CS300 Couchbase NoSQL Server Administration

Lab 4 Exercise Manual

Release: 4.5

Revised: Sept 1st, 2016

Lab #4: Gracefully removing nodes, replication, failover, MISC commands

Objective: This 90 min lab will cover some of the most important and common administrative features of couchbase, such as removing nodes, replication concepts, failover and rebalancing.

Overview: **The following high-level steps are involved in this lab:**

- Configure the linux .bash_profile for Couchbase commands
- Write 100,000 items to the 6-node Couchbase cluster using cbworkloadgen
- Change the workload balance between reads vs. writes in cbworkloadgen
- Increase the # of threads in cbworkloadgen
- Use the Web UI's performance metrics, to determine whether all nodes in the cluster are getting a balanced amount of i/o (with no skews towards certain nodes)
- Display individualized metrics for each node in the Web UI
- Delete Couchbase buckets
- Gracefully decommission and re-commission a node from the cluster
- Fail over a node in the cluster
- Replica management: Write 10 keys to Couchbase and track down which vBucket each key landed in using the 'cbc hash' command
- Use the REST API to check auto-failover settings and disable auto-failover

Write data to the 6-node Couchbase Cluster and benchmark each node:

Now that the 6-node cluster is set up, let's try writing 100,000 items from the App Server to the 6-node cluster.

Switch over to the Black App Client(AppServer) PuTTY/Terminal shell and first edit the Linux PATH so it's convenient in the future to run Couchbase commands without giving the full directory path of the command.

Add the /opt/couchbase/bin directory to your Linux PATH so that you can run the couchbase-cli tool (and other tools) by simply typing 'couchbase-cli' without providing the full directory path into the command.

Edit the .bash_profile file:

```
[ec2-user@AppServer ~]$ cd ~
[ec2-user@AppServer ~]$ vi .bash_profile
```

Line 10 should currently show the following:

```
PATH=$PATH:$HOME/.local/bin:$HOME/bin
```

Edit line 10 by appending the couchbase tools path to the end of the line, like so:

```
PATH=$PATH:$HOME/bin:/opt/couchbase/bin
```

Save and quit the vi or nano session.

Source the .bash_profile file so that the changes you made take effect in the current bash session:

```
[ec2-user@AppServer ~]$ source ~/.bash_profile
```

If you are not comfortable using the vi editor you can enter the following at the command line for this putty session.

 # PATH=$PATH:/opt/couchbase/bin
 # export PATH

The next command will insert 100,000 items of size 10 bytes with 95% of the workload set to writes. This is the same test we did from our AppServer with just a 1-node Couchbase cluster, which resulted in about 151,400 operations per second (your mileage may have varied depending on cloud conditions).

In the next command, change the IP address to your 1st VM's public hostname:

```
[ec2-user@AppServer ~]$ cbworkloadgen -n ec2-54-85-43-x.compute-1.amazonaws.com:8091 -u Administrator -p couchbase -i 100000 -r .95 -s 10
  [####################] 100.0% (199999/estimated 200000 msgs)
bucket: default, msgs transferred...
       :               total |       last |    per sec
 byte  :             1999990 |    1999990 |   321006.5
done
```

Notice that in the above output, the cluster is handling 321006.5 bytes of I/O per second. Try running the same command with the '–v' option added to the end of the command for verbose mode:

```
[ec2-user@AppServer ~]$ cbworkloadgen -n ec2-54-85-43-128.compute-1.amazonaws.com:8091 -u Administrator -p couchbase -i 100000 -r .95 -s 10 -v
2016-05-11 12:41:04,230: mt cbworkloadgen...
2016-05-11 12:41:04,230: mt  source : gen:json=0,max-items=100000,low-
compression=False,prefix=pymc,ratio-sets=0.95,exit-after-creates=1,min-value-size=10
2016-05-11 12:41:04,230: mt  sink   : http://ec2-54-152-187-112.compute-1.amazonaws.com:8091
2016-05-11 12:41:04,230: mt  opts   : {'username': '<xxx>', 'destination_vbucket_state':
'active', 'verbose': 1, 'extra': {'max_retry': 10.0, 'rehash': 0.0, 'dcp_consumer_queue_length':
1000.0, 'data_only': 0.0, 'uncompress': 0.0, 'nmv_retry': 1.0, 'cbb_max_mb': 100000.0, 'report':
5.0, 'mcd_compatible': 1.0, 'try_xwm': 1.0, 'backoff_cap': 0.1, 'batch_max_bytes': 400000.0,
'report_full': 2000.0, 'flow_control': 1.0, 'batch_max_size': 1000.0, 'seqno': 0.0,
'design_doc_only': 0.0, 'recv_min_bytes': 4096.0}, 'ssl': False, 'threads': 1, 'key': None,
'password': '<xxx>', 'id': None, 'destination_operation': None, 'source_vbucket_state': 'active',
'silent': False, 'dry_run': False, 'single_node': False, 'bucket_destination': 'default',
'vbucket_list': None, 'bucket_source': None}
2016-05-11 12:41:04,280: mt bucket: default
2016-05-11 12:41:07,611: w0   source : gen:json=0,max-items=100000,low-
compression=False,prefix=pymc,ratio-sets=0.95,exit-after-creates=1,min-value-size=10(default@N/A-
0)
2016-05-11 12:41:07,612: w0   sink   : http://ec2-54-152-187-112.compute-
1.amazonaws.com:8091(default@N/A-0)
2016-05-11 12:41:07,612: w0          :                total |       last |    per sec
2016-05-11 12:41:07,612: w0    batch :                  106 |        106 |       31.8
2016-05-11 12:41:07,612: w0    byte  :              1052640 |    1052640 |   315972.4
2016-05-11 12:41:07,612: w0    msg   :               105264 |     105264 |    31597.2
  [####################] 100.0% (105264/estimated 105263 msgs)
bucket: default, msgs transferred...
       :                total |       last |    per sec
 batch :                  106 |        106 |       30.9
 byte  :              1052640 |    1052640 |   307201.0
 msg   :               105264 |     105264 |    30720.1
done
```

You can see at the end of the above output, that the App Server was sending 30720.1 messages per second into the cluster for a total of about 105263 messages.

We can also redo the test with purely 100% writes (notice that we changed the –r to 1):

```
[ec2-user@AppServer ~]$ cbworkloadgen -n ec2-54-85-43-x.compute-1.amazonaws.com:8091 -u Administrator -p couchbase -i 100000 -r 1 -s 10 -v
<output truncated>
[####################] 100.0% (100000/estimated 100000 msgs)
bucket: default, msgs transferred...
       :                total |        last |    per sec
 batch :                  100 |         100 |       29.1
 byte  :              1000000 |     1000000 |   291458.4
 msg   :               100000 |      100000 |    29145.8
done
```

When running purely write workloads, the cluster is handling about **29145.8** write operations per second.

Note : this will vary depending on number of Vcpu's

Try running the same command as above with an increased number of threads to 10 on the App Server to see if we can get better performance:

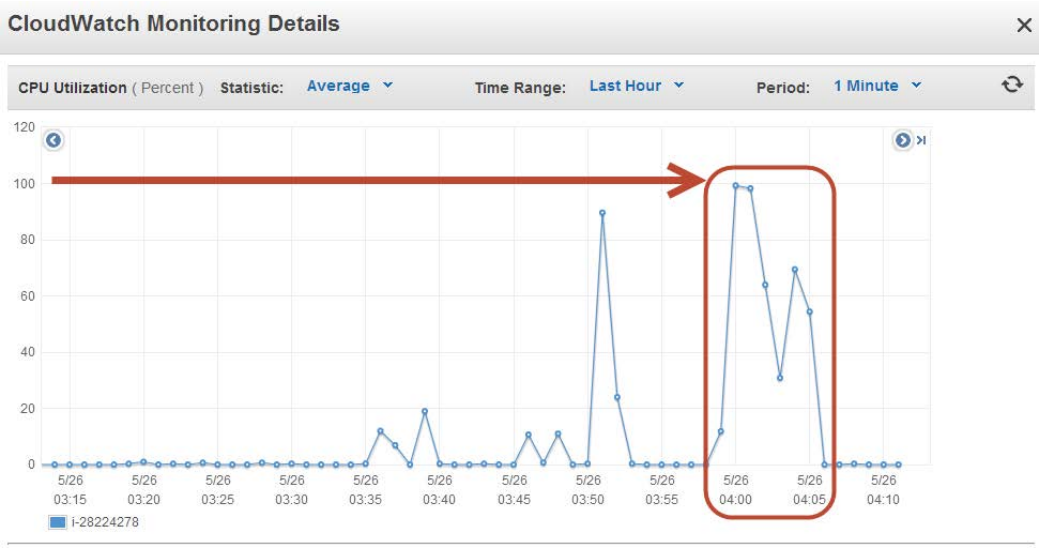
```
[ec2-user@AppServer ~]$ cbworkloadgen -n ec2-54-85-43-128.compute-1.amazonaws.com:8091 -u Administrator -p couchbase -i 100000 -r 1 -s 10 -t 10 -v
2015-08-06 16:49:34,469: mt cbworkloadgen...
<output truncated>
2015-08-06 16:50:14,478: w0   sink   : http://ec2-52-6-74-39.compute-1.amazonaws.com:8091(default@N/A-1)
<output truncated>
2015-08-06 16:50:14,744: w2   source : gen:json=0,max-items=100000,low-compression=False,prefix=pymc,ratio-sets=1.0,exit-after-creates=1,min-value-size=10(default@N/A-2)
2015-08-06 16:50:14,744: w2   sink   : http://ec2-52-6-74-39.compute-1.amazonaws.com:8091(default@N/A-2)
2015-08-06 16:50:14,745: w2         :                total |        last |    per sec
2015-08-06 16:50:14,746: w2    batch :                  100 |         100 |        2.5
2015-08-06 16:50:14,746: w2    byte  :              1000000 |     1000000 |    24863.1
2015-08-06 16:50:14,746: w2    msg   :               100000 |      100000 |     2486.3
2015-08-06 16:50:14,780: w5   source : gen:json=0,max-items=100000,low-compression=False,prefix=pymc,ratio-sets=1.0,exit-after-creates=1,min-value-size=10(default@N/A-6)
2015-08-06 16:50:14,781: w5   sink   : http://ec2-52-6-74-39.compute-1.amazonaws.com:8091(default@N/A-6)
2015-08-06 16:50:14,781: w5         :                total |        last |    per sec
2015-08-06 16:50:14,781: w5    batch :                  100 |         100 |        2.5
2015-08-06 16:50:14,781: w5    byte  :              1000000 |     1000000 |    24843.1
2015-08-06 16:50:14,782: w5    msg   :               100000 |      100000 |     2484.3
2015-08-06 16:50:14,818: w4   source : gen:json=0,max-items=100000,low-compression=False,prefix=pymc,ratio-sets=1.0,exit-after-creates=1,min-value-size=10(default@N/A-4)
<output truncated>
-size=10(default@N/A-0)
2015-08-06 16:50:14,907: w1   sink   : http://ec2-52-6-74-39.compute-
<output truncated>
```

```
2015-08-06 16:50:14,910: w9   source : gen:json=0,max-items=100000,low-
compression=False,prefix=pymc,ratio-sets=1.0,exit-after-creates=1,min-value-size=10(default@N/A-
5)
2015-08-06 16:50:14,910: w9   sink   : http://ec2-52-6-74-39.compute-
1.amazonaws.com:8091(default@N/A-5)
2015-08-06 16:50:14,910: w9          :                total |       last |    per sec
2015-08-06 16:50:14,910: w9    batch :                  100 |        100 |        2.5
2015-08-06 16:50:14,911: w9    byte  :              1000000 |    1000000 |    24763.8
2015-08-06 16:50:14,911: w9    msg   :               100000 |     100000 |     2476.4
  [####################] 99.9% (999000/estimated 1000000 msgs)
bucket: default, msgs transferred...
       :                total |       last |    per sec
 batch :                 1000 |       1000 |       24.4
 byte  :             10000000 |   10000000 |   244138.6
 msg   :              1000000 |    1000000 |    24413.9
done
```

The above output will print out the statistics from each thread individually, so above you can see the 2nd (w2) and 5th (w5) threads.

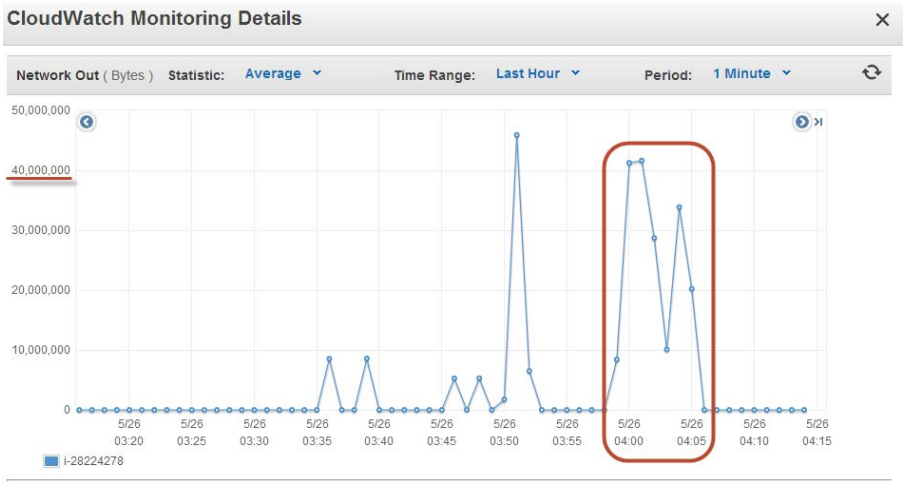
However, notice that the overall write throughput was actually reduced down to **24413.9** per second. Increasing the number of threads beyond a certain amount doesn't help performance, but actually hinders it.

The following screenshot was taken from Amazon CloudWatch dashboard and shows the CPU utilization of the App Server while the above command that generated 10 writer threads is running (see red circled area of the graph). *(Note that students will not have access to the CloudWatch metrics, however in the performance lab, different tools will be used to capture similar metrics.)*



Notice above that the CPU of the App Server is approaching 100% utilization when 10 threads are spawned.

The next screenshot from CloudWatch shows the Network out traffic in bytes on the App Server during the 10 thread write test. At its peak the App Server pushed 40,000,000 bytes or 305 Megabits of traffic (which is 38 Megabytes of traffic) out to the cluster.



