CS300 Couchbase NoSQL Server Administration

Lab 7 Exercise Manual

Release: 4.5

Revised: July 26th, 2016

Lab #7: Advanced XDCR and Backup/Restore

Objective: This 1 hour lab will walk you through a couple of more advanced XDCR scenarios and then cover how to backup a bucket's files and restore it to a live cluster.

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

- Study optimistic replication in further detail
- Use pillowfight to insert 100,000 items UNDER the optimistic replication threshold into the 6-node NYC cluster so that the metadata is NOT checked on the NYC side before optimistically replicating it to the destination cluster
- Use pillowfight to insert 100,000 items OVER the optimistic replication threshold into the 6-node NYC cluster so that the metadata is checked on the NYC side before replicating it to the destination cluster
- Demonstrate that XDCR replication keeps occurring to remaining nodes even after a node goes down and fails over
- Learn how to use the vbucketkeygen tool to insert 1 key into each of the 1024 vBuckets
- Learn about cbbackup and cbrestore
- Use cbbackup to backup data in a bucket to some files
- Use cbrestore to restore the files back into a live bucket

Studying Optimistic Replication:

Remember that the Optimistic Replication threshold in our bidirectional replication stream is set to the default of 256 bytes. This parameter is basically used to tune the tradeoff between latency and bandwidth. If you want the fastest latency, you will have to sacrifice some bandwidth. On the other hand, if you want to conserve bandwidth, you will have to suffer some latency.

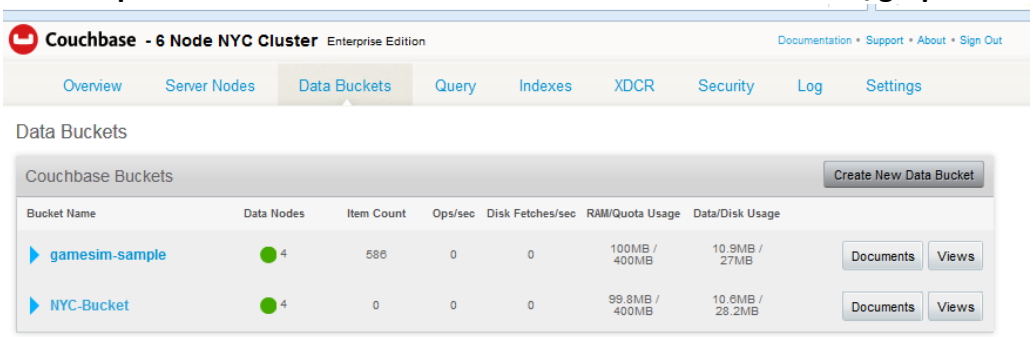
If the item that you write/update to either side cluster (NYC or London) is under 256 bytes, then the item is optimistically just thrown into the destination cluster's replication queue. However if the item you write/update is larger than 256 bytes, then the source cluster will first do a read of the same key's metadata over the WAN (Wide Area Network) to check if the destination cluster would win the conflict resolution. If the destination cluster's revision count is lower than the source cluster (where you did the write/update), then the source will send the item all the way down the WAN to the destination cluster. Before the destination cluster pushes that write/update to disk, it will do another metadata read locally to make sure that while the item was transiting over the WAN it didn't get updated with a higher revision count locally. If the destination cluster decides that the item it got from the source is still the higher revision count, then the destination cluster pushes that item into its local disk write queue.

At this point, NYC-bucket and London-bucket should both be empty with zero items in either bucket.

Let's use pillowfight from the App Server to first send 100,000 keys of size 200 bytes (under the optimistic replication threshold) to the 6-node NYC cluster. In this case, the NYC cluster will NOT attempt to query metadata for any of the items from the remote side. It will just optimistically push all 100,000 items over to the London side and let London's cluster deal with it.

Later in the lab we will also send keys of size over 256 bytes to see how that has a different effect in the cluster.

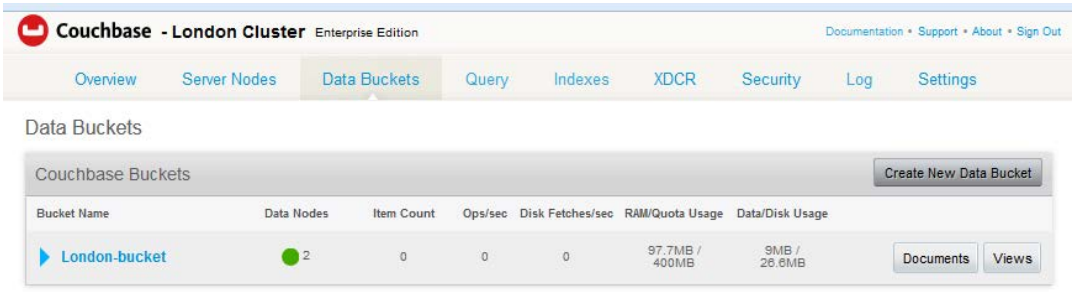
Before we generate any data, let's set up our Web UI properly to see the effects of optimistic replication. Switch browser tabs to the 6-node NYC cluster's Web UI, click on "Data Buckets" at the top menu and click on the "NYC-bucket" to load the charts/graphs for this bucket:



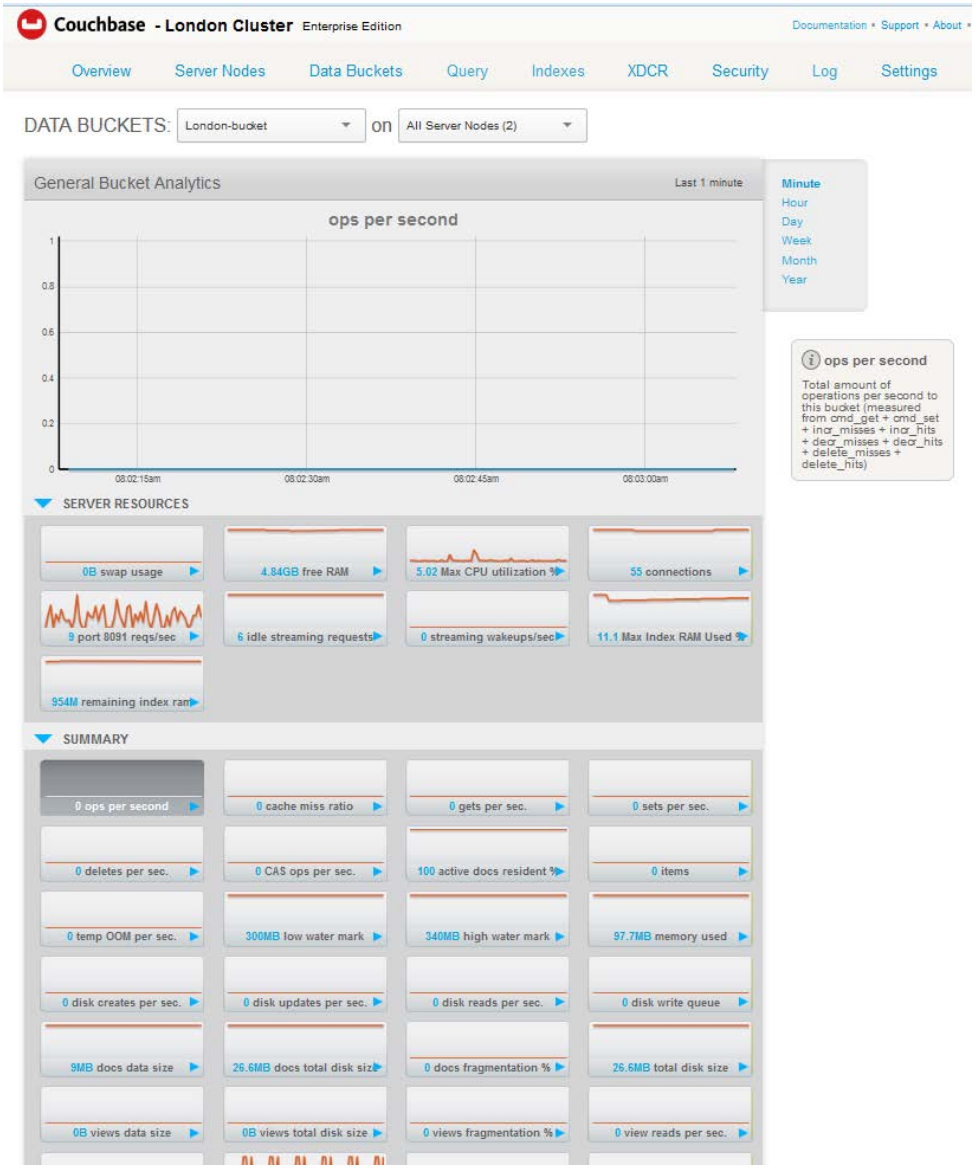
In the first tab, you should be seeing metrics for the "NYC-bucket" for "All Server Nodes":

Couchbase

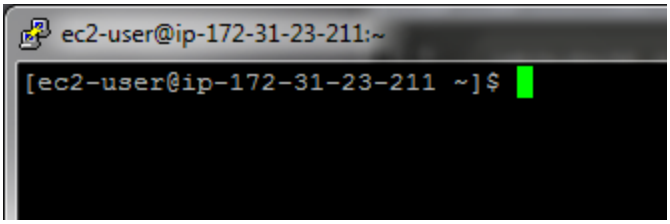
Next, switch to the 2nd browser tab for the 2-node London cluster's Web UI, click on "Data Buckets" at the top menu and click on the "London-bucket" to load the charts/graphs for this bucket:



In the second tab, you should be seeing metrics for the "London-bucket" for "All Server Nodes":



Switch to the App Server (black VM):

