecture

Component	Function
Agent	The JVM running Flume. One per machine. Runs many sources and sinks.
Client	Produces data in the form of events. Runs in a separate thread.
Sink	Receives events from a channel. Runs in a separate thread.
Channel	Connects sources to sinks (like a queue). Implements the reliability semantics.
Event	A single datum; a log record, an avro object, etc. Normally around ~4KB.



Events

- Payload of the data is called an *event*
  - composed of zero or more headers and a body
- Headers are key/value pairs
  - making routing decisions or
  - carry other structured information



Event

headers: timestamp =1361849757
hostname=web1.apache.org

body: Data

Channels

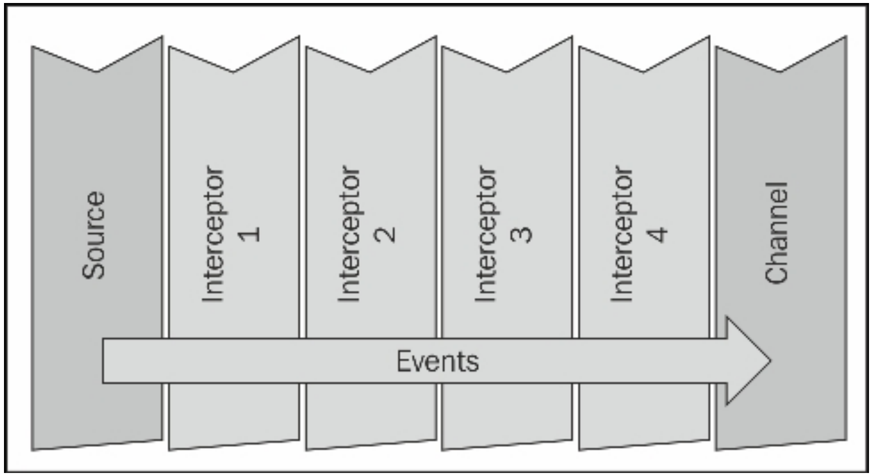
- Provides a buffer for in-flight events
  - after they are read from sources
  - until they can be written to sinks in the data processing pipelines
- Two (three) primary types are
  - a memory-backed/nondurable channel
  - a local-filesystem-backed/durable channel
  - (hybrid)

Channels

- The writing rate of the sink should faster than the ingest rare from the sources
  - ChannelException might lead to data loss

Interceptors

- An **interceptor** is a point in data flow where events can be inspected and altered
- zero or more interceptors can be chained after a source creates an event



Channel Selectors

- Responsible for how data moves from a source to one or more channels
- Flume comes with two selectors
  - **replicating channel selector** (the default) puts a copy of the event into each channel
  - **multiplexing channel selector** writes to different channels depending on headers
- Combined with interceptors forms the foundation for routing

Sinks

- Flume supports a set of sinks
  - HDFS, ElasticSearch, Solr, HBase, IRC, MongoDB, Cassandra, RabbitMQ, Redis, …
- HDFS Sink continuously
  - open a file in HDFS,
  - stream data into it,
  - at some point, close that file
  - start a new one

```
agent.sinks.k1.type=hdfs
agent.sinks.k1.hdfs.path=/path/in/hdfs
```

Sources

- Flume source consumes events delivered to it by an external source
  - like a web server