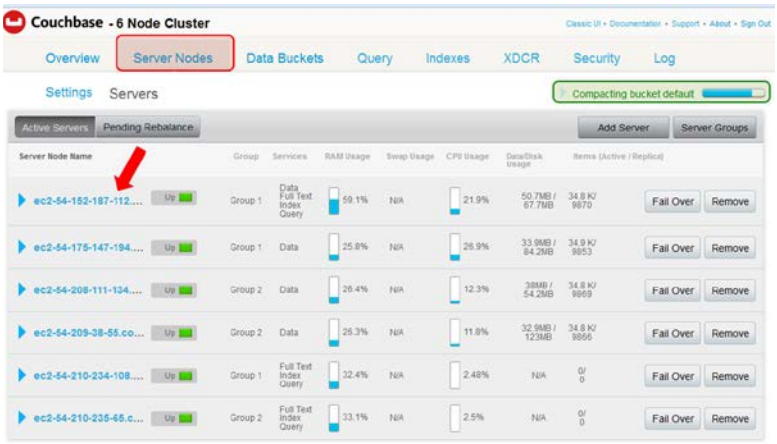
Is there a way to find out how those roughly 13,000 - 14,000 writes per second are being distributed across the 4 nodes? It would be good to investigate whether any of the servers are getting a skewed amount of the incoming writes, so that some nodes are working harder than others.

First, start a continuous stream of 100% writes from the App Server to the 6-node cluster. Use just 3 threads and write 10 byte sized items. The –l option in the command instructs the work load generator to loop forever until interrupted by the user:

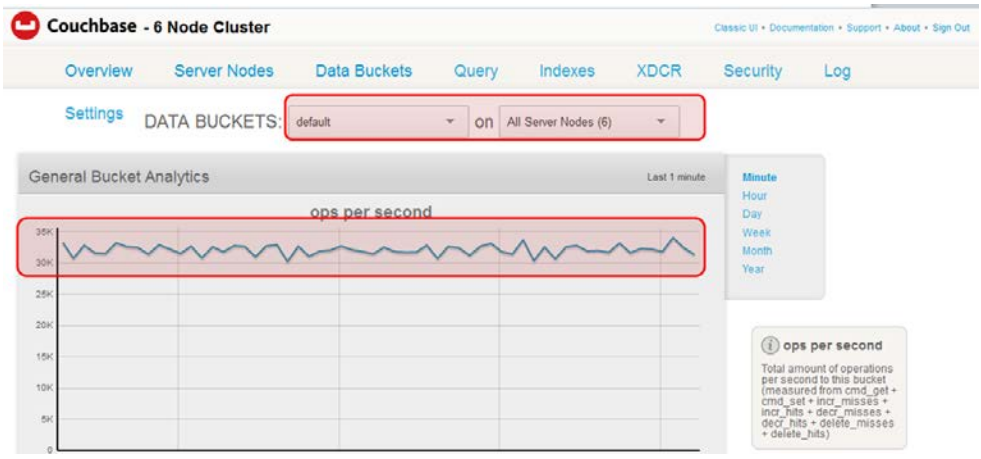
```
[ec2-user@AppServer ~]$ cbworkloadgen -n ec2-54-85-43-x.compute-
1.amazonaws.com:8091 -u Administrator -p couchbase -i 100000 -r 1 -s
10 -t 3 -l -v
.....................................................................
.....................
```

While the above command is running in a loop, switch over to the Couchbase Web UI to look at some statistics to answer our question.

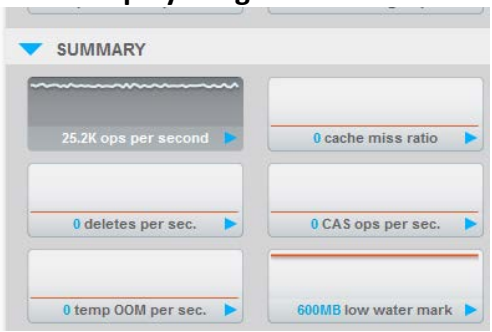
Click on Server Nodes and the click the hostname of the first server that host a data service to launch the graphs page:



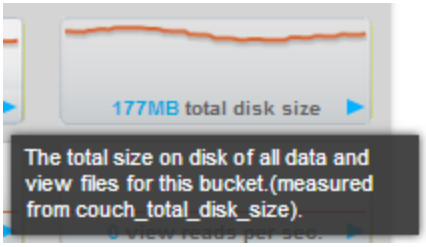
On the next page, choose the default data bucket on All Server Nodes:
You should be seeing approximately 30,000 operations per second:

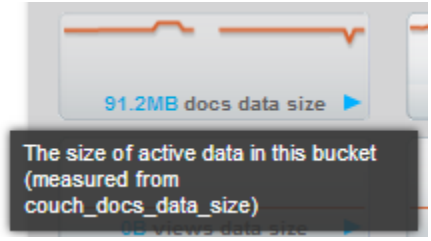


Scroll down and expand the SUMMARY section to notice the same 30,000 ops per second metric displayed again:



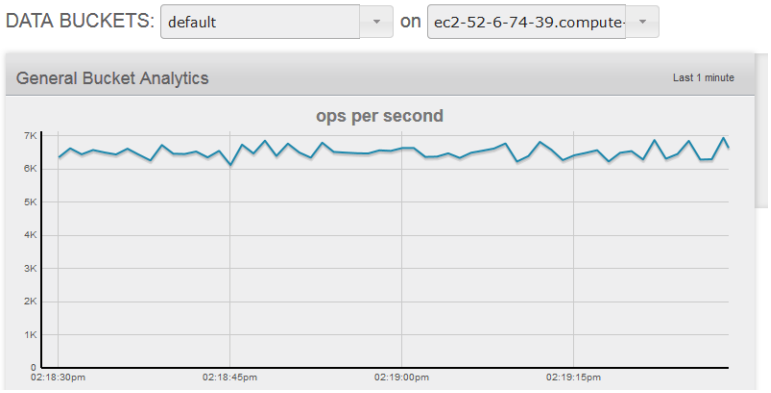
In the SUMMARY section, notice that the total disk size chart and docs data size charts show how much space the default bucket is taking up on disk:
Your size may vary.





The total docs size is relatively low at 91 MB even though we are repeatedly inserting 30,000 items per second at a rate of about 40 MB per second. This is because the workload generator is re-writing 100,000 keys of size 10 bytes over and over again and not necessarily inserting fresh, new keys. Keep in mind that 10 bytes for each key multiplied by 100,000 is only 976 Kilobytes.

Scroll back up on this page and choose one of the 4 data service nodes in the cluster to display individualized metrics for:

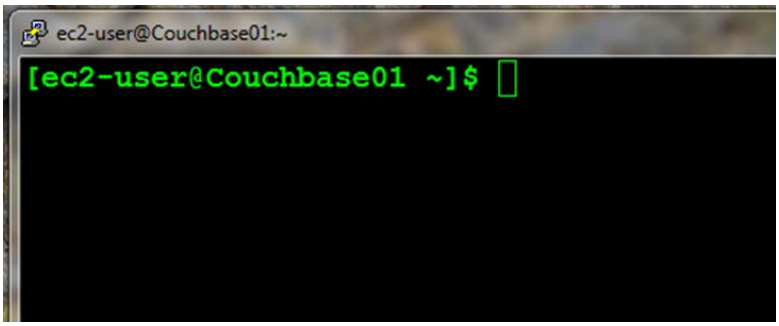


You should see roughly 7,000-8000 write operations being conducted on each of the 4 nodes. 7-8,000 ops per second multiplied by 4 nodes equals roughly 30,000 ops per second.

In the drop down, repeat the above step to verify that the other 3 nodes are also receiving about 7,000-8,000 ops per sec.

[Screenshots from the other 3 nodes not shown in lab]

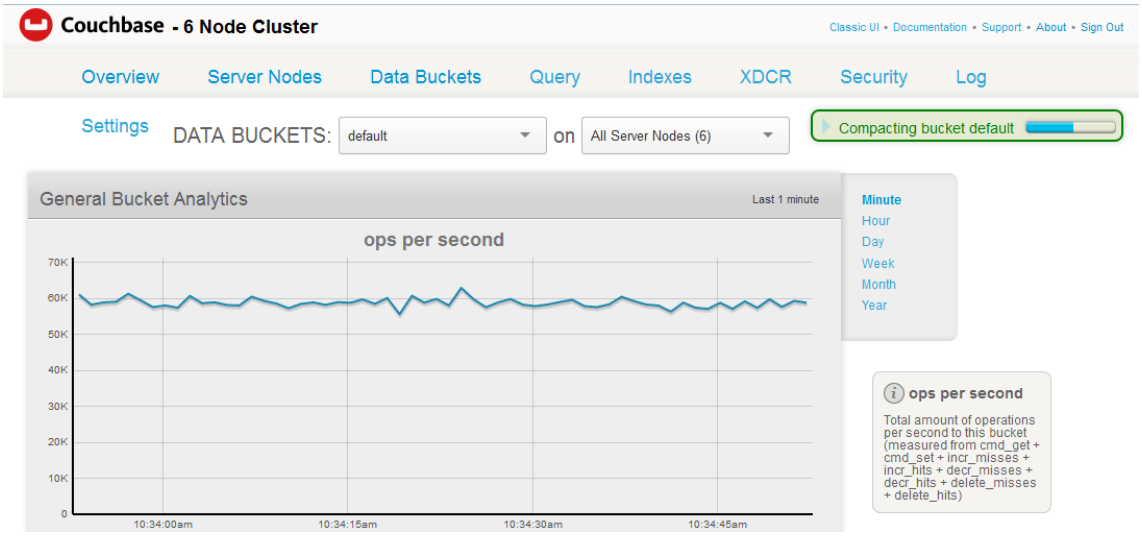
Switch to the 1st node (Couchbase01) in the cluster and run the workload generator in a loop.



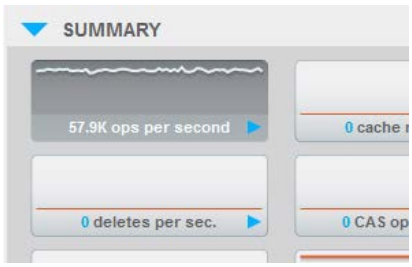
We will use the 1st Couchbase cluster node in a dual role to generate more writes for the cluster, and see how many more writes we can push into it. Perhaps the bottleneck currently is the App Server itself which cannot push any more than 30,000 items per second from its NIC in the cloud.
Run the workload generator from the 1st node with the exact same parameters as we ran it in a loop on the App Server:

```
[ec2-user@Couchbase01 ~]$  cbworkloadgen -n ec2-54-85-43-128.compute-
1.amazonaws.com:8091 -u Administrator -p couchbase -i 100000 -r 1 -s
10 -t 3 -l -v
.....
```

From the Web UI, you should now see the cluster handling about 50,000 – 60,000 write operations per second across 4 data service nodes. You will also occasionally see a message that the 'default' bucket is being compacted:

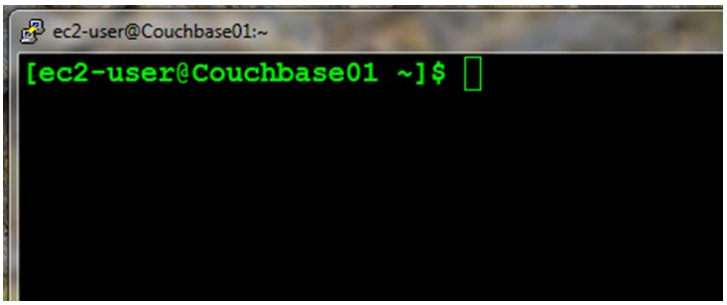


Anyway, the SUMMARY section will show an exact count such as ~57.9K ops per second:



Using two machines to push the writes demonstrates that the App Server by itself is not capable of saturating the cluster with the amount of traffic that it can handle.

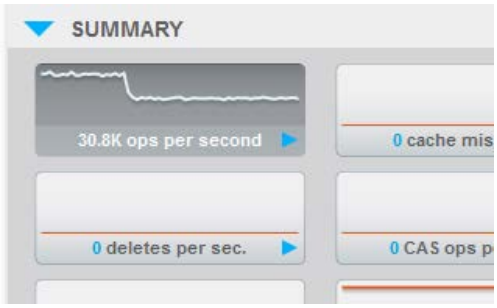
Return to the 1st Couchbase node (Couchbase01 dark blue VM) and hit CTRL + C to stop the cbworkloadgen command:



You will see a message such as this and be returned back to the command line:

```
.....................................^Cinterrupted.
[ec2-user@Couchbase01 ~]$
```

In the Web UI you will see the cluster go back down to 30,000 write operations per second:



Gracefully decommission and recommission a node from the cluster:

Suppose you have to upgrade some hardware on a Couchbase cluster node and need to temporarily remove it from the cluster, power it down to upgrade the hardware and then start the server back up and re-join it to the cluster.

In scenarios like these, where you can plan your outage ahead of time, you should "remove", "rebalance" and later on "add" the affected node back into the cluster.

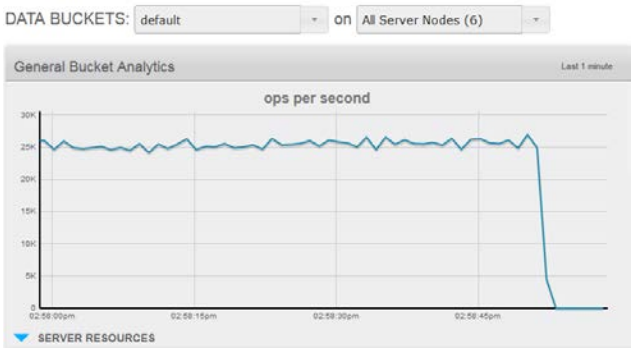
In this section, we will walk through the correct method for removing a node from the cluster and adding the node back into the cluster. There is no disruption in data service or no loss of data that can occur when you remove a node then rebalance the cluster. If you need to remove a functioning node for administration purposes, you should use the remove and rebalance functionality not failover.