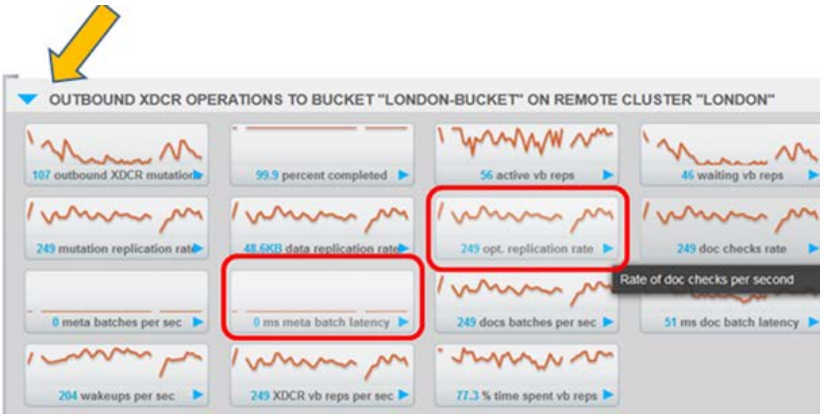


Run pillowfight to write 100,000 items, in 10 iteration, with a 100% write (set) ratio and a maximum payload size of 200 bytes. There will be a key prefix of "from_NYC" in front of all the keys. Use the public hostname of the 1st node in the 6-node NYC cluster in the command. So, we will push the writes into the 6-node NYC cluster and expect them to be optimistically replicated into the 2-node London cluster without first checking the metadata.

```
[ec2-user@ip-172-31-23-211 ~]$ cbc-pillowfight -U couchbase://ec2-54-172-130-66.compute-1.amazonaws.com/NYC-bucket  --num-items=100000 --batch-size=10000 --set-pct=100 --min-size=50 --max-size=200 --timings --num-cycles=10 -p from_NYC
```

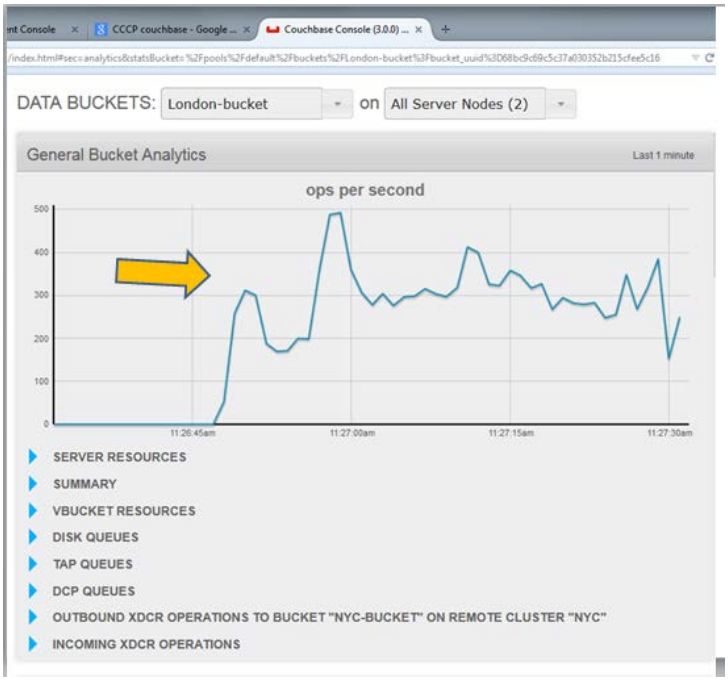
While the above command is generating data, quickly switch to the 6-node NYC cluster's Web UI and observe that the cluster is receiving 200-500 writes per second across 4 nodes from the App Server:

Then, in the same 6-node cluster's Web UI, scroll down and click the blue arrow to expand "OUTBOUND XDCR OPERATIONS" to the London-bucket:

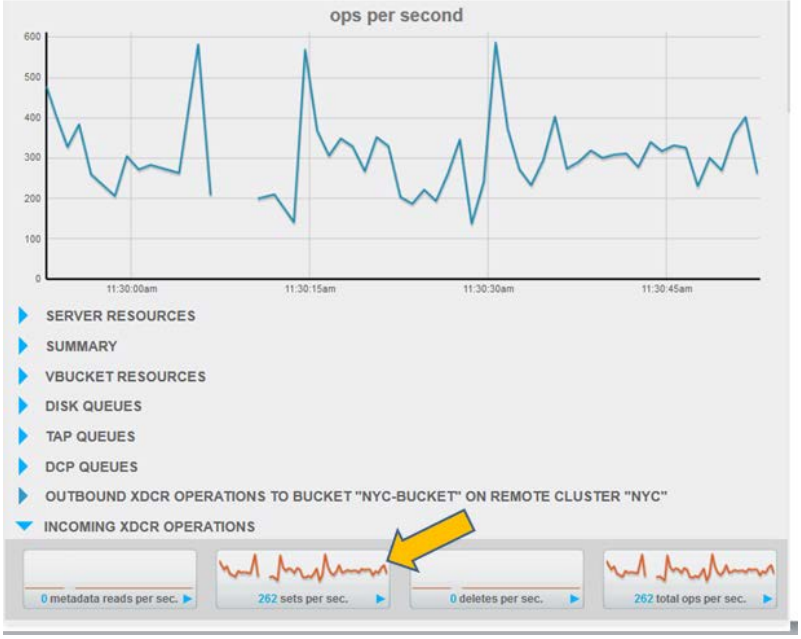


Observe in the screenshot above that the graph in the 2nd row, 3rd column shows mutations replicated optimistically so far. The graph in the 3rd row, 2nd column shows 0 millisecond latency for metadata operations. This is because there are no metadata reads occurring on the NYC side! The NYC cluster is just optimistically sending the items over to London right away. This is us favoring latency over bandwidth, b/c the 100,000 items will be sent to London ASAP.

Next, switch to the 2nd browser tab for the 2-node London cluster and in the large "ops per second graph" observe that it is receiving around 300-500 operations (writes) per second from the NYC side:



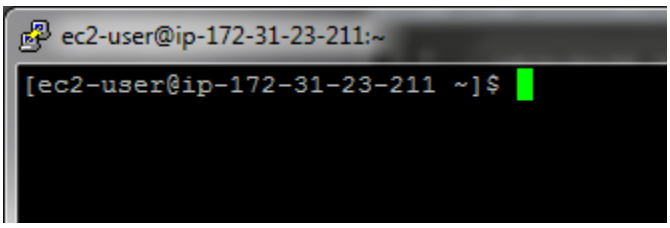
Scroll down on the same page, and expand "INCOMING XDCR OPERATIONS" and notice that the London cluster is getting around 200-300 sets per second (at least in my specific cluster at the time I captured the screenshot):



The "metadata reads per sec" graph in the 1st column above is NOT referring to metadata reads occurring on the destination/London side, but rather metadata reads happening from source/NYC. Since the 6-node NYC cluster is not doing any metadata reads, this "metadata reads per sec" graph is displaying zero.

Next we'll do a similar test of writing 100,000 keys into the NYC cluster with pillowfight, but this time we'll set the item size to 300 bytes, well above the optimistic replication threshold. And actually we won't write 100,000 new keys, but rather update the 100,000 keys from before (so we'll use the same key prefix). This time the NYC cluster will have to keep doing remote metadata reads a total of 100,000 times before sending each item to the destination (London) cluster.

Switch to the App Server (black VM):

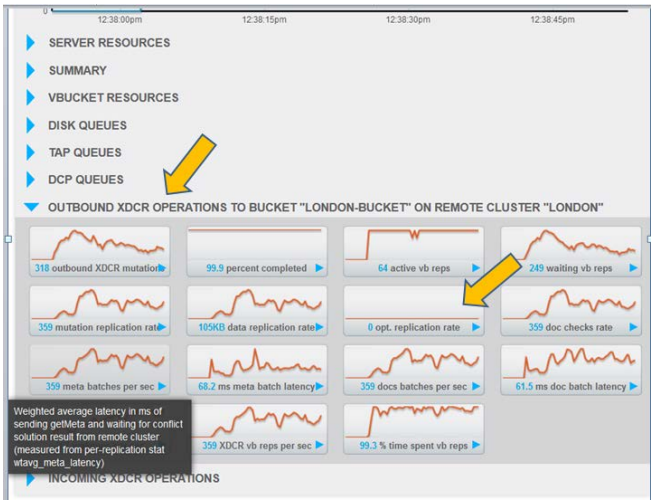


Run pillowfight to update the 100,000 items, in 10 iterations, with a 100% write (set) ratio and a maximum payload size of 10,000 bytes. There will be a similar key prefix of "from_NYC" in front of all the keys. Use the public hostname of the 1st node in the 6-node NYC cluster in the command. So, we will push the writes into the 6-node NYC cluster and expect 100,000 metadata reads to happen remotely from the 2-node London cluster before replicating the items over to the 2-node London cluster.

```
[ec2-user@ip-172-31-23-211 ~]$ cbc-pillowfight -U couchbase://ec2-54-172-130-66.compute-1.amazonaws.com/NYC-bucket  --num-items=100000 --batch-size=10000 --set-pct=100 --min-size=300 --max-size=300 --timings --num-cycles=10 -p from_NYC
```

Note that in the above command if you increase the item size to just barely over the optimistic threshold, like to 300 bytes, then optimistic replication won't technically kick in as this is an approximate setting. So, in our case, we are increasing the item size well over the 256 byte setting to 10,000 bytes to make sure that optimistic replication does not kick in at all for these keys. To be more technically accurate, for checking the optimistic replication threshold, Couchbase checks the compressed size of an item on disk when deciding whether to send it optimistically or not. However, it then actually sends the UNCOMPRESSED data over the wire.
Remember, we are writing 100,000 keys with the exact same key prefix, so we are actually doing updates to the existing 100,000 keys from the previous run of pillowfight.

