Couchbase Server 3.0 Documentation

Contents

Couchbase 3.0 beta introduction

This documentation set is for the Couchbase Server 3.0 Beta.

The documentation groups the new features in one section. Each feature section contains concepts or tasks, references to additional topics, and identifies topics that are impacted one way or another by the features. In some cases, existing topics may be modified or replaced.

Until General Availability (GA), the documentation may and, most likely, will undergo significant updates and restructuring.

PLEASE feel free to submit comments, requests, and suggestions to the Documentation Team at docs@couchbase.com.

Couchbase 3.0 Features

The Couchbase Server 3.0 release includes the following new features:

Related Links

Database Change Protocol on page 10
Database Change Protocol (DCP) is the protocol used to stream data changes to buckets.

Shared thread pool on page 14
A shared thread pool is a collection of threads which are shared across multiple buckets.

Metadata ejection on page 16
Metadata ejection specify whether value-only or full metadata is ejected from memory.

Disk I/O priority on page 19
Disk I/O priority allows workload priorities to be set at the bucket level.

Graceful failover on page 22
Graceful failover waits until data persists to disk and replicas are up-to-date before failing over the server node.

Delta node recovery on page 25
Delta node recovery provides the option to specify whether the replacement server node's disk is initialized or whether the data and data buckets are retained.

Incremental backup and restore on page 29
An incremental backup or restore is a backup (or restore) of the latest changes since the last backup (any level).

Stream-based views on page 31
Indexing is performed as soon as data changes in memory without waiting for persistence to disk. As a result, views have up-to-date data and faster queries.

Pause and resume replication on page 32
During XDCR replication, the process can be paused and resumed. Performed via the Couchbase command-line interface (CLI).

Stream-based XDCR on page 35
Stream-based XDCR collects data changes from memory on the source cluster and streams the data changes directly to memory on the destination cluster.

Encrypted admin access on page 36
Encrypted administrator access provides encrypted REST API access using Secure Socket Layer (SSL) authentication.

Encrypted data access on page 38
Encrypted data access is a communication connection between the client and Couchbase Server cluster using Secure Sockets Layer (SSL).

Configuring clients for SSL on page 40
Cluster-wide diagnostics on page 42
The Couchbase support team uses logs and other system diagnostics to analyze customer issues.

Database Change Protocol

Database Change Protocol (DCP) is the protocol used to stream data changes to buckets.

The Database Change Protocol (DCP) is a streaming protocol that significantly reduces latency for view updates. With DCP, changes made to documents in-memory are immediately streamed to be indexed without being written to disk. This provides faster view consistency which provides fresher data. DCP reduces latency for cross-datacenter replication. Data is replicated memory-to-memory from the source cluster to the destination cluster before being written to disk on the source cluster.

Impacted documentation

- cbstats - statistics, per client stats, and aggregate stats for DCP

Related Links

Couchbase 3.0 Features on page 9

DCP queues (UI) on page 10
UPR queues provide information about the Database Change Protocol (DCR) protocol used to stream data changes to buckets.

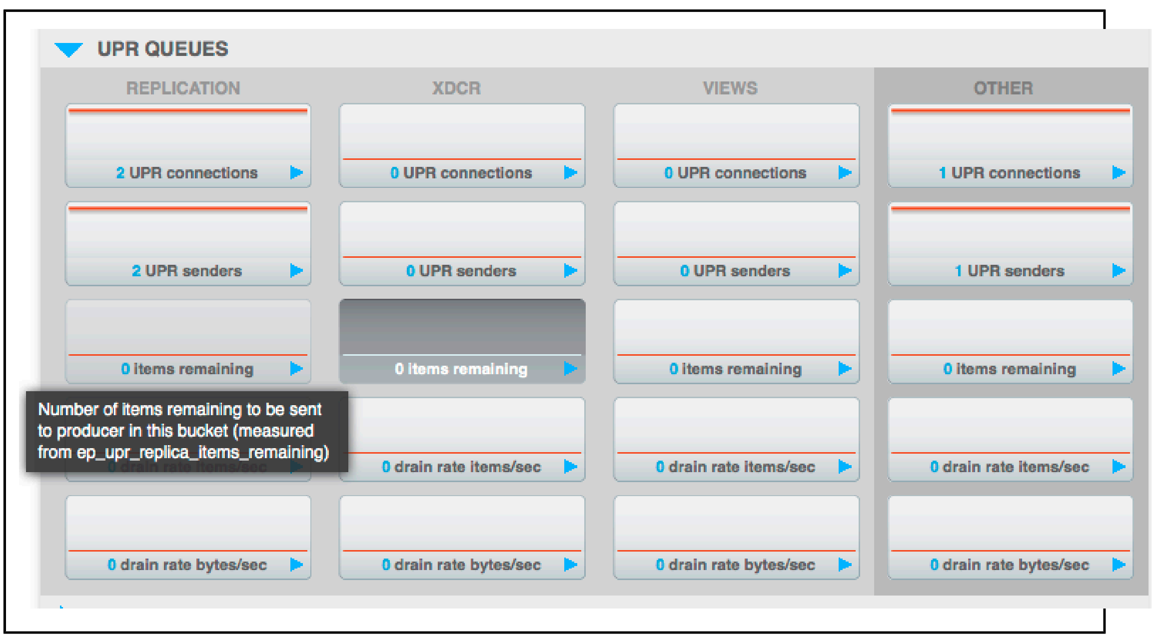
DCP statistics (CLI) on page 11
Statistics for the Database Change Protocol (DCP) are obtained via the CLI with the `cbstats` tool.

DCP queues (UI)

UPR queues provide information about the Database Change Protocol (DCR) protocol used to stream data changes to buckets.

The UPR queues information is available for each node via the **Server Nodes** tab.

1. Navigate to **Server Nodes > node link**.
2. Click on the node link (rather than expanding the triangle).
3. Expand the UPR QUEUES module triangle.
4. Hover over the UPR information for a description of the bucket analytics.
5. Expand the bucket analytics module triangle to show the information by server.



Related Links

Database Change Protocol on page 10

Database Change Protocol (DCP) is the protocol used to stream data changes to buckets.

DCP statistics (CLI)

Statistics for the Database Change Protocol (DCP) are obtained via the CLI with the `cbstats` tool.

Table 1: DCP statistics

Option	Description
dcp	Retrieves connections specific to statistics.
dcpagg	Retrieves statistics that are logically grouped and aggregated together by prefixes.
failovers	Retrieves vBucket failover logs.

Syntax

```
cbstats HOST:11210 dcp
cbstats HOST:11210 dcpagg
cbstats HOST:11210 failovers
```

Note: Specify the memcached port, 11210, when retrieving statistics.

cbstats dcp

```
 # cbstats 10.5.2.54:11210 dcp

 ep_dcp_count:
                           6
 ep_dcp_items_remaining:
                           0
 ep_dcp_items_sent:
                           0
 ep_dcp_producer_count:
                           3
 ep_dcp_queue_backfillremaining:
                           0
 ep_dcp_queue_fill:
                           0
 ep_dcp_total_bytes:
                           11264
 ep_dcp_total_queue:
                           0
 eq_uprq:replication:ns_1@10.5.2.117-
>ns_1@10.5.2.54:default:connected:                       true
 eq_uprq:replication:ns_1@10.5.2.117-
>ns_1@10.5.2.54:default:created:                         1812
 eq_uprq:replication:ns_1@10.5.2.117-
>ns_1@10.5.2.54:default:pending_disconnect:              false
 eq_uprq:replication:ns_1@10.5.2.117-
>ns_1@10.5.2.54:default:reserved:                        true
 eq_uprq:replication:ns_1@10.5.2.117-
>ns_1@10.5.2.54:default:stream_100_buffer_bytes:         0
 eq_uprq:replication:ns_1@10.5.2.117-
>ns_1@10.5.2.54:default:stream_100_buffer_items:         0
 eq_uprq:replication:ns_1@10.5.2.117-
>ns_1@10.5.2.54:default:stream_100_cur_snapshot_end:
 18446744073709551615
```

```
 eq_uprq:replication:ns_1@10.5.2.117-
>ns_1@10.5.2.54:default:stream_100_cur_snapshot_start:  0
 eq_uprq:replication:ns_1@10.5.2.117-
>ns_1@10.5.2.54:default:stream_100_cur_snapshot_type:   memory
 eq_uprq:replication:ns_1@10.5.2.117-
>ns_1@10.5.2.54:default:stream_100_end_seqno:
 18446744073709551615
 eq_uprq:replication:ns_1@10.5.2.117-
>ns_1@10.5.2.54:default:stream_100_flags:               0
 eq_uprq:replication:ns_1@10.5.2.117-
>ns_1@10.5.2.54:default:stream_100_items_ready:         false
 eq_uprq:replication:ns_1@10.5.2.117-
>ns_1@10.5.2.54:default:stream_100_last_received_seqno: 0
 eq_uprq:replication:ns_1@10.5.2.117-
>ns_1@10.5.2.54:default:stream_100_opaque:              74
 eq_uprq:replication:ns_1@10.5.2.117-
>ns_1@10.5.2.54:default:stream_100_snap_end_seqno:      0
 eq_uprq:replication:ns_1@10.5.2.117-
>ns_1@10.5.2.54:default:stream_100_snap_start_seqno:    0
 eq_uprq:replication:ns_1@10.5.2.117-
>ns_1@10.5.2.54:default:stream_100_start_seqno:         0
 eq_uprq:replication:ns_1@10.5.2.117-
>ns_1@10.5.2.54:default:stream_100_state:               reading
 eq_uprq:replication:ns_1@10.5.2.117-
>ns_1@10.5.2.54:default:stream_100_vb_uuid:
 164761896165868
 ...
```

cbstats dcpagg

Aggregate statistics are provided by the `dcpagg` parameter.

```
# cbstats 10.5.2.54:11210 dcpagg

 :total:count:                     3
 :total:items_remaining:           0
 :total:items_sent:                0
 :total:producer_count:            3
 :total:total_backlog_size:        0
 :total:total_bytes:               11264
 replication:count:                3
 replication:items_remaining:      0
 replication:items_sent:           0
 replication:producer_count:       3
 replication:total_backlog_size:   0
 replication:total_bytes:          11264
```

cbstats failovers

```
# cbstats 10.5.2.54:11210 failovers

 vb_1000:0:id:        101754288503529
 vb_1000:0:seq:       0
 vb_1000:num_entries: 1
 ...
```

Related Links

Database Change Protocol on page 10

Database Change Protocol (DCP) is the protocol used to stream data changes to buckets.

Shared thread pool

A shared thread pool is a collection of threads which are shared across multiple buckets.

A thread pool is a collection of threads used to perform similiar jobs. Each server node has a thread pool that is shared across multiple buckets. Shared thread pool optimizes dispatch tasks by decoupling buckets from thread allocation.

Threads are spawned at initial startup of a server node instance and are based on the number of CPU cores.

With the shared thread pool associated with each node, threads and buckets are decoupled. By decoupling threads from specific buckets, threads can run tasks for any bucket. Since the global thread pool allows for bucket priority levels, a separate I/O queue is available with the reader and writer workers at every priority level. This provides improved task queueing. For example, when a thread is assigned to running a task from an I/O queue and a second task is requested, another thread is assigned to pick up the second task.

Shared thread pool management promotes:

- Better parallelism for thread workers with more efficient I/O resource management.
- Better system scalability with more buckets being serviced with fewer worker threads.
- Availability of task priority if the disk bucket I/O priority setting is implemented.

The following circumstances describes how threads are scheduled to dispatch tasks:

- If all buckets have the same priority (default setting), each thread evenly round-robins over all the task queues of the buckets.
- If buckets have different priorities, the threads spend an appropriate fraction of time (scheduling frequency) dispatching tasks from queues of these bucket.
- If a bucket is being compacted, threads are not allocated to dispatch tasks for that bucket.
- If all buckets are either empty or being serviced by other threads, the thread goes to sleep.

Viewing thread status

The `cbstats raw workload` is used to view the status of the threads. The following is example code and results.

```
# cbstats 10.5.2.54:11210 -b default raw workload

 ep_workload:LowPrioQ_AuxIO:InQsize:   3
 ep_workload:LowPrioQ_AuxIO:OutQsize:  0
 ep_workload:LowPrioQ_NonIO:InQsize:   33
 ep_workload:LowPrioQ_NonIO:OutQsize:  0
 ep_workload:LowPrioQ_Reader:InQsize:  12
 ep_workload:LowPrioQ_Reader:OutQsize: 0
 ep_workload:LowPrioQ_Writer:InQsize:  15
 ep_workload:LowPrioQ_Writer:OutQsize: 0
 ep_workload:num_auxio:                1
 ep_workload:num_nonio:                1
 ep_workload:num_readers:              1
 ep_workload:num_shards:               4
 ep_workload:num_sleepers:             4
 ep_workload:num_writers:              1
 ep_workload:ready_tasks:              0
 ep_workload:shard0_locked:            false
 ep_workload:shard0_pendingTasks:      0
 ep_workload:shard1_locked:            false
 ep_workload:shard1_pendingTasks:      0
 ep_workload:shard2_locked:            false
 ep_workload:shard2_pendingTasks:      0
 ep_workload:shard3_locked:            false
 ep_workload:shard3_pendingTasks:      0
```

Impacted documentation

- Disk storage
- Multiple readers and writers
- Auxiliary I/O dispatcher
- Reader/writer threads

Related Links

Couchbase 3.0 Features on page 9

Related topics

Disk I/O priority on page 19
Disk I/O priority allows workload priorities to be set at the bucket level.

Couchbase Server statistics on page 179
Couchbase Server provides statistics at multiple levels throughout the cluster.

Caching layer on page 102
Couchbase Server includes a built-in caching layer which acts as a central part of the server and provides very rapid reads and writes of data.

Multiple readers and writers on page 103
Multi-threaded readers and writers provide multiple processes to simultaneously read and write data on disk. Simultaneous reads and writes increases disk speed and improves the read rate from disk.

Metadata ejection

Metadata ejection specify whether value-only or full metadata is ejected from memory.

Metadata ejection allows both value and full bucket ejection from memory. The following summarizes the behavior of value-only versus full bucket ejection: Value bucket ejection (the default) removes the data but keeps all keys and metadata in memory. When the value bucket ejection occurs, the item's value is reset. Full-bucket ejection removes all data including keys, metadata, and key-values, thus, reducing the RAM requirement for large buckets.

Full-bucket ejection supports very large data footprints (a large number of datasets or items/keys) since the working sets in memory are smaller. The smaller working sets allow efficient cache management and reduced warmup times. Metadata ejection is configured at the bucket-level.

For example, if your environment needs to store huge amounts of data (for example, tera or peta bytes), then you might want to enable metadata ejection on that bucket so that you are not limited by memory.

Impacted documentation

- Views and stored data > Document metadata
- Warm up operations
- Global thread pool
- Ejection and working set management
- Other ejection from hitting thresholds

Backward compatibility

In previous releases, the values for all active data sets are ejected and the metadata and key value are not ejection. In version 3.0, both value and metadata and key value are ejected.

Related Links

Couchbase 3.0 Features on page 9

Ejecting metadata (UI) on page 16
The type of bucket eviction from memory is set when creating or editing the bucket properties.

Ejecting metadata (CLI) on page 17
Bucket ejection from memory is set with the `couchbase-cli` tool.

Ejecting metadata (REST API) on page 18
Bucket metadata ejection from memory is set with `POST pools/default/buckets/default`.

Related topics

Buckets CLI on page 367
Buckets are managed with the `couchbase-cli` tool and the `bucket-*` commands.

Buckets REST API on page 456
The bucket management and configuration REST API endpoints provide a fine level of control over the individual buckets in the cluster, their configuration, and specific operations.

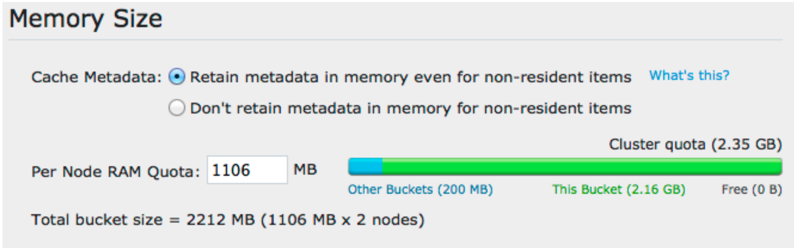
Ejecting metadata (UI)

The type of bucket eviction from memory is set when creating or editing the bucket properties.

To specify the bucket memory size:

1. Navigate to **Data Buckets > Create New Data Bucket** (new bucket) or **Data Buckets > bucket_name link > Edit** (existing bucket).
2. In the Memory Size panel, specify either to retain metadata in memory or not.

   Retaining metadata in memory needs more RAM, however, provides better performance for reads. Not retaining metadata in memory reduces RAM requirements.



3. Click **Save**.

Related Links

Metadata ejection on page 16
Metadata ejection specify whether value-only or full metadata is ejected from memory.

Ejecting metadata (CLI)

Bucket ejection from memory is set with the `couchbase-cli` tool.

Bucket ejection policy

Bucket ejection from memory is set with `couchbase-cli` tool, either `bucket-create` or `bucket-edit`, and the `--bucket-eviction-policy` parameter.

Policy for how to retain or eject metadata from memory:

```
--bucket-eviction-policy=[valueOnly|fullEviction]
```

Syntax

```
couchbase-cli bucket-create -c HOST:PORT
  --bucket=BUCKET_NAME \\
  --bucket-password=PASSWORD \\
  --bucket-eviction-policy=[VALUE]
  -u ADMIN -p PASSWORD


couchbase-cli bucket-edit -c HOST:PORT
  --bucket=BUCKET_NAME \\
  --bucket-eviction-policy=[VALUE]
    -u ADMIN -p PASSWORD
```

To create a new bucket and set metadata ejection to Value:

```
couchbase-cli bucket-create -c 192.168.0.1:8091 \\
  --bucket=test_bucket \\
  --bucket-password=password \\
  --bucket-ramsize=200 \\
  --bucket-eviction-policy=valueOnly \\
  --enable-flush=1 \\
  -u Administrator -p password
```

To edit a bucket and set metadata ejection to Full:

```
couchbase-cli bucket-edit -c 192.168.0.1:8091 \\
  --bucket=test_bucket \\
  --bucket-port=11222 \\
```

```
--bucket-ramsize=400 \\
--bucket-eviction-policy=fullEviction \\
--enable-flush=1 \\
-u Administrator -p password
```

Related Links

Metadata ejection on page 16
Metadata ejection specify whether value-only or full metadata is ejected from memory.

Ejecting metadata (REST API)

Bucket metadata ejection from memory is set with `POST pools/default/buckets/default`.

Metadata ejection REST API

The `evictionPolicy=[valueOnly|fullEviction]` setting specifies whether value-only or full eviction is configured.

Important: When editing bucket properties, be sure to specify all bucket properties. If a bucket property is not specified (whether or not you are changing the existing value), Couchbase Server may reset the property to the default. Even if you do not intend to change a certain property, re-specify the existing value to avoid this behavior.

Bucket REST API method and URI paths include:

HTTP method	URI path	Description
GET	/pools/default/buckets	Retrieves all bucket and bucket operations information from a cluster.
GET	/pools/default/buckets/default	Retrieves information for a single bucket associated with a cluster.
GET	/pools/default/buckets/bucket_name/stats	Retrieves bucket statistics for a specific bucket.
POST	/pools/default/buckets	Creates a new Couchbase bucket.
DELETE	/pools/default/buckets/bucket_name	Deletes a specific bucket.
POST	/pools/default/buckets/default/controller/doFlush	Flushes a specific bucket.

Related Links

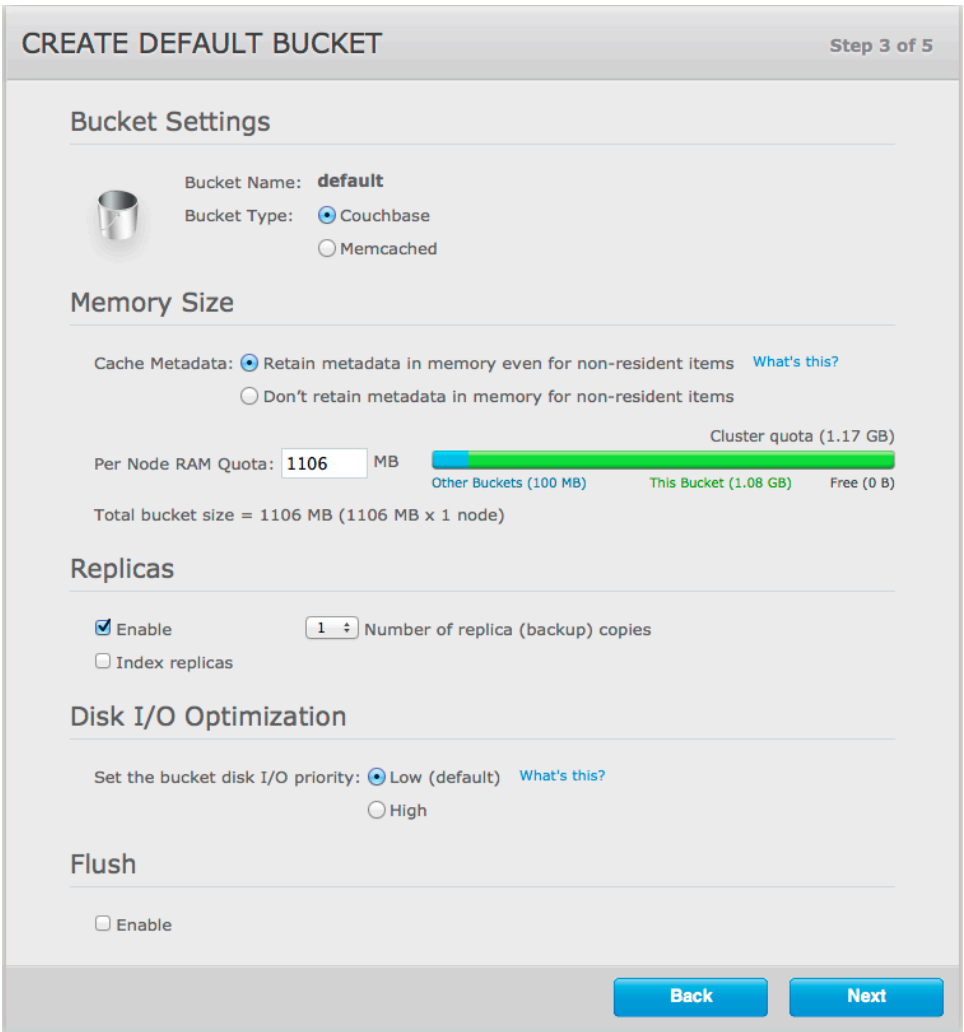
Metadata ejection on page 16
Metadata ejection specify whether value-only or full metadata is ejected from memory.

Disk I/O priority

Disk I/O priority allows workload priorities to be set at the bucket level.

Bucket priority settings can be specified at the bucket-level. The bucket disk I/O priority can be set as either high or low, whereas, low is the default. Bucket priority settings determine the number of threads assigned to a bucket allocated resources across buckets. Bucket latency and I/O operations are impacted by the setting value. When a bucket has a high priority, additional I/O resources are assigned for processing tasks.

The default buckets settings can be set during initial setup as well as be edited after setup.



Backward compatibility

When upgrading from a 2.x release to a 3.x release, Couchbase converts an existing thread value to either a low or a high priority based on the following:

- Buckets allocated six to eight (6-8) threads are high priority.
- Buckets allocated three to five (3-5) threads are low priority.

Related Links

Couchbase 3.0 Features on page 9

Setting disk I/O priority (UI) on page 20

The disk I/O priority for a bucket is set via the **Data Bucket** panel either when creating a data bucket or when editing a data bucket.

Setting disk I/O priority (CLI) on page 20
To set the disk I/O priority for a bucket, use the `couchbase-cli` tool `bucket-create` or `bucket-edit` command and the `--bucket-priority=[low|high]` option.

Setting disk I/O priority (REST API) on page 21
The disk I/O priority for a bucket is set with the `/pools/default/buckets/bucket_name` URI and the `threadNumber` setting.

Related topics

Shared thread pool on page 14
A shared thread pool is a collection of threads which are shared across multiple buckets.

Metadata ejection on page 16
Metadata ejection specify whether value-only or full metadata is ejected from memory.

Setting disk I/O priority (UI)

The disk I/O priority for a bucket is set via the **Data Bucket** panel either when creating a data bucket or when editing a data bucket.

To set disk I/O priority for a bucket:

1. Navigate to **Data Buckets > Create New Bucket** (new bucket) or **Data Buckets > bucket_name link > Edit** (existing bucket).
2. In the **Disk I/O Optimization** panel, set bucket disk I/O priority option. Specify either High or Low to set. Low is the default.
3. Click **Save**.

Related Links

Disk I/O priority on page 19
Disk I/O priority allows workload priorities to be set at the bucket level.

Setting disk I/O priority (CLI)

To set the disk I/O priority for a bucket, use the `couchbase-cli` tool `bucket-create` or `bucket-edit` command and the `--bucket-priority=[low|high]` option.

Creating a bucket and setting high priority

To create a new couchbase bucket with high priority:

```
couchbase-cli bucket-create -c 192.168.0.1:8091 \\
 --bucket=test_bucket \\
 --bucket-type=couchbase \\
 --bucket-port=11222 \\
 --bucket-ramsize=200 \\
 --bucket-replica=1 \\
 --bucket-priority=high \\
 -u Administrator -p password
```

Setting high priority

To modify a bucket to high priority:

```
couchbase-cli bucket-edit -c 192.168.0.1:8091 \\
 --bucket=test_bucket \\
 --bucket-priority=high \\
```

```
-u Administrator -p password
```

Related Links

Disk I/O priority on page 19
Disk I/O priority allows workload priorities to be set at the bucket level.

Setting disk I/O priority (REST API)

The disk I/O priority for a bucket is set with the `/pools/default/buckets/bucket_name` URI and the `threadNumber` setting.

Disk I/O priority REST API

To set the maximum of thread workers, use the `threadsNumber` option. To specify high priority, assign eight (8) threads. To specify low priority, assign three (3) threads.

Bucket REST API method and URI paths include:

HTTP method	URI path	Description
GET	/pools/default/buckets	Retrieves all bucket and bucket operations information from a cluster.
GET	/pools/default/buckets/default	Retrieves information for a single bucket associated with a cluser.
GET	/pools/default/buckets/bucket_name/stats	Retrieves bucket statistics for a specific bucket.
POST	/pools/default/buckets	Creates a new Couchbase bucket.
DELETE	/pools/default/buckets/bucket_name	Deletes a specific bucket.
POST	/pools/default/buckets/default/controller/doFlush	Flushes a specific bucket.

Important: When editing bucket properties, be sure to specify all bucket properties. If a bucket property is not specified (whether or not you are changing the existing value), Couchbase Server may reset the property to the default. Even if you do not intend to change a certain property, re-specify the existing value to avoid this behavior.

```
curl -v -X POST -u admin:password -d name=customer \
   -d flushEnabled=0 -d replicaNumber=1 -d authType=none \
   -d ramQuotaMB=200 -d proxyPort=11212 \
   http://localhost:8091/pools/default/buckets/bucket_name
```

Related Links

Disk I/O priority on page 19
Disk I/O priority allows workload priorities to be set at the bucket level.

Graceful failover

Graceful failover waits until data persists to disk and replicas are up-to-date before failing over the server node.

With graceful failover, failed over nodes can be safely removed from the cluster and quickly added back after they are repaired, catching up incrementally from where they left off. When adding a node to a cluster, the option is to specify either `delta` or `full`. By default, when **Failover** is selected from the Web Console, a graceful failover is performed.

In a graceful failover, a new replica is created on another node in the cluster, all of the data on a node that needs to be failed over is transferred to the replica, and then the node is removed from the cluster. This means that there is no data loss window. Once a new node is added to the cluster, the rebalance operation moves only the removed node's data to the newly added node.

However, a hard failover option is available in case the node is in a bad state. Hard failover immediately removes a node from the cluster which might lead to data loss. Auto-failover behavior is a hard failover.

Failing over nodes and rebalancing

- When a node is added back to the cluster after a graceful failover has been performed, the new node receives the vBuckets that existed on the old node. In this case, rebalancing is optimized.
- Topology changes after a graceful failover causes a full rebalance in when a node is added back. In this case, rebalancing is not optimized.
- A hard failover after a graceful failover causes a full rebalance in when a node is added back. In this case, rebalancing is not optimized.

Impacted documentation

- Failover
- Rebalance
- Swap rebalance

Related Links

Couchbase 3.0 Features on page 9

Failing over gracefully (UI) on page 22

Failing over gracefully (CLI) on page 23
By default, graceful failover is implemented with failing over a node from a cluster using `couchbase-cli failover`.

Failing over gracefully (REST API) on page 24
Graceful failover is set through the `POST /controller/startGracefulFailover` HTTP method and URI.

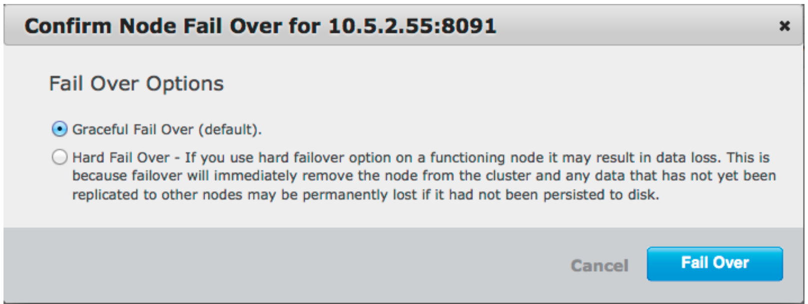
Related topics

Delta node recovery on page 25
Delta node recovery provides the option to specify whether the replacement server node's disk is initialized or whether the data and data buckets are retained.

Failing over gracefully (UI)

To manage failover:

1. Navigate to **Server Nodes**.
2. For the server node, select **Fail Over**.
3. Select **Graceful Failover** to gracefully failover. Alternatively, select **Hard Fail Over** to force the failover.



4. Click **Fail Over**.

Related Links

Graceful failover on page 22
Graceful failover waits until data persists to disk and replicas are up-to-date before failing over the server node.

Failing over gracefully (CLI)

By default, graceful failover is implemented with failing over a node from a cluster using `couchbase-cli failover`.

Graceful failover is implemented as the default when `couchbase-cli failover` is used without the `--force` option . The `--force` option fails over the node from the cluster immediately.

Note: Be sure to update any scripts that implement failover.

Syntax

```
couchbase-cli failover
    --cluster=HOST:PORT
    --server-failover=HOST:PORT
    --user=ADMIN
    --password=PASSWORD
```

Setting failover, readd, recovery, and rebalance operations

The following example shows a failover, readd, recovery and rebalance sequence operations, where as, a node in a cluster is gracefully failed over, the node is re-added to the cluster, a delta recovery is implemented for the node, and rebalance is performed on the cluster:

```
couchbase-cli failover -c 192.168.0.1:8091 \\
    --server-failover=192.168.0.2 \\
    -u Administrator -p password

couchbase-cli server-readd -c 192.168.0.1:8091 \\
    --server-add=192.168.0.2 \\
    -u Administrator -p password

couchbase-cli recovery -c 192.168.0.1:8091 \\
    --server-recovery=192.168.0.2 \\
    --recovery-type=delta \\
    -u Administrator -p password

couchbase-cli rebalance -c 192.168.0.1:8091 \\
    -u Administrator -p password
```

Failing over a node immediately

The following example shows a node failing over immediately, that is, graceful failover is not implemented.

```
couchbase-cli failover -c 192.168.0.1:8091 \\
    --server-failover=192.168.0.2 \\
    --force  \\
    -u Administrator -p password
```

Related Links

Graceful failover on page 22
Graceful failover waits until data persists to disk and replicas are up-to-date before failing over the server node.

Failing over gracefully (REST API)

Graceful failover is set through the `POST /controller/startGracefulFailover` HTTP method and URI.

Graceful failover REST API

Performing a POST on the `/controller/startGracefulFailover` initiates graceful failover. It's progress can be tracked just rebalance. After the data is persisted to disk and replicas are up-to-date, the node is put into the failed over state.

HTTP method	URI path	Description
POST	/controller/startGracefulFailover	Sets graceful failover.

```
POST /controller/startGracefulFailover
    otpNode=[node_name]
```

Note: Be sure to update any scripts that implement failover.

Related Links

Graceful failover on page 22
Graceful failover waits until data persists to disk and replicas are up-to-date before failing over the server node.

Delta node recovery

Delta node recovery provides the option to specify whether the replacement server node's disk is initialized or whether the data and data buckets are retained.

When re-adding a server node to the cluster, a new server node is initialized and re-populated, that is, the disk is wiped out (a warning is generated) and then re-populated with the original data buckets. This operation ensures a clean disk, but also has overhead in terms of resource use and downtime due to disk re-population. With delta node recovery the same server node is not initialized nor re-populated. The data and data buckets are retained so that the cluster can start functioning without downtime. This operation improves recovery time and network resource usage. For example, an environment with a large data footprint might re-add the server node as the same.

For a new server node, the disk is initialized (data is removed) and re-populated. For the same server node, the disk is not initialized or re-populated; the data buckets remain.

Server nodes are removed from clusters under many circumstances. Depending on the circumstance, the server node is added back to the cluster as either a new or the same server. The following are circumstances (among many others) where the server node might be re-added as the same server.

- Node goes down for a short period of time
- Routine maintenance is scheduled
- Network connectivity is briefly disrupted

Impacted documentation

- Failover
- Re-add a server

Related Links

Couchbase 3.0 Features on page 9

Setting delta node recovery (UI) on page 25

Setting delta node recovery (CLI) on page 26
To re-add a server node to a cluster, use the `couchbase-cli` tool and the `recovery` command with the `--recovery-type` parameter.

Setting delta node recovery (REST API) on page 27
The REST request to add a node to a cluster uses the `/controller/addNode` endpoint URI.

Related topics

Graceful failover on page 22
Graceful failover waits until data persists to disk and replicas are up-to-date before failing over the server node.