Switch over to the App Server (AppServer black VM)
and stop the cbworkloadgenerator by hitting CTRL + C:

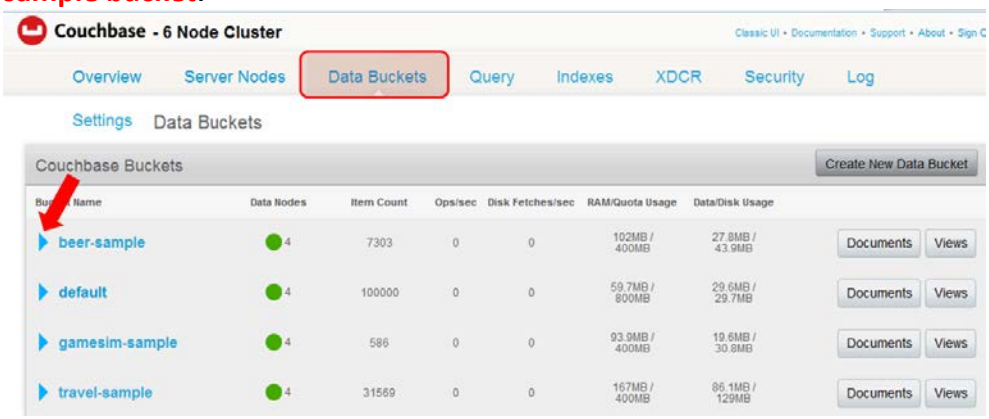
You will see a message such as this and be returned back to the command line:

```
.....................................^Cinterrupted.
[ec2-user@AppServer ~]$
```

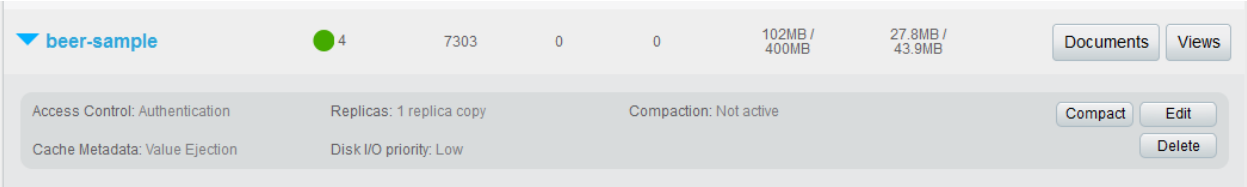
In the Web UI you will see the cluster go back down to 0 write operations per second:



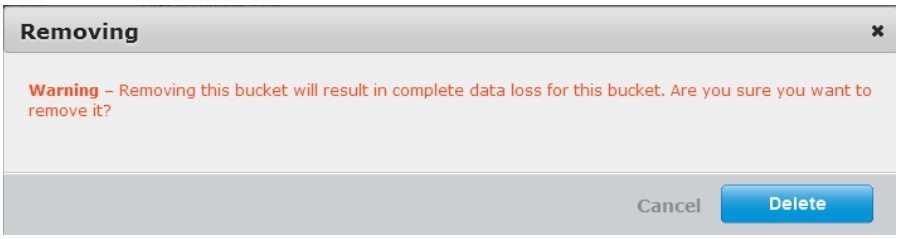
Switch to the Web UI and click on "Data Buckets" at the top and then expand the beer-sample bucket:



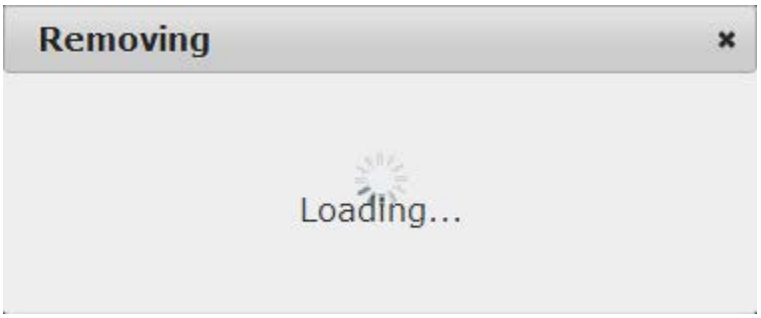
Once the details under beer-sample are expanded, click on Delete:



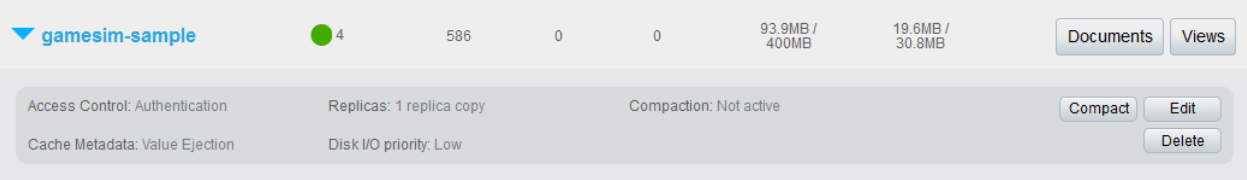
Click Delete on the Removing popup as well:



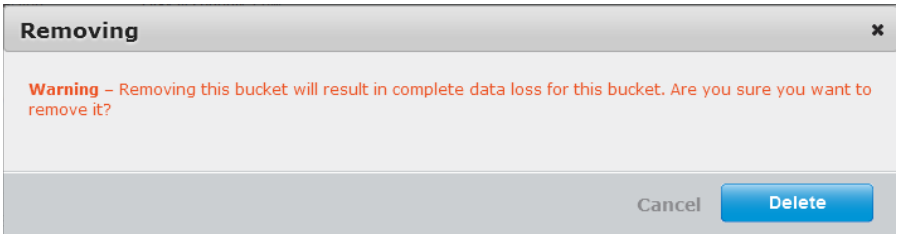
In a few seconds, the bucket will be deleted:



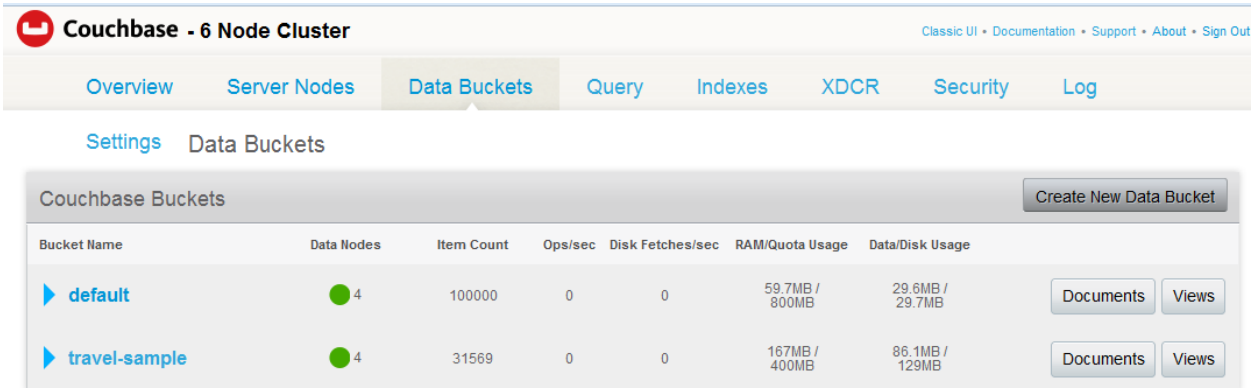
Delete the gamesim-sample bucket next. Expand the gamesim-sample details and then click "Delete":



Click Delete on the Removing popup as well:

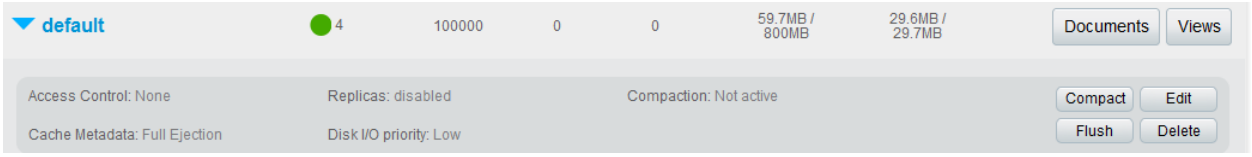


Now delete the Travel-sample bucket using the same procedures.



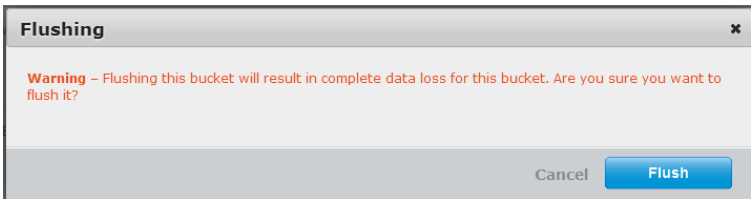
When completed will only have the default bucket left.

Finally, we will flush all of the contents in the default bucket. However, we will not delete the default bucket! Expand the default bucket and click Flush:



NOTE: You have to first enable the Flush feature on the bucket(should have been done during setup), before being able to flush it to empty it. Scroll down and place a check next to Enable and then click Save:

Click Flush on the popup as well:



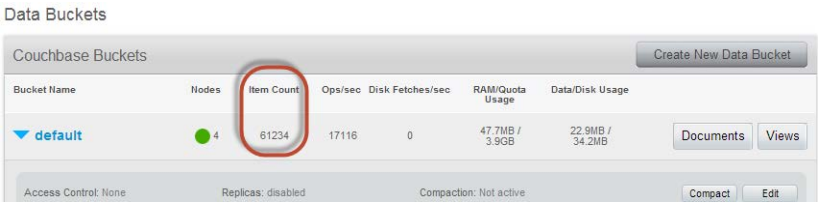
You should now see that the Item Count in the default bucket is 0:

Switch over to the App Server (AppServer black VM) and generate a 100% write load against the default bucket. Use 1 million unique items, item size of 20 bytes, 3 threads and loop forever until interrupted. Note that this is about 20 MB of unique data.

```
[ec2-user@AppServer ~]$ cbworkloadgen -n ec2-54-85-43-x.compute-1.amazonaws.com:8091 -u Administrator -p couchbase -i 1000000 -r 1 -s 20 -t 3 -l -v
.................................................
```

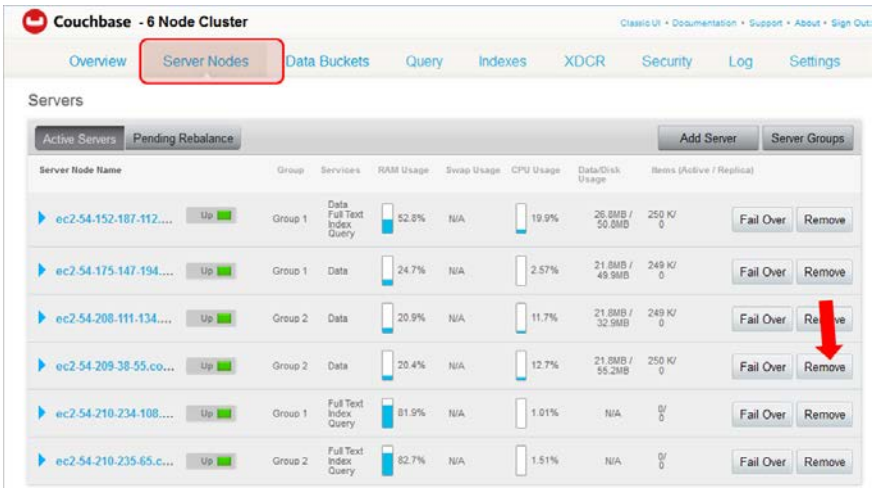
NOTE: this will be in your command line history (up and down arrow keys) and can be edited for the correct item count

If you refresh the Web UI, you should see the Item Count for the default bucket increase:

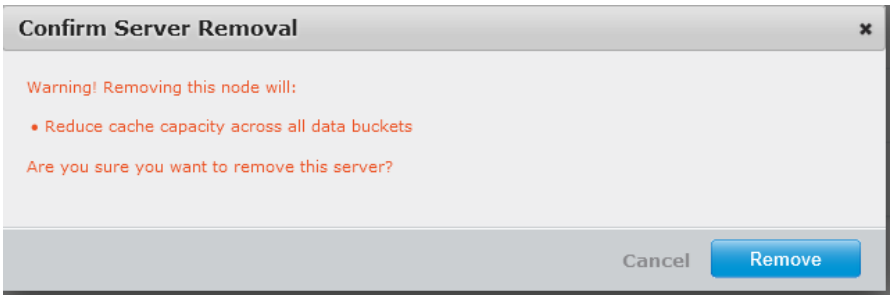


Eventually, the Item Count will hit 1 million:

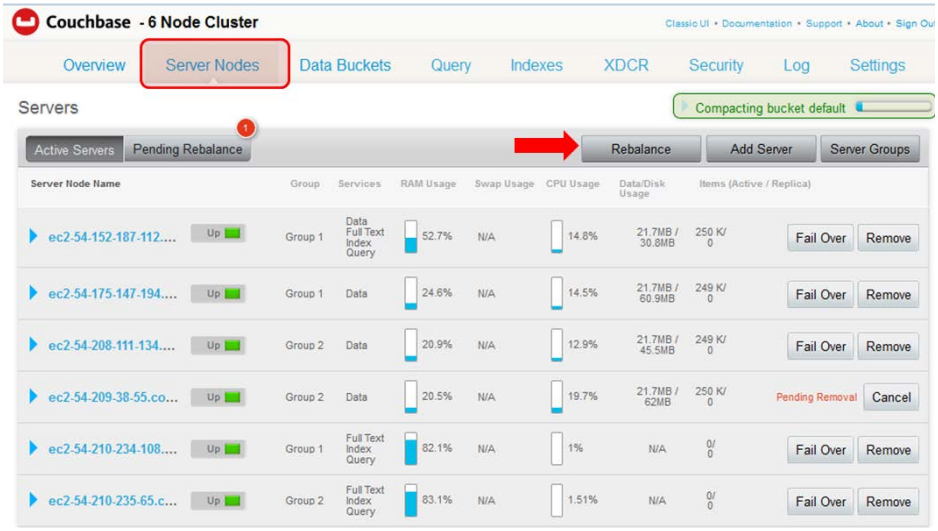
Click on "Server Nodes" at the top of the page and identify the 4th machine in the cluster. This should be in Group 2. Note, that the 4th machine will not necessarily be the last server in the list. Use the Cluster-IPs spreadsheet to match the public hostname of the 4th machine to one of the machines in this list. Then click Remove for that machine:



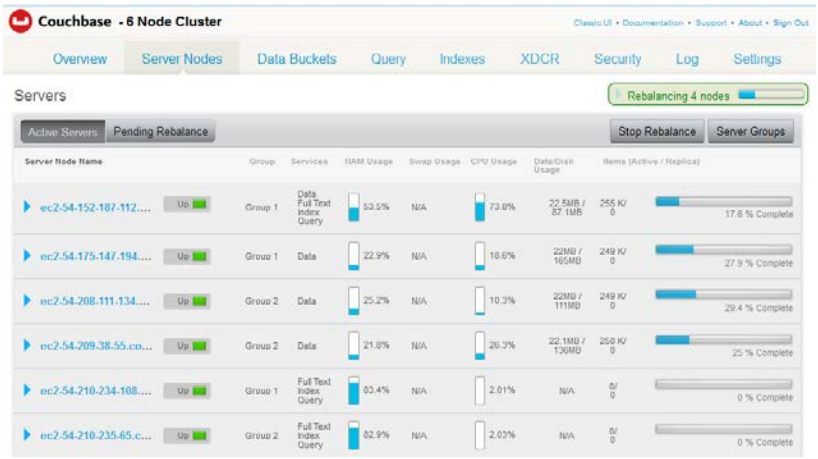
Confirm the server removal by clicking "Remove":