Anyway, switch to the 6-node NYC cluster's Web UI and look at the "OUTBOUND XDCR OPERATIONS" to the London-bucket:

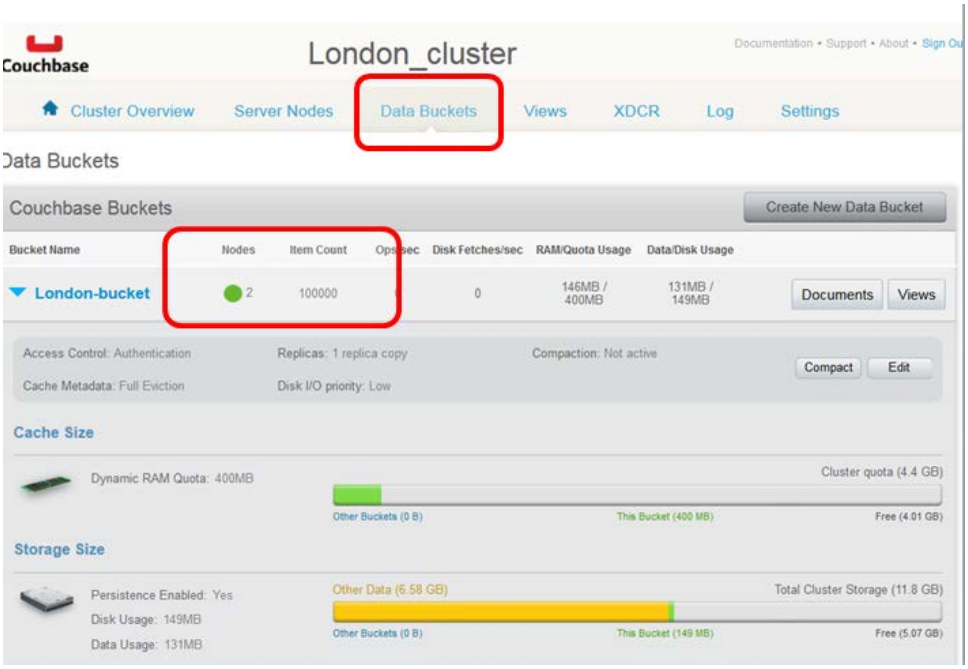


Notice in the screenshot above that the graph in the 2nd row, 3rd column shows no optimistic rate of replication. This is because there are now metadata reads occurring on the NYC side!

Switch to the 2-node London cluster's Web UI and look at the "INBOUND XDCR OPERATIONS":

Observe here also that there are around 250-350 metadata reads per second happening. This is the 6-node NYC cluster doing metadata reads from London. Also the # of metadata reads will equal the sets per second, which is shown in the 2nd graph.

When the pillowfight write is completely finished, remain in the 2-node London cluster's web UI and click on "Data Buckets" at the top:



You will see a total of 100,000 items in the London-bucket. This is because the 2nd time we ran pillowfight in this lab, the first 100,000 items written were actually updated.

Demonstrate that XDCR Replication keeps occurring to remaining nodes even after a node goes down and fails over:

Before switching over to backup and restore concepts, we will demonstrate one last XDCR concept.

Currently you should have bidirectional, encrypted replication set up between 6-node NYC and 2-node London. There are 100,000 keys with the prefix 'from_NYC' in buckets on both sides.

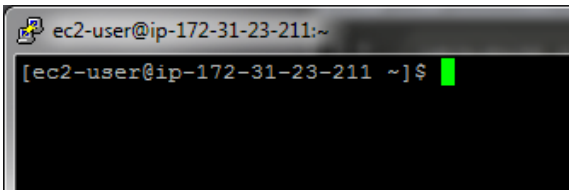
What do you think will happen if while you're inserting keys into the NYC cluster there is a node failure on one of the 2 nodes in London? If the 2nd node in London crashes, about half of the active vBuckets in London will be down and therefore half of the data from NYC can no longer be sent via XDCR to London.

But if we failover the node that crashed in London (2nd Node), then we would make the corresponding replicas on Node #1 in London active, thereby activating all 1024 vBuckets on the first London node.

In this section, you will then see that the items that could not be replicated temporarily while the 512 vBuckets in London were down, will suddenly start to be replicated to the newly promoted 512 vBuckets on the 1st node.
To demonstrate this concept we will use the vbucketkeygen tool to write 1 key to each of the 1024 vBuckets on the 6-node NYC cluster.

First, let's figure out how to use a combination of the Couchbase tools 'vbucketkeygen' and 'cbc create' along with the Linux tools curl, awk, echo and pipes to write exactly one key to each of the 1024 vBuckets in the NYC-bucket. This will essentially require us to write a simple script.

Switch to the App Server (black VM):



First pull the help menu for vbucketkeygen to learn how to use it:

```
[ec2-user@ip-172-31-23-211 ~]$ /opt/couchbase/bin/tools/vbucketkeygen
-h
vbucketkeygen mapfile <keys per vbucket> <keys to generate>

  vbucketkeygen will output a list of keys that equally
    distribute amongst every vbucket.

  vbucketkeygen expects a vBucketServerMap JSON mapfile, and
  will print the keyname and vBucketId.
  You may use '-' instead for the filename to specify stdin.

  Examples:
    ./vbucketkeygen file.json 10 10000

    curl http://HOST:8091/pools/default/buckets/default | \
       ./vbucketkeygen - 5 10000
```

Notice in the yellow highlighted section above, that you have to give this tool a mapfile, then 2 parameters: the keys per vbucket and how many keys to generate. This essentially means that we can generate 10,000 keys and then if the 'keys per vbucket' parameter is set to 1, we will randomly pick 1,024 of the 10,000 keys and push 1 key into each of the 1,024 vBuckets. This would NOT MEAN that we are going to distribute 10,000 keys evenly into 1,024 vBuckets! Instead, it would mean that we are going to distribute 1,024 random keys (from a seed pool of 10,000 keys) evenly into 1,024 vBuckets.

Run the following command to provide the output from curl as the mapfile and then generate 10,000 keys and place 1 of those keys into each of the 1024 vBuckets. Provide this command with the public hostname of Node #1 (dark blue VM in the 6-node NYC cluster).

```
[ec2-user@ip-172-31-23-211 ~]$ curl -u Administrator:couchbase
http://ec2-54-85-43-x.compute-
1.amazonaws.com:8091/pools/default/buckets/NYC-bucket |
/opt/couchbase/bin/tools/vbucketkeygen - 1 10000
  % Total    % Received % Xferd  Average Speed   Time    Time     Time
Current
                                 Dload  Upload   Total   Spent    Left
Speed
100 12275  100 12275    0     0   328k      0 --:--:-- --:--:-- --:--
:-- 1712k
key_0000008638 0
key_0000001149 1
key_0000000146 2
key_0000001320 3
key_0000000093 4
key_0000001365 5
key_0000000418 6
<output truncated>
```

Notice in the above output that key # ending 8638 would go into vBucket 0.

Now the above command does NOT actually push the keys into the vBucket. It is just a generator.

We have to pipe the output from the above command to series of other commands including 'cbc create' to actually push the keys into Couchbase.

In the following command, we are inserting 1 key into each of the 1,024 vBuckets in the bucket NYC-bucket in the 6-node NYC cluster. We are going to prefix each key with AA.key#. Prefixing with "AA." will lexographically sort all of the keys in the Web UI's view, so it'll be easy to find all the AA keys from this generation. Provide this command with the public hostname of Node #1 (dark blue VM in the 6-node NYC cluster).

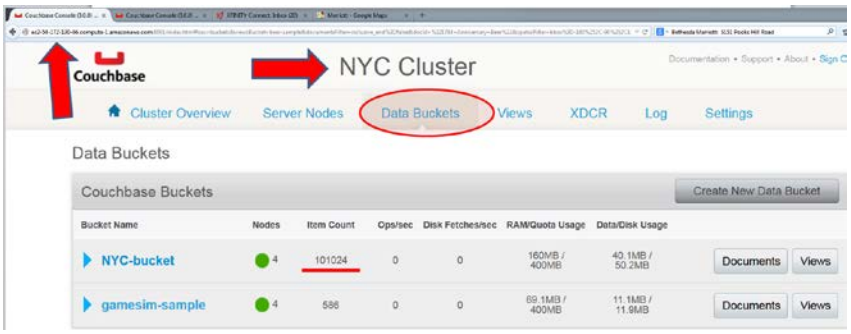
Note: the first command in Blue will work if using libcouchbase 2.3(we are not using this, move on to the next command)

```
[ec2-user@ip-172-31-23-211 ~]$ for i in `curl -u
Administrator:couchbase http://ec2-54-85-43-x.compute-
1.amazonaws.com:8091/pools/default/buckets/NYC-bucket |
/opt/couchbase/bin/tools/vbucketkeygen - 1 10000 | awk {'print $1'}`;
do echo $i; echo "value" -n | cbc create -h ec2-54-85-43-128.compute-
1.amazonaws.com AA.$i -b NYC-bucket; done
```

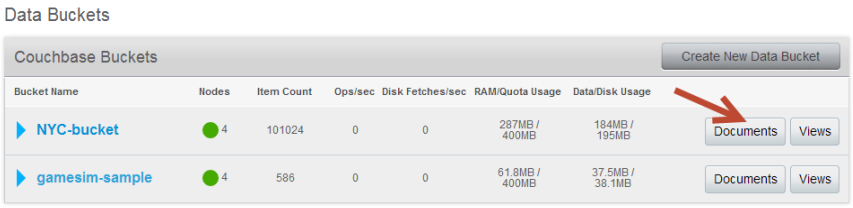
Note: the second command in red will work if using libcouchbase 2.4.2

```
[ec2-user@ip-172-31-23-211 ~]$ for i in `curl -u
Administrator:couchbase http://ec2-54-172-130-66.compute-
1.amazonaws.com:8091/pools/default/buckets/NYC-bucket |
/opt/couchbase/bin/tools/vbucketkeygen - 1 10000 | awk {'print $1'}`;
do echo $i; echo -n "value" | cbc-create -U couchbase://ec2-54-172-
130-66.compute-1.amazonaws.com/NYC-bucket AA.$i ; done

Stored "AA.key_0000000139" CAS:c33cc4a140170300
key_0000009648
Stored "AA.key_0000009648" CAS:6bca3ca440170300
key_0000000350
Stored "AA.key_0000000350" CAS:565d96a640170300
key_0000000832
Stored "AA.key_0000000832" CAS:93ac90a740170300
<output truncated>
```

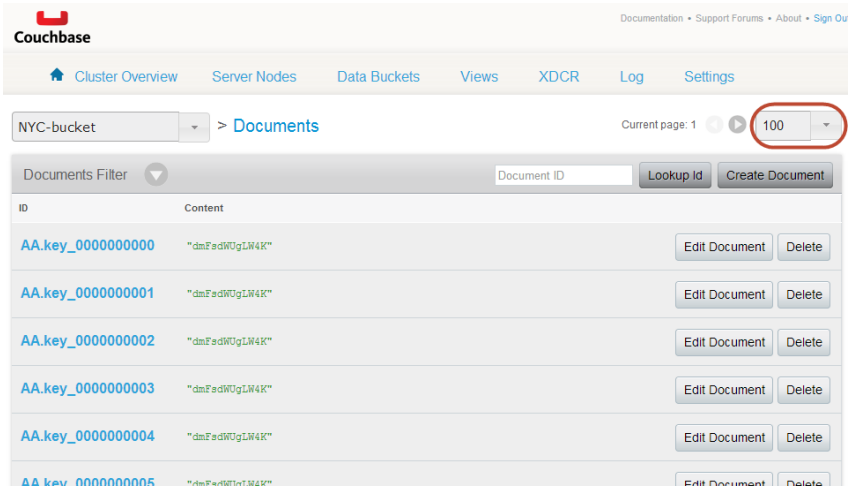
If you now switch to the browser tab for the 6-node NYC cluster and refresh the page for Data Buckets, you will see 101,024 keys in the NYC-bucket. We just added the last 1,024 keys.



