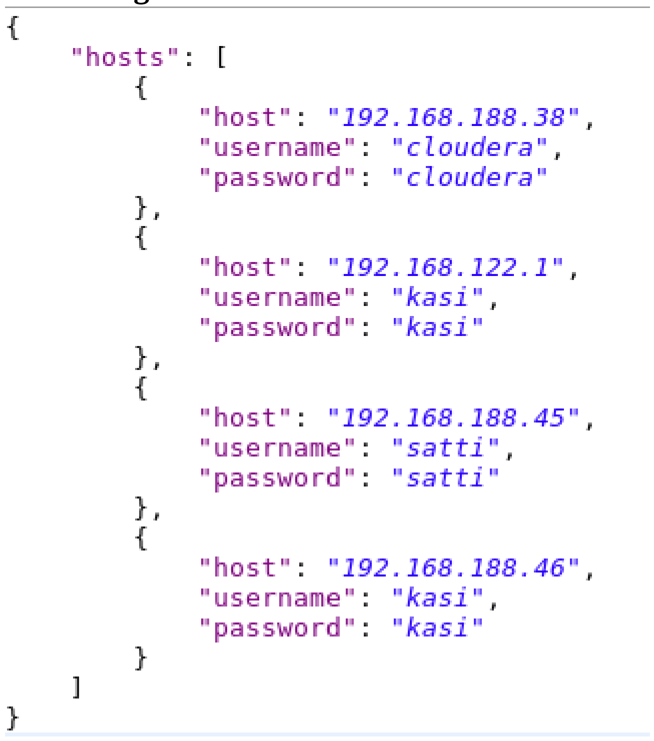
Cloudera Configuration Validator Java Application

Create a Cluster Access Configuration File:



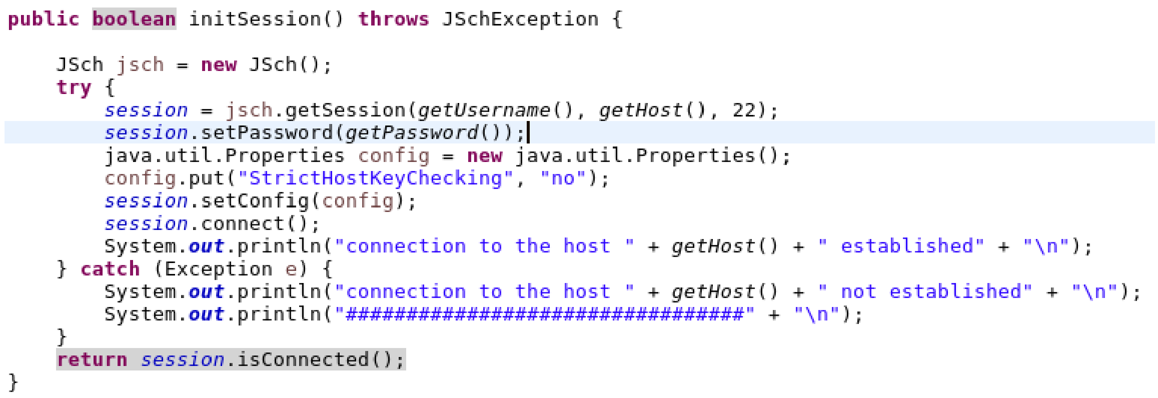
```
{
    "hosts": [
        {
            "host": "192.168.188.38",
            "username": "cloudera",
            "password": "cloudera"
        },
        {
            "host": "192.168.122.1",
            "username": "kasi",
            "password": "kasi"
        },
        {
            "host": "192.168.188.45",
            "username": "satti",
            "password": "satti"
        },
        {
            "host": "192.168.188.46",
            "username": "kasi",
            "password": "kasi"
        }
    ]
}
```

Configuration.json

'configuration.json' file contains array of json objects called hosts. Each object has 'host', 'username','password', used to connect to Remote Machine. we can add multiple host objects to this array in order to connect to Remote Machine.

Creating a JSch Session:

In order to connect to Remote host get the session for the host.
SSH (Secure Shell) is a protocol that allows to establish a session with another host via command line interface. JSch is a pure java implementation of SSH2 allows to connect to sshd server



```
public boolean initSession() throws JSchException {

    JSch jsch = new JSch();
    try {
        session = jsch.getSession(getUsername(), getHost(), 22);
        session.setPassword(getPassword());
        java.util.Properties config = new java.util.Properties();
        config.put("StrictHostKeyChecking", "no");
        session.setConfig(config);
        session.connect();
        System.out.println("connection to the host " + getHost() + " established" + "\n");
    } catch (Exception e) {
        System.out.println("connection to the host " + getHost() + " not established" + "\n");
        System.out.println("##################################" + "\n");
    }
    return session.isConnected();
}
```

The method initSession() returns true if all the user credentials are correct and then it establishes a session for the particular user.

JSch : The starting point used to create session. JSch() creates a new jsch object

getSession(String username, String host) of jsch object used to start a new Session. It Instantiates the Session object with username and host. The TCP port 22 will be used in making the connection. The TCP connection must not be established until the Session.connect() is called.

config.put("StrictHostKeyChecking", "no");

If this property is set to yes", jsch will never automatically add host keys to the $HOME/.ssh/known_hosts file, and refuses to connect to hosts whose host key has changed. This property forces the user to manually add all new hosts. If this property is set to no", jsch will automatically add new host keys to the user known hosts files.

Session: A connection to a SSH server. Used to configure settings and to open channels.

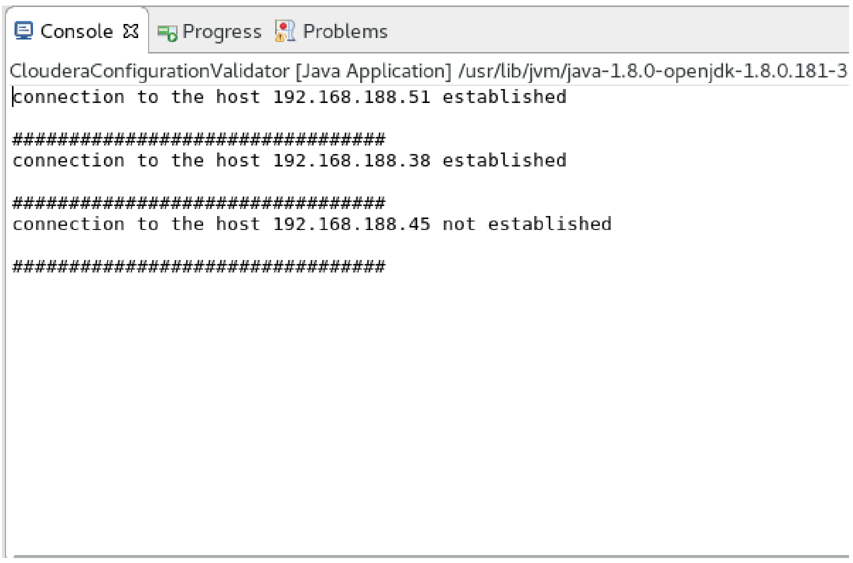
SetPasssword(byte[] password) : set the password to use for authentication

connect(): opens the connection.

