**Big Data Pipeline Installation Guide**

Stephen Blystone

Document Version 1.0

Last Updated: Mar 27, 2018

Table of Contents

[**Revision History** 3](#_Toc510306837)

[**Big Data Pipeline Architecture** 3](#_Toc510306838)

[**Pre-Requisites** 4](#_Toc510306839)

[**~~Installing StreamSets~~** 4](#_Toc510306840)

[**Installing IPFIXCol collector** 5](#_Toc510306841)

[**Configuring IPFIXCol** 6](#_Toc510306842)

[**Testing IPFIXCol** 7](#_Toc510306843)

[**Running IPFIXCol** 8](#_Toc510306844)

[**Installing Apache Flume** 8](#_Toc510306845)

[**Installing Apache Spark** 10](#_Toc510306846)

[**Testing Apache Spark Streaming** 11](#_Toc510306847)

[**Setting up Apache Spark Streaming** 11](#_Toc510306848)

[**Setting up Apache Spark SQL** 11](#_Toc510306849)

[**Streaming Applications** 12](#_Toc510306850)

# **Revision History**

|  |  |  |
| --- | --- | --- |
| **Version** | **Date** | **Changes** |
| 1.0 | 3/27/2018 | Initial Version |

# **Big Data Pipeline Architecture**

Below is a basic Big Data architecture for real-time processing from the Microsoft website. We are taking inspiration from there.

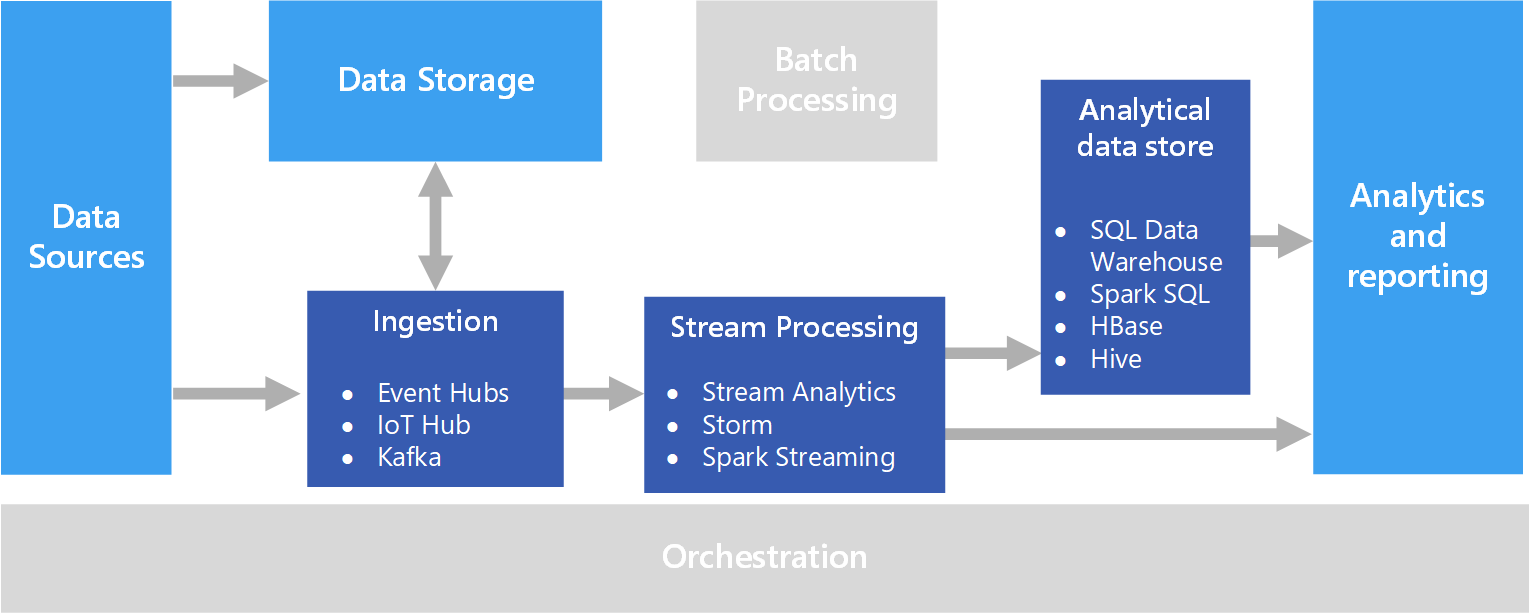


Figure - https://docs.microsoft.com/en-us/azure/architecture/data-guide/scenarios/real-time-processing

