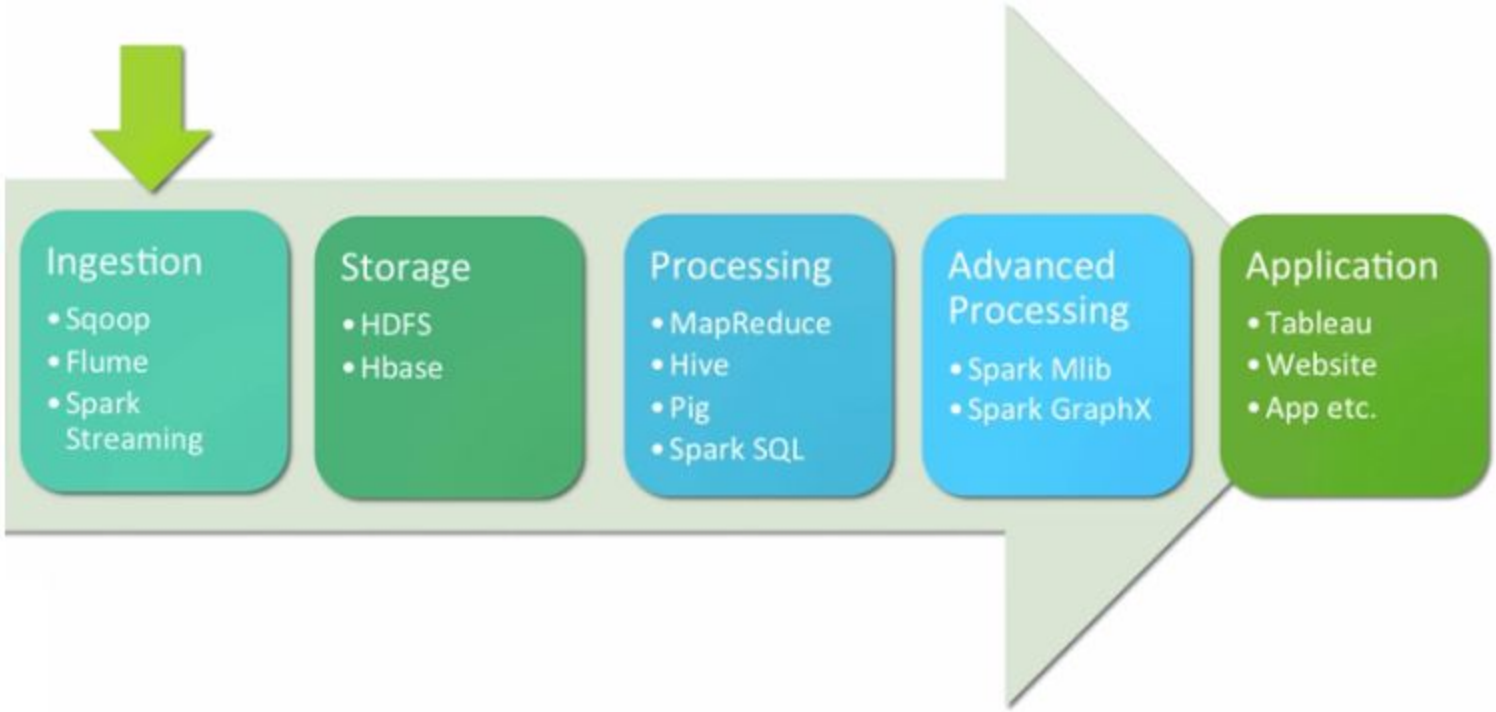
Apache Flume

https://flume.apache.org/

Big Data Project Stage



Introduction

- Flume is a distributed, reliable, and available service for efficiently collecting, aggregating, and moving large amounts of log data.
- It has a simple and flexible architecture based on streaming data flows.
- It is robust and fault tolerant with tunable reliability mechanisms and many failover and recovery mechanisms.
- It uses a simple extensible data model that allows for online analytic application.

Use of Flume

- The use of Apache Flume is not only restricted to log data aggregation.
- Since data sources are customizable, Flume can be used to transport massive quantities of event data including but not limited to network traffic data, social-media-generated data, email messages and pretty much any data source possible.

