Exercise 2

Configuring Flume for Data Loading

Contents

Lab 2 Configuring Flume for Data Loading

This exercise introduces how to configure Flume agents in order to load data into Hadoop.

After completing this hands-on lab, you'll be able to:

- Configure Flume agents for loading data into Hadoop

Allow 30 minutes to complete this lab.

This version of the lab was updated and tested on the InfoSphere BigInsights 4.1 Quick Start Edition. Throughout this lab you will be using the following account login information. If your passwords are different, please note the difference.

	Username	Password
VM image setup screen	root	password
Ambari	admin	admin

Flume is a distributed service for efficiently moving the large amounts of data. The original use case of Flume was to gather a set of log files on various machines in a cluster and aggregate them to a centralized persistent store such as Hadoop Distributed File System (HDFS). It has since been re-architected and expanded to cover a much wider variety of data.

Probably the most important ingredient for this exercise is your imagination. You are going to have to draw upon the inner child within you. Your exercises are running on a BigInsights cluster that has a whopping one node. So it should be obvious that you are not going to be able to move data from one node to another. For this exercise to be somewhat meaningful, you are going to have to image that, when you start multiple agents, you have multiple nodes and that Flume is running on each of those nodes.

Note:

Be aware that the initialization of the flume agents will take several minutes. Please be patient.

1.1 Getting Started

It is assumed that you have downloaded the VM image (BigInsights QuickStart 4.1) and completed the setup and configuration for VM Workstation 12 Player.

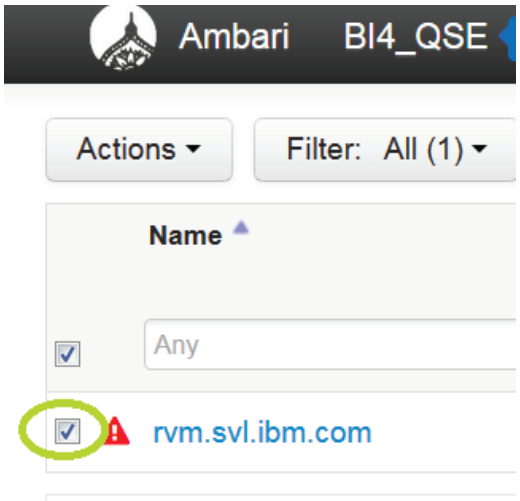
First, you want to start the BigInsights components.

__1. Open a Web browser and navigate to **http://rvm.svl.ibm.com:8080**, and sign in using the Ambari user id and password specified at the beginning of this document.

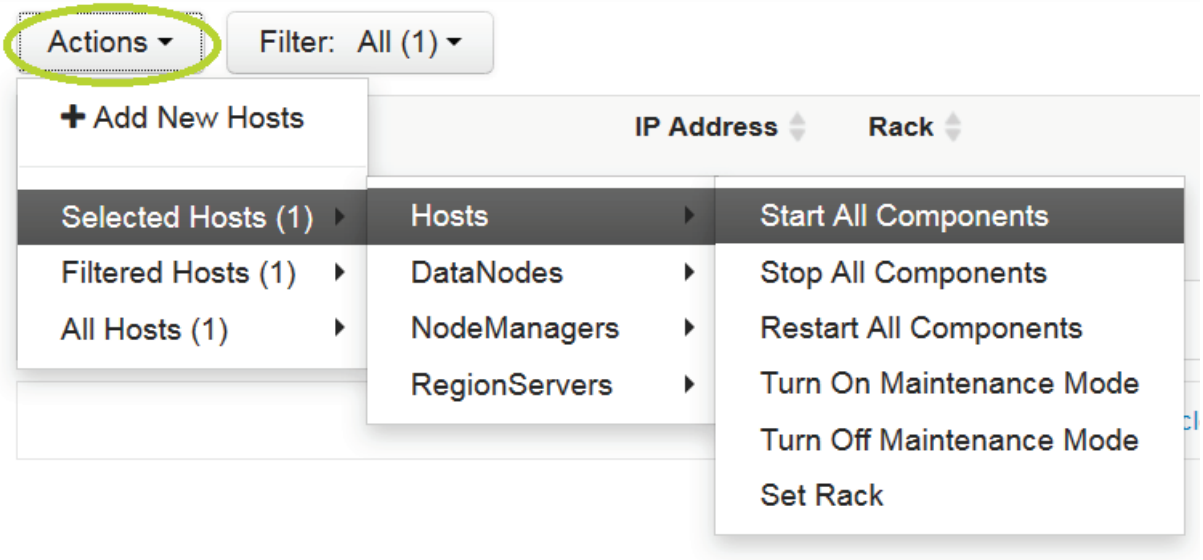
Notice that most of the BigInsights components listed at the left are in a Stopped state as indicated by the red, triangular warning icon.

__2. Click on **Hosts** in the top of web page.

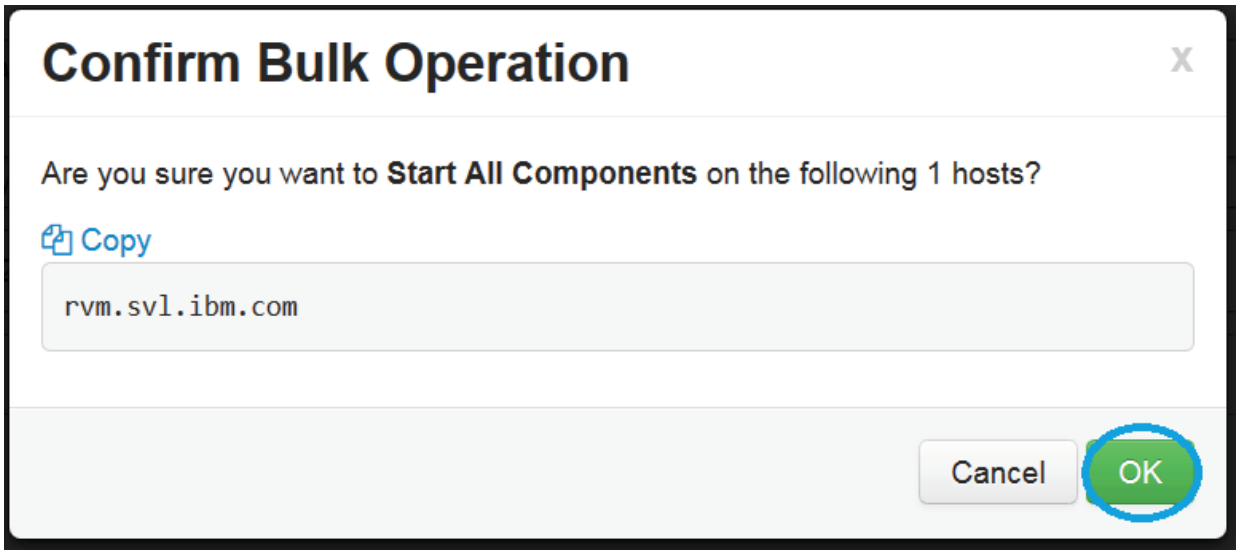
__3. Check the **box** next to your host. *"rvm.svl.ibm.com"*