Note: connect() only opens connection for a user, a channel is not opened yet.

Create a json file with array of json object contains 'hostname','username','password'. Parse each and every json object and pass the values to initSession();

If username, hostname, password matches then a session is established for the host.



ClouderaConfigurationValidator [Java Application] /usr/lib/jvm/java-1.8.0-openjdk-1.8.0.181-3

```
connection to the host 192.168.188.51 established

#################################
connection to the host 192.168.188.38 established

#################################
connection to the host 192.168.188.45 not established

#################################
```

open exec channel and run command:

```
public ArrayList<String> runCommand(String command) throws JSchException, IOException {

    ArrayList<String> mylist = new ArrayList<String>();
    channel = session.openChannel("exec");
    ((ChannelExec) channel).setCommand(command);
    channel.connect();
    try {
        BufferedReader bufferReader = new BufferedReader(new InputStreamReader(channel.getInputStream()));
        String line = null;
        while ((line = bufferReader.readLine()) != null) {
            mylist.add(line);
        }
        bufferReader.close();
    } catch (IOException ex) {
        ex.printStackTrace();
    }
    return mylist;
}
```

The method runCommand(String command) takes the shell command as input string, opens the exec channel, runs the given command and returns the output of the command as ArrayList of Strings.

InitSession() method only creates the session and opens the connection, in-order to **run shell commands a 'exec' channel has to be open for a particular session.**

A channel connected to a remotely executing program. Such a channel is created with:

ChannelExec channel = (ChannelExec)session.openChannel("exec");

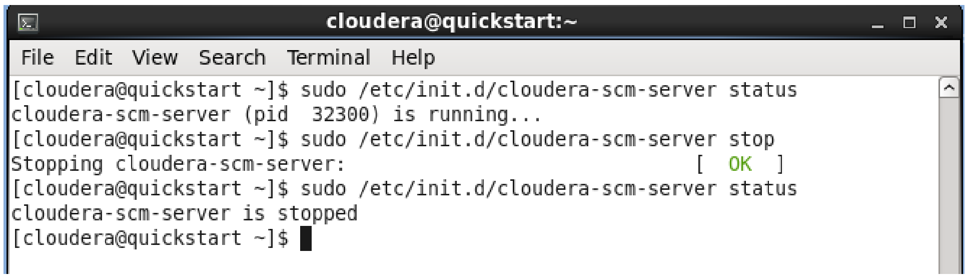
channel.setCommand(command);

Channel.connect(): opens the connection

The method runCommand() opens the 'exec channel' and runs the command. The output of the command passes to BufferedReader as input stream. InputStreamReader reads bytes and decodes them into characters using a specified charset, BufferedReader enables the efficient conversion of bytes to characters, more bytes may be read ahead from the underlying stream. For top efficiency, consider wrapping an InputStreamReader within a BufferedReader. runCommand() returns the ouput as ArrayList of strings.

Validate Running Cloudera Server:

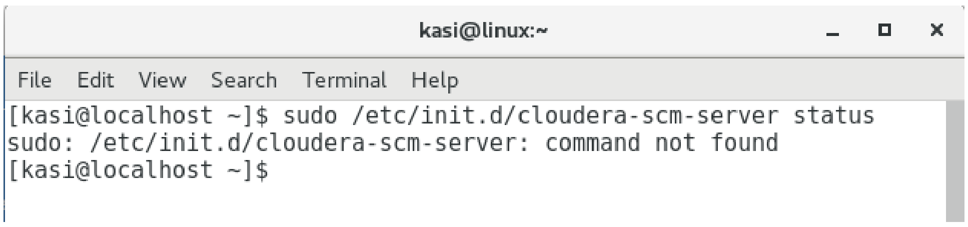
Running the shell command /etc/init.d/cloudera-scm-server returns the output :



```
[cloudera@quickstart ~]$ sudo /etc/init.d/cloudera-scm-server status
cloudera-scm-server (pid  32300) is running...
[cloudera@quickstart ~]$ sudo /etc/init.d/cloudera-scm-server stop
Stopping cloudera-scm-server:                              [  OK  ]
[cloudera@quickstart ~]$ sudo /etc/init.d/cloudera-scm-server status
cloudera-scm-server is stopped
[cloudera@quickstart ~]$
```

If cloudera is not installed on the host machine.

Running the shell command /etc/init.d/cloudera-scm-server returns the output:



```
[kasi@localhost ~]$ sudo /etc/init.d/cloudera-scm-server status
sudo: /etc/init.d/cloudera-scm-server: command not found
[kasi@localhost ~]$
```

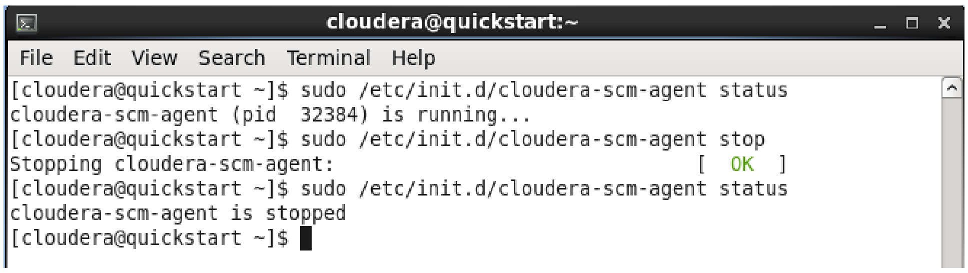
```
public void checkClouderaServer() throws JSchException, IOException {
    ArrayList<String> output = runCommand("/etc/init.d/cloudera-scm-server status");
    if (output.isEmpty()) {
        System.out.println("cloudera-scm-server is not running          [FAILED]");

    } else {
        String string = output.get(0);

        if ((string.substring(0, string.length())).contains("running...")) {
            System.out.println("cloudera-scm-server is running               [OK]");
        } else if ((string.substring(0, string.length())).contains("stopped")) {
            System.out.println("cloudera-scm-server is running               [OK]");
        } else if ((string.substring(0, string.length())).contains("No such file or directory")) {
            System.out.println("cloudera-scm-server is not running          [FAILED]");
        } else {
            System.out.println("cloudera-scm-server is not running          [FAILED]");
        }
    }

}
```

The method checkClouderServer() runs the shell command '/etc/init.d/cloudera-scm-server status' and stores the output to ArrayList, if the output is 'empty' or 'command not found' the method prints 'cloudera-scm-server is not running'.

If the output has some values, the method gets the first line of the output and compares the substring. If substring contains Running the method prints 'cloudera-scm-server is running', if substring contains stopped the method prints ‘cloudera-scm-server is stopped ’else it prints 'cloudera-scm-server is not running'.