A #1 will appear next to "Pending Rebalance". The 4th node will technically not be removed from the cluster, until a rebalance. Click on Rebalance:



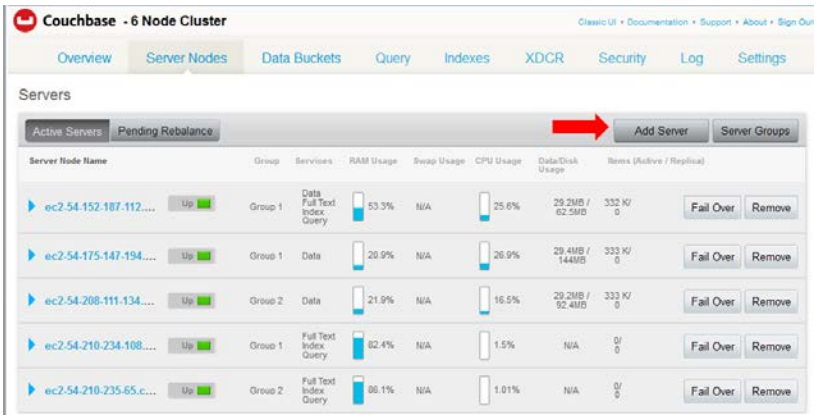
The Rebalance operation will start. Since there is only one bucket with around 20 MB of data in it, the rebalance should complete in under 2 minutes:



Switch to the App Server (black VM) and you will notice some warnings generated on the App Server such as:

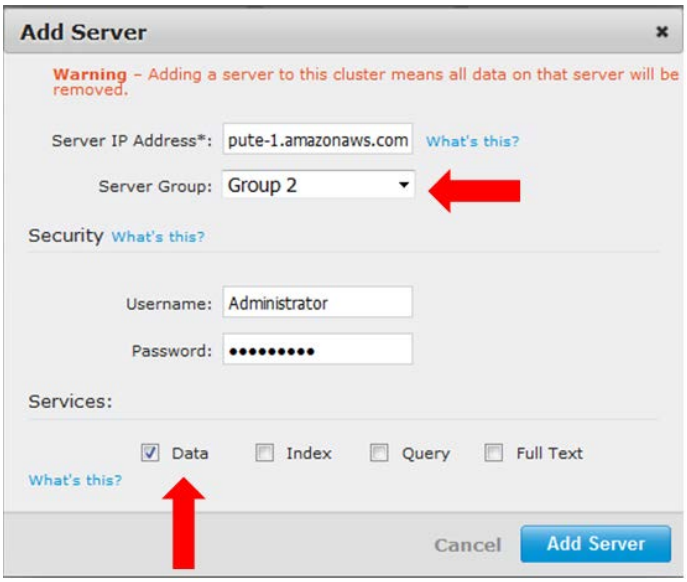
```
...2014-05-26 02:18:04,441: s0 warning: received NOT_MY_VBUCKET; perhaps the cluster is/was
rebalancing; vbucket_id: 276, key: pymc218749, spec: http://ec2-54-85-43-128.compute-
1.amazonaws.com:8091, host:port: ec2-54-85-206-193.compute-1.amazonaws.com:11210

2014-05-26 02:18:04,441: s0 warning: received NOT_MY_VBUCKET; perhaps the cluster is/was
rebalancing; vbucket_id: 276, key: pymc218749, spec: http://ec2-54-85-43-128.compute-
1.amazonaws.com:8091, host:port: ec2-54-85-206-193.compute-1.amazonaws.com:11210
2014-05-26 02:18:04,452: s0 refreshing sink map: http://ec2-54-85-43-128.compute-
1.amazonaws.com:8091
```

After the rebalance is complete, the above warning messages will stop appearing.
When the rebalance is finished, you will see only 5 nodes displayed under Server Nodes. Click on Add Server to provision the 4th node back into the cluster:

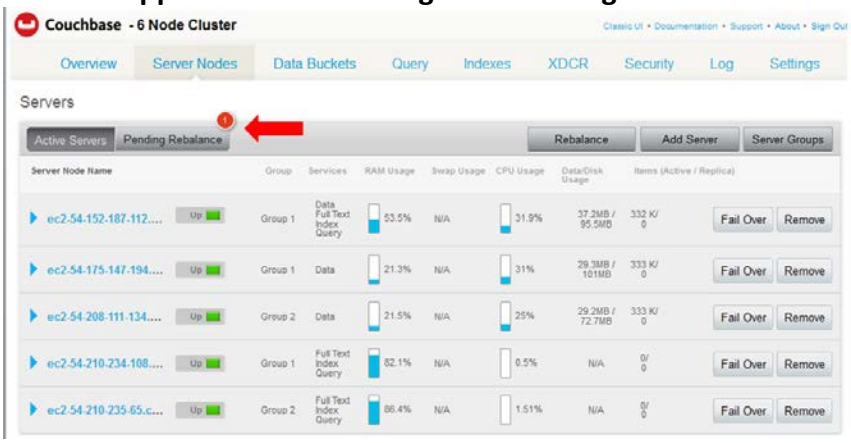


Enter the 4th node's public hostname into the "Server IP Address" field, choose Server Group 2, use Administrator for the username and ensure that the password is "couchbase".
Select Data box for service offering
Click Add Server:

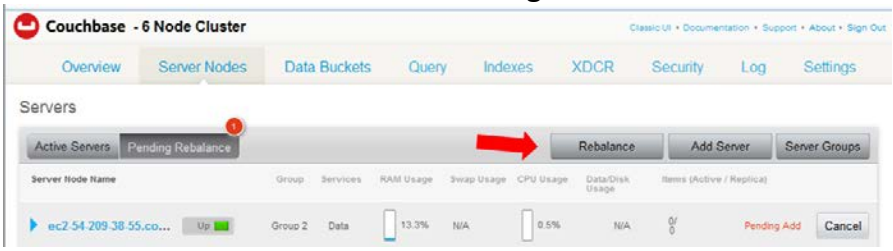
Couchbase



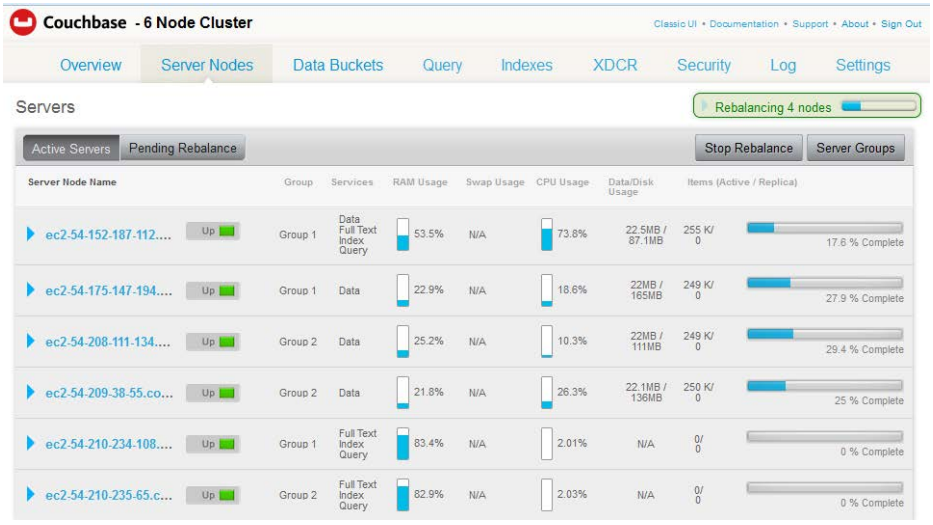
A #1 will appear next to Pending Rebalance again. Click on Pending Rebalance:



You should see the 4th node in a Pending Add state. Click on Rebalance:



Within 2 minutes, the rebalance operation should finish and the 4th node will be successfully added back into the cluster.



During this last rebalance, you will notice that the cbworkloadgen application on the App Server will have crashed and returned back to the command line:

```
.2014-05-26 02:29:05,553: s0 warning: received NOT_MY_VBUCKET; perhaps the cluster is/was
rebalancing; vbucket_id: 341, key: pymc197831, spec: http://ec2-54-85-43-128.compute-
1.amazonaws.com:8091, host:port: ec2-54-85-43-128.compute-1.amazonaws.com:11210
2014-05-26 02:29:05,553: s0 warning: received NOT_MY_VBUCKET; perhaps the cluster is/was
rebalancing; vbucket_id: 341, key: pymc197831, spec: http://ec2-54-85-43-128.compute-
1.amazonaws.com:8091, host:port: ec2-54-85-43-128.compute-1.amazonaws.com:11210
2014-05-26 02:29:05,558: s0 refreshing sink map: http://ec2-54-85-43-128.compute-
1.amazonaws.com:8091
2014-05-26 02:29:05,765: s0 MCSink exception:
2014-05-26 02:29:05,765: s0 error: async operation: error: MCSink exception:  on sink:
http://ec2-54-85-43-128.compute-1.amazonaws.com:8091(default@N/A-0)
error: MCSink exception:

[ec2-user@ip-172-31-23-211 ~]$
```

Do not restart the workload generator at this time. A typical application reading and writing to Couchbase will not see such an error. This is specifically a bug in cbworkloadgen and will be fixed in a future release.

Note that the "cbc-pillowfight" tool should not display any warnings or error messages at all if ran during a remove/add or rebalance operation.

Failing over a node in the cluster:

If a node in a cluster is unable to serve data you can failover that node. Failover means that Couchbase Server removes the node from a cluster and makes replicated data at other nodes available for client requests. As long as you have a sufficient number of replicas configured in Couchbase, the cluster can handle failure of one or more nodes without affecting your ability to access the stored data. Failover can be initiated manually via the Web UI, REST API or Couchbase CLI.

Alternatively, you can configure Couchbase server to automatically remove a failed node from a cluster and have the cluster operate in a degraded mode. If you choose this automatic option, the workload for functioning nodes that remain the cluster will increase. You will still need to address the node failure, return a functioning node to the cluster and then rebalance the cluster in order for the cluster to function as it did prior to node failure. Automatically failing components in any distributed system can cause problems. If you cannot identify the cause of failure, and you do not understand the load that will be placed on the remaining system, then automated failover can cause more problems than it is designed to solve.

Keep in mind that if you try to failover a functioning node it may result in data loss. This is because failover will immediately remove the node from the cluster and any data that has not yet been replicated to other nodes may be permanently lost if it had not been persisted to disk.

Be aware that failover is a distinct operation compared to removing/rebalancing a node. Typically you remove a functioning node from a cluster for maintenance, or other reasons; in contrast you perform a failover for a node that does not function. If you need to remove a functioning node for administration purposes, you should use the remove and rebalance functionality not failover.

Before we trigger an actual Failover, let's understand the set up and configuration of the default bucket, where its vBuckets are stored and whether autoFailover is turned on for our lab cluster.

Run the next set of commands from the App server (Black VM):

Run the cbstats command against the 1st node's public hostname and grep for active_num and replica_num:

```
[ec2-user@AppServer ~]$ cbstats ec2-54-85-43-x.compute-
1.amazonaws.com:11210 all -b default | egrep -i
"active_num|replica_num"
 vb_active_num:                     256
 vb_active_num_non_resident:        0
 vb_replica_num:                    0
 vb_replica_num_non_resident:       0
```

Notice that the 1st node is only hosting 256 active vBuckets and no replica vBuckets.

Each bucket in Couchbase is made of 1024 individual vBuckets (virtual buckets). The 1024 vBuckets are distributed throughout the cluster evenly so each node is hosting a somewhat fair load of the bucket.

Try running the same command against the 2nd node's public hostname (get this from the Cluster-IPs spreadsheet):

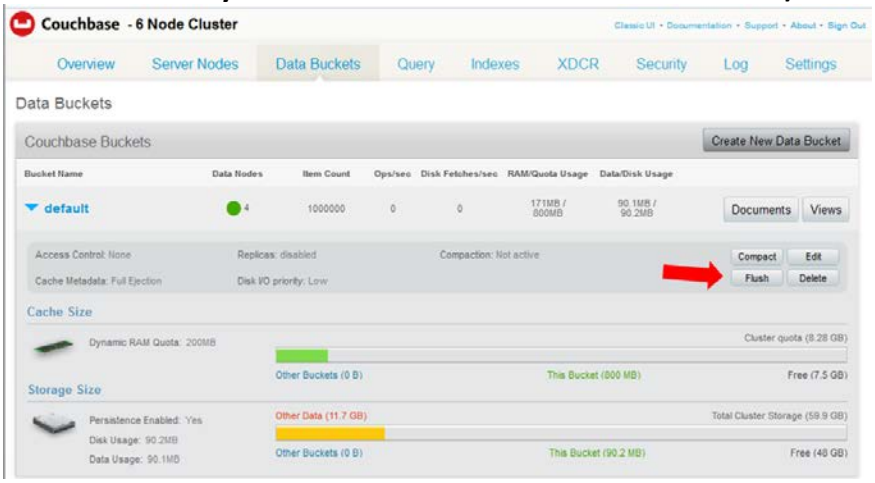
```
[ec2-user@AppServer ~]$ cbstats ec2-54-86-106-x.compute-1.amazonaws.com:11210 all -b default | egrep -i "active_num|replica_num"
 vb_active_num:                          256
 vb_active_num_non_resident:             0
 vb_replica_num:                         0
 vb_replica_num_non_resident:            0
```

You should see 256 active-num on the 2nd node as well.

Note that since we don't have any replicas configured for the default bucket, we are not utilizing the Rack Awareness/Server Groups feature to distribute replica vBuckets to different Server Groups.

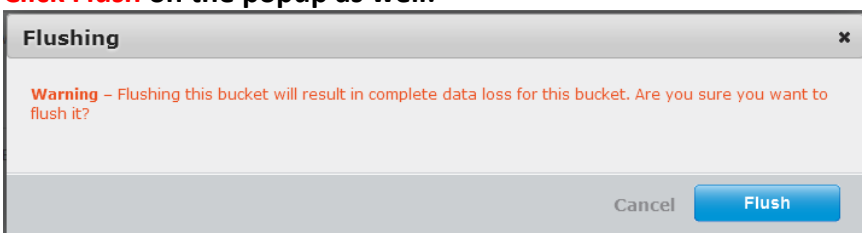
Let's switch to the Web UI and enable 1 replica for the default bucket. But first flush the default bucket to delete all of the data within it.

