# Infrastructure Required

**Operating System:**

* Non-Windows OS
* Passphraseless SSH\*

**Required Engines:**

**Functional Test Cases:** 2+ Hosts, 7+ Engine Instances

**Load Test Cases:** ?+ Hosts, ?+ Engines

# Demo Stacks

To facilitate testing and demonstration, a set of *demo stacks* have been implemented using the ANT framework. They are:

## hadoop-demo-psuedocluster

* 1 Psuedocluster
* 1 Datanode
* 1 Balancer
* 1 Tasktracker

## hadoop-demo-distributedcluster

* 1 Namenode
* 1 Secondary Namenode
* 1-3 Datanodes
* 1 Balancer
* 1 Jobtracker
* 1-3 Tasktrackers

## hadoop-demo-hdfscluster

* 1 Namenode
* 2-5 Datanodes
* 1 Balancer

# Test Cases

## Core Functionality

**Stack(s):** Psuedocluster, Distributed Cluster

**Execution:**

1. Start Stack
   1. Assert – All components start on the minimum number of engines configured
   2. Assert – Hadoop Logs created\*
   3. Assert – Rackawareness Log Created\*
   4. Assert – All Datanodes are assigned to /default/rack\*
   5. Assert – Web UI is accessible for each component\*
   6. Assert – Can add files to HDFS and retrieve them\*
   7. Assert – Can run a MapReduce job and view the results\*
   8. Assert – No exceptions or SEVERE errors in engine log or application logs
2. Stop Stack
   1. Assert – All components stop.
   2. Assert – No exceptions or SEVERE errors in engine log or application logs

## Rack Awareness

**Stack(s):** Psuedocluster

**Common Activities:**

* Asserting Datanode and Tasktracker Rack Values – Examine the Namenode log and search for lines containing the phrase “Adding node”.

### DEFAULT Policy

**Set-up:**

1. On the psuedocluster component, change value of RCV *hadoop\_enabler\_RACKAWARENESS\_DEFAULT* to “ /dc-default/rack-default””

**Execution:**

1. Start Stack
   1. Assert – Assert all datanodes and tasktrackers (including those in pseudo cluster have a Rack Value of “ /dc-default/rack-default”

### CONFIG\_FILE Policy

**Set-up:**

1. On the psuedocluster component, change value of RCV *hadoop\_enabler\_RACKAWARENESS\_POLICY* to “CONFIG\_FILE”
2. On the psuedocluster component, , add the content file hadoop-1.0.4/conf/hadoop\_enabler\_RACKAWARENESS.cfg with the following content below.
   * Replace [enghost1-ip] with the ip address of one of the engine host machines.
   * Replace [enghost2-regex] with a regular expression that mat ches the IP host of the other host. (e.g. 192.168.\*)

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

# Hadoop Rackaware Configuraiton

#

# This file is used to determine the rack value of a node. To

# use this file, the Rackawareness Policy must be set to

# 'CONFIG\_FILE'.

#

# Format is:

# [reg expresion] [rack value]

#

# Where:

# [reg expression] = A regular expression to compare to the IP

# Addresses of your slave nodes

# [rack value] = The rack value to be assigned for any matches

#

# If multiple matches exist, the first one located, starting at the

# top of the file will be used.

#

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

[enghost1-ip] /dc1/rack1

[enghost2-regex] /dc2/rack2

**Execution:**

1. Start Stack
   1. Assert – Assert all datanodes and tasktrackers instantiated on Engine Host 1 have a Rack Value of “/dc1/rack1”
   2. Assert – Assert all datanodes and tasktrackers instantiated on Engine Host 2 have a Rack Value of “/dc2/rack2”

### ENVIRON\_VARIABLE Policy

**Steps**

***Set-up:***

1. On the psuedocluster component, change value of RCV *hadoop\_enabler\_RACKAWARENESS\_POLICY* to “ENV\_VARIABLE”
2. On Engine Host 1 set an environment variable\* RACK=/dc1/rack2
3. On Engine Host 2 set an environment variable\* RACK=/dc2/rack2

\* The environment variable needs to be reset in the Silver Fabric’s user profile or in such a manner that it is set when an SSH session is initiated from another host.

***Execution:***

1. Start Stack
   1. Assert – Assert all datanodes and tasktrackers instantiated on Engine Host 1 have a Rack Value of “/dc1/rack1”
   2. Assert – Assert all datanodes and tasktrackers instantiated on Engine Host 2 have a Rack Value of “/dc2/rack2”

## Node Recovery

**Stack(s):** Distributedcluster

**Steps:**

***Set-up***

1. Add Resource Policy to Namenode component

Hostname = [Engine Host 1’s IP]

1. Add RCV to Namenode component

${hadoop\_enabler\_NAMENODE\_NAME\_DIR\_LOCAL} = [persistent location]

Where:

[persistent location] = A directory location that is accessible to Engine Host 1 and will not be deleted during a component restart (e.g. /tmp/dfs/name).

