CS300 Couchbase NoSQL Server Administration

Lab 6 Exercise Manual

Release: 6.5.1

Revised: June 22nd, 2020

Lab #6: XDCR

Objective: This 2 hour lab will walk you through the fundamentals of XDCR and teach you the core concepts behind Couchbase Cross Data Center Replication. This is a long lab, so please perform each step carefully as missing steps may cause unpredictable behavior!

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

- Connect to and create a 2-node remote Couchbase cluster. We will call this the "London" or Production/Source cluster. (The existing 6-node cluster will be called the "NYC" or Remote/Destination cluster.)
- Configure unencrypted, unidirectional replication between the 6-node and 2-node cluster for the beer-sample bucket.
- Use cbworkloadgen to push 250,000 keys from NYC (6-node) -> London (2-node)
- Learn that Design Docs and Views do not get replicated by default
- Learn about XDCR conflict resolution and learn how to get the revision count of a specific item from the data file
- See what a tombstone looks like in the data file and learn about tombstones
- Learn differences between version 1 (REST) and version 2 (memached REST) of the XDCR protocol
- Learn about advanced XDCR settings
- Viewing outbound and inbound replication statistics using the Web UI
- Viewing internal XDCR settings via the REST API
- Configure an encrypted, bidirectional replication between 2 clusters
- Learn about the optimistic threshold setting for replication streams
- Learn how to delete replication streams

Create a 2-node cluster on the Remote side:

Before we configure Cross Data Center Replication (XDCR) we need to first set up a 2nd cluster in a remote data center. If you check the updated `Cluster-IPs` spreadsheet, the instructor will have provided you with 2 additional Virtual Machines in the Amazon cloud. Assume that these 2 new hosts are in a different datacenter than the existing 6-node cluster and we want to set up different types of replication streams for different buckets.

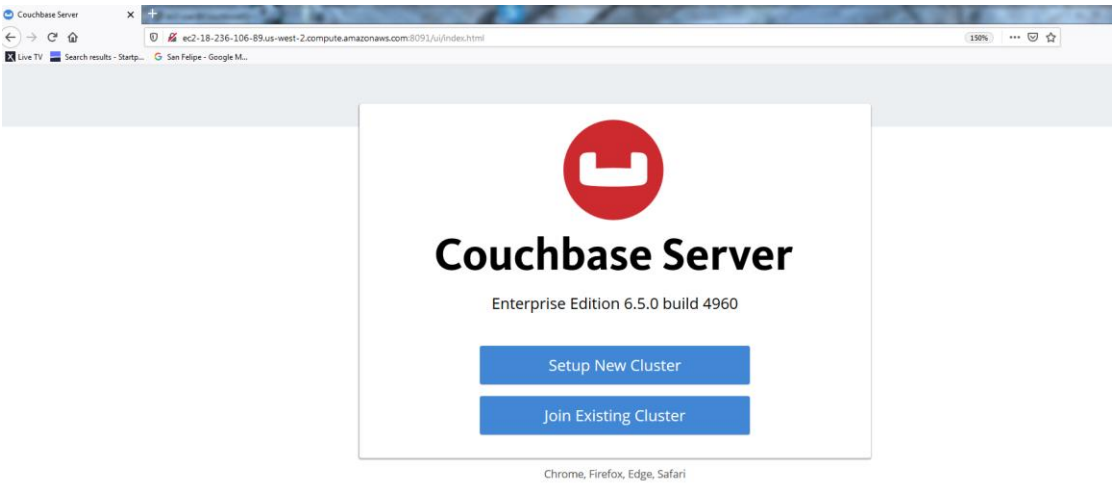
Your nodes for this 2 node cluster should already be set-up and couchbase installed from lab 5.1

Cluster #2
Couchbase07: Red
Couchbase08: Orange

Finally, let's configure a 2-node cluster on the remote side.

Open a new browser tab and connect to the Couchbase07 node (Red)'s setup wizard. Open a Chrome or Firefox browser and go to the following URL:

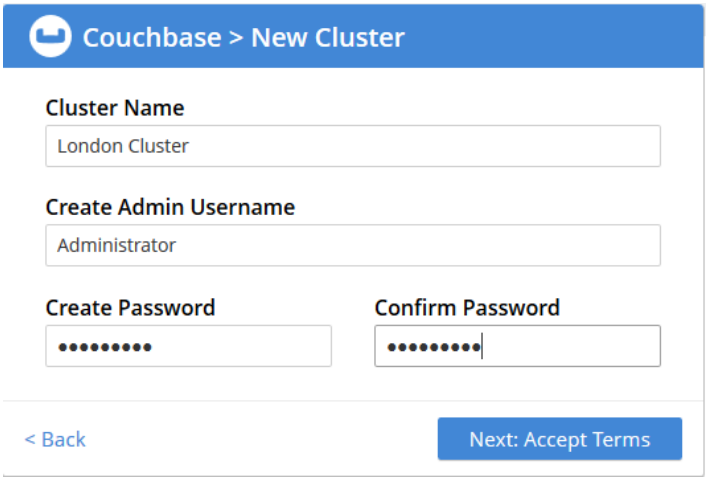
http://<public hostname of Couchbase07 VM>:8091



Click on the Setup New Cluster button in the bottom area to continue.
Edit in Cluster Name **London Cluster**
Password for Administrator **couchbase**

Couchbase

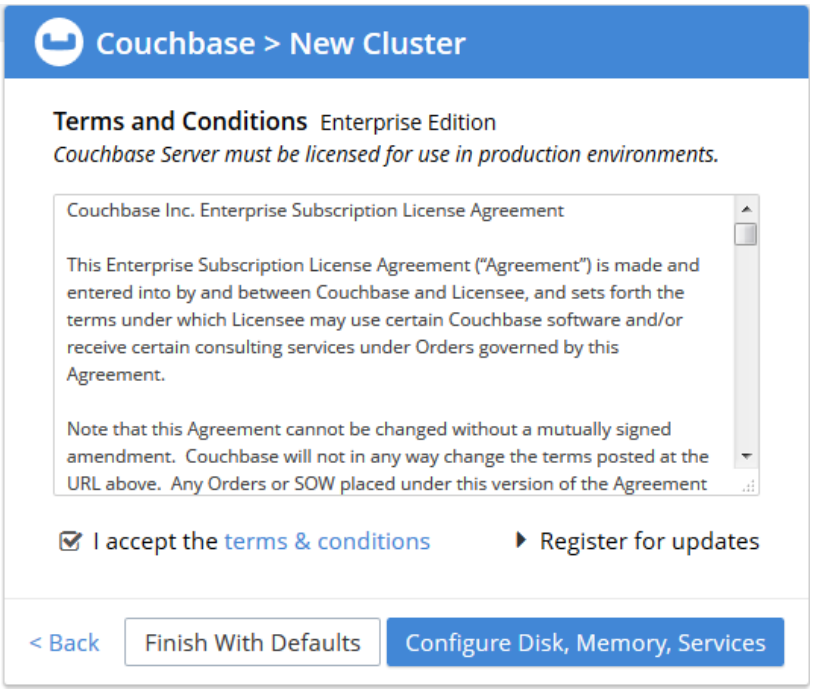
Lab-6: XDCR page 4



Do Not Change the Administrator name or password from these instructions!

Click Next: Accept Terms

Check the accept terms & conditions box



Click Configure Disk, Memory, Services

Couchbase

Some of the settings on the "Couchbase>New Cluster>Configure " page will need to be altered. Specifically, the items in red need to be changed

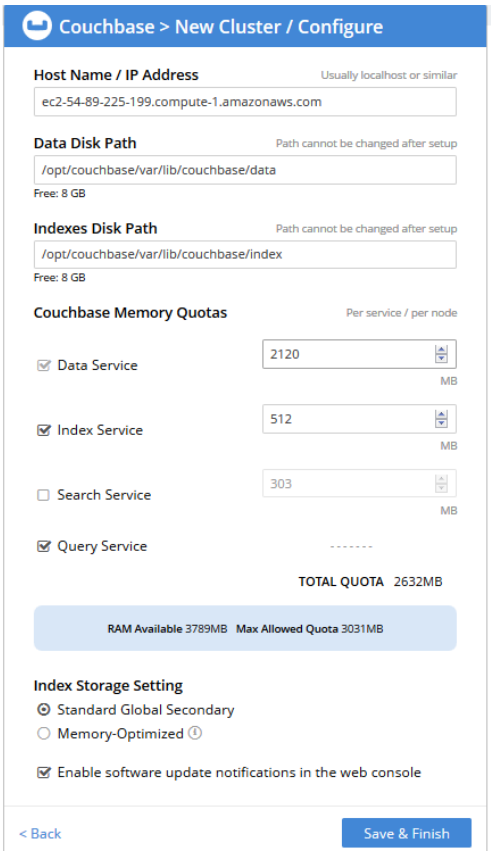
Hostname: <Public hostname of VM, retrieve this from the Cluster-IPs spreadsheet>
Or from the browser URL entry line. i.e. <**ec2-54-174-65-105.compute-1.amazonaws.com**>

Data Disk Path: /opt/couchbase/var/lib/couchbase/data
Indices Path: /opt/couchbase/var/lib/couchbase/index

Services select: Data, Index & Query

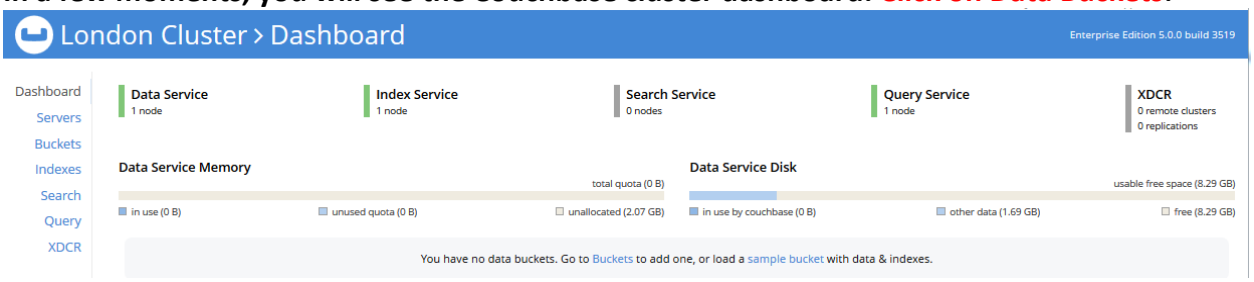
Per Server DATA RAM Quota: 2120 MB *Accept the default calculation if your memory value is different than shown*

Per Server Index RAM Quota: 512 MB *(Min ram value shown to right)*
Accept the default calculation if your memory value is different than shown or you are configuring a dedicated INDEX service with more RAM available.



Click on Save and Finish to continue:

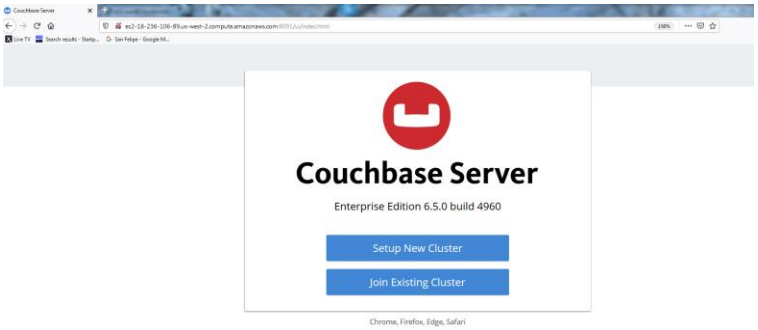
In a few moments, you will see the Couchbase cluster dashboard. Click on Data Buckets:



You should now see no buckets configured in the cluster:
Let's continue with adding the 2nd node into this remote cluster.

Next point your browser at Couchbase08 to obtain a setup splash screen

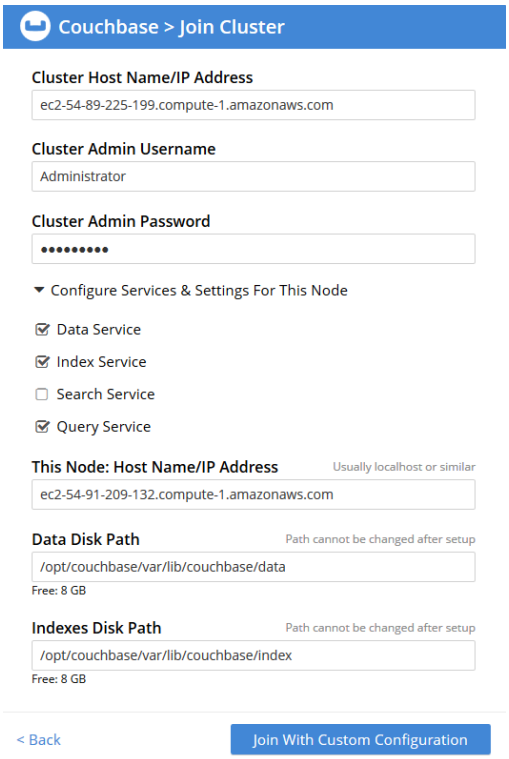
http://ec2-52-53-173-24.us-west-1.compute.amazonaws.com:8091



Choose Join Existing Cluster

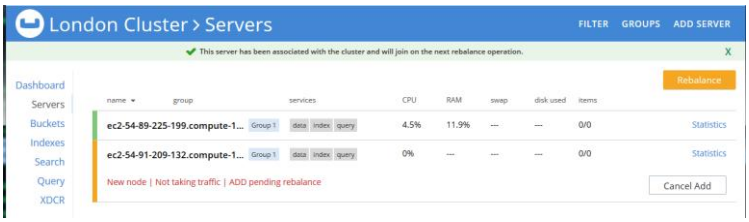
Open "Configure Services & Settings For This Node"
Select Data, index, and Query. Set pathing for data and index.
Join.

Couchbase

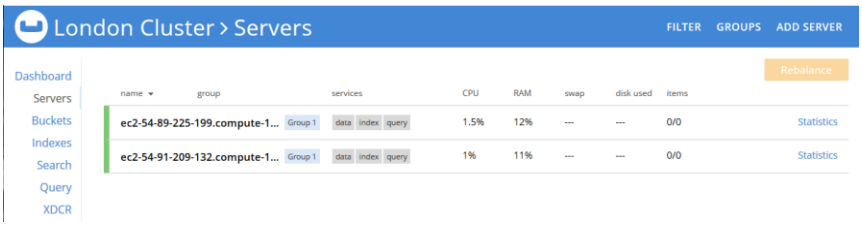


Fill in the information required making sure to use the Amazon ec2 address for both cluster and joining node name.

You should now be returned to the Web UI page for Server Nodes and see a Pending Rebalance. Click on Rebalance:



Within a few seconds the Rebalance operation should complete and you will have a healthy 2-node cluster in the remote datacenter!



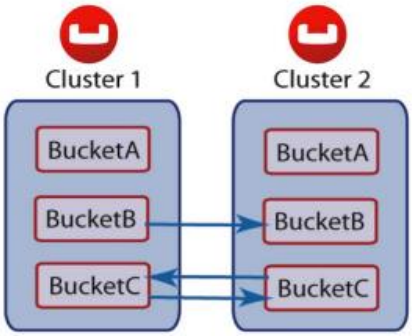
Configure unidirectional replication for the beer-sample bucket:

In the remainder of this lab, we will configure XDCR using different techniques and learn about some important replication settings.

First, let's configure our first replication to replicate the beer-sample bucket from the production 6-node cluster to the remote 2-node cluster.

There are 2 replication topologies: unidirectional and bidirectional.

The diagram below shows BucketB -> BucketB configured with unidirectional replication. Also notice that BucketC is configured with bidirectional replication.



Unidirectional Replication is one-way replication, where active data gets replicated from the source cluster to the destination cluster. You may use unidirectional replication when you want to create an active offsite backup, replicating data from one cluster to a backup cluster.

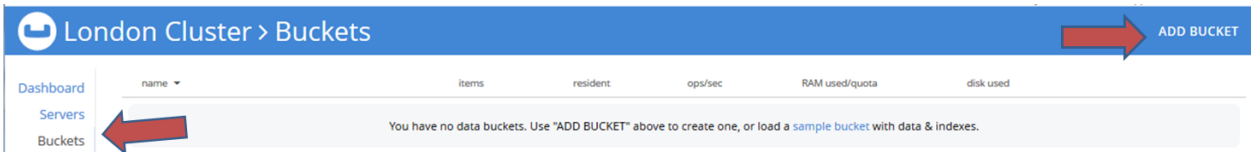
Bidirectional Replication allows two clusters to replicate data with each other. Setting up bidirectional replication in Couchbase Server involves setting up two unidirectional replication links from one cluster to the other. This is useful when you want to load balance your workload across two clusters where each cluster bidirectionally replicates data to the other cluster.

XDCR can be setup on a per bucket basis. A bucket is a logical container for documents in Couchbase Server. Depending on your application requirements, you might want to replicate only a subset of the data in Couchbase Server between two clusters. With XDCR you can selectively pick which buckets to replicate between two clusters in a unidirectional or bidirectional fashion.

We will configure unidirectional replication for the beer-sample bucket.

Before setting up replication, we need to create an empty beer-sample bucket in the remote data center's 2-node cluster. Replication will not automatically create the destination bucket for you.