__4. Click Actions -> Selected Hosts (1) -> Hosts -> Start All Components



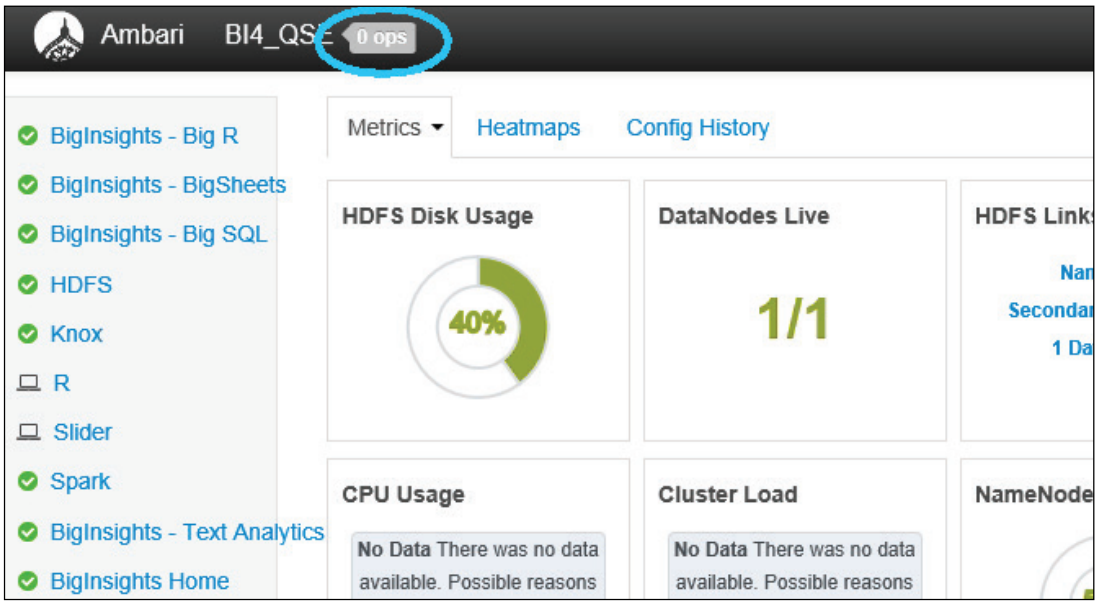
__5. Confirm by hitting **OK** when *"Confirm Bulk Operations"* pops up.



Clicking *Stop All* would stop the components in a similar manner.

Note: Be sure to allow ample time for all the components to start. The first time you start the components and services, it may take approximately 30 minutes or even longer, depending on the physical resources on your machine.

__6. Periodically examine the background operations indicator at the top of the screen. It should show that 1 operation is running. When it updates to 0 ops as shown below, the Start All script is complete and the components should all be running as indicated by the green check mark icons on the left.



Note: If your icons still show red warning signs after the startup, it may be that the Ambari interface did not refresh properly, even though the details in the background operations show 100% and display a successful message. Feel free to click the Admin button at the top right of the window, and then click Sign out. You will be presented with the Ambari login screen. Log back in using the credentials at the beginning of this document and the component list should be updated with the correct, green check-mark icons.

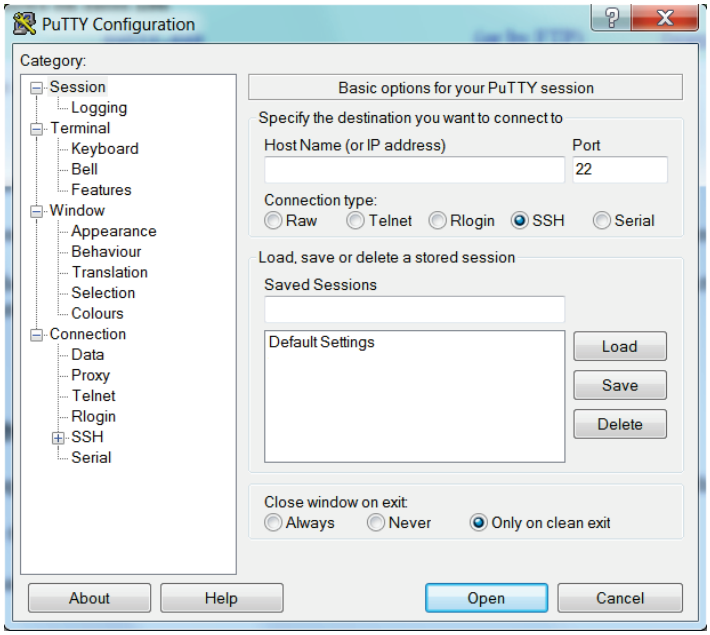
Now that the components are started you may move onto the next section.