Click on Data Buckets at the top of the Web UI, notice the Item Count in the bucket (your item count # may differ than the # in the screenshot below) and then click on Flush:



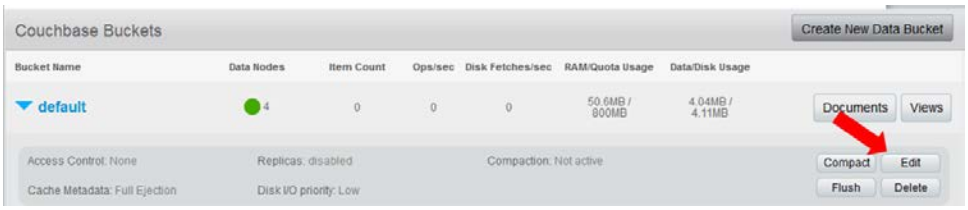
Scroll to the bottom of the Configure Bucket popup and click Flush:

Click Flush on the popup as well:

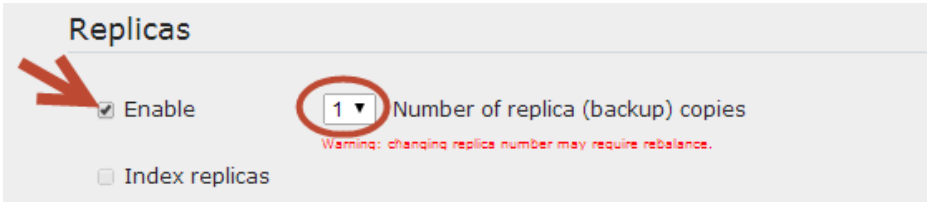


You will now see that the item count is 0 and that Replicas are still disabled for this bucket:

Enable one replica for the default bucket next. Click on Edit again:

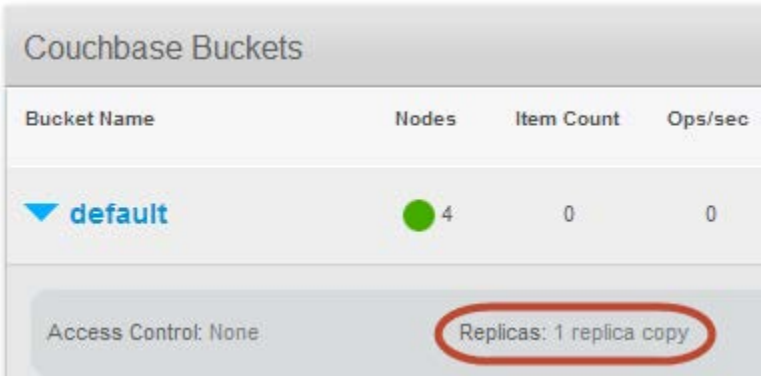


Mid-way on the Configure Buckets popup, you should see a setting for Replicas. Place a check mark next to Enable and ensure that the Number of replica copies is set to 1:

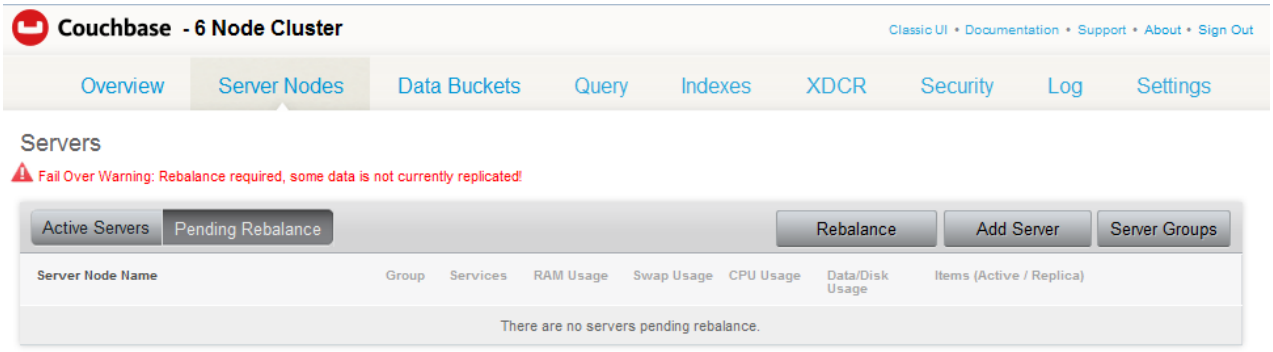


Click Save:

You should now see 1 replica copy configured:



In the Couchbase Web UI, click on "Server Nodes" at the top of the page and then click Pending Rebalance. Notice that there are no specific nodes that need to be rebalanced at this time, even though we added a replica to the default bucket. This is because the default bucket is empty and contains zero keys since we flushed it earlier. However, also notice that there is a red "Fail Over Warning" message displayed that is basically saying that a rebalance is required before data will be replicated. DO NOT click on "Rebalance" just yet… Instead switch over to the App Server cmd line…the next few commands will explain why a rebalance is required at this point.

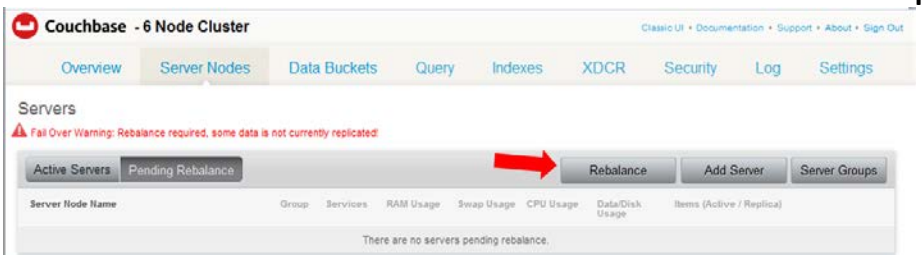


Run the next set of commands from the App Server (Black VM):

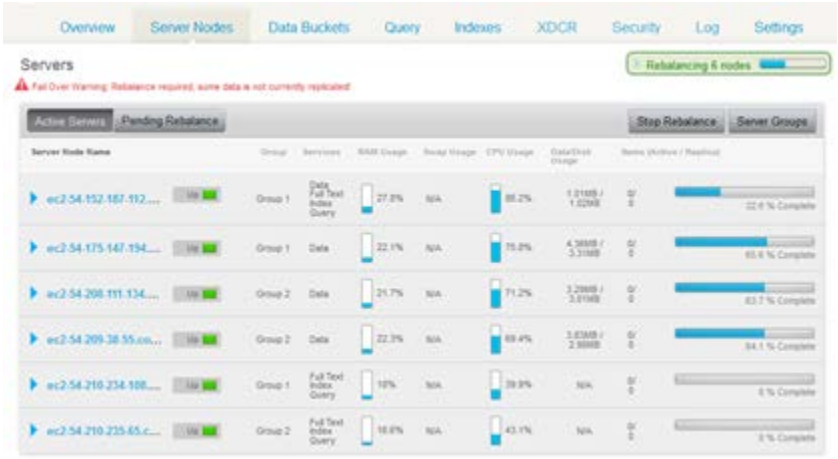
Once again, run the cbstats command against the 1st node's public hostname and grep for active_num and replica_num:

```
[ec2-user@AppServer ~]$ cbstats ec2-54-85-43-x.compute-
1.amazonaws.com:11210 all -b default | egrep -i
"active_num|replica_num"
 vb_active_num:                     256
 vb_active_num_non_resident:        0
 vb_replica_num:                    0
 vb_replica_num_non_resident:       0
```

Notice that the 1st node is still only hosting 256 active vBuckets and no replica vBuckets. Return to the Couchbase Web UI. Click on Rebalance to create the replica vBuckets:



The rebalance operation should take about 1 – 2 minutes to complete:

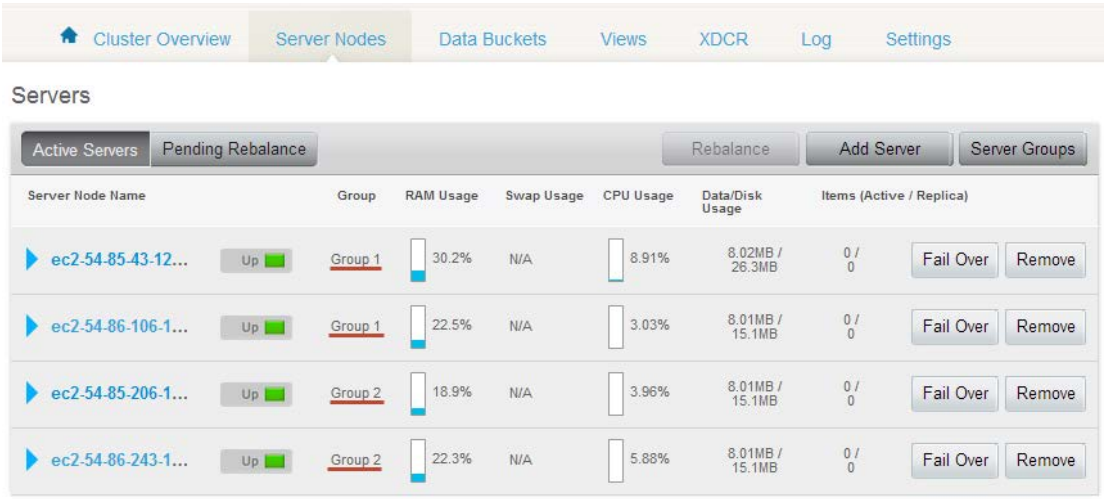


While the rebalance is occurring, quickly try running the cbstats command against the 1st node multiple times and you should see the # of replica vBuckets on this node increase slowly to a max of 256 vBuckets:

```
[ec2-user@AppServer ~]$ cbstats ec2-54-85-43-x.compute-
1.amazonaws.com:11210 all -b default | egrep -i
"active_num|replica_num"
 vb_active_num:                     256
vb_active_num_non_resident:         0
vb_replica_num:                     5
vb_replica_num_non_resident:        0

[ec2-user@AppServer ~]$ cbstats ec2-54-85-43-x.compute-
1.amazonaws.com:11210 all -b default | egrep -i
"active_num|replica_num"
 vb_active_num:                     256
vb_active_num_non_resident:         0
vb_replica_num:                     63
vb_replica_num_non_resident:        0


[ec2-user@AppServer ~]$ cbstats ec2-54-85-43-x.compute-
1.amazonaws.com:11210 all -b default | egrep -i
"active_num|replica_num"
vb_active_num:                      256
vb_active_num_non_resident:         0
vb_replica_num:                     201
vb_replica_num_non_resident:        0

[ec2-user@AppServer ~]$ cbstats ec2-54-85-43-128.compute-
1.amazonaws.com:11210 all -b default | egrep -i
"active_num|replica_num"
 vb_active_num:                     256
vb_active_num_non_resident:         0
vb_replica_num:                     256
vb_replica_num_non_resident:        0
```

You will finally see 256 replica vBuckets on the 1st node.

Once the Rebalance operation is complete, the red Warning message under the Server Nodes page will disappear. Also notice that currently, 2 nodes are under Server Group 1 and 2 nodes are under Server Group 2(not counting the Query and index service nodes):



Before we actually trigger a failover, let's write 10 keys to the default bucket. But where will the 10 keys be distributed within the 4-node cluster's active vBuckets? The following cbc hash command's output will display 10 lines, each line will identify which of the 4-nodes in the cluster the key will be directed to. Run this command on the App Server (black VM) against the 1st node's public IP:

```
[ec2-user@AppServer ~]$ cbc-hash -U couchbase://ec2-54-172-130-66.compute-1.amazonaws.com/default key-1 key-2 key-3 key-4 key-5 key-6 key-7 key-8 key-9 key-10

key-1: [vBucket=748, Index=2] Server: ec2-54-172-130-69.compute-1.amazonaws.com:11210, CouchAPI: http://ec2-54-172-130-69.compute-1.amazonaws.com:8092/default
Replica #0: Index=1, Host=ec2-54-172-130-66.compute-1.amazonaws.com:11210

key-2: [vBucket=997, Index=3] Server: ec2-54-172-130-8.compute-1.amazonaws.com:11210, CouchAPI: http://ec2-54-172-130-8.compute-1.amazonaws.com:8092/default
Replica #0: Index=1, Host=ec2-54-172-130-66.compute-1.amazonaws.com:11210

key-3: [vBucket=226, Index=1] Server: ec2-54-172-130-66.compute-1.amazonaws.com:11210, CouchAPI: http://ec2-54-172-130-66.compute-1.amazonaws.com:8092/default
Replica #0: Index=3, Host=ec2-54-172-130-8.compute-1.amazonaws.com:11210

key-4: [vBucket=646, Index=2] Server: ec2-54-172-130-69.compute-1.amazonaws.com:11210, CouchAPI: http://ec2-54-172-130-69.compute-1.amazonaws.com:8092/default
Replica #0: Index=1, Host=ec2-54-172-130-66.compute-1.amazonaws.com:11210

key-5: [vBucket=385, Index=0] Server: ec2-54-172-130-15.compute-1.amazonaws.com:11210, CouchAPI: http://ec2-54-172-130-15.compute-1.amazonaws.com:8092/default
Replica #0: Index=3, Host=ec2-54-172-130-8.compute-1.amazonaws.com:11210

key-6: [vBucket=136, Index=1] Server: ec2-54-172-130-66.compute-1.amazonaws.com:11210, CouchAPI: http://ec2-54-172-130-66.compute-1.amazonaws.com:8092/default
Replica #0: Index=3, Host=ec2-54-172-130-8.compute-1.amazonaws.com:11210

key-7: [vBucket=911, Index=3] Server: ec2-54-172-130-8.compute-1.amazonaws.com:11210, CouchAPI: http://ec2-54-172-130-8.compute-1.amazonaws.com:8092/default
Replica #0: Index=1, Host=ec2-54-172-130-66.compute-1.amazonaws.com:11210

key-8: [vBucket=816, Index=3] Server: ec2-54-172-130-8.compute-1.amazonaws.com:11210, CouchAPI: http://ec2-54-172-130-8.compute-1.amazonaws.com:8092/default
Replica #0: Index=0, Host=ec2-54-172-130-15.compute-1.amazonaws.com:11210
```

```
key-9: [vBucket=55, Index=1] Server: ec2-54-172-130-66.compute-1.amazonaws.com:11210, CouchAPI:
http://ec2-54-172-130-66.compute-1.amazonaws.com:8092/default
Replica #0: Index=2, Host=ec2-54-172-130-69.compute-1.amazonaws.com:11210

key-10: [vBucket=5, Index=1] Server: ec2-54-172-130-66.compute-1.amazonaws.com:11210, CouchAPI:
http://ec2-54-172-130-66.compute-1.amazonaws.com:8092/default
Replica #0: Index=2, Host=ec2-54-172-130-69.compute-1.amazonaws.com:11210
```

Note that running the above command against any of the 4 nodes' public hostnames would result in the same, identical output.