Server nodes CLI on page 371

Server nodes REST API on page 444
A Couchbase Server instance, known as node, is a physical or virtual machine running Couchbase Server. Each node is as a member of a cluster.

Setting delta node recovery (UI)

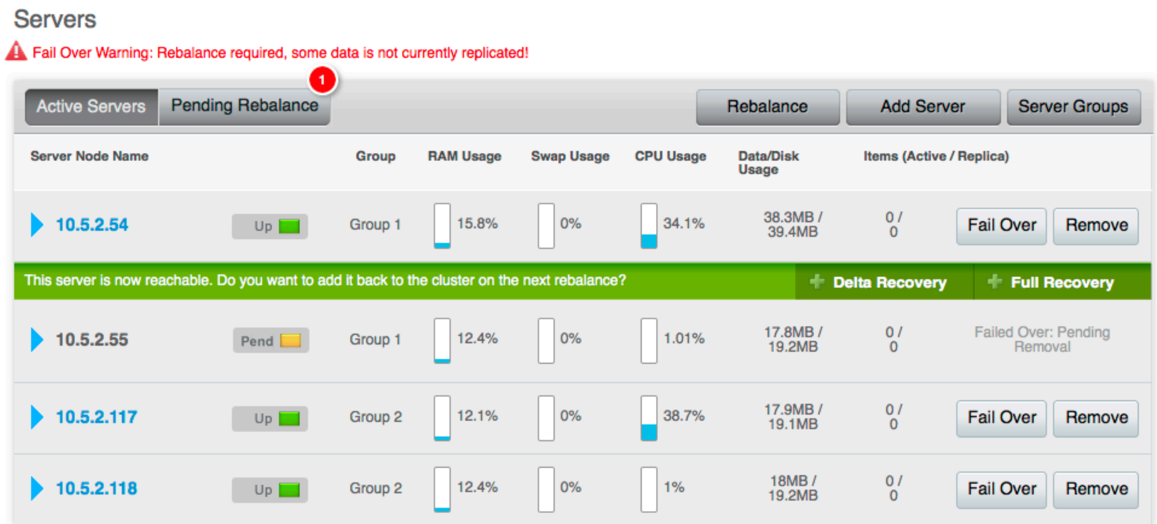
The process for re-adding a server isomg tje delta node recovery method involves:

1. Fail over the node using either the graceful or hard failover method. Graceful failover is recommended.
2. After the node is failed over, specify whether delta or full recovery is to be used.
3. During the next rebalance, the node is added back to the cluster using the specified recovery method.

To use delta recovery during failover, recovery, and rebalance operations:

1. Navigate to **Server Nodes**.
2. Click **Failover**.

3. Select **Graceful Failover**.
4. Click **Delta Recovery**.



5. Click **Rebalance**.

   The failed server node is re-added to the cluster.

Related Links

Delta node recovery on page 25
Delta node recovery provides the option to specify whether the replacement server node's disk is initialized or whether the data and data buckets are retained.

Setting delta node recovery (CLI)

To re-add a server node to a cluster, use the `couchbase-cli` tool and the `recovery` command with the `--recovery-type` parameter.

Recovery options are either `full` or `delta`.

Table 2: Recovery type

Option	Description
--server-recovery=HOST[:PORT]	Server to recover
--recovery-type=TYPE[delta|full]	Type of recovery to be performed for the node

Syntax

```
couchbase-cli failover
    --cluster=HOST:PORT
    --server-recovery=HOST:PORT
    --recovery-type=[delta|full]
    --user=ADMIN
    --password=PASSWORD
```

Setting recovery type

To set incremental node recovery type for a server:

```
couchbase-cli recovery -c 192.168.0.1:8091 \\
    --server-recovery=192.168.0.2          \\
    --recovery-type=delta                   \\
    -u Administrator -p password
```

Setting failover, readd, recovery, and rebalance operations

The following example shows a failover, readd, recovery and rebalance sequence operations, where as, a node in a cluster is gracefully failed over, the node is re-added to the cluster, a delta recovery is implemented for the node, and rebalance is performed on the cluster:

```
couchbase-cli failover -c 192.168.0.1:8091 \\
    --server-failover=192.168.0.2 \\
    -u Administrator -p password

couchbase-cli server-readd -c 192.168.0.1:8091 \\
    --server-add=192.168.0.2 \\
    -u Administrator -p password

couchbase-cli recovery -c 192.168.0.1:8091 \\
    --server-recovery=192.168.0.2 \\
    --recovery-type=delta \\
    -u Administrator -p password

couchbase-cli rebalance -c 192.168.0.1:8091 \\
    -u Administrator -p password
```

Related Links

Delta node recovery on page 25
Delta node recovery provides the option to specify whether the replacement server node's disk is initialized or whether the data and data buckets are retained.

Setting delta node recovery (REST API)

The REST request to add a node to a cluster uses the `/controller/addNode` endpoint URI.

Delta node recovery REST API

Performing a POST on the `/controller/setRecoveryType` sets the type of recovery. It's progress can be tracked just rebalance. After the data is persisted to disk and replicas are up-to-date, the node is put into the failed over state.

HTTP method	URI path	Description
POST	/controller/setRecoveryType	Sets the recovery type to be performed for a node. Options are full or delta.

```
POST /controller/setRecoveryType
    otpNode=[node_name]
    recoveryType=[full|delta]
```

Error codes

```
200       // Request succeeded
400       // recoveryType and/or otpNode could not be understood by the server
404       // The cluster is running in a pre-3.0 compatibility mode and thus
 cannot satisfy the request
```

Related Links

Delta node recovery on page 25

Delta node recovery provides the option to specify whether the replacement server node's disk is initialized or whether the data and data buckets are retained.

Incremental backup and restore

An incremental backup or restore is a backup (or restore) of the latest changes since the last backup (any level).

Incremental backup and restore allow administrators to backup and restore only modified part of the database instead of taking a full data backup every time. This makes backup more efficient for larger datasets and reduces the risk of data loss.

The purpose of an incremental backup is to preserve and protect data by creating copies that are based on the differences in those data and thus minimize the amount of time needed to perform the backup. With incremental backups, successive copies of the data contain only that portion which has changed since the preceding backup copy was made. Incremental backups are faster than full backups and reduce network utilization.

The following are the types of backups:

Full backup

A snapshot of the state of the database at a particular point-in-time on disk. A snapshot is a read-only copy of the dataset. The restoration process takes the last full backup plus all the incremental backups until the restoration point-in-time is reached.

Differential backup

An incremental backup that contains all data changes after the most recent incremental backup. Differential backups are deltas between the following:

- Last full backup and the first snapshot
- Last incremental snapshot the current snapshot

Accummulative backup

An incremental backup that contains all data changes after the last full backup. A snapshot of the data changes between the last full backup and the current snapshot.

Impacted documentation

- cbbackup
- cbrestore

Backward compatibility

- 2.0+

Related Links

Couchbase 3.0 Features on page 9

Setting incremental backup (CLI) on page 29
Backup operations are performed by using the `cbbackup` tool.

Setting incremental restore (CLI) on page 30
Restore operations are performed by the `cbbackup` tool. Incremental restore provides partial restore functionality.

Setting incremental backup (CLI)

Backup operations are performed by using the `cbbackup` tool.

cbbackup

The `cbbackup` tool provides provides differential, accumulative, or full backup.

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

```
cbbackup http://HOST:8091 /backups/backup-1
cbbackup couchbase://HOST:8091 /backups/backup-1
```

Table 3: cbbackup options

Option	Mode	Description
--mode [full | incr-diff | incr-accu | auto]		Backup running mode: full, incr-diff, incr-accu, auto
	full	Forces a full backup
	incr-diff	Differential incremental backup. Backs up the delta change from the last full or incremental backup
	incr-accu	Accumulative incremental backup. Backs up the delta change from the full backup.

Note: Existing `cbbackup` options are supported. For example, to backup a single bucket, use the `-b` option or to increase verbose level, use `-v`

Related Links

Incremental backup and restore on page 29
An incremental backup or restore is a backup (or restore) of the latest changes since the last backup (any level).

Setting incremental restore (CLI)

Restore operations are performed by the `cbbackup` tool. Incremental restore provides partial restore functionality.

cbrestore

Data from a specific period of time is restored. The `cbrestore` tool browses through the subdirectories and files in the backup directory and finds all related Couchbase backup files to restore.

```
cbrestore /backups/backup-1 http://HOST:8091
```

Table 4: cbrestore options

Option	Description
--from-date=[yyyy-mm-dd]	Restores the data from this date. The date is specified as year-month-day. Data collected before this date is not restored.
--to-date=[yyyy-mm-dd]	Restores data until this date. The date is specified as year-month-day. Data collected after this date is not restored.

Related Links

Incremental backup and restore on page 29
An incremental backup or restore is a backup (or restore) of the latest changes since the last backup (any level).

Stream-based views

Indexing is performed as soon as data changes in memory without waiting for persistence to disk. As a result, views have up-to-date data and faster queries.

Indexing occurs as soon as data changes in memory due to the Database Change Protocol (DCP), a stream-based protocol. With DCP, changes made to documents in-memory are immediately streamed to be indexed without being written to disk. This provides faster view consistency which provides fresher data.

- Views are updated with key-value sooner
- Lag time for view consistency is reduced
- Views are more closely synch with the data

> **Note:** When a Couchbase Server cluster has both 2.x and 3.0 servers, the Couchbase 3.0 server has faster indexing.

When the `stale=false` query is run, the indexer picks up changes that occurred (latest data) after the query was issued. The changes are picked up from memory and not disk. The index occurs in the following manner:

1. A set request comes in from the application.
2. Couchbase Server responds back that they key is written.
3. Couchbase Server quickly replicates the data out to memory on the other nodes.
4. At the same time, data is put into a write queue to be persisted to disk.

Impacted documentation

- Views and view operations

Related Links

Couchbase 3.0 Features on page 9

Related topics

Views operation on page 184

Pause and resume replication

During XDCR replication, the process can be paused and resumed. Performed via the Couchbase command-line interface (CLI).

XDCR streams between the source and destination cluster can be paused and later resumed. After XDCR is resumed, data continues to replicate between the source and destination clusters starting from where it previously left off.

CLI

XDCR pause and resume is implemented with `couchbase-cli xdcr-replicate`.

Table 5: xdcr-replicate options

Option	Description
--create	Create and start a new replication
--delete	Stop and cancel a replication
--list	List all XDCR replications
--pause	Pause the replication
--resume	Resume the replication
--settings	Update settings for the replication
--xdcr-replicator=REPLICATOR	Replication ID
--xdcr-from-bucket=BUCKET	Local bucket name to replicate from
--xdcr-cluster-name=CLUSTERNAME	Remote cluster to replicate to
--xdcr-to-bucket=BUCKETNAME	Remote bucket to replicate to
--max-concurrent - reps =[32]	Maximum concurrent replications per bucket, 8 to 256
--checkpoint-interval =[1800]	Intervals between checkpoints , 60 to 14400 seconds
--worker-batch-size =[500]	Document batch size, 500 to 10000
--doc-batch size =[2048] KB	Document batch size, 10 to 100000 KB
--failure-restart-interval =[30]	Interval for restarting failed xdcr, 1 to 300 seconds
--optimistic-replication-threshold =[256]	Document body size threshold (bytes) to trigger optimistic replication
--xdcr-replication-mode=[xmem|capi]	Replication protocol, either capi or xmem

CLI Syntax

```
couchbase-cli xdcr-replicate -c HOST:PORT
    --pause
    --xdcr-replicator=[REPLICATOR_ID]
    -u ADMIN -p PASSWORD

couchbase-cli xdcr-replicate -c HOST:PORT
  --resume
  --xdcr-replicator=[REPLICATOR_ID]
  -u ADMIN -p PASSWORD
```

Pausing a running replication stream

```
couchbase-cli xdcr-replicate -c 192.168.0.1:8091 \\
    --pause \\
    --xdcr-replicator=f4eb540d74c43fd3ac6d4b7910c8c92f/default/
default \\
    -u Administrator -p password
```

Resuming a paused replication stream

```
couchbase-cli xdcr-replicate -c 192.168.0.1:8091 \\
    --resume \\
    --xdcr-replicator=f4eb540d74c43fd3ac6d4b7910c8c92f/default/
default \\
    -u Administrator -p password
```

Updating settings for a replication stream

```
couchbase-cli xdcr-replicate -c 192.168.0.1:8091 \\
    --settings \\
    --xdcr-replicator=f4eb540d74c43fd3ac6d4b7910c8c92f/default/
default \\
    --max-concurrent-reps=32 \\
    --checkpoint-interval=1800 \\
    --worker-batch-size=500 \\
    --doc-batch-size=2048 \\
    --failure-restart-interval=30 \\
    --optimistic-replication-threshold=256 \\
    -u Administrator -p password
```

REST API

The setting that pauses and resumes replication is the `pauseRequested` setting. The setting values are true or false. Specify true to pause replication. Specify false to unpaused replication.

Table 6: XDCR URI paths for settings

URI path	Description
/settings/replications/	Global setting supplied to all replications for a cluster.
/settings/replications/[replication_id]	Settings for specific replication for a bucket.

Table 7: XDCR internal settings

Parameter	Value	Description
`pauseRequested`	True or False. Default: False	Specify true to pause the replication. Specify false to un-pause the replication.

Retrieve replication settings example

The following example retrieves the current settings for a replication.

```
# curl -X GET -u Administrator:password http://10.5.2.54:8091/
settings/replications

{
    "maxConcurrentReps":16,
    "checkpointInterval":1800,
    "docBatchSizeKb":2048,
    "failureRestartInterval":30,
    "workerBatchSize":500,
    "connectionTimeout":180,
    "workerProcesses":4,
    "httpConnections":20,
    "retriesPerRequest":2,
    "optimisticReplicationThreshold":256,
    "socketOptions":{"keepalive":true,"nodelay":false},
    "pauseRequested":false,
    "supervisorMaxR":25,
    "supervisorMaxT":5
}
```

Pause replication example

```
# curl -X POST -u Administrator:password http://10.5.2.54:8091/
settings/replications -d pauseRequested=true
```

Related Links

Couchbase 3.0 Features on page 9

Stream-based XDCR

Stream-based XDCR collects data changes from memory on the source cluster and streams the data changes directly to memory on the destination cluster.

Stream-based XDCR replication is available due to the Database Change Protocol (DCP), a stream-based protocol. Once the data changes are detected and streamed to the destination cluster's memory, each cluster persists the data to disk. On the source cluster, the data changes (in memory) are queued and then persisted to disk. Correspondingly, on the destination cluster, the data changes (stream to memory) are queued and then persisted to disk.

Stream-based XDCR replication provides:

- Lower latency, that is, the time gap between data replication
- High availability and disaster recovery
- Improves recovery point objective (RPO)
- Smaller data loss window

Backward compatibility

- Changes are made automatically through the upgrade.
- Only the source cluster has to be upgraded. The destination cluster accepts the data changes into memory.

Related Links

Couchbase 3.0 Features on page 9

Encrypted admin access

Encrypted administrator access provides encrypted REST API access using Secure Socket Layer (SSL) authentication.

The encrypted communication allows a connection to be configured for security using Secure Sockets Layer (SSL) encryption. Encryption protects data-in-flight from a remote machines to a Couchbase cluster using SSL. A secure channel is established between the remote machine and the server.

The server generates a self-signed certification for the initial node which is propagated throughout the server nodes in the cluster. If the self-signed certificate is regenerated or updated as part of setting up XDCR data encryption (or for any other reason), the certificate retrieved by the client must be obtained again before secure client-server communication is re-established. The secure connection is on the cluster-level (rather than bucket-level) and is through the dedicated HTTPS REST port, 18091 or the HTTPS CAPI port, 18092.

Encrypted administrator access is used under a variety of situations:

- An administrator is physically located in a different data center.
- An administrator is outside of the firewall.
- An additional level of security is required.

Secure administrative access is configured using the same method used for encrypted client-server communication:

1. Connect to the server through an unencrypted port, 8091 REST HTTP or 8092 CAPI HTTP, and using the administratrator username and password.
2. Retrieve the SSL self-signed certificate and store in the local file, `remoteCert.pem`.
3. Create the cluster remote reference using the an encrypted port, 18091 REST HTTP or 18092 CAPI HTTP, and enabling data encryption.

Secure client-server REST API

The following summarizes the HTTP methods used for defining data encryption:

Table 8: URI paths for setting secure communication

HTTP method	URI path	Destription
GET	/pools/default/certificate	Retrieves the SSL self-signed certificate from the remote cluster.
GET	/pools/default/remoteClusters	Retrieves the remote cluster reference.
POST	/pools/default/remoteClusters	Creates the remote cluster referece.
PUT	/pools/default/remoteClusters	Modifies the remote cluster reference.

Syntax for retrieving the remote certificate

```
curl -X GET  -u adminName:adminPassword
    http://localHost:Port/pools/default/certificate
```

Retrieving and storing the certificate

```
curl http://10.5.2.54:8091/pools/default/certificate > ./
remoteCert.pem
```

SSL certificate

```
-----BEGIN CERTIFICATE-----
```

```
MIIC/jCCAeigAwIBAgIIE3jc9BofgigwCwYJKoZIhvcNAQEFMCQxIjAgBgNVBAMT
GUNvdWNoYmFzZSBTZXJ2ZXIgOTRmYTE3YTUwHhcNMTMwMTAxMDAwMDAwWhcNNDkx
MjMxMjM1OTU5WjAkMSIwIAYDVQQDExlDb3VjaGJhc2UgU2VydmVyIDk0ZmExN2E1
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAxaaXsKm06xxzzYqejDAO
3qW1x6vLz9jcLdZkNQgxGk4+/ulrfK4PSLHARf4vml8Ev3bcOzCwfyDCp2/TCSX0
qDTn4iBRp9CJtxVyY/xqWkYkld+GGtj28P0CtZ1UKOHCRB7KInzxesxITg/a0vsL
M8GrcwFpmZEJjeY7HGdUuBRcoMfm2Yn28drmr92SNSsz+npdfEFkQloYStqemOOG
h1Jn7ldU5rBj/B2zcvh6guDXKKz/bMMeCTX84BmkG3rmiKQwxyizuxtYi5u1BthC
X3aO58lC9uRMja1lA5TrJnZOCRT24G6VTh2bYhN98W6YmvF9l4ESDR4I7nE8E6Gt
eQIDAQABozgwNjAOBgNVHQ8BAf8EBAMCAKQwEwYDVR0lBAwwCgYIKwYBBQUHAwEw
DwYDVR0TAQH/BAUwAwEB/zALBgkqhkiG9w0BAQUDggEBAF0Bz2MpQoBEdOdDRix3
j0/XGKjH7kI5zDFiOlUvANMeErVZf9kM8xqS7Yd3bCa2rjT1Y8BM3Sciurtrd/Cy
iTVzpXjQOR/K1AFtiBtuNb2Hx5SXvgeW4p4uNmK74u1UUNmAyb3mwSQ+duuqK/Ef
D4wTolPTZP5gcricyWI3qUCi3pTeCz/2jcAWn3DI4KVtlAsOy9sFFo4RxBDgmOuS
klUAb8eu4e2XxcLJ++geYoum0VIKa3ygjpZ800PupwZZetjD8/6tfbYFuoBTXL+r
27M9ArsOxkVbh3fDQ8b8qnr5sam1P7IfSzqq/Lq4vjh1mvred62zuJlMvY9KmNJU
rqw=
-----END CERTIFICATE-----
```


Syntax for creating an encrypted client-server connection

A POST to `/pools/default/remoteClusters` creates the remote cluster reference. Setting the `demandEncryption` to one (1) and providing the certificate name and location enables data encryption.

```
curl -X POST  -u Admin:myPassword
    https://localHost:sslPort/pools/default/remoteClusters
    -d demandEncryption=[0|1] --data-urlencode "certificate=$(cat
 remoteCert.pem)"
```

Creating an encrypted client-server connection

```
curl -X POST
    https://Administrator:asdasd@192.168.0.1:18091/pools/default/
remoteClusters/
    -d demandEncryption=1 --data-urlencode "certificate=$(cat
 remoteCert.pem)"
```

Related Links

Couchbase 3.0 Features on page 9

Encrypted data access

Encrypted data access is a communication connection between the client and Couchbase Server cluster using Secure Sockets Layer (SSL).

The encrypted data access allows client connections to be configured for security using Secure Sockets Layer (SSL) encryption. Client-to-Server encryption protects data-in-flight from client machines to a Couchbase cluster using SSL. A secure channel is established between the client and the server.

The server generates a self-signed certification for the initial node which is propagated throughout the server nodes in the cluster. If the self-signed certificate is regenerated or updated as part of setting up XDCR data encryption (or for any other reason), the certificate retrieved by the client must be obtained again before secure client-server communication is re-established. The secure connection is on the cluster-level (rather than bucket-level) and is through the dedicated HTTPS REST port, 18091 or the HTTPS CAPI port, 18092.

The client sets up SSL communication with a Couchbase server cluster in the following manner:

1. Connect to the server through an unencrypted port, 8091 REST HTTP or 8092 CAPI HTTP, and using the administrator username and password.
2. Retrieve the SSL self-signed certificate and store in the client local file.
3. Create the cluster destination reference using the an encrypted port, 18091 REST HTTP or 18092 CAPI HTTP, and enabling data encryption.

Secure client-server REST API

The following summarizes the HTTP methods used for defining data encryption:

Table 9: URI paths for setting secure communication

HTTP method	URI path	Destription
GET	/pools/default/certificate	Retrieves the SSL self-signed certificate from the remote cluster.
GET	/pools/default/remoteClusters	Retrieves the remote cluster reference.
POST	/pools/default/remoteClusters	Creates the remote cluster referece.
PUT	/pools/default/remoteClusters	Modifies the remote cluster reference.

Syntax for retrieving the remote certificate

```
curl -X GET  -u adminName:adminPassword
    http://localHost:Port/pools/default/certificate
```

Retrieving and storing the certificate

```
curl http://10.5.2.54:8091/pools/default/certificate > ./
remoteCert.pem
```

SSL certificate

```
-----BEGIN CERTIFICATE-----
MIIC/jCCAeigAwIBAgIIE3jc9BofgigwCwYJKoZIhvcNAQEFMCQxIjAgBgNVBAMT
GUNvdWNoYmFzZSBTZXJ2ZXIgOTRmYTE3YTUwHhcNMTMwMTAxMDAwMDAwWhcNNDkx
MjMxMjM1OTU5WjAkMSIwIAYDVQQDExlDb3VjaGJhc2UgU2VydmVyIDk0ZmExN2E1
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAxaaXsKm06xxzzYqejDAO
3qW1x6vLz9jcLdZkNQgxGk4+/ulrfK4PSLHARf4vml8Ev3bcOzCwfyDCp2/TCSX0
qDTn4iBRp9CJtxVyY/xqWkYkld+GGtj28P0CtZ1UKOHCRB7KInzxesxITg/a0vsL
```

```
M8GrcwFpmZEJjeY7HGdUuBRcoMfm2Yn28drmr92SNSsz+npdfEFkQloYStqemOOG
h1Jn7ldU5rBj/B2zcvh6guDXKKz/bMMeCTX84BmkG3rmiKQwxyizuxtYi5u1BthC
X3aO58lC9uRMja1lA5TrJnZOCRT24G6VTh2bYhN98W6YmvF9l4ESDR4I7nE8E6Gt
eQIDAQABozgwNjAOBgNVHQ8BAf8EBAMCAKQwEwYDVR0lBAwwCgYIKwYBBQUHAwEw
DwYDVR0TAQH/BAUwAwEB/zALBgkqhkiG9w0BAQUDggEBAF0Bz2MpQoBEdOdDRix3
j0/XGKjH7kI5zDFiOlUvANMeErVZf9kM8xqS7Yd3bCa2rjT1Y8BM3Sciurtrd/Cy
iTVzpXjQOR/K1AFtiBtuNb2Hx5SXvgeW4p4uNmK74u1UUNmAyb3mwSQ+duuqK/Ef
D4wTolPTZP5gcricyWI3qUCi3pTeCz/2jcAWn3DI4KVtlAsOy9sFFo4RxBDgmOuS
klUAb8eu4e2XxcLJ++geYoum0VIKa3ygjpZ800PupwZZetjD8/6tfbYFuoBTXL+r
27M9ArsOxkVbh3fDQ8b8qnr5sam1P7IfSzqq/Lq4vjh1mvred62zuJlMvY9KmNJU
rqw=
-----END CERTIFICATE-----
```


Syntax for creating an encrypted client-server connection

A POST to `/pools/default/remoteClusters` creates the remote cluster reference. Setting the `demandEncryption` to one (1) and providing the certificate name and location enables data encryption.

```
curl -X POST  -u Admin:myPassword
    https://localHost:sslPort/pools/default/remoteClusters
    -d demandEncryption=[0|1] --data-urlencode "certificate=$(cat
 remoteCert.pem)"
```

Creating an encrypted client-server connection

```
curl -X POST
    https://Administrator:asdasd@192.168.0.1:18091/pools/default/
remoteClusters/
    -d demandEncryption=1 --data-urlencode "certificate=$(cat
 remoteCert.pem)"
```

Related Links

Couchbase 3.0 Features on page 9

Configuring clients for SSL

The Couchbase Server client libraries support client-side encryption using the Secure Sockets Layer (SSL) protocol.To enable SSL on the client-side, you need to get an SSL certificate from Couchbase Server, as described in *Encrypted data access* on page 38, and then follow the steps specific to the client you are using.

The following clients support SSL:

- Java
- .NET
- Node.js
- PHP
- C

Configuring the Java client for SSL

To configure SSL for the Java client:

1. Get the SSL certificate from Couchbase Server.
2. Store the certificate in your Java virtual machine (JVM) keystore.

   ```
   $ keytool -import -file certificate-file-name
   ```
3. Enable SSL by modifying the `reference.conf` configuration file.

   Set `sslEnabled` to `true`, specify the path to the keystore, and provide the keystore password, as shown in the following example:

   ```
   com.couchbase.client.bootstrap {
       sslEnabled = true
       sslKeystoreFile = /path/to/keystorefile
       sslKeystorePassword = ""
   }
   ```

Configuring the .NET client for SSL

To configure SSL for the .NET client:

1. Get the SSL certificate from Couchbase Server.
2. Store the certificate in the Microsoft Windows Trusted Root Certification Authorities certificate store.
3. Enable SSL support on the client.

   You can enable SSL at either the cluster level or on a per-Bucket basis. Setting SSL support at the cluster level means that all buckets within the cluster will use SSL. If you want to use SSL only on specific buckets, just configure your individual bucket to use SSL. Here's some examples:

   ```
   var config = new ClientConfiguration()
   {
      UseSsl = true
   };
   var cluster = new CouchbaseCluster(config);
   using (var bucket = cluster.OpenBucket())
   {
    //all buckets opened with this configuration will use SSL
   }

   var config = new ClientConfiguration
   {
    BucketConfigs = new Dictionary<string, BucketConfiguration>
    {
     {"customers", new BucketConfiguration
      {
      UseSsl = true
   ```

```
   }}
  }
 };
var cluster = new CouchbaseCluster(config);
using (var bucket = cluster.OpenBucket("customers"))
{
 //only the customers bucket will use SSL
}
```

Configuring the Node.js client for SSL

To configure SSL for Node.js:

1. Get the SSL certificate from Couchbase Server.
2. Store the certificate in the appropriate certificate storage area for your environment.
3. Pass an SSL scheme with your connection string when creating your cluster object, as shown in the following example:

```
var couchbase = require('couchbase');
var myCluster = new couchbase.Cluster('couchbases://10.1.1.1');
var myBucket = myCluster.openBucket();
```

Configuring the PHP client for SSL

To configure SSL for PHP:

1. Get the SSL certificate from Couchbase Server.
2. Store the certificate in the appropriate certificate storage area for your environment.
3. TBD

Configuring the C client for SSL

To configure SSL for C:

1. Get the SSL certificate from Couchbase Server.
2. Store the certificate in the appropriate certificate storage area for your environment.
3. Pass an SSL scheme with your connection string when creating your cluster object, as shown in the following example:

```
cropts.v.v3.connstr = "couchbases://securehost.net/mybucket?
ca_path=server_ca.pem";
```

Related Links

Couchbase 3.0 Features on page 9

Cluster-wide diagnostics

The Couchbase support team uses logs and other system diagnostics to analyze customer issues.

If you contact Couchbase customer support, you might be asked to collect diagnostics from your cluster. Couchbase Server offers several methods that enable you to efficiently collect diagnostics for an entire cluster. To collect diagnostic information, you can use either the web console, the command-line interface, or the REST API.

Impacted documentation

- couchbase-cli tool
- cbcollect_info tool

Related Links

Couchbase 3.0 Features on page 9

Managing diagnostics (UI) on page 42
The web console provides a graphical interface for collecting and sending system diagnostics to the Couchbase support team.

Managing diagnostics (CLI) on page 43
The command-line interface provides commands to start, stop, and report status for log collection.

Managing diagnostics (REST API) on page 45
The REST API enables you to collect and send diagnostics by using HTTP requests.

Managing diagnostics (UI)

The web console provides a graphical interface for collecting and sending system diagnostics to the Couchbase support team.

Related Links

Cluster-wide diagnostics on page 42
The Couchbase support team uses logs and other system diagnostics to analyze customer issues.

Collecting and uploading log information

Collecting and uploading logs from your nodes by using the web console

1. Click **Collect Information**.
2. In the **Collect Information** window, select the nodes you want to report on.

   You can choose to report on all accessible nodes or select one or more specific accessible nodes. You can collect information only from accessible nodes. *Accessible nodes* are nodes that are in the Up or Pending state. You cannot select nodes in the Down state because they cannot respond to cluster messages.

Option	Description
All accessible nodes	Collects logs for all accessible nodes in the cluster.
Selected nodes	Collects logs for a subset of the nodes in the cluster.

3. If you want to send the report to Couchbase, select the **Upload to Couchbase** check box and fill in the additional fields.

   The **Upload to Couchbase** check box is selected by default. The additional fields are:

   - **Upload to host**—enter the name of the host that you want the logs uploaded to. The host name can be either your own server or a specific server to which Couchbase support asked you to upload the files.
   - **Customer name**—enter the name of your company. This field is used to create file names for the logs and the URL that the files are uploaded to.
   - **Ticket number**—enter the Couchbase support ticket number associated with the request for logs. This field is optional, but if specified it is used as a path component of the URL that the files are uploaded to.
4. Click **Collect**.

The **Collect Results** window appears and begins to display the status of the collection process. The status display continues to update until the collection process is complete. The current status of each node is displayed. The node statuses are:

- **Pending**—collection hasn’t started on the node.
- **Collection in progress**—collection is currently running on the node.
- **Collected**—collection is complete. This status is displayed only if the logs will not be uploaded to a server.
- **Uploaded**—collection is complete and the logs have been uploaded to a server.
- **Failed to collect**—collection was unsuccessful. When available, additional information about the failure is shown in the Details column.
- **Collected, failed to upload**—collection was successful, but the logs could not be uploaded. When available, additional information about the failure is shown in the Details column.
- **Cancelled**—the user canceled collection for this node.

5. View the results.

After the collection process finishes, the **Collection Results** window contains a summary of the collection process result. The summary contains the following lists:

- Logs that were successfully uploaded. The list includes the destination URL for each log file. If you didn't choose to upload the files to Couchbase, you can use this information to locate the files and upload them manually.
- Logs that were collected but couldn't be uploaded. The list includes the node and path for each log file that couldn't be uploaded.
- Nodes that could not be collected from.

Canceling information collection

You can cancel the collection process at any time before it finishes.

1. In the **Collect results** window, click **Cancel**.
2. Verify whether you want to cancel the collection process.

Hiding the collection results window

You can hide the **Collection results** window before the collection process finishes by clicking the **Close** button in the window title bar. You will receive a notification when the collection process finishes.

Showing the collection results window

If you closed the **Collection results** window before the collection process finished, you can bring it back up to review the results.

1. Select the **Logs** tab.
2. Click **View Log Collect Status**.

Managing diagnostics (CLI)

The command-line interface provides commands to start, stop, and report status for log collection.

You can collect diagnostics through the command-line interface by using either the couchbase-cli tool or the cbcollect_info tool.

Related Links

Cluster-wide diagnostics on page 42
The Couchbase support team uses logs and other system diagnostics to analyze customer issues.

Collecting diagnostics via the couchbase-cli tool on page 44
You can start, stop, and view status of a diagnostics collection by using the couchbase-cli tool.

Collecting diagnostics via the cbcollect_info tool on page 44
You can start, stop, and get collection status by using the cbcollect_info tool.

Collecting diagnostics via the couchbase-cli tool

You can start, stop, and view status of a diagnostics collection by using the couchbase-cli tool.

Start log collection command

The start log collection command initiates a log collection process. You must specify whether to collect logs from all nodes or only specified nodes. You can optionally request the logs to be uploaded to Couchbase. If you request an upload to Couchbase, you must include the name of your organization and optionally a support ticket number. The command uses the following syntax:

`couchbase-cli collect-logs-start` -c *host*:8091 -u *username* -p *password* { --nodes=*node1,node2,...* | --all-nodes } [--upload --upload-host=*host* --customer=*name* [--ticket=*ticketNumber*]]

Stop log collection command

The stop log collection command cancels collection on each node. It uses the following syntax:

`couchbase-cli collect-logs-stop` -c *host*:8091 -u *username* -p *password*

Report log collection status command

The report log collection status command returns information about each node. It uses the following syntax:

`couchbase-cli collect-logs-status` -c *host*:8091 -u *username* -p *password*

Related Links

Managing diagnostics (CLI) on page 43
The command-line interface provides commands to start, stop, and report status for log collection.

Collecting diagnostics via the cbcollect_info tool

You can start, stop, and get collection status by using the cbcollect_info tool.