1. Verify that the directory defined in the step above does not yet exist.
2. Increase the Datanode components minimum number of instances to 2
3. Start Stack
4. Enter HDFS Client command\*

> bin/hadoop fs –put conf input

1. Enter HDFS Client command\*:

> bin/hadoop fs –ls /input

and note the file creation times

***Name Node Failures***

1. Kill the Namenode process.

From a terminal on the engine host where the name node component is running:

> kill -9 `ps ax | grep [p]roc\_namenode | awk ‘{print $1}’

1. Enter HDFS Client command\*:

> bin/hadoop fs –ls /input

* 1. Assert – The files are visible and the creation time of the files has not changed.

1. Kill the namenode engine instance

From a terminal on the engine host where the name node component is running:

> kill -9 `ps ax | grep [n]vok | grep “*n* -Dj”| awk ‘{print $1}’

Where:

*n* = The number of the engine instance where the component is running.

* 1. Assert – Namenode component restarts on same engine instance

1. View Small Dataset in HDFS
   1. Assert - The files are visible and the creation time of the files have not changed.

***Datanode Failures***

1. Kill the datanode process.

From a terminal on an engine host where the component is running:

> kill -9 `ps ax | grep [d]atanode | grep *working-directory* | awk ‘{print $1}’

Where:

*working-directory* = working directory for engine running the component.

* 1. Assert – Datanode component restarts

1. Enter HDFS Client command\*:

> bin/hadoop fs –ls /input

* 1. Assert – The files are visible and the creation time of the files have not changed.

1. Kill the datanode engine instance

From a terminal on an engine host where the component is running:

> kill -9 `ps ax | grep [n]vok | grep “*n* -Dj”| awk ‘{print $1}’

Where:

*n* = The number of the engine instance where the component is running.

* 1. Assert – Namenode component restarts on same engine instance

1. View Small Dataset to HDFS
   1. Assert – The files are visible and the creation time of the has not changed.

***Jobtracker Failures***

1. Kill the Jobtracker process

From a terminal on an engine host where the component is running:

> kill -9 `ps ax | grep [j]obtracker | awk ‘{print $1}’

* 1. Assert – Jobtracker component restarts

1. Kill the Jobtracker engine instance

From a terminal on an engine host where the component is running:

> kill -9 `ps ax | grep [n]vok | grep “*n* -Dj”| awk ‘{print $1}’

Where:

*n* = The number of the engine instance where the component is running.

* 1. Assert – Jobtracker component restarts

***Tasktracker Failures***

1. Kill the Tasktracker process

From a terminal on an engine host where the component is running:

> kill -9 `ps ax | grep [t]asktracker | awk ‘{print $1}’

* 1. Assert – Tasktracker component restarts

1. Kill the Tasktracker engine instance

From a terminal on an engine host where the component is running:

> kill -9 `ps ax | grep [n]vok | grep “*n* -Dj”| awk ‘{print $1}’

Where:

*n* = The number of the engine instance where the component is running.

* 1. Assert – Tasktracker component restarts

***Secondary Namenode Failures***

1. Kill the Secondary Namenode process

From a terminal on an engine host where the component is running:

> kill -9 `ps ax | grep [p]roc\_secondarynamenode | awk ‘{print $1}’

* 1. Assert – Secondary Namenode component restarts

1. Kill the Secondary Namenode engine instance

From a terminal on an engine host where the component is running:

> kill -9 `ps ax | grep [n]vok | grep “*n* -Dj”| awk ‘{print $1}’

Where:

*n* = The number of the engine instance where the component is running.

* 1. Assert – Secondary Namenode component restarts

### Datanode Decommissioning

**Requirements:**

* Use Hadoop’s Decommissioning feature when shutting down (or restarting) a subset of datanodes.
  + Note: Decommissioning enables a graceful shutdown of datanodes. It involves:
    - 1 ) Notifying theNamenode that the Datanode is to be shutdown
    - 2) Allowing the Namenode to automatically move the data off of the targeted Datanode and shutdown the datanode remotely
* Shutdown the datanodes immediately (i.e. do *not* use the Decommissioning feature) When the the entire HDFS cluster is being stopped.
  + Note: A shutdown of the entire HDFS Cluster is to be assumed whenever both the Namenode component and the current Datanode component have an expected engine count of 0.

**Stack(s):** HDFScluster

**Steps:**

***Set-up***

1. Start Stack
2. Via the console, decrease the Datanode components minimum number of instances to 1
   1. Assert – SF UI indicates Datanode beginning to shutdown
   2. Assert – Datanode Engine log indicates shutdown waiting for process to be terminated
   3. Assert – Namenode Engine log indicates decommission request processed successfully
   4. Asert – Datanode Engine log indicates process terminated and shutdown completed without errors
   5. Assert – SF UI indicates that only one Datanode is still running (We will refer to this as Datanode #1).
3. Via a terminal on any host with a hadoop component, load data into HDFS.