Since we have Server Groups configured, we should see that if active data lands one of the first 2 nodes (Group 1), its replica ends up on one of the second 2 nodes (Group 2).

In other words, in my specific lab environment, I have the following two nodes configured in Server Group #1:

Node #1: ec2-54-85-43-128
Node #2: ec2-54-86-106-120

Server Group #2 contains:
Node #3: ec2-54-86-243-136
Node #4: ec2-54-85-206-193

So, in my example, any of the active data that lands in Node #1 or #2, should have its replica stored on Node #3 or #4.

If you scroll back up 1 or 2 pages to the output of where the 10 keys will be stored, this pattern should now become more apparent.
For example, in my environment:

key-1: Active: Node #4 / Replica: Node #2

key-2: Active: Node # 4 / Replica: Node #2

key-3: Active: Node # 1 / Replica: Node #3

key-4: Active: Node # 2 / Replica: Node #3

key-5: Active: Node # 2 / Replica: Node #4

key-6: Active: Node # 1 / Replica: Node #3

key-7: Active: Node # 3 / Replica: Node #2

key-8: Active: Node # 3 / Replica: Node #2

key-9: Active: Node # 1 / Replica: Node #4

key-10: Active: Node # 1 / Replica: Node #4

Notice above that Node #1 is holding 4 Active keys and 0 Replica keys.

At this time, please take a few minutes to make a quick document similar to the 10 lines I have above using your cluster's custom configuration. You should also see in your specific lab environment, that replicas for the active data are being placed in a different Server Group.

Switch to the App Server (black VM) and insert the following 10 keys with 10 values into the cluster. You can use the public hostname of any of the 4 nodes in the next two pages for the 'cbc create' and 'cbc cat' commands (*note, you may need to hit CTRL + D a few times below before it takes effect)*:

```
[ec2-user@AppServer ~]$ cbc-create  -U couchbase://ec2-54-172-130-
66.compute-1.amazonaws.com/default key-1<hit enter>
<Then type in>

value-1

<hit CTRL + D (you may have to do this slowly 2 or 3 times, however
when you regain your command prompt then stop)>
key-1                Stored. CAS=0xb70d431da0910400
```

Verify that key-1 was successfully inserted by reading it:

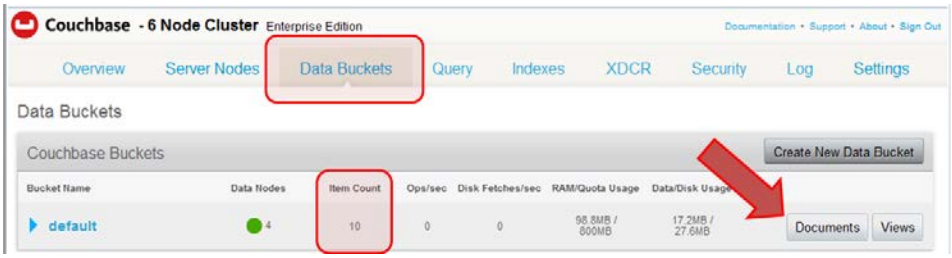
```
[ec2-user@AppServer ~]$ cbc-cat  -U couchbase://ec2-54-172-130-
66.compute-1.amazonaws.com/default key-1
key-1                CAS=0x8ddf5365d5910400, Flags=0x0, Datatype=0x0
value-1
```

Next, go ahead and enter the remaining 9 keys as follows:

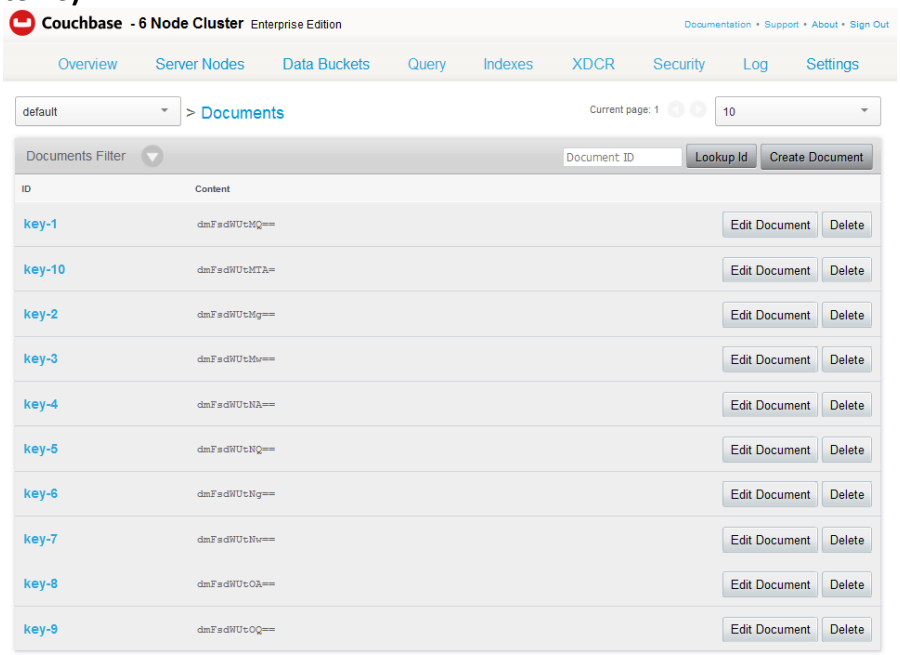
key-2 : value-2
key-3 : value-3
key-4 : value-4
key-5 : value-5
key-6 : value-6
key-7 : value-7
key-8 : value-8
key-9 : value-9
key-10 : value-10


```
[ec2-user@AppServer ~]$ cbc-create  -U couchbase://ec2-54-172-130-
66.compute-1.amazonaws.com/default key-2<hit enter>
value-2<hit CTRL + D>
value-2key-2                Stored. CAS=0x735ce8272f920400

[ec2-user@AppServer ~]$ cbc-create -U couchbase://ec2-54-172-130-
66.compute-1.amazonaws.com/default key-3<hit enter>
value-3<hit CTRL + D>
value-3key-3                Stored. CAS=0xfd57a352bc920400

[ec2-user@AppServer ~]$ cbc-create -U couchbase://ec2-54-172-130-
66.compute-1.amazonaws.com/default key-4<hit enter>
value-4<hit CTRL + D>
value-4key-4                Stored. CAS=0xe3ae3cfbd1920400

[ec2-user@AppServer ~]$ cbc-create -U couchbase://ec2-54-172-130-
66.compute-1.amazonaws.com/default key-5<hit enter>
value-5<hit CTRL + D>
value-5key-5                Stored. CAS=0xb62d16b992380000

[ec2-user@AppServer ~]$ cbc-create -U couchbase://ec2-54-172-130-
66.compute-1.amazonaws.com/default key-6<hit enter>
value-6<hit CTRL + D>
value-6key-6                Stored. CAS=0x4570e03ae0920400

[ec2-user@AppServer ~]$ cbc-create -U couchbase://ec2-54-172-130-
66.compute-1.amazonaws.com/default key-7<hit enter>
value-7<hit CTRL + D>
value-7key-7                Stored. CAS=0x1da067b2e1920400

[ec2-user@AppServer ~]$ cbc-create -U couchbase://ec2-54-172-130-
66.compute-1.amazonaws.com/default key-8<hit enter>
value-8<hit CTRL + D>
value-8key-8                Stored. CAS=0x713bce5b3e930400

[ec2-user@AppServer ~]$ cbc-create -U couchbase://ec2-54-172-130-
66.compute-1.amazonaws.com/default key-9<hit enter>
value-9<hit CTRL + D>
value-9key-9                Stored. CAS=0xea941c014f930400

[ec2-user@AppServer ~]$ cbc-create -U couchbase://ec2-54-172-130-
66.compute-1.amazonaws.com/default key-10<hit enter>
value-10<hit CTRL + D>
value-10key-10               Stored. CAS=0x439796f256930400
```

Switch to the Web UI and click on Data Buckets at the top to verify that there are now 10 items in the default bucket. Then click on Documents:



On the next page, you should see all 10 keys sorted in lexographical order(change page view to 10):



Switch back to the App Server (black VM) and check the Auto-failover settings on the default bucket via the REST API (you can use the public hostname of any of the 4 nodes in this command):

```
[ec2-user@AppServer ~]$ curl -u Administrator:couchbase http://ec2-54-85-43-x.compute-1.amazonaws.com:8091/settings/autoFailover
{"enabled":false,"timeout":120,"count":0}
```

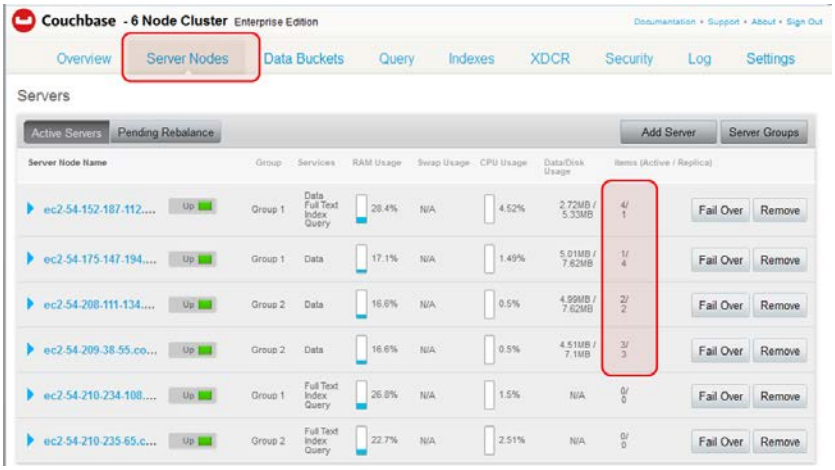
The following parameters and settings appear in the output above:
enabled : either true if auto-failover is enabled or false if it is not.
timeout : seconds that must elapse before auto-failover executes on a cluster.
count : can be 0 or 1. Number of times any node in a cluster can be automatically failed-over. After one auto-failover occurs, count is set to 1 and Couchbase server will not perform auto-failure for the cluster again unless you reset the count to 0. If you want to failover more than one node at a time in a cluster, you will need to do so manually.

The following command disables auto-failover and then next verifies that it is disabled:
Note : The default setting of disabled and <u>you do not need to run these commands now</u>.

```
[ec2-user@ip-172-31-23-211 ~]$ curl "http://ec2-54-85-43-x.compute-
1.amazonaws.com:8091/settings/autoFailover" -i -u Administrator:couchbase -d 'enabled=false'
HTTP/1.1 200 OK
Server: Couchbase Server
Pragma: no-cache
Date: Mon, 26 May 2014 20:50:35 GMT
Content-Length: 0
Cache-Control: no-cache

[ec2-user@ip-172-31-23-211 ~]$ curl -u Administrator:couchbase http://ec2-54-85-43-x.compute-
1.amazonaws.com:8091/settings/autoFailover
{"enabled":false,"timeout":120,"count":0}
```

Note that the autoFailover settings can also be configured using the Couchbase CLI as described here:
http://docs.couchbase.com/admin/admin/CLI/CBcli/cbcli-commands.html

Switch back to the Web UI and click on Server Nodes at the top:



In the column for Items (Active/Replica) you should see the 10 keys that we created and 10 replica keys for them also, so the total count should be 20.
The numbers that you see here should correlate to the chart that you created for the 10 keys earlier in this lab.

For example, Node #1 in my cluster starts with the Public hostname "ec2-54-85-43-128", which also just happens to be the 1st node in the screenshot above. This node is holding 4 active keys and 0 replica keys. This matches what I was expecting after creating the chart earlier. Perform the same check for a few of the nodes in your environment.

Next we will stop the Couchbase service on both nodes in Server Group 2. Note by looking at the screenshot below, in my specific cluster, I will lose 4 active keys by doing this (2 active keys on each of the nodes).

More specifically for my specific installation, active key-7 and key-8 will no longer be available on Node #3 and active key-1 and key-2 will no longer be available on Node #4.

After stopping the Couchbase process on Nodes 3 and 4, we will need to failover Node 3 and Node 4 before the replicas get activated and we can read those 4 keys from the activated replicas.

Warning: The keys that you will need to test for your specific installation will differ from the keys that I will be using. Use the chart that you created earlier to figure out which keys are active on Node 3 and Node 4 and those are the keys that you will be reading below. You may want to write down which active keys are hosted on Node 3 and which active keys are hosted on Node 4 before continuing:

Fill out this form:

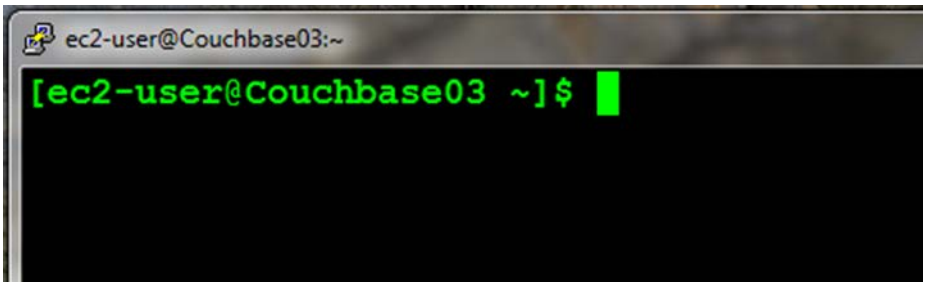
Active Keys hosted on Node #3: _____
Active Keys hosted on Node # 4: _____

First read an active key from Node #3. Use the form you filled right above to pick a key from Node #3. In my specific lab, I have picked key-7. You may have to pick a different key. Run the following command from the App Server (use the 1st node's public hostname for the command below):

```
[ec2-user@AppServer ~]$ cbc-cat -U couchbase://ec2-54-172-130-
66.compute-1.amazonaws.com/default key-7
key-7                CAS=0x1da067b2e1920400, Flags=0x0, Datatype=0x0
value-7
```

So far so good.