We will have several components in our architecture:

1. Data Sources
   1. These are the Open vSwitch nodes sending either sFlow, NetFlow, or IPFIX data (Still to be determined).
2. Real-time Message Ingestion
   1. This component essentially performs Stream Buffering, providing the “Stream Processing” block enough time to process the stream data without being overrun with data.
   2. We will use Apache Flume. The diagram mentions Kafka because it is better for mission-critical architectures. However, Apache Flume is easier to set up.
3. Stream Processing
   1. This component filters, aggregates, and prepares the data for analysis.
   2. We will use Apache Spark Streaming.
4. Analytical Data Store
   1. This is where the data is stored after being processed.
   2. We will use Spark SQL, since we are already using Apache Spark.
5. Analytics and Reporting
   1. We will not use this part.
6. Machine Learning
   1. Not pictured in the above diagram.
   2. We will use the data in the Analytical data store to perform Machine Learning.

# **Pre-Requisites**

1. Install Java 8 onto Application VM
   1. Java8 commands:

sudo apt-get install software-properties-common -y && \

sudo add-apt-repository ppa:webupd8team/java -y && \

sudo apt-get update && \

echo "oracle-java8-installer shared/accepted-oracle-license-v1-1 select true" | sudo debconf-set-selections && \

sudo apt-get install oracle-java8-installer oracle-java8-set-default -y

1. Add JAVA\_HOME to your .profile file and to PATH

vi ~/.profile

JAVA\_HOME=/usr/lib/jvm/java-8-oracle

PATH=$PATH:$JAVA\_HOME

1. Log out and back in to enable the changes.

# **~~Installing StreamSets~~**

1. ~~To simplify the flow between the different products, we can use Streamsets.~~ [~~https://streamsets.com/opensource/~~](https://streamsets.com/opensource/)
2. ~~Download the file:~~

~~wget~~ [~~https://archives.streamsets.com/datacollector/3.1.2.0/tarball/streamsets-datacollector-core-3.1.2.0.tgz~~](https://archives.streamsets.com/datacollector/3.1.2.0/tarball/streamsets-datacollector-core-3.1.2.0.tgz)

~~tar -xvzf streamsets-datacollector-core-3.1.2.0.tgz~~

1. ~~Increase the limit of open file descriptors.~~
   1. ~~sudo vi /etc/security/limits.conf~~
   2. ~~At the end of the file (above where it says # End of file) put the following entry:~~

~~\* soft nofile 32768~~

~~\* hard nofile 32768~~

1. ~~Log out and back in for this to take effect.~~
2. ~~You can verify this worked by running “ulimit -n” and you should see 32768~~
3. ~~Run the Data Collector (from the directory you downloaded it in):~~

~~./streamsets-datacollector-3.1.2.0/bin/streamsets dc~~

1. ~~Go to http://<system-ip>:18630/~~
   1. ~~Default username and password are “admin” and “admin”~~

# **~~Installing IPFIXCol collector~~**

1. ~~We need to convert the incoming data to a form that Apache Flume will recognize.~~
2. ~~I found~~ [~~https://stream4flow.ics.muni.cz/#architecture~~](https://stream4flow.ics.muni.cz/#architecture) ~~that uses IPFIXCol collector to convert the IPFIX records into JSON format.~~
3. ~~Apache Flume recognizes JSON format.~~
4. ~~IPFIXCol is located at~~ [~~https://github.com/CESNET/ipfixcol~~](https://github.com/CESNET/ipfixcol)
5. ~~IPFIXCol is released under a GPL v2.0 license~~
   1. [~~https://github.com/CESNET/ipfixcol/blob/master/LICENSE~~](https://github.com/CESNET/ipfixcol/blob/master/LICENSE)
   2. [~~https://choosealicense.com/licenses/gpl-2.0/~~](https://choosealicense.com/licenses/gpl-2.0/)
6. ~~Install the pre-requisites:~~

~~sudo apt-get install autoconf bison build-essential docbook-xsl doxygen flex git   
liblzo2-dev libbz2-dev libtool libsctp-dev libssl-dev libxml2 libxml2-dev   
pkg-config xsltproc libgeoip-dev librrd-dev libsqlite3-dev libpq-dev libcpg-dev   
corosync-dev~~