The cbcollect_info tool uses the following syntax:

`cbcollect_info` [`OPTIONS`] [{ *output-file* }]

Option	Description
--upload -u	Enable upload mode. When you use this option, you must also specify the --upload-host and --customer options, and you can optionally specify the --ticket option.
--upload-host=hostname -n hostname	Fully-qualified domain name of the host you want the logs uploaded to. The protocol prefix of the domain name, `https://`, is optional. It is the default only supported protocol. This option is required when you specify the `--upload` option.
--customer=customer-name -c customer-name	Customer name. This value is a string with a maximum length of 50 characters that contains only the following characters: [A-Za-z0-9_.-]. If any other characters are included in the string, the request will be rejected. This option is required when you specify the `--upload` option.
--ticket=ticket-number -t ticket-number	Couchbase support ticket number. This value is a string with a maximum length of 7 character that contains only digits 0-9. The ticket number is optional and is used only in conjunction with the --upload option.

Option	Description
`--script`	Send output to `stdout` in script mode format. Use this option when you invoke cbcollect_info programatically.

If the `--upload` option is used, the collected logs are compressed into a zip file and uploaded to a URL that has the following form: `protocol://hostname/customer-name/ticket-number/filename.zip`.

Related Links

Managing diagnostics (CLI) on page 43
The command-line interface provides commands to start, stop, and report status for log collection.

Managing diagnostics (REST API)

The REST API enables you to collect and send diagnostics by using HTTP requests.

You can use the following REST APIs to manage log collection.

HTTP method	URI path	Description
`GET`	`/pools`	Retrieve a list of nodes in the cluster
`POST`	`/collectLogs/start`	Start log collection
`POST`	`/collectLogs/cancel`	Cancel log collection
`GET`	`/pools/default/tasks`	Retrieve log collection status

Retrieving a list of cluster nodes

You can retrieve a list of the nodes in a cluster with the following HTTP request:

```
GET /pools
```

The response contains a list of the nodes in the cluster and the status of each node. The node status can be up, pending, or down.

Starting log collection

You can start log collection with the following REST request:

```
POST /collectLogs/start
```

The message body for this request contains a JSON document that uses the following parameters:

Name	Type	Description	Required
`customer`	string	Customer name. The valid characters are uppercase and lowercase letters, digits, underscore, period, and space. Maximum length is 50 characters. If spaces are used in the customer name, they are replaced with underscores. Invalid values are rejected.	yes, when the `upload` parameter is set to `true`.
`from`	string	Nodes to collect logs from. The value can be:	yes

Name	Type	Description	Required
		• allnodes—collect from all nodes in the cluster. • nodes—collect only from nodes specified in the nodes parameter.	
nodes	array	List of nodes to collect logs from.	yes, when the from parameter is set to nodes.
ticket	string	Couchbase support ticket number. The valid characters are digits 0 through 9. Maximum length is 7 characters. Invalid values are rejected.	no, but applicable only when the upload parameter is set to true.
upload	Boolean	Indicates whether to upload the collected logs to Couchbase support.	no
upload_host	string	Fully-qualified domain name, optionally prefixed with the https:// protocol specifier. If no protocol is specified, it defaults to https://, which is the only valid protocol. If an invalid protocol is specified, the request returns an error.	yes, when the upload parameter is set to true.

Stopping log collection

You can stop log collection with the following HTTP request:

```
POST /collectLogs/cancel
```

Retrieving log collection status

You can retrieve the status of the log collection operation with the following HTTP request:

```
GET /pools/default/tasks
```

The response to this request includes a JSON document that contains the following objects:

Name	Type	Description
per_node	array	Status information for each node. Each entry in the array contains the following items: • artifact—name of the artifact • details—additional information about the status

Name	Type	Description
		• `node_name`—name of the node • `result`—status of the collection operation on this node. Possible values are: `pending`, `in_progress`, `collected`, `uploaded`, `upload_failed`, `failed`, `cancelled`.
`status`	string	Overall status of the log collection operation. The value can be `idle` or `running`.
`type`	string	The kind of information the document contains. The value is `collect_logs`

The structure of the result document looks similar to this:

```
{
    "type": "collect_logs"
    "status": "status",
    "perNode": [
       {
            "artifact": "artifact_name",
            "node_name": "node_name",
            "result": "status",
            "details”: "optional additional details on failure reason etc."
        },
    ],
}
```

Related Links

Cluster-wide diagnostics on page 42
The Couchbase support team uses logs and other system diagnostics to analyze customer issues.

Couchbase Installation

The Couchbase Server 3.0 installation guide provides the following topics:

Related Links

Couchbase setup overview on page 49

Pre-installation on page 50

Pre-installation describes requirements, supported configurations, reserved or restricted items that are taken under consideration prior to installing Couchbase Server.

Red Hat Linux installation on page 54

Ubuntu Linux installation on page 60

Microsoft Windows installation on page 62

Mac OS X installation on page 64

Post-installation on page 66

Post-installation activities includes setting up the initial server and testing the server connection.

Uninstalling on page 76

Upgrading on page 78

Migrating on page 86

Couchbase setup overview

To start using Couchbase Server, follow these steps:

1. Make sure your machine meets the system requirements.
2. Install Couchbase Server.

   To install Couchbase Server on your machine, download the appropriate package for your chosen platform. For each platform, follow the corresponding platform-specific instructions.

   - To perfom a fresh installation (not an upgrade) over a previous Couchbase Server installation, remove Couchbase Server and any associated data from your machine before installing.
   - To retain existing datasets from a previous Couchbase Server installation, perform an upgrade installation.
3. Test the installation by connecting and storing some data using the native Memcached protocol.
4. Setup the new Couchbase Server system by completing the web-based setup instructions.

Warning: Implement the same operating system on all machines within a cluster. If XDCR is used, implement the same operating system on the other clusters. Mixed deployments, such as a cluster with both Linux and Windows server nodes, are not supported. This incompatibility is due to differences in the number of shards between platforms.

Related Links

Couchbase Installation on page 48

Related topics

Couchbase downloads

Initial server setup on page 66

Testing Couchbase Server on page 73

Uninstalling on page 76

Upgrading on page 78

Pre-installation

Pre-installation describes requirements, supported configurations, reserved or restricted items that are taken under consideration prior to installing Couchbase Server.

Related Links

Couchbase Installation on page 48

Resource requirements on page 50

Supported platforms on page 50

Network ports on page 51
Couchbase Server specific network ports for communication between server components and for communicating with the clients that access the data stored in the Couchbase cluster.

Supported web browsers on page 53
Mozzila Firefox, Safari, Google Chrome, and IE are supported

Resource requirements

The following hardware requirements are recommended for installation:

- Quad-core for key-value store, 64-bit CPU running at 3GHz
- Six cores if XDCR and views are used.
- 16GB RAM (physical)
- Block-based storage device (hard disk, SSD, EBS, iSCSI). Network filesystems (e.g. CIFS, NFS) are not supported.

A minimum specification machine should have the following characteristics:

- Dual-core CPU running at 2GHz for key-value store
- 4GB RAM (physical)

> **Note:** For development and testing purposes, a reduced CPU and RAM than the minimum specified can be used. This can be as low as 1GB of free RAM beyond operating system requirements and a single CPU core. However, for production, the minumum specification must be implemented. Performance on machines lower than the minimum specification is significantly lower and should not be used as an indication of the performance on a production machine. View performance on machines with less than 2 CPU cores is significantly reduced.

You must have enough memory to run your operating system and the memory reserved for use by Couchbase Server. For example, if 8GB of RAM is dedicated to Couchbase Server, there must be enough RAM to host the operating system. If additional applications and servers are running, additional RAM is required. For smaller systems, such as those with less than 16GB, allocate at least 40% of RAM to the operating system.

The following amount of storage must be available:

- 1GB for application logging
- At least twice the disk space to match your physical RAM for persistence of information

Related Links

Pre-installation on page 50
Pre-installation describes requirements, supported configurations, reserved or restricted items that are taken under consideration prior to installing Couchbase Server.

Related topics

Sizing guidelines on page 120
The primary considerations for sizing a Couchbase Server cluster are the number of nodes and node size.

Supported platforms

Your system must meet the following system requirements.

Couchbase Server provides platform support for Windows 2012 and separate packages for Ubuntu 12.04 and CentOS 6.

Platform	Version	32 / 64 bit	Supported	Recommended Version
Red Hat Enterprise Linux	5	32 and 64 bit	Developer and Production	RHEL 5.8
Red Hat Enterprise Linux	6	32 and 64 bit	Developer and Production	RHEL 6.3
CentOS	5	32 and 64 bit	Developer and Production	CentOS 5.8
CentOS	6	32 and 64 bit	Developer and Production	CentOS 6.3
Amazon Linux	2011.09	32 and 64 bit	Developer and Production	
Ubuntu Linux	10.04	32 and 64 bit	Developer and Production	
Ubuntu Linux	12.04	32 and 64 bit	Developer and Production	Ubuntu 12.04
Windows 2008	R2 with SP1	64 bit	Developer and Production	Windows 2008
Windows 2012		64 bit	Developer and Production	
Windows 7		32 and 64 bit	Developer only	
Windows 8		32 and 64 bit	Developer only	
Mac OS	10.7	64 bit	Developer only	
Mac OS	10.8	64 bit	Developer only	Mac OS 10.8

Important: Couchbase clusters with mixed platforms are not supported. Specifically, Couchbase Server on Mac OS X uses 64 vBuckets as opposed to the 1024 vBuckets used by other platforms. Due to this difference, if you need to move data between a Mac OS X cluster and a cluster hosted on another platform use `cbbackup` and `cbrestore`.

Related Links

Pre-installation on page 50
Pre-installation describes requirements, supported configurations, reserved or restricted items that are taken under consideration prior to installing Couchbase Server.

Related topics

Backup and restore on page 150
The predominant method for backing up and restoring data is using the CLI `cbbackup` and `cbrestore` tools.

Network ports

Couchbase Server specific network ports for communication between server components and for communicating with the clients that access the data stored in the Couchbase cluster.

The ports listed must be available on the host for Couchbase Server to run and operate correctly. Couchbase Server configures these ports automatically, but you must ensure that your firewall or IP tables configuration allow communication on the specified ports for each usage type. On Linux, the installer will notify you that you need to open these ports.

The following table lists the ports used for different types of communication with Couchbase Server, as follows:

Node to node

Where noted, these ports are used by Couchbase Server for communication between all nodes within the cluster. You must have these ports open on all to enable nodes to communicate with each other.

Node to client

Where noted, these ports should be open between each node within the cluster and any client nodes accessing data within the cluster.

Cluster administration

Where noted, these ports should be open and accessible to allow administration, whether using the REST API, command-line clients, and Web browser.

XDCR

Ports are used for XDCR communication between all nodes in both the source and destination clusters.

Port	Description	Node to Node	Node to Client	Cluster Administration	XDCR (version 1)	XDCR (version 2)
8091	Web Administration Port	Yes	Yes	Yes	Yes	Yes
8092	Couchbase API Port	Yes	Yes	No	Yes	Yes
11209	Internal Bucket Port	Yes	No	No	No	No
11210	Internal/ External Bucket Port	Yes	Yes	No	No	Yes
11211	Client interface (proxy)	No	Yes	No	No	No
11214	Incoming SSL Proxy	No	No	No	Yes	Yes
11215	Internal Outgoing SSL Proxy	No	No	No	Yes	Yes
18091	Internal REST HTTPS for SSL	No	No	No	Yes	Yes
18092	Internal CAPI HTTPS for SSL	No	No	No	Yes	Yes
4369	Erlang Port Mapper (`epmd`)	Yes	No	No	No	No

Port	Description	Node to Node	Node to Client	Cluster Administration	XDCR (version 1)	XDCR (version 2)
21100 to 21199 (inclusive)	Node data exchange	Yes	No	No	No	No

Port 8091	Used by the Web Console from outside the second level firewall (for REST/HTTP traffic).
Port 8092	Used to access views, run queries, and update design documents.
Port 11210	Used by smart client libraries or client-side Moxi to directly connect to the data nodes.
Port 11211	Used by pre-existing Couchbase and memcached (non-smart) client libraries that are outside the second level firewall to work.
Ports 11214, 11215, 18091, and 18092	Used for SSL XDCR data encryptions.
All other Ports	Used for other Couchbase Server communictions.

Related Links

Pre-installation on page 50
Pre-installation describes requirements, supported configurations, reserved or restricted items that are taken under consideration prior to installing Couchbase Server.

Supported web browsers

Mozzila Firefox, Safari, Google Chrome, and IE are supported

The Couchbase Web Console runs on the following browsers, with JavaScript support enabled:

- Mozilla Firefox 3.6 or higher

  To enable JavaScript, select the `Enable JavaScript` option within the `Content` panel of the application preferences.
- Safari 5 or higher

  To enable JavaScript, use the checkbox on the security tab of the application preferences.
- Google Chrome 11 or higher

  To enable JavaScript, use the `Allow all sites to run JavaScript (recommended)` option within the `Content` button of the `Under the Hood` section of the application preferences.
- Internet Explorer 8 or higher

  To enable JavaScript, by enabling `Active Scripting` within the `Custom Level`, section of the `Security` section of the `Internet Options` item of the `Tools` menu.

Related Links

Pre-installation on page 50
Pre-installation describes requirements, supported configurations, reserved or restricted items that are taken under consideration prior to installing Couchbase Server.

Red Hat Linux installation

Installation is supported on Red Hat and Red Hat-based operating systems such as CentOS.

Related Links

Couchbase Installation on page 48
Prerequisites on page 54
Installing on RHEL on page 55
Using user-defined ports on page 55
Installing as non-root, non-sudo on page 56
Installing as non-root, non-sudo is for development purposes only.

Installing multiple instances on page 57
Multiple instances of Couchbase Server can be installed on one physical machine for the LInux operating system. The number of instances depend on the capacity of the physical machine.

Related topics

Red Hat documentation
Initial server setup on page 66

Prerequisites

Before you install, check the supported platforms. The Red Hat installation uses the RPM package.

OpenSSL dependency checking

For Red Hat Enterprise Linux version 6.0, Couchbase Server RPM performs dependency checks for OpenSSL using `pkg-config`. Check that this pkg-config is installed and if not, install it:

```
root-> sudo yum install -y pkgconfig
```

Upon successful install the following output displays:

```
Loaded plugins .... Installed: pkgconfig.x86_64 1:0.21-2.el5 Complete!
```

OpenSSL 0983 installation

For Red Hat Enterprise Linux version 6.0 and above, install a specific OpenSSL dependency by running:

```
root-> yum install openssl098e
```

Some users can not install `openssl098e` with this command without having administrative privileges. If this issue is experienced, put the contents of the `lib64`directory into `opt/couchbase/lib`:

- Download `openssl098e–0.9.8e–17.el6.centos.2.x86_64.rpm’.
- Go to the directory where you extracted Couchbase Server: `cd opt/couchbase`.
- Extract the openssl098e RPM:

  ```
  rpm2cpio
  openssl098e-0.9.8e-17.el6.centos.2.x86_64.rpm | cpio --extract
  --make-directories --no-absolute-filenames
  ```
- Move the extracted files to the `/lib` directory for Couchbase Server: `mv usr/lib64/* lib/`

Related Links

Red Hat Linux installation on page 54

Installing on RHEL

To install Couchbase Server, use the `rpm` command-line tool with the RPM package that was downloaded. Log in as root (Superuser) to complete the installation:

```
root-> rpm --install couchbase-server version.rpm
```

Where `version` is the version number of the downloaded package.

Once the `rpm` command completes, Couchbase Server starts automatically, and is configured to automatically start during boot under the 2, 3, 4, and 5 runlevels. Refer to the Red Hat RPM documentation for more information about installing packages using RPM.

After installation finishes, the installation process displays a message similar to that below:

```
Minimum RAM required : 4 GB
    System RAM configured : 8174464 KB
    Minimum number of processors required :  4 cores
    Number of processors on the system : 4 cores


Starting couchbase-server[ OK ]

You have successfully installed Couchbase Server.
    Please browse to http://host_name:8091/ to configure your server.
    Please refer to http://couchbase.com for additional resources.

    Please note that you have to update your firewall configuration
    to allow connections to the following ports:

    11211, 11210, 11209, 4369, 8091, 8092 and from 21100 to 21299.


By using this software you agree to the End User License Agreement.
See /opt/couchbase/LICENSE.txt.
```

Once installed, use the Red Hat `chkconfig` command to manage the Couchbase Server service, including checking the current status and creating the links to enable and disable automatic start-up. To do the initial setup for Couchbase, open a web browser and access the Couchbase Web Console.

Related Links

Red Hat Linux installation on page 54

Using user-defined ports

This section describes how to install and run Couchbase server with user-defined ports rather than with the default 8091 port.

To run Couchbase server on user-defined ports, ensure that the minimum RAM and CPU are available for the Couchbase instance. For information about Red Hat Linux installation, see http://docs.couchbase.com/couchbase-manual–2.5/cb-install/#red-hat-linux-installation.

Note: Refer to the reserved network ports before creating user-defined ports.

Setting up Couchbase with user-defined ports

1. Install Couchbase.
    - If Couchbase is already installed and running, stop the Couchbase server.
2. Add the new user-defined ports to the **/opt/couchbase/etc/couchbase/static_config** file.

- The /opt/couchbase/etc/couchbase/static_config file is where Couchbase picks up the configuration parameters from.
- If port numbers are not specified, default ports are used.
- To override some or all default ports, append the user-defined ports to the file.

3. (Optional) CAPI port (default 8092) can be edited in the **/opt/couchbase/etc/couchdb/default.d/capi.ini** file by replacing 8092 with the new port name.
4. If the Couchbase server was previously configured, delete the **opt/couchbase/var/lib/couchbase/config/config.dat** file to remove the old configuration.
5. Start the Couchbase server

Ports to change

The following are the user-defined ports to add, replace, or append to the **/opt/couchbase/etc/couchbase/static_config** file.

```
{rest_port, 9000}.
{mccouch_port, 8999}.
{memcached_port, 12000}.
{memcached_dedicated_port, 11999}.
{moxi_port, 12001}.
{short_name, "ns_1"}.
{ssl_rest_port,11000}.
{ssl_capi_port,11001}.
{ssl_proxy_downstream_port,11002}.
{ssl_proxy_upstream_port,11003}.
```

Note: If the newly configured ports overlap with ports used by other running applications, Couchbase fails to start. If the newly configured ports overlap with ports used by Couchbase buckets, erlang crash notifications display in the log file.

Related Links

Red Hat Linux installation on page 54

Related topics

Network ports on page 51

Couchbase Server specific network ports for communication between server components and for communicating with the clients that access the data stored in the Couchbase cluster.

Installing as non-root, non-sudo

Installing as non-root, non-sudo is for development purposes only.

Attention: This installation is for development purposes only.

There may be cases when you want to install the server as a non-root, non-sudo user. A non-sudo, non-root installation still runs Couchbase Server and all Couchbase command-line tools. To do so on CentOS/Red Hat:

1. After downloading the Couchbase RPM, go to the directory where it is located and extract it:

   ```
   rpm2cpio couchbase-server-community_x86_64_2.0.0-1767-rel.rpm | cpio --
   extract --make-directories \
   --no-absolute-filenames
   ```

   In the directory where the files were extracted, the `opt` and `etc` subdirectories are available.
2. If you need to separately provide openssl098e, put the contents of this library into `opt/couchbase/lib`:

   a. Download `openssl098e–0.9.8e–17.el6.centos.2.x86_64.rpm’.
   b. Go to the directory where you extracted Couchbase Server: `cd opt/couchbase`.
   c. Extract the openssl098e RPM:

```
rpm2cpio
openssl098e-0.9.8e-17.el6.centos.2.x86_64.rpm | cpio --extract
--make-directories --no-absolute-filenames
```

   d. Move the extracted files to the `/lib` directory for Couchbase Server: `mv usr/lib64/* lib/`
3. After you extract the Couchbase Server install files, go to the subdirectory: `cd opt/couchbase`
4. Run the following password-related script: `./bin/install/reloc.sh \`pwd``

   This enables you to continue the install as a non-root, non-sudo user.
5. To run the server: `./bin/couchbase-server \-- -noinput -detached`
6. To stop the server: `./bin/couchbase-server -k`

Related Links

Red Hat Linux installation on page 54

Installing multiple instances

Multiple instances of Couchbase Server can be installed on one physical machine for the LInux operating system. The number of instances depend on the capacity of the physical machine.

Attention: Multiple instance on one machine is recommended for development purposes only.

Requirements

Ensure that a minumum of 4GN RAN and 8 Core CPU are available for each Couchbase instance. When installing multiple instances of Couchbase on a physical machine, install as one of the following:

- sudo user
- non-root, non-sudo user

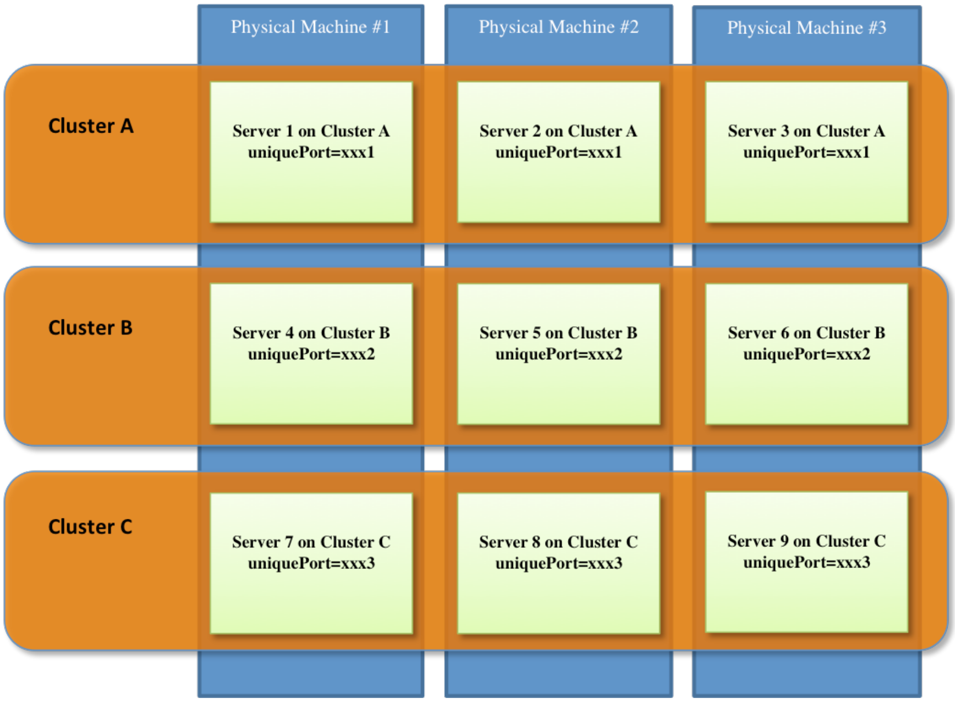
Note: See the reserved Couchbase Server Network ports and User-defined ports before creating user-defined ports.

Recommendations

Install each instance of a cluster on a different physical machine to ensure data recovery if a failure occurs.

Note: The number of Couchbase servers that can be installed on a physical machine depends on the RAM and CPU available.

The following graphic shows a cluster configuration with multiple Couchbase instances on a physical machine. In addition, by having three (3) Couchbase server in a cluster and each server installed on different physical machines, the configuration reduces the risk of data loss from a hardware failure.



Setting up multiple instances

To set up multiple instances running on a physical machine:

1. Install Couchbase Server as a sudo user or as a non-root, non-sudo user. For more information about installing as non-root, non-sudo.
2. Create user-defined ports in the **/opt/couchbase/etc/couchbase/static_config** file.
3. In the **/etc/security/limits.conf** file, ensure that the hard and soft limits for the `nofile` parameter are set to a value greater than 10240.
4. Change the short_name parameter that identifies the instance (default: ns_1), to a different short_name in the **/opt/couchbase/etc/couchbase/static_config** file.
    - The short_name value must different for each instance that resides on the same physical server.
5. Change the two occurrences short_name in the **/opt/couchbase/bin/couchbase-server** file. For example, use the `sed` utility.
    - `sed -i 's/ns_1/ns_inst1/g' bin/couchbase-server`
6. Start the Couchbase instance.
7. Repeat the steps to install other instances.

Important: While creating the cluster make sure the perServer RAM quota is calculated keeping in mind the number of instances planned to be installed on the machine.

When configuring the instance for the cluster, Couchbase provides a default value for the perServer RAM quota. This default value is based on the total RAM quota available on the physical machine. Modify this value.

Troubleshooting

If any bucket created on the nodes appear to be in pending state or if rebalance fails with not_all_nodes_are_ready_yet, there could be a mismatch of the short_name value in the following files:

```
/opt/couchbase/bin/couchbase-server
/opt/couchbase/etc/couchbase/static_config
```

Limitations

- Cbrecovery is unavailable on customized ports
- Cbworkloadgen is unavailable
- Offline upgrade is unavailable
- If a bucket is created on a dedicated port, some of the operations results in the error, "could not listen on port xxx", even though the operation still succeeds. This error is logged regardless of the port that is used.

Related Links

Red Hat Linux installation on page 54

Ubuntu Linux installation

Installing on Ubuntu

To install:

1. For Ubuntu version 12.04, you need to install a specific OpenSSL dependency by running:

   ```
   root-> apt-get install libssl0.9.8
   ```
2. The Ubuntu Couchbase installation uses the DEB package. To install, use the `dpkg` command-line tool using the DEB file that you downloaded. The following example uses `sudo` which will require root-access to allow installation:

   ```
   > dpkg -i couchbase-server version.deb
   ```

   Where `version` is the version number of the downloaded package.

   Once the `dpkg` command has been executed, the Couchbase server starts automatically, and is configured to automatically start during boot under the 2, 3, 4, and 5 run levels. Refer to the Ubuntu documentation for more information about installing packages using the Debian package manager.

   After installation has completed, the installation process displays a message similar to the following:

```
Selecting previously deselected package couchbase-server.
(Reading database … 73755 files and directories currently installed.)
Unpacking couchbase-server (from couchbase-server_x86_64_2.1.0-xxx-rel.deb) …
libssl0.9.8 is installed. Continue installing
Minimum RAM required : 4 GB
System RAM configured : 4058708 KB


Minimum number of processors required : 4 cores
Number of processors on the system : 4 cores
Setting up couchbase-server (2.1) …

* Started couchbase-server

You have successfully installed Couchbase Server.
Please browse to http://slv-0501:8091/ to configure your server.
Please refer to http://couchbase.com for additional resources.

Please note that you have to update your firewall configuration to
allow connections to the following ports: 11211, 11210, 11209, 4369,
8091, 8092, 18091, 18092, and from 21100 to 21299.

By using this software you agree to the End User License Agreement.
See /opt/couchbase/LICENSE.txt

Processing triggers for ureadahead …
ureadahead will be reprofiled on next reboot
```

After successful installation, use the `service` command to manage the Couchbase Server service, including checking the current status. Refer to the Ubuntu documentation for instructions. To provide initial setup for Couchbase, open a web browser and access the web administration interface.

Installing on Ubuntu as non-root, non-sudo

This installation is for development purposes only. There may be cases when you want to install the server as a non-root, non-sudo user. If you perform a non-sudo, non-root installation you will be still be able to run Couchbase Server and all Couchbase command-line tools. To do so on Ubuntu:

1. After you download the Couchbase DEB package, go to the directory where it is located and extract it:

   ```
   > dpkg-deb -x couchbase-server-community_x86_64_2.0.0-1767-rel.deb $HOME
   ```

   In the directory where you extracted the files, you will see `opt` and `etc` subdirectories.
2. After you extract the Couchbase Server install files, go to the subdirectory:

   ```
   cd opt/couchbase
   ```
3. Run this password-related script:

   ```
   ./bin/install/reloc.sh `pwd`
   ```

   This enables you to continue the install as a non-root, non-sudo user.
4. To run the server:

   ```
   ./bin/couchbase-server -- -noinput -detached
   ```
5. To stop the server:

   ```
   ./bin/couchbase-server -k
   ```

Related Links

Couchbase Installation on page 48

Related topics

Supported platforms on page 50

Network ports on page 51

Couchbase Server specific network ports for communication between server components and for communicating with the clients that access the data stored in the Couchbase cluster.

Startup and shutdown on Linux on page 71

Microsoft Windows installation

Before installing, check the supported platforms. To install on Windows, download the Windows installer package. This is supplied as a Windows executable. You can install the package either using the wizard, or by doing an unattended installation process. In either case, make sure that you have no anti-virus software running on the machine before you start the installation process. You also need administrator privileges on the machine where you install it.

The TCP/IP port allocation on Windows by default includes a restricted number of ports available for client communication. For more information on this issue, including information on how to adjust the configuration and increase the available ports, see the MSDN: Avoiding TCP/IP Port Exhaustion article.

Important: Couchbase Server uses the Microsoft C++ redistributable package, which automatically downloads during installation. However, if another application on your machine is already using the package, your installation process may fail. To ensure that your installation process completes successfully, shut down all other running applications during installation.

For Windows 2008, you must upgrade your Windows Server 2008 R2 installation with Service Pack 1 installed before running Couchbase Server. You can obtain Service Pack 1 from [Microsoft TechNet](http://technet.microsoft.com/en-us/library/ff817647(v=ws.10).aspx).

The standard Microsoft Server installation does not provide an adequate number of ephemeral ports for Couchbase clusters. Without the correct number of open ephemeral ports, you may experience errors during rebalance, timeouts on clients, and failed backups. The Couchbase Server installer will check for your current port setting and adjust it if needed. See [Microsoft KB-196271](http://support.microsoft.com/kb/196271).

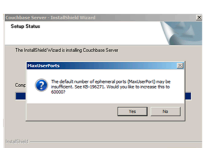
Installation wizard

1. Double click on the downloaded executable file.

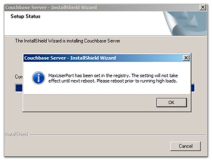
   The installer for windows will detect if any redistributable packages included with Couchbase need to be installed or not. If these packaged are not already on your system, the install will automatically install them along with Couchbase Server.
2. Follow the install wizard to complete the installation.

   You will be prompted with the `Installation Location` screen. You can change the location where the Couchbase Server application is located. Note that this does not configure the location of where the persistent data will be stored, only the location of the server itself.

   The installer copies necessary files to the system. During the installation process, the installer will also check to ensure that the default administration port is not already in use by another application. If the default port is unavailable, the installer will prompt for a different port to be used for administration of the Couchbase server. The installer asks you to set up sufficient ports available for the node. By default Microsoft Server will not have an adequate number of ephemeral ports, see *Microsoft Knowledge Base Article 196271*

   
3. Click **Yes**.

   Without a sufficient number of ephemeral ports, a Couchbase cluster fails during rebalance and backup; other operations such as client requests will timeout. If you already changed this setting you can click no. The installer will display this panel to confirm the update:



4. Restart the server for the port changes to be applied.

After installation you should follow the server setup instructions.

Unattended installation

To use the unattended installation process, you first record your installation settings in wizard installation. These settings are saved to a file. You can use this file to silently install other nodes of the same version.

To record your install options, open a Command Terminal or Power and start the installation executable with the `/r` command-line option:

```
> couchbase_server_version.exe /r /f1your_file_name.iss
```

You will be prompted with installation options, and the wizard will complete the server install. We recommend you accept an increase in `MaxUserPort`. A file with your options will be recorded at `C:\Windows\your_file_name.iss`.

To perform an installation using this recorded setup file, copy the `your_file_name.iss` file into the same directory as the installer executable. Run the installer from the command-line using the `/s` option:

```
> couchbase_server_version.exe /s -f1your_file_name.iss
```

You can repeat this process on multiple machines by copying the install package and the `your_file_name.iss` file to the same directory on each machine.

Related Links

Couchbase Installation on page 48

Related topics

Supported platforms on page 50

Startup and shutdown on Windows on page 71

Microsoft Knowledge Base Article 196271

MSDN: Avoiding TCP/IP Port Exhaustion

Mac OS X installation

Before you install, make sure you check the supported platforms. Couchbase Server on Mac OS X is for development purposes only. The Mac OS X installation uses a Zip file which contains a standalone application that can be copied to the `Applications` folder or to any other location you choose. The installation location is not the same as the location of the Couchbase data files.

Please use the default archive file handler in Mac OS X, Archive Utility, when you unpack the Couchbase Server distribution. It is more difficult to diagnose non-functioning or damaged installations after extraction by other third party archive extraction tools.

Warning: Due to limitations within the Mac OS X operating system, the Mac OS X implementation is incompatible with other operating systems. It is not possible either to mix operating systems within the same cluster, or configure XDCR between a Mac OS X and Windows or Linux cluster. If you need to move data between a Mac OS X cluster and a cluster hosted on another platform, use \`cbbackup\` and \`cbrestore\`. For more information, see [Backup and Restore Between Mac OS X and Other Platforms](../cb-admin/#couchbase-backup-restore-mac).

Installing on Mac OS X

To install:

1. Delete any previous installs of Couchbase Server at the command line or by dragging the icon to the Trash can.
2. Remove remaining files from previous installations:

```
> rm -rf ~/Library/Application Support/Couchbase
> rm -rf ~/Library/Application Support/Membase
```

3. Download the Mac OS X zip file.
4. Double-click the downloaded Zip installation file to extract the server. This creates a single folder, the `Couchbase Server.app` application.
5. Drag and Drop `Couchbase Server.app` to your chosen installation folder, such as the system `Applications` folder.

After the installation completes, you can double-click on `Couchbase Server.app` to start it. The Couchbase Server icon appears in the menu bar on the right-hand side. If you have not yet configured your server, then the Couchbase Web Console opens and you should to complete the Couchbase Server setup process.

The Couchbase application runs as a background application. If you click on the icon in the menu bar you see a list of operations that can be performed.

The command line tools are included in the Couchbase Server application directory. You can access them in Terminal by using the full path of the Couchbase Server installation. By default, this is `/Applications/Couchbase Server.app/Contents/Resources/couchbase-core/bin/`.

Installing on Mac OS X as non-root, non-sudo

This installation is for development purposes only. There may be cases when you want to install the server as a non-root, non-sudo user. If you perform a non-sudo, non-root installation you will be still be able to run Couchbase Server and all Couchbase command-line tools. To do so on Mac OS X:

1. After you download Couchbase Server, open Terminal and go to the Downloads directory:

```
cd ~/Downloads/
```

2. Unzip the package containing Couchbase Server:

```
open couchbase-server-enterprise_x86_64_2.1.0.zip
```

3. Move Couchbase App to your `/Applications` folder:

```
mv couchbase-server-enterprise_x86_64_2.1.0/Couchbase\ Server.app /
Applications/
```

4. Start the server from Terminal:

   ```
   open /Applications/Couchbase\ Server.app
   ```

   This will enable you to use Couchbase Server as a non-sudo, non-root user.
5. To stop the server, Click on the Couchbase icon in the menu bar and select Quit Couchbase Server.