You should already be logged into the remote 2-node cluster (using Couchbase07's public hostname). There should be no buckets configured. Click on Buckets at the side, then ADD BUCKET:

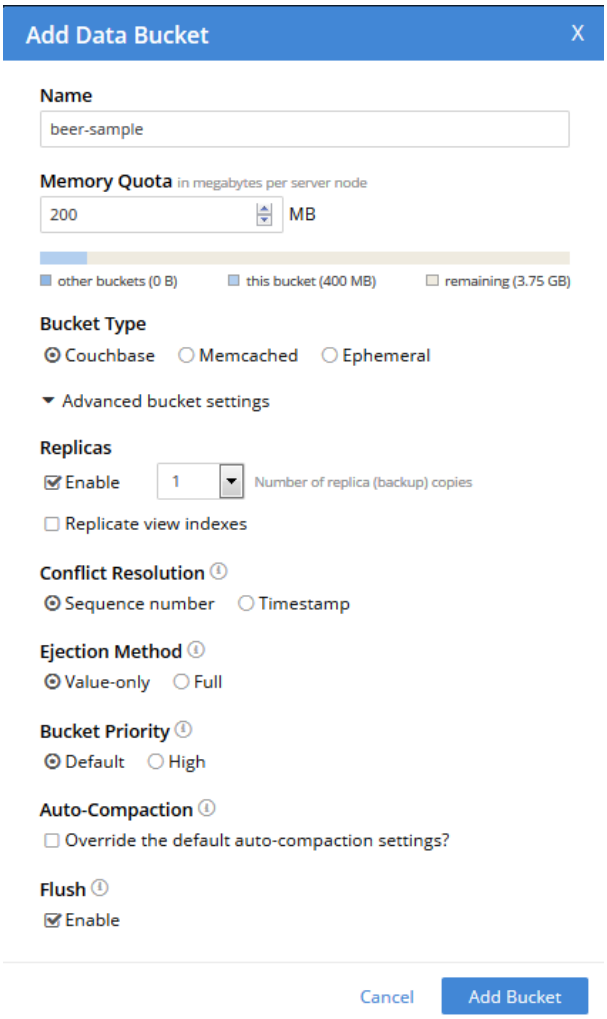


On the Create Bucket popup, enter the following settings (only items in red need to be changed):

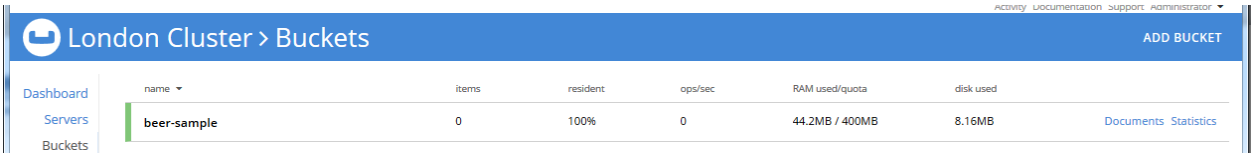
Bucket Name: beer-sample
Bucket Type: Couchbase
Per Node Ram Quota: 200 MB
Ejection Method: Value-only
Replicas: Enable and set to 1
Replicate View Index: Unchecked
Bucket Priority: Default
Auto-Compaction: leave unchecked
Flush: Place a check to set it as Enabled

Click Add Bucket

Couchbase

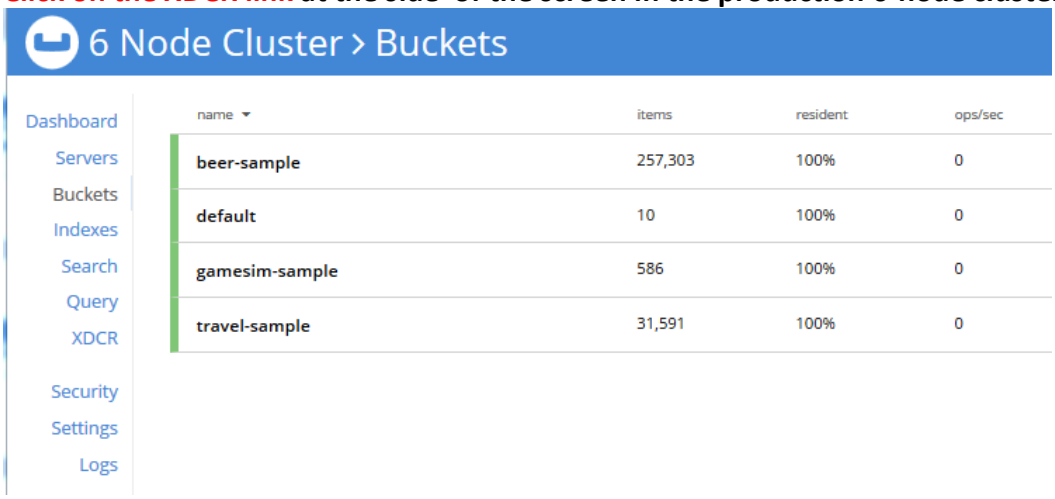


You should now see the new, empty beer-sample bucket.
Note that the Item count below is 0:



Now, switch to the Web UI for the production 6-node cluster. At this point, it will be useful to open 2 tabs in your browser window: one for the 6-node cluster and one for the 2-node cluster.
For both clusters, connect to their Node #1's(Couchbase01 & Couchbase07) public hostname for each cluster(New York & London).

Click on the XDCR link at the side of the screen in the production 6-node cluster:

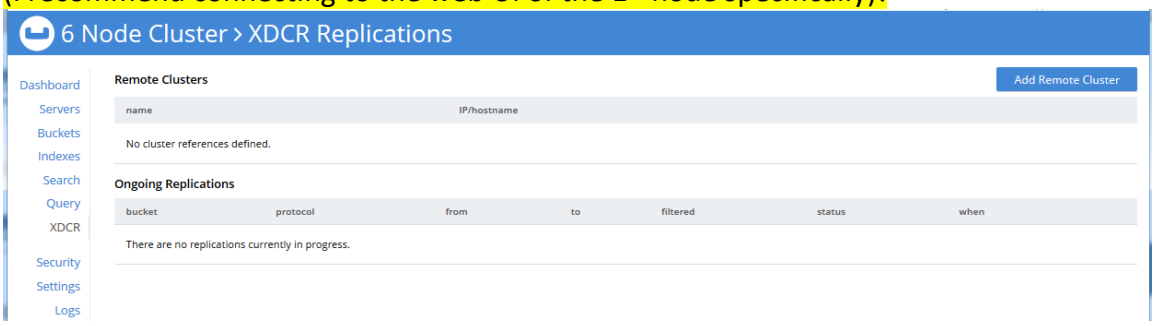


We will use the 6-node production cluster as the source cluster and the remote 2-node cluster(London) as the destination.

To set up a destination cluster reference, click the Add Remote Cluster Reference button:

Add Remote Cluster

Warning: you should be completing the next steps from the 6-node production cluster's Web UI (I recommend connecting to the web UI of the 1st node specifically)!

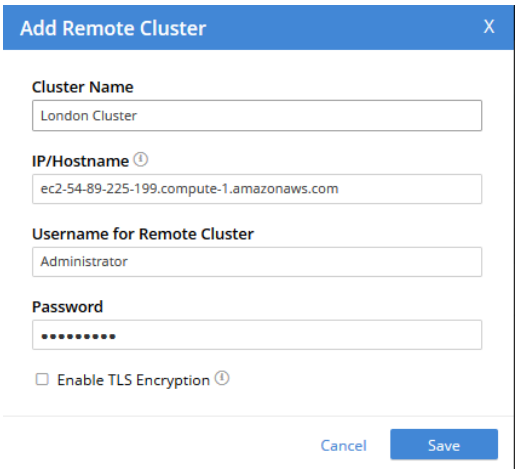


On the pop-up enter:

Cluster Name: London Cluster
IP/hostname: <public hostname of the Node #1 (red) in the London cluster>
Username: Administrator
Password: couchbase
Enable Encryption: unchecked

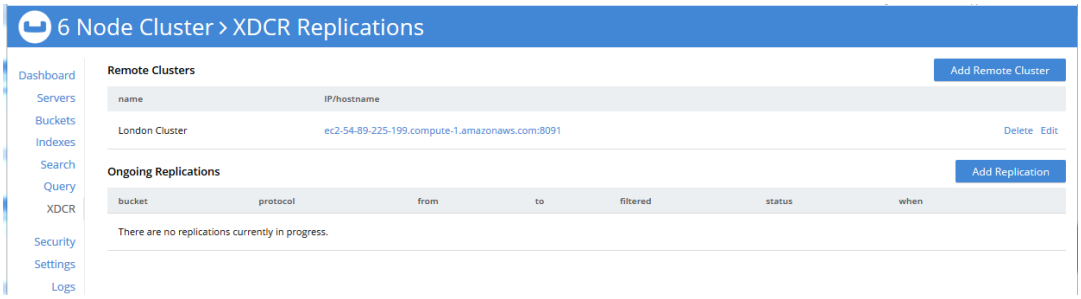
Click Save

Couchbase



The Web UI will now show a remote cluster configured as London Cluster.

Click on Add Replication **to set up the unidirectional replication for the beer-sample bucket:**



On the Add Replication pop-up, edit the following three settings:

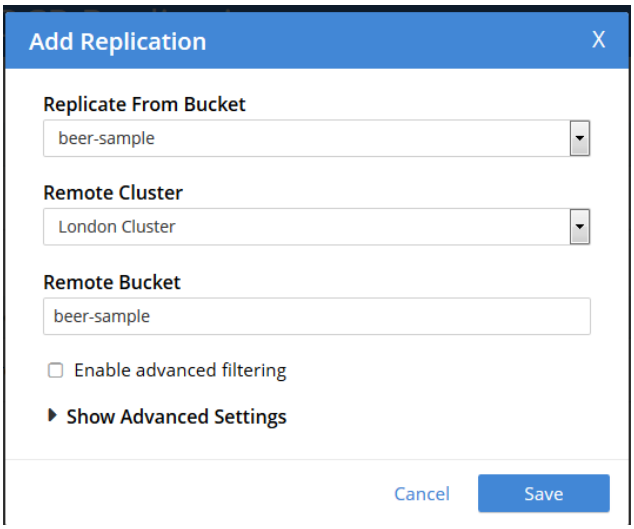
Replicate From Bucket:

Bucket: beer-sample

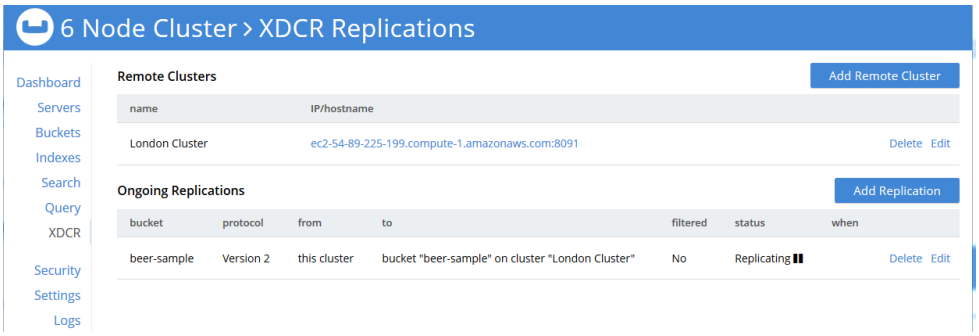
Remote Cluster:London Cluster

Remote Bucket: beer-sample

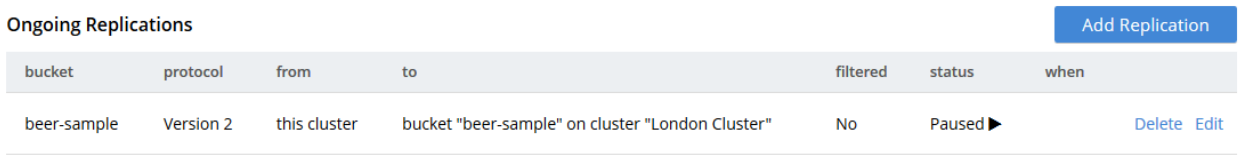
Click Save



The Web UI will now update to show that there is an on-going replication occurring using Protocol version 2 for beer-sample from this cluster to a remote cluster London. You should see the <u>Status</u> as "Replicating"
Notice the pause button next to Replicating



Replication can be paused as shown in this screen capture.

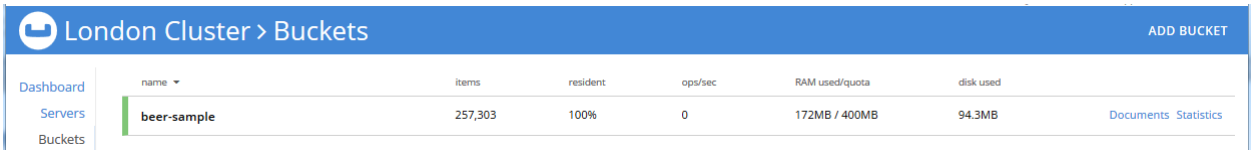


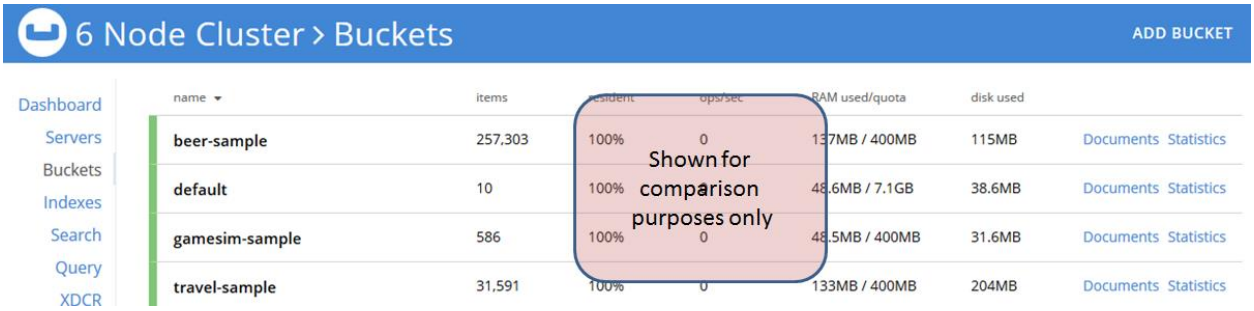
Note that the "When" column setting is currently blank for When with Couchbase 6.0.2 So there's no way to change this currently. In a future version of Couchbase there may be different options to choose from.

Let's check if the replication as really occurred. Switch to the 2nd tab in your browser, which should be for Node #1 (red) in the remote 2-node cluster(London).
Click Refresh in the browser and under the Data Buckets screen, you should see the same number of items under beer-sample as in 6 node cluster.

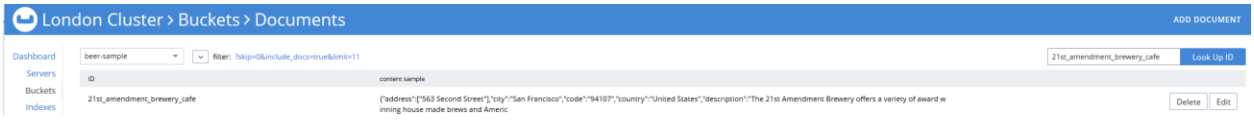
(Note that if you verify this same # in the production 6-node cluster, you will also see the same value. Screen caps show 257,303,yours may vary.

Click on the Documents button next to the beer-sample bucket:



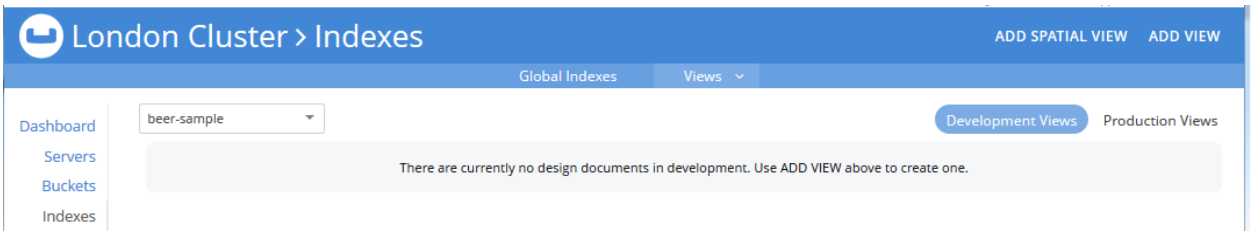


Under Documents, you may see a lot of random data, not pertaining to beer-sample JSON files!
This is because in the previous lab, we used cbworkloadgen to push 250,000 writes into this bucket to observe the indexing of the views.
You can verify that the original JSON data that came with the beer-sample bucket is still in this bucket by searching for a specific brewery. In the search box for this bucket, enter the document ID '`21st_amendment_brewery_cafe`' (without the single quote marks) and click on Lookup id:

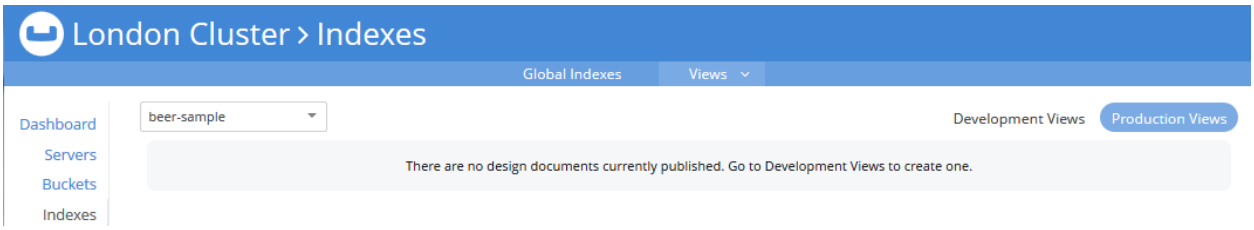


The specific brewery that you requested will now be displayed:

Remain within the Web UI of the remote cluster(London) and click on indexes and then click Views for beer-sample:

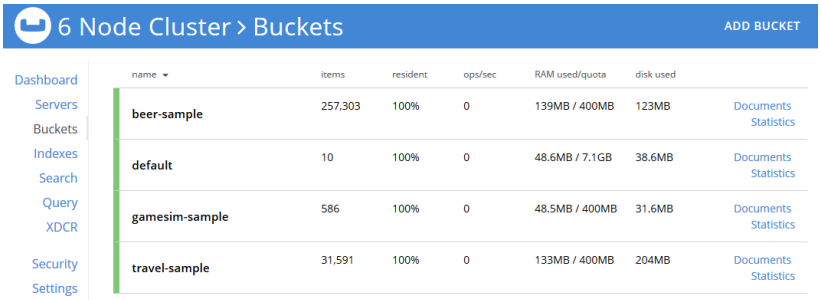


Notice that there are no Development Views on the remote cluster for beer-sample.
Click on Production Views:



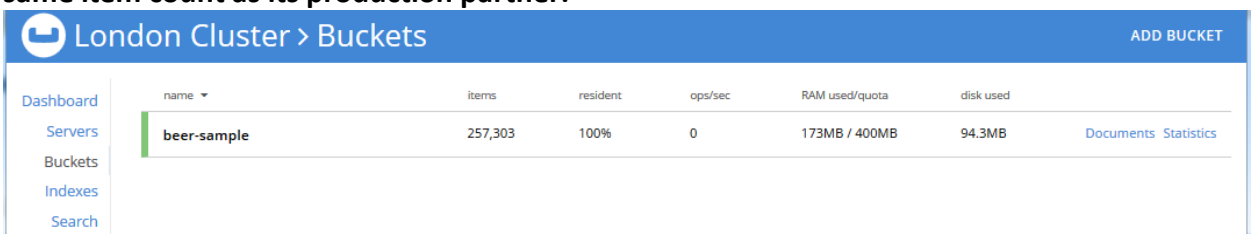
Once again, notice that there are no Production Views for beer-sample on the remote-cluster:

At this point in the lab you should have 3-4 buckets(some may have travel-sample) on the production 6-node cluster side with the item counts displayed below:



name	items	resident	ops/sec	RAM used/quota	disk used	
beer-sample	257,303	100%	0	139MB / 400MB	123MB	Documents Statistics
default	10	100%	0	48.6MB / 7.1GB	38.6MB	Documents Statistics
gamesim-sample	586	100%	0	48.5MB / 400MB	31.6MB	Documents Statistics
travel-sample	31,591	100%	0	133MB / 400MB	204MB	Documents Statistics

And the remote 2-node cluster should have just 1 bucket (which we're replicating to) with the same item count as its production partner:

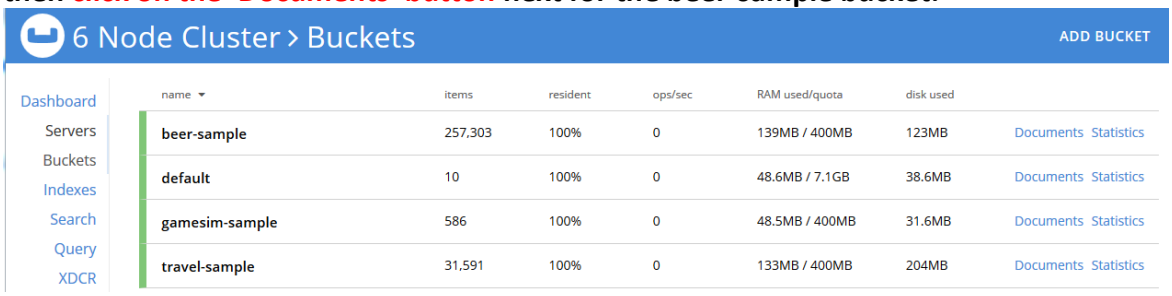


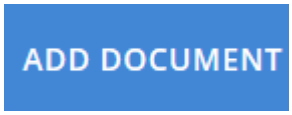
Understanding which keys will be replicated with XDCR and exploring the revision count parameter:

Some of the algorithms that XDCR uses could be non-intuitive. So, in this section we will create items on both the source and destination cluster and check to see if it gets replicated. We will also learn about how the revision count is used to decide which items to push from source -> destination.