1. ~~Download git repository for libfastbit (Pre-requisite for IPFIXCol)~~

~~cd ~~~

~~git clone --recursive https://github.com/CESNET/libfastbit.git~~

1. ~~CD into libfastbit directory~~

~~cd ~/libfastbit~~

1. ~~Switch to root user~~

~~sudo su~~

1. ~~Build and install~~

~~autoreconf -i~~

~~./configure~~

~~make~~

~~NOTE: This command takes a long time to run.~~

~~make install~~

1. ~~Download git repository for IPFIXCol (do NOT forget --recursive option)~~

~~cd ~~~

~~git clone --recursive https://github.com/CESNET/ipfixcol.git~~

1. ~~CD into new directory~~

~~cd ~/ipfixcol~~

1. ~~Build the framework.~~

~~autoreconf -i~~

1. ~~Generate configure script from configure.ac, Makefile.in from Makefile.am and install missing files.~~

~~./configure~~

1. ~~Configure packages in subdirectories and generate Makefiles~~

~~make~~

~~make install~~

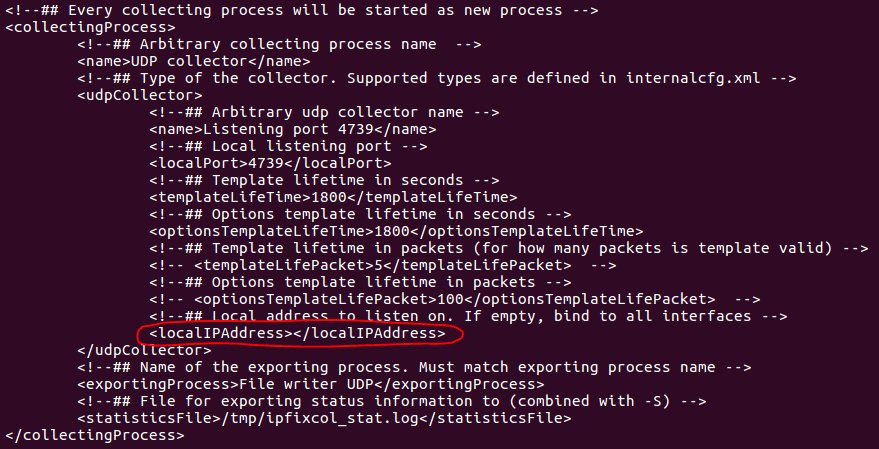
1. ~~All files are installed in the /usr/local/ subdirectories~~
   1. ~~The ipfixcol application is in /usr/local/bin/~~
   2. ~~The configuration XML files are in /usr/local/etc/ipfixcol/~~

# **~~Configuring IPFIXCol~~**

1. ~~Edit UDP Collector local IP Address~~

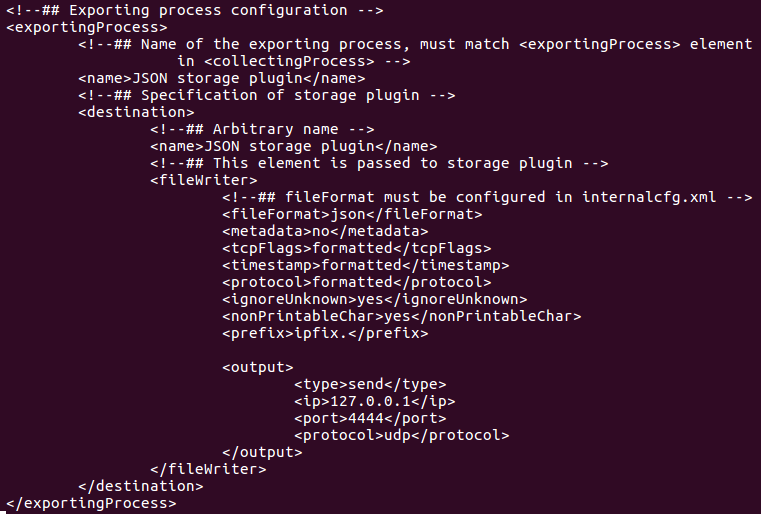
~~sudo vi /usr/local/etc/ipfixcol/startup.xml~~

1. ~~Delete 127.0.0.1 so that it leaves <localIPAddress></localIPAddress>~~

~~~~

1. ~~Also change the <exportingProcess> section of startup.xml to match the following:~~

[~~https://github.com/CESNET/ipfixcol/tree/master/plugins/storage/json~~](https://github.com/CESNET/ipfixcol/tree/master/plugins/storage/json)