Try clicking on Documents for the "NYC-bucket" to actually display the items:

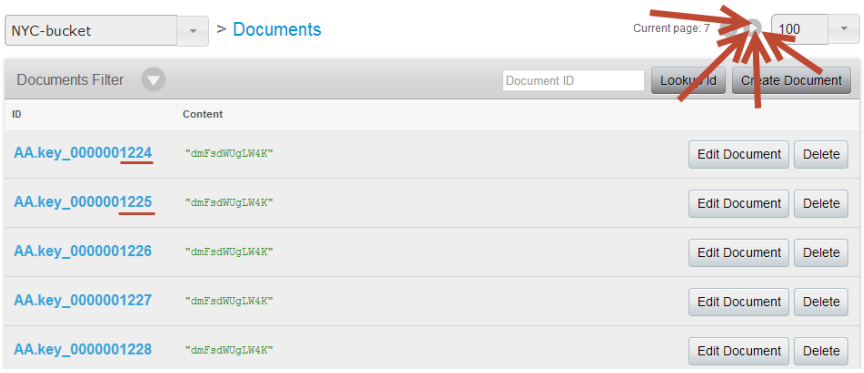


Here you will see the keys starting with "AA." that we just added. Change the dropdown for item count on the page in the top right corner from the default of 5 to 100:

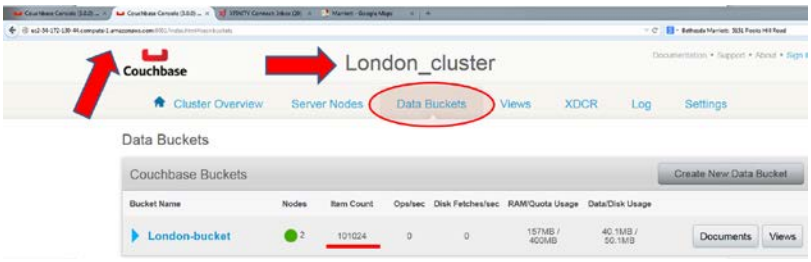


You may be surprised that all of the keys you see are in sequential order. For example the screenshot above shows keys 00 – 05 all in order. This does not mean that the script we ran inserts key 0 into vBucket 0 and key 24 into vBucket 24!

Try clicking the right arrow 7 times and you will start to see that the AA. keys can get higher than 1024:



Did the new 1,024 keys also get replicated to London? Switch to the browser tab for the 2-node London cluster, refresh the page and look at the Data Buckets page to see if the item count has increased:

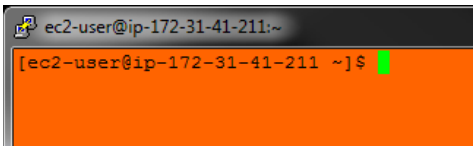


It has increased. So replication is also working as expected.

Now we will do something interesting.

Let's cause a simulated failure on the 2nd node in the London cluster.

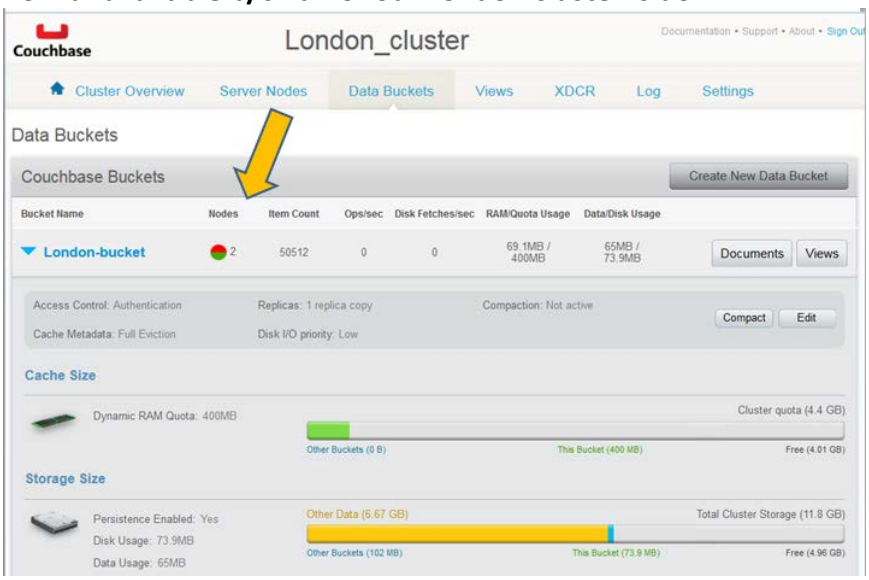
Log into the XDCR Node #2 in London (Orange VM):



We're going to pretend that the Couchbase service has crashed here… or pretend that the entire machine has crashed. We'll simulate this by simply stopping the Couchbase service here:

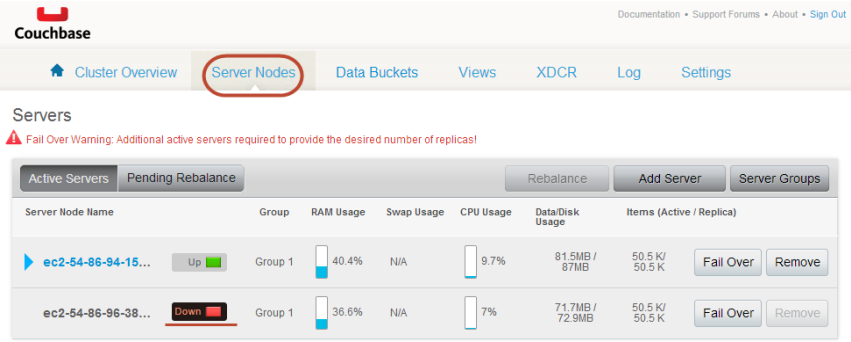
```
[ec2-user@ip-172-31-41-211 ~]$ sudo systemctl stop couchbase-server
Stopping couchbase-server
[ec2-user@ip-172-31-41-211 ~]$ sudo systemctl status couchbase-server
couchbase-server is not running
```

Switch back to the Web UI for the 2-node London cluster and under the Data Buckets page, notice that the icon for nodes is now half red and half green, signaling that half of the data is now unavailable b/c half of our London cluster is down.



Notice in the above screenshot also that the current Item Count is only 50,512 since half the active vBuckets are now missing! Keep this # in mind.

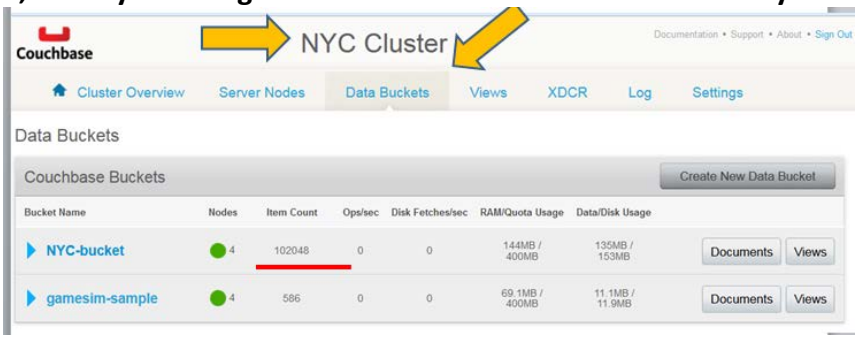
Click on Server Nodes at the top and notice that the 2nd machine is in a down state:



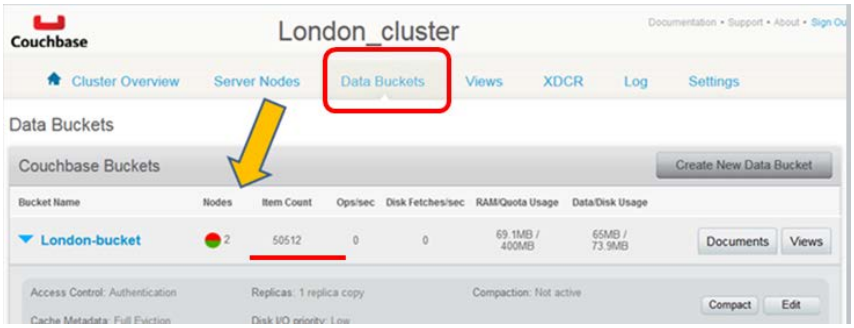
Switch back to the App Server (black VM). Now, let's try to write 1,024 keys into the 6-node NYC cluster and see how many of them successfully replicate to the 1-node London cluster. This time prefix the keys with "AB." b/c this is the 2nd time we're writing 1,024 keys. Still provide this command the public hostname of Node #1 in the 6-node NYC cluster.

```
[ec2-user@ip-172-31-23-211 ~]$ for i in `curl -u
Administrator:couchbase http://ec2-54-172-130-66.compute-
1.amazonaws.com:8091/pools/default/buckets/NYC-bucket |
/opt/couchbase/bin/tools/vbucketkeygen - 1 10000 | awk {'print $1'}`;
do echo $i; echo "value" -n | cbc-create -U couchbase://ec2-54-172-
130-66.compute-1.amazonaws.com/NYC-bucket AB.$i ; done
Stored "AB.key_0000000139" CAS:c8cb691fa630300
key_0000009648
Stored "AB.key_0000009648" CAS:eef64492fa630300
key_0000000350
Stored "AB.key_0000000350" CAS:21be4f93fa630300
key_0000000832
Stored "AB.key_0000000832" CAS:c85dda93fa630300
<output truncated>
```

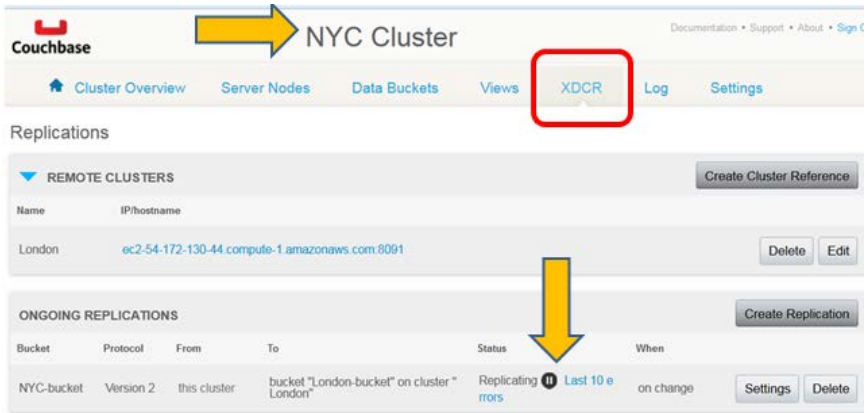
Switch to the browser tab for the 6-node NYC cluster, go to the Data Bucket page, refresh it and notice that the item count is 102,048. This makes sense. Originally the bucket had 100,000 keys (from the pillowfight command we ran in the prior section). Then we added 1,024 keys starting with AA. The new added 1024 more keys starting with AB.