Next, switch to the 3rd node's (Green VM/Couchbase03) PuTTY window and stop the Couchbase service on the node:




```
[ec2-user@Couchbase03 ~]$ sudo systemctl stop couchbase-server
Stopping couchbase-server

[ec2-user@Couchbase03 ~]$ sudo systemctl status couchbase-server
Aug 10 13:37:11 node3 systemd[1]: Stopped LSB: couchbase server.
```

Switch over to the App Server (black VM) and try to read the key you just read from Node #3 again. I will attempt reading key-7 again for my specific environment. All of the following commands will be run from the App Server, until noted otherwise:


```
[ec2-user@AppServer ~]$ cbc-cat -U couchbase://ec2-54-172-130-
66.compute-1.amazonaws.com/default key-7
key-7 The remote host refused the connection. Is the service up? (0x2c)

or you can re-run the previous cbc-cat command from your buffer(command
history)

cbc-cat -U couchbase://ec2-54-152-187-112.compute-1.amazonaws.com/default
key-1 key-2 key-3 key-4 key-5 key-6 key-7 key-8 key-9 key-10

key-3                CAS=0x24e23f2e5614, Flags=0x0. Size=7
value-3
key-6                CAS=0x686f502e5614, Flags=0x0. Size=7
value-6
key-9                CAS=0x1c1c592e5614, Flags=0x0. Size=7
value-9
key-10               CAS=0x132b5c2e5614, Flags=0x0. Size=8
value-10
key-5                CAS=0xd5764d2e5614, Flags=0x0. Size=7
value-5
key-2                CAS=0xe9b3c2e5614, Flags=0x0. Size=7
value-2
```

```
key-7                CAS=0x6110532e5614, Flags=0x0. Size=7
value-7
key-8                CAS=0x412b562e5614, Flags=0x0. Size=7
value-8
key-1                The remote host refused the connection. Is the service up? (0x2c)
key-4                The remote host refused the connection. Is the service up? (0x2c)
```

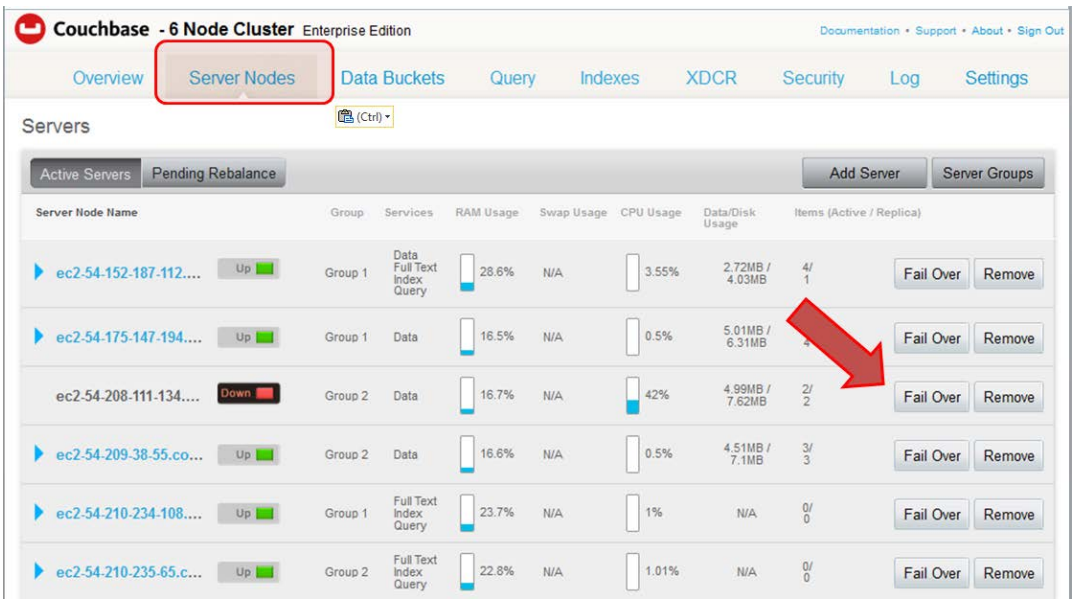
Reading key-8 will also fail on my cluster since that was another key located on Node #3:

```
[ec2-user@AppServer ~]$ cbc-cat -U couchbase://ec2-54-172-130-66.compute-1.amazonaws.com/default key-8
key-X The remote host refused the connection. Is the service up? (0x2c)
```

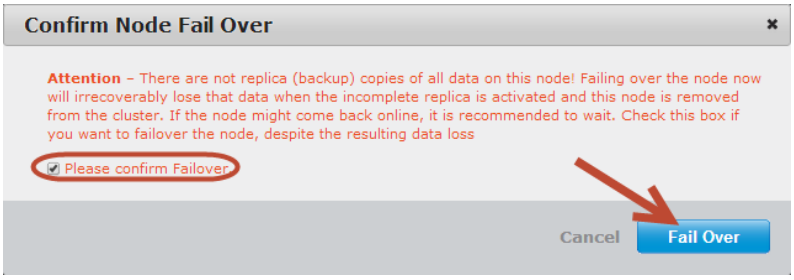
However reading key-1 and key-2 will work successfully, since those keys are stored on Node #4 in my environment. Run a similar test in your environment to ensure that you can still read the active keys from Node #4:

```
[ec2-user@AppServer ~]$ cbc-cat -U couchbase://ec2-54-172-130-66.compute-1.amazonaws.com/default key-1
key-1                CAS=0x44ed48769d920400, Flags=0x0, Size=7
value-1
[ec2-user@AppServer ~]$ cbc-cat -U couchbase://ec2-54-172-130-66.compute-1.amazonaws.com/default key-2
key-2                CAS=0x780fc549ab920400, Flags=0x0, Size=7
value-2
```

Switch over to the Web UI and under Server Nodes, you should see the 3rd node marked as Down. Click on Fail Over for the 3rd/Down node:



On the Confirm Node Failover popup, place a check mark next to "Please confirm Failover" and click on Fail Over:



Run cbstats against all four data service nodes(Couch01,02,03,04)

```
[ec2-user@AppServer ~]$ cbstats ec2-54-152-187-112.compute-
1.amazonaws.com:11210 all -b default | egrep -i
"active_num|replica_num"
 vb_active_num:                             384
 vb_active_num_non_resident:                0
 vb_replica_num:                            128
 vb_replica_num_non_resident:               0

[ec2-user@AppServer ~]$ cbstats ec2-54-175-147-194.compute-
1.amazonaws.com:11210 all -b default | egrep -i
"active_num|replica_num"
 vb_active_num:                             384
 vb_active_num_non_resident:                0
 vb_replica_num:                            128
 vb_replica_num_non_resident:               0

[ec2-user@AppServer ~]$ cbstats ec2-54-208-111-134.compute-
1.amazonaws.com:11210 all -b default | egrep -i
"active_num|replica_num"

Could not connect to ec2-54-208-111-134.compute-
1.amazonaws.com:11210: Connection refused
```

Note: Couch03 couchservice is dead

```
[ec2-user@AppServer ~]$ cbstats ec2-54-209-38-55.compute-
1.amazonaws.com:11210 all -b default | egrep -i
"active_num|replica_num"
 vb_active_num:                             256
 vb_active_num_non_resident:                0
 vb_replica_num:                            256
 vb_replica_num_non_resident:               0
```

```
[ec2-user@AppServer ~]$
```

Do the math….. how many active vbuckets are there? (1024)

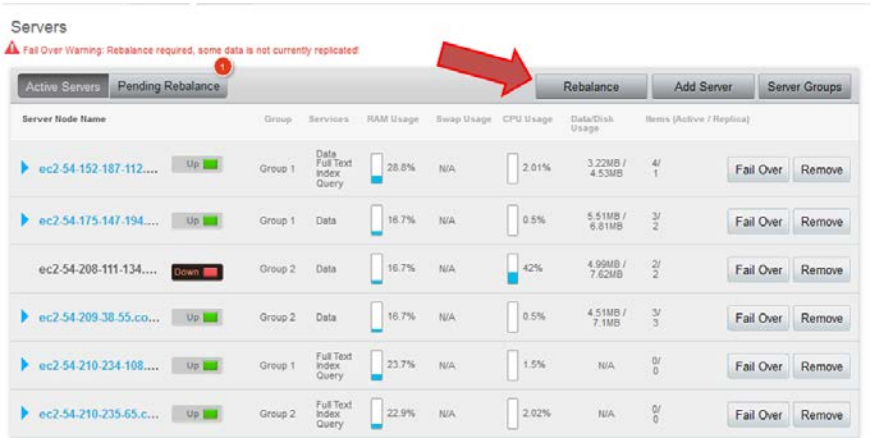
How many replica vbuckets are there? (512)

Why?

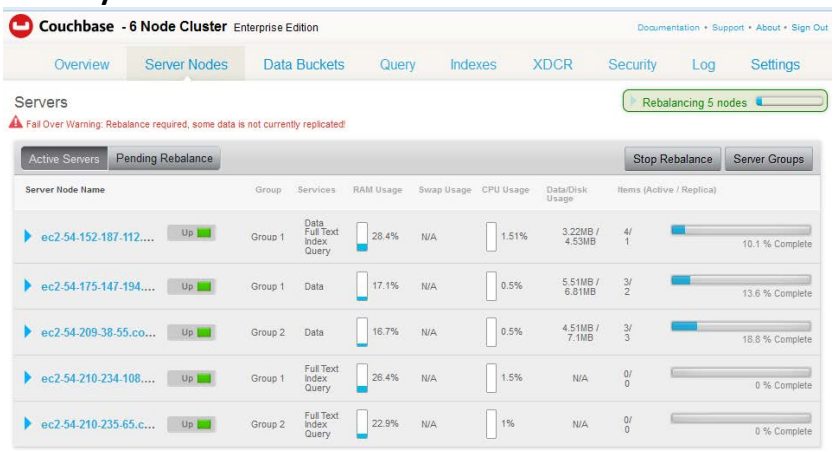
Run the following command.

```
[ec2-user@AppServer ~]$cbc-cat -U couchbase://ec2-54-152-187-
112.compute-1.amazonaws.com/default key-1 key-2 key-3 key-4 key-
5 key-6 key-7 key-8 key-9 key-10

key-3                CAS=0x24e23f2e5614, Flags=0x0. Size=7
value-3
key-6                CAS=0x686f502e5614, Flags=0x0. Size=7
value-6
key-9                CAS=0x1c1c592e5614, Flags=0x0. Size=7
value-9
key-10               CAS=0x132b5c2e5614, Flags=0x0. Size=8
value-10
key-1                CAS=0xb421192e5614, Flags=0x0. Size=7
value-1
key-4                CAS=0xc5c2422e5614, Flags=0x0. Size=7
value-4
key-5                CAS=0xd5764d2e5614, Flags=0x0. Size=7
value-5
key-2                CAS=0xe9b3c2e5614, Flags=0x0. Size=7
value-2
key-7                CAS=0x6110532e5614, Flags=0x0. Size=7
value-7
key-8                CAS=0x412b562e5614, Flags=0x0. Size=7
value-8

notice it now returns all 10 keys?

Summary. You can read all ten keys now but there is no backup if
there was a second failure of another node.
```

A Rebalance will now be required. Click on "Rebalance":



The Rebalance will now start and will take about 2 – 3 minutes to complete and you should see only 5 nodes in the cluster:



After the Rebalance query for the active keys that were on Node #3 (key-7 and key-8 in my environment) and you should successfully see them again since their replicas have been activated:

```
[ec2-user@AppServer~]$ cbc-cat -U couchbase://ec2-54-172-130-66.compute-1.amazonaws.com/default key-7
key-7                CAS=0x1da067b2e1920400, Flags=0x0, Size=7
value-7
```

Reading key-8 will also work on my cluster since this was another key located on Node #3 and now has its replica activated:

```
[ec2-user@ AppServer ~]$ cbc-cat -U couchbase://ec2-54-172-130-66.compute-1.amazonaws.com/default key-8
key-8                CAS=0x713bce5b3e930400, Flags=0x0, Size=7
value-8
```

By looking at the chart I created earlier, I can assume that the key-7 and key-8 are being served from Node #2 as this was the node holding the replica for the data. Here is the chart reprinted for my environment:

key-1: Active: Node #4 / Replica: Node #2
key-2: Active: Node # 4 / Replica: Node #2
key-3: Active: Node # 1 / Replica: Node #3
key-4: Active: Node # 2 / Replica: Node #3
key-5: Active: Node # 2 / Replica: Node #4
key-6: Active: Node # 1 / Replica: Node #3
key-7: Active: Node # 3 / Replica: Node #2
key-8: Active: Node # 3 / Replica: Node #2
key-9: Active: Node # 1 / Replica: Node #4
key-10: Active: Node # 1 / Replica: Node #4

Look at the chart for your environment and figure out which node would be hosting the replicas for the active keys that were originally on Node #3.

Check which nodes are hosting those keys now (I will check for key-7 and key-8 myself):

```
[ec2-user@AppServer ~]$ cbc-hash  -U couchbase://ec2-54-172-130-66.compute-1.amazonaws.com/default key-7 key-8

key-7: [vBucket=911, Index=3] Server: ec2-54-172-130-8.compute-1.amazonaws.com:11210, CouchAPI: http://ec2-54-172-130-8.compute-1.amazonaws.com:8092/default
Replica #0: Index=1, Host=ec2-54-172-130-66.compute-1.amazonaws.com:11210

key-8: [vBucket=816, Index=3] Server: ec2-54-172-130-8.compute-1.amazonaws.com:11210, CouchAPI: http://ec2-54-172-130-8.compute-1.amazonaws.com:8092/default
Replica #0: Index=0, Host=ec2-54-172-130-66.compute-1.amazonaws.com:11210
```

For my cluster, the above output translates to:

key-7: Active: Node # 2 / Replica: Node #1
key-8: Active: Node # 2 / Replica: Node #1

At this point, it makes sense that Node #2 is hosting the active replica.

You may find it odd that the Replicas for key-7 and key-8 are now hosted in the same server group as the active data (both are in Group #1 in my specific environment, but this may not be true for your environment). This can occur in very small Couchbase clusters which have an

uneven number of nodes in each server group. The best practice in Couchbase is to replace a failed hardware node in a server group with a working node and *then* run rebalance across all the nodes.

Notice that in the 'cbc-hash' command above, key-7's active copy is currently in vBucket 911 on the server with the public hostname starting with 'ec2-54-86-106-120', which translates to Node #2 in my environment (it may be on another machine in your specific lab).

Finding key-7's active replica in your cluster:

Let's try to find the data file containing key-7 in the Couchbase cluster. Regardless of which keys you were working with (that were originally active on Node #3) in the previous section, in this section try to find key-7.