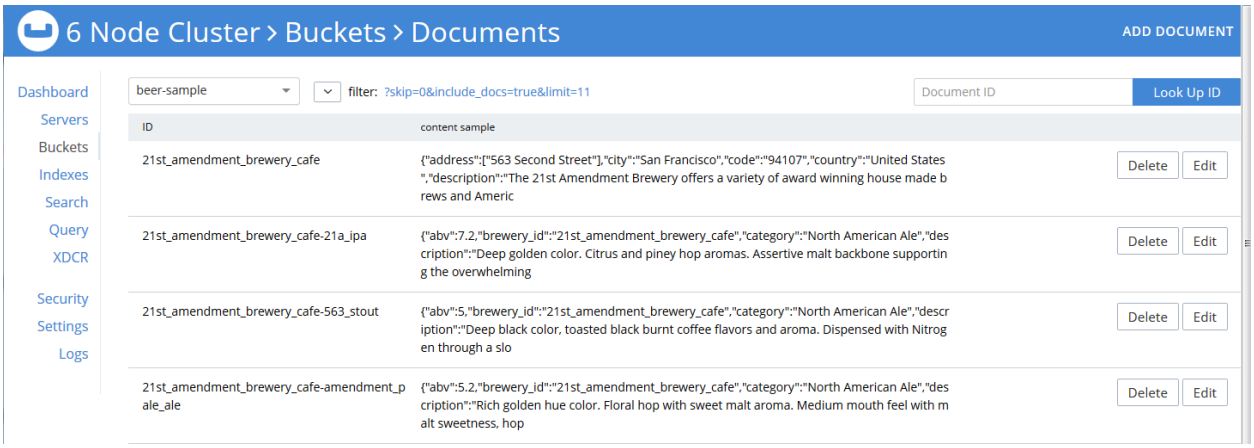
First, let's create a new key/document on the source side under beer-sample and then check to see if this fresh item gets replicated to the destination cluster.

On the source (6-node production cluster's browser tab), click on Data Buckets at the side and then click on the 'Documents' button next for the beer-sample bucket:



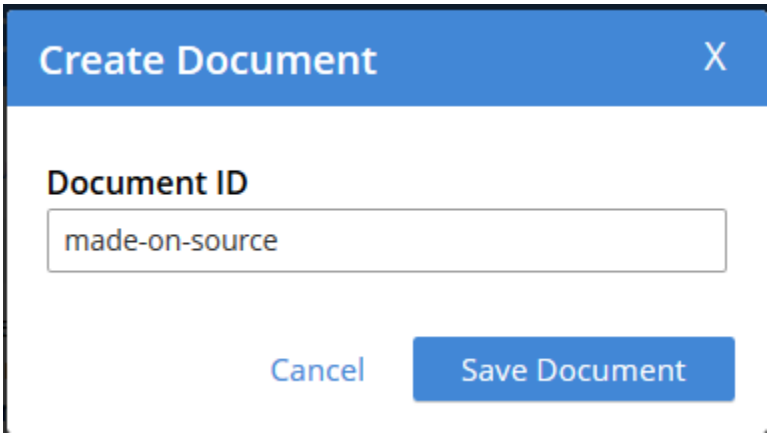
Click 

Couchbase

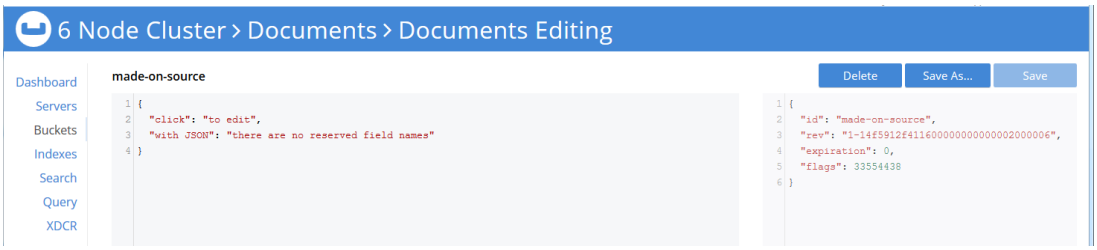


In the pop-up, name the document 'made-on-source'

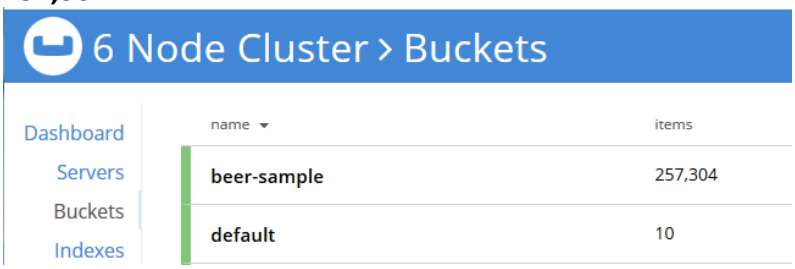
and click Save Document



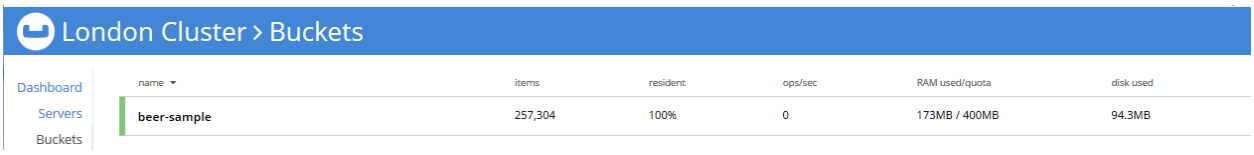
The new document will be created in the beer-sample bucket. Do not edit and resave this document! Just leave the JSON the way it is in its default state and click on Data Buckets at the side of the screen:



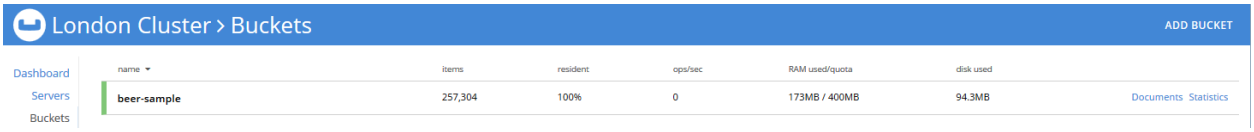
You should see the item count field for the beer-sample bucket as increased by one to 257,304:



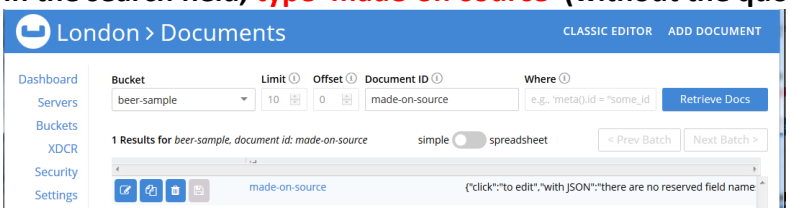
Next, switch to the browser tab for the remote/destination 2-node cluster(London) and refresh the page(on the Data Buckets tab):



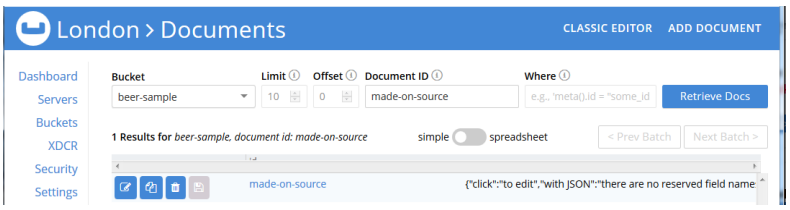
You will see the item count will refresh on the remote side to 257,304 items, matching the source side. Click on Documents so we can attempt reading the new document:



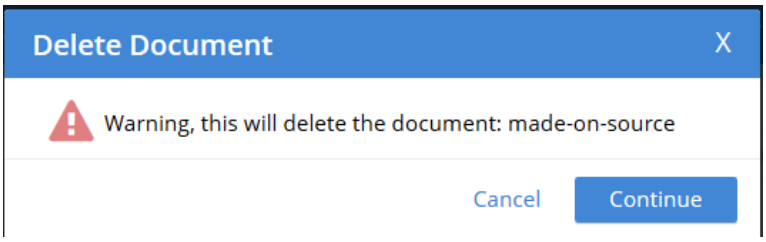
In the search field, type 'made-on-source' (without the quotes) and click "Lookup id":



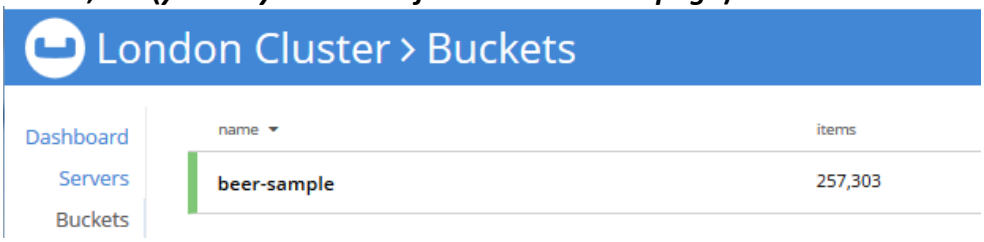
You should now see the document you created on the source side. Delete this document (destination side/2 node cluster)by clicking  **and then let's see if it reappears on the destination side:**



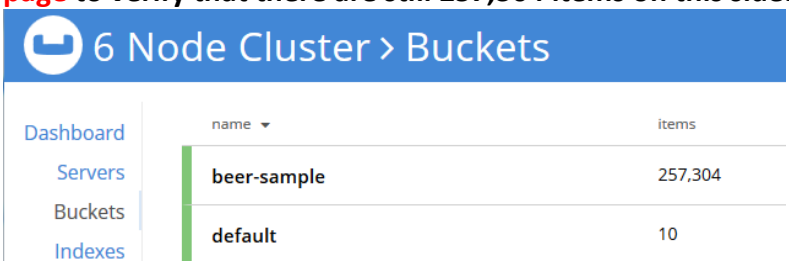
Click Continue **on the pop-up to confirm the deletion:**



Finally, click on Data Buckets at the top and verify that the item count as been reduced by 1 to 257,303 (*you may have to refresh the browser page*):



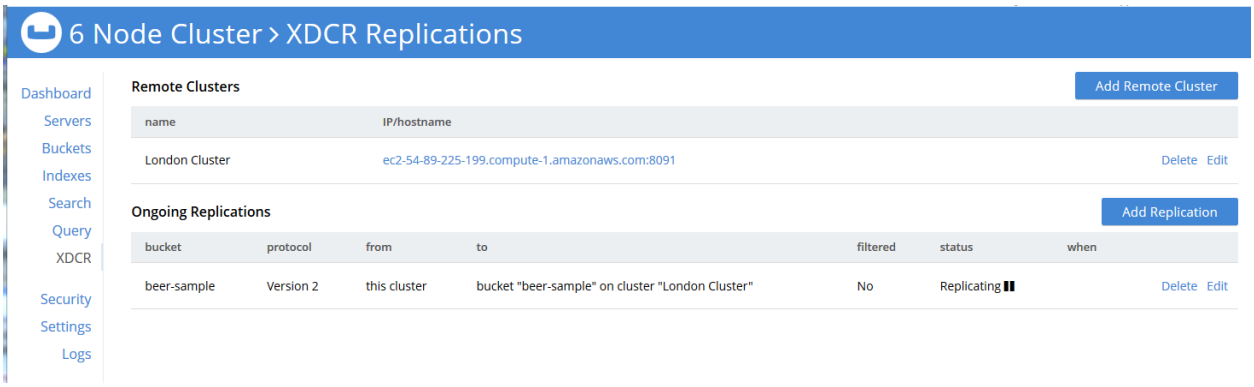
Switch browser tabs to the production/source 6-node cluster and refresh the "Data Buckets" page to verify that there are still 257,304 items on this side:



So at this point, even if you wait 5 minutes the item that we deleted on the destination cluster will NOT get automatically replicated back to the destination. Why is this?

First, let's go and look at the advanced settings in the Web UI for the ongoing replication to see if there's something we can change to fix this issue.

On the source/production 6-node cluster, click on XDCR at the side of the page and then click Edit **for the beer-sample <u>Ongoing replications</u>:**



Notice above that this replication is using Version 2 of the protocol.

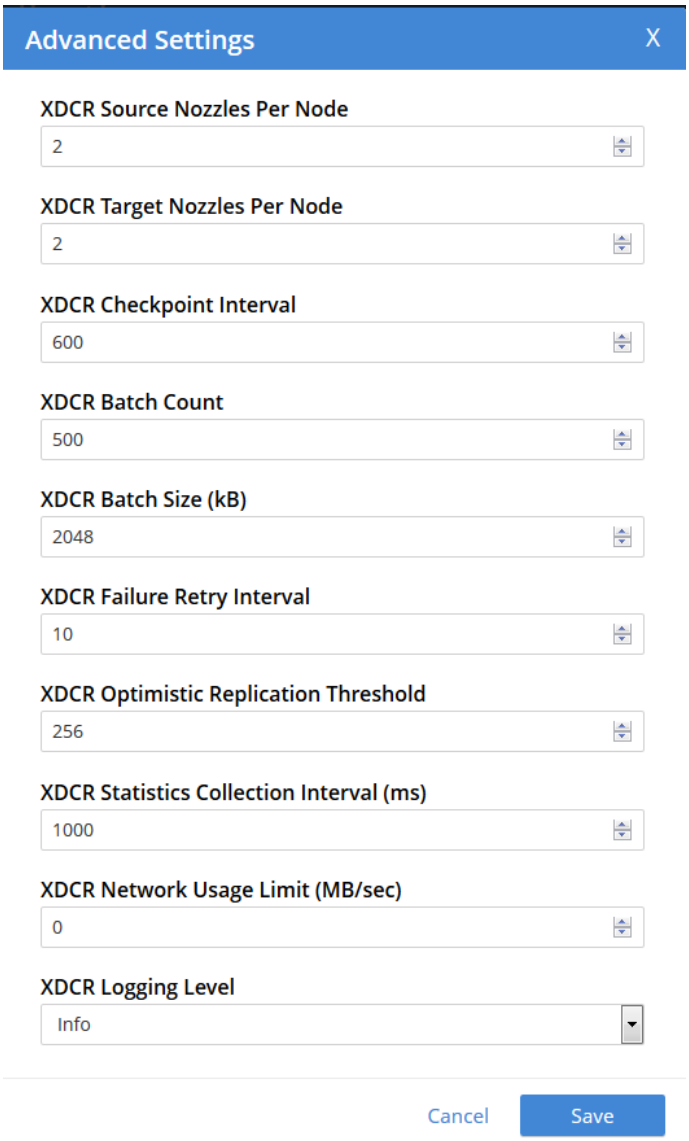
The XDCR protocol defaults to version 2.

- Version 1 uses the REST protocol for replication. This increases XDCR throughput at destination clusters. If you use the Elasticsearch plug-in, which depends on XDCR.

- Version 2 uses memcached REST protocol for replication. It is a high-performance mode that directly uses the memcached protocol on destination nodes. Choose version 2 when setting up a new replication with Couchbase Server 2.2 or later.

You can change this setting via the REST API for XDCR internal settings or the couchbase-cli tool.

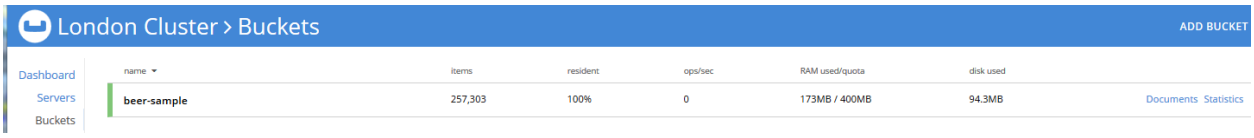
The Advanced Settings pop-up will be displayed. Here is a quick explanation of these settings. After reading through the explanations below, click "Cancel":



XDCR Source and Target Nozzles per node.

XDCR Checkpoint Interval:
Interval between checkpoints, 60 to 14400 (seconds). Default 1800. At this time interval, batches of data via XDCR replication will be placed in the front of the disk persistence queue. Changing this to a smaller value could impact cluster operations when you have significant amount of write operations on a destination cluster and you are performing bidirectional replication with XDCR. For instance, if you set this to 5 minutes, the incoming batches of data via XDCR replication will take priority in the disk write queue over incoming write workload for a destination cluster. This may result in the problem of having an ever growing disk-write queue on a destination cluster; also items in the disk-write queue that are higher priority than the XDCR items will grow staler/older before they are persisted.

XDCR Batch Count:
Document batching count, 500 to 10000. Default 500. In general, increasing this value by 2 or 3 times will improve XDCR transmissions rates, since larger batches of data will be sent in the same timed interval. For unidirectional replication from a source to a destination cluster, adjusting this setting by 2 or 3 times will improve overall replication performance as long as persistence to disk is fast enough on the destination cluster.
XDCR Batch Size (KB):
Document batching size, 10 to 100000 (KB). Default 2048. In general, increasing this value by 2 or 3 times will improve XDCR transmissions rates, since larger batches of data will be sent in the same timed interval.
XDCR Failure Retry Interval:
Interval for restarting failed XDCR, 1 to 300 (seconds). Default 30. If you expect more frequent network or server failures, you may want to set this to a lower value. This is the time that XDCR waits before it attempts to restart replication after a server or network failure.
XDCR Optimistic Replication Threshold:
This will improve latency for XDCR.
This is document size in bytes. 0 to 2097152 Bytes (20MB). Default is 256 Bytes. XDCR will get metadata for documents larger than this size on a single time before replicating the document to a destination cluster.
- - - -
None of the above settings really helps explain why the document named with the key 'made-on-source' is not being automatically re-replicated from source -> destination since we deleted it on the destination.
It has probably been about 3 minutes since you deleted the item at <Destination>. **If you switch over to the remote/destination cluster's Web UI tab and refresh the 'Data Buckets' page, you will still not see this item:**

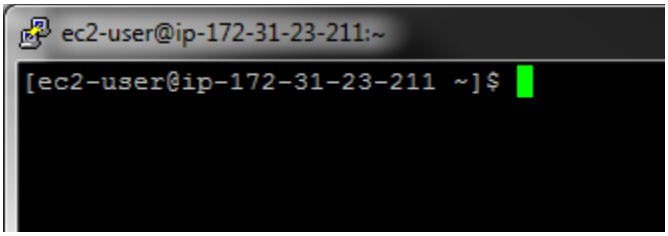


How do we explain this issue? Why is that document with the key 'made-on-source' not getting re-replicated?

To explain this, we need to do some investigating. Let's check what the revision # is for the item with the key 'made-on-source' in the source cluster and in the destination cluster. Our hunch should currently be that when we deleted the item in the destination cluster, we actually just updated the document with a tombstone as the value and incremented its revision count # to 2.
The revision count # for the original document created on the source side is probably still set to 1. So the source cluster will simply not push the item to the destination side b/c the revision # is higher on the destination.

But let's check this logic via an investigation.

Connect to the App Server (black VM):



Run the 'cbc cat' command to first simply read the item from the source cluster. Provide the command the public hostname of Node #1 (dark blue VM) in the 6-node production/source cluster:

```
[ec2-user@appserver ~]$ cbc-cat -U $NODE1/beer-sample made-on-source
made-on-source       CAS=0x158e55e996ed0000, Flags=0x2000006, Size=72,
Datatype=0x01(JSON)
{
"click": "to edit",
"with JSON": "there are no reserved field names"
}
```

Then run the 'cbc hash' command to find out where this key is actually stored in this 6-node cluster (again provide the command the public hostname of Node #1 on the production side):

```
[ec2-user@appserver ~]$ cbc-hash -U $NODE1/beer-sample made-on-source
made-on-source: [vBucket=788, Index=2] Server: ec2-13-57-200-84.us-
west-1.compute.amazonaws.com:11210, CouchAPI: http://ec2-13-57-200-
84.us-west-1.compute.amazonaws.com:8092/beer-sample
Replica #0: Index=0, Host=ec2-13-56-188-91.us-west-
1.compute.amazonaws.com:11210
```

You can see this in my specific setup, the active copy of the key is hashed to vBucket # 788 on the Server with publichostname "ec2-13-57-200-84.us-west-1.compute.amazonaws.com". In my setup, this translates to the 3rd Node (Green VM) in the 6-node cluster.