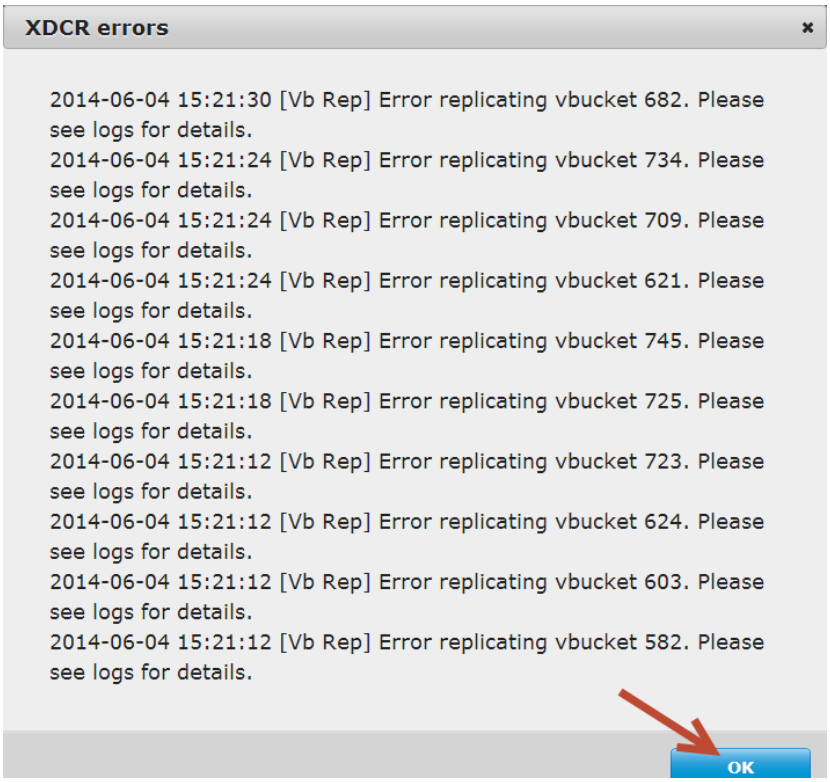
Switch to the browser tab for the 2-node London cluster (which is down to 1 healthy node), go to the Data Bucket page, refresh it and notice that the item count is 51,024. This makes sense also. Remember that after we lost 1 node in London, the item count went down from 101,024 to 50,512 (it became half). Then we replicated 512 or so more keys with the prefix AB from NYC -> London. So 50,512 + 512 = 51,024. The issue here is that there are 512 keys in a sort of "pending replication" state in the NYC cluster. The NYC cluster cannot send them to London b/c half the active vBuckets in London are missing. We have to Fail Over the down node in London to promote the replicas on the remaining node. Then the updated cluster map from London (with one node owning all 1,024 vBuckets) will be sent to all nodes in NYC and the 4 NYC nodes will push out the last 512 keys with prefix AB over to the London 1-node cluster. Note that a Rebalance is not required in London to get the cluster map updated on the NYC side.



Also, if you switch to the 6-node NYC cluster's Web UI, click on XDCR at the top, and click on "Last 10 errors" under ongoing replication, you will see that the NYC cluster cannot connect to certain vBuckets (512 of them) in London:

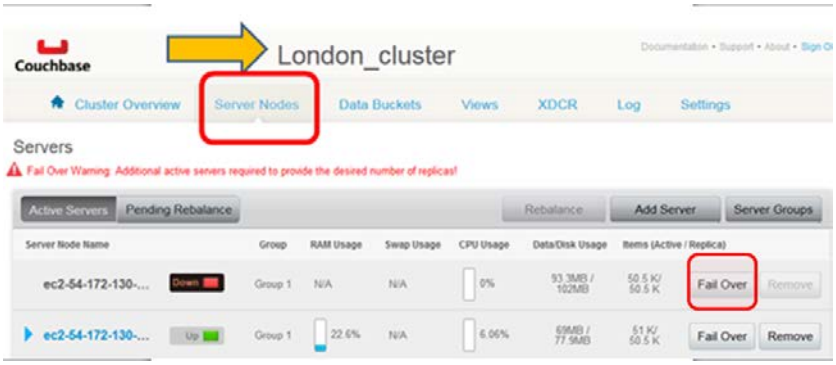


The time stamps that you see in the popup will be current (you can verify this by running the 'date' command on one of the 4 NYC nodes). The NYC clusters is RIGHT NOW complaining repeatedly that it cannot reach some vBuckets in London. This is not happening because our VM is running with only 1 CPU core. This time, the issue is real. Click on OK to close the popup.

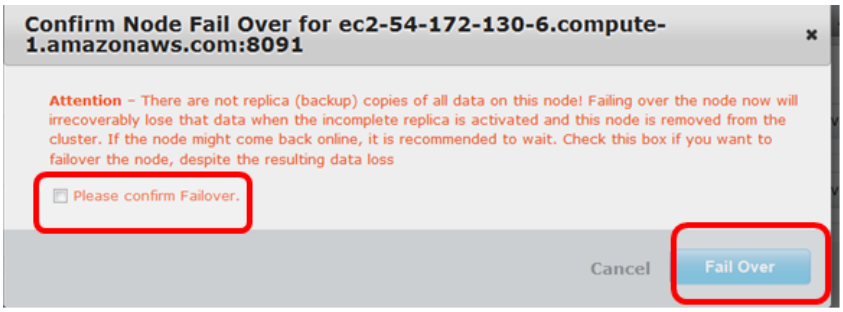


Switch back to the London cluster's Web UI, click on "Server Nodes" at the top, then click the "Fail Over" button next to the down node to promote its replicas on the remaining node. Remember that we configured the London-bucket with 1 replica (so 2 copies, one active and one replica).

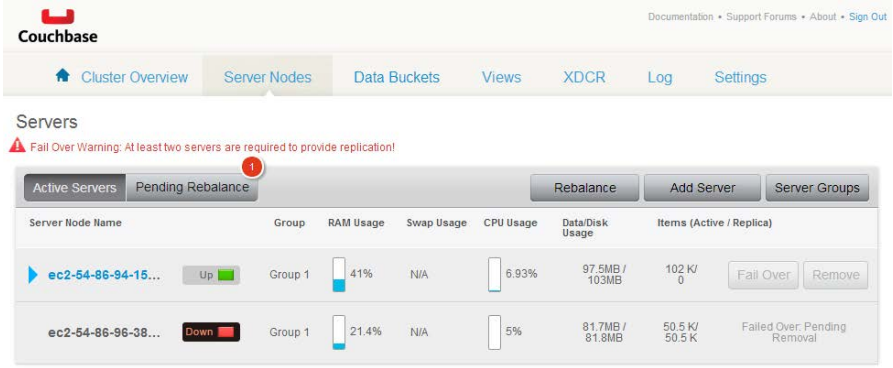
Side note: The Fail Over cannot be configured to occur automatically in London, since there are only 2 nodes in the cluster. There have to be at least 3 nodes in a Couchbase cluster before auto-failover can be enabled. This is because with a 2-node cluster, if failover was automatic, then a split-brain condition could have occurred. (Well, technically it is possible to configure auto-failover in a 2-node cluster, but it won't get triggered if a node goes down. If you grow the same cluster to 3+ nodes, then auto-failover will start to get triggered when a node goes down.)



On the pop-up, place a check next to "Please confirm Failover" and click "Fail Over":



Even though the GUI is saying that a Rebalance is required, this is not exactly accurate:



At this point, the updated cluster map from London (that one node owns all 1,024 vBuckets) has been sent to all 4 data nodes in NYC and the 4 data NYC nodes have started pushing out the last 512 keys with prefix AB over to the London 1-node cluster.

In this case, the Web UI is simply asking you to run a Rebalance to remove down node from the cluster. If you had clicked on Rebalance, no rebalancing would actually occur. The rebalance is being requested to permanently remove the down node from the vBucket cluster map.

Let's verify this.

In the London (now 1-node) cluster's Web UI, click on Data Buckets at the top and look at the item count:

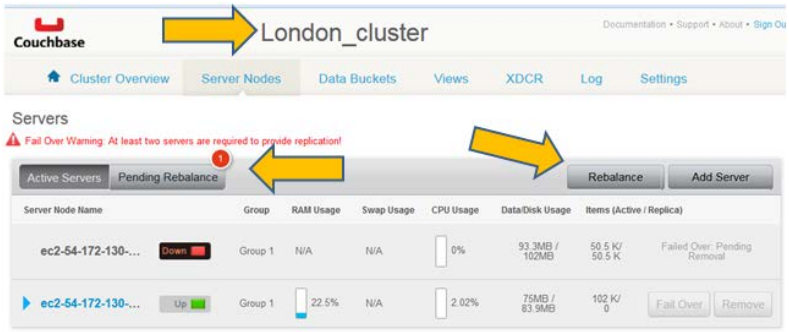
The item count now matches what we see in the 6-node NYC cluster. Perfect! (*If you don't see the same item count yet, you may need to wait 1 – 2 mins for XDCR to finish replicating and then refresh the page*)

This proves that as soon as you Failed Over the down node in London, the 6-node NYC cluster has the 512 vBuckets become active again on the remaining London node and then the 4 nodes in NYC started to replicate out those last 512 keys to London.