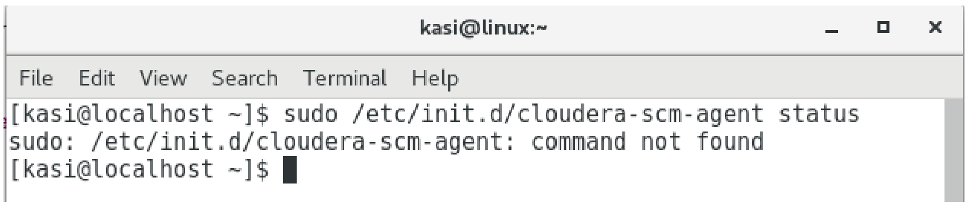
Validate Running Cloudera Agents:

If cloudera is installed on the host machine, Running the shell command ‘/etc/init.d/cloudera-scm-agent' returns the output.



```
[cloudera@quickstart ~]$ sudo /etc/init.d/cloudera-scm-agent status
cloudera-scm-agent (pid  32384) is running...
[cloudera@quickstart ~]$ sudo /etc/init.d/cloudera-scm-agent stop
Stopping cloudera-scm-agent:                               [  OK  ]
[cloudera@quickstart ~]$ sudo /etc/init.d/cloudera-scm-agent status
cloudera-scm-agent is stopped
[cloudera@quickstart ~]$ 
```

If cloudera is not installed on the host machine. Running the shell command '/etc/init.d/cloudera-scm-agent' returns the output.



```
[kasi@localhost ~]$ sudo /etc/init.d/cloudera-scm-agent status
sudo: /etc/init.d/cloudera-scm-agent: command not found
[kasi@localhost ~]$ 
```

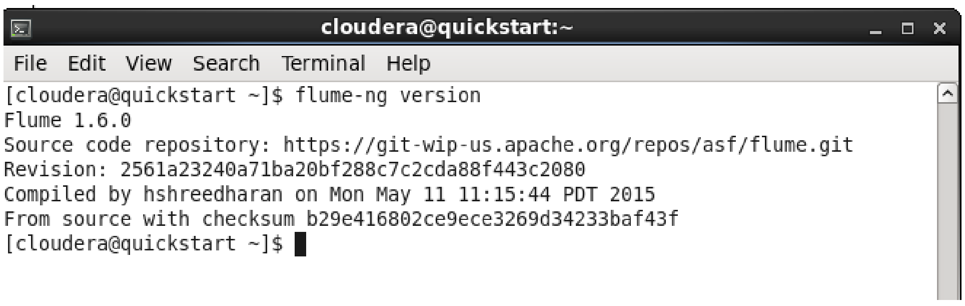
```java
public void checkClouderaAgent() throws JSchException, IOException {
    ArrayList<String> output = runCommand("/etc/init.d/cloudera-scm-agent status");
    if (output.isEmpty() || output.contains("command not found")) {
        System.out.println("cloudera-scm-agent is not running            [FAILED]");
    } else {
        String string = output.get(0);
        if ((string.substring(0, string.length())).contains("running...")) {
            System.out.println("cloudera-scm-agent is running                [OK]");

        } else if ((string.substring(0, string.length())).contains("stopped")) {
            System.out.println("cloudera-scm-agent is running                [OK]");
        } else if ((string.substring(0, string.length())).contains("No such file or directory")) {
            System.out.println("cloudera-scm-agent is not running            [FAILED]");
        } else {
            System.out.println("cloudera-scm-agent is not running            [FAILED]");
        }

    }

}
```

The method checkClouderAgent() runs the shell command '/etc/init.d/cloudera-scm-agent status' and stores the output to ArrayList, if the output is 'empty' or 'command not found' the method prints 'cloudera-scm-agent is not running'.

If the output has some values, the method gets the first line of the output and compares the substring. If substring contains Running the method prints 'cloudera-scm-agent is running', if substring contains stopped the method prints 'cloudera-scm-server stopped' else it prints 'cloudera-scm-agent is not running'.

Validate Flume Package Availability and Run Test:

When running the shell command 'flume-ng version' it returns the output :



```
[cloudera@quickstart ~]$ flume-ng version
Flume 1.6.0
Source code repository: https://git-wip-us.apache.org/repos/asf/flume.git
Revision: 2561a23240a71ba20bf288c7c2cda88f443c2080
Compiled by hshreedharan on Mon May 11 11:15:44 PDT 2015
From source with checksum b29e416802ce9ece3269d34233baf43f
[cloudera@quickstart ~]$ 
```

```
public void checkFlume() throws JSchException, IOException, InterruptedException {
    ArrayList<String> output = runCommand("flume-ng version");
    if (output.isEmpty() || output.contains("command not found")) {
        System.out.println("Flume packages are not installed             [FAILED]");
    } else if ((output.get(0).substring(0, output.size())).contains("Flume")) {
        System.out.println("Flume Packages are installed                 [OK]");
        boolean containstxt = false;
        ArrayList<String> flumecheck = runCommand("hadoop fs -ls /exec-hdfs/");
        for (String string : flumecheck) {
            if (string.contains(".txt")) {
                containstxt = true;
                break;
            }
        }
        System.out.println(containstxt);
        if (containstxt) {
            System.out.println("Flume is working");

        } else {
            System.out.println("Flume is not working");
        }
    } else {
        System.out.println("Flume packages are not installed             [FAILED]");
    }
}
```

The method checkFlume() runs the shell command 'flume-ng version' and stores the output to ArrayList. If the output is not empty then the method gets the first line of the output and then compares with the substring.

If substring contains 'Flume' the method prints 'Flume packages are installed[OK]' else it prints 'Flume packages are not installed [FAILED]'. If Flume packages are installed, run flume agent and test against it.

Run Flume Agent:

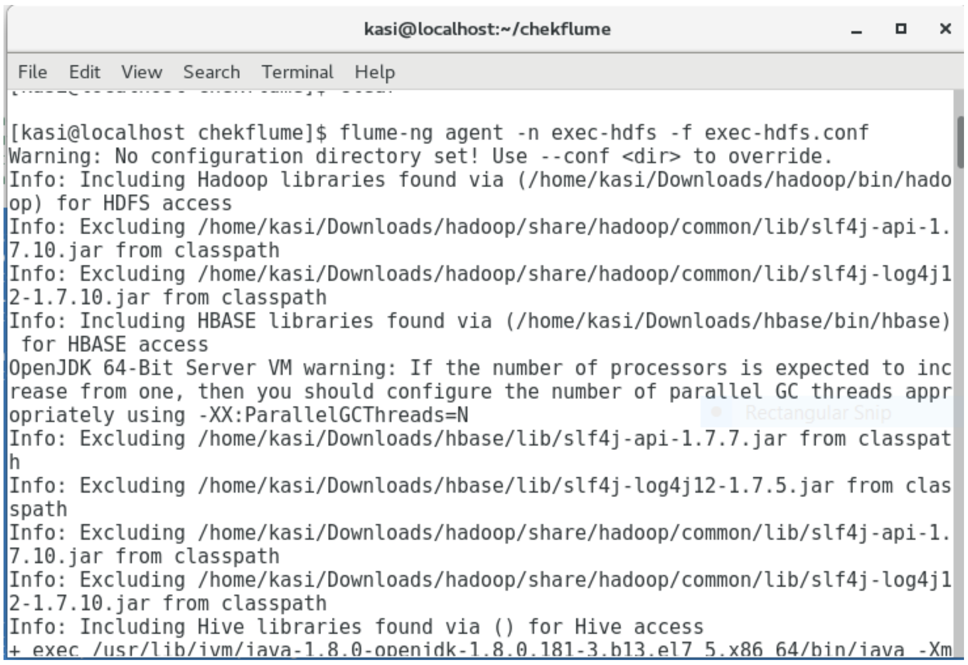
flume-ng agent -n exec-hdfs -f exec-hdfs.conf.