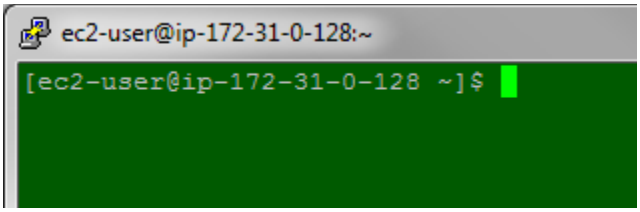
Write down the specific vBucket #, hostname and shell window color for your environment:

vBucket #: _____
Public Hostname: _____
Hostname shell color or nodename: _____

Since in my specific setup this key hashed to the 3rd/Green node, I will go to the green window/Couchbase03 for the next step.

Warning: Even though I'm switching to the Green node, you may have to go to the dark blue, light blue, green or yellow(Couchbase01,02,03,04) VM in your specific environment!

Switch to the node where the 'made-on-source' key's active replica hashed to:



Print a listing of all the data files for the beer-sample bucket on this node:

```
[ec2-user@couchbase03 ~]$ sudo ls -al
/opt/couchbase/var/lib/couchbase/data/beer-sample
total 43744
drwxrwx---. 2 couchbase couchbase   20480 Jun  2 15:40 .
drwxrwx---. 7 couchbase couchbase    4096 May 31 13:25 ..
-rw-rw----. 1 couchbase couchbase   53339 May 31 17:41 1000.couch.2
-rw-rw----. 1 couchbase couchbase   49243 May 31 17:41 1001.couch.2
-rw-rw----. 1 couchbase couchbase   49243 May 31 17:41 1002.couch.2
-rw-rw----. 1 couchbase couchbase   49243 May 31 17:41 1003.couch.2
-rw-rw----. 1 couchbase couchbase   57435 May 31 17:41 1004.couch.2
-rw-rw----. 1 couchbase couchbase   49243 May 31 17:41 1005.couch.2
-rw-rw----. 1 couchbase couchbase   45147 May 31 17:41 1006.couch.2
<output truncated>
```

Well, that's a lot of results. (You'll probably see around 520 lines). I am specifically interested in vBucket 788 b/c that's where the key is supposed to be. Note, the vBucket # will be the same in your environment b/c the key will hash to the same vBucket #, however the vBucket might be hosted on a different node in your cluster.

Print a listing of all the files again, this time grepping for the specific vBucket # you are interested in:

```
[ec2-user@couchbase03 ~]$ sudo ls -al
/opt/couchbase/var/lib/couchbase/data/beer-sample | grep 788
-rw-rw----. 1 couchbase couchbase   28763 Mar 22 16:49 788.couch.1
```

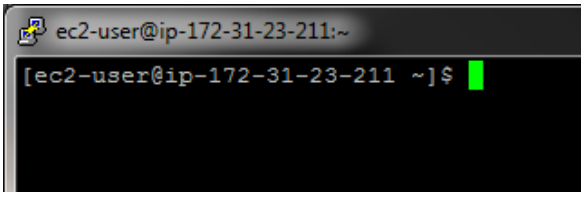
Next let's take a look inside the file to look for the specific key 'made-on-source'. (Remember that you may have to issue a different file name and a different vBucket #!):

```
[ec2-user@couchbase03 ~]$ sudo /opt/couchbase/bin/couch_dbdump
/opt/couchbase/var/lib/couchbase/data/beer-sample/788.couch.1 | grep -
B 2 -A 6 made-on-source
```

```
Doc seq: 7
     id: made-on-source
     rev: 1
     content_meta: 128
     size (on disk): 80
     cas: 1553273383237255168, expiry: 0, flags: 33554438, datatype:
0x01 (json)
     size: 72
     data: (snappy) {
"click": "to edit",
"with JSON": "there are no reserved field names"
```

Okay, great. So the revision count for the item in the source cluster is set to 1. This makes sense as I simply created the item and never updated or deleted it.
Let's try to find the revision count for the same item in the remote/destination cluster.

Switch to the App Server (black VM) to run the 'cbc hash' command:



Run the 'cbc cat' command (on the destination cluster) to first simply read the item. Provide the command the public hostname of Node #1 (Red VM) in the 2-node remote/destination cluster:

```
[ec2-user@appserver ~]$ cbc-cat -U $NODE7/beer-sample made-on-source
made-on-source       LCB_KEY_ENOENT (0x0D)
```

Well, you can't read the key here b/c you deleted it earlier, but the tombstone for the key should still exist. Let's check.

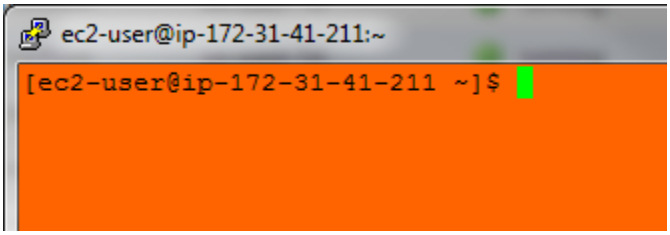
Run the 'cbc hash' command to find out where this key is actually stored in this 2-node cluster (again provide the command the public hostname of Node #1 on the remote/destination side):

```
[ec2-user@appserver ~]$ cbc-hash -U $NODE7/beer-sample made-on-source

made-on-source: [vBucket=788, Index=1] Server: ec2-54-219-174-110.us-
west-1.compute.amazonaws.com:11210, CouchAPI: http://ec2-54-219-174-
110.us-west-1.compute.amazonaws.com:8092/beer-sample
Replica #0: Index=0, Host=ec2-54-219-170-26.us-west-
1.compute.amazonaws.com:11210
```

The vBucket ID is the same here, #788 for my specific environment. I now also know which server is supposed to be hosting the active copy of this item.

In my environment that host name (`ec2-54-219-174-110.us-west-1.compute.amazonaws.com`) corresponds to Node #2 in the remote/destination side, which is the orange node.

Although I am going to be switching to the Node #2's shell (Orange VM/Couchbase08), you will have to either switch to the Red node (#1/Couchbase07) or the Orange node (#2/Couchbase08) in your environment:



From that node, let's try to find the tombstone marker for the delete. But first we need to find what the data file name is that contains vBucket #788 (or whichever one you are hunting for):

```
[ec2-user@couchbase08 ~]$ sudo ls -al
/opt/couchbase/var/lib/couchbase/data/beer-sample | grep 788
-rw-rw----. 1 couchbase couchbase   16475 Mar 22 17:04 788.couch.1
```

In my environment the file name is '788.couch.1' so it has never been compacted. That's fine. Let's now look for the actual item 'made-on-source':

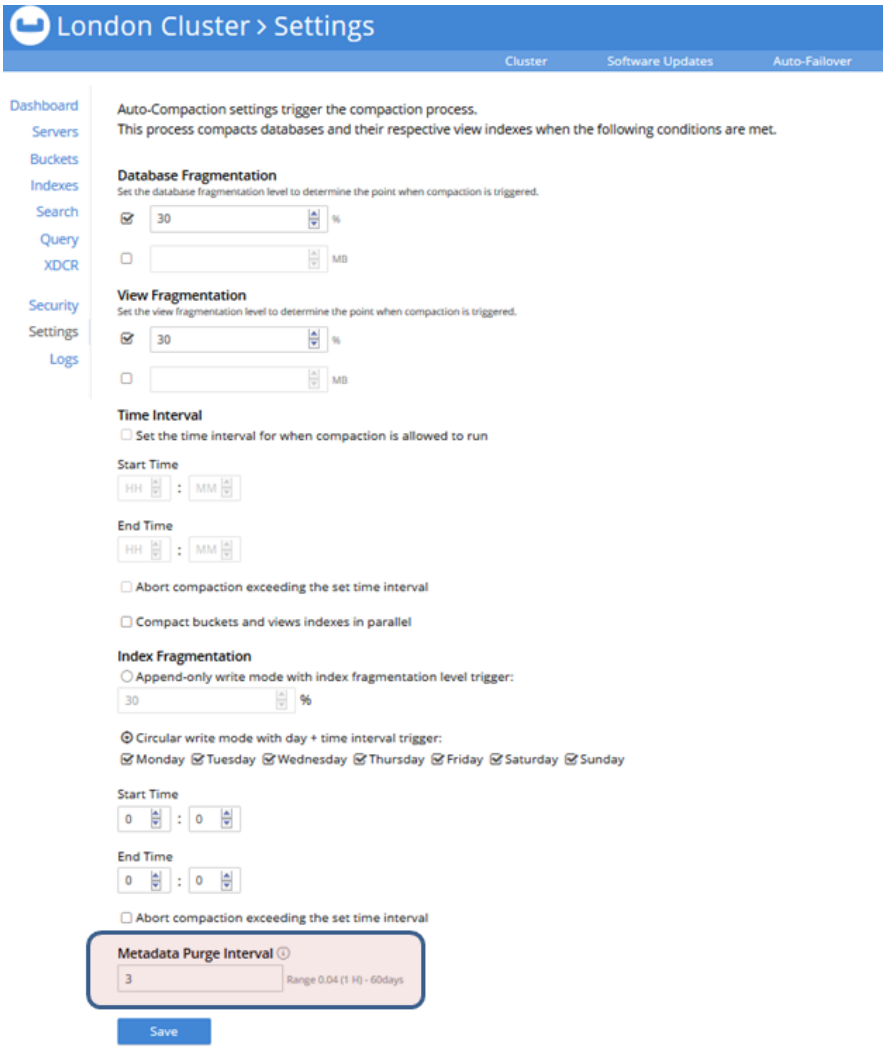
```
[ec2-user@couchbase08 ~]$ sudo /opt/couchbase/bin/couch_dbdump
/opt/couchbase/var/lib/couchbase/data/beer-sample/788.couch.1 |
grep -B 1 -A 8 made-on-source
Doc seq: 8
     id: made-on-source
     rev: 2
     content_meta: 3
     size (on disk): 0
     cas: 1553274265439371264, expiry: 1553274265, flags:
33554438, datatype: 0x00 (raw)
     doc deleted
     could not read document body: document not found

Total docs: 7
```

Aha! As suspected the revision count here is 2, which it was 1 on the source side. Perhaps a quick discussion of tombstones is in order...

Couchbase Server and other distributed databases maintain tombstones in order to provide eventual consistency between nodes and between clusters. Tombstones are records of expired or deleted items and they include the key for the item as well as metadata. Couchbase Server stores the key plus several bytes of metadata per deleted item. With millions of mutations, the space taken up by tombstones can grow quickly. This is especially the case if you have a large number of deletions or expired documents.

You can now configure the Metadata Purge Interval which sets how frequently a node will permanently purge metadata on deleted and expired items. This new setting will run as part of auto-compaction.

Note that you can set the metadata purge interval under the Settings screen in Couchbase. But this is just FYI. There is no need to change anything under the Settings in the Web UI.



Going back to our original problem, there are still 257,304 items on the source side's beer-sample bucket and 257,303 items in the destination side's beer-sample bucket.

Source:

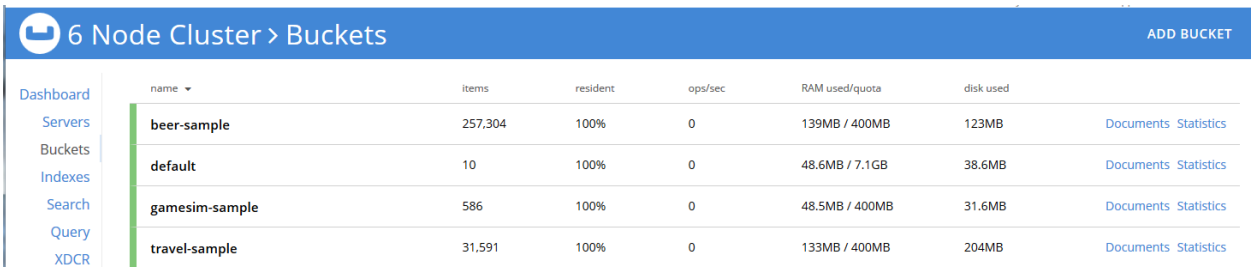
name	items
beer-sample	257,304

Destination:

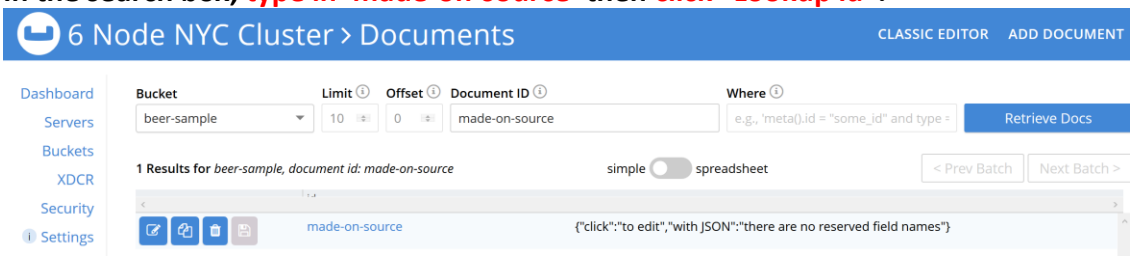
name	items
beer-sample	257,303

Switch to the Web UI browser tab for the 6-node production/source cluster and let's edit the 'made-on-source' item twice so that it has a higher revision count than the destination cluster (which has the count set to 2). Click on Data Buckets at the side, then click the Documents button for the beer-sample bucket:

Documents



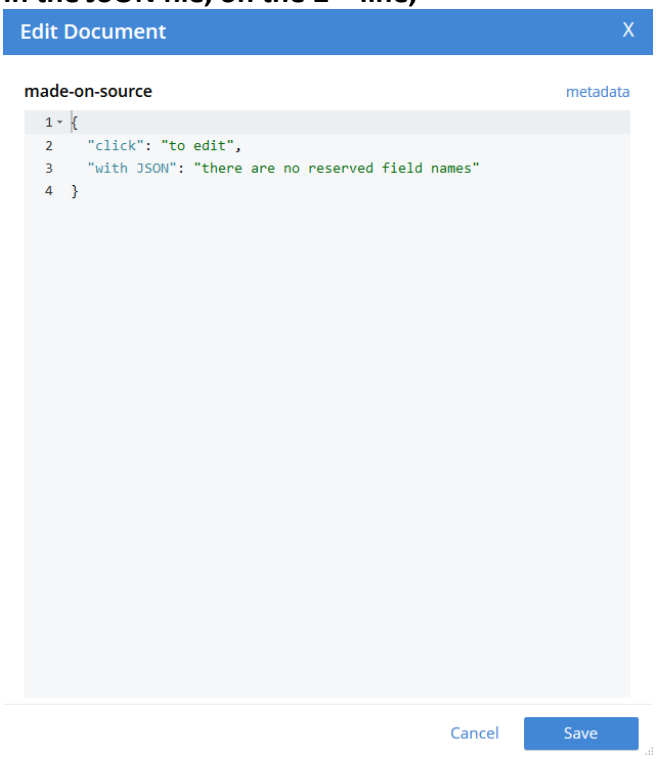
In the search box, type in 'made-on-source' then click "Lookup id":



Couchbase

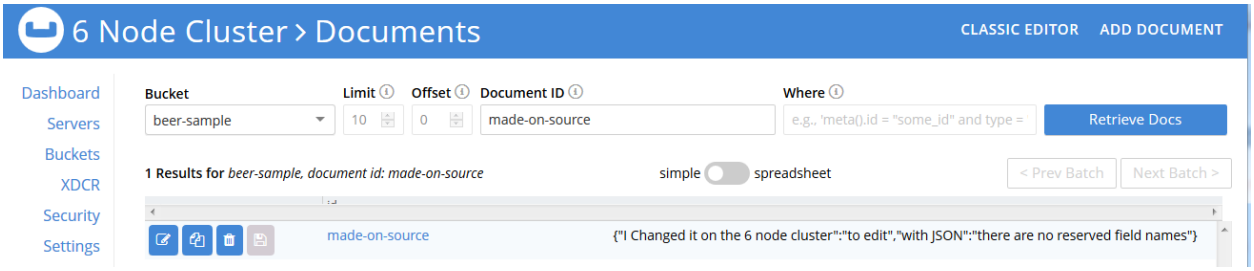
Click on the edit Icon 

In the JSON file, on the 2nd line,



change the word 'click' to "I Changed it on the 6 node cluster"

as seen in the screenshot below then click 



That was revision #2.

Now go back to terminal windows for each cluster and run "couch_dbdump syntax" and check on Source and destination clusters for rev level. Now , Source = rev:2, and Destination =

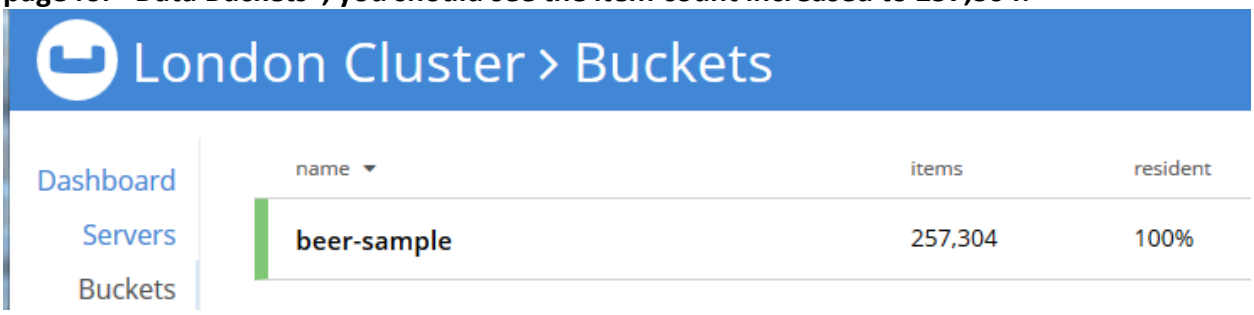
rev:2 (however now it shows doc size instead of "doc deleted") This should be in your command buffer already and accessible by arrow keys.

Now edit the JSON file's line #2 again.

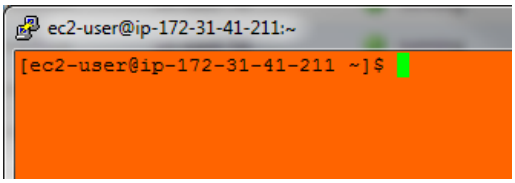
Where it says "to edit" change the words to "AGAIN" and click Save

And that was revision #3.

If you switch to the browser tab for the remote/destination cluster(London) and refresh the page for "Data Buckets", you should see the item count increased to 257,304:



Now switch to the Orange or Red node (which ever node you logged into a few pages prior when you were looking for the revision # for the item on the remote/destination side).
In my lab, that would mean XDCR Node # 2, Orange VM (but this might be red or orange for you!)