Related Links

Couchbase Installation on page 48

Related topics

Supported platforms on page 50

Startup and shutdown on Mac OS X on page 72

Post-installation

Post-installation activities includes setting up the initial server and testing the server connection.

Related Links

Couchbase Installation on page 48
Initial server setup on page 66
Using hostnames on page 69
Startup and shutdown on Linux on page 71
Startup and shutdown on Windows on page 71
Startup and shutdown on Mac OS X on page 72
Testing Couchbase Server on page 73

Initial server setup

Tip: Clear your browser cache before doing the setup process. You can find notes and tips on how to do this on different browsers and platforms on www.wikihow.com.

On all platforms you can access the web console by connecting to the embedded web server on port 8091. For example, if your server can be identified on your network as `servera`, you can access the web console by opening `http://servera:8091/`. You can also use an IP address or, if you are on the same machine, `http://localhost:8091`. If you set up Couchbase Server on another port other than `8091`, go to that port.

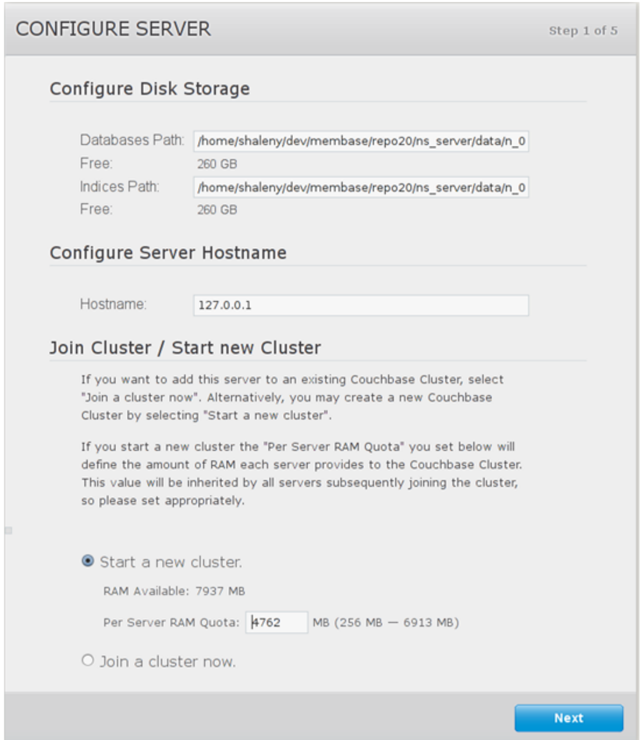
1. Open Couchbase Web Console.
2. Set the disk storage and cluster configuration.

   The `Configure Disk Storage` option specifies the location of the persistent storage used by Couchbase Server. The setting affects only this node and sets the directory where all the data will be stored on disk. This will also set where the indices created by views will be stored. If you are not indexing data with views you can accept the default setting. For the best performance, you may want to configure different disks for the server, for storing your document and for index data.

   The `Configure Server Memory` section sets the amount of physical RAM that will be allocated by Couchbase Server for storage.

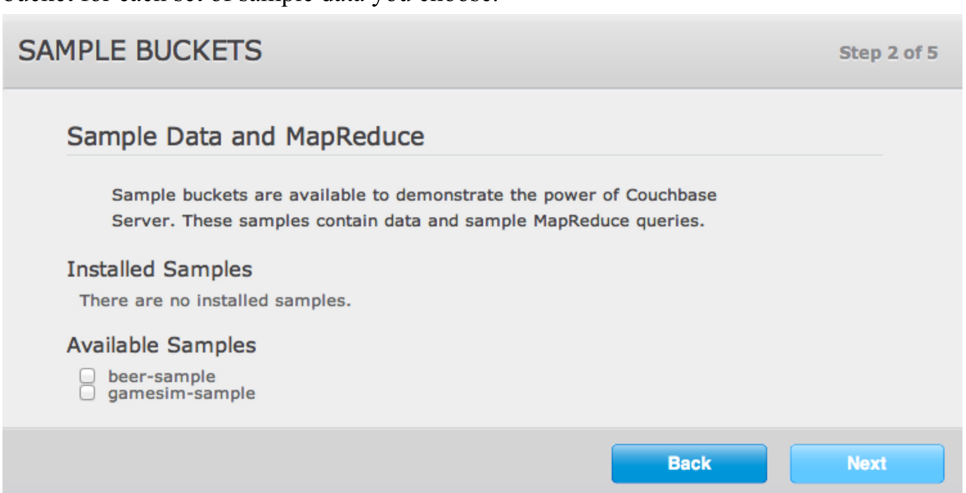
   If you are creating a new cluster, this is the amount of memory that will be allocated on each node within your Couchbase cluster. The memory for each node in a cluster must be the same amount. You must specify a value that can be supported by all the nodes in your cluster as this setting will apply to the entire cluster.

   The default value is 60% of your total free RAM. This figure is designed to allow RAM capacity for use by the operating system caching layer when accessing and using views.
3. Provide a node IP or hostname under Configure Server Hostname.



4. Provide the IP Address or hostname of an existing node, and administrative credentials for that existing cluster.
5. To join an existing cluster, Check Join a cluster now.
6. Click Next.

   The Sample Buckets panel appears where you can select the sample data buckets you want to load.
7. Click the names of sample buckets to load Couchbase Server. These data sets demonstrate Couchbase Server and help you understand and develop views. If you decide to install sample data, the installer creates one Couchbase bucket for each set of sample data you choose.



   After you create sample data buckets a Create Bucket panel appears where you create new data buckets
8. Set up a test bucket for Couchbase Server. You can change all bucket settings later except for the bucket name.

   Enter ‘default’ as the bucket name and accept all other defaults in this panel.

   Couchbase Server will create a new data bucket named ‘default.’ You can use this test bucket to learn more about Couchbase and can use it in a test environment.
9. Select `Update Notifications`.

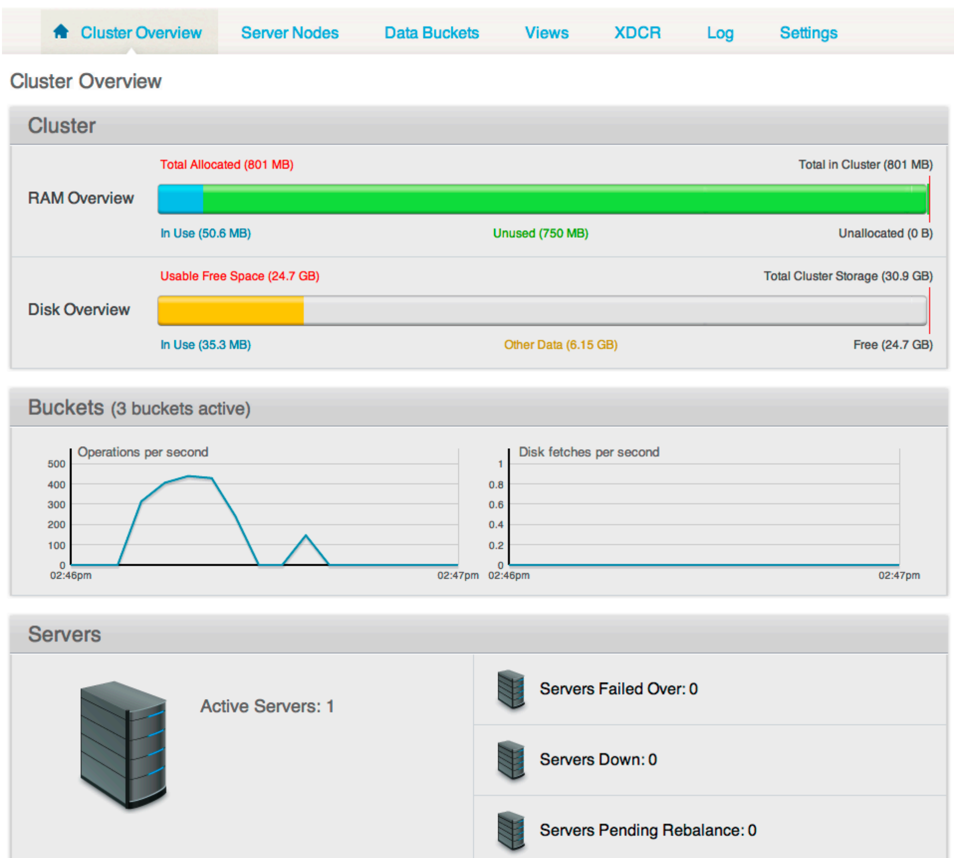
Couchbase Web Console communicates with Couchbase nodes and confirms the version numbers of each node. As long as you have internet access, this information will be sent anonymously to Couchbase corporate.. Couchbase corporate only uses this information to provide you with updates and information that will help us improve Couchbase Server and related products.

When you provide an email address we will add it to the Couchbase community mailing list for news and update information about Couchbase and related products. You can unsubscribe from the mailing list at any time using the unsubscribe link provided in each newsletter. Web Console communicates the following information:

- The current version. When a new version of Couchbase Server exists, you get information on where you can download the new version.
- Information about the size and configuration of your Couchbase cluster to Couchbase corporate. This information helps us prioritize our development efforts.

10. Enter a username and password. Your username must have no more than 24 characters, and your password must have 6 to 24 characters. You use these credentials each time you add a new server into the cluster. These are the same credentials you use for Couchbase REST API. See the Couchbase REST API.

Once you finish this setup, you see Couchbase Web Console with the Cluster Overview page:



Your server is now running and ready to use. After you install your server and finish initial setup you can also optionally configure other settings, such as the port, RAM, using any of the following methods:

- Using command-line tools

  The command line tools provided with your Couchbase Server installation includes `couchbase-cli`. This tool provides access to the core functionality of the Couchbase Server by providing a wrapper to the REST API. For information about CLI, see couchbase-cli tool.
- Using the REST API

Couchbase Server can be configured and controlled using a REST API. In fact, the REST API is the basis for both the command-line tools and Web interface to Couchbase Server.

For more information on using the REST API see, the Couchbase REST API.

Related Links

Post-installation on page 66
Post-installation activities includes setting up the initial server and testing the server connection.

Related topics

Sizing guidelines on page 120
The primary considerations for sizing a Couchbase Server cluster are the number of nodes and node size.

Using hostnames on page 69

Beer sample bucket on page 306

Game Simulation sample bucket on page 309

Data Buckets on page 259

Command-line interface overview on page 353

couchbase-cli tool on page 361

REST API overview on page 426

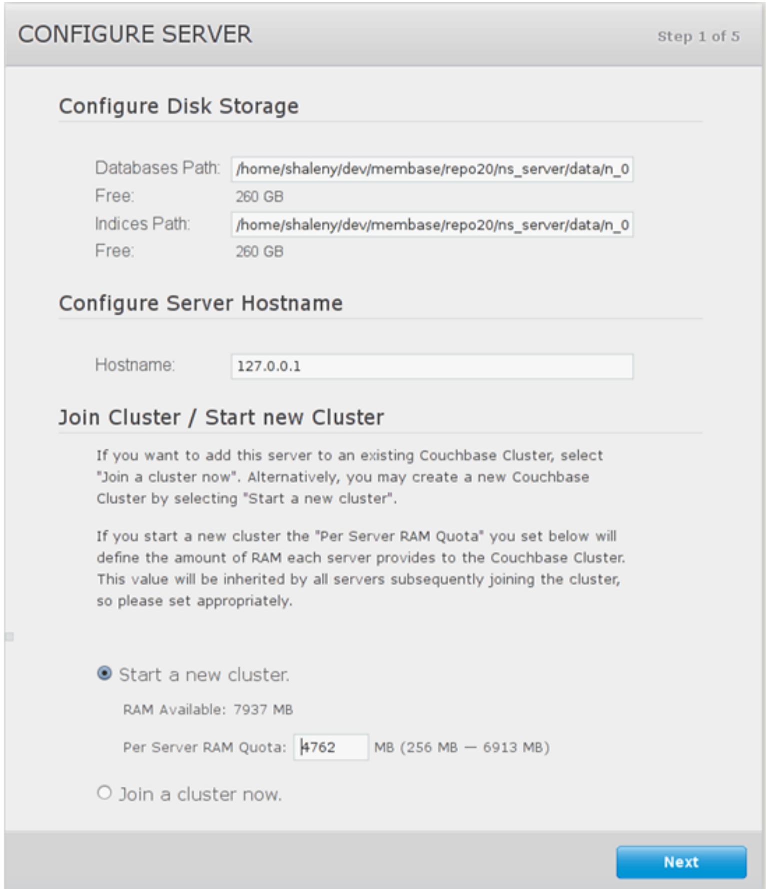
Using hostnames

When you first install Couchbase Server you can access using a default IP address. There may be cases where you want to provide a hostname for each instance of a server. Each hostname you provide should be a valid one and will ultimately resolve to a valid IP Address. If you restart a node, it will use the hostname once again. If you failover or remove a node from a cluster, the node needs to be configured with the hostname once again.

There are several ways you can provide hostnames. You can provide a hostname when installing a Couchbase Server on a machine, when adding a node to an existing cluster for online upgrade, or via a REST API call. Couchbase Server stores this in a config file on disk.

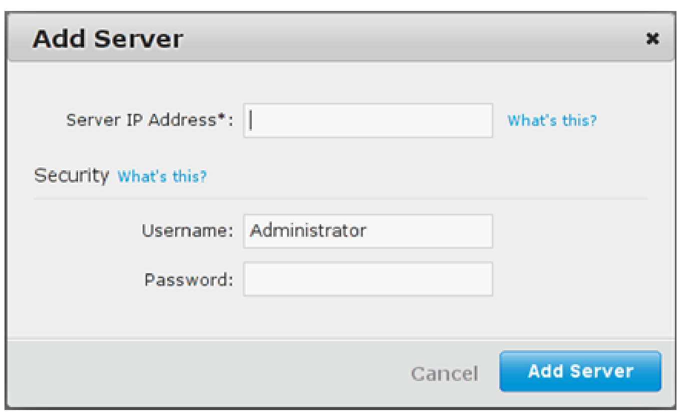
- On initial setup

  In the first screen, provide either a hostname or IP address under **Configure Server Hostname**. The provided hostname survives node restart.



- While adding a node

  If a new node is being added to an existing 2.0.1 or older Couchbase cluster, first setup the hostname for the new node in the setup wizard. Add a new node to a cluster by providing either a hostname or IP address under **Add Server > Server IP Address field**.



- Via REST API

  Provide a hostname for a node a hostname with the REST request at the `/node/controller/rename` endpoint.

  If this method is used, provide the hostname before adding a node to a cluster. If a hostname is provided for a node that is already part of a Couchbase cluster, the server rejects the request and returns `error 400 reason: unknown ["Renaming is disallowed for nodes that are already part of a cluster"]`.

```
curl -v -X POST -u Administrator:asdasd \
http://127.0.0.1:8091/node/controller/rename -d hostname=shz.localdomain
```

Hostname errors

Where you provide the IP address and port for the node and administrative credentials for the cluster. The value you provide for `hostname` should be a valid hostname for the node. Possible errors that may occur when you do this request:

- Could not resolve the hostname. The hostname you provide as a parameter does not resolve to a IP address.
- Could not listen. The hostname resolves to an IP address, but no network connection exists for the address.
- Could not rename the node because name was fixed at server start-up.
- Could not save address after rename.
- Requested name hostname is not allowed. Invalid hostname provided.
- Renaming is disallowed for nodes that are already part of a cluster.

Hostnames when upgrading from 1.8.1

If you perform an offline upgrade from Couchbase 1.8.1+ and you have a configured hostname using the instructions from cloud deployment informaiton, the uses this configuration. If an online upgrade from 1.8.1 is performed, add the hostname when creating the new node. node.

Related Links

Post-installation on page 66
Post-installation activities includes setting up the initial server and testing the server connection.

Related topics

Couchbase in the cloud on page 128
Couchbase Server is extremely easy to deploy in the cloud.

Startup and shutdown on Linux

On Linux, Couchbase Server is installed as a standalone application with support for running as a background (daemon) process during startup through the use of a standard control script, `/etc/init.d/couchbase-server`. The startup script is automatically installed during installation from one of the Linux packaged releases (Debian/Ubuntu or Red Hat/CentOS). By default Couchbase Server is configured to be started automatically at run levels 2, 3, 4, and 5, and explicitly shutdown at run levels 0, 1 and 6.

To manually start Couchbase Server using the startup/shutdown script:

```
>> sudo /etc/init.d/couchbase-server start
```

To manually stop Couchbase Server using the startup/shutdown script:

```
> sudo /etc/init.d/couchbase-server stop
```

Related Links

Post-installation on page 66
Post-installation activities includes setting up the initial server and testing the server connection.

Startup and shutdown on Windows

On Windows, Couchbase Server is installed as a Windows service. You can use the `Services` tab within the Windows Task Manager to start and stop Couchbase Server.

You will need power user or administrator privileges, or have been separately granted the rights to manage services to start and stop Couchbase Server. By default, the service automatically starts when the machine boots.

Couchbase can be started and stopped via:

- Windows Task Manager

- Windows system `net` command
- Couchbase-supplied `.bat` scripts

Via Windows Task Manager

To manually start the service from the Windows interface:

1. Open the Windows Task Manager and choose the **Services** tab to open the Services management console.

   Alternatively, select the **Start**, choose **Run** and then type `Services.msc` to open the Services management console.
2. Find the `CouchbaseServer` service and right-click.
3. Select **Start** or **Stop** as appropriate.

Note: You can also alter the configuration so that the service is not automatically started during boot.

Via Windows system `net` command

To start and stop Couchbase Server using `net`:

```
net start CouchbaseServer
```

```
net stop CouchbaseServer
```

Via Couchbase-supplied `.bat` scripts

The Couchbase-supplied start and stop scripts are provided in the standard installation in the `bin` directory.

To start and stop the server using these scripts:

```
C:\Program Files\Couchbase\Server\bin\service_start.bat
```

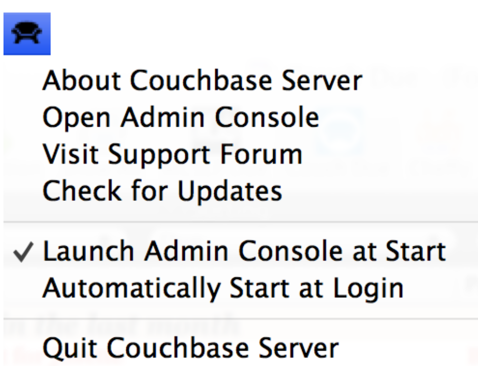
```
C:\Program Files\Couchbase\Server\bin\service_stop.bat
```

Related Links

Post-installation on page 66
Post-installation activities includes setting up the initial server and testing the server connection.

Startup and shutdown on Mac OS X

On Mac OS X, Couchbase Server is supplied as a standard application. You can start Couchbase Server by double clicking on the application. Couchbase Server runs as a background application which installs a menu bar item through which you can control the server.



The individual menu options perform the following actions:

- About Couchbase

  Opens a standard About dialog containing the licensing and version information for the Couchbase Server installed.

- Open Admin Console

  Opens the Web Administration Console in your configured default browser.
- Visit Support Forum

  Opens the Couchbase Server support forum within your default browser at the Couchbase website where you can ask questions to other users and Couchbase developers.
- Check for Updates

  Checks for updated versions of Couchbase Server. This checks the currently installed version against the latest version available at Couchbase and offers to download and install the new version. If a new version is available, you will be presented with a dialog containing information about the new release.

  If a new version is available, you can choose to skip the update, notify the existence of the update at a later date, or to automatically update the software to the new version.

  If you choose the last option, the latest available version of Couchbase Server will be downloaded to your machine, and you will be prompted to allow the installation to take place. Installation will shut down your existing Couchbase Server process, install the update, and then restart the service once the installation has been completed.

  Once the installation has been completed you will be asked whether you want to automatically update Couchbase Server in the future.

  Using the update service also sends anonymous usage data to Couchbase on the current version and cluster used in your organization. This information is used to improve our service offerings.

  You can also enable automated updates by selecting the `Automatically download and install updates in the future` checkbox.
- Launch Admin Console at Start

  If this menu item is checked, then the Web Console for administrating Couchbase Server will be opened whenever the Couchbase Server is started. Selecting the menu item will toggle the selection.
- Automatically Start at Login

  If this menu item is checked, then Couchbase Server will be automatically started when the Mac OS X machine starts. Selecting the menu item will toggle the selection.
- Quit Couchbase

  Selecting this menu option will shut down your running Couchbase Server, and close the menu bar interface. To restart, you must open the Couchbase Server application from the installation folder.

Related Links

Post-installation on page 66
Post-installation activities includes setting up the initial server and testing the server connection.

Testing Couchbase Server

Testing the connection to the Couchbase Server can be performed in a number of different ways. Connecting to the node using the web client to connect to the admin console should provide basic confirmation that your node is available. Using the `couchbase-cli` command to query your Couchbase Server node will confirm that the node is available.

Note: The Couchbase Server web console uses the same port number as clients use when communicated with the server. If you can connect to the Couchbase Server web console, administration and database clients should be able to connect to the core cluster port and perform operations. The Web Console will also warn if the console loses connectivity to the node.

To verify your installation works for clients, you can use either the `cbworkloadgen` command, or `telnet`. The `cbworkloadgen` command uses the Python Client SDK to communicate with the cluster, checking both the cluster administration port and data update ports..

Using `telnet` only checks the Memcached compatibility ports and the memcached text-only protocol.

Related Links

Post-installation on page 66
Post-installation activities includes setting up the initial server and testing the server connection.

Testing with cbworkloadgen on page 74
The `cbworkloadgen` is a basic tool that can be used to check the availability and connectivity of a Couchbase Server cluster.

Testing installation with telnet on page 74
The simplest method to determine whether Couchbase Server is running is by using Telnet to connect to the server and using the Memcached text protocol.

Testing with cbworkloadgen

The `cbworkloadgen` is a basic tool that can be used to check the availability and connectivity of a Couchbase Server cluster.

The `cbworkloadgen` executes a number of different operations to provide basic testing functionality for your server.`cbworkloadgen` provides basic testing functionality. It does not provide performance or workload testing.

To test a Couchbase Server installation using `cbworkloadgen`, execute the command supplying the IP address of the running node:

```
>> cbworkloadgen -n localhost:8091
Thread 0 - average set time : 0.0257480939229 seconds , min : 0.00325512886047
 seconds , max : 0.0705931186676 seconds , operation timeouts 0
```

The progress and activity of the tool can also be monitored within the web console.

For a longer test you can increase the number of iterations:

```
> cbworkloadgen -n localhost:8091 --items=100000
```

Related Links

Testing Couchbase Server on page 73

Testing installation with telnet

The simplest method to determine whether Couchbase Server is running is by using Telnet to connect to the server and using the Memcached text protocol.

Telnet must be installed on your server to connect to Couchbase Server using this method. Telnet is supplied as standard on most platforms, or may be available as a separate package that should be easily installable via your operating systems standard package manager.

Note: You do not need to use the Telnet method for communicating with your server within your application. Instead, use one of the Couchbase SDKs.

Connect to the server:

```
> telnet localhost1
Trying 127.0.0.1...
Connected to localhost.localdomain (127.0.0.1).
Escape character is '^]'.
```

Make sure it’s responding (stats is a great way to check basic health):

```
stats
STAT delete_misses 0
STAT ep_io_num_write 0
STAT rejected_conns 0
...
STAT time 1286678223
...
STAT curr_items_tot 0
...
STAT threads 4
STAT pid 23871
...
```

```
END
```

Put a key in:

```
set test_key 0 0 1
a
STORED
```

Retrieve the key:

```
get test_key
VALUE test_key 0 1
a
END
```

Disconnect:

```
quit
Connection closed by foreign host.
>
```

All of the Memcached protocols commands will work through Telnet.

Related Links

Testing Couchbase Server on page 73

Uninstalling

If you want to uninstall Couchbase Server from your system you must choose the method appropriate for your operating system.

Before removing Couchbase Server from your system, you should do the following:

- Shutdown your Couchbase Server.
- If your machine is part of an active cluster, you should rebalance your cluster to take the node out of your configuration.
- Update your clients to point to an available node within your Couchbase Server cluster.

Uninstalling on Deb ian/Ubuntu Linux

To uninstall the software on a Ubuntu Linux system, run the following command:

```
> sudo dpkg -r couchbase-server
```

Refer to the Ubuntu documentation for more information about uninstalling packages using `dpkg`.

You may need to delete the data files associated with your installation. The default installation location is `/opt`. If you selected an alternative location for your data files, you will need to separately delete each data directory from your system.

Uninstalling on Windows

To uninstall the software on a Windows system you must have Administrator or Power User privileges to uninstall Couchbase.

To remove, choose `Start` > `Settings` > `Control Panel`, choose `Add or Remove Programs`, and remove the Couchbase Server software.

Uninstalling on Mac OS X

To uninstall on Mac OS X:

1. Open the `Applications` folder, and then drag the `Couchbase Server` application to the trash. You may be asked to provide administrator credentials to complete the deletion.
2. To remove the application data, you will need to delete the `Couchbase` folder from the `~/Library/Application Support` folder for the user that ran Couchbase Server.

Related Links

Couchbase Installation on page 48
Uninstalling on RHEL on page 76
Uninstalling on Ubuntu Linux on page 77
Uninstalling on Windows on page 77

Uninstalling on RHEL

To uninstall the software on a Red Hat Linux system, run the following command:

```
> sudo rpm -e couchbase-server
```

Refer to the Red Hat RPM documentation for more information about uninstalling packages using RPM.

The data files associated with your installation may need to be deleted. The default installation location is `/opt`. If an alternative location for your data files was specified, each data directory must be individually deleted from your system.

Related Links

Uninstalling on page 76

Uninstalling on Ubuntu Linux

To uninstall the software on a Ubuntu Linux system, run the following command:

```
> sudo dpkg -r couchbase-server
```

Refer to the Ubuntu documentation for more information about uninstalling packages using `dpkg`.

You may need to delete the data files associated with your installation. The default installation location is `/opt`. If you selected an alternative location for your data files, you will need to separately delete each data directory from your system.

Related Links

Uninstalling on page 76

Uninstalling on Windows

To uninstall Couchbase software on a Windows system, you must have Administrator or Power User privileges.

To remove:

1. Navigate **Start > Settings > Control Panel**
2. Select **Add or Remove Programs**.
3. Remove the Couchbase Server software.

Related Links

Uninstalling on page 76

Upgrading

The following are the officially supported upgrade paths for Couchbase Server for both online upgrades or offline upgrades:

- Couchbase 1.8.1 to Couchbase 2.0 or higher
- Couchbase 2.0.x to Couchbase 2.1 or higher
- Couchbase 2.1.x to Couchbase 2.2 or higher
- Couchbase 2.2 to Couchbase 2.5 or higher

You cannot perform a direct upgrade from Couchbase Server 1.8.0 to a higher release of Couchbase Server. You must first upgrade from Couchbase Server 1.8 or earlier to Couchbase Server 1.8.1 to provide data compatibility with a higher release of Couchbase Server. After performing this initial upgrade, upgrade to a higher release of Couchbase Server.

You can perform a cluster upgrade in two ways:

- Online upgrades
- Offline upgrades

Prior to upgrade, perform a backup of the files.

Platform	Location
Linux	`/opt/couchbase/var/lib/couchbase/config/config.dat`
Windows	`C:\Program Files\Couchbase\Server\Config\var\lib\couchbase\config\config.dat`

Upgrading from Community Edition to Enterprise Edition

Use the same Couchbase Server version number when upgrading to the Enterprise Edition. Version differences may result in a failed upgrade.

Related Links

Couchbase Installation on page 48
Online upgrades on page 79
Offline upgrade on page 79
Upgrading individual nodes on page 81
Upgrading with XDCR on page 81
The upgrade process depends on the scenario.

Upgrading from 1.8 on page 82
To upgrade from version 1.8.0, upgrade to version 1.8.1 first.

Upgrading to 2.5 on page 83
Upgrading on Mac on page 84
Upgrading on RHEL on page 84
Upgrading on Ubuntu on page 84
Upgrading on Windows on page 85

Related topics

cbbackup tool on page 357
The `cbbackup` tool creates a copy of data from an entire running cluster, an entire bucket, a single node, or a single bucket on a single functioning node. Your node or cluster needs to be functioning in order to create the backup. Couchbase Server will write a copy of data onto disk.

cbrestore tool on page 390

Online upgrades

You can upgrade your cluster without taking your cluster down and so your application keeps running during the upgrade process. There are two ways you can perform this process: as a standard online upgrade, or as a swap rebalance. We highly recommend using a swap rebalance for online upgrade so that cluster capacity is always maintained. The standard online upgrade should only be used if swap rebalance is not possible.

Using the standard online upgrade, you take down one or two nodes from a cluster, and rebalance so that remaining nodes handle incoming requests. This is an approach you use if you have enough remaining cluster capacity to handle the nodes you remove and upgrade. You will need to perform rebalance twice for every node you upgrade: the first time to move data onto remaining nodes, and a second time to move data onto the new nodes.

Standard online upgrades may take a while because each node must be taken out of the cluster, upgraded to a current version, brought back into the cluster, and then rebalanced. However since you can upgrade the cluster without taking the cluster down, you may prefer this upgrade method.

For swap rebalance, you add a node to the cluster then perform a swap rebalance to shift data from an old node to a new node. You might prefer this approach if you do not have enough cluster capacity to handle data when you remove an old node. This upgrade process is also much quicker than performing a standard online upgrade because you only need to rebalance each upgraded node once.

Related Links

Upgrading on page 78

Related topics

Rebalancing on page 158

Offline upgrade

An offline upgrade must be well-planned and scheduled. For offline upgrades, you shut down your application first so that no more incoming data arrives. Then you verify the disk write queue is 0 then shut down each node. This way you know that Couchbase Server has stored all items onto disk from during shutdown. You then perform an install of the latest version of Couchbase onto the machine. The installer will automatically detect the files from the older install and convert them to the correct format, if needed.

Offline upgrades can take less time than online upgrades because you can upgrade every node in the cluster at once. The cluster must be shut down for the upgrade to take place. Both the cluster and all the applications built on it will not be available during this time.

Feature	Online Upgrades	Offline Upgrades
Applications Remain Available	Yes	No
Cluster Stays in Operation	Yes	No
Cluster must be Shutdown	No	Yes
Time Required	Requires Rebalance, Upgrade, Rebalance per Node	All nodes in Cluster Upgraded at Once

Remember: Before performing an upgrade, whether it is online or offline, backup your data.

Offline upgrade process

The offline upgrade process requires that all the applications and the entire Couchbase Server cluster be shutdown. After shutting down applications and nodes, perform the upgrade the software on each machine, and bring your cluster and application back up again.

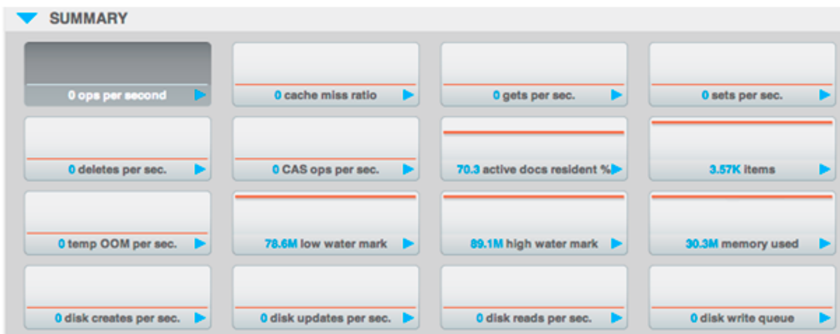
Note: If you are upgrading from Couchbase Server 1.8 to Couchbase Server 2.0 or higher, there are more steps for the upgrade because you must first upgrade to Couchbase 1.8.1 for data compatibility.

Tip: Check that the disk write queue is completely drained to ensure all data has been persisted to disk and will be available after the upgrade. Couchbase recommends that you turn off your application and allow the queue to drain before upgrading. Couchbase also recommends all data be backed up before upgrading.

To perform an offline upgrade:

1. Under Settings | Auto-Failover, disable auto-failover for all nodes in the cluster. If you leave this enabled, the first node that you shut down will be auto-failed-over.
2. Shut down your application, so that no more requests go to Couchbase Cluster.

   You can monitor the activity of your cluster by using Couchbase Web Console. The cluster needs to finish writing all information to disk. This will ensure that when you restart your cluster, all of your data can be brought back into the caching layer from disk. You can do this by monitoring the Disk Write Queue for every bucket in your cluster. The disk write queue should reach zero; this means no data remains to be written to disk.
3. Open Web Console at a node in your cluster.
4. Click Data Buckets | *your_bucket*. In the Summary section, check that `disk write queue` reads 0. If you have more than one data bucket for your cluster, repeat this step to check each bucket has a disk write queue of 0.



5. Create a backup of your cluster data using `cbbackup`.
6. Shutdown Couchbase Server on each machine in your cluster.
7. After you shutdown your nodes, perform a standard node upgrade to the new version of Couchbase Server.

   Couchbase Server starts automatically on each node after you perform the node upgrade.
8. As the cluster warms up, you can monitor the status of the warmup process to determine when you can switch on your application.

Once the cluster finishes warmup, you can re-enable your application on the upgraded cluster.

Offline upgrade to Enterprise Edition

Shutdown the entire cluster, and uninstall Couchbase Server Community Edition from each machine. Then install Couchbase Server Enterprise Edition. The data files will be retained, and the cluster can be restarted.

Related Links

Upgrading on page 78

Related topics

Backup and restore on page 150
The predominant method for backing up and restoring data is using the CLI `cbbackup` and `cbrestore` tools.

Server warmup on page 110
Whenever the Couchbase server is restarted or data is restored to a server instance, the server undergoes a warmup process before data requests can be handled.