The flume agent reads data from ‘exec source’, and channel is ‘memory’, sink is ‘hdfs’.

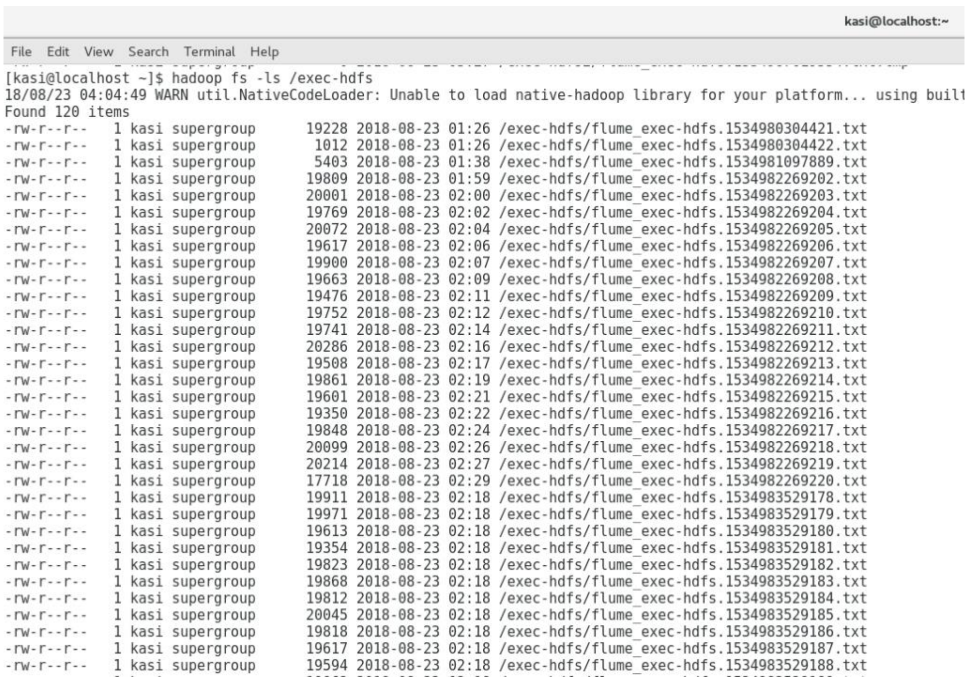
If we run the flume-agent then the data is stored to hdfs in ‘.txt’ format, look for ‘.txt’ file in the directory, if it exists then flume-agent is successfully running.

Flume agent cofiguration file 'exec-hdfs.conf'.

```
# Name the components on this agent
exec-hdfs.sources = ws
exec-hdfs.sinks = hd
exec-hdfs.channels = mem

# Describe/configure the source
exec-hdfs.sources.ws.type = exec
exec-hdfs.sources.ws.command = tail -F /opt/gen_logs/logs/access.log


# Describe the sink
exec-hdfs.sinks.hd.type = hdfs
exec-hdfs.sinks.hd.hdfs.path = hdfs://localhost:8020/exec-hdfs
exec-hdfs.sinks.hd.hdfs.filePrefix = flume_exec-hdfs
exec-hdfs.sinks.hd.hdfs.fileSuffix = .txt
exec-hdfs.sinks.hd.hdfs.rollInterval = 120
exec-hdfs.sinks.hd.hdfs.rollSize = 1048576
exec-hdfs.sinks.hd.hdfs.rollCount = 100
exec-hdfs.sinks.hd.hdfs.fileType = DataStream


# Use a channel which buffers events in memory
exec-hdfs.channels.mem.type = memory
exec-hdfs.channels.mem.capacity = 1000
exec-hdfs.channels.mem.transactionCapacity = 100

# Bind the source and sink to the channel
exec-hdfs.sources.ws.channels = mem
exec-hdfs.sinks.hd.channel = mem
```


kasi@localhost:~/chekflume
File Edit View Search Terminal Help

```
[kasi@localhost chekflume]$ flume-ng agent -n exec-hdfs -f exec-hdfs.conf
Warning: No configuration directory set! Use --conf <dir> to override.
Info: Including Hadoop libraries found via (/home/kasi/Downloads/hadoop/bin/hado
op) for HDFS access
Info: Excluding /home/kasi/Downloads/hadoop/share/hadoop/common/lib/slf4j-api-1.
7.10.jar from classpath
Info: Excluding /home/kasi/Downloads/hadoop/share/hadoop/common/lib/slf4j-log4j1
2-1.7.10.jar from classpath
Info: Including HBASE libraries found via (/home/kasi/Downloads/hbase/bin/hbase)
 for HBASE access
OpenJDK 64-Bit Server VM warning: If the number of processors is expected to inc
rease from one, then you should configure the number of parallel GC threads appr
opriately using -XX:ParallelGCThreads=N
Info: Excluding /home/kasi/Downloads/hbase/lib/slf4j-api-1.7.7.jar from classpat
h
Info: Excluding /home/kasi/Downloads/hbase/lib/slf4j-log4j12-1.7.5.jar from clas
spath
Info: Excluding /home/kasi/Downloads/hadoop/share/hadoop/common/lib/slf4j-api-1.
7.10.jar from classpath
Info: Excluding /home/kasi/Downloads/hadoop/share/hadoop/common/lib/slf4j-log4j1
2-1.7.10.jar from classpath
Info: Including Hive libraries found via () for Hive access
+ exec /usr/lib/jvm/java-1.8.0-openjdk-1.8.0.181-3.b13.el7_5.x86_64/bin/java -Xm
```