~~~~

1. ~~Save and exit from vi.~~

~~[Esc]~~

~~:x~~

# **~~Testing IPFIXCol~~**

1. ~~In ONOS VM:~~
   1. ~~Launch ONOS.~~

~~onos-buck run onos-local~~

1. ~~In Mininet VM:~~
   1. ~~Launch Mininet connecting to ONOS Controller.~~

~~sudo mn --controller=remote,ip=10.28.34.39,port=6633~~

* 1. ~~Configure Open vSwitch to use IPFIX to send to IPFIXCol VM and port.~~

~~sudo ovs-vsctl set bridge s1 ipfix=@i -- --id=@i create IPFIX targets=\”10.28.34.16:4739\” obs\_domain\_id=123 obs\_point\_id=456 cache\_active\_timeout=60 cache\_max\_flows=13 sampling=2   
other\_config:enable-input-sampling=true   
other\_config:enable-tunnel-sampling=true~~

1. ~~In Application VM:~~
   1. ~~Launch IPFIXCol.~~

~~ipfixcol~~

* 1. ~~Use netcat to create a UDP server on port 4444 to listen.~~

~~nc -lu 4444~~

1. ~~In Mininet VM:~~
   1. ~~Send pings~~

~~h1 ping h2~~

1. ~~You should see JSON data show up in your netcat output.~~

# **~~Running IPFIXCol~~**

1. [~~https://github.com/CESNET/ipfixcol/tree/master/base#howrun~~](https://github.com/CESNET/ipfixcol/tree/master/base#howrun)
2. ~~To run:~~

~~ipfixcol~~

# **Installing Apache Flume**

1. Download the Apache Flume binary from <https://flume.apache.org/download.html>. Below is the mirror it gave me:

wget <http://mirror.cogentco.com/pub/apache/flume/1.8.0/apache-flume-1.8.0-bin.tar.gz>

tar -xzvf apache-flume-1.8.0-bin.tar.gz

cd ./apache-flume-1.8.0-bin

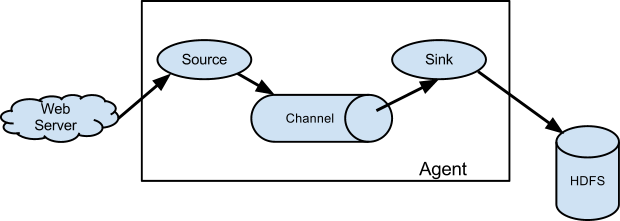
1. User Guide is found at <https://flume.apache.org/FlumeUserGuide.html>
2. Flume Definitions:  
   

Figure - https://flume.apache.org/FlumeUserGuide.html

* 1. **Flume Source** = consumes events delivered to it by an external source (like web server)
  2. **External Source** = sends events to Flume in format that is recognized by target Flume source.
  3. **Channel** = Passive store that keeps the event until it is consumed by a Flume sink. When Flume receives an event, it stores it into one or more channels.
  4. **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.

1. Setup an Agent
   1. Flume agent configuration is stored in a local configuration file. Configurations for one or more agents can be specified in the same config file. It contains properties for each source, sink, and channel in an agent and how they are wired together to form data flows.
   2. Each component (source, sink, or channel) in the flow has:
      1. Name
      2. Type
      3. Set of properties that are specific to the type and instantiation
         1. Example: hostname/IP address and port number
   3. A memory channel can have a max queue size.
   4. An HDFS sink needs to know the file system URI, path to create files, and frequency of file rotation.
   5. Wiring the pieces together involves listing the names of each of the sources, sinks, and channels in an agent and specifying the connecting channel for each sink and source.
2. Starting an agent
   1. Below is the sample format of the command:

./bin/flume-ng agent -n $agent\_name -c conf -f ./conf/flume-conf.properties.template

You must specify the agent name and the config directory and filename.

* 1. Below is an example of the command running.

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

* 1. Instead of specifying the logger in command-line, in a full deployment you can also include “--conf=<conf-dir>” where “<conf-dir>” is a directory that includes   
     *flume-env.sh* and potentially a log4j properties file.