Monitoring startup (warmup) on page 177
Disk write queue on page 178
Disk writing is implemented as a 2-queue system: commit to DRAM and then queued to be written to disk

cbbackup tool on page 357

The `cbbackup` tool creates a copy of data from an entire running cluster, an entire bucket, a single node, or a single bucket on a single functioning node. Your node or cluster needs to be functioning in order to create the backup. Couchbase Server will write a copy of data onto disk.

Auto-Failover tab on page 296

Upgrading individual nodes

Whether you are performing an online or offline upgrade, the steps for upgrading an individual nodes in a cluster remain the same.

1. Download Couchbase Server
2. Backup data for that node. To backup an existing Couchbase Server installation, use `cbbackup`.
3. Backup the node-specific configuration files. While the upgrade script perform a backup of the configuration and data files, as a best practice, make your own backup of these files.
4. Stop Couchbase Server.
5. Check your hostname configurations. If you have deployed Couchbase Server in a cloud service, or you are using hostnames rather than IP addresses, you must ensure that the hostname has been configured correctly before performing the upgrade.
6. Check for required components and if needed, install them. This ensures that Couchbase Server upgrades and migrates your existing data files.
7. Perform the installation upgrade for your platform:

Related Links

Upgrading on page 78

Related topics

Startup and shutdown on Linux on page 71

Startup and shutdown on Windows on page 71

Using hostnames on page 69

cbbackup tool on page 357

The `cbbackup` tool creates a copy of data from an entire running cluster, an entire bucket, a single node, or a single bucket on a single functioning node. Your node or cluster needs to be functioning in order to create the backup. Couchbase Server will write a copy of data onto disk.

Upgrading with XDCR

The upgrade process depends on the scenario.

The `xmem` replication mode performs replication on a destination cluster with the memcached protocol. This is the default mode for Couchbase Server replications. The `capi` mode performs replications over a REST protocol.

- `xmem` - only 2.2 servers and above support it.
- `capi` - both 2.2 and pre–2.2 servers support it.

The following prerequisites need to be considered:

- Network port 11210 needs to be open between nodes for 'xmem' mode of replication to work.
- In general, delete the replication, complete the upgrade, then recreate the replication.
- When upgrading Couchbase Server, make sure that both the source and destination clusters support the desired replication mode.
- To use XDCR data encryption with Secure Socket Layer (SSL), all nodes must be upgraded to at least Couchbase Server 2.5 Enterprise Edition.

Upgrade scenarios

- Both source and destination clusters are pre-2.2 and both are upgrading to pre-2.2 versions. This scenario is supported since both clusters use `capi`.

- Both source and destination clusters are pre-2.2 and the destination is upgrading to 2.2. This is a safe upgrade path since the source cluster communicates via `capi` to the destination.
- The source cluster is upgrading to 2.2 or higher and the destination is pre-2.2. This is not a safe upgrade path because the destination cannot receive replication via `xmem`. This results in incorrect data replication and failures in conflict resolution. For this scenario, upgrade both source and destination clusters to 2.2 or higher.
- Both source and destination clusters are upgrading from pre-2.2 to 2.2. This is not a safe upgrade path because the cluster upgrades are not synchronized. If the source upgrade completes prior to destination upgrade, incorrect data replication and failures in conflict resolution may occur. For this scenario:

  1. Delete all XDCR replications on your source cluster.
  2. Upgrade the source cluster to 2.2 or higher.
  3. Upgrade the destination cluster to 2.2 or higher.
  4. Re-create the XDCR replications using Version 2 for the XDCR protocol. Version 2 is `xmem` replication.

  Alternatively:

  1. Allow the rebalance upgrade to complete.
  2. Delete the Version 1 (`capi`) XDCR protocol replications.
  3. Create Version 2 (`xmem`) XDCR protocol replications.
- The source cluster is upgrading from pre–2.2 to 2.2 or higher and the destination cluster is Elastic Search. Since the source cluster must use `capi` for replication:

  1. Delete all XDCR replications on your source cluster.
  2. Upgrade the source cluster to 2.2 or higher.
  3. Re-create the XDCR replications using Version 1 for the XDCR protocol. Version 1 is `capi` replication.
- Both source and destination clusters are upgraded from pre-2.2 to 2.5 or higher. This is not a safe upgrade path because the cluster upgrades are not synchronized. If the source upgrade completes prior to destination upgrade, incorrect data replication and failures in conflict resolution may occur.

  1. Delete all XDCR replications on your source cluster.
  2. Upgrade the source cluster to 2.5 or higher.
  3. Upgrade the destination cluster to 2.5 or higher.
  4. Re-create the XDCR replications using Version 2 for the XDCR protocol. Version 2 is `xmem` replication.

Related Links

Upgrading on page 78

Upgrading from 1.8

To upgrade from version 1.8.0, upgrade to version 1.8.1 first.

Couchbase Server 1.8 cannot be upgraded directly to 2.0 or higher, instead, first upgrade to Couchbase Server 1.8.1 for data compatibility and then upgrade to a higher release.

Until all nodes in a cluster are upgraded from 1.8.1 or earlier, features in the new Couchbase Server release are disabled. This means views or XDCR do not function until all nodes in your cluster are migrated. After all nodes are upgraded, the features are enabled.

Important: If you plan to upgrade from 1.8.0 to 2.0 or higher, ensure that there is enough disk space available for both your original Couchbase Server 1.8 data files and the new format for the new Couchbase Server files. Provide approximately three (3) times the disk space for functionality such as indexing and compaction.

Upgrade from 1.8.1 notes

You can upgrade from Couchbase Server 1.8.1 to a higher release of Couchbase Server using either the online or offline upgrade method.

Use Online Upgrades from Couchbase Server 1.8.1

We recommend the online upgrade method when upgrading from 1.8.1 to a higher release of Couchbase Server. The process is quicker and can take place while clusters and applications are up and running. When upgrading from Couchbase Server 1.8.1 to a higher release of Couchbase Server, the data files are updated to use the new Couchstore data format instead of the SQLite format used in 1.8.1 and earlier. This increases the upgrade time, and requires additional disk space to support the migration.

Be aware that if you perform a scripted online upgrade from 1.8.1 to a higher release of Couchbase Server, expect a 10 second delay from adding a node to the cluster and rebalancing. If a rebalance request occurs too soon after adding a node, the rebalance may fail.

Upgrade from 1.8 and earlier notes

If you run Couchbase Server 1.8 or earlier, including Membase 1.7.2 and earlier, upgrade to Couchbase Server 1.8.1 first. The intermediate upgrade to Couchbase Server 1.8.1 allows the data files to be converted into the new Couchbase Server compatible formats. This conversion is only available when upgrading from 1.8.1 to a higher release of Couchbase Server.

To perform an offline upgrade, you use the standard installation system such as `dpkg`, `rpm` or Windows Setup Installer to upgrade the software on each machine. Each installer will perform the following operations:

```
* Shutdown Couchbase Server 1.8. Do not uninstall the server.

* Run the installer. The installer will detect any prerequisite software or
  components. An error is raised if the pre-requisites are missing. If you
 install
  additional required components such as OpenSSL during the upgrade, you must
  manually restart Couchbase after you install the components.
```

The installer copies 1.8.1-compatible data and configuration files to a backup location.

The `cbupgrade` program automatically starts. This will non-destructively convert data from the 1.8.1 database file format (SQLite) to the newer database file format (couchstore). The 1.8 database files are left “as-is”, and new database files are created. There must be enough disk space to handle this conversion operation (e.g., 3x more disk space).

Note: The data migration process from the old file format to the new file format may take some time. You should wait for the process to finish before starting Couchbase Server.

Once the upgrade process finishes, Couchbase Server starts automatically. Repeat this process on all nodes within your cluster.

Related Links

Upgrading on page 78

Upgrading to 2.5

This section covers upgrade behavior and important steps.

Before upgrading to 2.5

If buckets are using any of the following reserved ports, change the port for the bucket. Otherwise, XDCR data encryption is unavailable. (This applies to both offline and online upgrades.)

Important

Ensure that the Secure Socket Layer (SSL) reserved ports are available prior to using XDCR data encryption.

With XDCR data encryption, the following ports are reserved:

Port	Description
11214	Incoming SSL Proxy

Port	Description
11215	Internal Outgoing SSL Proxy
18091	Internal REST HTTPS for SSL
18092	Internal CAPI HTTPS for SSL

During a 2.5 upgrade

If Couchbase Server 2.5 has more than two (2) replicas, the first swap rebalance takes additional time. This behavior is expected.

Related Links

Upgrading on page 78

Upgrading on Mac

There is currently no officially supported upgrade installer for Mac OS X. If you want to migrate Couchbase Server on OS X, backup your data files with `cbbackup`, install the latest version, then restore your data with `cbrestore`.

Related Links

Upgrading on page 78

Related topics

cbbackup tool on page 357

The `cbbackup` tool creates a copy of data from an entire running cluster, an entire bucket, a single node, or a single bucket on a single functioning node. Your node or cluster needs to be functioning in order to create the backup. Couchbase Server will write a copy of data onto disk.

cbrestore tool on page 390

Upgrading on RHEL

For RHEL or CentoOS, you can perform an upgrade install using the RPM package — this will keep the data and existing configuration.

```
rpm -U couchbase-server-architecture___meta_current_version__.rpm
```

Linux upgrade from 1.8.1 notes

When you upgrade from Couchbase Server 1.8 to a higher release of Couchbase Server on Linux, be aware of the **OpenSSL** requirement. OpenSSL is a required component and an error message occurs during upgrade if it is not installed. To install it Red Hat-based systems, use `yum`:

```
root-> yum install openssl098e
```

Related Links

Upgrading on page 78

Upgrading on Ubuntu

You can perform a package upgrade for Ubuntu/Debian Linux by installing the updated `.pkg` package:

```
> sudo dpkg -i couchbase-server-architecture___meta_current_release.deb
```

Linux upgrade from 1.8.1 notes

When you upgrade from Couchbase Server 1.8 to a higher release of Couchbase Server on Linux, be aware of the **OpenSSL** requirement. OpenSSL is a required component and an error message occurs during upgrade if it is not installed.

On Debian-based systems, use `apt-get` to install the required OpenSSL package:

```
> sudo apt-get install libssl0.9.8
```

Related Links

Upgrading on page 78

Upgrading on Windows

The Install Wizard will upgrade your server installation using the same installation location. For example, if you have installed Couchbase Server in the default location, `C:\Program Files\Couchbase\Server`, the Couchbase Server installer will put the latest version at the same location.

Windows upgrade from 1.8.1 notes

If you have configured your Couchbase Server nodes to use hostnames, rather than IP addresses, to identify themselves within the cluster, ensure that the IP and hostname configuration is correct both before the upgrade and after upgrading the software.

Related Links

Upgrading on page 78

Related topics

Using hostnames on page 69

Migrating

Couchbase Server is based on components from both Membase Server and CouchDB. If you are a user of these database systems, or are migrating from these to Couchbase Server, the following information may help in translating your understanding of the main concepts and terms.

Related Links

Couchbase Installation on page 48
Migrating for CouchDB users on page 86
Migrating for Membase users on page 87

Migrating for CouchDB users

Although Couchbase Server incorporates the view engine functionality built into CouchDB, the bulk of the rest of the functionality is supported through the components and systems of Membase Server.

This change introduces a number of significant differences for CouchDB users that want to use Couchbase Server, particularly when migrating existing applications. However, you also gain the scalability and performance advantages of the Membase Server components.

Term and concept differences

Within CouchDB information is stored into the database using the concept of a document ID (either explicit or automatically generated), against which the document (JSON) is stored. Within Couchbase, there is no document ID, instead information is stored in the form of a key/value pair, where the key is equivalent to the document ID, and the value is equivalent to the document. The format of the data is the same.

Almost all of the HTTP REST API that makes up the interface for communicating with CouchDB does not exist within Couchbase Server. The basic document operations for creating, retrieving, updating and deleting information are entirely supported by the memcached protocol.

Also, beyond views, many of the other operations are unsupported at the client level within CouchDB. For example, you cannot create a new database as a client, store attachments, or perform administration-style functions, such as view compaction.

Couchbase Server does not support the notion of databases, instead information is stored within logical containers called Buckets. These are logically equivalent and can be used to compartmentalize information according to projects or needs. With Buckets you get the additional capability to determine the number of replicas of the information, and the port and authentication required to access the information.

Consistent functionality

The operation and interface for querying and creating view definitions in Couchbase Server is mostly identical. Views are still based on the combination of a map/reduce function, and you should be able to port your map/reduce definitions to Couchbase Server without any issues. The main difference is that the view does not output the document ID, but, as previously noted, outputs the key against which the key/value was stored into the database.

Querying views is also the same, and you use the same arguments to the query, such as a start and end docids, returned row counts and query value specification, including the requirement to express your key in the form of a JSON value if you are using compound (array or hash) types in your view key specification. Stale views are also supported, and just as with CouchDB, accessing a stale view prevents Couchbase Server from updating the index.

Changed functionality

There are many changes in the functionality and operation of Couchbase Server than CouchDB, including:

```
* Basic data storage operations must use the memcached API.

* Explicit replication is unsupported. Replication between nodes within a
 cluster
```

```
  is automatically configured and enabled and is used to help distribute
  information around the cluster.

* You cannot replicate between a CouchDB database and Couchbase Server.

* Explicit attachments are unsupported, but you can store additional files as
 new
  key/value pairs into the database.

* CouchApps are unsupported.

* Update handlers, document validation functions, and filters are not
 supported.

* Futon does not exist, instead there is an entire Web Administration Console
  built into Couchbase Server that provides cluster configuration, monitoring
 and
  view/document update functionality.
```

Operational and deployment differences

From a practical level the major difference between CouchDB and Couchbase Server is that options for clustering and distribution of information are significantly different. With CouchDB you would need to handle the replication of information between multiple nodes and then use a proxy service to distribute the load from clients over multiple machines.

With Couchbase Server, the distribution of information is automatic within the cluster, and any Couchbase Server client library will automatically handle and redirect queries to the server that holds the information as it is distributed around the cluster. This process is automatic.

Client and application changes

As your CouchDB based application already uses JSON for the document information, and a document ID to identify each document, the bulk of your application logic and view support remain identical. However, the HTTP REST API for basic CRUD operations must be updated to use the memcached protocol.

Additionally, because CouchApps are unsupported you will need to develop a client side application to support any application logic.

Related Links

Migrating on page 86

Migrating for Membase users

For an existing Membase user the primary methods for creating, adding, manipulating and retrieving data remain the same. In addition, the background operational elements of your Couchbase Server deployment will not differ from the basic running of a Membase cluster.

Term and concept differences

The following terms are new, or updated, in Couchbase Server:

```
* `Views`, and the associated terms of the `map` and `reduce` functions used
 to
  define views. Views provide an alternative method for accessing and querying
  information stored in key/value pairs within Couchbase Server. Views allow
 you
  to query and retrieve information based on the values of the contents of a
  key/value pair, providing the information has been stored in JSON format.

* *JSON (JavaScript Object Notation)*, a data representation format that is
  required to store the information in a format that can be parsed by the View
```

```
  system is new.

* *Membase Server* is now *Couchbase Server*.

* *Membase Buckets* are now *Couchbase Buckets*.
```

Consistent functionality

The core functionality of Membase, including the methods for basic creation, updating and retrieval of information all remain identical within Couchbase Server. You can continue to use the same client protocols for setting and retrieving information.

The administration, deployment, and core of the web console and administration interfaces are also identical. There are updates and improvements to support additional functionality which is included in existing tools. These include View-related statistics, and an update to the Web Administration Console for building and defining views.

Changed functionality

The main difference of Couchbase Server is that in addition to the key/value data store nature of the database, you can also use Views to convert the information from individual objects in your database into lists or tables of records and information. Through the view system, you can also query data from the database based on the value (or fragment of a value) of the information that you have stored in the database against a key.

This fundamental differences means that applications no longer need to manually manage the concept of lists or sets of data by using other keys as a lookup or compounding values.

Operational and deployment differences

The main components of the operation and deployment of your Couchbase Server remain the same as with Membase Server. You can add new nodes, failover, rebalance and otherwise manage your nodes as normal.

However, the introduction of Views means that you will need to monitor and control the design documents and views that are created alongside your bucket configurations. Indexes are generated for each design document (i.e. multiple views), and for optimum reliability you may want to backup the generated index information to reduce the time to bring up a node in the event of a failure, as building a view from raw data on large datasets may take a significant amount of time.

In addition, you will need to understand how to recreate and rebuild View data, and how to compact and clean-up view information to help reduce disk space consumption and response times.

Client and application ahanges

Clients can continue to communicate with Couchbase Server using the existing memcached protocol interface for the basic create, retrieve, update and delete operations for key/value pairs. However, to access the View functionality you must use a client library that supports the view API (which uses HTTP REST).

To build Views that can output and query your stored data, your objects must be stored in the database using the JSON format. This may mean that if you have been using the native serialization of your client library to convert a language specific object so that it can be stored into Membase Server, you will now need to structure your data and use a native to JSON serialization solution, or reformat your data so that it can be formatted as JSON.

Related Links

Migrating on page 86

Couchbase Administration

The Couchbase Server 3.0 administration guide provides the following topics:

Related Links

Couchbase Server concepts include the differenct components and systems that make up an individual Couchbase Server instance and a Couchbase cluster including information and concepts needed to understand the fast and elastic nature, high availability, and high performance of the Couchbase Server database.

Deployment configuration take into account topics such as restricted access, node communication, swap configuration, and connection timeouts.

There are a number of different ways to monitor Couchbase servers including underlying processes, ports, and queueing.

The Couchbase web console is the main tool for managing the Couchbase environment.

Troubleshooting covers general tips, common errors, log information, and other issues.

Administration basics

Couchbase administration is primarily supported through the following methods:

- Web Console
- Command-line interface
- REST API

Accessing Couchbase Server directly

To access Couchbase Server and use the CLI or REST API directly, log into the server directly (depending on the operating system). To access Couchbase Server and use the Web Console, log into the server through a browser such as Firefox, Chrome, Safari, or Internet Explorer.

Accessing Couchbase Server through a client

If you already have an application that uses the Memcached protocol then you can start using your Couchbase Server immediately. If so, you can simply point your application to this server like you would any other memcached server. No code changes or special libraries are needed, and the application will behave exactly as it would against a standard memcached server. Without the client knowing anything about it, the data is being replicated, persisted, and the cluster can be expanded or contracted completely transparently.

If you do not already have an application, then you should investigate one of the available Couchbase client libraries to connect to your server and start storing and retrieving information.

Related Links

Couchbase Administration on page 89

Administration tools on page 90
The administration tools are the Web Console, Command-line interface (CLI) and REST API.

Common administrative tasks on page 91

Starting and stopping Couchbase on page 92
Summary of starting and stopping Couchbase on Linux (RHEL and Ubuntu), Windows, and Mac.

Data file location on page 92
Couchbase Server stores data files (database and indices) under the **var** > **lib** > **couchbase** > **data** path.

Installation location on page 94
The default installation location for Couchbase Server depends on the operating system.

Limits on page 94
Couchbase Server limits and limitations may affect server usage and implementation.

Related topics

Supported web browsers on page 53
Mozzila Firefox, Safari, Google Chrome, and IE are supported

Command-line interface overview on page 353

REST API overview on page 426

Deployment considerations on page 119
Deployment configuration take into account topics such as restricted access, node communication, swap configuration, and connection timeouts.

Couchbase SDKs

Administration tools

The administration tools are the Web Console, Command-line interface (CLI) and REST API.

Couchbase Server was designed to be as easy to use as possible, and does not require constant attention. Administration is however offered in a number of different tools and systems.

Couchbase Server provides the following solutions for managing and monitoring your Couchbase Server and cluster:

- **Web Console**

  Couchbase Server includes a built-in web-administration console that provides a complete interface for configuring, managing, and monitoring your Couchbase Server installation.
- **Command-line Interface (CLI)**

  Couchbase Server includes a suite of command-line tools that provide information and control over your Couchbase Server and cluster installation. These can be used in combination with your own scripts and management procedures to provide additional functionality, such as automated failover, backups and other procedures. The command-line tools make use of the REST API.
- **REST API**

  In addition to the Web Administration console, Couchbase Server incorporates a management interface exposed through the standard HTTP REST protocol. This REST interface can be called from your own custom management and administration scripts to support different operations.

Related Links

Administration basics on page 90

Related topics

Common administrative tasks on page 91

Command-line interface overview on page 353

REST API overview on page 426

Common administrative tasks

For general running and configuration, Couchbase Server is self-managing. The management infrastructure and components of the Couchbase Server system are able to adapt to the different events within the cluster. There are only a few different configuration variables. The majority of these configuration variables do not need to be modified or altered in most installations.

However, there are a number of different tasks that are performed over the lifetime of the cluster environment including:

- Expand your cluster when you need to expand the RAM or disk I/O capabilities.
- Failover and altering the size of your cluster as your application demands change.
- Monitoring and reacting to the various statistics reported by the server to ensure that your cluster is operating at the highest performance level.
- Backing up the cluster.

Increasing or reducing your cluster size

When your cluster requires additional RAM, disk I/O or network capacity, you will need to expand the size of your cluster. If the increased load is only a temporary event, then you may later want to reduce the size of your cluster.

You can add or remove multiple nodes from your cluster at the same time. Once the new node arrangement has been configured, the process redistributing the data and bringing the nodes into the cluster is called *rebalancing*. The rebalancing process moves the data around the cluster to match the new structure, and can be performed live while the cluster is still servicing application data requests.

Warming up a server

There may be cases where you want to explicitly shutdown a server and then restart it. Typically the server had been running for a while and has data stored on disk when you restart it. In this case, the server needs to undergo a warmup process before it can again serve data requests.

Handling a failover situation

A failover situation occurs when one of the nodes within your cluster fails, usually due to a significant hardware or network problem. Couchbase Server is designed to cope with this situation through the use of replicas which provide copies of the data around the cluster which can be activated when a node fails.

Couchbase Server provides two mechanisms for handling failover. Automated Failover allows the cluster to operate autonomously and react to failovers without human intervention. Monitored failover enables you to perform a controlled failure by manually failing over a node. There are additional considerations for each failover type, and you should read the notes to ensure that you know the best solution for your specific situation.

Managing database and view fragmentation

The database and view index files created by Couchbase Server can become fragmented. This can cause performance problems, as well as increasing the space used on disk by the files, compared to the size of the information they hold. Compaction reduces this fragmentation to reclaim the disk space.

Backing up and restoring your cluster data

Couchbase Server automatically distributes your data across the nodes within the cluster, and supports replicas of that data. It is good practice, however, to have a backup of your bucket data in the event of a more significant failure.

Related Links

Administration basics on page 90

Related topics

Command-line interface overview on page 353

REST API overview on page 426

Starting and stopping Couchbase

Summary of starting and stopping Couchbase on Linux (RHEL and Ubuntu), Windows, and Mac.

Table 10: Starting and stopping Couchbase

Operating system	Start	Stop
Linux	`sudo /etc/init.d/couchbase-server start`	`sudo /etc/init.d/couchbase-server stop`
Windows	`net start CouchbaseServer`	`net stop CouchbaseServer`
	`C:\Program Files\Couchbase\Server\bin\service_start.bat`	`C:\Program Files\Couchbase\Server\bin\service_stop.bat`
	Windows Task Manager > Services > CouchbaseServer service (right-click) > Start	**Windows Task Manager > Services > CouchbaseServer service (right-click) > Stop**
	Start > Run > type Services.msc > CouchbaseServer service (right-click) > Start	**Start > Run > type Services.msc > CouchbaseServer service (right-click) > Stop**
Mac	Double-click on the Couchbase application	From the Couchbase application, select **Quit Couchbase Server**.

Related Links

Administration basics on page 90

Data file location

Couchbase Server stores data files (database and indices) under the **var > lib > couchbase > data** path.

The disk path for the database and indices files is set during the initial setup of the server node. The default disk path is typically used for development purposes only. If the server node is used for production, configure a different disk path.

Platform	Default directory
Linux	`/opt/couchbase/var/lib/couchbase/data`
Windows	`C:\Program Files\couchbase\server\var\lib\couchbase\data`
Mac OS X	`~/Library/Application Support/Couchbase/var/lib/couchbase/data`

Changing the data file path

The disk path where the data and index files are stored cannot be changed on a running server. To change the disk path, the node must be removed from the cluster, configured with the new path, and added back to the cluster. The data file path can be changed for each node via the Web UI at setup, REST API, or CLI. Once a node or cluster has already been setup and is storing data, the path cannot be changed while the node is part of a running cluster.

The quickest and easiest method is to provision a new node with the correct disk path configured and then use swap rebalance to add the new node in while taking the old node out. This ensures that cluster performance is not impacted.

To change the disk path by replacing a node (with swap rebalance):

1. Setup a new node with a different disk path.
2. Swap rebalance the new node for the existing node.
3. Repeat the process for every node in the cluster.

To change the disk path of an existing node (without swap rebalance):

1. Remove the node from the cluster and rebalance.
2. Change the path on the running node either via the REST API or using the Couchbase CLI (commands above).
3. Re-add the node back to the cluster and rebalance.

To change the disk path on multiple nodes, swap out each node and change the disk path individually.

Important: Changing the data path for a node that is already part of a cluster permanently deletes the stored data.

Tip: When using the command line tool, the data file and index file path settings cannot be changed individually. To change the setting individually, use the REST API.

CLI example

```
couchbase-cli node-init -c 10.5.2.54:8091 \
 --node-init-data-path=new_path \
 -u user -p password
```

Related Links

Administration basics on page 90

Related topics

Swap rebalance on page 161
Swap rebalance is an automatic feature that optimizes the movement of data when you are adding and removing the same number of nodes within the same operation.

Adding nodes on page 164

Performing a rebalance on page 163
Once you have configured the nodes that you want to add or remove from your cluster, you must perform a rebalance operation.

Rebalancing a cluster on page 162
Rebalancing a cluster involves marking nodes to be added or removed from the cluster, and then starting the rebalance operation so that the data is moved around the cluster to reflect the new structure.

Setting index paths on page 449
The path for the index files can be configured with `POST /nodes/self/controller/settings`.

Removing nodes from clusters on page 436

REST API overview on page 426

Server nodes REST API on page 444
A Couchbase Server instance, known as node, is a physical or virtual machine running Couchbase Server. Each node is as a member of a cluster.

Installation location

The default installation location for Couchbase Server depends on the operating system.

The default, Couchbase Server installs under the following locations:

Platform	Directory
Linux	`/opt/couchbase`
Windows	`C:\Program Files\Couchbase Server\`
Mac OS X	`/Applications/Couchbase Server.app/Contents/Resources/couchbase-core/bin/`

Related Links

Administration basics on page 90

Limits

Couchbase Server limits and limitations may affect server usage and implementation.

Limit	Value
Max key length	250 bytes
Max value size	20 Mbytes
Max data size	none
Max metadata	Approximately 150 bytes per document
Max Buckets per Cluster	10
Max View Key Size	4096 bytes

Related Links

Administration basics on page 90

Architecture and concepts

Couchbase Server concepts include the differenct components and systems that make up an individual Couchbase Server instance and a Couchbase cluster including information and concepts needed to understand the fast and elastic nature, high availability, and high performance of the Couchbase Server database.

Related Links

Couchbase Administration on page 89

Nodes and clusters on page 96
Couchbase Server can be used either in a standalone configuration or cluster configuration. A cluster configuration is multiple Couchbase Servers connected together to provide a single, distributed data store.

Cluster Manager on page 96
The Cluster Manager is responsible for node and cluster management. Every node within a Couchbase cluster includes the Cluster Manager component.

Data storage on page 97
Couchbase Server provides data management services using *buckets*. Buckets are isolated virtual containers for data. A bucket is a logical grouping of physical resources within a cluster of Couchbase Servers.

RAM quotas on page 99
RAM is allocated to Couchbase Server in the following configurable quantities: *Server Quota* and *Bucket Quota*.

vBuckets on page 100
A vBucket is defined as the *owner* of a subset of the key space of a Couchbase cluster. These vBuckets are used to distributed information effectively across a cluster.

Caching layer on page 102
Couchbase Server includes a built-in caching layer which acts as a central part of the server and provides very rapid reads and writes of data.

Disk storage on page 102
Couchbase Server mainly stores and retrieves information for clients using RAM. At the same time, Couchbase Server eventually stores all data to disk to provide a higher level of reliability.

Working set management and ejection on page 105
Working set management is the process of freeing up space and ensuring that the most used items are available in RAM. Ejection is the process of removing data from RAM to provide room for frequently used items.

Expiration on page 109
Each document stored in the database has an optional expiration value (TTL, time to live) that is used to automatically deleted items.

Server warmup on page 110
Whenever the Couchbase server is restarted or data is restored to a server instance, the server undergoes a warmup process before data requests can be handled.

Rebalancing on page 111
Rebalancing is the process of redistributing information across nodes

Replicas and replication on page 111
Replicas are copies of data that are proved on another node in a cluster.

Rack Awareness on page 112
The Rack Awareness feature allows logical groupings of servers on a cluster where each server group physically belongs to a rack or availability zone.

Failover on page 115
Failover is the process of indicating that a node in a cluster is unavailable and that enabling replica vBuckets.

TAP on page 116
The TAP protocol is an internal part of the Couchbase Server system and is used to exchange data throughout the system.

Statistics and monitoring on page 117
A complete set of statical and monitoring information are provided through the Web Console, CLI, and REST API.

Nodes and clusters

Couchbase Server can be used either in a standalone configuration or cluster configuration. A cluster configuration is multiple Couchbase Servers connected together to provide a single, distributed data store.

Couchbase Server or Node	A single instance of the Couchbase Server software running on a machine, whether a physical machine, virtual machine, EC2 instance or other environment. All instances of Couchbase Server are identical, provide the same functionality, interfaces, and systems, and consist of the same components.
Cluster	A cluster is a collection of one or more instances of Couchbase Server that are configured as a logical cluster. All nodes within the cluster are identical and provide the same functionality. Each node is capable of managing the cluster and each node can provide aggregate statistics and operational information about the cluster. User data is stored across the entire cluster through the vBucket system. Clusters operate in a completely horizontal fashion. To increase the size of a cluster, add another node. There are no parent/child relationships or hierarchical structures involved. This means that Couchbase Server scales linearly, both in terms of increasing the storage capacity and performance and scalability.

Related Links

Architecture and concepts on page 95
Couchbase Server concepts include the differenct components and systems that make up an individual Couchbase Server instance and a Couchbase cluster including information and concepts needed to understand the fast and elastic nature, high availability, and high performance of the Couchbase Server database.

Cluster Manager

The Cluster Manager is responsible for node and cluster management. Every node within a Couchbase cluster includes the Cluster Manager component.

The Cluster Manager is responsible for the following within a cluster:

- Cluster management
- Node administration
- Node monitoring
- Statistics gathering and aggregation
- Run-time logging
- Multi-tenancy
- Security for administrative and client access
- Client proxy service to redirect requests

Access to the Cluster Manager is provided through the administration interface on a dedicated network port and through dedicated network ports for client access. Additional ports are configured for inter-node communication.

Related Links

Architecture and concepts on page 95

Couchbase Server concepts include the differenct components and systems that make up an individual Couchbase Server instance and a Couchbase cluster including information and concepts needed to understand the fast and elastic nature, high availability, and high performance of the Couchbase Server database.

Data storage

Couchbase Server provides data management services using *buckets*. Buckets are isolated virtual containers for data. A bucket is a logical grouping of physical resources within a cluster of Couchbase Servers.

Buckets provide a secure mechanism for organizing, managing, and analyzing data storage resources. Two types of data buckets, memcached and couchbase, enable you to store data either in-memory only or both in-memory and on disk (for added reliability). During Couchbase Server set up, the type of bucket that you need for your implementation is selected.

Note: Buckets can be used by multiple client applications across a cluster.

Bucket Type	Description
Couchbase	Provides highly-available and dynamically reconfigurable distributed data storage, providing persistence and replication services. Couchbase buckets are 100% protocol compatible with, and built in the spirit of, the memcached open source distributed key-value cache.
Memcached	Provides a directly-addressed, distributed (scale-out), in-memory, key-value cache. Memcached buckets are designed to be used alongside relational database technology – caching frequently-used data, thereby reducing the number of queries a database server must perform for web servers delivering a web application.

The different bucket types support different capabilities.

Capability	memcached Buckets	Couchbase Buckets
Item Size Limit	1 MByte	20 MByte
Persistence	No	Yes
Replication	No	Yes
Rebalance	No	Yes
Statistics	Limited set for in-memory stats	Full suite
Client Support	Memcached, should use Ketama consistent hashing	Full Smart Client Support

Couchbase-type buckets provide a highly-available and dynamically reconfigurable distributed data store, survive node failures, and allow cluster reconfiguration while continuing to service requests. Couchbase-type buckets provide the following core capabilities:

Couchbase bucket capability	Description
Caching	Couchbase buckets operate through RAM. Data is kept in RAM and persisted down to disk. Data will be cached in RAM until the configured RAM is exhausted, when data is ejected from RAM. If requested data is not currently in the RAM cache, it will be loaded automatically from disk.
Persistence	Data objects can be persisted asynchronously to hard-disk resources from memory to provide protection from server restarts or minor failures. Persistence properties are set at the bucket level.
Replication	A configurable number of replica servers can receive copies of all data objects in the Couchbase-type bucket. If the host machine fails, a replica server can be promoted to be the host server, providing high availability cluster operations via failover. Replication is configured at the bucket level.

Couchbase bucket capability	Description
Rebalancing	Rebalancing enables load distribution across resources and dynamic addition or removal of buckets and servers in the cluster.

The following bucket interface types that can be configured. Both memcached or Couchbase buckets can be authenticated via SASL or not authenticated (non-SASL).

Default bucket

The default bucket is a Couchbase bucket that always resides on port 11211 and is a non-SASL authenticating bucket. When Couchbase Server is first installed this bucket is automatically set up during installation. This bucket may be removed after installation and may also be re-added later, but when re-adding a bucket named “default”, the bucket must be place on port 11211 and must be a non-SASL authenticating bucket. A bucket not named default may not reside on port 11211 if it is a non-SASL bucket. The default bucket may be reached with a vBucket aware smart client, an ASCII client or a binary client that doesn’t use SASL authentication.