kasi@localhost:~

File Edit View Search Terminal Help

```
[kasi@localhost ~]$ hadoop fs -ls /exec-hdfs
18/08/23 04:04:49 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using built
Found 120 items
-rw-r--r--   1 kasi supergroup      19228 2018-08-23 01:26 /exec-hdfs/flume_exec-hdfs.1534980304421.txt
-rw-r--r--   1 kasi supergroup       1012 2018-08-23 01:26 /exec-hdfs/flume_exec-hdfs.1534980304422.txt
-rw-r--r--   1 kasi supergroup       5403 2018-08-23 01:38 /exec-hdfs/flume_exec-hdfs.1534981097889.txt
-rw-r--r--   1 kasi supergroup      19809 2018-08-23 01:59 /exec-hdfs/flume_exec-hdfs.1534982269202.txt
-rw-r--r--   1 kasi supergroup      20001 2018-08-23 02:00 /exec-hdfs/flume_exec-hdfs.1534982269203.txt
-rw-r--r--   1 kasi supergroup      19769 2018-08-23 02:02 /exec-hdfs/flume_exec-hdfs.1534982269204.txt
-rw-r--r--   1 kasi supergroup      20072 2018-08-23 02:04 /exec-hdfs/flume_exec-hdfs.1534982269205.txt
-rw-r--r--   1 kasi supergroup      19617 2018-08-23 02:06 /exec-hdfs/flume_exec-hdfs.1534982269206.txt
-rw-r--r--   1 kasi supergroup      19900 2018-08-23 02:07 /exec-hdfs/flume_exec-hdfs.1534982269207.txt
-rw-r--r--   1 kasi supergroup      19663 2018-08-23 02:09 /exec-hdfs/flume_exec-hdfs.1534982269208.txt
-rw-r--r--   1 kasi supergroup      19476 2018-08-23 02:11 /exec-hdfs/flume_exec-hdfs.1534982269209.txt
-rw-r--r--   1 kasi supergroup      19752 2018-08-23 02:12 /exec-hdfs/flume_exec-hdfs.1534982269210.txt
-rw-r--r--   1 kasi supergroup      19741 2018-08-23 02:14 /exec-hdfs/flume_exec-hdfs.1534982269211.txt
-rw-r--r--   1 kasi supergroup      20286 2018-08-23 02:16 /exec-hdfs/flume_exec-hdfs.1534982269212.txt
-rw-r--r--   1 kasi supergroup      19508 2018-08-23 02:17 /exec-hdfs/flume_exec-hdfs.1534982269213.txt
-rw-r--r--   1 kasi supergroup      19861 2018-08-23 02:19 /exec-hdfs/flume_exec-hdfs.1534982269214.txt
-rw-r--r--   1 kasi supergroup      19601 2018-08-23 02:21 /exec-hdfs/flume_exec-hdfs.1534982269215.txt
-rw-r--r--   1 kasi supergroup      19350 2018-08-23 02:22 /exec-hdfs/flume_exec-hdfs.1534982269216.txt
-rw-r--r--   1 kasi supergroup      19848 2018-08-23 02:24 /exec-hdfs/flume_exec-hdfs.1534982269217.txt
-rw-r--r--   1 kasi supergroup      20099 2018-08-23 02:26 /exec-hdfs/flume_exec-hdfs.1534982269218.txt
-rw-r--r--   1 kasi supergroup      20214 2018-08-23 02:27 /exec-hdfs/flume_exec-hdfs.1534982269219.txt
-rw-r--r--   1 kasi supergroup      17718 2018-08-23 02:29 /exec-hdfs/flume_exec-hdfs.1534982269220.txt
-rw-r--r--   1 kasi supergroup      19911 2018-08-23 02:18 /exec-hdfs/flume_exec-hdfs.1534983529178.txt
-rw-r--r--   1 kasi supergroup      19971 2018-08-23 02:18 /exec-hdfs/flume_exec-hdfs.1534983529179.txt
-rw-r--r--   1 kasi supergroup      19613 2018-08-23 02:18 /exec-hdfs/flume_exec-hdfs.1534983529180.txt
-rw-r--r--   1 kasi supergroup      19354 2018-08-23 02:18 /exec-hdfs/flume_exec-hdfs.1534983529181.txt
-rw-r--r--   1 kasi supergroup      19823 2018-08-23 02:18 /exec-hdfs/flume_exec-hdfs.1534983529182.txt
-rw-r--r--   1 kasi supergroup      19868 2018-08-23 02:18 /exec-hdfs/flume_exec-hdfs.1534983529183.txt
-rw-r--r--   1 kasi supergroup      19812 2018-08-23 02:18 /exec-hdfs/flume_exec-hdfs.1534983529184.txt
-rw-r--r--   1 kasi supergroup      20045 2018-08-23 02:18 /exec-hdfs/flume_exec-hdfs.1534983529185.txt
-rw-r--r--   1 kasi supergroup      19818 2018-08-23 02:18 /exec-hdfs/flume_exec-hdfs.1534983529186.txt
-rw-r--r--   1 kasi supergroup      19617 2018-08-23 02:18 /exec-hdfs/flume_exec-hdfs.1534983529187.txt
-rw-r--r--   1 kasi supergroup      19594 2018-08-23 02:18 /exec-hdfs/flume_exec-hdfs.1534983529188.txt
```

Validate Kafka Package Availability and Run Test:

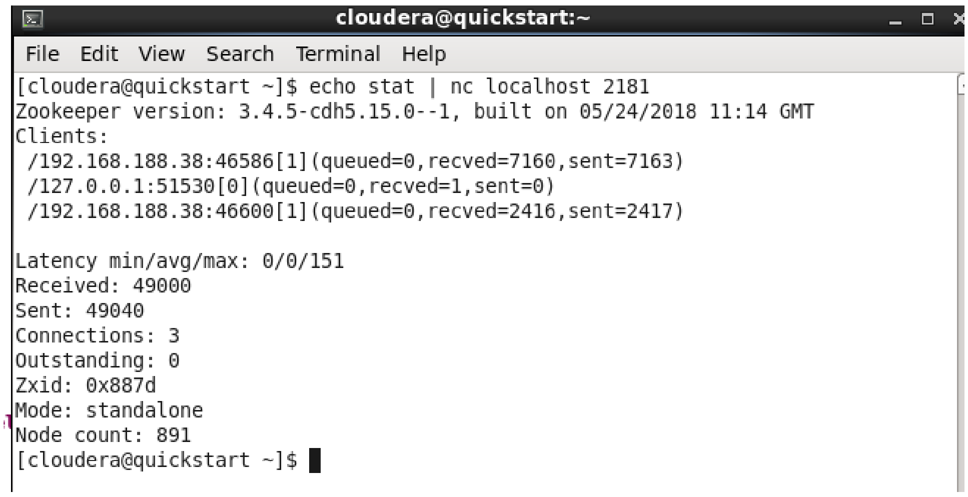
Kafka brokers (servers) depend on Zookeeper for membership & failure detection, leader election (deciding which broker is in charge of which partition).

Zookeeper determines the state. That means, it notices, if the Kafka Broker is alive, always when it regularly sends heartbeats requests.

Basically, Zookeeper in Kafka stores nodes and topic registries. It is possible to find there all available brokers in Kafka and, more precisely, which Kafka topics are held by each broker, under '/brokers/ids' they're stored.