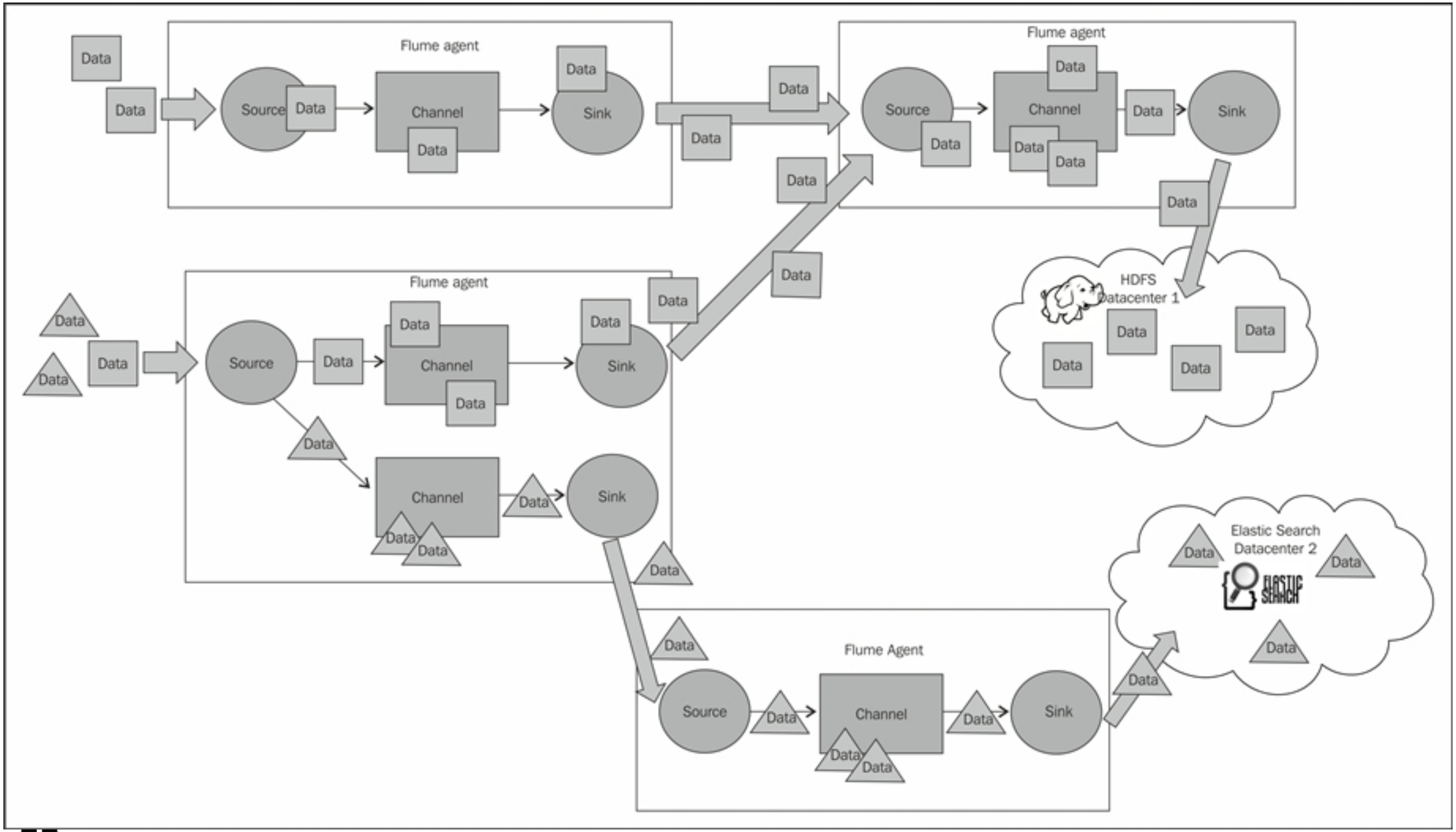
Tiered Collection

- Send events from agents to another tier of agents to aggregate
- Use an Avro sink (really just a client) to send events to an Avro source (really just a server) in another machine
- Failover supported
- Load balancing (soon)
- Transactions guarantee handoff

Tiered Collection Handoff

- Agent 1: Tx begin
- Agent 1: Channel take event
- Agent 1: Sink send
- Agent 2: Tx begin
- Agent 2: Channel put
- Agent 2: Tx commit, respond OK
- Agent 1: Tx commit (or rollback)

Tiered Data Collection



Apache Flume

- A source writes events to one or more channels
- A channel is the holding area as events are passed from a source to a sink
- A sink receives events from one channel only
- An agent can have many channels

Flume Configuration File

- Simple Java property file of key/value pairs
- Several agents can be configured in a single file
  - agents are identified by **agent identifier** (called a **name**)
- Each agent is configured, starting with three parameters:

```
agent.sources=<list of sources>
agent.channels=<list of channels>
agent.sinks=<list of sinks>
```

- Each source, channel and sink has a unique name within the context of that agent
  - Prefix for channel named access

    agent.channels.access
- Each item has a type
  - E.g. in-memory channel is *memory*

    agent.channels.access.type=memory

Hello, World!

```
agent.sources=s1
agent.channels=c1
agent.sinks=k1

agent.sources.s1.type = spooldir
agent.sources.s1.spoolDir = /etc/spool
…
agent.sinks.k1.type = hdfs
agent.sinks.k1.hdfs.path =
hdfs://localhost:9001/user/hduser/log-data
…
```

Hello, World!

- Config has one agent (called *agent*) with
  - a source named s1
  - a channel named c1
  - a sink named k1
- The s1 source's type is *spooldir*
  - Files appearing in */etc/spool* are ingested
- The type of the sink named *k1* is *hdfs*
  - writes data files to *log-data*

Command Line Usage

```
% flume-ng help
Usage: /usr/local/flume/apache-flume-1.6.0-bin/bin/flume-ng <command> [options]...

commands:
  help                    display this help text
  agent                    run a Flume agent
  avro-client               run an avro Flume client
  version                  show Flume version info
…
```

Command Line Usage

- The agent command requires 2 parameters
  - a configuration file to use and
  - the agent name
- Example

```
% flume-ng agent -n agent -f myConf.conf …
```

- Test

```
% cp log-data/* /etc/spool
```

Sources Code

```
public class MySource implements PollableSource {
    public Status process() {
        // Do something to create an Event..
        Event e = EventBuilder.withBody(…).build();
        // A channel instance is injected by Flume.
        Transaction tx = channel.getTransaction();
        tx.begin();
        try {
            channel.put(e);
            tx.commit();
        } catch (ChannelException ex) {
            tx.rollback();
            return Status.BACKOFF;
        } finally {
            tx.close();
        }
        return Status.READY;
}
```

Sinks Code

```
public class MySink implements PollableSink {
    public Status process() {
    Transaction tx = channel.getTransaction();
    tx.begin();
    try {
        Event e = channel.take();
        if (e != null) {
            // …
            tx.commit();
        } else {
            return Status.BACKOFF;
        }
    } catch (ChannelException ex) {
        tx.rollback();
        return Status.BACKOFF;
    } finally {
        tx.close();
    }
        return Status.READY;
    }
```