Non-SASL buckets

Non-SASL buckets may be placed on any available port with the exception of port 11211 if the bucket is not named “default”. Only one Non-SASL bucket may placed on any individual port. These buckets may be reached with a vBucket aware smart client, an ASCII client or a binary client that doesn’t use SASL authentication.

SASL buckets

SASL authenticating Couchbase buckets may only be placed on port 11211 and each bucket is differentiated by its name and password. SASL bucket may not be placed on any other port beside 11211. These buckets can be reached with either a vBucket aware smart client or a binary client that has SASL support. These buckets cannot be reached with ASCII clients.

Smart clients discover changes in the cluster using the Couchbase Management REST API. Buckets can be used to isolate individual applications to provide multi-tenancy or to isolate data types in the cache to enhance performance and visibility. Couchbase Server allows you to configure different ports to access different buckets, and provides the option to access isolated buckets using either the binary protocol with SASL authentication or the ASCII protocol with no authentication

Couchbase Server allows you to use and mix different types of buckets, Couchbase and Memcached, in your environment. Buckets of different types still share the same resource pool and cluster resources. Quotas for RAM and disk usage are configurable per bucket so that resource usage can be managed across the cluster. Quotas can be modified on a running cluster so that administrators can reallocate resources as usage patterns or priorities change over time.

Related Links

Architecture and concepts on page 95
Couchbase Server concepts include the differenct components and systems that make up an individual Couchbase Server instance and a Couchbase cluster including information and concepts needed to understand the fast and elastic nature, high availability, and high performance of the Couchbase Server database.

Related topics

Command-line interface overview on page 353

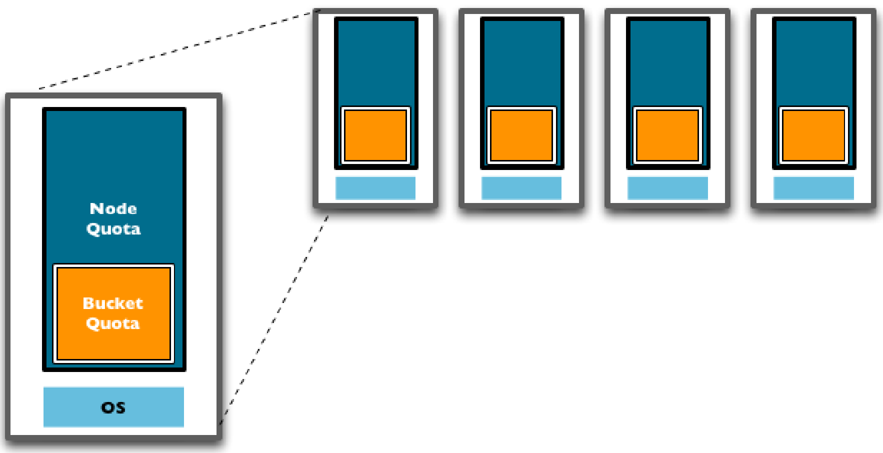
Buckets REST API on page 456
The bucket management and configuration REST API endpoints provide a fine level of control over the individual buckets in the cluster, their configuration, and specific operations.

RAM quotas

RAM is allocated to Couchbase Server in the following configurable quantities: *Server Quota* and *Bucket Quota*.

Server quota	The Server Quota is the RAM that is allocated to the server when Couchbase Server is first installed. This sets the limit of RAM allocated by Couchbase for caching data *for all buckets* and is configured on a per-node basis. The Server Quota is initially configured in the first server in your cluster is configured, and the quota is identical on all nodes. For example, if you have 10 nodes and a 16GB Server Quota, there is 160GB RAM available across the cluster. If you were to add two more nodes to the cluster, the new nodes would need 16GB of free RAM, and the aggregate RAM available in the cluster would be 192GB.
Bucket quota	The Bucket Quota is the amount of RAM allocated to an individual bucket for caching data. Bucket Quotas are configured on a per-node basis, and is allocated out of the RAM defined by the Server Quota. For example, if you create a new bucket with a Bucket Quota of 1GB, in a 10 node cluster there would be an aggregate bucket quota of 10GB across the cluster. Adding two nodes to the cluster would extend your aggregate bucket quota to 12GB.

The following diagram shows that adding new nodes to the cluster expands the overall RAM quota and the bucket quota, increasing the amount of information that can be kept in RAM.



Bucket Quota is used by the system to determine when data should be ejected from memory. Bucket Quotas are dynamically configurable, within the Server Quota limits, and enable individual control of information cached in memory on a per bucket basis. Therefore, buckets can be configured differently depending your caching RAM allocation requirements.

Note: The Server Quota is also dynamically configurable, however, ensure that the cluster nodes have the available RAM to support the chosen RAM quota configuration.

Related Links

Architecture and concepts on page 95
Couchbase Server concepts include the differenct components and systems that make up an individual Couchbase Server instance and a Couchbase cluster including information and concepts needed to understand the fast and elastic nature, high availability, and high performance of the Couchbase Server database.

Related topics

Working set management and ejection on page 105
Working set management is the process of freeing up space and ensuring that the most used items are available in RAM. Ejection is the process of removing data from RAM to provide room for frequently used items.

vBuckets

A vBucket is defined as the *owner* of a subset of the key space of a Couchbase cluster. These vBuckets are used to distributed information effectively across a cluster.

The vBucket system is used both for distributing data and for supporting replicas (copies of bucket data) on more than one node. vBuckets are not a user-accessible component, but they are a critical component of Couchbase Server and are vital to the availability support and elastic nature.

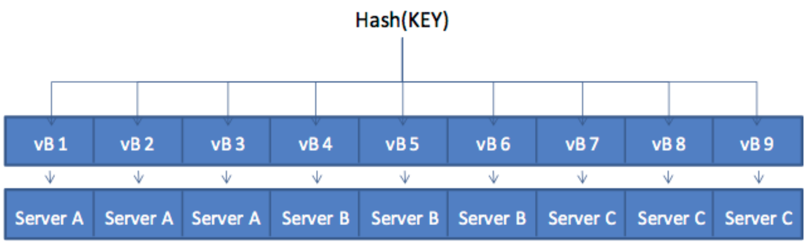
Clients access the information stored in a bucket by communicating directly with the node responsible for the corresponding vBucket. This direct access enables clients to communicate with the node storing the data, rather than using a proxy or redistribution architecture. The result abstracts the physical topology from the logical partitioning of data. This architecture gives Couchbase Server elasticity and flexibility

Every document ID belongs to a vBucket. A mapping function is used to calculate the vBucket in which a given document belongs. In Couchbase Server, that mapping function is a hashing function that takes a document ID as input and outputs a vBucket identifier. Once the vBucket identifier has been computed, a table is consulted to lookup the server that "hosts" that vBucket. The table contains one row per vBucket, pairing the vBucket to its hosting server. A server appearing in this table can be (and usually is) responsible for multiple vBuckets.

The following diagrams shows how the Key to Server mapping (vBucket map) works.

In this scenario, there are three servers in the cluster and client wants to look up the value of KEY using the `GET` operation.

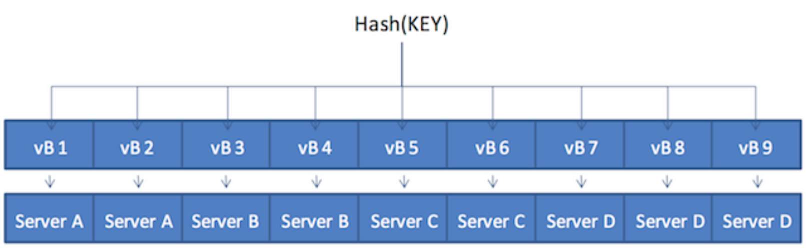
1. The client first hashes the key to calculate the vBucket which owns KEY. In this example, the hash resolves to vBucket 8 (vB8).
2. By examining the vBucket map, the client determines Server C hosts vB8.
3. The client sends the `GET` operation directly to Server C.



In the next scenario, a server added to the original cluster of three. A new node, Server D, is added to the cluster and the vBucket Map is updated (during the rebalance operation). The updated map is then sent to all the cluster participants including the other nodes, any connected "smart" clients, and the Moxi proxy service.

Within the new four-node cluster model, when a client again wants to determine the value of KEY using the `GET` operation:

- The hashing algorithm still resolves to vBucket 8 (vB8).
- The new vBucket Map now maps vBucket 8 to Server D.
- The client sends the `GET` operation directly to Server D.



Note: This architecture allows Couchbase Server to cope with changes without using the typical RDBMS sharding method. In addition, the architecture differs from the method used by `memcached`, which uses client-side key hashes to determine the server from a defined list. The memcached method requires active management of the list of servers and specific hashing algorithms such as Ketama to cope with changes to the topology.

Related Links

Architecture and concepts on page 95
Couchbase Server concepts include the differenct components and systems that make up an individual Couchbase Server instance and a Couchbase cluster including information and concepts needed to understand the fast and elastic nature, high availability, and high performance of the Couchbase Server database.

Related topics

Rebalancing on page 111
Rebalancing is the process of redistributing information across nodes

Caching layer

Couchbase Server includes a built-in caching layer which acts as a central part of the server and provides very rapid reads and writes of data.

Couchbase Server automatically manages the caching layer and coordinates with disk space to ensure that enough cache space exists to maintain performance. Couchbase Server automatically places items that come into the caching layer into disk queue so that it can write these items to disk. If the server determines that a cached item is infrequently used, it removes it from RAM to free space for other items. Similarly the server retrieves infrequently-used items from disk and stores them into the caching layer when the items are requested. In order to provide the most frequently-used data while maintaining high performance, Couchbase Server manages a *working set* of your entire information. The working set is the data most frequently accessed and is kept in RAM for high performance.

Couchbase automatically moves data from RAM to disk asynchronously, in the background, to keep frequently used information in memory and less frequently used data on disk. Couchbase constantly monitors the information accessed by clients and decides how to keep the active data within the caching layer. Data is ejected to disk from memory while the server continues to service active requests. During sequences of high writes to the database, clients are notified that the server is temporarily out of memory until enough items have been ejected from memory to disk. The asynchronous nature and use of queues in this way enables reads and writes to be handled at a very fast rate, while removing the typical load and performance spikes that would otherwise cause a traditional RDBMS to produce erratic performance.

When the server stores data on disk and a client requests the data, an individual document ID is sent and then the server determines whether the information exists or not. Couchbase Server does this with metadata structures. The metadata holds information about each document in the database and this information is held in RAM. This means that the server returns a ‘document ID not found’ response for an invalid document ID, returns the data from RAM, or returns the data after being fetched from disk.

Note: Other database solutions read and write data from disk, which results in much slower performance. One approach used by other database solutions is to install and manage a caching layer as a separate component which works with a database. This approach has drawbacks because of the significant custom code and effort due to the burden of managing the caching layer and the data transfers between the caching layer and database.

Related Links

Architecture and concepts on page 95
Couchbase Server concepts include the differenct components and systems that make up an individual Couchbase Server instance and a Couchbase cluster including information and concepts needed to understand the fast and elastic nature, high availability, and high performance of the Couchbase Server database.

Disk storage

Couchbase Server mainly stores and retrieves information for clients using RAM. At the same time, Couchbase Server eventually stores all data to disk to provide a higher level of reliability.

It writes data to the caching layer and puts the data into a disk write queue to be persisted to disk. Disk persistence enables you to perform backup and restore operations and to grow your datasets larger than the built-in caching layer. This disk storage process is called *eventual persistence* since the server does not block a client while it writes to disk.

If a node fails and all data in the caching layer is lost, the items are recovered from disk. When the server identifies an item that needs to be loaded from disk, because it is not in active memory, the process is handled by a background process that processes the load queue and reads the information back from disk and into memory. The client waits until the data has been loaded back into memory before the information is returned.

Related Links

Architecture and concepts on page 95
Couchbase Server concepts include the differenct components and systems that make up an individual Couchbase Server instance and a Couchbase cluster including information and concepts needed to understand the fast and elastic nature, high availability, and high performance of the Couchbase Server database.

Multiple readers and writers on page 103

Multi-threaded readers and writers provide multiple processes to simultaneously read and write data on disk. Simultaneous reads and writes increases disk speed and improves the read rate from disk.

Document deletion on page 104
Couchbase Server never deletes entire items from disk unless a client explicitly deletes the item from the database or the expiration value for the item is reached.

Tombstone purging on page 104
Tombstones are records of expired or deleted items that include item keys and metadata.

Related topics

Working set management and ejection on page 105
Working set management is the process of freeing up space and ensuring that the most used items are available in RAM. Ejection is the process of removing data from RAM to provide room for frequently used items.

Multiple readers and writers

Multi-threaded readers and writers provide multiple processes to simultaneously read and write data on disk. Simultaneous reads and writes increases disk speed and improves the read rate from disk.

Multiple readers and writers are supported to persist data onto disk. For earlier versions of Couchbase Server, each server instance had only single disk reader and writer threads. Disk speeds have now increased to the point where single read/write threads do not efficiently keep up with the speed of disk hardware. The other problem caused by single read/writes threads is that if you have a good portion of data on disk and not RAM, you can experience a high level of cache misses when you request this data.

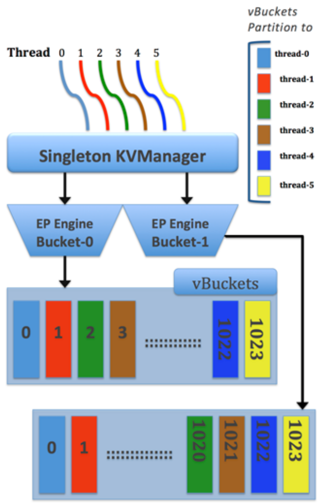
The multi-reader and writer setting depends on the following hardware in your Couchbase cluster:

Note: The recommended hardware requirements are quad-core processes on 64-bit CPU and 3GHz, 16GB RAM physical storage. Solid state drives are also recommended.

- If you deploy your cluster on the minimum hardware requirement which is dual-core CPUs running on 2GHz and 4GB of physical RAM, stay with the default setting of Low (which is three threads).
- If you deploy your servers on recommended hardware requirements or above, this setting can be increased to High (which is eight threads).
- If you have a hardware configuration which conforms to pre–2.1 hardware requirements, change this setting to the default.

Note: In a Couchbase Server cluster, since earlier version of Couchbase Server coexists with higher versions of Couchbase Server. Pre-2.1 nodes remain with single readers and writer for the data bucket. Post-2.1 nodes have multiple readers and writers. When server nodes are upgraded, the multiple readers and writers setting is implemented with bucket restart and warmup. In this case, install the new node, add it to the cluster, edit the existing bucket setting for readers and writers. After rebalancing the cluster, the new node performs reads and writes with multiple readers and writers and the data bucket does not restart or go through a warmup. All existing pre–2.1 nodes remain with a single readers and writers for the data bucket. As pre-2.1 nodes are upgraded and added to the cluster, these new nodes automatically pick up the setting and use multiple readers and writers for the bucket.

The multi-threaded engine includes additional synchronization among threads that are accessing the same data cache to avoid conflicts. To maintain performance while avoiding conflicts over data, Couchbase uses a form of locking between threads as well as thread allocation among vBuckets with static partitioning. When Couchbase Server creates multiple reader and writer threads, the server assesses a range of vBuckets for each thread and assigns each thread exclusively to certain vBuckets. With this static thread coordination, the server schedules threads so that only a single reader and single writer thread can access the same vBucket at any given time. The previous image shows six pre-allocated threads and two data Buckets. Each thread has the range of vBuckets that is statically partitioned for read and write access.



Related Links

Disk storage on page 102
Couchbase Server mainly stores and retrieves information for clients using RAM. At the same time, Couchbase Server eventually stores all data to disk to provide a higher level of reliability.

Document deletion

Couchbase Server never deletes entire items from disk unless a client explicitly deletes the item from the database or the expiration value for the item is reached.

The ejection mechanism removes an item from RAM, while keeping a copy of the key and metadata for that document in RAM and also keeping copy of that document on disk.

Important: If only memcached buckets are used with Couchbase Server, the server provides only a caching layer as storage and no data persistence on disk. If your server runs out of space in RAM, items are evicted from RAM on a least recently used basis (LRU). Eviction means the server removes the key, metadata and all other data for the item from RAM. After eviction, the item is irretrievable.

Related Links

Disk storage on page 102
Couchbase Server mainly stores and retrieves information for clients using RAM. At the same time, Couchbase Server eventually stores all data to disk to provide a higher level of reliability.

Related topics

Architecture and concepts on page 95
Couchbase Server concepts include the differenct components and systems that make up an individual Couchbase Server instance and a Couchbase cluster including information and concepts needed to understand the fast and elastic nature, high availability, and high performance of the Couchbase Server database.

Tombstone purging

Tombstones are records of expired or deleted items that include item keys and metadata.

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 in two structures per node. With millions of mutations, the space taken up by tombstones can grow quickly. This is especially the case if there are a large number of deletions or expired documents.

The Metadata Purge Interval sets how frequently a node permanently purges metadata on deleted and expired items. The Metadata Purge Interval setting runs as part of auto-compaction. This helps reduce the storage requirement by roughly 3x times lower than before and also frees up space much faster.

Related Links

Disk storage on page 102

Couchbase Server mainly stores and retrieves information for clients using RAM. At the same time, Couchbase Server eventually stores all data to disk to provide a higher level of reliability.

Related topics

Command-line interface overview on page 353

REST API overview on page 426

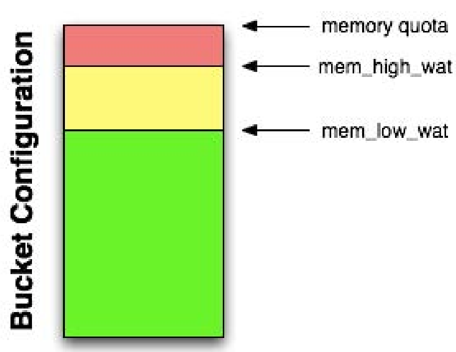
Working set management and ejection

Working set management is the process of freeing up space and ensuring that the most used items are available in RAM. Ejection is the process of removing data from RAM to provide room for frequently used items.

The process that Couchbase Server performs to free space in RAM, and to ensure the most-used items are still available in RAM is also known as working set management. Ejection is the process of removing data from RAM to provide room for frequently-used items. Ejections is automatically performed by Couchbase Server. When Couchbase Server ejects information, it works in conjunction with the disk persistence system to ensure that data in RAM has been persisted to disk and can be safely retrieved back into RAM if the item is requested.

In addition to memory quota for the caching layer, there are two watermarks the engine uses to determine when it is necessary to start persisting more data to disk. These are `mem_low_wat` and `mem_high_wat`.

As the caching layer becomes full of data, eventually the mem_low_wat is passed. At this time, no action is taken. As data continues to load, it eventually reaches mem_high_wat. At this point, a background job is scheduled to ensure items are migrated to disk and that memory is available for other Couchbase Server items. This job runs until measured memory reaches mem_low_wat. If the rate of incoming items is faster than the migration of items to disk, the system may return errors indicating there is not enough space. This continues until there is available memory. The process of removing data from the caching to make way for the actively used information is called ejection and is controlled automatically through thresholds set on each configured bucket in the Couchbase Server cluster.



Working set management process

Couchbase Server actively manages the data stored in a caching layer; this includes the information which is frequently accessed by clients and which needs to be available for rapid reads and writes. When there are too many items in RAM, Couchbase Server removes certain data to create free space and to maintain system performance. This process is called “working set management” and the set of data in RAM is referred to as the “working set”.

In general the working set consists of all the keys, metadata, and associated documents which are frequently used require fast access. The process the server performs to remove data from RAM is known as ejection. When the server performs this process, it removes the document, but not the keys or metadata for the item. Keeping keys and metadata in RAM serves three important purposes in a system:

- Couchbase Server uses the remaining key and metadata in RAM if a request for that key comes from a client. If a request occurs, the server then tries to fetch the item from disk and return it into RAM.
- The server can also use the keys and metadata in RAM for “miss access”. This means that it is quickly determine whether an item is missing and if so, perform some action, such as add it.

- Finally, the expiration process in Couchbase Server uses the metadata in RAM to quickly scan for items that are expired and later remove them from disk. This process is known as the "expiry pager" and runs every 60 minutes by default.

Not Frequently Used (NFU) items

All items in the server contain metadata indicating whether the item has been recently accessed or not. This metadata is known as not-recently-used (NRU). If an item has not been recently used, then the item is a candidate for ejection if the high water mark has been exceeded. When the high water mark has been exceeded, the server evicts items from RAM.

Couchbase Server provides two NRU bits per item and also provides a replication protocol that can propagate items that are frequently read, but not mutated often.

For earlier versions, Couchbase Server provided only a single bit for NRU and a different replication protocol which resulted in two issues: metadata could not reflect how frequently or recently an item had been changed, and the replication protocol only propagated NRUs for mutation items from an active vBucket to a replica vBucket. This second behavior meant that the working set on an active vBucket could be quite different than the set on a replica vBucket. By changing the replication protocol, the working set in replica vBuckets will be closer to the working set in the active vBucket.

NRUs are decremented or incremented by server processes to indicate an item is more frequently used, or less frequently used. Items with lower bit values have lower scores and are considered more frequently used. The bit values, corresponding scores and status are as follows:

Binary NRU	Score	Working Set Replication Status (WSR)	Access Pattern	Description
00	0	TRUE	Set by write access to 00. Decremented by read access or no access.	Most heavily used item.
01	1	Set to TRUE	Decremented by read access.	Frequently access item.
10	2	Set to FALSE	Initial value or decremented by read access.	Default for new items.
11	3	Set to FALSE	Incremented by item pager for eviction.	Less frequently used item.

When WSR is set to TRUE it means that an item should be replicated to a replica vBucket. There are two processes which change the NRU for an item:

- A client reads or writes an item, the server decrements NRU and lowers the item's score
- A daily process which creates a list of frequently-used items in RAM. After this process runs, the server increments one of the NRU bits.

Because the two processes changes NRUs, they also affect which items are candidates for ejection.

Couchbase Server settings can be adjusted to change behavior during ejection. For example, specify the percentage of RAM to be consume before items are ejected or specify whether ejection should occur more frequently on replicated data than on original data. Couchbase recommends that the default settings be used.

Understanding the item pager

The item pager process, which runs periodically, removes documents from RAM and retains the item's key and metadata. If the amount of RAM used by items reaches the high water mark (upper threshold), both active and replica data are ejected until the memory usage (amount of RAM consumed) reaches the low water mark (lower threshold). Evictions of active and replica data occur with the ratio probability of 40% (active data) to 60% (replica data) until the memory usage reaches the low watermark. Both the high water mark and low water mark are expressed as a percentage amount of RAM, such as 80%.

Both the high water mark and low water mark can be changed by providing a percentage amount of RAM for a node, for example, 80%. Couchbase recommends that the following default settings be used:

Version	High Water Mark	Low Water Mark
2.0	75%	60%
2.0.1 and higher	85%	75%

The item pager ejects items from RAM in two phases:

- **Phase 1: Eject based on NRU**. Scan NRU for items and create list of all items with score of 3. Eject all items with a NRU score of 3. Check RAM usage and repeat this process if usage is still above the low water mark.
- **Phase 2: Eject based on Algorithm**. Increment all item NRUs by 1. If an NRU is equal to 3, generate a random number and eject that item if the random number is greater than a specified probability. The probability is based on current memory usage, low water mark, and whether a vBucket is in an active or replica state. If a vBucket is in active state the probability of ejection is lower than if the vBucket is in a replica state. The default probabilities for ejection from active of replica vBuckets is as follows:

The following is the probability of ejection based on active vs. replica vBuckets:

Active vBucket	Replica vBucket
60%	40%

Related Links

Architecture and concepts on page 95
Couchbase Server concepts include the differenct components and systems that make up an individual Couchbase Server instance and a Couchbase cluster including information and concepts needed to understand the fast and elastic nature, high availability, and high performance of the Couchbase Server database.

Ejection and working set management on page 107
Working set management refers to the ejection of data from RAM. Couchbase Server ejects data to create free space and to maintain system performance.

Related topics

cbepctl tool on page 379

Changing disk cleanup interval on page 383

Ejection and working set management

Working set management refers to the ejection of data from RAM. Couchbase Server ejects data to create free space and to maintain system performance.

Couchbase Server actively manages the data stored in a caching layer; this includes the information which is frequently accessed by clients and which needs to be available for rapid reads and writes. When there are too many items in RAM, Couchbase Server removes certain data to create free space and to maintain system performance. This process is called “working set management” and the set of data in RAM is referred to as the “working set”.

In general the working set consists of all the keys, metadata, and associated documents which are frequently used require fast access. The process the server performs to remove data from RAM is known as ejection. When the server performs this process, it removes the document, but not the keys or metadata for the item. Keeping keys and metadata in RAM serves three important purposes in a system:

- Couchbase Server uses the remaining key and metadata in RAM if a request for that key comes from a client. If a request occurs, the server then tries to fetch the item from disk and return it into RAM.

- The server can also use the keys and metadata in RAM for “miss access”. This means that it is quickly determine whether an item is missing and if so, perform some action, such as add it.
- Finally, the expiration process in Couchbase Server uses the metadata in RAM to quickly scan for items that are expired and later remove them from disk. This process is known as the “expiry pager” and runs every 60 minutes by default

Not-Frequently-Used Items

All items in the server contain metadata indicating whether the item has been recently accessed or not. This metadata is known as not-recently-used (NRU). If an item has not been recently used, then the item is a candidate for ejection if the high water mark has been exceeded. When the high water mark has been exceeded, the server evicts items from RAM.

Couchbase Server provides two NRU bits per item and also provides a replication protocol that can propagate items that are frequently read, but not mutated often.

For earlier versions, Couchbase Server provided only a single bit for NRU and a different replication protocol which resulted in two issues: metadata could not reflect how frequently or recently an item had been changed, and the replication protocol only propagated NRUs for mutation items from an active vBucket to a replica vBucket. This second behavior meant that the working set on an active vBucket could be quite different than the set on a replica vBucket. By changing the replication protocol, the working set in replica vBuckets will be closer to the working set in the active vBucket.

NRUs are decremented or incremented by server processes to indicate an item is more frequently used, or less frequently used. Items with lower bit values have lower scores and are considered more frequently used. The bit values, corresponding scores and status are as follows:

Binary NRU	Score	Working Set Replication Status (WSR)	Access Pattern	Description
00	0	TRUE	Set by write access to 00. Decremented by read access or no access.	Most heavily used item.
01	1	Set to TRUE	Decremented by read access.	Frequently access item.
10	2	Set to FALSE	Initial value or decremented by read access.	Default for new items.
11	3	Set to FALSE	Incremented by item pager for eviction.	Less frequently used item.

When WSR is set to TRUE it means that an item should be replicated to a replica vBucket. There are two processes which change the NRU for an item:

- A client reads or writes an item, the server decrements NRU and lowers the item’s score
- A daily process which creates a list of frequently-used items in RAM. After this process runs, the server increments one of the NRU bits.

Because the two processes changes NRUs, they also affect which items are candidates for ejection.

Couchbase Server settings can be adjusted to change behavior during ejection. For example, specify the percentage of RAM to be consume before items are ejected or specify whether ejection should occur more frequently on replicated data than on original data. Couchbase recommends that the default settings be used.

Understanding the Item Pager

The item pager process, which runs periodically, removes documents from RAM and retains the item’s key and metadata. If the amount of RAM used by items reaches the high water mark (upper threshold), both active and replica data are ejected until the memory usage (amount of RAM consumed) reaches the low water mark (lower threshold). Evictions of active and replica data occur with the ratio probability of 40% (active data) to 60% (replica data) until the memory usage reaches the low watermark. Both the high water mark and low water mark are expressed as a percentage amount of RAM, such as 80%.

Both the high water mark and low water mark can be changed by providing a percentage amount of RAM for a node, for example, 80%. Couchbase recommends that the following default settings be used:

Version	High Water Mark	Low Water Mark
2.0	75%	60%
2.0.1 and higher	85%	75%

The item pager ejects items from RAM in two phases:

- **Phase 1: Eject based on NRU**. Scan NRU for items and create list of all items with score of 3. Eject all items with a NRU score of 3. Check RAM usage and repeat this process if usage is still above the low water mark.
- **Phase 2: Eject based on Algorithm**. Increment all item NRUs by 1. If an NRU is equal to 3, generate a random number and eject that item if the random number is greater than a specified probability. The probability is based on current memory usage, low water mark, and whether a vBucket is in an active or replica state. If a vBucket is in active state the probability of ejection is lower than if the vBucket is in a replica state. The default probabilities for ejection from active of replica vBuckets is as follows:

The following is the probability of ejection based on active vs. replica vBuckets:

Active vBucket	Replica vBucket
60%	40%

Related Links

Working set management and ejection on page 105
Working set management is the process of freeing up space and ensuring that the most used items are available in RAM. Ejection is the process of removing data from RAM to provide room for frequently used items.

Related topics

Changing thresholds for ejection on page 381

Changing disk cleanup interval on page 383

Expiration

Each document stored in the database has an optional expiration value (TTL, time to live) that is used to automatically deleted items.

The expiration option can be used for data that has a limited life and could be automatically deleted. The expiration value is user-specified on a document basis at the point when the data is stored. The expiration can also be updated when the data is updated or explicitly changed through the Couchbase protocol. The expiration time can either be specified as a relative time (for example, in 60 seconds) or absolute time (31st December 2012, 12:00pm).

The default is no expiration, that is, the information is stored indefinitely. Typical uses for an expiration value include web session data where the actively stored information needs to be removed from the system once the user activity has stopped. With an expiration value, the data times out and is removed from the system without being explicitly deleted. This frees up RAM and disk for more active data.

Related Links

Architecture and concepts on page 95
Couchbase Server concepts include the differenct components and systems that make up an individual Couchbase Server instance and a Couchbase cluster including information and concepts needed to understand the fast and elastic nature, high availability, and high performance of the Couchbase Server database.

Related topics

cbepctl tool on page 379

Server warmup

Whenever the Couchbase server is restarted or data is restored to a server instance, the server undergoes a warmup process before data requests can be handled.

During server warmup, the server loads data persisted on disk into RAM. Couchbase Server provides an optimized warmup process which loads data sequentially from disk into RAM, it divides the data to be loaded and handles it in multiple phases. After the warmup process completes, the data is available for clients to read and write. The time needed for server warmup depends on system size, system configuration and the amount of data persisted in the system.

Note: Couchbase Server is able to begin serving data before it has actually loaded all the keys and data from vBuckets.

Couchbase server identifies items that are frequently used, prioritizes them, and loads them before sequentially loading the remainer of the data. The frequently-used items are prioritized in an access log. The server pre-fetches a list of the most frequently accessed keys and then fetches these documents before any other items from disk.

The server runs a configurable scanner process that determines which keys are most frequently-used. The scanner process is pre-set and is configurable. The command-line tool, `cbepctl flush_param` is used to change the initial time and interval for the scanner process. For example, you might want to configure the scanner process to be run during a specific time period when certain keys need to be identified and made available sooner.

The server can also switch into a ready mode before it has actually retrieved all documents for keys into RAM, therfore, data can be served before all stored items are loaded. Switching into ready mode is a configurable setting so that server warmup time can be adjusted.

The following describes the initial warmup phases for the Couchbase Server. In these first phase, the server begins to fetch all keys and metadata from disk. Then the server accesses the log information that it needs to retrieve the most used keys:

1. **Initialize** - At this phase, the server does not have any data that it can serve yet. The server starts populating a list of all vBuckets stored on disk by loading the recorded, initial state of each vBucket.
2. **Key Dump** - The server begins pre-fetching all keys and metadata from disk based on items in the vBucket list.
3. **Check Access Logs** - The server then reads a single cached access log which indicates which keys are frequently accessed. The server generates and maintains this log on a periodic basis and it is configurable. If this log exists, the server first loads items based on this log before it loads other items from disk.

Once Couchbase Server has the information about keys and has read in any access log information, it loads documents based on the following criteria:

- **Loading based on Access Logs** - Couchbase Server loads documents into memory based on the frequently-used items identified in the access log.
- **Loading Data** - If the access log is empty or is disabled, the server sequentially loads documents for each key based on the vBucket list.

Couchbase Server is able to serve information from RAM when one of the following conditions is met during warmup:

- The server has finished loading documents for all keys listed in the access log
- The server has finished loading documents for every key stored on disk for all vBuckets
- The percentage of documents loaded into memory is greater than, or equal to, the setting for the `cbepctl ep_warmup_min_items_threshold` parameter
- If total % of RAM filled by documents is greater than, or equal to, the setting for the `cbepctl ep_warmup_min_memory_threshold` parameter
- If total RAM usage by a node is greater than or equal to the setting for `mem_low_wat`.

When the server reaches one of these states, known as the *run level*, the server stops loading documents for the remaining keys. and loads the remaining documents from disk into RAM as a background data fetch.

Related Links

Architecture and concepts on page 95
Couchbase Server concepts include the differenct components and systems that make up an individual Couchbase Server instance and a Couchbase cluster including information and concepts needed to understand the fast and elastic nature, high availability, and high performance of the Couchbase Server database.

Related topics

cbepctl tool on page 379

Changing access log settings on page 382

Getting warmup information on page 398

Handling server warmup on page 135
Couchbase server warmup behavior can be modified changing the access scanner and warmup threshold settings via the `cbepctl` tool.

Rebalancing

Rebalancing is the process of redistributing information across nodes

Data is stored within Couchbase Server through the distribution method that is provided by the vBucket structure. When a Couchbase Server cluster is expanded or shrunk, the information stored in the vBuckets is redistributed between the available nodes and the corresponding vBucket map is updated to reflect the new structure. This process is called *rebalancing*.

Rebalancing is an deliberate process that you need to initiate manually when the structure of your cluster changes. The rebalance process changes the allocation of the vBuckets used to store the information and then physically moves the data between the nodes to match the new structure.

The rebalancing process can take place while the cluster is running and servicing requests. Clients using the cluster read and write to the existing structure with the data being moved in the background between nodes. Once the moving process has been completed, the vBucket map is updated and communicated to the smart clients and the proxy service (Moxi).