System Requirements

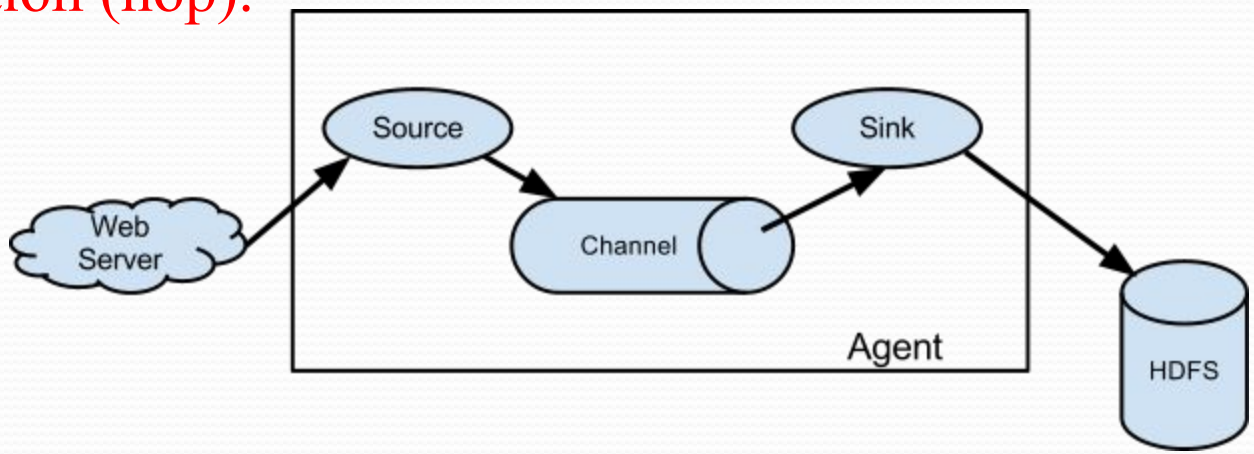
- Java Runtime Environment - Java 1.8 or later
- Memory - Sufficient memory for configurations used by sources, channels or sinks
- Disk Space - Sufficient disk space for configurations used by channels or sinks
- Directory Permissions - Read/Write permissions for directories used by agent

Flume Architecture

- **Data flow model**

A Flume event is defined as a unit of data flow having a byte payload and an optional set of string attributes.

A Flume agent is a (JVM) process that hosts the components through which events flow from an external source to the next destination (hop).



Flume Data flow

- A Flume source consumes events delivered to it by an external source like a web server.
- The external source sends events to Flume in a format that is recognized by the target Flume source.

For example, an Avro Flume source can be used to receive Avro events from Avro clients or other Flume agents in the flow that send events from an Avro sink.

- A similar flow can be defined using a Thrift Flume Source to receive events from a Thrift Sink or a Flume Thrift Rpc Client or Thrift clients written in any language generated from the Flume thrift protocol.

Flume Data flow

- When a Flume source receives an event, it stores it into one or more channels.
- The channel is a passive store that keeps the event until it's consumed by a Flume sink.
- The file channel is one example – it is backed by the local filesystem.
- The sink removes the event from the channel and puts it into an external repository like HDFS (via Flume HDFS sink) or forwards it to the Flume source of the next Flume agent (next hop) in the flow.
- The source and sink within the given agent run asynchronously with the events staged in the channel.

Complex flows

- Flume allows a user to build multi-hop flows where events travel through multiple agents before reaching the final destination. It also allows fan-in and fan-out flows, contextual routing and backup routes (fail-over) for failed hops.

Reliability

- The events are staged in a channel on each agent.
- The events are then delivered to the next agent or terminal repository (like HDFS) in the flow.
- The events are removed from a channel only after they are stored in the channel of next agent or in the terminal repository.
- This is a how the single-hop message delivery semantics in Flume provide end-to-end reliability of the flow.

Reliability

- Flume uses a transactional approach to guarantee the reliable delivery of the events.
- The sources and sinks encapsulate in a transaction the storage/retrieval, respectively, of the events placed in or provided by a transaction provided by the channel.
- This ensures that the set of events are reliably passed from point to point in the flow.
- In the case of a multi-hop flow, the sink from the previous hop and the source from the next hop both have their transactions running to ensure that the data is safely stored in the channel of the next hop.

Recoverability

- The events are staged in the channel, which manages recovery from failure.
- Flume supports a durable file channel which is backed by the local file system.
- There's also a memory channel which simply stores the events in an in-memory queue, which is faster but any events still left in the memory channel when an agent process dies can't be recovered.

Flume Setup

Setting up an agent

- Flume agent configuration is stored in a local configuration file.
- This is a text file that follows the Java properties file format.
- Configurations for one or more agents can be specified in the same configuration file.
- The configuration file includes properties of each source, sink and channel in an agent and how they are wired together to form data flows.