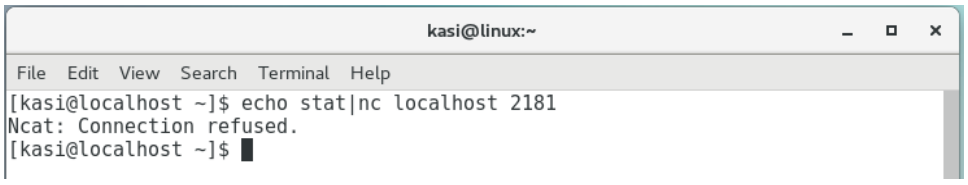
```
public void checkKafka() throws JSchException, IOException {
    ArrayList<String> output1 = runCommand("echo stat|nc localhost 2181");
    if (output1.isEmpty()) {
        System.out.println("Kafka server is not Running                  [FAILED]");
    } else {
        String string = output1.get(0);
        if (string.contains("Zookeeper version")) {
            ArrayList<String> output2 = null;
            try {
                output2 = runCommand("echo dump | nc localhost 2181 | grep brokers");
                if (output2.isEmpty()) {
                    System.out.println("Kafka server is not Running                  [FAILED]");
                } else if ((output2.get(0).contains("/brokers/ids/"))) {
                    System.out.println("Kafka server is Running                      [OK]");
                    ArrayList<String> kafkacheck = runCommand(
                        "${KAFKA_HOME}/bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic test");
                    boolean containsTopic = false;
                    for(String string1: kafkacheck) {
                        if ((string1.substring(0, string1.length())).contains("Created topic")
                            || (string1.substring(0, string1.length())).contains("already exists")) {
                        containsTopic = true;
                        break;
                        }
                    }
                    if(containsTopic = true) {
                        System.out.println("Kafka is working");
                    }else {
                        System.out.println("kafka is not working");
                    }
                } else {
                    System.out.println("Kafka server is not Running                  [FAILED]");
                }
            } catch (Exception e) {
                System.out.println("Kafka server is not Running                  [FAILED]");
            }
        } else {
            System.out.println("Kafka server is not Running                  [FAILED]");
        }

    }

}
```

The method checkKafka() initially runs the shell command 'echo stat | nc localhost 2181' and stores the output to ArrayList. This command validates the running state of the zookeeper



```
[cloudera@quickstart ~]$ echo stat | nc localhost 2181
Zookeeper version: 3.4.5-cdh5.15.0--1, built on 05/24/2018 11:14 GMT
Clients:
 /192.168.188.38:46586[1](queued=0,recved=7160,sent=7163)
 /127.0.0.1:51530[0](queued=0,recved=1,sent=0)
 /192.168.188.38:46600[1](queued=0,recved=2416,sent=2417)

Latency min/avg/max: 0/0/151
Received: 49000
Sent: 49040
Connections: 3
Outstanding: 0
Zxid: 0x887d
Mode: standalone
Node count: 891
[cloudera@quickstart ~]$
```

The Mode: standalone states that zookeeper is running on port 2181.
If the zookeeper is not running the command returns 'NCat: connection refused'

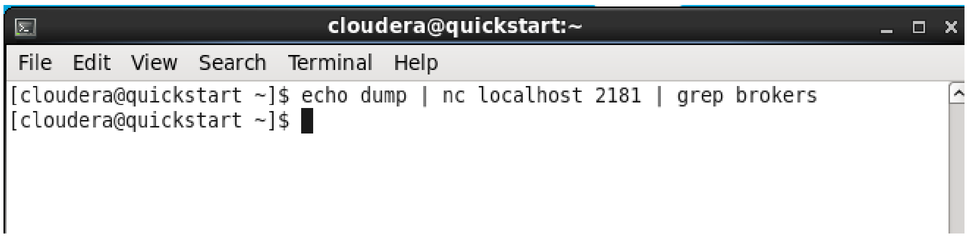


```
[kasi@localhost ~]$ echo stat|nc localhost 2181
Ncat: Connection refused.
[kasi@localhost ~]$
```

The method checkKafka() initially check's for zookeeper status. If zookeeper is running, next step is to check kafka cluster is running, if the kafka cluster is started then it is possible to find available brokers in kafka .since kafka brokers depends on zookeeper.

Command to get kafka broker list from zookeeper.

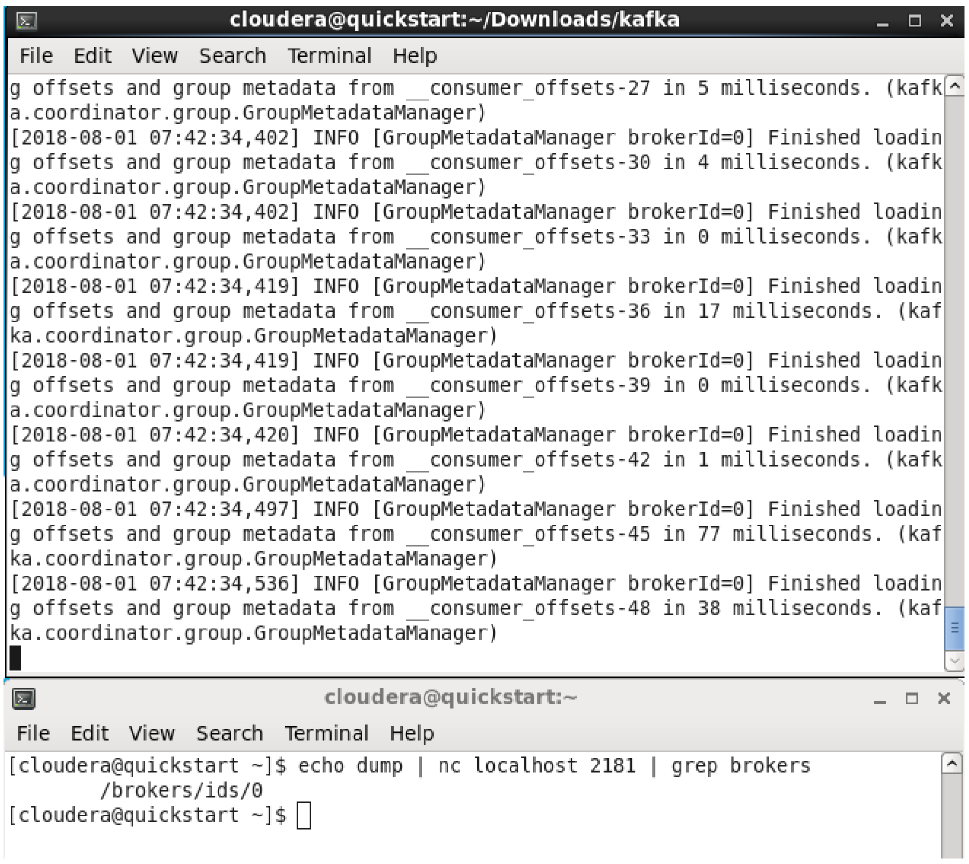
'echo dump | nc localhost 2181 | grep brokers'.



```
[cloudera@quickstart ~]$ echo dump | nc localhost 2181 | grep brokers
[cloudera@quickstart ~]$
```

The command returns nothing from the zookeeper because, no kafka server is running.