The result is that the distribution of data across the cluster has been rebalanced, or smoothed out, so that the data is evenly distributed across the database. Rebalancing takes into account the data and replicas of the data required to support the system.

Related Links

Architecture and concepts on page 95
Couchbase Server concepts include the differenct components and systems that make up an individual Couchbase Server instance and a Couchbase cluster including information and concepts needed to understand the fast and elastic nature, high availability, and high performance of the Couchbase Server database.

Replicas and replication

Replicas are copies of data that are proved on another node in a cluster.

In addition to distributing information across the cluster for even data distribution and cluster performance, you can also establish *replica vBuckets* within a single Couchbase cluster.

A copy of data from one bucket, known as a source is copied to a destination, which we also refer to as the replica, or replica vBucket. The node that contains the replica vBucket is also referred to as the *replica node* while the node containing original data to be replicated is called a *source node*. Distribution of replica data is handled in the same way as data at a source node; portions of replica data will be distributed around the cluster to prevent a single point of failure.

After Couchbase has stored replica data at a destination node, the data will also be placed in a queue to be persisted on disk at that destination node.

When replication is performed between two Couchbase clusters, it is called cross datacenter replication (XDCR). Use cases for XDCR is for a copy of your data on a cluster that is closer to your users or for backup data in case of disaster recovery.

Related Links

Architecture and concepts on page 95
Couchbase Server concepts include the differenct components and systems that make up an individual Couchbase Server instance and a Couchbase cluster including information and concepts needed to understand the fast and elastic nature, high availability, and high performance of the Couchbase Server database.

Related topics

Managing XDCR on page 169
Cross datacenter replication (XDCR) provides an easy method of replicating data from one cluster to another for disaster recovery as well as better data locality (getting data closer to its users).

Handling replication on page 136
Data replication is distributed throughout the Couchbase cluster to prevent a single point of failure. Data replication is configurable on a bucket-level and node-basis.

Rack Awareness

The Rack Awareness feature allows logical groupings of servers on a cluster where each server group physically belongs to a rack or availability zone.

Rack Awareness provides the ability to specify that active and corresponding replica partitions be created on servers that are part of a separate rack or zone. To use and enable Rack Awareness, all servers in a cluster must be upgraded to Couchbase Server Enterprise Edition and minimally, version 2.5. By design, Couchbase Server evenly distributes data of active and replica vBuckets across the cluster for cluster performance and redundancy purposes. With Rack Awareness, server partitions are laid out so the replica partitions for servers in one server group are distributed in servers for a second group and vice versa. If one of the servers becomes unavailable or if an entire rack goes down, data is retained since the replicas are available on the second server group.

Replica vBuckets are evenly distributed from one server group to another server group to provide redundancy and data availability. The rebalance operation also evenly distributes the replica vBuckets from one server group to another server group across the cluster. If an imbalance occurs where there is an unequal number of servers in one server group, the rebalance operation performs a "best effort" of evenly distributing the replica vBuckets across the cluster.

Distribution of vBuckets and replica vBuckets

The following example shows how Rack Awareness functionality implements replica vBuckets to provide redundancy. In this example, there are two (2) server groups in the cluster and four (4) servers in each server group. Since there is equal number of servers in each server group, the cluster is balanced which guarantees that replica vBuckets for one server group are on a different server group.

The following diagram shows a cluster of servers on two racks, Rack #1 and Rack #2, where each rack has a group of four (4) servers.

- Group 1 has Servers 1, 2, 3, and 4.
- Group 1 servers have their active vBuckets and replica vBuckets from Group 2.
- Group 2 has Servers 5, 6, 7, and 8.
- Group 2 servers have their active vBuckets and replica vBuckets from Group 1.

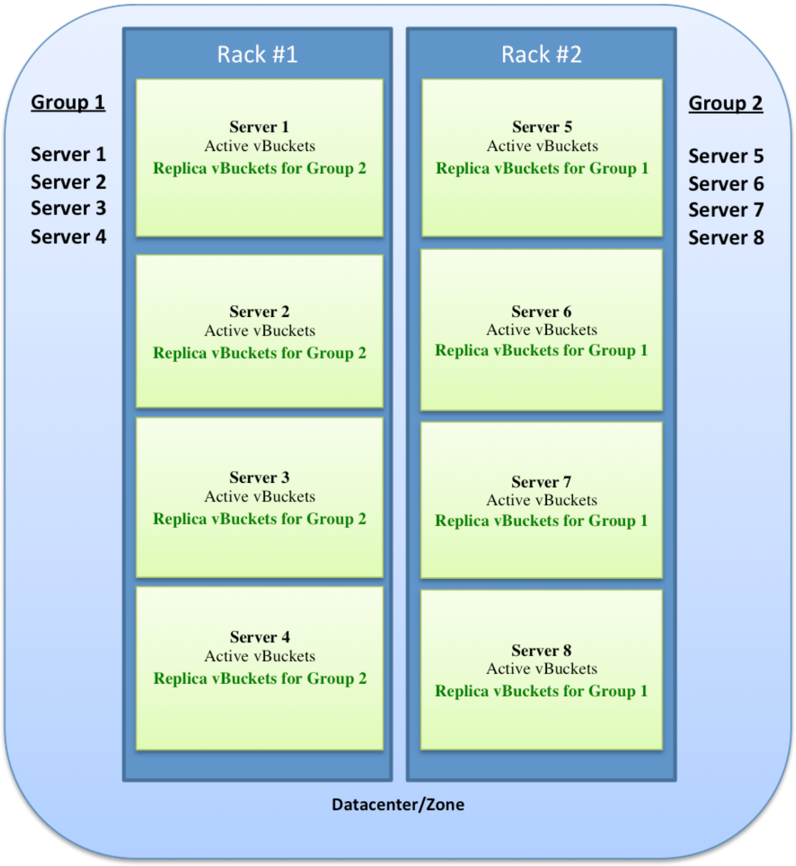


Figure 1: Rack Awareness

Distribution with additional server

The following scenario shows how Rack Awareness functionality implements replica vBuckets when an imbalance is caused by an additional server being added to one server group. In this example, an additional server (Server 9) is added to a server group (Group 1). An imbalance occurs because one server group has more servers than the other server group. In this case, the rebalance operation performs a "best effort" of evenly distributing the replica vBuckets of the additional server across the nodes on all the racks in the cluster.

The following diagram shows a cluster of servers on two racks, Rack #1 and Rack #2, where one rack has a group of five (5) servers and the other rack has a group of four (4) servers.

- Group 1 has Servers 1, 2, 3, 4, and 9.
- Group 1 servers have their active vBuckets and replica vBuckets from Group 2.
- Group 1 Servers 1 - 4 also has replica vBuckets for Server 9.
- Group 2 has Servers 5, 6, 7, and 8.
- Group 2 servers have their active vBuckets and replica vBuckets from Group 1 including the replica vBuckets from Server 9 in Group 1.

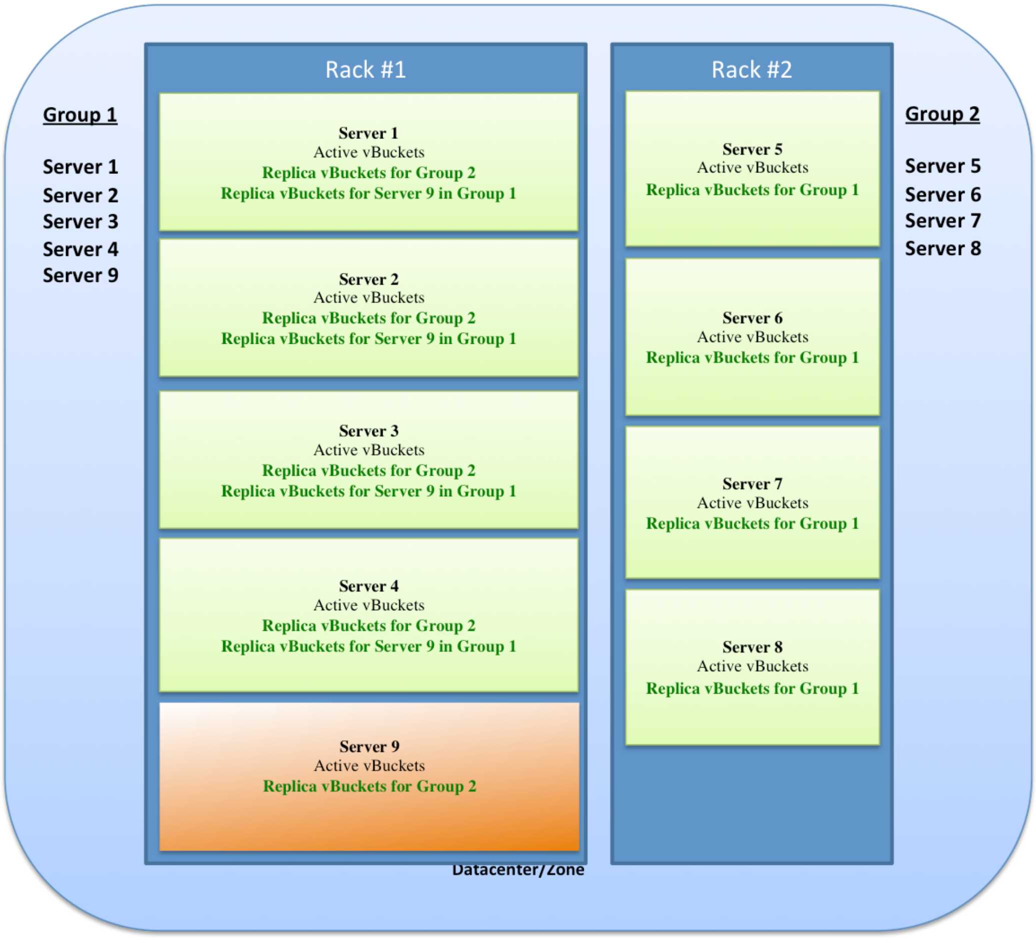


Figure 2: Rack Awareness with additional server

Distribution with unavailable server

The following scenario shows how Rack Awareness functionality implements replica vBuckets when an imbalance is caused by a server being removed or unavailable in a server group. In this example, a server (Server 2) is unavailable to a server group (Group 1). An imbalance occurs because one server group has fewer servers than the other server group. In this case, if the rebalance operation is performed, a "best effort" of evenly distributing the replica vBuckets across the cluster occurs.

Note: If the cluster becomes imbalanced, add servers to balance the cluster. For optimal Rack Awareness functionality, a balanced cluster is recommended. If there is only one server or only one server group, default behavior is automatically implemented, that is, Rack Awareness functionality is disabled.

The following diagram shows the loss of a server resulting in an imbalance. In this case, Server 2 (from Group 1, Rack #1) becomes unavailable. The replica vBuckets for Server 2 in Group 2, Rack #2 become enabled and rebalancing occurs.

- Group 1 has Servers 1, 2, 3, and 4.
- Group 1 servers have their active vBuckets and replica vBuckets from Group 2.
- Group 1 Server 2 becomes unavailable.
- Group 2 has Servers 5, 6, 7, and 8.
- Group 2 servers have their active vBuckets and replica vBuckets from Group 1.
- Group 2 server activates the replica vBuckets for Server 2 in Group 1.

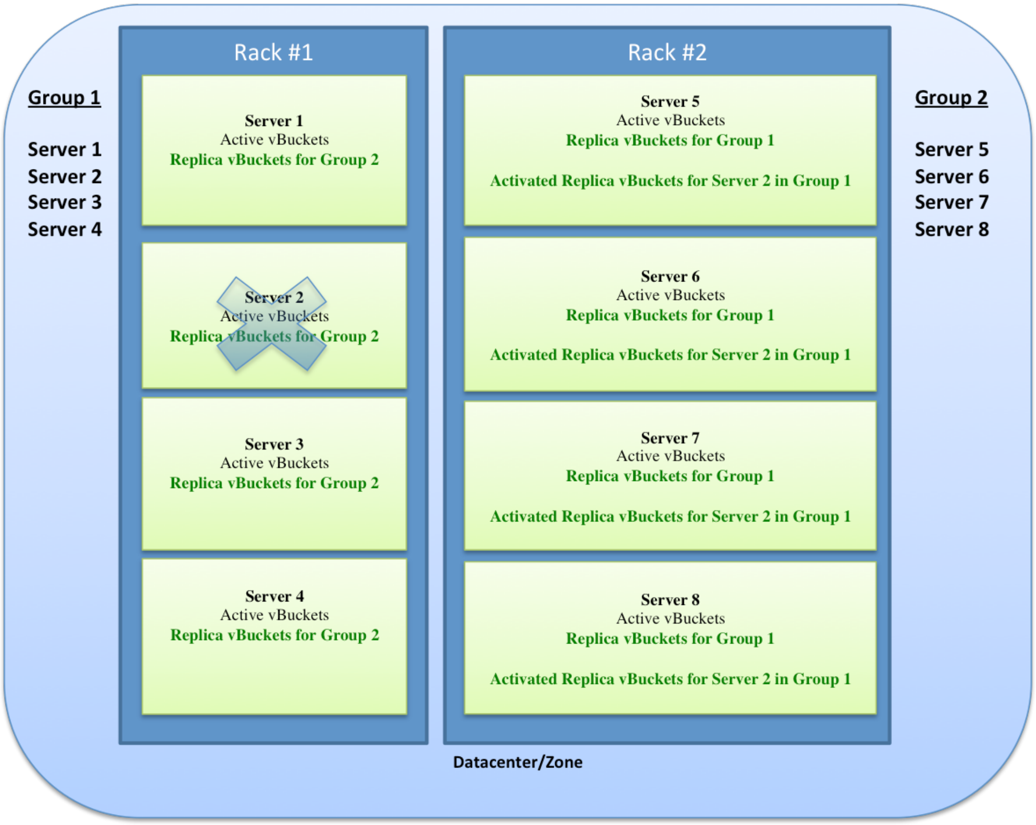


Figure 3: Rack Awareness with unavailable server

Related Links

Architecture and concepts on page 95
Couchbase Server concepts include the differenct components and systems that make up an individual Couchbase Server instance and a Couchbase cluster including information and concepts needed to understand the fast and elastic nature, high availability, and high performance of the Couchbase Server database.

Related topics

Managing Rack Awareness on page 255
The Rack Awareness feature (Enterprise Edition) allows logical groupings of servers on a cluster where each server group physically belongs to a rack or availability zone. This feature provides the ability to specify that active and corresponding replica partitions be created on servers that are part of a separate rack or zone.

Failover

Failover is the process of indicating that a node in a cluster is unavailable and that enabling replica vBuckets.

Information is distributed around a cluster using a series of replicas. For Couchbase buckets you can configure the number of `replicas` (complete copies of the data stored in the bucket) that should be kept within the Couchbase Server Cluster.

In the event of a failure in a server (either due to transient failure, or for administrative purposes), you can use a technique called `failover` to indicate that a node within the Couchbase Cluster is no longer available, and that the replica vBuckets for the server are enabled.

The failover process contacts each server that was acting as a replica and updates the internal table that maps client requests for documents to an available server.

Failover can be performed manually, or you can use the built-in automatic failover that reacts after a preset time when a node within the cluster becomes unavailable.

Related Links

Architecture and concepts on page 95
Couchbase Server concepts include the differenct components and systems that make up an individual Couchbase Server instance and a Couchbase cluster including information and concepts needed to understand the fast and elastic nature, high availability, and high performance of the Couchbase Server database.

Related topics

Failing over nodes on page 141
Failing over a node means that Couchbase Server removes the node from a cluster and makes replicated data at other nodes available for client requests.

TAP

The TAP protocol is an internal part of the Couchbase Server system and is used to exchange data throughout the system.

TAP provides a stream of data of the changes that are occurring within the system. TAP is used during replication, to copy data between vBuckets used for replicas. It is also used during the rebalance procedure to move data between vBuckets and redistribute the information across the system.

Client interface

Within Couchbase Server, the techniques and systems used to get information into and out of the database differ according to the level and volume of data that you want to access. The different methods can be identified according to the base operations of Create, Retrieve, Update and Delete:

Create

Information is stored into the database using the memcached protocol interface to store a *value* against a specified *key*. Bulk operations for setting the key/value pairs of a large number of documents at the same time are available, and these are more efficient than multiple smaller requests.

The value stored can be any binary value, including structured and unstructured strings, serialized objects (from the native client language), native binary data (for example, images or audio). For use with the Couchbase Server View engine, information must be stored using the JavaScript Object Notation (JSON) format, which structures information as a object with nested fields, arrays, and scalar datatypes.

Retrieve

To retrieve information from the database, there are two methods available: By Key and BY View

- By Key

  If you know the key used to store a particular value, then you can use the memcached protocol (or an appropriate memcached compatible client-library) to retrieve the value stored against a specific key. You can also perform bulk operations.
- By View

  f you do not know the key, you can use the View system to write a view that outputs the information you need. The view generates one or more rows of information for each JSON object stored in the database. The view definition includes the keys (used to select specific or ranges of information) and values. For example, you could create a view on contact information that outputs the JSON record by the contact's name, and with a value containing the contacts address. Each view also outputs the key used to store the original object. If the view doesn't contain the information you need, you can use the returned key with the memcached protocol to obtain the complete record.

Update

To update information in the database, you must use the memcached protocol interface. The memcached protocol includes functions to directly update the entire contents, and also to perform simple operations, such as appending information to the existing record, or incrementing and decrementing integer values.

Delete

To delete information from Couchbase Server, you need to use the memcached protocol which includes an explicit delete command to remove a key/value pair from the server.

However, Couchbase Server also allows information to be stored in the database with an expiry value. The expiry value states when a key/value pair should be automatically deleted from the entire database, and can either be specified as a relative time (for example, in 60 seconds), or absolute time (31st December 2012, 12:00pm).

Note: The methods of creating, updating and retrieving information are critical to the way you work with storing data in Couchbase Server.

Related Links

Architecture and concepts on page 95
Couchbase Server concepts include the differenct components and systems that make up an individual Couchbase Server instance and a Couchbase cluster including information and concepts needed to understand the fast and elastic nature, high availability, and high performance of the Couchbase Server database.

Statistics and monitoring

A complete set of statical and monitoring information are provided through the Web Console, CLI, and REST API.

In order to understand what your cluster is doing and how it is performing, Couchbase Server incorporates a complete set of statistical and monitoring information. The statistics are provided through all of the administration interfaces. Within the Web Administration Console, a complete suite of statistics are provided, including built-in real-time graphing and performance data.

The statistics are divided into a number of groups, allowing you to identify different states and performance information within your cluster:

- **By Node**

Node statistics show CPU, RAM and I/O numbers on each of the servers and across your cluster as a whole. This information can be used to help identify performance and loading issues on a single server.

- **By vBucket**

The vBucket statistics show the usage and performance numbers for the vBuckets used to store information in the cluster. These numbers are useful to determine whether you need to reconfigure your buckets or add servers to improve performance.

- **By View**

View statistics display information about individual views in your system, including the CPU usage and disk space used so that you can monitor the effects and loading of a view on your Couchbase nodes. This information may indicate that your views need modification or optimization, or that you need to consider defining views across multiple design documents.

- **By Disk Queues**

These statistics monitor the queues used to read and write information to disk and between replicas. This information can be helpful in determining whether you should expand your cluster to reduce disk load.

- **By TAP Queues**

The TAP interface is used to monitor changes and updates to the database. TAP is used internally by Couchbase to provide replication between Couchbase nodes, but can also be used by clients for change notifications.

In nearly all cases the statistics can be viewed both on a whole of cluster basis, so that you can monitor the overall RAM or disk usage for a given bucket, or an individual server basis so that you can identify issues within a single machine.

Related Links

Architecture and concepts on page 95

Couchbase Server concepts include the differenct components and systems that make up an individual Couchbase Server instance and a Couchbase cluster including information and concepts needed to understand the fast and elastic nature, high availability, and high performance of the Couchbase Server database.

Deployment considerations

Deployment configuration take into account topics such as restricted access, node communication, swap configuration, and connection timeouts.

- **Restricted access to Moxi ports** - Make sure that only trusted machines (including the other nodes in the cluster) can access the ports that Moxi uses.
- **Restricted access to web console (port 8091)** - The web console is password protected. However, we recommend that you restrict access to port 8091; an abuser could do potentially harmful operations (like remove a node) from the web console.
- **Node to Node communication on ports** - All nodes in the cluster should be able to communicate with each other on 11210 and 8091.
- **Swap configuration** - Swap should be configured on the Couchbase Server. This prevents the operating system from killing Couchbase Server should the system RAM be exhausted. Having swap provides more options on how to manage such a situation.
- **Idle connection timeouts** - Some firewall or proxy software will drop TCP connections if they are idle for a certain amount of time (e.g. 20 minutes). If the software does not allow you to change that timeout, send a command from the client periodically to keep the connection alive.
- **Port Exhaustion on Windows** - The TCP/IP port allocation on Windows by default includes a restricted number of ports available for client communication.

Related Links

Couchbase Administration on page 89

Cluster design considerations on page 119
Cluster design takes into account topics such as RAM, number of nodes and cores, clied or server-side moxi.

Sizing guidelines on page 120
The primary considerations for sizing a Couchbase Server cluster are the number of nodes and node size.

Deployment strategies on page 125

Ongoing monitoring and maintenance on page 127
There are a number of different statistics

Couchbase behind a secondary firewall on page 128

Couchbase in the cloud on page 128
Couchbase Server is extremely easy to deploy in the cloud.

XDCR in cloud deployments on page 133

Related topics

MSDN: Avoiding TCP/IP Port Exhaustion

Cluster design considerations

Cluster design takes into account topics such as RAM, number of nodes and cores, clied or server-side moxi.

- RAM: Memory is a key factor for smooth cluster performance. Couchbase best fits applications that want most of their active dataset in memory. It is very important that all the data you actively use (the working set) lives in memory. When there is not enough memory left, some data is ejected from memory and will only exist on disk. Accessing data from disk is much slower than accessing data in memory. As a result, if ejected data is accessed frequently, cluster performance suffers. Use the formula provided in the next section to verify your configuration, optimize performance, and avoid this situation.
- Number of nodes: Once you know how much memory you need, you must decide whether to have a few large nodes or many small nodes.
    - Many small nodes: You are distributing I/O across several machines. However, you also have a higher chance of node failure (across the whole cluster).
    - Few large nodes: Should a node fail, it greatly impacts the application.
    - It is a trade off between reliability and efficiency.

- Couchbase prefers a client-side moxi (or a smart client) over a server-side moxi. However, for development environments or for faster, easier deployments, you can use server-side moxis. A server-side moxi is not recommended because of the following drawback: if a server receives a client request and doesn't have the requested data, there's an additional hop.
- Number of cores: Couchbase is relatively more memory or I/O bound than is CPU bound. However, Couchbase is more efficient on machines that have at least two cores.
- Storage type: You may choose either SSDs (solid state drives) or spinning disks to store data. SSDs are faster than rotating media but, currently, are more expensive. Couchbase needs less memory if a cluster uses SSDs as their I/O queue buffer is smaller.
- WAN Deployments: Couchbase is not intended to be used in WAN configurations. Couchbase requires that the latency should be very low between server nodes and between servers nodes and Couchbase clients.

Related Links

Deployment considerations on page 119
Deployment configuration take into account topics such as restricted access, node communication, swap configuration, and connection timeouts.

Sizing guidelines

The primary considerations for sizing a Couchbase Server cluster are the number of nodes and node size.

When sizing your Couchbase Server cluster, ask the following questions:

- How many nodes do I need?
- How large (RAM, CPU, disk space) should those nodes be?

To determine the number of nodes needed for a cluster, consider the following:.

- RAM
- Disk throughput and sizing
- Network bandwidth
- Data distribution and safety

Due to the in-memory nature of Couchbase Server, RAM is usually the determining factor for sizing. But ultimately, the primary factor depends on the data set and information being stored.

For example:

- If you have a very small data set that gets a very high load, you'll need to base your size more off of network bandwidth than RAM.
- If you have a very high write rate, you'll need more nodes to support the disk throughput needed to persist all that data (and likely more RAM to buffer the incoming writes).
- Even with a very small dataset under low load, you may want three nodes for proper distribution and safety.

With Couchbase Server, you can increase the capacity of your cluster (RAM, Disk, CPU, or network) by increasing the number of nodes within your cluster, since each limit will be increased linearly as the cluster size is increased.

RAM sizing

RAM is usually the most critical sizing parameter. It's also the one that can have the biggest impact on performance and stability.

Working set

The working set is the data that the client application actively uses at any point in time. Ideally, all of the working set lives in memory. This impacts how much memory is needed.

Memory quota

It is very important that your Couchbase cluster's size corresponds to the working set size and total data you expect.

The goal is to size the available RAM to Couchbase so that all your document IDs, the document ID meta data, and the working set values fit. The memory should rest just below the point at which Couchbase will start evicting values to disk (the High Water Mark).

Important: You will not be able to allocate all your machine RAM to the Couchbase server node (`per_node_ram_quota` parameter) because other programs might be running on your machine.

How much memory and disk space per node you will need depends on several different variables, which are defined below:

The following calculations are per-bucket calculations. The calculations need to be summed up across all buckets. If all the buckets have the same configuration, treat the total data as a single bucket. There is no per-bucket overhead that needs to be considered.

Variable	Description
documents_num	The total number of documents you expect in your working set
ID_size	The average size of document IDs
value_size	The average size of values
number_of_replicas	The number of copies of the original data you want to keep
working_set_percentage	The percentage of your data you want in memory
per_node_ram_quota	How much RAM can be assigned to Couchbase

Use the following items to calculate how much memory you need:

Constant	Description
Metadata per document (metadata_per_document)	This is the amount of memory that Couchbase needs to store metadata per document. Metadata uses 56 bytes. All the metadata needs to live in memory while a node is running and serving data.
SSD or Spinning	SSDs give better I/O performance.
headroom	The cluster needs additional overhead to store metadata. That space is called the headroom. This requires approximately 25–30% more space than the raw RAM requirements for your dataset. Since SSDs are faster than spinning (traditional) hard disks, you should set aside 25% of memory for SSDs and 30% of memory for spinning hard disks.
High Water Mark (high_water_mark)	By default, the high water mark for a node's RAM is set at 85%.

This is a rough guideline to size your cluster:

Variable	Calculation
no_of_copies	`1 + number_of_replicas`
total_metadata All the documents need to live in the memory.	`(documents_num) * (metadata_per_document + ID_size) * (no_of_copies)`
total_dataset	`(documents_num) * (value_size) * (no_of_copies)`

Variable	Calculation
working_set	`total_dataset * (working_set_percentage)`
Cluster RAM quota required	`(total_metadata + working_set) * (1 + headroom) / (high_water_mark)`
number of nodes	`Cluster RAM quota required / per_node_ram_quota`

Important: You will need at least the number of replicas + 1 nodes regardless of your data size.

The following is a sample sizing calculation:

Input Variable	value
documents_num	1,000,000
ID_size	100
value_size	10,000
number_of_replicas	1
working_set_percentage	20%

Constants	value
Type of Storage	SSD
overhead_percentage	25%
metadata_per_document	56 for 2.1 and higher, 64 for 2.0.x
high_water_mark	85%

Variable	Calculation
no_of_copies	= 1 for original and 1 for replica
total_metadata	= 1,000,000 * (100 + 56) * (2) = 312,000,000
total_dataset	= 1,000,000 * (10,000) * (2) = 20,000,000,000
working_set	= 20,000,000,000 * (0.2) = 4,000,000,000
Cluster RAM quota required	= (440,000,000 + 4,000,000,000) * (1+0.25)/(0.7) = 7,928,000,000

For example, if you have 8GB machines and you want to use 6 GB for Couchbase…

```
number of nodes =
    Cluster RAM quota required/per_node_ram_quota =
    7.9 GB/6GB = 1.3 or 2 nodes
```

Disk throughput and sizing

Couchbase Server decouples RAM from the I/O layer. Decoupling allows high scaling at very low and consistent latencies and enables very high write loads without affecting client application performance.

Couchbase Server implements an append-only format and a built-in automatic compaction process. Previously, in Couchbase Server 1.8.x, an “in-place-update” disk format was implemented, however, this implementation occasionally produced a performance penalty due to fragmentation of the on-disk files under workloads with frequent updates/deletes.

The requirements of your disk subsystem are broken down into two components: size and IO.

Size

Disk size requirements are impacted by the Couchbase file write format, append-only, and the built-in automatic compaction process. Append-only format means that every write (insert/update/delete) creates a new entry in the file(s).

The required disk size increases from the update and delete workload and then shrinks as the automatic compaction process runs. The size increases because of the data expansion rather than the actual data using more disk space. Heavier update and delete workloads increases the size more dramatically than heavy insert and read workloads.

Size recommendations are available for key-value data only. If views and indexes or XDCR are implemented, contact Couchbase support for analysis and recommendations.

Depending on the workload, the required disk size is **2–3x** your total dataset size (active and replica data combined).

Important: The disk size requirement of 2-3x your total dataset size applies to key-value data only and does not take into account other data formats and the use of views and indexes or XDCR.

IO

IO is a combination of the sustained write rate, the need for compacting the database files, and anything else that requires disk access. Couchbase Server automatically buffers writes to the database in RAM and eventually persists them to disk. Because of this, the software can accommodate much higher write rates than a disk is able to handle. However, sustaining these writes eventually requires enough IO to get it all down to disk.

To manage IO, configure the thresholds and schedule when the compaction process kicks in or doesn't kick in keeping in mind that the successful completion of compaction is critical to keeping the disk size in check. Disk size and disk IO become critical to size correctly when using views and indexes and cross-data center replication (XDCR) as well as taking backup and anything else outside of Couchbase that need space or is accessing the disk.

Best practice

Use the available configuration options to separate data files, indexes and the installation/config directories on separate drives/devices to ensure that IO and space are allocated effectively.

Network bandwidth

Network bandwidth is not normally a significant factor to consider for cluster sizing. However, clients require network bandwidth to access information in the cluster. Nodes also need network bandwidth to exchange information (node to node).

In general, calculate your network bandwidth requirements using the followingformula:

```
Bandwidth = (operations per second * item size) + overhead for rebalancing
```

Calculate the `operations per second` with the following formula:

```
Operations per second = Application reads + (Application writes * Replica
 copies)
```

Data safety

Make sure you have enough nodes (and the right configuration) in your cluster to keep your data safe. There are two areas to keep in mind: how you distribute data across nodes and how many replicas you store across your cluster.

Data distribution

Basically, more nodes are better than less. If you only have two nodes, your data is split across the two nodes, half and half. This means that half of your dataset is impacted if one goes away. On the other hand, with ten nodes, only

10% of the dataset is impacted if one nodes goes away. Even with automatic failover, there still is some period of time when data is unavailable if nodes fail. This is mitigated by having more nodes.

After a failover, the cluster takes on an extra load. The question is - how heavy is that extra load and are you prepared for it? Again, with only two nodes, each one needs to be ready to handle the entire load. With ten, each node only needs to be able to take on an extra tenth of the workload should one fail.

While two nodes does provide a minimal level of redundancy, we recommend that you always use at least three nodes.

Replication

Couchbase Server allows you to configure up to three replicas (creating four copies of the dataset). In the event of a failure, you can only "failover" (either manually or automatically) as many nodes as you have replicas. For example:

- In a five node cluster with one replica, if one node goes down, you can fail it over. If a second node goes down, you no longer have enough replica copies to fail over to and will have to go through a slower process to recover.
- In a five node cluster with two replicas, if one node goes down, you can fail it over. If a second node goes down, you can fail it over as well. Should a third one go down, you now no longer have replicas to fail over.

After a node goes down and is failed over, try to replace that node as soon as possible and rebalance. The rebalance recreates the replica copies (if you still have enough nodes to do so).

Tip: As a rule of thumb, configure the following:

- One replica for up to five nodes.
- One or two replicas for five to ten nodes.
- One, two, or three replicas for over ten nodes.

While there may be variations to this, there are diminishing returns from having more replicas in smaller clusters.

Hardware requirements

In general, Couchbase Server has very low hardware requirements and is designed to be run on commodity or virtualized systems. However, as a rough guide to the primary concerns for your servers, the following is recommended:

- RAM: This is your primary consideration. We use RAM to store active items, and that is the key reason Couchbase Server has such low latency.
- CPU: Couchbase Server has very low CPU requirements. The server is multi-threaded and therefore benefits from a multi-core system. We recommend machines with at least four or eight physical cores.
- Disk: By decoupling the RAM from the I/O layer, Couchbase Server can support low-performance disks better than other databases. As a best practice, have separate devices for server install, data directories, and index directories.
- Network: Most configurations work with Gigabit Ethernet interfaces. Faster solutions such as 10GBit and Infiniband will provide spare capacity.

Known working configurations include SAN, SAS, SATA, SSD, and EBS with the following recommendations:

- SSDs have been shown to provide a great performance boost both in terms of draining the write queue and also in restoring data from disk (either on cold-boot or for purposes of rebalancing).
- RAID generally provides better throughput and reliability.
- Striping across EBS volumes (in Amazon EC2) has been shown to increase throughput.

Considerations for Cloud environments (i.e. Amazon EC2)

Due to the unreliability and general lack of consistent I/O performance in cloud environments, we highly recommend lowering the per-node RAM footprint and increasing the number of nodes. This will give better disk throughput as well as improve rebalancing since each node will have to store (and therefore transmit) less data. By distributing the data further, it lessens the impact of losing a single node (which could be fairly common).

Related Links

Deployment considerations on page 119
Deployment configuration take into account topics such as restricted access, node communication, swap configuration, and connection timeouts.