Configuring individual components

- Each component (source, sink or channel) in the flow has a name, type, and set of properties that are specific to the type and instantiation.
- For example, an Avro source needs a hostname (or IP address) and a port number to receive data from.
- A memory channel can have max queue size ("capacity"), and an HDFS sink needs to know the file system URI Uiform Ressourses Identifier, path to create files, frequency of file rotation ("hdfs.rollInterval") etc.
- All such attributes of a component needs to be set in the properties file of the hosting Flume agent.

Wiring the pieces together

- The agent needs to know what individual components to load and how they are connected in order to constitute the flow.
- This is done by listing the names of each of the sources, sinks and channels in the agent, and then specifying the connecting channel for each sink and source.

For example, an agent flows events from an Avro source called avroWeb to HDFS sink hdfs-cluster1 via a file channel called file-channel.

- The configuration file will contain names of these components and file-channel as a shared channel for both avroWeb source and hdfs-cluster1 sink.

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.

Example

- Here, we give an example configuration file, describing a single-node Flume deployment.
- This configuration lets a user generate events and subsequently logs them to the console.

```
# example.conf: A single-node Flume configuration

# Name the components on this agent
a1.sources = r1
a1.sinks = k1
a1.channels = c1

# Describe/configure the source
a1.sources.r1.type = netcat
a1.sources.r1.bind = localhost
a1.sources.r1.port = 44444

# Describe the sink
a1.sinks.k1.type = logger

# Use a channel which buffers events in memory
a1.channels.c1.type = memory
a1.channels.c1.capacity = 1000
a1.channels.c1.transactionCapacity = 100

# Bind the source and sink to the channel
a1.sources.r1.channels = c1
a1.sinks.k1.channel = c1
```

Example

- This configuration defines a single agent named a1. a1 has a source that listens for data on port 44444, a channel that buffers event data in memory, and a sink that logs event data to the console. The configuration file names the various components, then describes their types and configuration parameters. A given configuration file might define several named agents; when a given Flume process is launched a flag is passed telling it which named agent to manifest.
- Given this configuration file, we can start Flume as follows:

```
$ bin/flume-ng agent --conf conf --conf-file example.conf --name a1 -Dflume.root.logger=INFO,console
```

Example

- Note that in a full deployment we would typically include one more option: --conf=<conf-dir>.
  The <conf-dir> directory would include a shell script *flume-env.sh*and potentially a log4j properties file.
  In this example, we pass a Java option to force Flume to log to the console and we go without a custom environment script.
- From a separate terminal, we can then telnet port 44444 and send Flume an event:

```
$ telnet localhost 44444
Trying 127.0.0.1...
Connected to localhost.localdomain (127.0.0.1).
Escape character is '^]'.
Hello world! <ENTER>
OK
```

Example

- The original Flume terminal will output the event in a log message.

```
12/06/19 15:32:19 INFO source.NetcatSource: Source starting
12/06/19 15:32:19 INFO source.NetcatSource: Created serverSocket:sun.nio.ch.ServerSocketChannelImpl[/127.0.0.1:44444]
12/06/19 15:32:34 INFO sink.LoggerSink: Event: { headers:{} body: 48 65 6C 6C 6F 20 77 6F 72 6C 64 21 0D         Hello world!. }
```

- Congratulations - you've successfully configured and deployed a Flume agent! Subsequent sections cover agent configuration in much more detail.

Zookeeper based Configuration

- Flume supports Agent configurations via Zookeeper.

……This is an experimental feature……

- The configuration file needs to be uploaded in the Zookeeper, under a configurable prefix.
- The configuration file is stored in Zookeeper Node data.
- Following is how the Zookeeper Node tree would look like for agents a1 and a2

```
- /flume
 |- /a1 [Agent config file]
 |- /a2 [Agent config file]
```

Zookeeper based Configuration

- Once the configuration file is uploaded, start the agent with following options
- $ bin/flume-ng agent –conf conf -z zkhost:2181,zkhost1:2181 -p /flume –name a1 -Dflume.root.logger=INFO,console

Argument Name	Default	Description
z	–	Zookeeper connection string. Comma separated list of hostname:port
p	/flume	Base Path in Zookeeper to store Agent configurations

Data ingestion

- Flume supports a number of mechanisms to ingest data from external sources.