1.2 Getting PuTTY setup

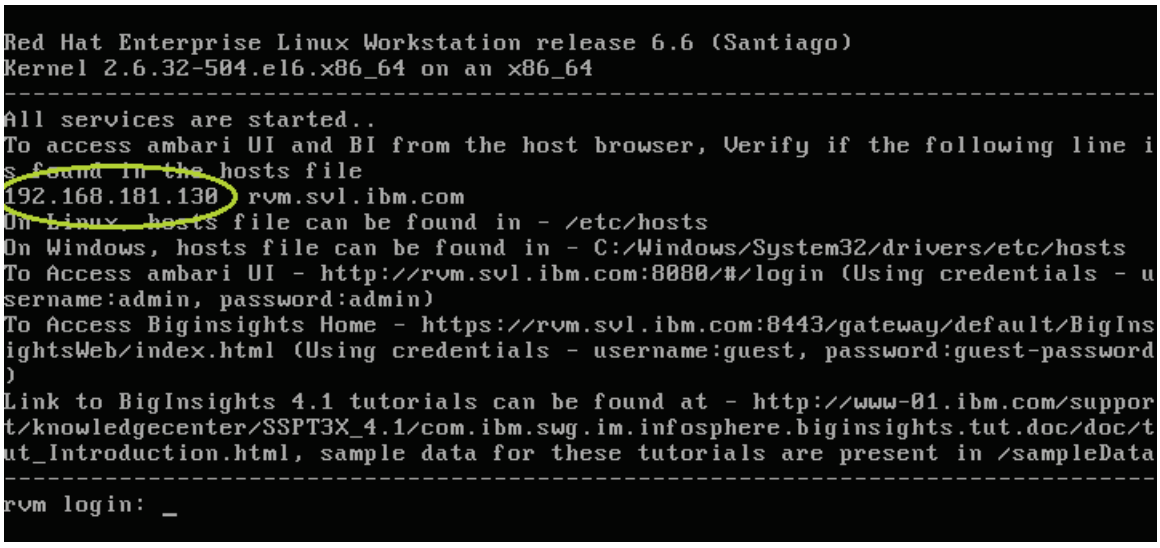
We will be requiring the assistance of PuTTY for this lab, as since this lab involves the use of 4 command line prompts to be open and running at the same time. We run into a little problem since we only have one available to us on the VM. That's where PuTTY comes in, we will be able to use multiple PuTTYs to get remote shell access your VM and have multiple command lines to our disposal! *Note: Make sure your VM is up and running when trying to connect.*

__ 1. Navigate to www.chiark.greenend.org.uk/~sgtatham/putty/download.html and download PuTTY

__ 2. After the download, open PuTTY. It should look like this:

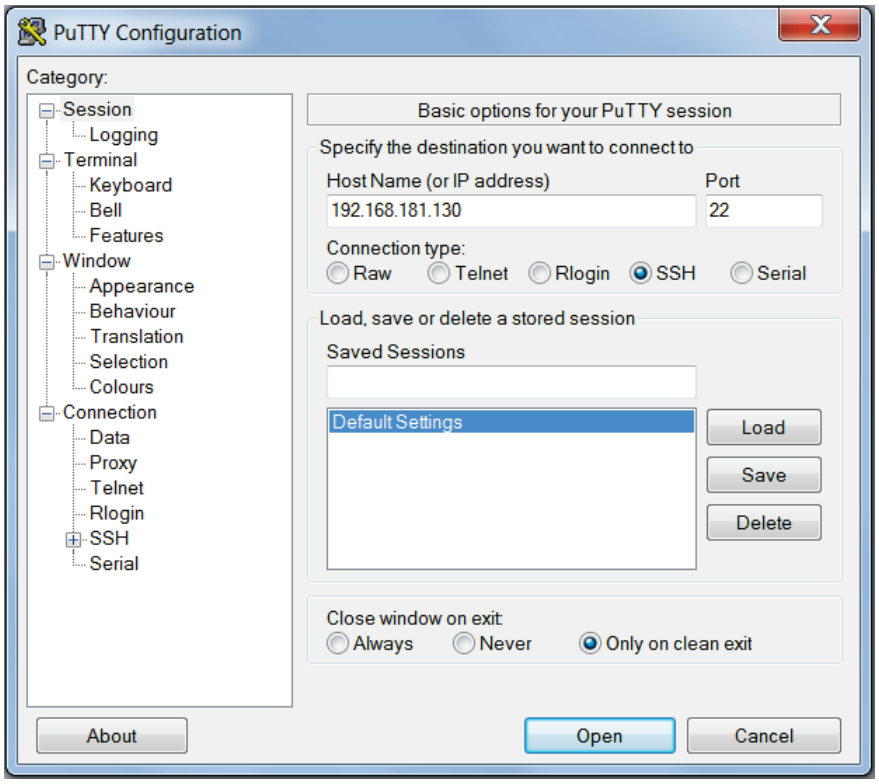


__3. Now for your Host Name, you should be able to find it easily on the page when you first log into BigInsights QuickStart 4.1 VM image. It should show it on this page.



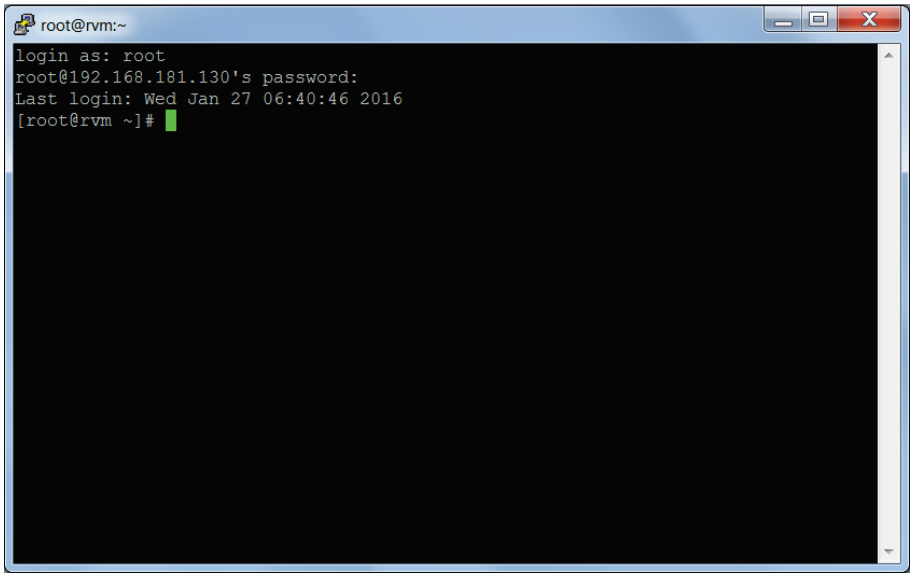
Note: To get to this page when your already logged in, just type *"exit"* until you get back to this screen (instead of restart the whole VM)

__4. Now just input the Host Name (IP Address) into your PuTTY client under *"Host Name"* and hit *"Open."* It should look like this: (Optional: Hit *"Save"* after you input your details to keep it for next time!)



Note: Your Host Name will be different from the above.

__5. Next you should be greeted with a command line requesting login details. Just enter the *VM image setup screen* login (default- username: **root** password: **password**). Then you have successfully setup PuTTY for to connect to your VM!