From this node, let's check the revision count for the item again on the destination side. Run a command similar to the following to see the revision count and data for this key. Note the file name/vBucket # might be different for you. You essentially should just have to hit the up arrow on your keyboard and switch the –A 5 to –A 6:

```
[ec2-user@couchbase08 ~]$ sudo /opt/couchbase/bin/couch_dbdump
/opt/couchbase/var/lib/couchbase/data/beer-sample/788.couch.1 | grep -
B 1 -A 8 made-on-source
Doc seq: 10
     id: made-on-source
     rev: 3
     content_meta: 128
     size (on disk): 113
     cas: 1553277687611916288, expiry: 0, flags: 33554438, datatype:
0x01 (json)
     size: 103
     data: (snappy) {
  "I Changed it on the 6 node cluster": "AGAIN",
  "with JSON": "there are no reserved field names"
```

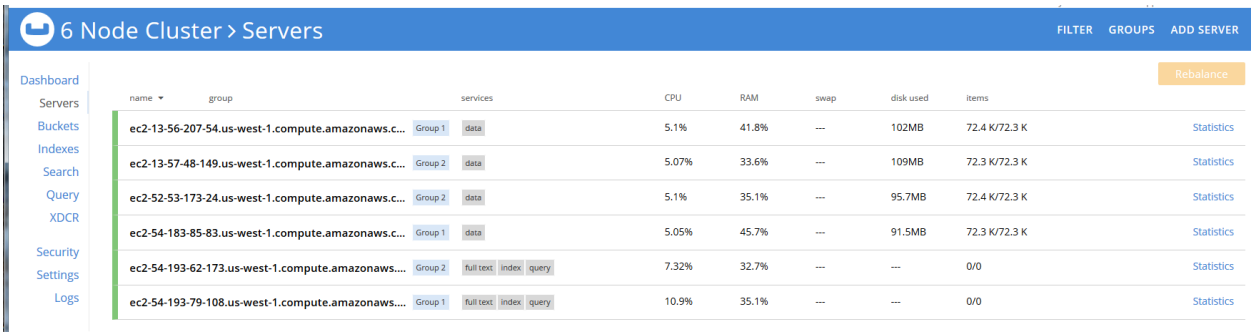
Notice that the data now appears on the destination side.

Viewing outbound and inbound replication statistics:

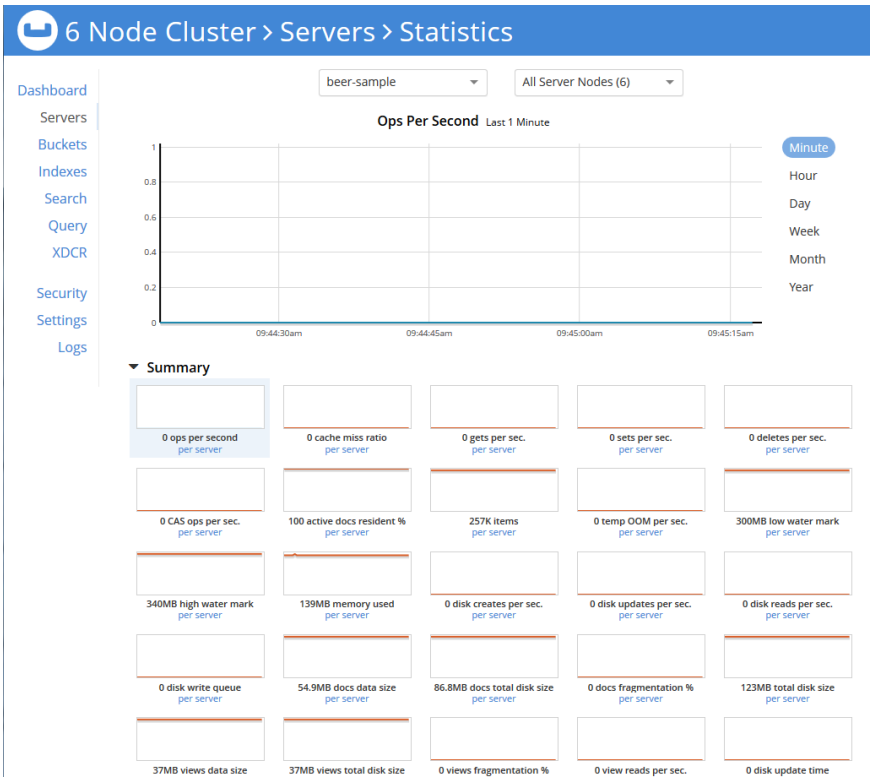
The Couchbase Web UI shows many relevant statistics for XDCR. In this section we will first load the correct web pages for outbound and inbound metrics and then use cbworkloadgen to push traffic into the production/source cluster and watch the traffic flow into the remote cluster.

Switch to the Web UI for the Production/Source 6-node cluster and click on "Server Nodes Link" then click on statistics link for that server :

Couchbase

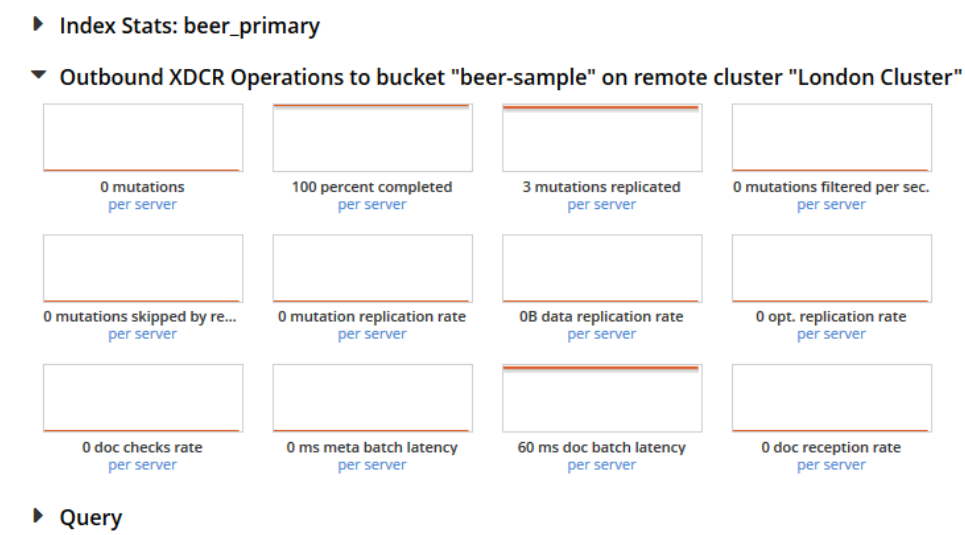


On the metrics page, switch to the beer-sample bucket on All Server Nodes (6):



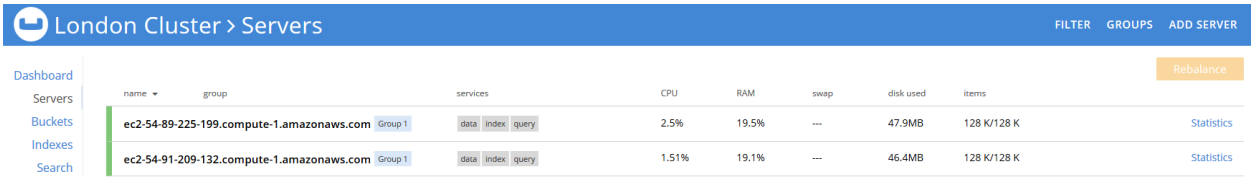
Scroll down and click the relevant arrow to expand the "OUTBOUND XDCR OPERATIONS" section:

Couchbase



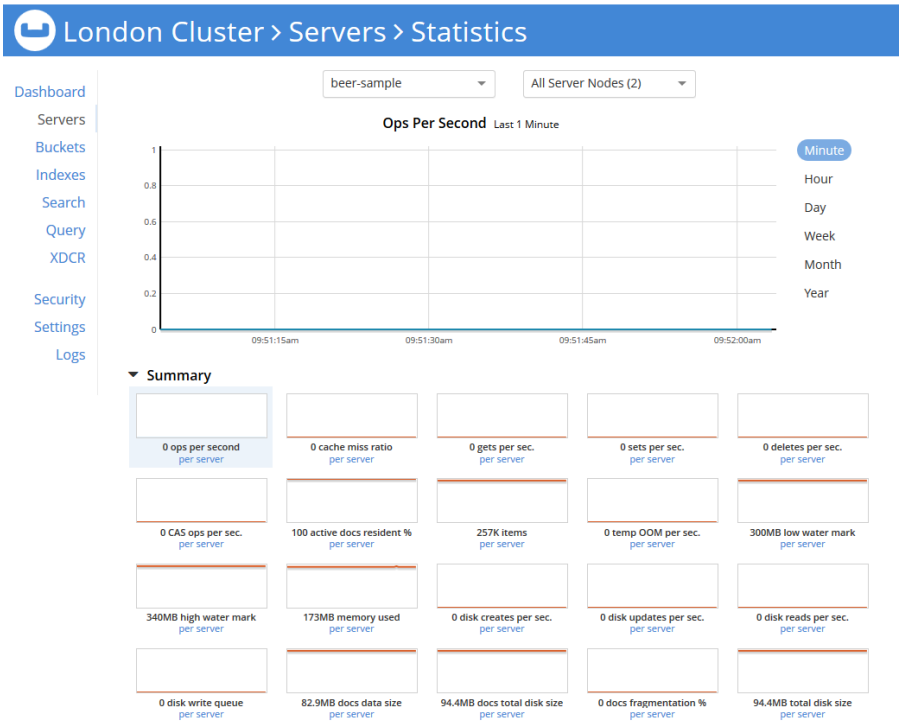
You should see 0 outbound XDCR mutations here as there is no data currently being written to, updated or deleted on the source/production side.
Keep this tab as is b/c we will return to it once we start generating traffic.
Next we need to also open the INBOUND XDCR metrics on the remote/destination side.

Switch to the Web UI for the remote/destination 2-node cluster(London) and click on "Server Nodes Link " then click on the statistics for that server :

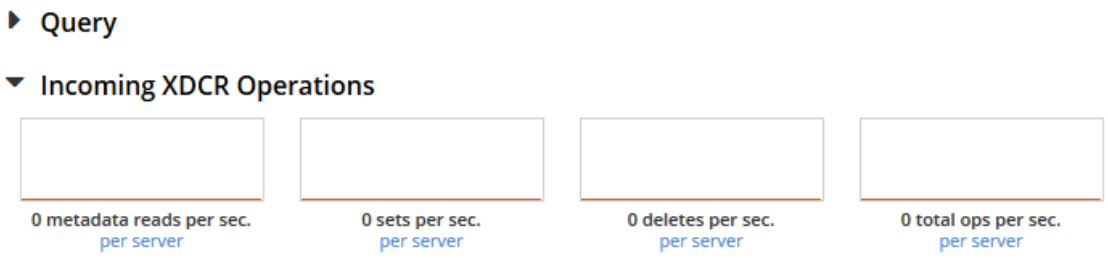


On the metrics page, switch to the beer-sample bucket on All Server Nodes (2):

Couchbase



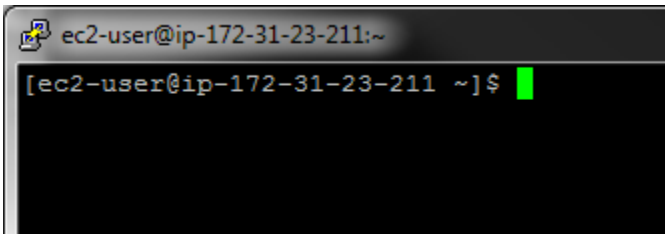
Scroll down and click the relevant blue arrow to expand the "INCOMING XDCR OPERATIONS" section:



This section should also show 0 for all 4 metrics it displays.

Now we are ready to dump some fake data into the 6-node production/source cluster's beer-sample bucket!

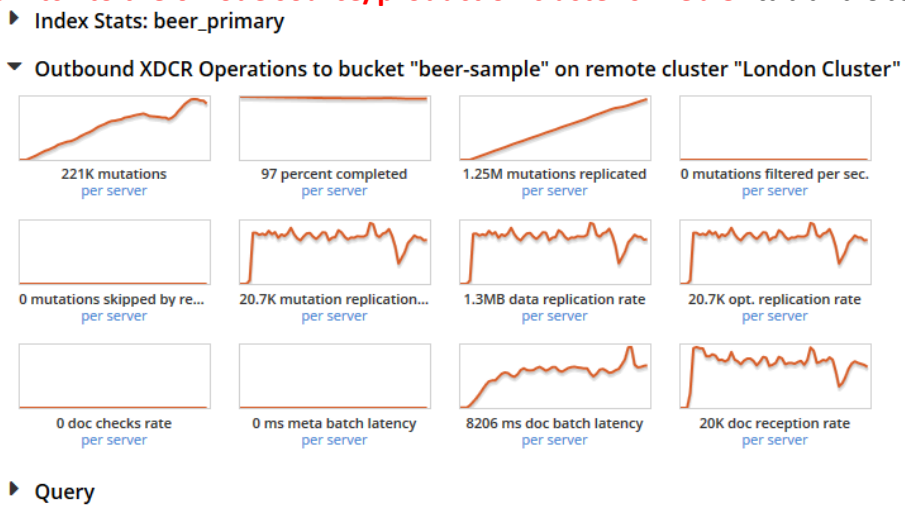
Switch to the App Server (black VM):



Run the following command to insert 2 million keys (2,000,000 keys) of size 10 bytes with 100% write workload into the source beer-sample bucket.
Use Node #1 (dark blue VM) in the source/production cluster's public hostname for the command below.

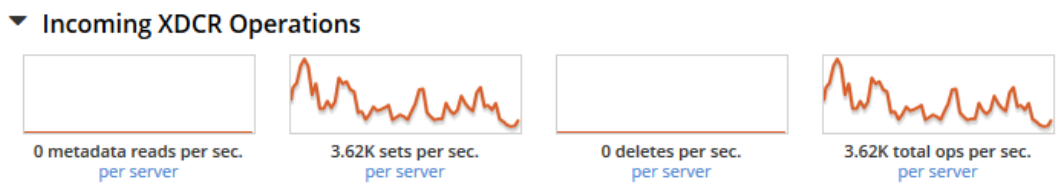
```
[ec2-user@appserver ~]$ cbworkloadgen -n $NODE1:8091 -b beer-sample -i 2000000 -r 1 -s 10
[#                    ] 5.6% (39000/estimated 700000 msgs)
```

Switch to the 6-node source/production cluster's Web UI tab and observe the activity:



Notice in the screenshot above that my source cluster is pushing out about 221K mutations outbound. Yours will vary.

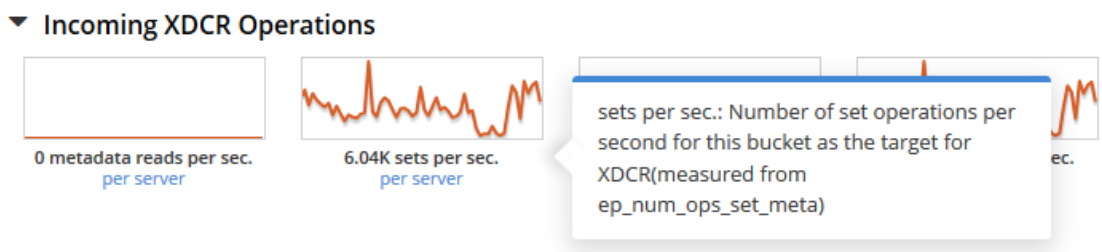
Switch to the 2-node destination cluster's Web UI and observe the activity here:



Notice the destination cluster is seeing about 3.62K sets per second.

Note that cbworkloadgen will take about 5 - 7 minutes to insert the 2 million keys. So, while that is occurring, I suggest clicking on some of the graphs in the source and destination cluster's web UI to see what each chart means.

For example, here I clicked on a specific graph and pulled up an explanation of what the set operations per second chart actually means:



When you are finished studying the XDCR metrics, switch back to the App Server (black VM) and if the cbworkloadgen is still running just hit <CTRL> + C in the PuTTY shell to stop the workload generator. If the generator stopped on its own b/c it already inserted 2 million keys, that is fine as well, no need to terminate it.(most likely it will run bucket out of memory)

```
2014-06-02 17:37:34,796: s0 backing off, secs: 10.0
2014-06-02 17:37:45,113: s0 backing off, secs: 10.0
2014-06-02 17:37:55,389: s0 backing off, secs: 10.0
2014-06-02 17:38:05,595: s0 backing off, secs: 10.0
<CTRL> + C
^Cinterrupted.
[ec2-user@ip-172-31-23-211 ~]$
```

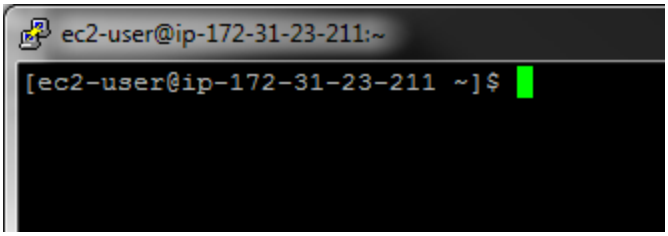
The specific error messages that you see above mean that the memory on the source cluster is too full and the nodes are sending 'tmp_oom' messages.

You can ignore these errors.

Viewing internal XDCR settings via REST API:

There are internal settings for XDCR which are only exposed via the REST API. These settings will change the replication behavior, performance, and timing.

Switch to the App Server (black VM):



Run the following curl command against the public hostname of Node #1 on the 6-node production/source side:

```
[ec2-user@appserver ~]$ curl -u Administrator:couchbase -n
http://$NODE1:8091/internalSettings
```

```
{"indexAwareRebalanceDisabled":false,"rebalanceIndexWaitingDisabled":false,"rebalanceIndexPausing
Disabled":false,"rebalanceIgnoreViewCompactions":false,"rebalanceMovesPerNode":1,"rebalanceMovesB
eforeCompaction":64,"maxParallelIndexers":4,"maxParallelReplicaIndexers":2,"maxBucketCount":10,"g
otraceback":"crash","indexAutoFailoverDisabled":true,"certUseSha1":false}
```

The above internal settings are applied to all servers for this cluster.

The next command will also display global settings for all replications for the cluster (Notice that some of these settings are the same as the command output above):

```
[ec2-user@appserver ~]$ curl -u Administrator:couchbase -n
http://$NODE1:8091/settings/replications/
```

```
{"checkpointInterval":600,"docBatchSizeKb":2048,"failureRestartInterval":10,"goGC":100,"goMaxProc
s":4,"logLevel":"Info","networkUsageLimit":0,"optimisticReplicationThreshold":256,"sourceNozzlePe
rNode":2,"statsInterval":1000,"targetNozzlePerNode":2,"workerBatchSize":500}
```

Finally, it is also possible to retrieve the settings for a specific replication for a bucket. But to do this, we first have to get the replication ID for the beer-sample -> beer-sample replication.

Run the following curl command against the Node #1 public hostname in the 6-node source cluster:

```
[ec2-user@appserver ~]$ curl -u Administrator:couchbase -n
http://$NODE1:8091/pools/default/tasks
```

```
[{"statusId":"79c344a944861b2f44c437e53522023b","type":"rebalance","status":"notRunning","statusI
sStale":false,"masterRequestTimedOut":false},{"cancelURI":"/controller/cancelXDCR/d93371a8b0cd141
71cab4a1c47122c0a%2Fbeer-sample%2Fbeer-
sample","settingsURI":"/settings/replications/d93371a8b0cd14171cab4a1c47122c0a%2Fbeer-
```

sample%2Fbeer-sample","status":"running","replicationType":"xmem","continuous":true,"filterExpression":"","id":"d93371a8b0cd14171cab4a1c47122c0a/beer-sample/beer-sample","pauseRequested":false,"source":"beer-sample","target":"/remoteClusters/d93371a8b0cd14171cab4a1c47122c0a/buckets/beer-sample","type":"xdcr","recommendedRefreshPeriod":10,"changesLeft":0,"docsChecked":8382,"docsWritten":1707577,"maxVBReps":null,"errors":[]}]

Notice that the replication ID is highlighted in yellow in the output above. You will need this specific ID from YOUR output for the next command. Also notice that this replication ID has checked and written 1,707,577 items (the specific # you see may vary).