> cd [Engine Work Directory]/fabric/hadoop-1.0.4

> bin/hadoop fs –put conf input

* 1. Assert – Hadoop meta data successfully created.

The result should look roughly like the following:

> bin/hadoop fs –ls input

-rw-r--r-- 3 demouser supergroup 7457 2013-03-29 13:08 /user/demouser/input/capacity-scheduler.xml

-rw-r--r-- 3 demouser supergroup 535 2013-03-29 13:08 /user/demouser/input/configuration.xsl

-rw-r--r-- 3 demouser supergroup 461 2013-03-29 13:08 /user/demouser/input/core-site.xml

-rw-r--r-- 3 demouser supergroup 327 2013-03-29 13:08 /user/demouser/input/fair-scheduler.xml

-rw-r--r-- 3 demouser supergroup 3194 2013-03-29 13:08 /user/demouser/input/hadoop-env.sh

-rw-r--r-- 3 demouser supergroup 858 2013-03-29 13:08 /user/demouser/input/hadoop-metrics2.properties

-rw-r--r-- 3 demouser supergroup 4644 2013-03-29 13:08 /user/demouser/input/hadoop-policy.xml

-rw-r--r-- 3 demouser supergroup 558 2013-03-29 13:08 /user/demouser/input/hadoop\_enabler\_RACKAWARENESS.cfg

-rw-r--r-- 3 demouser supergroup 1005 2013-03-29 13:08 /user/demouser/input/hdfs-site.xml

-rw-r--r-- 3 demouser supergroup 4441 2013-03-29 13:08 /user/demouser/input/log4j.properties

-rw-r--r-- 3 demouser supergroup 2033 2013-03-29 13:08 /user/demouser/input/mapred-queue-acls.xml

-rw-r--r-- 3 demouser supergroup 490 2013-03-29 13:08 /user/demouser/input/mapred-site.xml

-rw-r--r-- 3 demouser supergroup 10 2013-03-29 13:08 /user/demouser/input/masters

-rw-r--r-- 3 demouser supergroup 10 2013-03-29 13:08 /user/demouser/input/slaves

-rw-r--r-- 3 demouser supergroup 1243 2013-03-29 13:08 /user/demouser/input/ssl-client.xml.example

-rw-r--r-- 3 demouser supergroup 1195 2013-03-29 13:08 /user/demouser/input/ssl-server.xml.example

-rw-r--r-- 3 demouser supergroup 382 2013-03-29 13:08 /user/demouser/input/taskcontroller.cfg

* 1. Assert – Loaded files contain data.

> bin/hadoop fs -cat input/masters

localhost

1. Via the console, increase the Datanode components minimum number of instances to 2
   1. Assert – SF UI indicates a second Datanode (we will refer to this as Datanode #2) has been started.
2. Via the SF UI, Assert – Restart the component that was previously identified as Datanode #1.
   1. Assert – SF UI indicates Datanode beginning to shutdown
   2. Assert – Datanode Engine log indicates shutdown waiting for process to be terminated
   3. Assert – Namenode Engine log indicates decommission request processed successfully
   4. Asert – Datanode Engine log indicates process terminated and shutdown completed without errors
   5. Assert – Hadoop meta data remains intact and unchanged.
      1. > bin/hadoop fs –ls input

The result should look exactly the same as the results from Assertion 3.1

* 1. Assert – Loaded files still contain data.

> bin/hadoop fs -cat input/masters

localhost

* 1. Assert – SF UI indicates Datanode has restarted.

Note: It does not matter to shutdown

1. Enter HDFS Client command\*:

> bin/hadoop fs –ls /input

* 1. Assert - returns a listing of all files and no errors.

### Balancer

**Requirements:**

* Run Balancer when a new datanode is added to the cluster
* If Datanode is added while Balancer is running, stop and retstart Balancer

**Stack(s):** HDFScluster

**Steps:**

1. Start Stack
   1. Assert - Via the Datanode Logs, A Balancer request is generated after the Datanode has started.
   2. Assert – Via the Balancer Log, Balancer runs after an intial delay (because the HDFS cluster is just starting and we want to wait for the intial Datanodes to join)
   3. Assert – Via the Balancer Log, After the Balancer completes the Balancer request is deleted.
2. Via the SF Console, increase the number of datanodes by 1.
   1. Assert - Via the Datanode Logs, A Balancer request is generated after the Datanode has started.
   2. Assert – Via the Balancer Log, Balancer run is started
   3. Assert – Via the Balancer Log, After the Balancer completes the Balancer request is deleted.
3. Via a terminal on any host with a hadoop component, load data into HDFS.

> cd [Engine Work Directory]/fabric/hadoop-1.0.4

> bin/hadoop fs –put src input

1. Via the SF Console, increase the number of datanodes by 1.
   1. Assert – Via the Balancer Log, Balancer run is started
2. (Before the Balancer run completes) Via the SF Console, increase the number of datanodes by 1.
   1. Assert – Via the Balancer Log, the current Balancer run is stopped and a new one is started within 30 seconds.