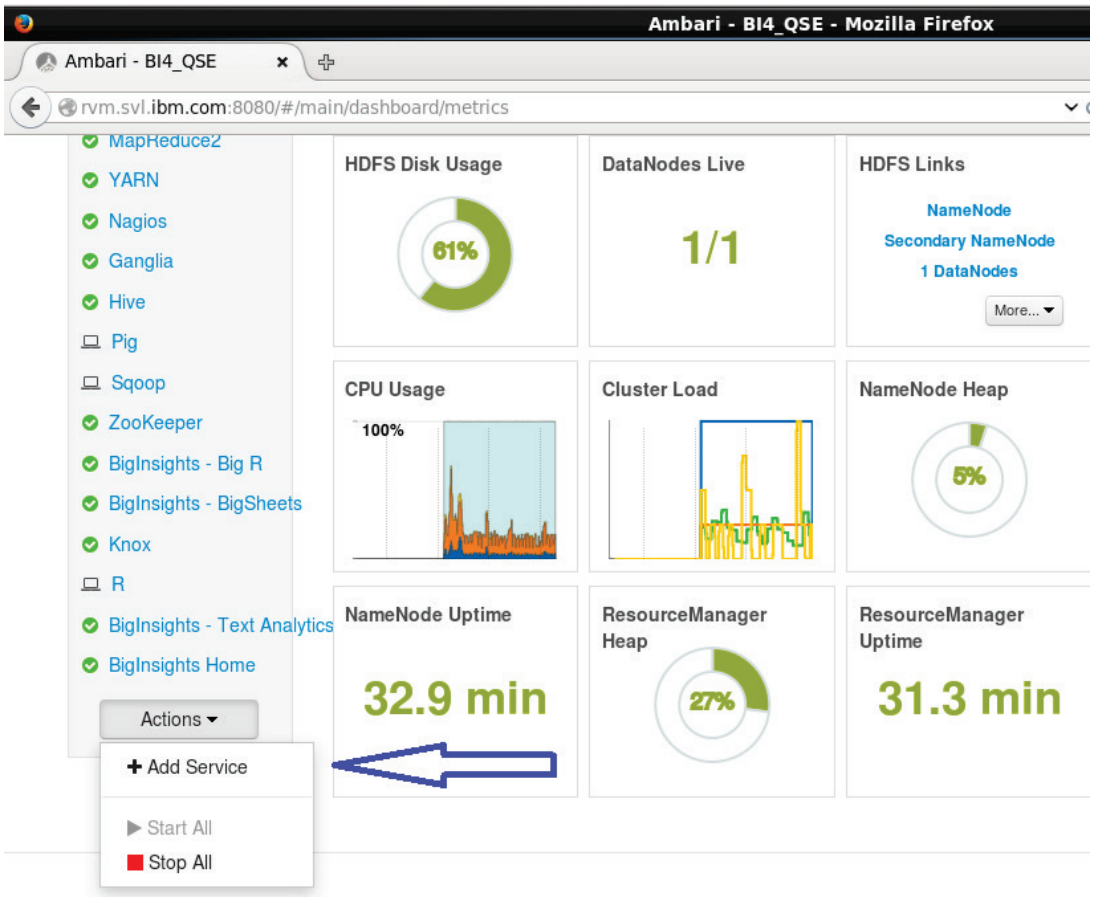


1.3 Install the Flume service

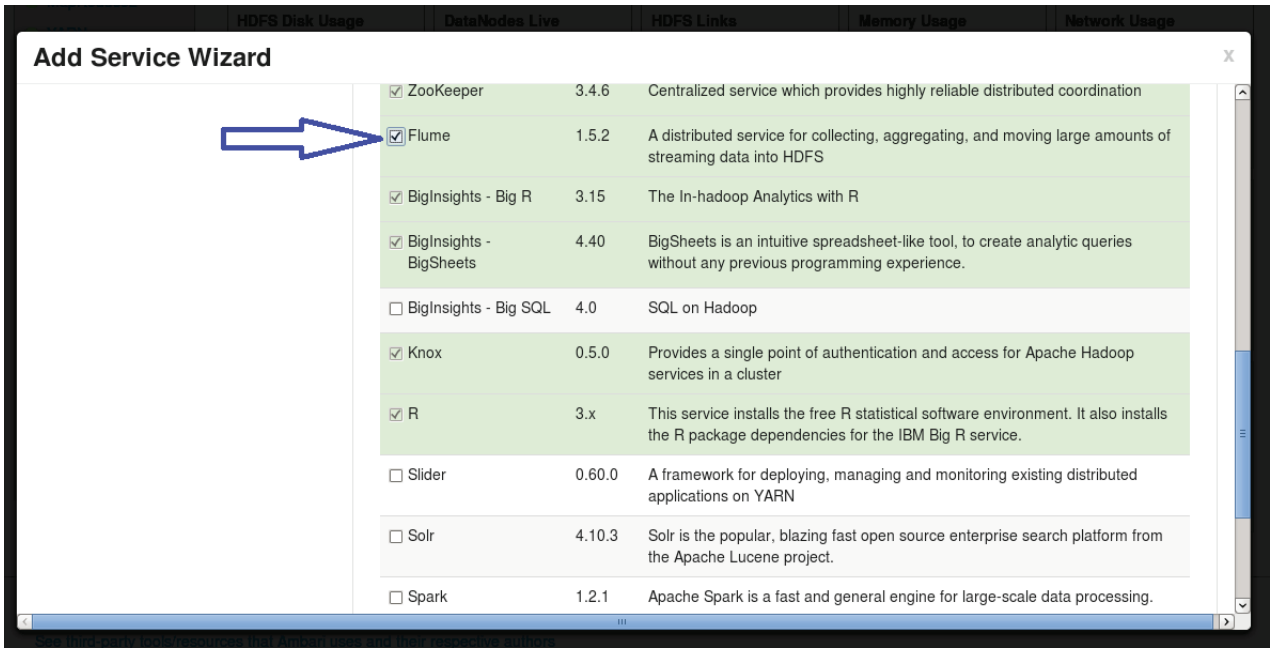
Before you proceed with this lab, we first need to use the Ambari interface to add the Flume service to our BigInsights virtual machine.