1. RPC

An Avro client included in the Flume distribution can send a given file to Flume Avro source using avro RPC mechanism:

$ bin/flume-ng avro-client -H localhost -p 41414 -F /usr/logs/log.10

The above command will send the contents of /usr/logs/log.10 to to the Flume source listening on that ports.

2. Executing commands

There's an exec source that executes a given command and consumes the output. A single 'line' of output ie. text followed by carriage return ('\r') or line feed ('\n') or both together.

3. Network streams

Flume supports the following mechanisms to read data from popular log stream types, such as:

- Avro
- Thrift
- Syslog
- Netcat

Flume Sources

- **Avro Source**
- Listens on Avro port and receives events from external Avro client streams. When paired with the built-in Avro Sink on another (previous hop) Flume agent, it can create tiered collection topologies. Required properties are in **bold**.

Flume Sources

Property Name	Default	Description
channels	–	
type	–	The component type name, needs to be `avro`
bind	–	hostname or IP address to listen on
port	–	Port # to bind to
threads	–	Maximum number of worker threads to spawn
selector.type		
selector.*		
interceptors	–	Space-separated list of interceptors
interceptors.*		
compression-type	none	This can be "none" or "deflate". The compression-type must match the compression-type of matching AvroSource
ssl	false	Set this to true to enable SSL encryption. You must also specify a "keystore" and a "keystore-password".
keystore	–	This is the path to a Java keystore file. Required for SSL.
keystore-password	–	The password for the Java keystore. Required for SSL.
keystore-type	JKS	The type of the Java keystore. This can be "JKS" or "PKCS12".
exclude-protocols	SSLv3	Space-separated list of SSL/TLS protocols to exclude. SSLv3 will always be excluded in addition to the protocols specified.
ipFilter	false	Set this to true to enable ipFiltering for netty
ipFilterRules	–	Define N netty ipFilter pattern rules with this config.

Example for agent named a1:

```
a1.sources = r1
a1.channels = c1
a1.sources.r1.type = avro
a1.sources.r1.channels = c1
a1.sources.r1.bind = 0.0.0.0
a1.sources.r1.port = 4141
```

Example of ipFilterRules

- ipFilterRules defines N netty ipFilters separated by a comma a pattern rule must be in this format.
- <'allow' or deny>:<'ip' or 'name' for computer name>:<pattern> or allow/deny:ip/name:pattern
- example: ipFilterRules=allow:ip:127.*,allow:name:localhost,deny:ip:*
- Note that the first rule to match will apply as the example below shows from a client on the localhost
- This will Allow the client on localhost be deny clients from any other ip "allow:name:localhost,deny:ip:" *This will deny the client on localhost be allow clients from any other ip "deny:name:localhost,allow:ip:"*

Thrift Source

- Listens on Thrift port and receives events from external Thrift client streams. When paired with the built-in ThriftSink on another (previous hop) Flume agent, it can create tiered collection topologies. Thrift source can be configured to start in secure mode by enabling kerberos authentication. agent-principal and agent-keytab are the properties used by the Thrift source to authenticate to the kerberos KDC. Required properties are in **bold**.

Property Name	Default	Description
channels	–	
type	–	The component type name, needs to be `thrift`
bind	–	hostname or IP address to listen on
port	–	Port # to bind to
threads	–	Maximum number of worker threads to spawn
selector.type		
selector.*		
interceptors	–	Space separated list of interceptors
interceptors.*		
ssl	false	Set this to true to enable SSL encryption. You must also specify a "keystore" and a "keystore-password".
keystore	–	This is the path to a Java keystore file. Required for SSL.
keystore-password	–	The password for the Java keystore. Required for SSL.
keystore-type	JKS	The type of the Java keystore. This can be "JKS" or "PKCS12".
exclude-protocols	SSLv3	Space-separated list of SSL/TLS protocols to exclude. SSLv3 will always be excluded in addition to the protocols specified.
kerberos	false	Set to true to enable kerberos authentication. In kerberos mode, agent-principal and agent-keytab are required for successful authentication. The Thrift source in secure mode, will accept connections only from Thrift clients that have kerberos enabled and are successfully authenticated to the kerberos KDC.
agent-principal	–	The kerberos principal used by the Thrift Source to authenticate to the kerberos KDC.
agent-keytab	—-	The keytab location used by the Thrift Source in combination with the agent-principal to authenticate to the kerberos KDC.