If the kafka server is running the command returns the broker(server) id.


cloudera@quickstart:~/Downloads/kafka
File Edit View Search Terminal Help

```
g offsets and group metadata from __consumer_offsets-27 in 5 milliseconds. (kafk
a.coordinator.group.GroupMetadataManager)
[2018-08-01 07:42:34,402] INFO [GroupMetadataManager brokerId=0] Finished loadin
g offsets and group metadata from __consumer_offsets-30 in 4 milliseconds. (kafk
a.coordinator.group.GroupMetadataManager)
[2018-08-01 07:42:34,402] INFO [GroupMetadataManager brokerId=0] Finished loadin
g offsets and group metadata from __consumer_offsets-33 in 0 milliseconds. (kafk
a.coordinator.group.GroupMetadataManager)
[2018-08-01 07:42:34,419] INFO [GroupMetadataManager brokerId=0] Finished loadin
g offsets and group metadata from __consumer_offsets-36 in 17 milliseconds. (kaf
ka.coordinator.group.GroupMetadataManager)
[2018-08-01 07:42:34,419] INFO [GroupMetadataManager brokerId=0] Finished loadin
g offsets and group metadata from __consumer_offsets-39 in 0 milliseconds. (kafk
a.coordinator.group.GroupMetadataManager)
[2018-08-01 07:42:34,420] INFO [GroupMetadataManager brokerId=0] Finished loadin
g offsets and group metadata from __consumer_offsets-42 in 1 milliseconds. (kafk
a.coordinator.group.GroupMetadataManager)
[2018-08-01 07:42:34,497] INFO [GroupMetadataManager brokerId=0] Finished loadin
g offsets and group metadata from __consumer_offsets-45 in 77 milliseconds. (kaf
ka.coordinator.group.GroupMetadataManager)
[2018-08-01 07:42:34,536] INFO [GroupMetadataManager brokerId=0] Finished loadin
g offsets and group metadata from __consumer_offsets-48 in 38 milliseconds. (kaf
ka.coordinator.group.GroupMetadataManager)
```

```
[cloudera@quickstart ~]$ echo dump | nc localhost 2181 | grep brokers
        /brokers/ids/0
[cloudera@quickstart ~]$
```

The method checkKafka() initially run's command to check zookeeper status. If the zookeeper is running the method again run's command to check kafka server is running.

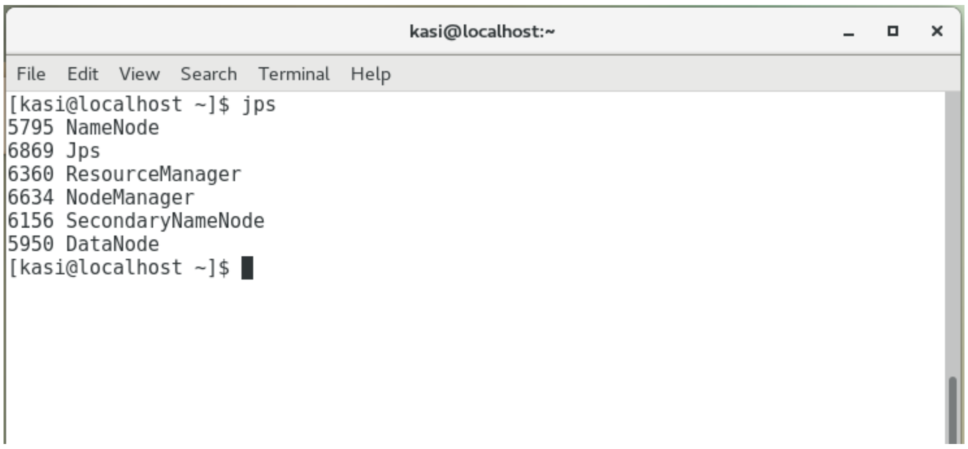
The method run's the command 'echo dump| nc localhost 2181| grep brokers' and returns the output, if the output is null then method returns 'kafka server is not running'. If output is not null and contains '/brokers/ids' then method returns 'kafka Server is running'.

Run Test:

If kafka server is running then create a topic, by running the command '${KAFKA_HOME}/bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic test' , if topic is created successfully then kafka is runnning, else kafka is not running.

Validate HDFS Availability and Run Test:

Running ‘jps’ command gives the status of ‘hadoop daemons’



```
[kasi@localhost ~]$ jps
5795 NameNode
6869 Jps
6360 ResourceManager
6634 NodeManager
6156 SecondaryNameNode
5950 DataNode
[kasi@localhost ~]$
```

```
public void checkHadoop() throws JSchException, IOException {
    ArrayList<String> output = runCommand("sudo jps");
    System.out.println(output);
    if (output.isEmpty()) {
        System.out.println("Hadoop File System Daemon's are not Running  [FAILED] ");
    } else {
        for (String string : output) {
            if ((string.substring(0, string.length())).contains("DataNode")) {
                System.out.println("Hadoop File System Daemon's are Running      [OK]");
                ArrayList<String> hdfscheck = runCommand("hadoop fs -ls /");
                for (String string1 : hdfscheck) {
                    if ((string1.substring(0, string1.length()))
                            .contains("localhost:8020 failed on connection exception:")) {
                        System.out.println("Hadoop Distributed file System is not working");
                        break;
                    } else if ((string1.substring(0, string1.length())).contains("Found")) {
                        System.out.println("Hadoop Distributed file System is working");
                        break;
                    }
                }
                break;
            } else if ((string.substring(0, string.length())).contains("Connection refused")) {
                System.out.println("Hadoop File System Daemon's are not Running  [FAILED] ");
                break;
            }
        }

    }

}
```

.

The method checkHadoop() runs the command 'jps'. If the output of the command is not null the method comapare's each line of the output with substring. If substring contains Data Node' then method prints 'Hadoop File System Daemon's are Running [OK]' and breaks out of the loop. If substring contains 'Connection refused' then method prints 'Hadoop File System Daemon's are not Running [FAILED]'.

Run Test:

If Hadoop daemons are running, run the command ‘hadoop fs –ls /’, if it found the items then hadoop file system is running, else hadoop file system is not running.

Validate Cluster Size:

```
JSONParser parser = new JSONParser();
Object obj = parser.parse(new FileReader("configuration.json"));
JSONObject jsonObject = (JSONObject) obj;
JSONArray hosts = (JSONArray) jsonObject.get("hosts");
System.out.println("Total Cluster size: " + hosts.size());
```

Read the configuration file(json format) and convert to json object , get the number of objects(hosts) in the json array , will give the size of the cluster.