__ 1. Log into your lab image as the root.

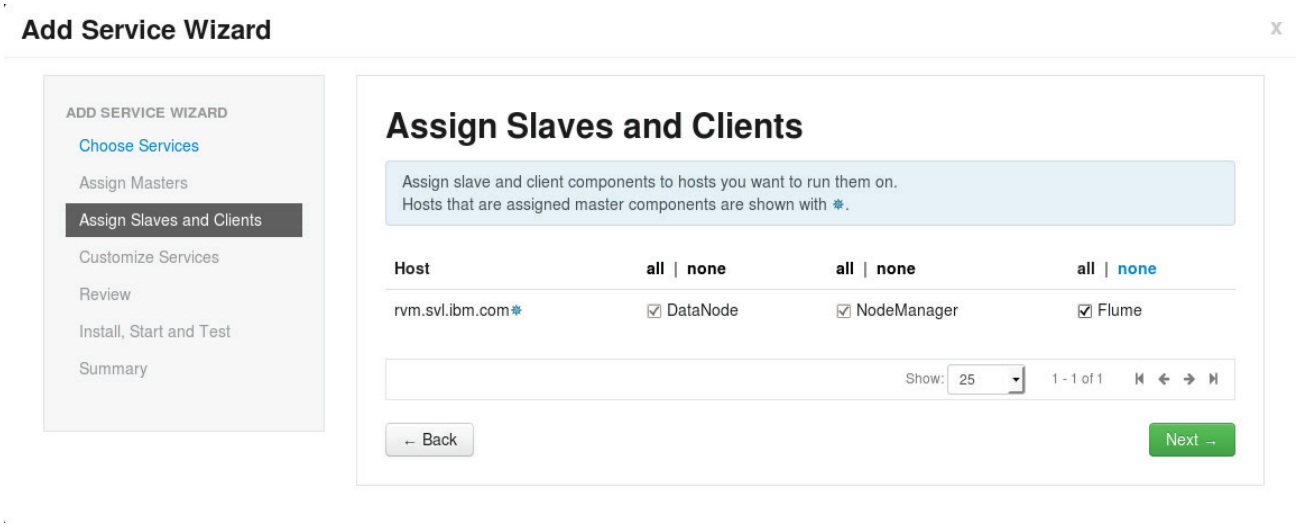
__ 2. Open Ambari. Once all the Hadoop components have started up, click the Actions dropdown and Add Service.



__ 3. Scroll down and check Flume. Then click Next.



Click Next on the following screen.



Keep clicking Next through the screens and finally click the Deploy button. You should receive confirmation that the service has been deployed and started. After the install you may be asked to restart some of the Hadoop components. Restart the requested components and move on to the next section of the lab.

1.4 Creating a configuration file for basic testing

All of the information that is required by a Flume agent is acquired from a configuration file. So to begin with, you are going to code up a simple configuration file. You are going to first define a source and a target that use some built-in Flume testing capabilities.

__ 1. Make sure you are logged into your lab image as root**.**

__ 2. If Hadoop is not running, start it using Ambari.

__ 3. Your Flume configuration file can reside anywhere as long as the agent can access it. The convention is to place the configuration file in Flume's *conf* directory. We will just place our Flume configuration files into the /home/virtuser directory for this lab.

Start the *vi* editor or use a notepad (transferred via VM Shared Folder).

Name of the file will be *flume_agent1.properties.*

Entries in the configuration file are prefixed with an agent's name. Assume that the first agent with which you are going to work is to be named *agent1.* Also, since this is possibly the first time that you have worked with Flume, you will initially make use of some of the Flume testing capabilities.

The sequential generator source is an easy source to use since it creates the source data for you. The logger is a good sink with which to play since it can display the results in your console window.

Remember from the presentation material that a source and a sink are connected together via a channel. You will use the memory channel for this exercise.

__ 4. Although the order in which the Flume elements are defined is immaterial, I am going to present them in a way in which I am comfortable. I am going to first tell you what is to be defined followed by the actual statements. If you want to try your luck in coding the configuration statements before seeing the answers, I suggest that you cover the answers with a sheet of paper, code your own statements, and then do a comparison.

Define your source, sink, and channel. Remember your agent's name is *agent1.*

__ a. Source name is *seqGenSource*

__ b. Sink name is *loggerSink*

__ c. Channel name is *memChannel*

Code the following in your editor:

```
agent1.sources = seqGenSource
agent1.sinks = loggerSink
agent1.channels = memChannel
```

__ 5. Code the properties for *seqGenSource*

__ a. The source type is *seq*

Code the following in your editor:

```
agent1.sources.seqGenSource.type = seq
```

__ 6. Code the properties for *loggerSink*

__ a. The sink type is *logger*

Code the following in your editor:

```
agent1.sinks.loggerSink.type = logger
```

__ 7. Code the properties for *memChannel*

__ a. The channel type is *memory*

__ b. Its capacity is 100

Code the following in your editor:

```
agent1.channels.memChannel.type = memory
agent1.channels.memChannel.capacity = 100
```

__ 8. Connect your source to your defined channel.

```
agent1.sources.seqGenSource.channels = memChannel
```

__ 9. Connect your sink to your defined channel.

```
agent1.sinks.loggerSink.channel = memChannel
```

Important:

Did you note that the binding definition for the source contains the keyword *channels* (plural) and the binding definition for the sink contains the keyword *channel*? This is because a source can read from multiple channels whereas a sink can only write to a single channel.

__ 10. Save your work into **File System->home->virtuser** and make sure the file is named *flume_agent1.properties.*

1.5 Testing your first agent

__ 1. From a command line, change to the flume directory.

```
cd /usr/iop/4.1.0.0/flume/
```

__ 2. Since this is just an exercise and exercises should never mirror real life, you are not going to worry about specifying a configuration directory and setting environment variables. Information about that was covered in the presentation material.

Note:

To terminate your running agent, do a ctrl-z in the console window. This is true for both an agent that did not initialize properly due to a configuration error and one that is running just fine. The cntl-z does not terminate the Java process however.

If you have a configuration error, do a ctrl-z, correct your problem and restart your agent. You do not have to worry about killing the process. The existing process is able to reload the configuration file.

Now start your flume agent. Override the default logging information and write informational records to the console. Your agent's name is *agent1.* **Note:** a lot of data will be written to the console. cnlt-z will terminate the output.

```
bin/flume-ng agent --name agent1 --conf conf --conf-file
/home/virtuser/flume_agent1.properties -Dflume.root.logger=INFO,console
```

Or

```
bin/flume-ng agent -n agent1 --conf conf -f / home/virtuser
/flume_agent1.properties -Dflume.root.logger=INFO,console
```

1.6 A more complicated configuration overview

You did not have to really use your imagination when working with the first flume agent. But you will now. Here is the configuration that you are to implement.