Thrift Source

- **Example for agent named a1:**

```
a1.sources = r1
a1.channels = c1
a1.sources.r1.type = thrift
a1.sources.r1.channels = c1
a1.sources.r1.bind = 0.0.0.0
a1.sources.r1.port = 4141
```

Exec Source

- Exec source runs a given Unix command on start-up and expects that process to continuously produce data on standard out (stderr is simply discarded, unless property logStdErr is set to true). If the process exits for any reason, the source also exits and will produce no further data. This means configurations such
  as cat [named pipe]or tail -F [file] are going to produce the desired results where as date will probably not - the former two commands produce streams of data where as the latter produces a single event and exits.

JMS Source

- JMS Source reads messages from a JMS destination such as a queue or topic. Being a JMS application it should work with any JMS provider but has only been tested with ActiveMQ. The JMS source provides configurable batch size, message selector, user/pass, and message to flume event converter. Note that the vendor provided JMS jars should be included in the Flume classpath using plugins.d directory (preferred), –classpath on command line, or via FLUME_CLASSPATH variable in flume-env.sh.

Converter

- The JMS source allows pluggable converters, though it's likely the default converter will work for most purposes.
- The default converter is able to convert Bytes, Text, and Object messages to FlumeEvents. In all cases, the properties in the message are added as headers to the FlumeEvent.
- BytesMessage:

  Bytes of message are copied to body of the FlumeEvent. Cannot convert more than 2GB of data per message.
- TextMessage:

  Text of message is converted to a byte array and copied to the body of the FlumeEvent. The default converter uses UTF-8 by default but this is configurable.
- ObjectMessage:

  Object is written out to a ByteArrayOutputStream wrapped in an ObjectOutputStream and the resulting array is copied to the body of the FlumeEvent.

Spooling Directory Source

- This source lets you ingest data by placing files to be ingested into a “spooling” directory on disk. This source will watch the specified directory for new files, and will parse events out of new files as they appear. The event parsing logic is pluggable.
- After a given file has been fully read into the channel, it is renamed to indicate completion (or optionally deleted).
- Unlike the Exec source, this source is reliable and will not miss data, even if Flume is restarted or killed.
- In exchange for this reliability, only immutable, uniquely-named files must be dropped into the spooling directory. Flume tries to detect these problem conditions and will fail loudly if they are violated:

1. If a file is written to after being placed into the spooling directory, Flume will print an error to its log file and stop processing.
2. If a file name is reused at a later time, Flume will print an error to its log file and stop processing.

- To avoid the above issues, it may be useful to add a unique identifier (such as a timestamp) to log file names when they are moved into the spooling directory.
- Despite the reliability guarantees of this source, there are still cases in which events may be duplicated if certain downstream failures occur.
- This is consistent with the guarantees offered by other Flume components.

Kafka Source

- Kafka Source is an Apache Kafka consumer that reads messages from Kafka topics. If you have multiple Kafka sources running, you can configure them with the same Consumer Group so each will read a unique set of partitions for the topics.

Security and Kafka Source:

- Secure authentication as well as data encryption is supported on the communication channel between Flume and Kafka. For secure authentication SASL/GSSAPI (Kerberos V5) or SSL (even though the parameter is named SSL, the actual protocol is a TLS implementation) can be used from Kafka version 0.9.0.
- As of now data encryption is solely provided by SSL/TLS.
- Setting kafka.consumer.security.protocol to any of the following value means:
- **SASL_PLAINTEXT** - Kerberos or plaintext authentication with no data encryption
- **SASL_SSL** - Kerberos or plaintext authentication with data encryption
- **SSL** - TLS based encryption with optional authentication.

Other Flume sources

- NetCat TCP Source
- Sequence Generator Source
- Syslog Sources
- Syslog TCP Source
- HTTP Source
- NetCat UDP Source
- JSONHandler
- Legacy Sources
- Custom Source
- Scribe Source

Flume Sinks

HDFS Sink

- This sink writes events into the Hadoop Distributed File System (HDFS). It currently supports creating text and sequence files. It supports compression in both file types. The files can be rolled (close current file and create a new one) periodically based on the elapsed time or size of data or number of events. It also buckets/partitions data by attributes like timestamp or machine where the event originated. The HDFS directory path may contain formatting escape sequences that will replaced by the HDFS sink to generate a directory/file name to store the events. Using this sink requires hadoop to be installed so that Flume can use the Hadoop jars to communicate with the HDFS cluster. Note that a version of Hadoop that supports the sync() call is required.