Use the replication ID to run a curl command to get the specific settings for that replication ID:

```
[ec2-user@appserver ~]$ curl -u Administrator:couchbase -n
http://$NODE1:8091/settings/replications/d93371a8b0cd14171cab4a1c47122
c0a%2Fbeer-sample%2Fbeer-sample
{"checkpointInterval":600,"compressionType":"Auto","docBatchSizeKb":2048,"failureRestartInterval"
:10,"filterExpression":"","logLevel":"Info","networkUsageLimit":0,"optimisticReplicationThreshold
":256,"pauseRequested":false,"sourceNozzlePerNode":2,"statsInterval":1000,"targetNozzlePerNode":2
,"type":"xmem","workerBatchSize":500}
```

More details about how to change the advanced settings can be found here:
http://docs.couchbase.com/admin/admin/Tasks/xdcr-specify-adv-settings.html

Configuring encrypted, bidirectional, optimistic replication between 2 clusters:

Bidirectional Replication

In this section of the lab, we will create a bidirectional replication between the production 6-node and remote 2-node cluster.

Replication is normally unidirectional from one cluster to another. Bidirectional Replication allows two clusters to replicate data with each other. Setting up bidirectional replication in Couchbase Server involves setting up two unidirectional replication links from one cluster to the other. This is useful when you want to load balance your workload across two clusters where each cluster bi-directionally replicates data to the other cluster.

To configure bidirectional replication between two clusters, you need to provide settings for two separate replication streams. One stream replicates changes from Cluster A to Cluster B, another stream replicates changes from Cluster B to Cluster A. To configure a bidirectional replication:

Create a replication from Cluster A to Cluster B on Cluster A.

Create a replication from Cluster B to Cluster A on Cluster B.

You do not need identical topologies for both clusters; you can have a different number of nodes in each cluster, and different RAM and persistence configurations.

Optimistic Replication

The other setting we will alter in this section is the optimistic replication threshold which is set to 256 bytes by default. The setting can be between 0 bytes to 20 MB (2097152 bytes). This setting can improve the latency for XDCR. An explanation for why follows, but first you should understand how XDCR typically works.

Typically XDCR on the source cluster will get metadata for documents larger than this size (256 bytes by default) from the remote cluster once before replicating the document to a destination cluster. One of the reasons for this is to verify the revision count of the item in the destination cluster versus the revision count for the same item in the source cluster. If the revision count is higher in the destination cluster, there is no need to send the item to the destination as it would be discarded anyway. If the source item's revision count is higher, then the item is placed into the replication queue. Once the item arrives at the destination cluster, the revision count from the metadata in the destination cluster is queried again to make sure that a change did not occur on the destination cluster while the item was traveling over the network from the source to the destination. So, then a node on the destination side also verifies that this incoming item has a higher revision count than the item in the destination cluster and only then does the incoming item overwrite the item in the destination cluster.

However, when a document is smaller than the number of bytes provided as this parameter, XDCR on the source side immediately puts it into the replication queue without getting metadata from the destination side to the source cluster.

If the document is deleted on a source cluster, XDCR source will not fetch metadata for the document from the destination before it sends this update to a destination cluster. Once a document reaches the destination cluster, XDCR will fetch the metadata from the local destination cluster and perform conflict resolution between documents. If the document 'loses' conflict resolution, Couchbase Server discards it on the destination cluster and keeps the version on the destination. This new feature improves replication latency, particularly when you replicate small documents.

There are tradeoffs when you change this setting. If you set this low relative to document size, XDCR on the source will frequently check metadata from the destination. This will increase latency during replication, it also means that it will get metadata from destination cluster before it puts a document into the replication queue, and will get it again for the destination to

perform conflict resolution. The advantage is that you do not waste network bandwidth since XDCR will send less documents that will 'lose' the conflict resolution.

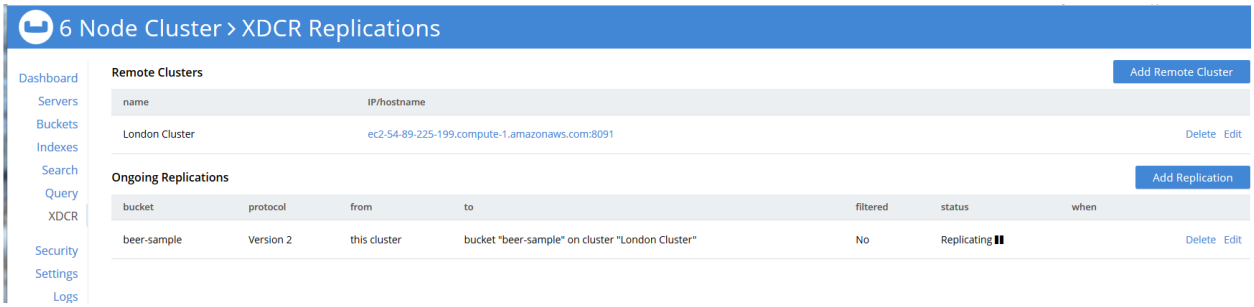
If you set this very high relative to document size, XDCR source cluster will fetch less metadata from the destination cluster which will improve latency during replication. This also means that you will increase the rate at which XDCR puts items immediately into the replication queue which can potentially overwhelm your network, especially if you set a high number of parallel replicators. This may increase the number of documents sent by XDCR which ultimately 'lose' conflicts at the destination which wastes network bandwidth.

So how to configure the setting for optimistic replication depends on your specific use case the insert/update/delete traffic patterns.

Note: XDCR does not fetch metadata for documents that are deleted.

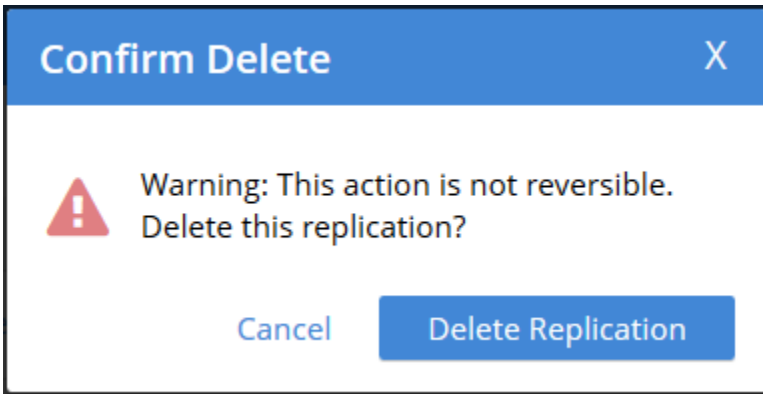
First let's delete the existing ongoing unidirectional replication so we can get a fresh start. Then we will delete the beer-sample and default buckets on both clusters. Then finally we will set up a fresh bucket on both sides for bidirectional replication and then study the optimistic replication threshold.

Switch to the browser tab for the 6-node production cluster and click on XDCR link.
Then click Delete next to the beer-sample replication:

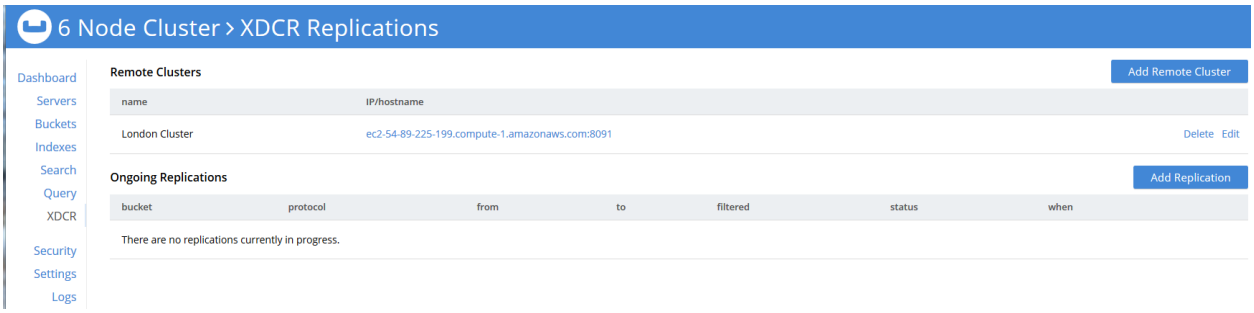


Click 

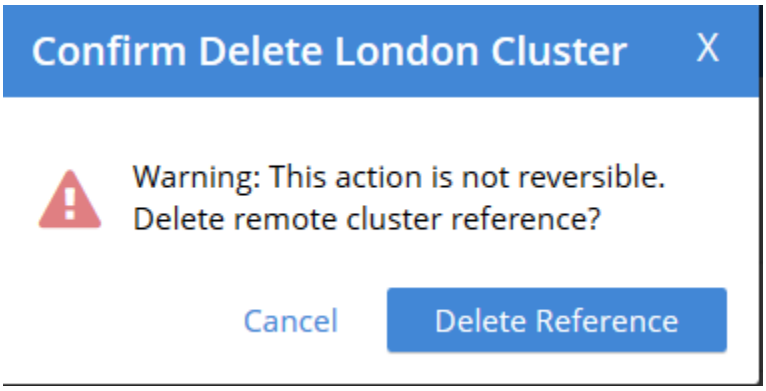
on the pop-up:



In the same Web UI, click Delete next to London to delete the remote cluster as well:

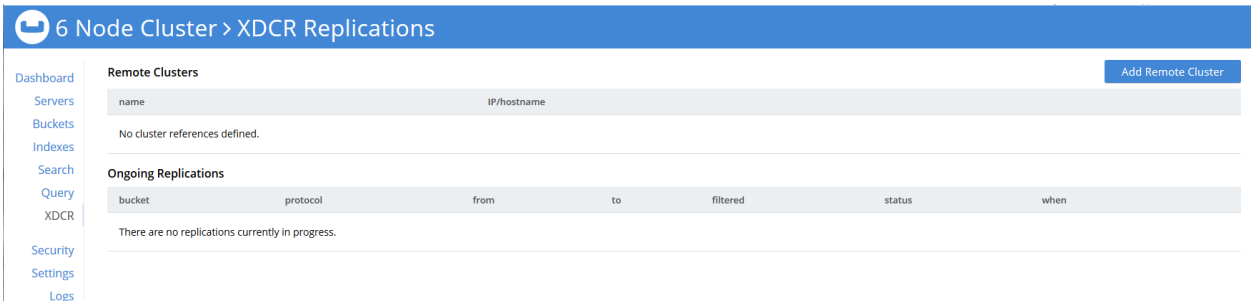


Confirm the deletion by clicking Delete Reference

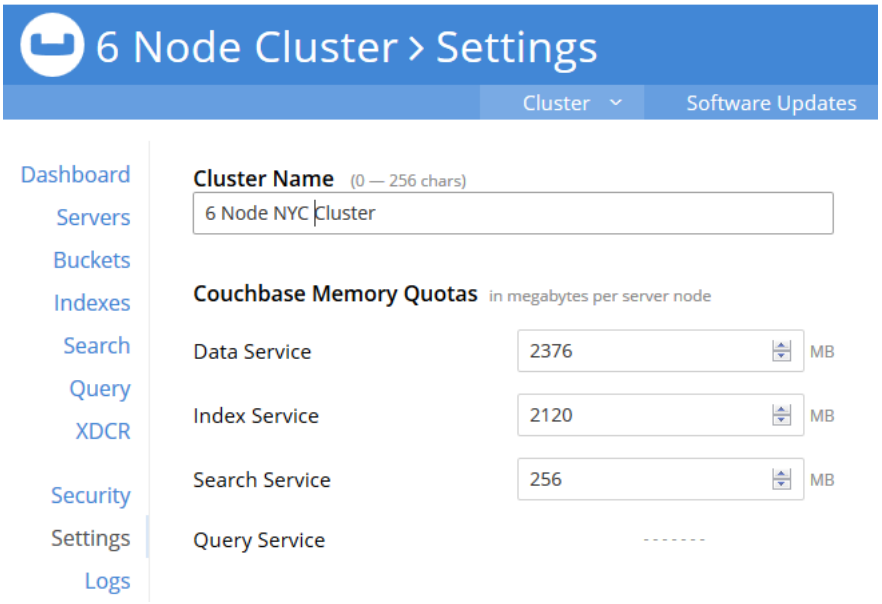


You should now see no replication settings defined for the 6-node cluster:

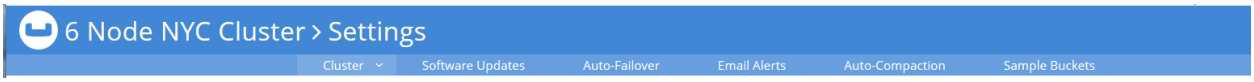
Couchbase



Go to the settings tab and rename the cluster "6 node NYC cluster"

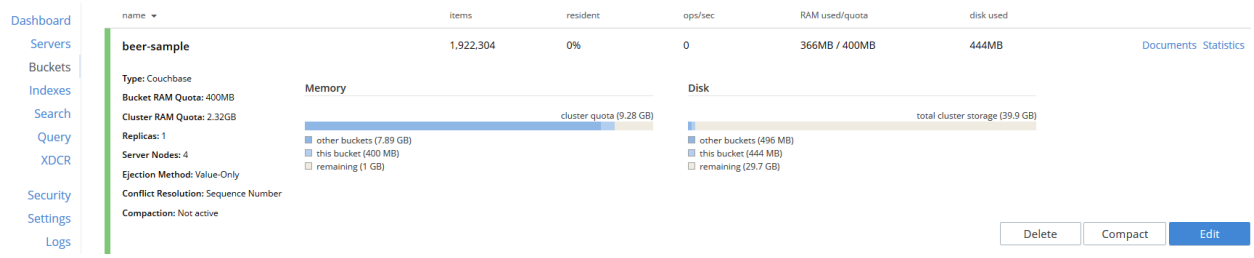


Click

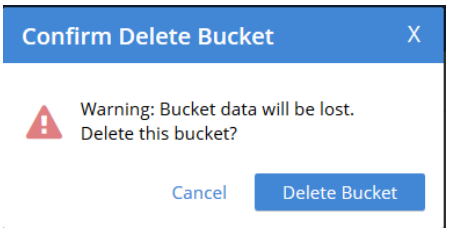


Next, in the same 6-node cluster's UI, click on Data Buckets at the side and then expand the beer-sample bucket by clicking in its area and finally click on Delete

Couchbase



Click Delete Bucket **to confirm the removal:**

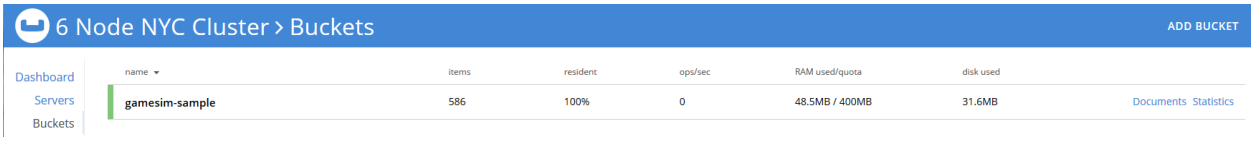


The beer-sample bucket will now disappear from the 6-node cluster's UI.
Next expand the 'default' bucket's settings by clicking the blue arrow next to 'default' and then click Delete:

Scroll down on the 'Configure Bucket' pop-up and click Delete:

Next delete the travel-sample bucket.

You should now see only the gamesim-sample bucket. Leave this bucket alone:

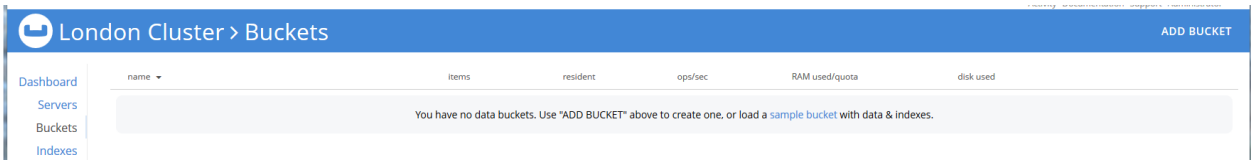


Now we'll delete the beer-sample bucket on the remote side. Switch to the remote 2-node cluster's(London) Web UI and click on Data Buckets at the side,
then expand the settings for beer-sample by click the blue arrow next to beer-sample and finally click on Delete:

Click Delete Bucket to confirm the removal on the popup :

You should now see zero buckets on the remote side's Web UI.
Click on "ADD BUCKET":

Couchbase

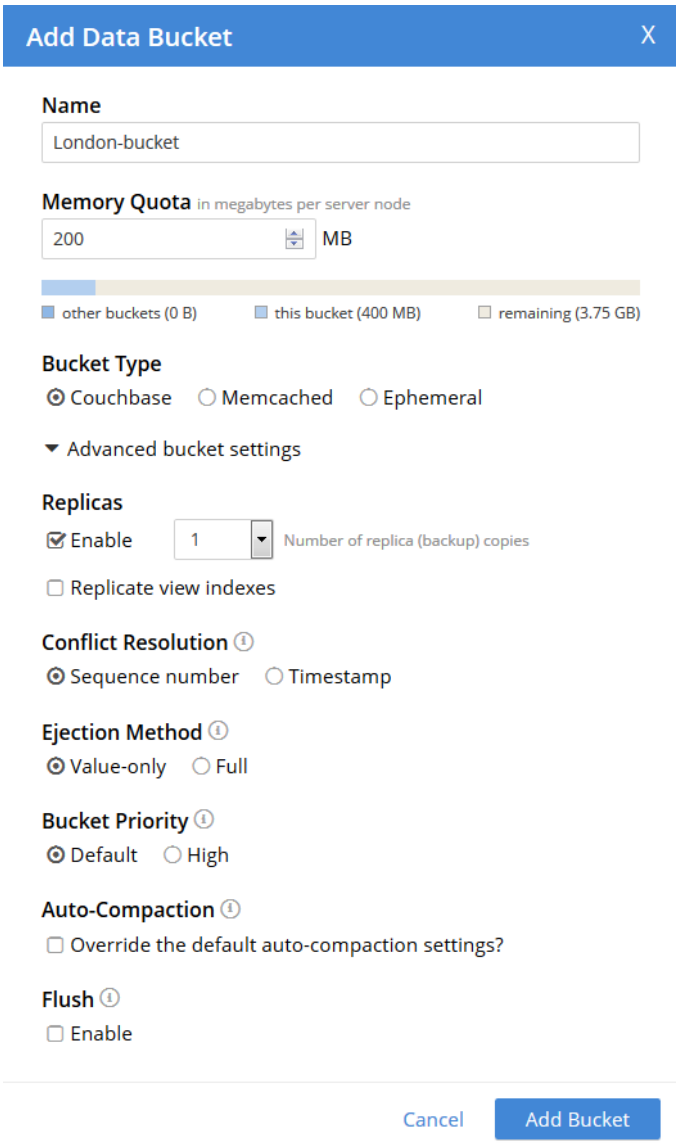


On the Create Bucket pop-up, use the following settings. Note only the settings in Red need to be changed:

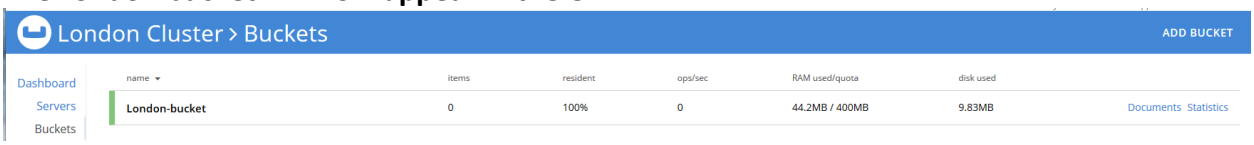
Bucket Name: London-bucket
Bucket Type: Couchbase
Per Node RAM Quota: 200 MB
Ejection Method: Value-only
Replicas: Enabled and set to 1
Index replicas: Not checked
Bucket Priority: Default
Override the default autocompaction settings: Not checked
Flush Enable: Not checked