Files are periodically dropped into a directory on system A. The data from each of those files is to be read and turned into events. Each of those events is to be forwarded to system B where the data is to be enhanced by adding timestamp information into the header for each event. The enhanced events are then to be sent to system C where the events are to be loaded into HDFS. The timestamp information in the event header is to be used to define the directory names in which the events are to be stored.

In real life each of the three agents would run on separate systems and so you would have three configuration files. But since your three agents are running on the same system, you can use just a single configuration file.

To add some humor, at least for older people in the U.S.A. who remember the TV series, G*et Smart,* the name of the three agents are *agent13*, *agent99*, and *agent86*.

1.7 Setup your agents

Note:

I purposely chose to use the same channel name for all three agents. This is to show you that the names only have to be unique within an agent.

agent13 is to do the initial read of the data from files dropped into a specified directory.

__ 1. First create your directory. From a command line

```
mkdir /home/virtuser/flumesourcedata
```

__ 2. Open a new file in your text editor.

__ 3. Define the source, sink, and channel to be used by agent13 as well as the bindings.

__ a. The source name is *spoolDirSource*. Its type is *spoolDir.*

__ b. The sink name is *avroSink*. Its type is *avro*. (Remember that to pass events from one agent to another requires avro sinks and sources.)

The *hostname* for binding is *localhost* and the port is 10013.

__ c. The channel name is *memChannel.* Its type is *memory* and it has a capacity of 100.

```
#These statements are for agent13
agent13.sources = spoolDirSource
agent13.sinks = avroSink
agent13.channels = memChannel

agent13.sources.spoolDirSource.type = spooldir
agent13.sources.spoolDirSource.spoolDir= /home/virtuser/flumesourcedata
agent13.sinks.avroSink.type = avro
agent13.sinks.avroSink.hostname = localhost
agent13.sinks.avroSink.port = 10013
agent13.channels.memChannel.type = memory
agent13.channels.memChannel.capacity = 100

agent13.sources.spoolDirSource.channels = memChannel
agent13.sinks.avroSink.channel = memChannel
```

__ 4. agent99 is to get its events from agent13. Since the avro sink for agent13 was bound to *localhost* at port 10013, that implies that the avro source for agent99 will also be bound to locahost at port 10013.

Also, you are going to enhance your events by adding a timestamp to the header for each event.

Define the source, sink, and channel to be used by agent99 as well as the bindings.

__ a. The source name is *avroSource*. Its type is *avro.*

The bind parameter is *localhost*
The port is 10013
The interceptor type is *ts*
The interceptor type is *timestamp*

__ b. The sink name is *avroSink*. Its type is *avro*.

The hostname for binding is localhost and the port is 10099.

__ c. The channel name is *memChannel.* its type is *memory* and it has a capacity of 100.

```
#These statements are for agent99
agent99.sources = avroSource
agent99.sinks = avroSink
agent99.channels = memChannel

agent99.sources.avroSource.type = avro
agent99.sources.avroSource.bind = localhost
agent99.sources.avroSource.port = 10013
agent99.sources.avroSource.interceptors = ts
agent99.sources.avroSource.interceptors.ts.type = timestamp
agent99.sinks.avroSink.type = avro
agent99.sinks.avroSink.hostname = localhost
agent99.sinks.avroSink.port = 10099
agent99.channels.memChannel.type = memory
agent99.channels.memChannel.capacity = 100

agent99.sources.avroSource.channels = memChannel
agent99.sinks.avroSink.channel = memChannel
```

__ 5. agent86 is to get its events from agent99. So there must be an avro source to receive the data and the data is to be passed to an hdfs sink.

Define the source, sink, and channel to be used by agent86 as well as the bindings.

__ a. The source name is *avroSource*. Its type is *avro.*

The bind parameter is *localhost*
The port is 10099

__ b. The sink name is *hdfsSink*. Its type is *hdfs*.

A portion of the hdfs path is created by extracting date and time information from the header of each event. hdfs://rvm.svl.ibm.com:8020/user/virtuser/flume/%y-%m-%d/%H%M

The filePrefix is *log.*
The writeFormat is *Text*

The fileType is *DataStream*

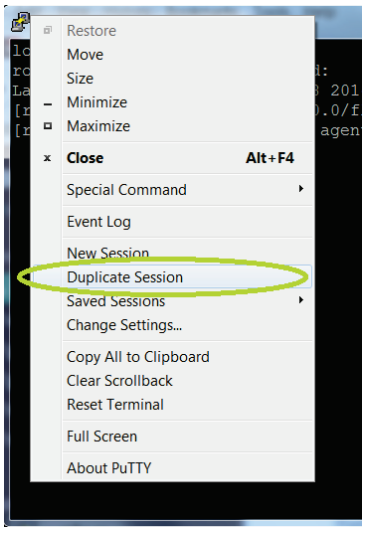
__ c. The channel name is *memChannel.* Its type is *memory* and it has a capacity of 100.

```
#These statements are for agent86
agent86.sources = avroSource
agent86.sinks = hdfsSink
agent86.channels = memChannel

agent86.sources.avroSource.type = avro
agent86.sources.avroSource.bind = localhost
agent86.sources.avroSource.port = 10099
agent86.sinks.hdfsSink.type = hdfs
agent86.sinks.hdfsSink.hdfs.path =
hdfs://rvm.svl.ibm.com:8020/user/virtuser/flume/%y-%m-%d/%H%M
agent86.sinks.hdfsSink.hdfs.filePrefix = Log
agent86.sinks.hdfsSink.hdfs.writeFormat = Text
agent86.sinks.hdfsSink.hdfs.fileType = DataStream
agent86.channels.memChannel.type = memory
agent86.channels.memChannel.capacity = 100

agent86.sources.avroSource.channels = memChannel
agent86.sinks.hdfsSink.channel = memChannel
```

__ 6. Save your work into **File System->home->virtuser** and call the file *flume_agents.properties.*

1.8 Test your configuration

You are going to have to be working with a number of terminal windows. A quick way to open the same session of PuTTY, just right-click on the PuTTY and select *"Duplicate Session"*