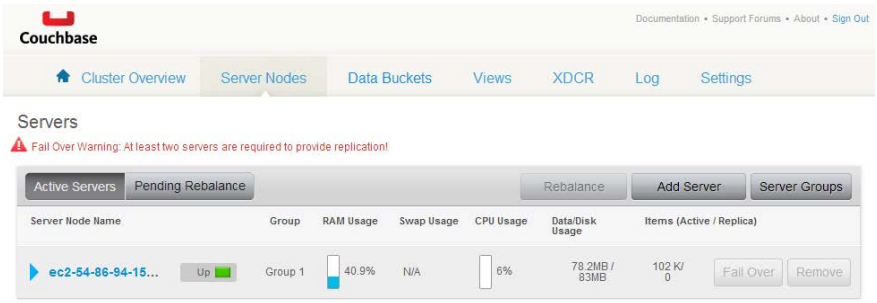
The steps we completed above demonstrate that if we had auto-failover configured in London with 3+ nodes in the London cluster, then even if we lose a node, replication would eventually resume automatically after the auto-failover is triggered.

Before concluding this section, let's Rebalance the London cluster so that the Web UI shows only 1 node, then we'll re-add the down node back into the cluster and do another rebalance so that the London cluster becomes a healthy 2 node cluster again.

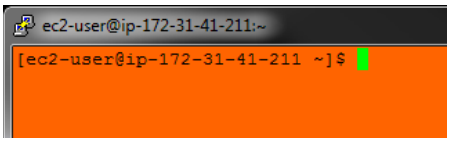
Switch to the browser tab for the London cluster (currently 1 node), and click on "Server Nodes" at the top and then hit Rebalance:



The rebalance will finish in just a few seconds as there is really nothing to rebalance:



Log into the XDCR Node #2 in London (Orange VM):

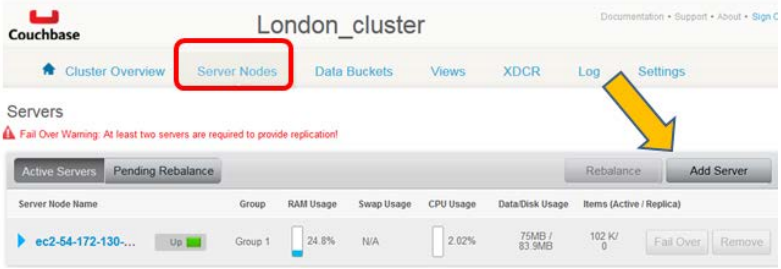


This is the node in London that we had previously stopped the Couchbase service on. Verify that the Couchbase service is still stopped and then start it again:

```
[ec2-user@ip-172-31-41-211 ~]$ sudo systemctl status couchbase-server
couchbase-server is not running
```

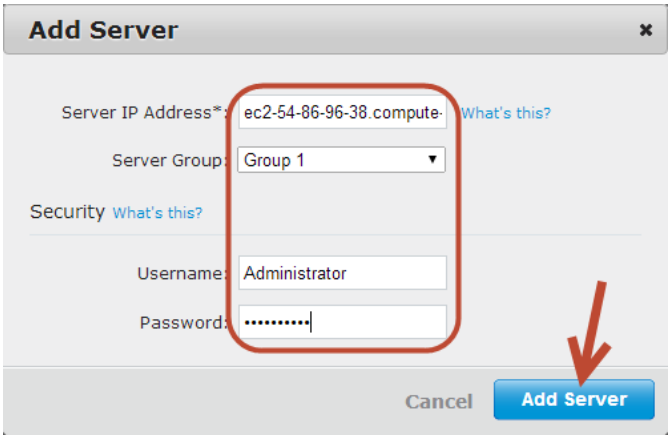
```
[ec2-user@ip-172-31-41-211 ~]$ sudo systemctl start couchbase-server
Starting couchbase-server                                  [  OK  ]
```

Then switch back to the browser tab for the London cluster, and let's add this Node #2 back into the cluster and rebalance it. From the Web UI for the London cluster (currently 1 node), from the Server Nodes page, click on the Add Server button:

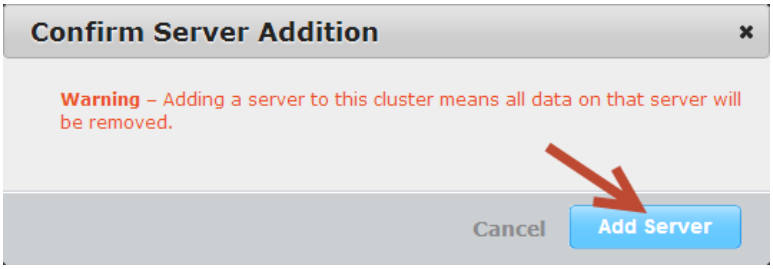


On the Add Server pop-up, type in:

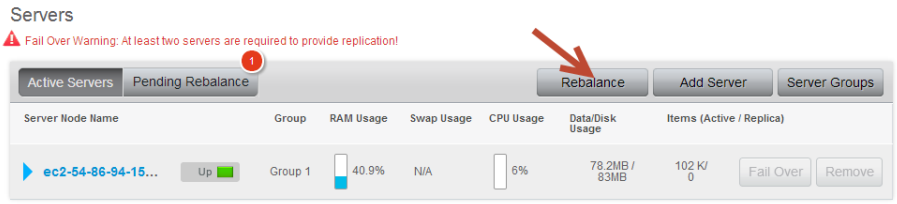
- **Server IP Address: <public hostname for XDCR Node #2 in London (Orange VM)**
- **Server Group: Group 1**
- **Username: Administrator**
- **Password: couchbase2**
- **Click "Add Server"**



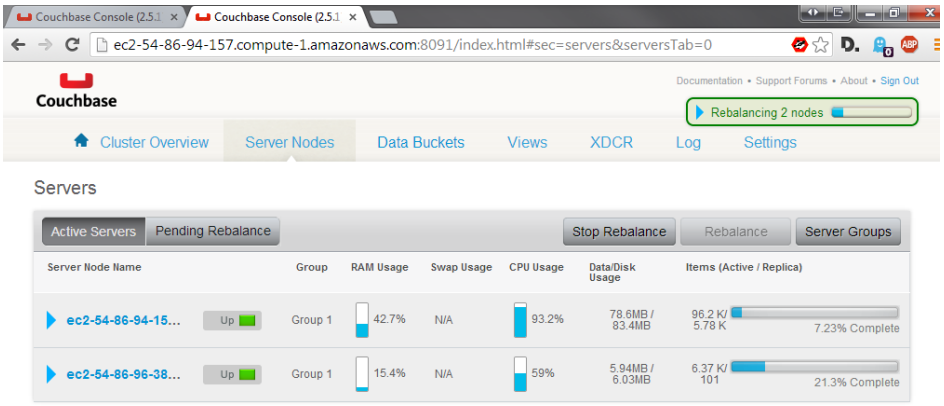
Confirm the server addition by clicking Add Server in the popup:



You will see a #1 under Pending Rebalance now as the new node has been added to the cluster, but the active and replica vBuckets still have to be redistributed across the two nodes. Click on Rebalance:



The rebalance will start and should take about 2 – 3 minutes to complete:



Go ahead and continue with the next section, but keep an eye on the Rebalance to make sure it successfully finishes!

Backing up data from the 6-node NYC cluster and restoring the data on the 2-node London cluster:

Let's now switch over to backup and restore concepts.

Backing up your data should be a regular process on your cluster to ensure that you do not lose information in the event of a serious hardware or installation failure.

There are two methods for performing a backup:

1) **cbbackup:** this command enables you to back up a single node, single buckets, or the entire cluster into a flexible backup structure that allows for restoring the data into the same, or different, clusters and buckets. All backups can be performed on a live cluster or node. Using cbbackup is the most flexible and recommended backup tool.

2) **file copies:** A running or offline cluster can be backed up by copying the files on each of the nodes. Using this method you can only restore to a cluster with an identical configuration.

In this section, we will use the preferred cbbackup tool for backups.

Note: Due to the active nature of Couchbase Server it is impossible to create a complete in-time backup and snapshot of the entire cluster. Because data is always being updated and modified, it would be impossible to take an accurate snapshot. It is a best practice to backup and restore your entire cluster to minimize any inconsistencies in data. Couchbase is always per-item consistent, but does not guarantee total cluster consistency or in-order persistence.

The cbbackup tool is a flexible backup command that enables you to backup both local data and remote nodes and clusters involving different combinations of your data:

- Single bucket on a single node
- All the buckets on a single node
- Single bucket from an entire cluster
- All the buckets from an entire cluster

Backups can be performed either locally, by copying the files directly on a single node, or remotely by connecting to the cluster and then streaming the data from the cluster to your backup location. Backups can be performed either on a live running node or cluster, or on an offline node.

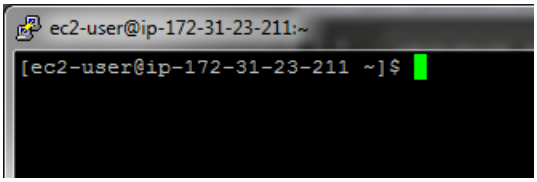
The cbbackup command stores data in a format that allows for easy restoration. When restoring, using cbrestore, you can restore back to a cluster of any configuration. The source and destination sizes clusters do not need to match if you used cbbackup to store the information.

The cbbackup command will copy the data from the source definition to a destination backup directory. The backup file format is a SQLite file Couchbase and enables you to restore, all or part of the backed up data when restoring the information to a cluster. Selection can be made on a key (by regular expression) or all the data stored in a particular vBucket ID. You can also select to copy the source data from a bucketname into a bucket of a different name on the cluster on which you are restoring the data.