Run the following cbc hash command from the App Server (black VM) against any node's public hostname in the cluster to find where key-7 is active:

```
[ec2-user@AppServer ~]$ cbc-hash  -U couchbase://ec2-54-172-130-66.compute-1.amazonaws.com/default key-7
key-7: [vBucket=911, Index=3] Server: ec2-54-172-130-8.compute-1.amazonaws.com:11210, CouchAPI: http://ec2-54-172-130-8.compute-1.amazonaws.com:8092/default
Replica #0: Index=1, Host=ec2-54-172-130-66.compute-1.amazonaws.com:11210
```

Since key-7 appears to be on Node #2 in my environment in vBucket 911, I will switch to the PuTTY window for Node #2 (Couchbase02/Light Blue PuTTY window) to verify that key-7 is indeed on Node #2.

Warning: For your specific lab, switch over to the node where the Active copy of key-7 is located. This may not be Node #2 in your environment!

Let's go to the node where key-7 is located and actually locate it in a vBucket data file.

First switch to the root user:

```
[ec2-user@Couchbase02 ~]$ sudo -s
```

Change directories to the default bucket's data files:

```
[root@Couchbase02 ec2-user]# cd
/opt/couchbase/var/lib/couchbase/data/default/
```

Search for vBucket # 911 *(note, in your environment, this may be a different numbered bucket!)*:

```
[root@Couchbase02 default]# ls | grep 911
911.couch.1
```

Once you've located the relevant bucket, print its contents with the couch_dbdump command:

```
[root@Couchbase02 default]# /opt/couchbase/bin/couch_dbdump
/opt/couchbase/var/lib/couchbase/data/default/911.couch.1
Dumping "/opt/couchbase/var/lib/couchbase/data/default/911.couch.1":
Doc seq: 1
     id: key-7
     rev: 1
     content_meta: 131
     size (on disk): 17
     cas: 99994350654531, expiry: 0, flags: 0
     size: 7
     data: (snappy) value-7

Total docs: 1
```

Exit root:

```
[root@Couchbase02 default]# exit
exit
```

Excellent! You've successfully located key-7 in the underlying data files.
Next, run the cbstats command against the 1st 2nd ,3rd and 4th Node's public hostname to see how many vBuckets are currently assigned to it (this command should be run from the App Server/black node):

```
[ec2-user@AppServer ~]$ cbstats ec2-54-152-187-112.compute-
1.amazonaws.com:11210 all -b default | egrep -i
"active_num|replica_num"
vb_active_num:                                341
 vb_active_num_non_resident:                   0
 vb_replica_num:                               341
 vb_replica_num_non_resident:                  0

[ec2-user@AppServer ~]$ cbstats ec2-54-175-147-194.compute-
1.amazonaws.com:11210 all -b default | egrep -i
"active_num|replica_num"
 vb_active_num:                               342
 vb_active_num_non_resident:                   0
 vb_replica_num:                               341
 vb_replica_num_non_resident:                  0
```

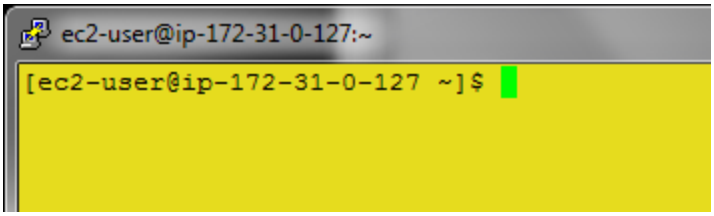
```
[ec2-user@AppServer ~]$ cbstats ec2-54-208-111-134.compute-
1.amazonaws.com:11210 all -b default | egrep -i
"active_num|replica_num"
Could not connect to ec2-54-208-111-134.compute-1.amazonaws.com:11210:
Connection refused

[ec2-user@AppServer ~]$ cbstats ec2-54-209-38-55.compute-
1.amazonaws.com:11210 all -b default | egrep -i
"active_num|replica_num"
 vb_active_num:                             341
 vb_active_num_non_resident:                0
 vb_replica_num:                            342
 vb_replica_num_non_resident:               0
[ec2-user@AppServer ~]$
```

The 1st node in the cluster now has about 342 active vBuckets and 341 replica vBuckets buckets. This makes sense because 342 vBuckets * 3 nodes = 1,026. There are a total of 1024 vBuckets in Couchbase, but when there is an odd number of nodes, Couchbase will make a best effort to distribute the vBuckets evenly. In this case, the other 2 remaining nodes will have 341 active vBuckets, so 342 + 341 + 341 = 1024. The number of vBuckets you see on the 1st node in your environment could vary slightly.

Failover the 4th node in your cluster:

Finally, let's stop the Couchbase service on the 4th Node as well, so that all of the servers in Server Group #2 will be decommissioned.

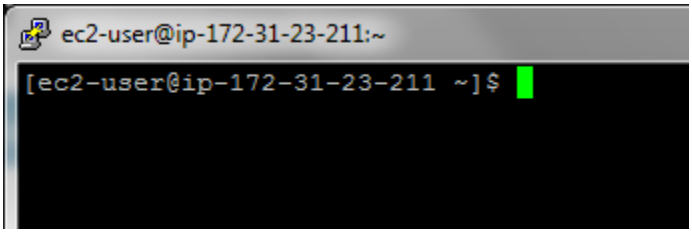
Switch to the 4th node's (Yellow VM) PuTTY window and stop the Couchbase service on the node:



Run the following commands to stop Couchbase and check its status:

```
[ec2-user@Couchbase04 ~]$ sudo systemctl stop couchbase-server

[ec2-user@Couchbase04 ~]$ sudo systemctl status couchbase-server
Aug 10 13:53:09 node4 systemd[1]: Stopped LSB: couchbase server.
```

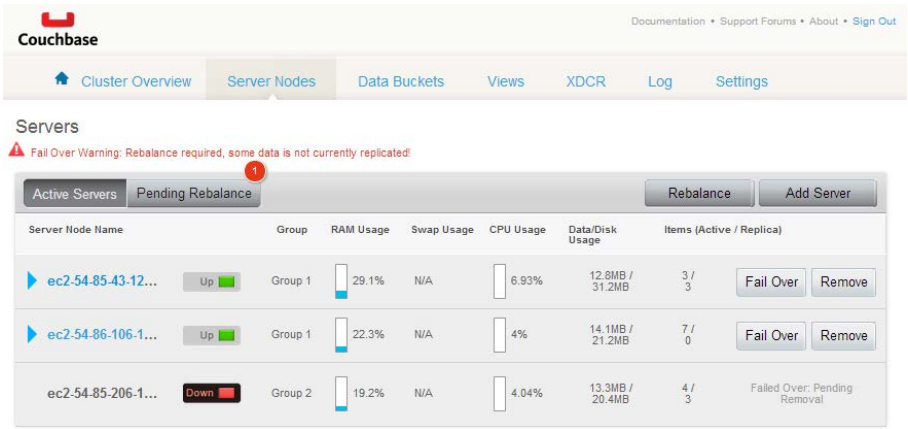
This time, we'll fail over Node #4 using the Couchbase CLI. Switch to the App Server (black VM):



Run the following Couchbase CLI failover command to failover Node #4 (yellow VM). Note that in this command there are two public hostname's provided. For the --cluster hostname use the public hostname of Node #1. For the --server-failover hostname, provide the public hostname for Node #4:

```
[ec2-user@AppServer ~]$ couchbase-cli failover --cluster=ec2-54-85-43-
x.compute-1.amazonaws.com:8091 --server-failover=ec2-54-85-206-
x.compute-1.amazonaws.com --user=Administrator --password=couchbase
SUCCESS: failover ns_1@ec2-54-85-206-193.compute-1.amazonaws.com
```

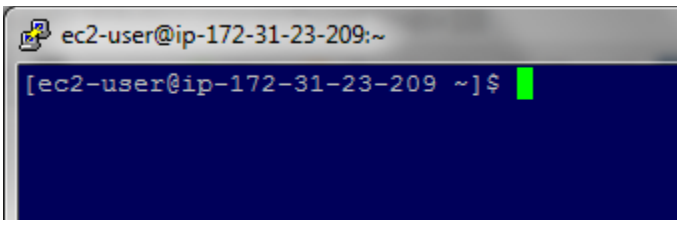
Once you see a success message, switch over to the Web UI and you should see a Down node #4 with a Rebalance pending:



Switch back to the App Server's command line and trigger a Rebalance operation using the Couchbase CLI as well. *(Note that the –cluster parameter should provide the public hostname of Node #1 and the –server-remove parameter should provide the public hostname for Node #4 (the node we want to remove))*:

```
[ec2-user@AppServer ~]$ couchbase-cli rebalance --cluster=ec2-54-85-
43-x.compute-1.amazonaws.com:8091 --server-remove=ec2-54-85-206-
x.compute-1.amazonaws.com --user=Administrator --password=couchbase
INFO: rebalancing . . . . . .
```

Which the rebalance is running, quickly switch to Node #1 (dark blue) and run the following command to check the status of the Rebalance operation:




```
[ec2-user@ip-172-31-23-209 ~]$ couchbase-cli rebalance-status --
cluster=ec2-54-85-43-x.compute-1.amazonaws.com:8091 --
user=Administrator --password=couchbase
(u'running', None)
```

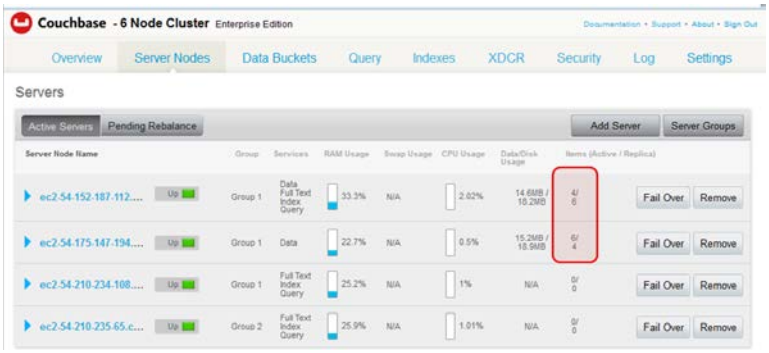
Eventually on the App Server node where you ran the Rebalance operation, you should see a message like this:

```
SUCCESS: rebalanced cluster
```

After you see the SUCCESS message, run the same rebalance-status command on Node #1 (dark blue VM):

```
[ec2-user@Couchbase01 ~]$ couchbase-cli rebalance-status --
cluster=ec2-54-85-43-x.compute-1.amazonaws.com:8091 --
user=Administrator --password=couchbase
(u'notRunning', None)
```

Great, the rebalance operation is definitely complete. The web UI should also reflect the change:



Notice that in the Web UI, now there are 5 active items and 5 replica(possibly 6/4) items on each of the 2 remaining nodes in Server Group #1.

Run the cbstats command from the App Server (black VM) and provide the first Node's public hostname:

```
[ec2-user@AppServer ~]$ cbstats ec2-54-85-43-x.compute-
1.amazonaws.com:11210 all -b default | egrep -i
"active_num|replica_num"
 vb_active_num:                     512
 vb_active_num_non_resident:        0
 vb_replica_num:                    512
 vb_replica_num_non_resident:       0
```

You can see that the rebalance operation has placed 512 active vBuckets and 512 replica vBuckets on each of the remaining 2 nodes.
Finally, try querying for the 10 keys we originally inserted to verify that they are still available after a loss of an entire Server Group. You can run all of the remaining commands on the App Server and provide it with the 1st node's public hostname:

```
[ec2-user@ip-172-31-23-211 ~]$ cbc-cat -U couchbase://ec2-54-172-130-
66.compute-1.amazonaws.com/default key-1
key-1               CAS=0x44ed48769d920400, Flags=0x0, Datatype=0x0
value-1

[ec2-user@ip-172-31-23-211 ~]$ cbc-cat -U couchbase://ec2-54-172-130-
66.compute-1.amazonaws.com/default key-2
key-2               CAS=0x780fc549ab920400, Flags=0x0, Datatype=0x0
value-2

[ec2-user@ip-172-31-23-211 ~]$ cbc-cat -U couchbase://ec2-54-172-130-
66.compute-1.amazonaws.com/default key-3
key-3               CAS=0xfd57a352bc920400, Flags=0x0, Datatype=0x0
value-3

[ec2-user@ip-172-31-23-211 ~]$ cbc-cat -U couchbase://ec2-54-172-130-
66.compute-1.amazonaws.com/default key-4
key-4               CAS=0xe3ae3cfbd1920400, Flags=0x0, Datatype=0x0
value-4

[ec2-user@ip-172-31-23-211 ~]$ cbc-cat -U couchbase://ec2-54-172-130-
66.compute-1.amazonaws.com/default key-5
key-5               CAS=0xb62d16b992380000, Flags=0x0, Datatype=0x0
value-5

[ec2-user@ip-172-31-23-211 ~]$ cbc-cat -U couchbase://ec2-54-172-130-
66.compute-1.amazonaws.com/default key-6
key-6               CAS=0x4570e03ae0920400, Flags=0x0, Datatype=0x0
value-6
```

```
[ec2-user@ip-172-31-23-211 ~]$ cbc-cat -U couchbase://ec2-54-172-130-
66.compute-1.amazonaws.com/default key-7
key-7                CAS=0x1da067b2e1920400, Flags=0x0, Datatype=0x0
value-7

[ec2-user@ip-172-31-23-211 ~]$ cbc-cat -U couchbase://ec2-54-172-130-
66.compute-1.amazonaws.com/default key-8
key-8                CAS=0x713bce5b3e930400, Flags=0x0, Datatype=0x0
value-8

[ec2-user@ip-172-31-23-211 ~]$ cbc-cat -U couchbase://ec2-54-172-130-
66.compute-1.amazonaws.com/default key-9
"key-9" Size:7 Flags:0 CAS:31ae17ac0ca60000
value-9

[ec2-user@ip-172-31-23-211 ~]$ cbc-cat -U couchbase://ec2-54-172-130-
66.compute-1.amazonaws.com/default key-10
key-10               CAS=0x439796f256930400, Flags=0x0, Datatype=0x0
value-10

[ec2-user@ip-172-31-23-211 ~]$ cbc-cat -U couchbase://ec2-54-172-130-
66.compute-1.amazonaws.com/default key-11
key-11               The key does not exist on the server (0xd)
```

In the above output, you will see that all of the 10 keys are successfully found. If a key was not found (as in key-11), you would have seen a "Failed to get key" error message.

If you have finished this and have time remaining, remove server

This concludes Lab #4. In the next lab, we will rejoin Nodes #3 and #4 back into the cluster and study server warmup.