Click Add Bucket

Couchbase

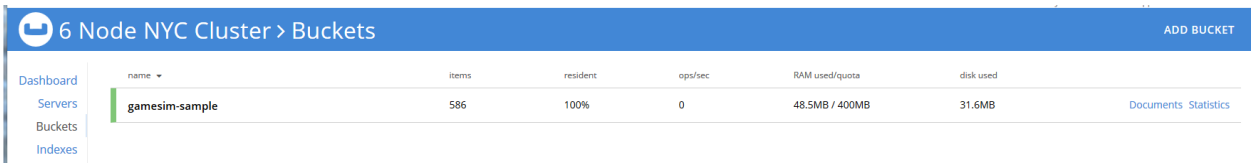


The London bucket will now appear in the UI:



Now, switch back to the 6-node production cluster(NYC) and let's create a bucket there named NYC-bucket. Click on Data Buckets at the side and then click "ADD BUCKET":

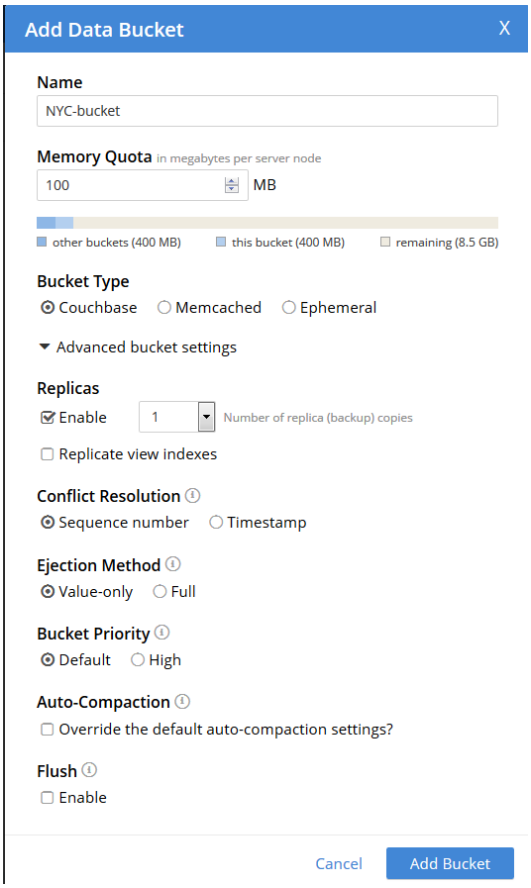
Couchbase



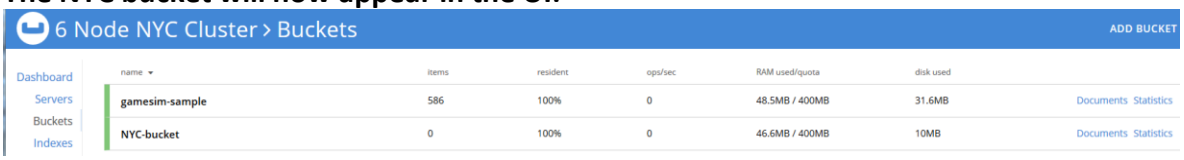
On the Create Bucket pop-up, use the following settings. Note only the settings in Red need to be changed:

Bucket Name: NYC-bucket
Bucket Type: Couchbase
Per Node RAM Quota: 100 MB
Ejection Method: Value-only
Replicas: Enabled and set to 1
Index replicas: Not checked
Bucket Priority: Default
Override the default autocompaction settings: Not checked
Flush Enable: Not checked

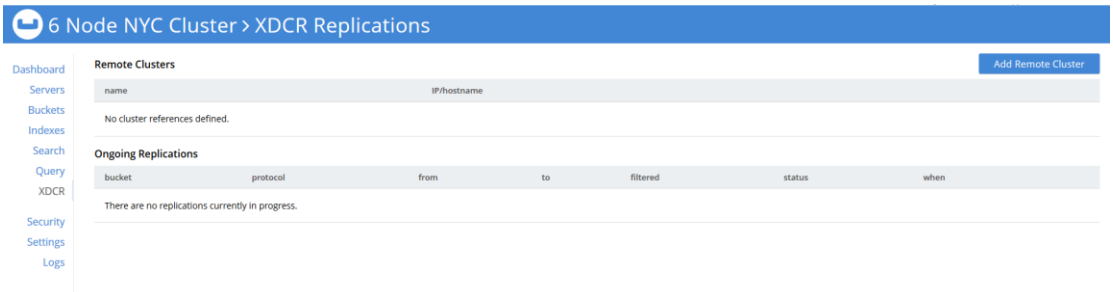
Click Add Bucket



The NYC bucket will now appear in the UI:



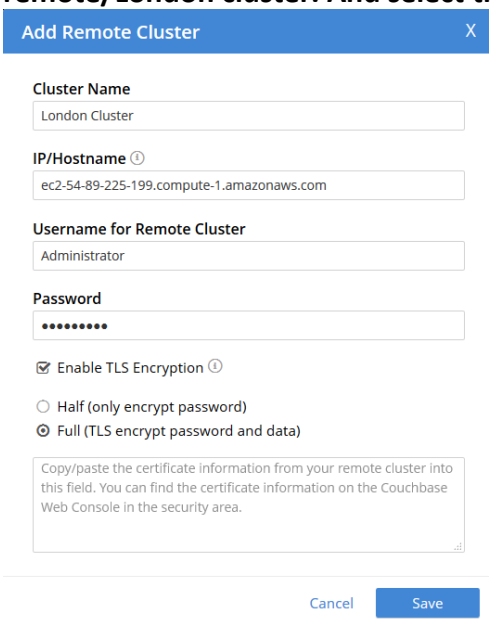
Next, let's configure encrypted, XDCR bidirectional replication. In this 6-node(NYC Cluster) cluster's UI, click on XDCR at the side and then click Add Remote Cluster



In the Add Remote Cluster Reference popup, enter the following:

Cluster Name: London Cluster
IP/hostname: <Node #1 in the remote/London cluster's Public Hostname (Red VM)
Username: Administrator
Password: couchbase
Enable Encryption: Place a check here

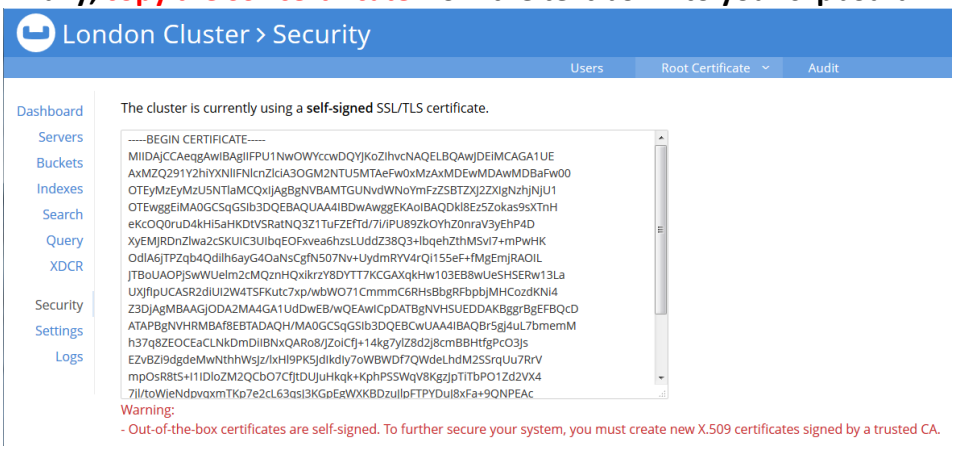
Now you will see the pop-up expand and request the encryption certificate from the remote/London cluster. And select the Full Encryption option.



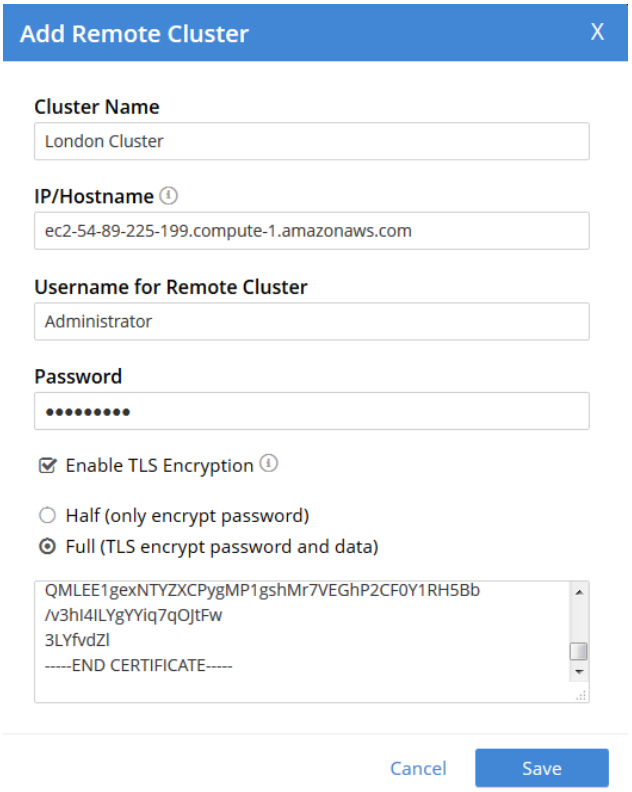
Couchbase

Switch to the 2-node cluster's browser tab and click on Security(link) at the side and then click the "Root Certificate" button.

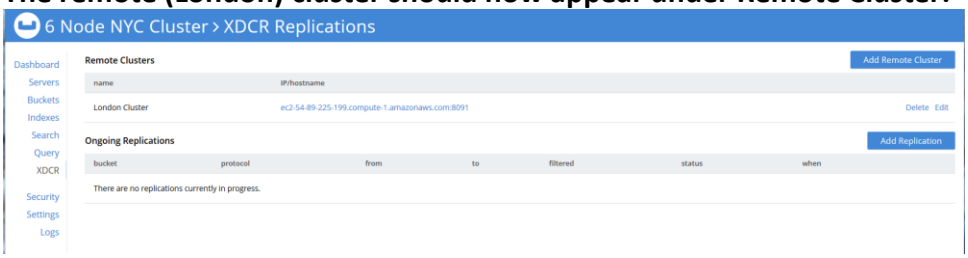
Finally, copy the SSL certificate from the text box into your clipboard.



Switch back to the other browser tab for the 6-node NYC cluster, paste the SSL certificate that you copied and click Save



The remote (London) cluster should now appear under Remote Cluster:

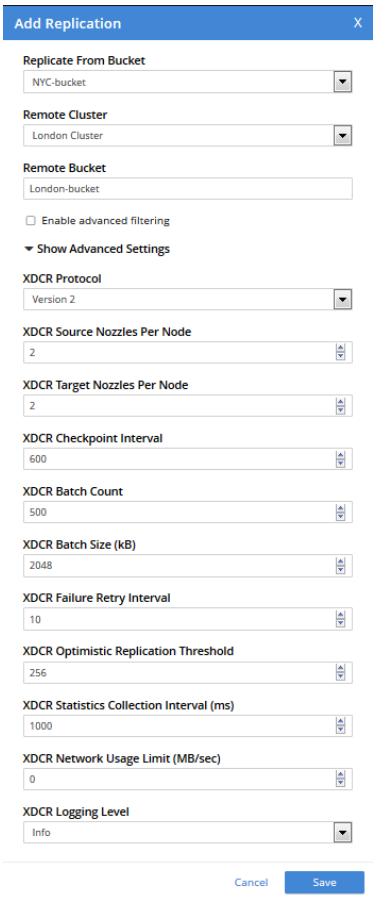


Next, let's configure XDCR bidirectional replication for the new bucket we created. In this same 6-node cluster's UI, click Add Replication

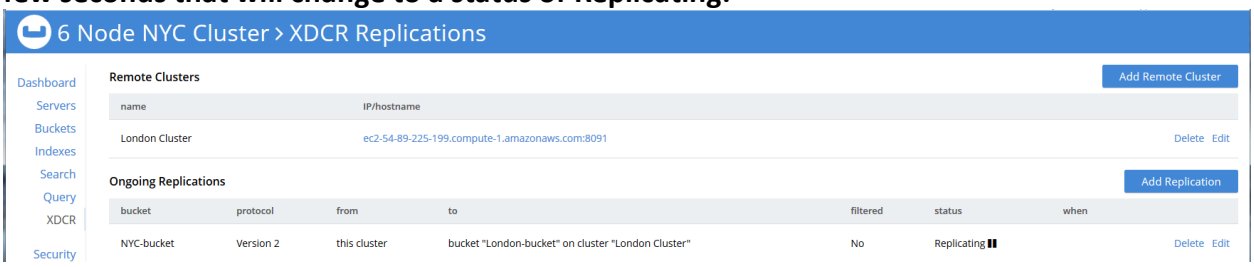
On the Create Replication pop-up, for 'Replicate From Bucket', choose the "NYC-bucket". For the 'Remote Cluster', choose "London Cluster" and for the Remote Bucket, Type in "London-bucket".

Then click on Advanced settings, just to see the options, but we will not change any of them. Just notice that the Optimistic Replication Threshold is set to 256.

Click Save



The ongoing replications section will now show the replication status as "Starting Up". In a few seconds that will change to a status of Replicating:

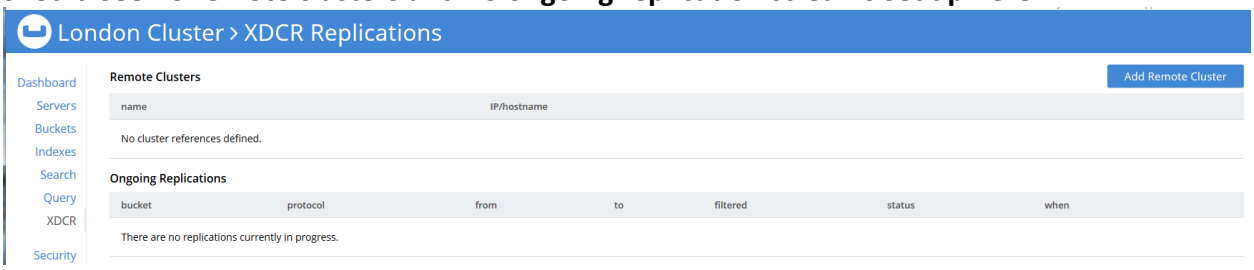


We are finished setting up the encrypted, unidirectional replication from the 6-node cluster (NYC) to the 2-node cluster (London). Let's imagine for now that the 6-node cluster is in NYC and the 2-node cluster is in London, even though this is not the case for the actual VMs in Amazon.

Users for our global application in the United States will hit the 6-node NYC cluster for reads + writes while users in Europe will hit the 2-node London cluster for reads + writes.

Next, need to set up unidirectional replication from the 2-node London cluster back to the 6-node NYC cluster.

Switch browser tabs to the 2-node London cluster and click XDCR at the side menu. You should see no remote clusters and no ongoing replication streams set up here.

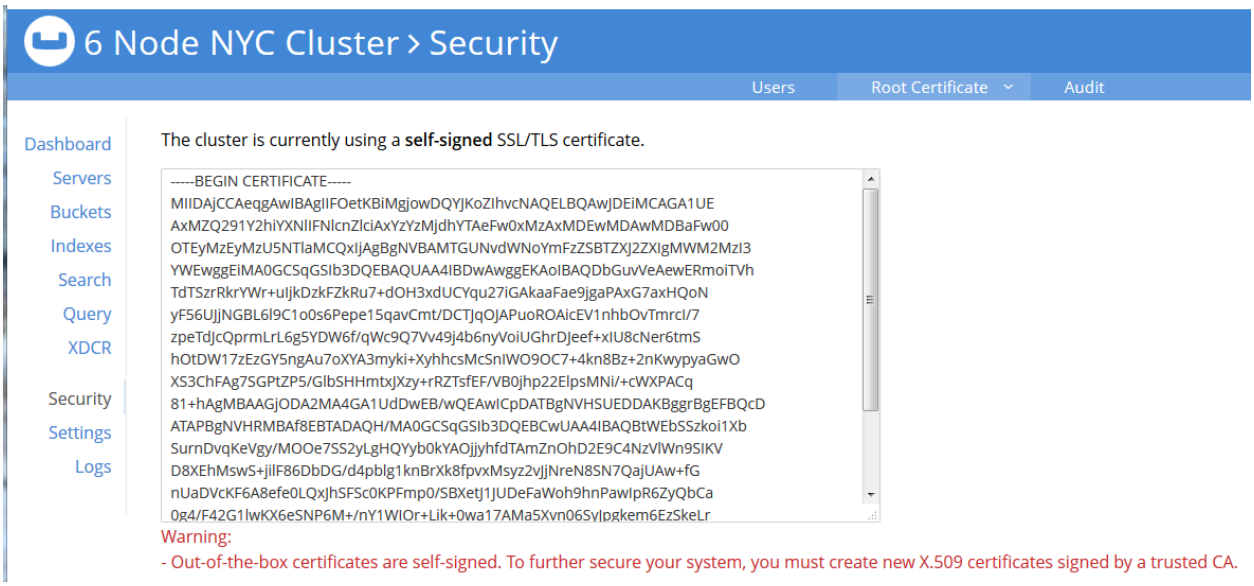


This time, let's use the couchbase-cli tool to create a remote XDCR cluster via an encrypted channel and then once again use the couchbase-cli tool to create a replication stream.

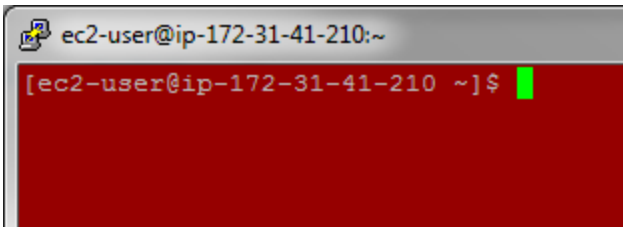
Note that the REST API can also be used to manage XDCR clusters and streams, but we will not explore the REST API further in this lab:
http://docs.couchbase.com/admin/admin/REST/rest-xdcr-intro.html

Switch browser tabs to the 6-node NYC cluster and click 'Security' at the side menu.
Then click on "Root Certificate" , and you should see the SSL Certificate.
Copy this SSL certificate to your clipboard.

Couchbase



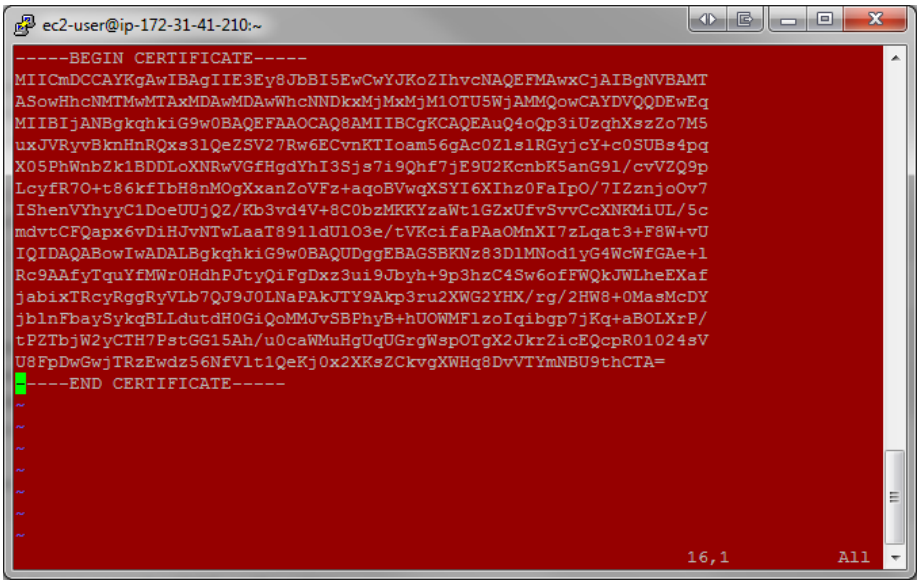
Next to create a cluster reference via the command line, switch to XDCR Node #1 (Red VM) in the 2-node London cluster:



Then we need to create a certificate file locally on this machine. You can use vi, vim, nano, emacs or any other linux text editor of your choice to create the .pem file. Ask the instructor for assistance if you are not familiar with any linux text editors! The vi command below may differ according to which text editor you choose.

```
[ec2-user@couchbase07 ~]$ cd ~
[ec2-user@couchbase07 ~]$ vi NYC-certfile.pem
```

Then paste the certificate into the text editor:



Finally save and quit out of the file.