The cbbackup command takes the following arguments:

```
cbbackup [options] [source] [backup_dir]
```

Switch to the App Server (black VM):



Run the following command to print the help menu for cbbackup:

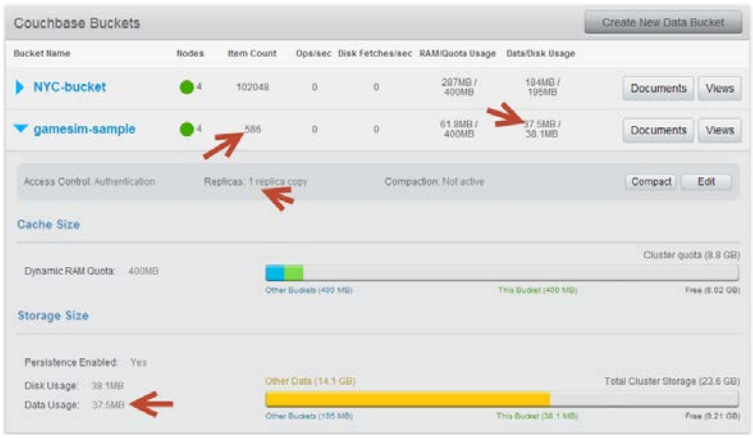
```
[ec2-user@ip-172-31-23-211 ~]$ cbbackup -h
Usage: cbbackup [options] source backup_dir

Online backup of a couchbase cluster or server node.

Examples:
  cbbackup http://HOST:8091 /backups/backup-42
  cbbackup couchbase://HOST:8091 /backups/backup-42

Options:
  -h, --help            show this help message and exit
  -b BUCKET_SOURCE, --bucket-source=BUCKET_SOURCE
                        single bucket from source to backup
  --single-node         use a single server node from the source only,
                        not all server nodes from the entire cluster;
                        this single server node is defined by the source URL
  -i ID, --id=ID        Transfer only items that match a vbucketID
  -k KEY, --key=KEY     Transfer only items with keys that match a regexp
  -n, --dry-run         No actual transfer; just validate parameters, files,
                        connectivity and configurations
  -u USERNAME, --username=USERNAME
                        REST username for source cluster or server node
  -p PASSWORD, --password=PASSWORD
                        REST password for source cluster or server node
  -t THREADS, --threads=THREADS
                        Number of concurrent workers threads performing the
                        transfer
  -v, --verbose         verbose logging; more -v's provide more verbosity
  -x EXTRA, --extra=EXTRA
                        Provide extra, uncommon config parameters;
                        comma-separated key=val(,key=val)* pairs

<Advanced options truncated>
```

Remember the gamesim-sample bucket? This is one of the sample buckets that comes with Couchbase. It currently exists on the 6-node NYC cluster, but we never enabled it on the 2-node London cluster.

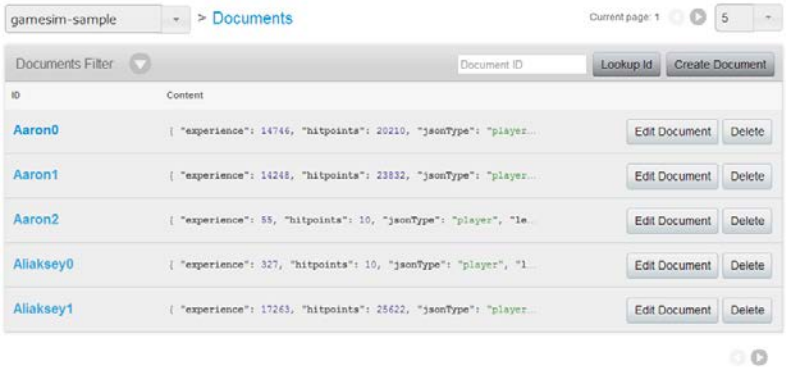
The next few screenshots show what the bucket looks like, its item count, its data size (37.5 MB), and some sample data from the bucket. By now you should know how to get to all of these screens in the Web UI of the NYC cluster, so we won't go through all those steps per say… however below you will find screenshots describing the gamesim-sample bucket. You can actually view these screens in your own lab environment if you'd like, or just review the screenshots below and continue on to the backup and restore commands after the screenshots.

Review the next 5 screenshots and then continue:

Observe in the 6-node NYC cluster: the item count, data size and replica count for the gamesim-sample bucket:

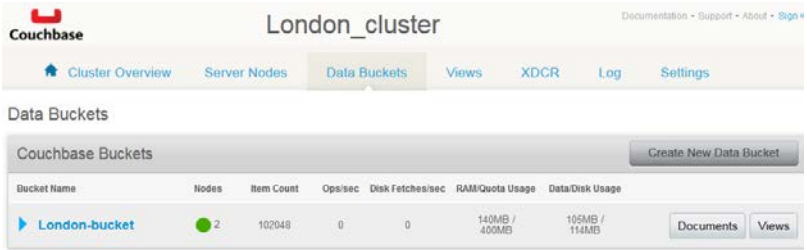


The items in this bucket are JSON documents:

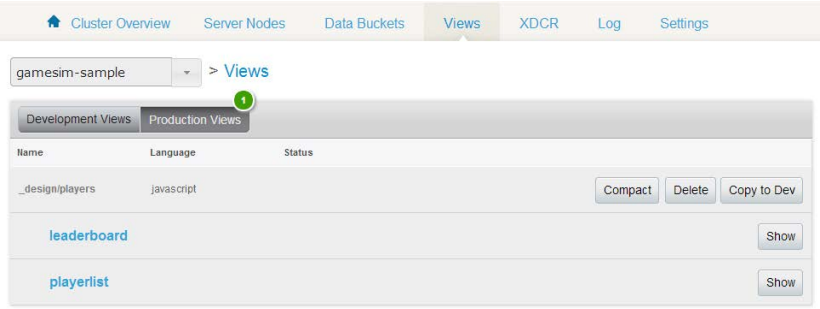


Here is a sample JSON document:

Also, notice that this bucket does NOT exist in the London cluster:



Finally, notice that the gamesim-sample bucket has 1 production Design Document with 2 Views (we are expecting this Design Doc to also be backed up):



Now we are ready to attempt backing up the gamesim-sample bucket from the 6-node NYC cluster. In the next section, we will restore the backup file to the 2-node London cluster into a differently named bucket.

First, on the App Server (black VM) create a directory that we can backup the gamesim data into:

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

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

Next run the cbbackup command to take the backup of the gamesim-sample bucket across all 4 nodes in the NYC cluster and store it in the directory created above. Provide the following command the public hostname of Node #1 (dark blue VM) in the 6-node NYC cluster.

```
[ec2-user@ip-172-31-23-211 ~]$ cbbackup http://ec2-54-85-43-x.compute-1.amazonaws.com:8091 /home/ec2-user/backup-gamesim -u Administrator -p couchbase -b gamesim-sample
  [####################] 100.0% (586/estimated 586 msgs)
bucket: gamesim-sample, msgs transferred...
       :                total |       last |    per sec
 byte  :                94693 |      94693 |    82573.2
done
```

Looks like the backup completed successfully. Change directories to the backup folder and descend into the directory structure to take a look at what's inside it:

```
[ec2-user@ip-172-31-23-211 ~]$ cd backup-gamesim/

[ec2-user@ip-172-31-23-211 backup-gamesim]$ ls –al
drwxrwxr-x. 3 ec2-user ec2-user 4096 Nov 13 17:55 .
drwx------. 4 ec2-user ec2-user 4096 Nov 13 17:55 ..
drwxrwxr-x. 3 ec2-user ec2-user 4096 Nov 13 17:55 2014-11-13T225542Z

[ec2-user@ip-172-31-23-211 ~]$ cd 2014-11-13T225542Z
```

```
[ec2-user@Chris_AppServer 2014-11-13T225542Z]$ ls
2014-11-13T225542Z-full
[ec2-user@ip-172-31-23-211 ~]$ cd 2014-11-13T225542Z-full
[ec2-user@ip-172-31-23-211 ~]$ ls

bucket-gamesim-sample
```

There a directory there. Go inside it:

```
[ec2-user@ip-172-31-23-211 backup-gamesim]$ cd bucket-gamesim-sample/

[ec2-user@ip-172-31-23-211 bucket-gamesim-sample]$ ls
design.json
node-ec2-54-85-206-x.compute-1.amazonaws.com%3A8091
node-ec2-54-85-43-x.compute-1.amazonaws.com%3A8091
node-ec2-54-86-106-x.compute-1.amazonaws.com%3A8091
node-ec2-54-86-243-x.compute-1.amazonaws.com%3A8091
```

So, there's a design JSON file and 4 directories that seem to correspond to the 4 nodes in my cluster. The directory that I highlighted above in yellow corresponds to Node #1 in the 6-node NYC cluster. Figure out which directory name's public hostname corresponds to Node #1 in your NYC cluster and CD into that directory.

```
[ec2-user@ip-172-31-23-211 bucket-gamesim-sample]$ cd node-ec2-54-85-43-x.compute-1.amazonaws.com%3A8091/
```

Check what is within this directory

```
[ec2-user@ip-172-31-23-211 node-ec2-54-85-43-128.compute-1.amazonaws.com%3A8091$ ls -lh
total 152K
drwxrwxr-x. 2 ec2-user ec2-user 4.0K Nov 13 17:55 .
drwxrwxr-x. 6 ec2-user ec2-user 4.0K Nov 13 17:55 ..
-rw-r--r--. 1 ec2-user ec2-user 114K Nov 13 17:55 data-0000.cbb
-rw-rw-r--. 1 ec2-user ec2-user  13K Nov 13 17:55 failover.json
-rw-rw-r--. 1 ec2-user ec2-user   12 Nov 13 17:55 meta.json
-rw-rw-r--. 1 ec2-user ec2-user  539 Nov 13 17:55 seqno.json
-rw-rw-r--. 1 ec2-user ec2-user  819 Nov 13 17:55 snapshot_markers.json
```

Hmm, notice that this file is only 114 kilobytes. Does that make sense?

There should be four such files from the backup, one from each node. If you dig through the rest of the folders, you will see that each of these files is around 120 – 200 KB in size. But if you remember from earlier, the data size for this bucket as reported in the Web UI was about 37 MB. What's happening here is a bit of an edge case scenario. When the bucket's data is very small (less than 1 MB) then the UI may not report the 'real' data size correctly. This issue will not occur if the bucket's data is sufficiently large, like over 128 MB. Ignore this nuance for now and continue.

Now that the gamesim-sample bucket is backed on to the App Server, how do we restore it to the 2-node London cluster?