Disconnect the Channel and Session:

```
public void disconnect() {
    if (session.isConnected()) {
        channel.disconnect();
        session.disconnect();
    }
}
```

The method disconnect, disconnects the channel and session for connected session.

Generate Cluster Status Report:

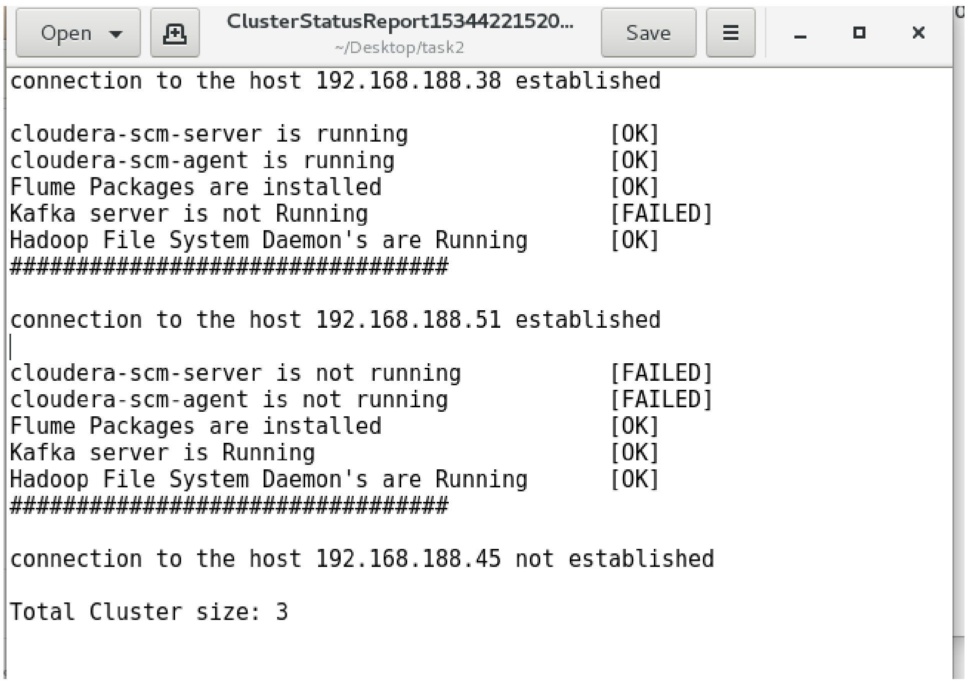
```
PrintStream out = new PrintStream(new FileOutputStream("ClusterStatusReport" + new Date().getTime() + ".txt"));
System.setOut(out);
```

FileOutputStream writes data to a file.

PrintStream sets functionality to FileOutputStream , here it flushes the output to a file.

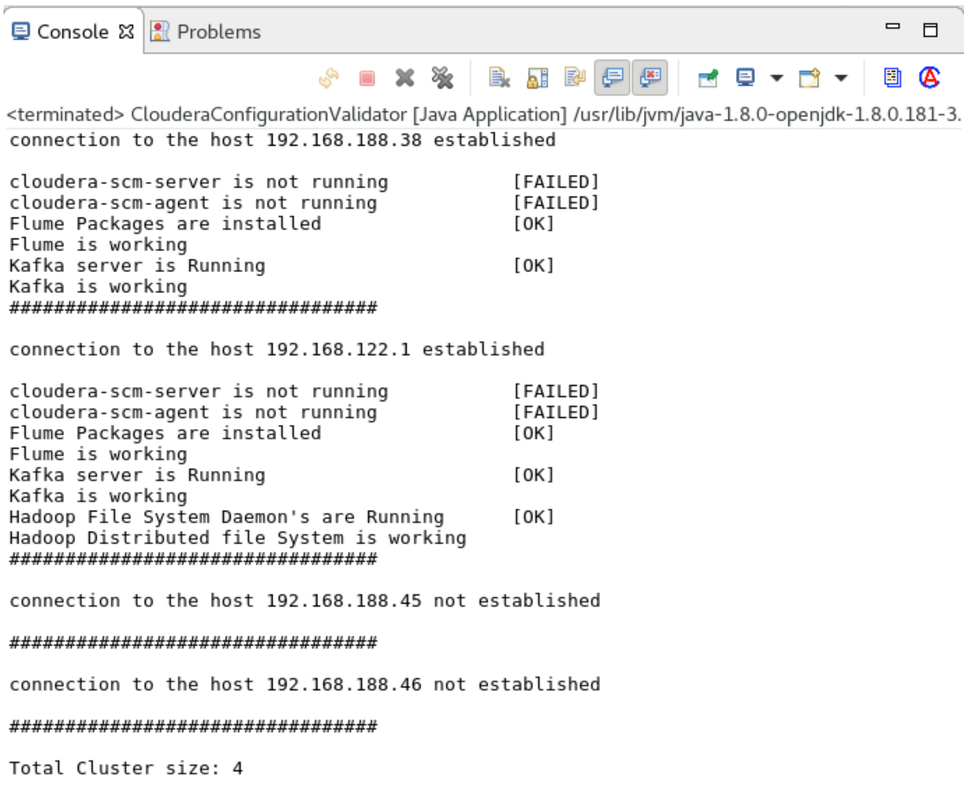
System.setOut(out) redirects the standard outputstream to a file.

The status of the services running in each machine will redirects into a ClusterStatusReport file.



```
connection to the host 192.168.188.38 established

cloudera-scm-server is running                [OK]
cloudera-scm-agent is running                 [OK]
Flume Packages are installed                  [OK]
Kafka server is not Running                   [FAILED]
Hadoop File System Daemon's are Running       [OK]
#################################

connection to the host 192.168.188.51 established

cloudera-scm-server is not running            [FAILED]
cloudera-scm-agent is not running             [FAILED]
Flume Packages are installed                  [OK]
Kafka server is Running                       [OK]
Hadoop File System Daemon's are Running       [OK]
#################################

connection to the host 192.168.188.45 not established

Total Cluster size: 3
```

When packages are available run and test the services



```
<terminated> ClouderaConfigurationValidator [Java Application] /usr/lib/jvm/java-1.8.0-openjdk-1.8.0.181-3.
connection to the host 192.168.188.38 established

cloudera-scm-server is not running            [FAILED]
cloudera-scm-agent is not running             [FAILED]
Flume Packages are installed                  [OK]
Flume is working
Kafka server is Running                       [OK]
Kafka is working
#################################

connection to the host 192.168.122.1 established

cloudera-scm-server is not running            [FAILED]
cloudera-scm-agent is not running             [FAILED]
Flume Packages are installed                  [OK]
Flume is working
Kafka server is Running                       [OK]
Kafka is working
Hadoop File System Daemon's are Running       [OK]
Hadoop Distributed file System is working
#################################

connection to the host 192.168.188.45 not established

#################################

connection to the host 192.168.188.46 not established

#################################

Total Cluster size: 4
```