If you cat the file, you should see the certificate:

```
[ec2-user@couchbase07 ~]$ cat NYC-certfile.pem
-----BEGIN CERTIFICATE-----
MIICmDCCAYKgAwIBAgIIE3Ey8JbBI5EwCwYJKoZIhvcNAQEFMAwxCjAIBgNVBAMT
ASowHhcNMTMwMTAxMDAwMDAwWhcNNDkxMjMxMjM1OTU5WjAMMQowCAYDVQQDEwEq
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAuQ4oQp3iUzqhXszZo7M5
uxJVRyvBknHnRQxs3lQeZSV27Rw6ECvnKTIoam56gAc0ZlslRGyjcY+c0SUBs4pq
X05PhWnbZk1BDDLoXNRwVGfHgdYhI3Sjs7i9Qhf7jE9U2KcnbK5anG9l/cvVZQ9p
LcyfR7O+t86kfIbH8nMOgXxanZoVFz+aqoBVwqXSYI6XIhz0FaIpO/7IZznjoOv7
IShenVYhyyC1DoeUUjQZ/Kb3vd4V+8C0bzMKKYzaWt1GZxUfvSvvCcXNKMiUL/5c
mdvtCFQapx6vDiHJvNTwLaaT891ldUlO3e/tVKcifaPAaOMnXI7zLqat3+F8W+vU
IQIDAQABowIwADALBgkqhkiG9w0BAQUDggEBAGSBKNz83DlMNod1yG4WcWfGAe+l
Rc9AAfyTquYfMWr0HdhPJtyQiFgDxz3ui9Jbyh+9p3hzC4Sw6ofFWQkJWLheEXaf
jabixTRcyRggRyVLb7QJ9J0LNaPAkJTY9Akp3ru2XWG2YHX/rg/2HW8+0MasMcDY
jblnFbaySykqBLLdutdH0GiQoMMJvSBPhyB+hUOWMFlzoIqibgp7jKq+aBOLXrP/
tPZTbjW2yCTH7PstGG15Ah/u0caWMuHgUqUGrgWspOTgX2JkrZicEQcpR01024sV
U8FpDwGwjTRzEwdz56NfVlt1QeKj0x2XKsZCkvgXWHq8DvVTYmNBU9thCTA=
-----END CERTIFICATE-----
```

Finally, submit the following command to create the cluster reference. Note that the --xdcr-hostname attribute will be the public hostname for Node #1 (dark blue VM) in the 6-node cluster:

```
[ec2-user@couchbase07 ~]$ couchbase-cli xdcr-setup -u Administrator -p couchbase -c localhost:8091 --create --xdcr-cluster-name=NYC-cluster --xdcr-hostname=ec2-13-56-188-91.us-west-1.compute.amazonaws.com:8091 --xdcr-username=Administrator --xdcr-password=couchbase --xdcr-certificate=/home/ec2-user/NYC-certfile.pem --xdcr-secure-connection=full
```

```
SUCCESS: Cluster reference created
```

The above SUCCESS message means the cluster reference creation was successful.

NOTE: you had to specify the ec2 amazon address since the variables for hostnames had not been set on this node.

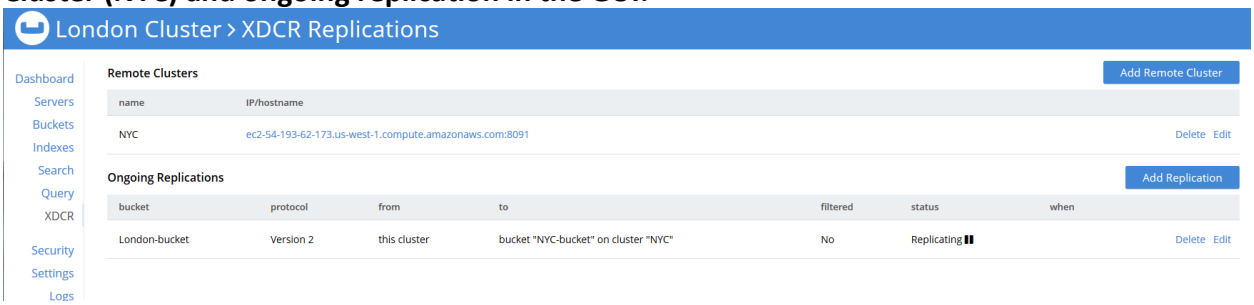
Almost there. Now run the following command to start a replication stream from London-bucket back to NYC-bucket:

```
[ec2-user@couchbase07 ~]$ couchbase-cli xdcr-replicate -c localhost:8091 -u Administrator -p couchbase --create --xdcr-cluster-name=NYC-cluster --xdcr-from-bucket=London-bucket --xdcr-to-bucket=NYC-bucket
```

```
SUCCESS: XDCR replication created
```

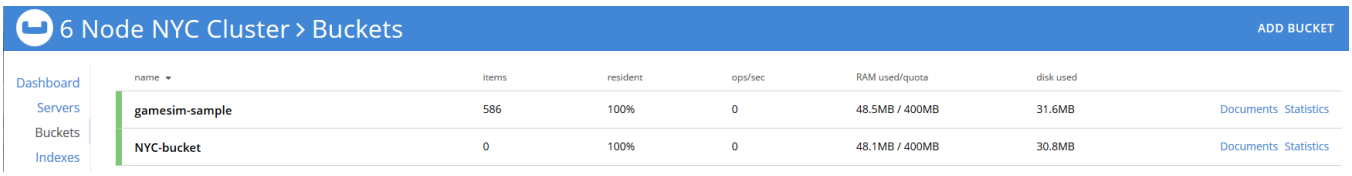
The above SUCCESS message means the replication stream creation was successful.

Next, switch over to the 2-node London cluster's Web UI and click on XDCR to see the Remote Cluster (NYC) and ongoing replication in the GUI:

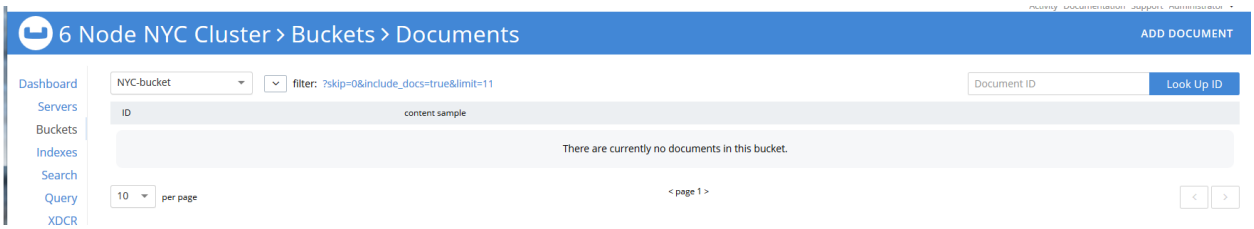


Now that our encrypted, bidirectional replication is set up, let's create a simple item in NYC/6-node cluster and see if it replicates to the London side. And then let's create a simple item in London/2-node cluster and see if it replicates to the NYC side.
We'll create the simple item in the NYC cluster first.

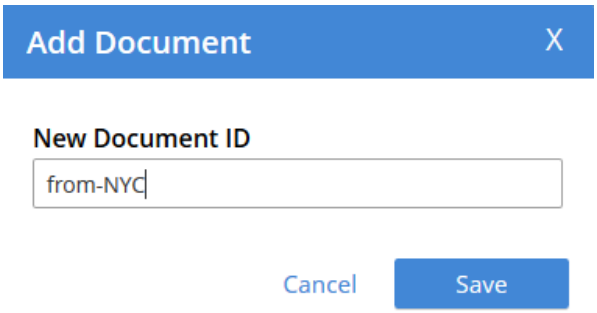
Switch to the 6-node NYC cluster's browser tab.
Then click "Buckets link" in the side menu and click on Documents link for the NYC-bucket:



Click ADD DOCUMENT

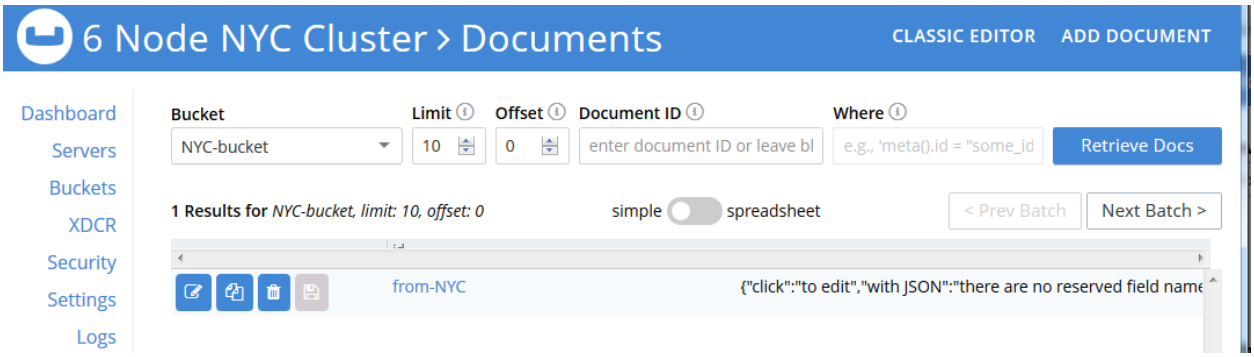


Name the Document ID "from-NYC" and Save



On the edit document popup click Save

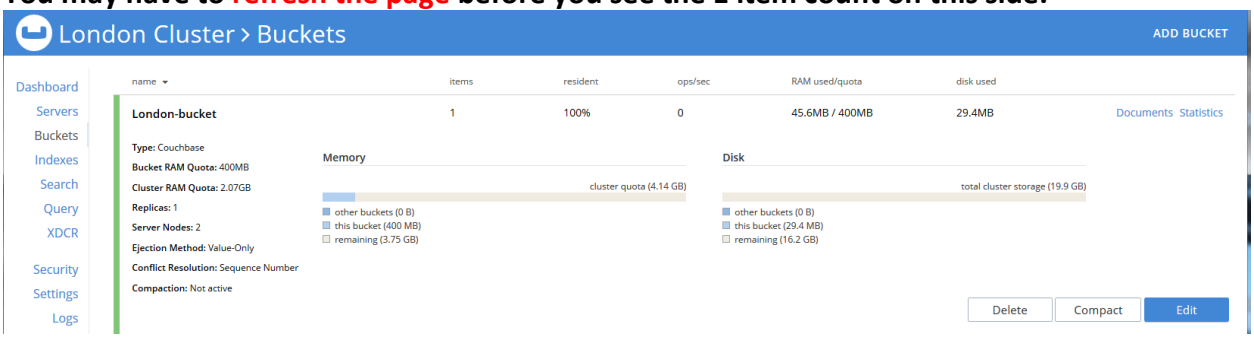
The default JSON document now appears in the UI:



Switch to the 2-node London cluster's browser tab. Then click "Buckets link" in the side menu.

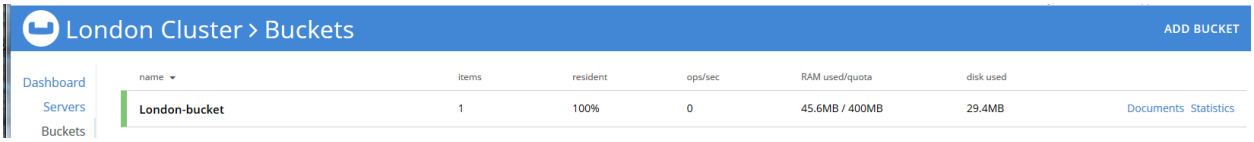
Couchbase

You may have to refresh the page before you see the 1 item count on this side.



So, the item we created in NYC successfully replicated to London.

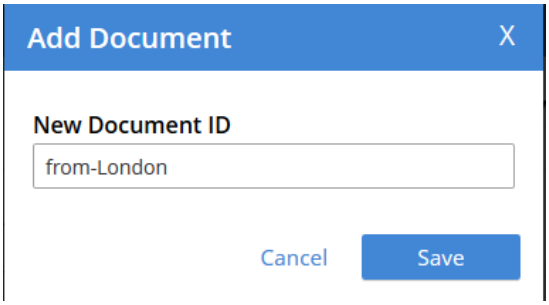
Click Documents:



You should now see the 'from-NYC' key here in London.

Click on the ADD DOCUMENT **in the London-cluster on the London-bucket**

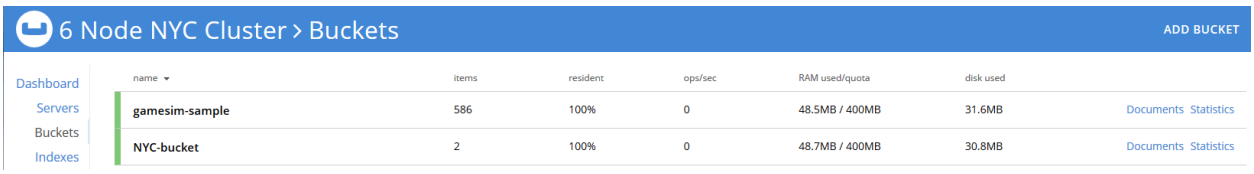
In the pop-up, name the Document ID "from-London" and Save



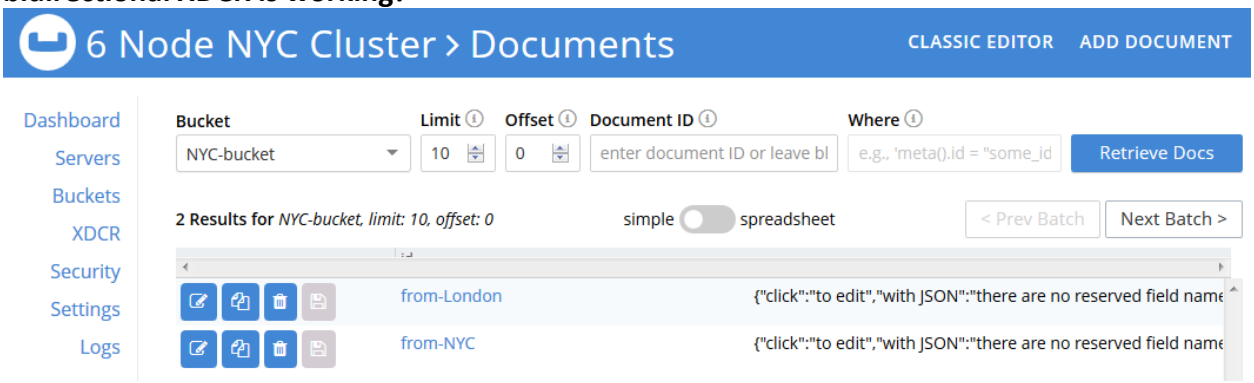
On the edit document popup , click

Save

Now switch back to the 6-node NYC cluster's browser tab to verify that the 2nd key we created shows up there.
Click 'Data Buckets' and see that the item count is 2 (you may have to refresh the page).
Next click on Documents:

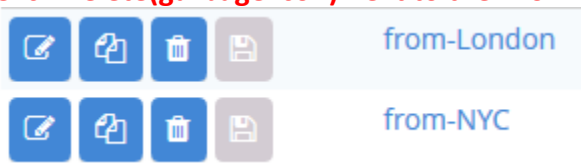


You should now see both keys in the NYC cluster. Excellent, we have now verified that bidirectional XDCR is working!

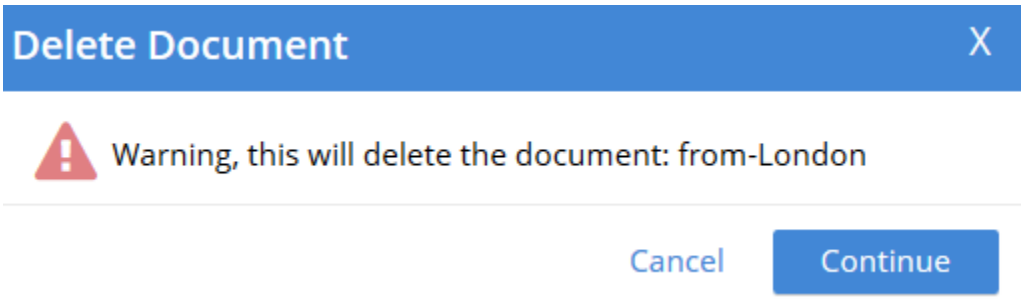


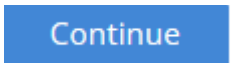
What will happen if we delete both keys from NYC? Do you think both keys will also get deleted from London? Let's find out…

Click Delete(garbage icon) next to the 'from-London' key to delete it:

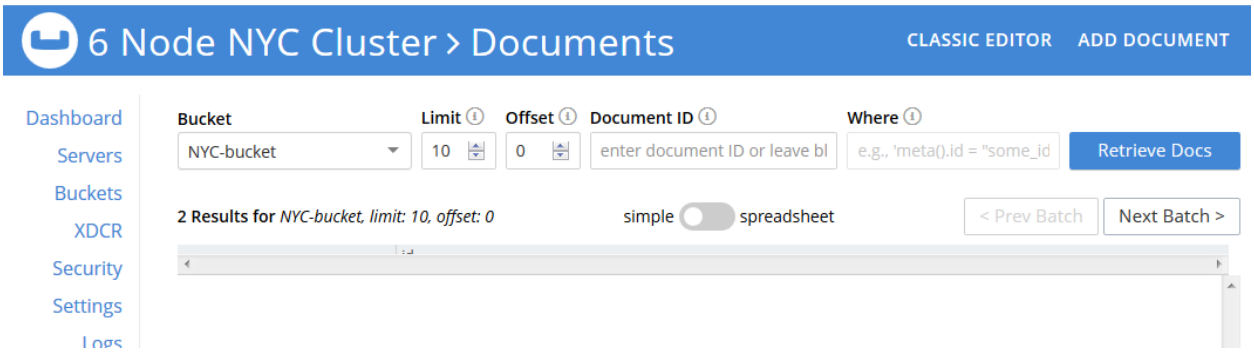


Couchbase



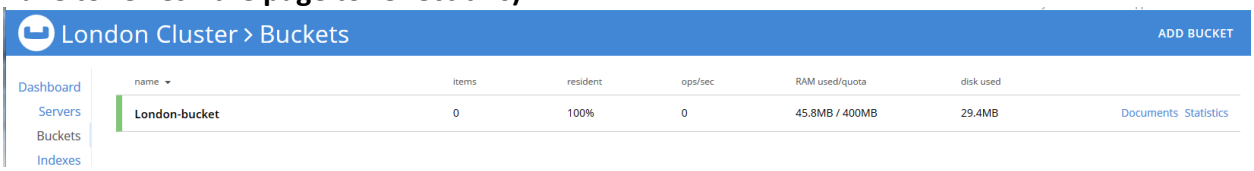
Click continue 

Click Delete for the 'from-NYC' key also:



There will now be zero keys in the NYC cluster's NYC-bucket:

Switch to the browser tab for the 2-node London cluster. Then click "Buckets link" in the side menu and notice that the item count is zero, so both keys did indeed get deleted. (You may have to refresh the page to reflect this)



This concludes Lab #6.