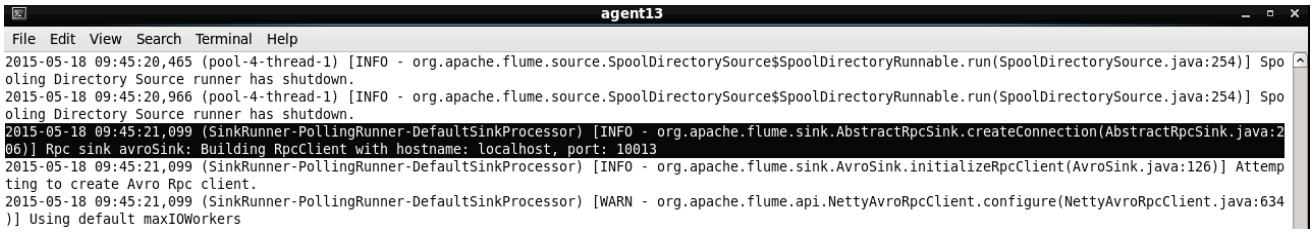
agent13

__ 1. Open a PuTTY window, connect and change to the *flume* directory.

```
cd /usr/iop/4.1.0.0/flume/
```

__ 2. When you start agent13, even though you coded your configuration statements correctly, you will see what looks like a Java exception when you start the agent. That is because the avro sink is not able to connect to the source yet. Once agent99 starts and the avro source does its bind, you should see a statement something like the following:

```
INFO sink.AvroSink: Avro sink avroSink: Building RpcClient with hostname:
rvm, port: 10013
```



Execute the following:

```
bin/flume-ng agent -n agent13 --conf conf -f
/home/virtuser/flume_agents.properties -Dflume.root.logger=INFO,console
```

agent99

__ 3. Open PuTTY window, connect and change to the *flume* directory.

```
cd /usr/iop/4.1.0.0/flume/
```

__ 4. When you start agent99, even though you coded your configuration statements correctly, you will see what looks like a Java exception when you start the agent. That is because the avro sink is not able to connect to the source yet. Once agent86 starts and the avro source does its bind, you should see a statement something like the following:

```
Rpc sink avroSink: Building RpcClient with hostname: localhost, port:
10099
```

Execute the following:

```
bin/flume-ng agent -n agent99 --conf conf -f
/home/virtuser/flume_agents.properties -Dflume.root.logger=INFO,console
```

agent86

__ 5. Open a PuTTY window, connect and change to the *flume* directory.

```
cd /usr/iop/4.1.0.0/flume/
```

__ 6. Start agent86. You should see a statement as follows:

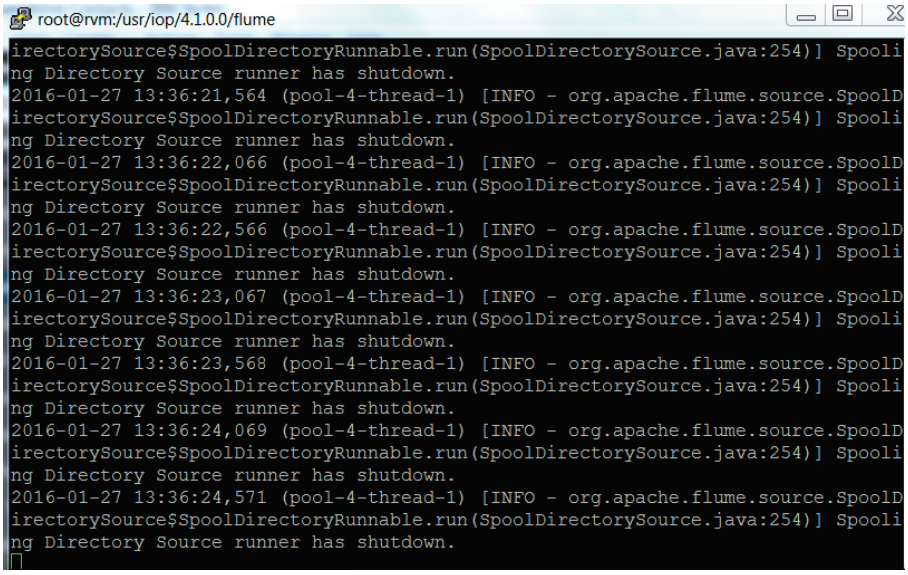
```
INFO source.AvroSource: Avro source avroSource started.
```

Execute the following:

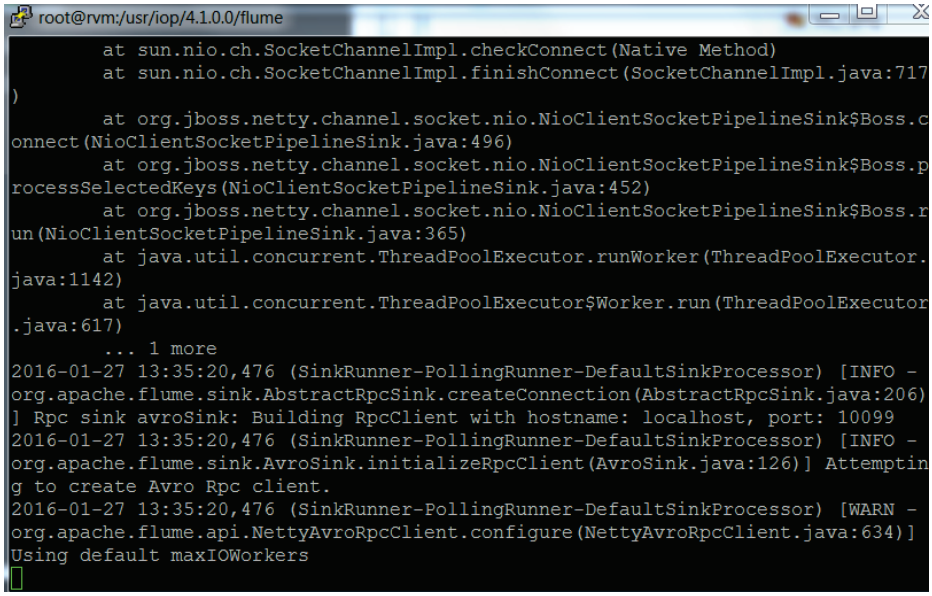
```
bin/flume-ng agent -n agent86 --conf conf -f
/home/virtuser/flume_agents.properties -Dflume.root.logger=INFO,console
```

You should have 3 windows that look similar to this after they are all running:

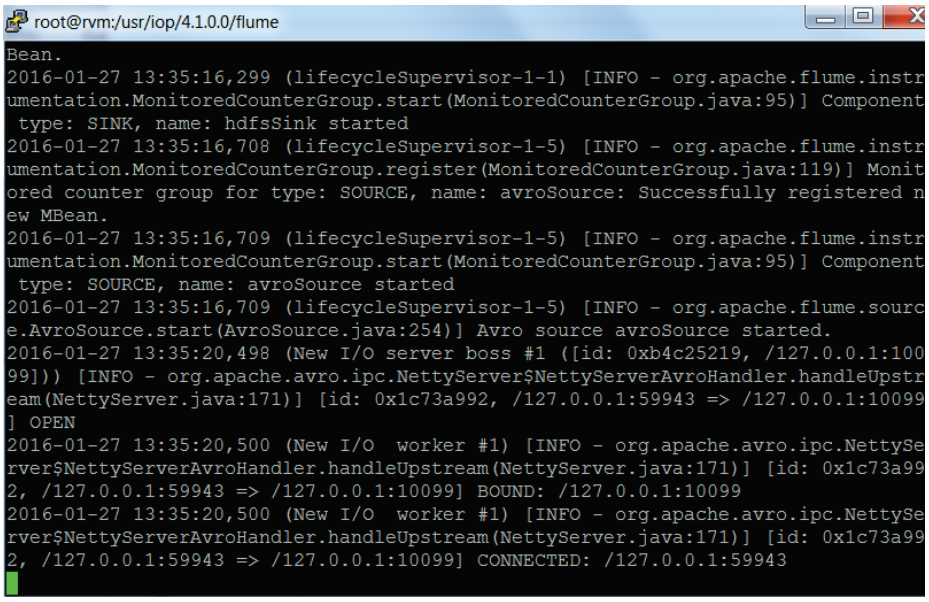
agent13:



agent99:



agent86:



1.9 Transfer data

Assuming that all of your agents have properly started, you need to test the moving of data from agent13 into HDFS.

__ 1. Open another terminal window.

__ 2. Change to the virtuser directory.

```
cd /home/virtuser/
```

__ 3. Create a file with test data.

```
cat > test.txt
this is some data
to upload to hadoop
ctrl-c
```

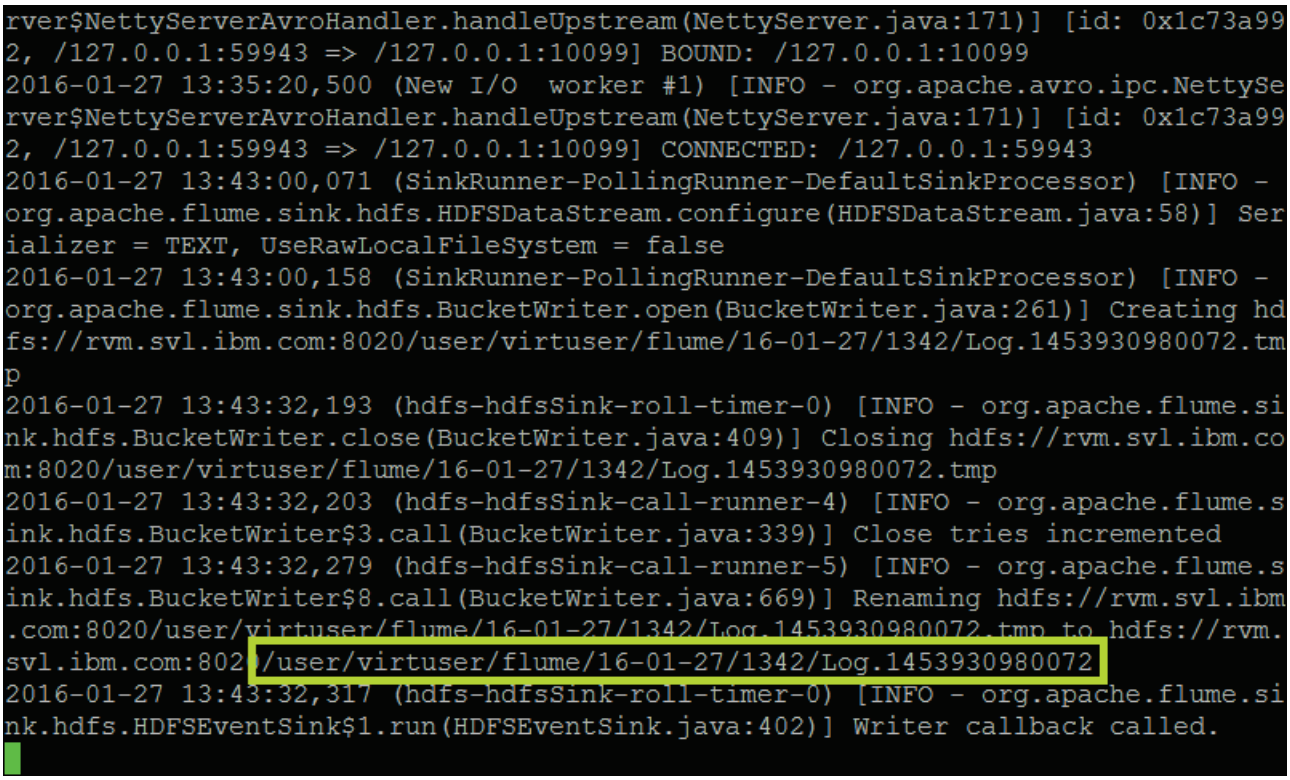
__ 4. Copy your file to the flume*sourcedata* directory.

```
cp test.txt flumesourcedata
```

__ 5. As soon as the file was added to the *sourcedata* directory, it gets processed. List the contents of the *sourcedata* directory. You should see that the *test.txt* file has been renamed to *test.txt.COMPLETED*

```
ls flumesourcedata
```

__ 6. Next check to see that the data was moved to HDFS. Return to the terminal window where you started *agent86.* You should see some statements indicating that a file was created as a temporary file and then has been renamed. Notice that the directory structure has been made up of some data and time information.



__ 7. View the contents of the newly created file. Return to the terminal window where you created the *text.txt* file. Execute the following replacing the file name with your file name. (You can do a copy and paste of the file name from the terminal window for *agent86.*) *Note: You must be in hdfs to use hadoop unless you've changed permissions.*

```
su hdfs
hadoop fs -cat /user/virtuser/flume/16-01-27/1342/Log.1453930980072
```

__ 8. Execute ctrl-z in each of the three windows where the agents are running in order to terminate them.

__ 9. You can close your open terminal windows.

End of exercise

NOTES

NOTES

Please Recycle

IBM Software