1. Using environment variables in configuration files.
   1. Examples:

a1.sources = r1

a1.sources.r1.type = netcat

a1.sources.r1.bind = 0.0.0.0

a1.sources.r1.port = ${NC\_PORT}

a1.sources.r1.channels = c1

1. For troubleshooting issues, you can log the raw data.
   1. To enable configuration-related logging, you need to pass the argument:  
      -Dorg.apache.flume.log.printconfig=true
   2. To enable data logging, you need to pass the argument:  
      -Dorg.apache.flume.log.rawdata=true
   3. The log4j logging level must also be set to DEBUG or TRACE.
   4. Example of the full command:

./bin/flume-ng agent --conf conf --conf-file example.conf --name a1   
-Dflume.root.logger=DEBUG,console   
-Dorg.apache.flume.log.printconfig=true   
-Dorg.apache.flume.log.rawdata=true

1. Configuration for our Flume:
   1. Source is JSON data from IPFIXCol
      1. We are using HTTP Source with a BlobHandler.
         1. From <https://blog.datatons.com/2016/12/16/ingest-search-json-events-in-real-time-i-collecting-data/> it recommends BlobHandler over JSONHandler. JSONHandler requires a very specific structure.
   2. Channel is memoryChannel
   3. Sink is avroSink
   4. Our configuration:
      1. <https://spark.apache.org/docs/2.2.0/streaming-flume-integration.html>
      2. <https://flume.apache.org/FlumeUserGuide.html>

agent.sources = jsonSrc

agent.channels = memChannel

agent.sinks = avroSink

agent.sources.jsonSrc.type = http

agent.sources.jsonSrc.port = 4444

agent.sources.jsonSrc.channels = memChannel

agent.sources.jsonSrc.handler = org.apache.flume.sink.solr.morphline.BlobHandler

agent.channels.memChannel.type = memory

agent.channels.memChannel.capacity = 1000

agent.channels.memChannel.transactionCapacity = 100

agent.sinks.avroSink.type = avro

agent.sinks.avroSink.channel = memChannel

agent.sinks.avroSink.hostname = localhost

agent.sinks.avroSink.port = 9876

1. Launching our Flume configuration

./bin/flume-ng agent -n agent -c conf -f ./conf/flume-conf.properties

. is the directory you downloaded flume into.

./conf/flume-conf.properties is the properties file we placed our configuration above.

# **Installing Apache Spark**

1. Download Apache Spark 2.3.0 to your VM. To choose your version you can go to <https://spark.apache.org/downloads.html>. Below is the mirror it gave me:

wget <http://mirrors.ocf.berkeley.edu/apache/spark/spark-2.3.0/spark-2.3.0-bin-hadoop2.7.tgz>

tar -xzvf spark-2.3.0-bin-hadoop2.7.tgz

cd ./spark-2.3.0-bin-hadoop2.7/

1. To launch standalone Master Server:

./sbin/start-master.sh

**NOTE**: You can view the log that is written to check if there are any errors. The log file is displayed when you run the above command.

* 1. Launch the Spark Master web GUI on 10.28.34.14:8080
     1. IP address is the address of the VM.
     2. Record the “URL: value” for launching workers.
        1. 
        2. My VM name is “Application”. Yours may be different.

1. Launch a Worker by running the following command:

./sbin/start-slave.sh spark://Application:7077

The highlighted portion is what you recorded above on the Spark Master web GUI.

**NOTE**: You can view the log that is written to check if there are any errors. The log file is displayed when you run the above command.

1. For more details: <https://spark.apache.org/docs/latest/spark-standalone.html>

# **Testing Apache Spark Streaming**

1. <https://spark.apache.org/docs/2.3.0/streaming-programming-guide.html#a-quick-example>
2. <https://stackoverflow.com/questions/31579204/spark-streaming-example-not-working-for-me-network-word-count-maybe-data-not>
3. Before you run the example on the streaming-programming-guide (in 1 above), you need to do the following steps:
   1. Go to the “conf” directory and run the following command:

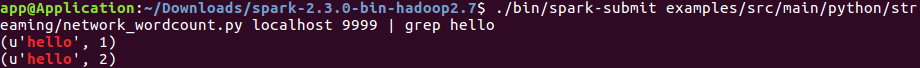
cp spark-env.sh.template spark-env.sh

* 1. Vi into spark-env.sh and add “MASTER=local[2]” in that file.
  2. Now you can run the example and it will work correctly.

1. NOTE: A lot of logging messages appear in the output. To quickly verify the example was working, I added “| grep hello” at the end of the command to launch the network\_wordcount.py in Terminal 2. Then in my Terminal 1 (nc) I typed “hello” and “hello world hello” to verify I was seeing counts in my Terminal 2.
   1. Terminal 1



* 1. Terminal 2



# **Setting up Apache Spark Streaming**

Asdf

# **Setting up Apache Spark SQL**

Asdf

# **Streaming Applications**

1. <https://spark.apache.org/docs/latest/streaming-programming-guide.html>
2. Asdf

./bin/spark-submit --packages org.apache.spark:spark-streaming-flume\_2.11:2.3.0 python/sparkIPFIX.py