Flume Sinks

HDFS Sink

Alias	Description
%{host}	Substitute value of event header named "host". Arbitrary header names are supported.
%t	Unix time in milliseconds
%a	locale's short weekday name (Mon, Tue, ...)
%A	locale's full weekday name (Monday, Tuesday, ...)
%b	locale's short month name (Jan, Feb, ...)
%B	locale's long month name (January, February, ...)
%c	locale's date and time (Thu Mar 3 23:05:25 2005)
%d	day of month (01)
%e	day of month without padding (1)
%D	date; same as %m/%d/%y
%H	hour (00..23)
%I	hour (01..12)
%j	day of year (001..366)
%k	hour (0..23)
%m	month (01..12)
%n	month without padding (1..12)
%M	minute (00..59)
%p	locale's equivalent of am or pm
%s	seconds since 1970-01-01 00:00:00 UTC
%S	second (00..59)
%y	last two digits of year (00..99)
%Y	year (2010)
%z	+hhmm numeric timezone (for example, -0400)

Flume Sinks

Hive Sink

This sink streams events containing delimited text or JSON data directly into a Hive table or partition. Events are written using Hive transactions. As soon as a set of events are committed to Hive, they become immediately visible to Hive queries. Partitions to which flume will stream to can either be pre-created or, optionally, Flume can create them if they are missing. Fields from incoming event data are mapped to corresponding columns in the Hive table.

Flume Sinks

Logger Sink

Logs event at INFO level. Typically useful for testing/debugging purpose. Required properties are in **bold**. This sink is the only exception which doesn't require the extra configuration explained in the Logging raw data section.

Property Name	Default	Description
channel	–	
type	–	The component type name, needs to be `logger`
maxBytesToLog	16	Maximum number of bytes of the Event body to log

Example for agent named a1:

```
a1.channels = c1
a1.sinks = k1
a1.sinks.k1.type = logger
a1.sinks.k1.channel = c1
```

Flume Sinks

Avro Sink

- This sink forms one half of Flume's tiered collection support. Flume events sent to this sink are turned into Avro events and sent to the configured hostname / port pair. The events are taken from the configured Channel in batches of the configured batch size. Required properties are in **bold**.

Flume Sinks

Thrift Sink

- This sink forms one half of Flume's tiered collection support. Flume events sent to this sink are turned into Thrift events and sent to the configured hostname / port pair. The events are taken from the configured Channel in batches of the configured batch size.
- Thrift sink can be configured to start in secure mode by enabling kerberos authentication. To communicate with a Thrift source started in secure mode, the Thrift sink should also operate in secure mode. client-principal and client-keytab are the properties used by the Thrift sink to authenticate to the kerberos KDC. The server-principal represents the principal of the Thrift source this sink is configured to connect to in secure mode

Flume Sinks

Other Flume Sinks

- IRC Sink
- File Roll Sink
- Null Sink
- HBaseSinks
  - HBaseSink
  - AsyncHBaseSink
- MorphlineSolrSink
- ElasticSearchSink
- Kite Dataset Sink
- Kafka Sink
- HTTP Sink
- Custom Sink

Flume Channels

Channels are the repositories where the events are staged on a agent. Source adds the events and Sink removes it.

Flume Channels

- The events are stored in an in-memory queue with configurable max size. It's ideal for flows that need higher throughput and are prepared to lose the staged data in the event of a agent failures.

Flume Channels

JDBC Channel

- The events are stored in a persistent storage that's backed by a database. The JDBC channel currently supports embedded Derby. This is a durable channel that's ideal for flows where recoverability is important.

Flume Channels

Kafka Channel

- The events are stored in a Kafka cluster (must be installed separately). Kafka provides high availability and replication, so in case an agent or a kafka broker crashes, the events are immediately available to other sinks
- The Kafka channel can be used for multiple scenarios:
- With Flume source and sink - it provides a reliable and highly available channel for events
- With Flume source and interceptor but no sink - it allows writing Flume events into a Kafka topic, for use by other apps
- With Flume sink, but no source - it is a low-latency, fault tolerant way to send events from Kafka to Flume sinks such as HDFS, HBase or Solr

Flume Channels

- **Other Channels**
  - File Channel
  - Spillable Memory Channel
  - Pseudo Transaction Channel
  - Flume Channel Selectors

Flume Sink Processors

- Default Sink Processor
- Default sink processor accepts only a single sink. User is not forced to create processor (sink group) for single sinks. Instead user can follow the source - channel - sink pattern that was explained above in this user guide.

But

- Sink groups allow users to group multiple sinks into one entity. Sink processors can be used to provide load balancing capabilities over all sinks inside the group or to achieve fail over from one sink to another in case of temporal failure.