If cbbackup was used to backup the bucket data, you can restore back to a cluster with the same or different number of nodes. The cbrestore command takes the information that has been backed up via the cbbackup command and streams the stored data into a cluster. The configuration of the cluster does not have to match the cluster configuration when the data was backed up, allowing it to be used when transferring information to a new cluster or updated or expanded version of the existing cluster in the event of disaster recovery.

Because the data can be restored flexibly, it allows for a number of different scenarios to be executed on the data that has been backed up:

- You want to restore data into a cluster of a different size and configuration.
- You want to transfer/restore data into a different bucket on the same or different cluster.
- You want to restore a selected portion of the data into a new or different cluster, or the same cluster but a different bucket.

The basic format of the cbrestore command is as follows:

```
cbrestore [options] [source] [destination]
```

Let's use cbrestore to restore the backed up gamesim bucket to the 2-node London cluster.

Run the following command from the App Server and in the command be sure to provide the public hostname of Node #1 (Red VM) in the 2-node London cluster:

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

[ec2-user@ip-172-31-23-211 ~]$ cbrestore /home/ec2-user/backup-gamesim
http://ec2-54-86-94-x.compute-1.amazonaws.com:8091 -u Administrator -p
couchbase2 --bucket-source=gamesim-sample --bucket-
destination=gamesim-restored

error: missing bucket-destination: gamesim-restored at destination: http://ec2-54-86-94-
157.compute-1.amazonaws.com:8091; perhaps your username/password is missing or incorrect
```

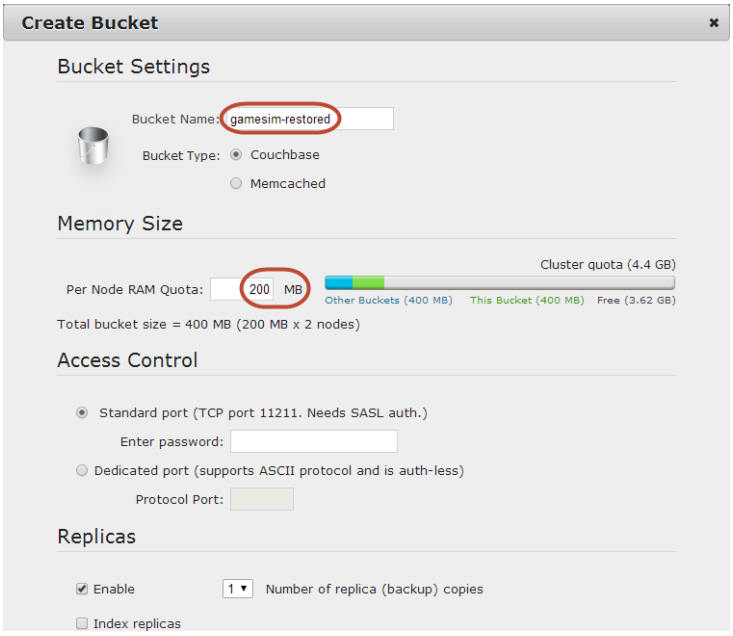
We got an error!

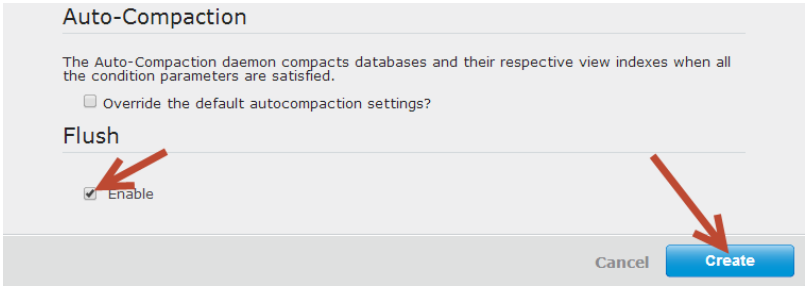
Looks like we need to first create the 'gamesim-restored' bucket in the 2-node London cluster.

Switch to the web UI for the 2-node London cluster and click on "Data Buckets" at the top and then click "Create New Data Bucket":

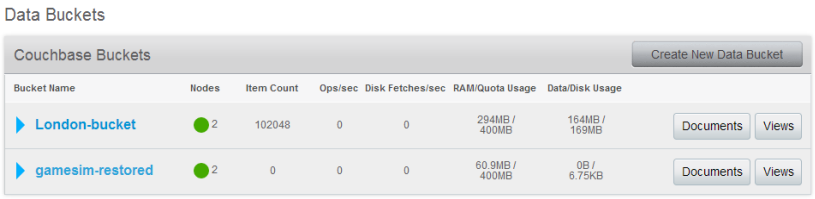
Use the following settings on the Create Bucket screen (only parameters in red should need to be altered):

Bucket Name: gamesim-restored
Bucket Type: Couchbase
Per Node RAM Quota: 200 MB
Access Control: Standard port (no password)
Replicas: Place a check next to Enable and set the # to 1
Auto-Compaction: DO NOT place a check to override the default autocompaction settings
Flush: Enabled
Click Create





You should now see the 'gamesim-restored' data bucket:

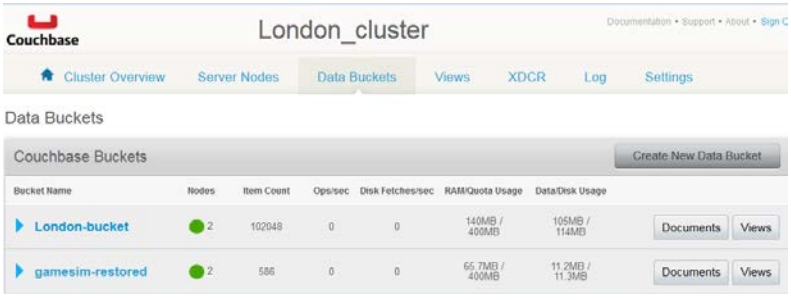


Switch back to the App Server (black VM). Again, run the following command and in the command be sure to provide the public hostname of Node #1 (Red VM) in the 2-node London cluster:

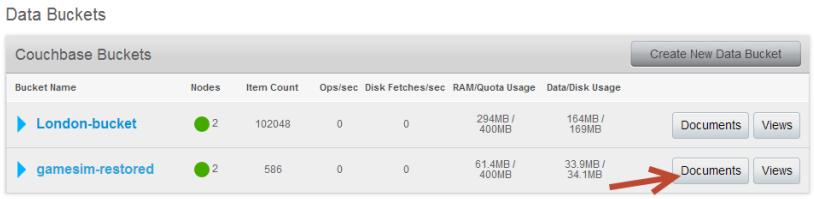
```
[ec2-user@ip-172-31-23-211 ~]$ cbrestore /home/ec2-user/backup-gamesim
http://ec2-54-86-94-x.compute-1.amazonaws.com:8091 -u Administrator -p
couchbase2 --bucket-source=gamesim-sample --bucket-
destination=gamesim-restored
  [####################] 100.0% (586/estimated 586 msgs)
bucket: gamesim-sample, msgs transferred...
       :                total |       last |    per sec
 byte  :                94693 |      94693 |   763332.8
done
```

The 586 items should now be restored at the London side.

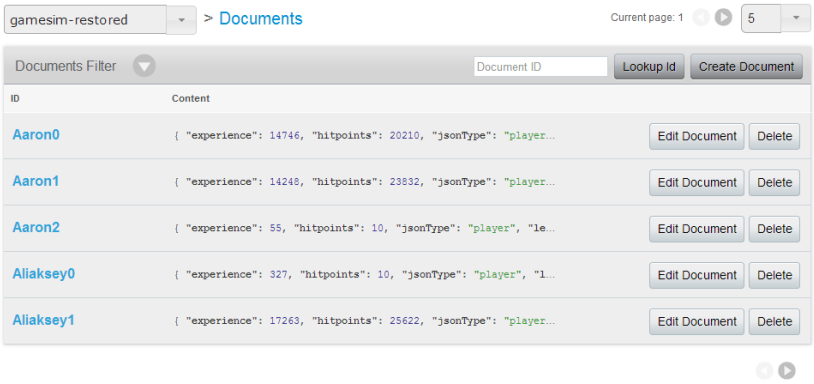
Switch to the 2-node London cluster's Web UI and click on Data Buckets at the top (you may have to refresh this page):



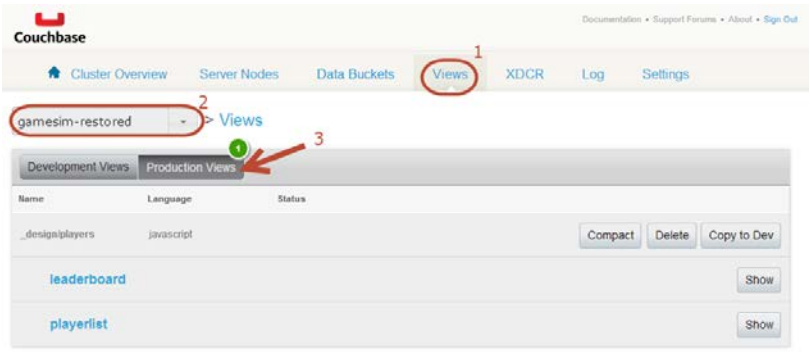
You should see 586 items in the restored bucket. Click Documents for the gamesim-restored bucket:



You should see the first 5 JSON items that you previously saw at the source/NYC side.



What about that Design Document with the 2 views? Let's check if that also got restored. Click on Views at the top, then from the drop down choose "gamesim-restored" and finally click on "Production Views":



Here you should see the same Design Document along with 2 views.

Note that it is also possible to filter keys during the backup process. This can be useful if you want to specifically backup a portion of your dataset, or you want to move part of your dataset to a different bucket. The filter specification is in the form of a regular expression, and is performed on the client-side within the cbbackup tool. This means that the entire bucket contents must be accessed by the cbbackup command and then discarded if the regular expression does not match. For example, using the filter, it is possible to backup just the 1,024 keys starting with "AB" from the NYC-bucket.

It is also possible to backup the entire NYC-bucket and then during the restore process, filter out only the keys starting with "AB".

For more details on how to use cbbackup and cbrestore, check out this section of the Couchbase Admin Guide:
http://docs.couchbase.com/admin/admin/CLI/cbbackup_tool.html

http://docs.couchbase.com/admin/admin/CLI/cbrestore_tool.html

This concludes Lab #7.