Related topics

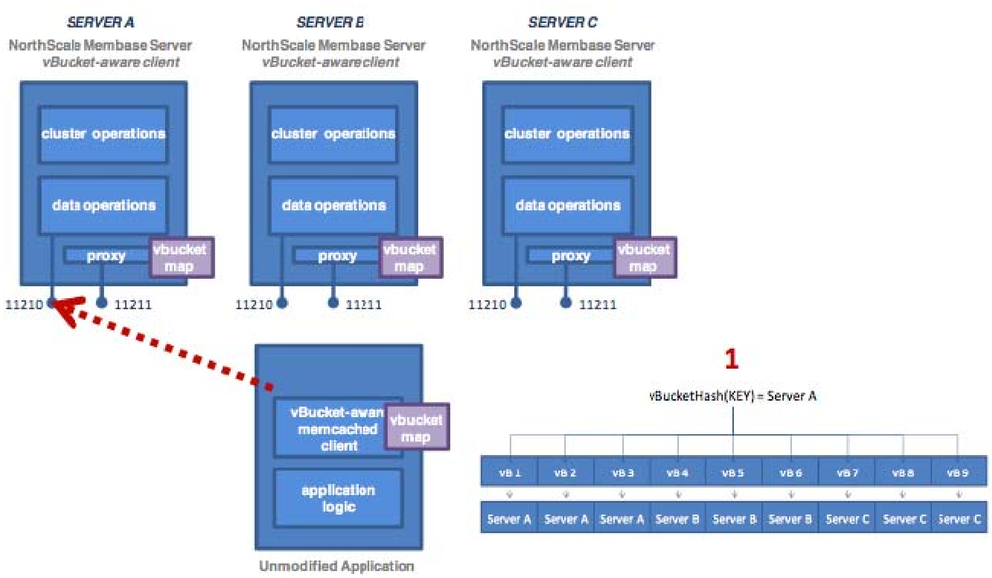
Couchbase in the cloud on page 128
Couchbase Server is extremely easy to deploy in the cloud.

Deployment strategies

Here are a number of deployment strategies that you may want to use. Smart clients are the preferred deployment option if your language and development environment supports a smart client library. If not, use the client-side Moxi configuration for the best performance and functionality.

Using a smart (vBucket-aware) client

When using a smart client, the client library provides an interface to the cluster and performs server selection directly via the vBucket mechanism. The clients communicate with the cluster using a custom Couchbase protocol. This allows the clients to share the vBucket map, locate the node containing the required vBucket, and read and write information from there.



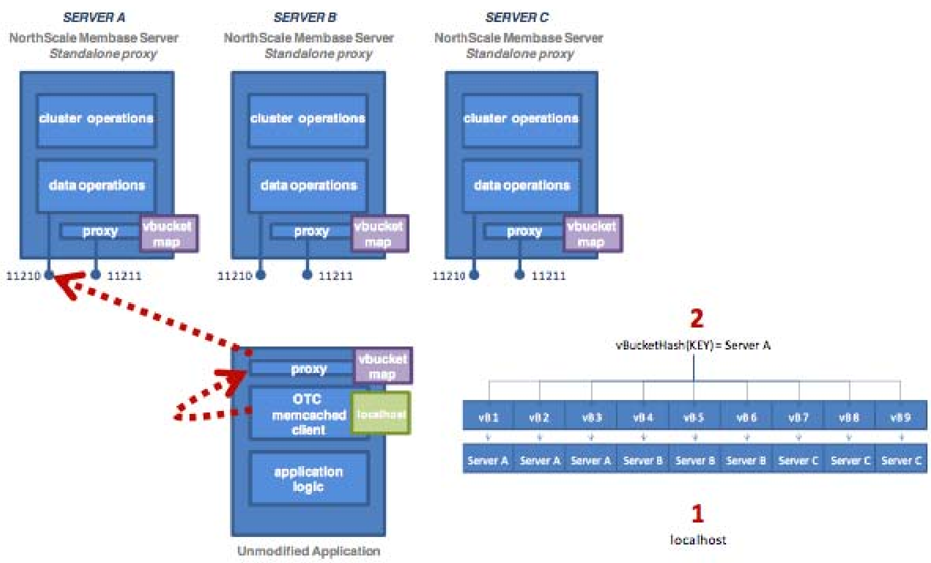
In releases prior to Couchbase Server 2.5, a developer, via a client library of their choice, randomly selects a host from which to request an initial topology configuration. Any future changes to the cluster map following the initial bootstrap are based on the NOT_MY_VBUCKET response from the server. This connection is made to port 8091 and is based on an HTTP connection.

With Couchbase Server 2.5 or higher, client libraries query a cluster for initial topology configuration for a bucket from one of the nodes in the cluster. This is similar to prior releases. However, this information is transmitted via the memcached protocol on port 11210 (rather than via persistent HTTP connections to port 8091). This significantly improves connection scaling capabilities.

Note: This change is only applicable to Couchbase-type buckets (not memcached buckets). An error is returned if a configuration request is received on port 8091.

Client-side (standalone) proxy

If a smart client is not available for your chosen platform, you can deploy a standalone proxy. This provides the same functionality as the smart client while presenting a memcached compatible interface layer locally. A standalone proxy deployed on a client may also be able to provide valuable services, such as connection pooling. The diagram below shows the flow with a standalone proxy installed on the application server.



We configured the memcached client to have just one server in its server list (localhost), so all operations are forwarded to `localhost:11211` — a port serviced by the proxy. The proxy hashes the document ID to a vBucket, looks up the host server in the vBucket table, and then sends the operation to the appropriate Couchbase Server on port 11210.

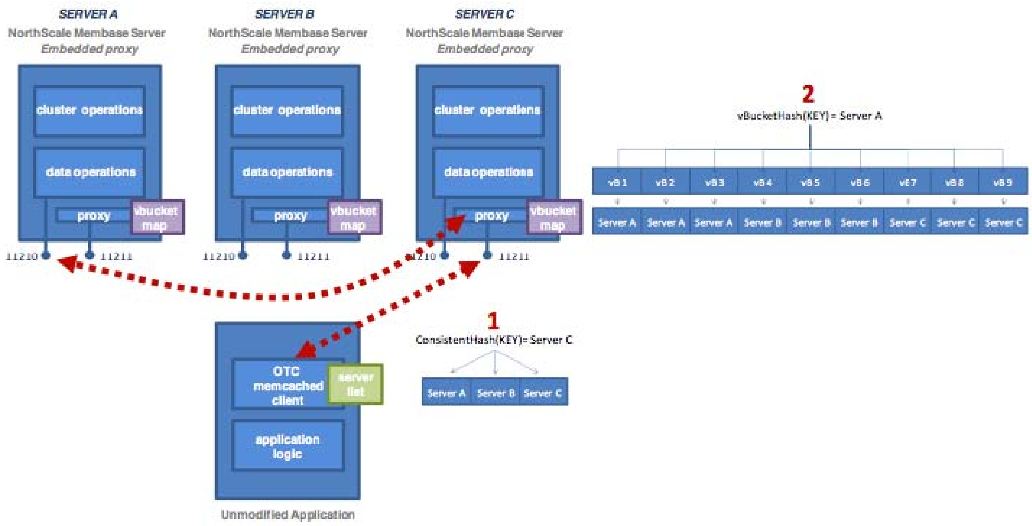
For the corresponding Moxi product, use the Moxi 1.8 series.

Using server-side (Couchbase embedded) proxy

Attention: Server-side proxy configuration is *not* recommended for production use. Use either a smart client or the client-side proxy configuration unless your platform and environment do not support that deployment type.

The server-side (embedded) proxy exists within Couchbase Server using port 11211. It supports the memcached protocol and allows an existing application to communicate with Couchbase Cluster without installing another piece of proxy software. The downside to this approach is performance.

In this deployment option versus a typical memcached deployment, in a worse-case scenario, server mapping happens twice (e.g. using Ketama hashing to a server list on the client, then using vBucket hashing and server mapping on the proxy) with an additional round trip network hop introduced.



Related Links

Deployment considerations on page 119

Deployment configuration take into account topics such as restricted access, node communication, swap configuration, and connection timeouts.

Related topics

vBucket article

Moxi 1.8 Manual

Ongoing monitoring and maintenance

There are a number of different statistics

To understand how your cluster is working and whether it is working effectively, there are a number of different statistics that you should monitor to diagnose and identify problems.

Key statistics include the following:

- Memory Used (`mem_used`) - This is the current size of memory used. If `mem_used` hits the RAM quota then you will get `OOM_ERROR`. The `mem_used` must be less than `ep_mem_high_wat`, which is the mark at which data is ejected from the disk.
- Disk Write Queue Size (`ep_queue_size`) - This is the amount of data waiting to be written to disk.
- Cache Hits (`get_hits`) - As a rule of thumb, this should be at least 90% of the total requests.
- Cache Misses (`get_misses`) - Ideally this should be low, and certainly lower than `get_hits`. Increasing or high values mean that data that your application expects to be stored is not in memory.

The water mark is another key statistic to monitor cluster performance. The ‘water mark’ determines when it is necessary to start freeing up available memory. Two important statistics related to water marks include:

- High Water Mark (`ep_mem_high_wat`) - The system starts ejecting data out of memory when this water mark is met. Ejected values need to be fetched from disk when accessed before being returned to the client.
- Low Water Mark (`ep_mem_low_wat`) - When the low water mark threshold is reached, it indicates that memory usage is moving toward a critical point and system administration action is should be taken before the high water mark is reached.

Tip: Use the following command to get statistic information :

```
shell> cbstats IP:11210 all | \
    egrep "todo|ep_queue_size|_eject|mem|max_data|hits|misses"
```

The following statistic information is provided:

```
ep_flusher_todo:
ep_max_data_size:
```

```
ep_mem_high_wat:
ep_mem_low_wat:
ep_num_eject_failures:
ep_num_value_ejects:
ep_queue_size:
mem_used:
get_misses:
get_hits:
```

Tip: Monitor the disk space, CPU usage, and swapping on all nodes, using the standard monitoring tools.

Important UI stats to watch

You can add the following graphs to watch on the Couchbase console. These graphs can be de/selected by clicking on the **Configure View** link at the top of the **Bucket Details** on the Couchbase web console.

- Disk write queues - The value should not keep growing. The actual numbers depend on your application and deployment.
- Ram ejections - There should be no sudden spikes.
- Vbucket errors - An increasing value for vBucket errors is bad.
- OOM errors per sec - This should be 0.
- Temp OOM errors per sec - This should be 0.
- Connections count - This should remain flat in a long running deployment.
- Get hits per second
- Get misses per second - This should be much lower than Get hits per second.

Related Links

Deployment considerations on page 119
Deployment configuration take into account topics such as restricted access, node communication, swap configuration, and connection timeouts.

Related topics

Disk storage on page 102
Couchbase Server mainly stores and retrieves information for clients using RAM. At the same time, Couchbase Server eventually stores all data to disk to provide a higher level of reliability.

Couchbase behind a secondary firewall

If Couchbase is being deployed behind a secondary firewall, ensure that the reserved Couchbase network ports are open.

Related Links

Deployment considerations on page 119
Deployment configuration take into account topics such as restricted access, node communication, swap configuration, and connection timeouts.

Related topics

Network ports on page 51
Couchbase Server specific network ports for communication between server components and for communicating with the clients that access the data stored in the Couchbase cluster.

Couchbase in the cloud

Couchbase Server is extremely easy to deploy in the cloud.

From the software's perspective, there is really no difference between being installed on bare-metal or virtualized operating systems. On the other hand, the management and deployment characteristics of the cloud warrant a separate discussion on the best ways to use Couchbase.

For the purposes of this discussion, “the cloud” is referred to as Amazon’s EC2 environment since that is by far the most common cloud-based environment. However, the same considerations apply to any environment that acts like EC2 (an organization’s private cloud for example). In terms of the software itself, extensive testing has been done within EC2 (and some of Couchbase's largest customers have already deployed Couchbase there for production use). Because of this, we have encountered and resolved a variety of bugs only exposed by the sometimes unpredictable characteristics of this environment.

We have written a number of RightScale templates to help you deploy within Amazon. Sign up for a free RightScale account to try it out. The templates handle almost all of the special configuration needed to make your experience within EC2 successful. Direct integration with RightScale also allows us to do some pretty cool things with auto-scaling and pre-packaged deployment.

We’ve also authored an AMI for use within EC2 independent of RightScale. When using these, you will have to handle the specific complexities yourself. You can find this AMI by searching for ‘couchbase’ in Amazon’s EC2 portal.

When deploying within the cloud, consider the following areas:

- Local storage being ephemeral
- IP addresses of a server changing from runtime to runtime
- Security groups/firewall settings
- Swap Space

Handling instance reboot in cloud

Many cloud providers warn users that they need to reboot certain instances for maintenance. Couchbase Server ensures these reboots won’t disrupt your application. Take the following steps to make that happen:

1. Install Couchbase on the new node.
2. From the user interface, add the new node to the cluster.
3. From the user interface, remove the node that you wish to reboot.
4. Rebalance the cluster.
5. Shut down the instance.

Local storage

Dealing with local storage is not very much different than a data center deployment. However, EC2 provides an interesting solution. Through the use of EBS storage, you can prevent data loss when an instance fails. Writing Couchbase data and configuration to EBS creates a reliable medium of storage. There is direct support for using EBS within RightScale and, of course, you can set it up manually.

Using EBS is definitely not required, but you should make sure to follow the best practices around performing backups.

Keep in mind that you will have to update the per-node disk path when configuring Couchbase to point to wherever you have mounted an external volume.

Handling changes in IP addresses

When you use Couchbase Server in the cloud, server nodes can use internal or public IP addresses. Because IP addresses in the cloud may change quite frequently, you can configure Couchbase to use a hostname instead of an IP address.

For Amazon EC2 we recommend you use Amazon-generated hostnames which then will automatically resolve to either the internal or external address.

By default Couchbase Servers use specific IP addresses as a unique identifier. If the IP changes, an individual node will not be able to identify its own address, and other servers in the same cluster will not be able to access it. To configure Couchbase Server instances in the cloud to use hostnames, follow the steps later in this section. Note that RightScale server templates provided by Couchbase can automatically configure a node with a provided hostname.

Make sure that your hostname always resolves to the IP address of the node. This can be accomplished by using a dynamic DNS service such as DNSMadeEasy which will allow you to automatically update the hostname when an underlying IP address changes.

The following steps completely destroys any data and configuration from the node, so you should start with a fresh Couchbase install. If you already have a running cluster, you can rebalance a node out of the cluster, make the change, and then rebalance it back into the cluster.

Nodes with both IPs and hostnames can exist in the same cluster. When you set the IP address using this method, you should not specify the address as `localhost` or `127.0.0.1` as this will be invalid when used as the identifier for multiple nodes within the cluster. Instead, use the correct IP address for your host.

Linux and Windows 2.1 and above

As a rule, you should set the hostname before you add a node to a cluster. You can also provide a hostname in these ways: when you install a Couchbase Server node or when you do a REST API call before the node is part of a cluster. You can also add a hostname to an existing cluster for an online upgrade. If you restart, any hostname you establish with one of these methods will be used.

Linux and Windows 2.0.1 and earlier

For Couchbase Server 2.0.1 and earlier you must follow a manual process where you edit config files for each node which we describe below for Couchbase in the cloud.

Security groups/firewall settings

It's important to make sure you have both allowed AND restricted access to the appropriate ports in a Couchbase deployment. Nodes must be able to talk to one another on various ports, and it is important to restrict external and/or internal access to only authorized individuals. Unlike a typical data center deployment, cloud systems are open to the world by default, and steps must be taken to restrict access.

Swap space

On Linux, swap space is used when the physical memory (RAM) is full. If the system needs more memory resources and the RAM is full, inactive pages in memory are moved to the swap space. Swappiness indicates how frequently a system should use swap space based on RAM usage. The swappiness range is from 0 to 100 where, by default, most Linux platforms have swappiness set to 60.

Recommendation

For optimal Couchbase Server operations, set the swappiness to **0** (zero).

To change the swap configuration:

1. Execute `cat /proc/sys/vm/swappiness` on each node to determine the current swap usage configuration.
2. Execute `sudo sysctl vm.swappiness=0` to immediately change the swap configuration and ensure that it persists through server restarts.
3. Using sudo or root user privileges, edit the kernel parameters configuration file, `/etc/sysctl.conf`, so that the change is always in effect.
4. Append `vm.swappiness = 0` to the file.
5. Reboot your system.

Note: Executing `sudo sysctl vm.swappiness=0` ensures that the operating system no longer uses swap unless memory is completely exhausted. Updating the kernel parameters configuration file, `sysctl.conf`, ensures that the operating system always uses swap in accordance with Couchbase recommendations even when the node is rebooted.

Using Couchbase Server on RightScale

Couchbase partners with RightScale to provide preconfigured RightScale ServerTemplates that you can use to create an individual or array of servers and start them as a cluster. Couchbase Server RightScale ServerTemplates enable you to quickly set up Couchbase Server on Amazon Elastic Compute Cloud (Amazon EC2) servers in the Amazon Web Services (AWS) cloud through RightScale.

The templates also provide support for Amazon Elastic Block Store (Amazon EBS) standard volumes and Provisioned IOPS volumes. (IOPS is an acronym for input/output operations per second.) For more information about Amazon EBS volumes and their capabilities and limitations, see Amazon EBS Volume Types.

Couchbase provides RightScale ServerTemplates based on Chef and, for compatibility with existing systems, non-Chef-based ServerTemplates.

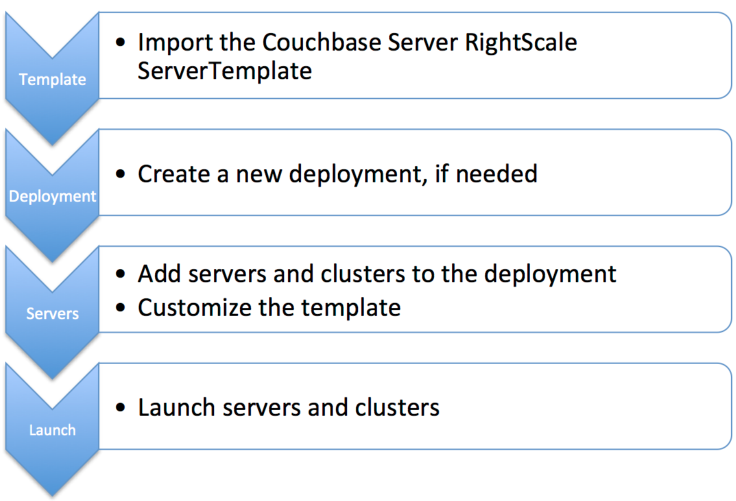
Note: As of Couchbase Server 2.2, non-Chef templates are deprecated. Do not choose non-Chef templates for new installations.

Before you can set up Couchbase Server on RightScale, you need a RightScale account and an AWS account that is connected to your RightScale account.

At a minimum, you need RightScale user role privileges to work with the Couchbase RightScale ServerTemplates: actor, designer, library, observer, and server_login. To add privileges: from the RightScale menu bar, click **Settings** > **Account Settings** > **Users** and modify the permission list.

To set up Couchbase Server on RightScale, you need to import and customize a ServerTemplate. After the template is customized, you can launch server and cluster instances. The following figure illustrates the workflow:

Creating Couchbase Server Instances on RightScale



The following procedures do not describe every parameter that you can modify when working with the RightScale ServerTemplates. If you need more information about a parameter, click the info button located near the parameter name.

To import the Couchbase Server RightScale ServerTemplate:

1. From the RightScale menu bar, select **Design** > **MultiCloud Marketplace** > **ServerTemplates**.
2. In the **Keywords** box on the left under Search, type **couchbase**, and then click **Go**.
3. In the search results list, click on the latest version of the Couchbase Server ServerTemplate.

   The name of each Couchbase template in the list contains the Couchbase Server version number.
4. Click **Import**.
5. Review each page of the end user license agreement, and then click **Finish** to accept the agreement.

To create a new deployment:

1. From the RightScale menu bar, select **Manage** > **Deployments** > **New**.
2. Enter a Nickname and Description for the new deployment.
3. Click **Save**.

To add a server or cluster to a deployment:

1. From the RightScale menu bar, select **Manage > Deployments**.
2. Click the nickname of the deployment that you want to place the server or cluster in.
3. From the deployment page menu bar, add the server or cluster:
    - To add a server, click **Add Server**.
    - To add a cluster, click **Add Array**.
4. In the Add to Deployment window, select a cloud and click **Continue**.
5. On the Server Template page, select a template from the list.

    If you have many server templates in your account, you can reduce the number of entries in the list by typing a keyword from the template name into the Server Template Name box under Filter Options.
6. Click **Server Details**.
7. On the **Server Details** page, choose settings for Hardware:

    Server Name or **Array Name**—Enter a name for the new server or array.

    Instance Type—The default is extra large. The template supports only large or extra large instances and requires a minimum of 4 cores.

    EBS Optimized—Select the check box to enable EBS-optimized volumes for Provisioned IOPS.
8. Choose settings for Networking:
    - **SSH Key**—Choose an SSH key.
    - **Security Groups**—Choose one or more security groups.
9. If you are adding a cluster, click **Array Details**, and then choose settings for Autoscaling Policy and Array Type Details.

    Under Autoscaling Policy, you can set the minimum and maximum number of active servers in the cluster by modifying the **Min Count** and **Max Count** parameters. If you want a specific number of servers, set both parameters to the same value.
10. Click **Finish**.

To customize the template for a server or a cluster:

1. From the RightScale menu bar, select **Manage > Deployments**.
2. Click the nickname of the deployment that the server or cluster is in.
3. Click the nickname of the server or cluster.
4. On the Server or Server Array page, click the **Inputs** tab, and then click **edit**.
5. Expand the **BLOCK_DEVICE** category and modify inputs as needed.

    The BLOCK_DEVICE category contains input parameters that are specific to storage. Here's a list of some advanced inputs that you might want to modify:
    - **I/O Operations per Second**—Number of input/output operations per second (IOPS) that the volume can support
    - **Volume Type**—Type of storage device
6. Expand the **DB_COUCHBASE** category and modify inputs as needed.

    The DB_COUCHBASE category contains input parameters that are specific to Couchbase Server. In general, the default values are suitable for one server. If you want to create a cluster, you need to modify the input parameter values. Here's a list of the advanced inputs that you can modify:
    - **Bucket Name**—Name of the bucket. The default bucket name is `default`.
    - **Bucket Password**—Password for the bucket.
    - **Bucket RAM Quota**—RAM quota for the bucket in MB.
    - **Bucket Replica Count**—Bucket replica count.
    - **Cluster REST/Web Password**—Password for the administrator account. The default is `password`.

- **Cluster REST/Web Username**—Administrator account user name for access to the cluster via the REST or web interface. The default is `Administrator`.
- **Cluster Tag**—Tag for nodes in the cluster that are automatically joined.
- **Couchbase Server Edition**—The edition of Couchbase Server. The default is `enterprise`.
- **Rebalance Count**—The number of servers to launch before doing a rebalance. Set this value to the total number of target servers you plan to have in the cluster. If you set the value to 0, Couchbase Server does a rebalance after each server joins the cluster.

7. Click **Save**.
8. If you are ready to launch the server or cluster right now, click **Launch**.

To launch servers or clusters:

1. From the RightScale menu bar, select **Manage > Deployments**.
2. Click the nickname of the deployment that the server or cluster is in.
3. Click the nickname of the server or cluster.
4. On the Server or Server Array page, click **Launch**.

To log in to the Couchbase Web Console:

You can log in to the Couchbase Web Console by using your web browser to connect to the public IP address on port 8091. The general format is `http://<server:port>`. For example: if the public IP address is 192.236.176.4, enter `http://192.236.176.4:8091/` in the web browser location bar.

Related Links

Deployment considerations on page 119
Deployment configuration take into account topics such as restricted access, node communication, swap configuration, and connection timeouts.

Related topics

Couchbase setup overview on page 49
Initial server setup on page 66
Upgrading on page 78
RightScale
Couchbase on RightScale
RightScale
Amazon Web Services
Amazon Elastic Compute Cloud
Amazon Elastic Block Store
Amazon EBS Volume Types
Chef
Add AWS Credentials to RightScale
RightScale user role privileges

XDCR in cloud deployments

If you want to use XDCR within a cloud deployment to replicate between two or more clusters that are deployed in the cloud, there are some additional configuration requirements:

- Use a public DNS names and public IP addresses for nodes in your clusters.

Cloud services support the use of a public IP address to allow communication to the nodes within the cluster. Within the cloud deployment environment, the public IP address will resolve internally within the cluster, but allow external communication. In Amazon EC2, for example, ensure that you have enabled the public interface in your instance configuration, that the security parameters allow communication to the required ports, and that public DNS record exposed by Amazon is used as the reference name.

Configure the cluster with a fixed IP address and the public DNS name according to the recommendations for using Couchbase in the cloud.

- Use a DNS service to identify or register a CNAME that points to the public DNS address of each node within the cluster. This will allow you to configure XDCR to use the CNAME to a node in the cluster. The CNAME will be constant, even though the underlying public DNS address may change within the cloud service.

The CNAME record entry can then be used as the destination IP address when configuring replication between the clusters using XDCR. If a transient failure causes the public DNS address for a given cluster node to change, update the CNAME to point to the updated public DNS address provided by the cloud service.

By updating the CNAME records, replication should be able to persist over a public, internet- based connection, even though the individual IP of different nodes within each cluster configured in XDCR.

For additional security, you should configure your security groups to allow traffic only on the required ports between the IP addresses for each cluster. To configure security groups, you will need to specify the inbound port and IP address range. You will also need to ensure that the security also includes the right port and IP addresses for the remainder of your cluster to allow communication between the nodes within the cluster.

Related Links

Deployment considerations on page 119
Deployment configuration take into account topics such as restricted access, node communication, swap configuration, and connection timeouts.

Related topics

Couchbase setup overview on page 49

Network ports on page 51
Couchbase Server specific network ports for communication between server components and for communicating with the clients that access the data stored in the Couchbase cluster.

Couchbase in the cloud on page 128
Couchbase Server is extremely easy to deploy in the cloud.

Cluster management

The core administrative aspects of Couchbase Server center around managing the Couchbase Server cluster.

Related Links

Couchbase Administration on page 89

Handling server warmup on page 135
Couchbase server warmup behavior can be modified changing the access scanner and warmup threshold settings via the `cbepctl` tool.

Handling replication on page 136
Data replication is distributed throughout the Couchbase cluster to prevent a single point of failure. Data replication is configurable on a bucket-level and node-basis.

Compaction on page 138
Database and view compaction helps to reclaim disk space and reduce fragmentation.

Failing over nodes on page 141
Failing over a node means that Couchbase Server removes the node from a cluster and makes replicated data at other nodes available for client requests.

Data recovery from remote clusters on page 146
Data recovery from remote clusters requires an XDCR environment and adequate amount of memory and disk space to support the workload and recovered data.

Backup and restore on page 150
The predominant method for backing up and restoring data is using the CLI `cbbackup` and `cbrestore` tools.

Rebalancing on page 158

Managing XDCR on page 169
Cross datacenter replication (XDCR) provides an easy method of replicating data from one cluster to another for disaster recovery as well as better data locality (getting data closer to its users).

Handling server warmup

Couchbase server warmup behavior can be modified changing the access scanner and warmup threshold settings via the `cbepctl` tool.

In order to adjust warmup behavior, it is also important for you to understand the access log and scanning process in Couchbase Server. The server uses the access log to determine which documents are most frequently used, and therefore which documents should be loaded first.

The server has a process that periodically scans every key in RAM and compile them into a log, named `access.log` as well as maintain a backup of this access log, named `access.old`. The server can use this backup file during warmup if the most recent access log has been corrupted during warmup or node failure. By default this process runs initially at 2:00 GMT and runs again in 24 hour time periods after that point. You can configure this process to run at a different initial time and at a different fixed interval.

If a client tries to contact Couchbase Server during warmup, the server produces a `ENGINE_TMPFAIL (0x0d)` error code. This error indicates that data access is still not available because warmup has not yet finished.

> **Note:** For those of you who are creating your own Couchbase SDK, you will need to handle this error in your library. This may mean that the client waits and retries, performs a backoff of requests, or produces an error and does not retry the request.

> **Note:** For those of you who are building an application with a Couchbase SDK, be aware that how this error is delivered and handled is dependent upon the individual SDKs.

Getting warmup information

The `cbstats` tool is used to get information about server warmup, including the warmup status and whether warmup is enabled.

Changing the warmup threshold

Modify warmup behavior by changing the `cbepctl ep_warmup_min_items_threshold` parameter. This indicates the percentage of items loaded in RAM that must be reached for Couchbase Server to begin serving data. The lower this number, the sooner your server can begin serving data. Be aware, however that if you set this value to be too low, once requests come in for items, the item may not be in memory and Couchbase Server will experience cache-miss errors.

Changing access scanner settings

The server runs a periodic scanner process which determines which keys are most frequently-used, and therefore, which documents should be loaded first during server warmup. The settings, `cbepctl flush_param alog_sleep_ time` and `alog_task_ time`parameters are used to change the initial time and the interval for the process.

Related Links

Cluster management on page 135

Related topics

cbepctl tool on page 379

Changing access log settings on page 382

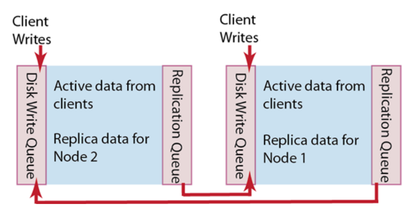
Getting warmup information on page 398

Handling replication

Data replication is distributed throughout the Couchbase cluster to prevent a single point of failure. Data replication is configurable on a bucket-level and node-basis.

Within a Couchbase cluster, you have *replica data* which is a copy of an item at another node. After writing an item to Couchbase Server, it makes a copy of this data from the RAM of one node to another node. Distribution of replica data is handled in the same way as active data; portions of replica data will be distributed around the Couchbase cluster onto different nodes to prevent a single point of failure. Each node in a cluster will have *replica data* and *active data*. Replica data is the copy of data from another node while active data is data that had been written by a client on that node.

Replication of data between nodes is entirely peer-to-peer based; information will be replicated directly between nodes in the cluster. There is no topology, hierarchy or master-slave relationship between nodes in a cluster. When a client writes to a node in the cluster, Couchbase Server stores the data on that node and then distributes the data to one or more nodes within a cluster. The following shows two different nodes in a Couchbase cluster, and illustrates how two nodes can store replica data for one another:



When a client application writes data to a node, that data will be placed in a replication queue and then a copy will be sent to another node. The replicated data will be available in RAM on the second node and will be placed in a disk write queue to be stored on disk at the second node.

Notice that a second node will also simultaneously handle both replica data and incoming writes from a client. The second node will put both replica data and incoming writes into a disk write queue. If there are too many items in the disk write queue, this second node can send a *backoff message* to the first node. The first node will then reduce the rate at which it sends items to the second node for replication. This can sometimes be necessary if the second node is already handling a large volume of writes from a client application.

If multiple changes occur to the same document waiting to be replicated, Couchbase Server is able to *de-duplicate, or 'de-dup'* the item; this means for the sake of efficiency, it will only send the latest version of a document to the second node.

If the first node fails in the system the replicated data is still available at the second node. Couchbase can serve replica data from the second node nearly instantaneously because the second node already has a copy of the data in RAM; there is no need for the data to be copied over from the failed node or to be fetched from disk. Once replica data is enabled at the second node, Couchbase Server updates a map indicating where the data should be retrieved, and the server shares this information with client applications. Client applications can then get the replica data from the functioning node.

Providing data replication

You can configure data replication for each bucket in cluster. You can also configure different buckets to have different levels of data replication, depending how many copies of your data you need. For the highest level of data redundancy and availability, you can specify that a data bucket will be replicated three times within the cluster.

Replication is enabled once the number of nodes in your cluster meets the number of replicas you specify. For example, if you configure three replicas for a data bucket, replication will only be enabled once you have four nodes in the cluster.

Note

After you specify the number of replicas you want for a bucket and then create the bucket, you cannot change this value. Therefore be certain you specify the number of replicas you truly want.

Specifying backoff for replication

Your cluster is set up to perform some level of data replication between nodes within the cluster for any given node. Every node will have both *active data* and *replica data*. Active data is all the data that had been written to the node from a client, while replica data is a copy of data from another node in the cluster. Data replication enables high availability of data in a cluster. Should any node in cluster fail, the data will still be available at a replica.

On any give node, both active and replica data must wait in a disk write queue before being written to disk. If you node experiences a heavy load of writes, the replication queue can become overloaded with replica and active data waiting to be persisted.

By default a node will send backoff messages when the disk write queue on the node contains one million items or 10%. When other nodes receive this message, they will reduce the rate at which they send replica data. You can configure this default to be a given number so long as this value is less than 10% of the total items currently in a replica partition. For instance if a node contains 20 million items, when the disk write queue reaches 2 million items a backoff message will be sent to nodes sending replica data. You use the Couchbase command-line tool, `cbepctl` to change this configuration:

In the following example, a node sends replication backoff requests when it has two million items or 10% of all items, whichever is greater.

```
> ./cbepctl 10.5.2.31:11210 -b bucket_name -p bucket_password set tap_param
 tap_throttle_queue_cap 2000000
```

In the following example, the default percentage, used to manage the replication stream, is changed. If the items in a disk write queue reach the greater of this percentage or a specified number of items, replication requests slow down:

```
setting param: tap_throttle_queue_cap 2000000
```

In the following example, the threshold is set to 15% of all items at a replica node. When a disk write queue on a node reaches this point, it sends replication backoff requests to other nodes.

```
> ./cbepctl  10.5.2.31:11210 set -b bucket_name tap_param
 tap_throttle_cap_pcnt 15
```

Important: Be aware that this tool is a per-node, per-bucket operation. That means that if you want to perform this operation, you must specify the IP address of a node in the cluster and a named bucket. If you do not provided a named bucket, the server applies the setting to any default bucket that exists at the specified node. If you want to perform this operation for an entire cluster, perform the command for every node/bucket combination that exists for that cluster.

You can also monitor the progress of this backoff operation in Couchbase Web Console under Tap Queue Statistics | back-off rate.

Related Links

Cluster management on page 135

Related topics

Failing over nodes on page 141
Failing over a node means that Couchbase Server removes the node from a cluster and makes replicated data at other nodes available for client requests.

Rebalancing on page 158

Data Buckets on page 259

cbepctl tool on page 379

Changing disk write queue quotas on page 383

Monitoring statistics on page 266
Within the **Data Bucket** tab, information and statistics about bucketsand nodes is displayed for the entire Couchbase Server cluster. The information is aggregated from all the server nodes within the configured cluster for the selected bucket.

Compaction

Database and view compaction helps to reclaim disk space and reduce fragmentation.

The data files in which information is stored in a persistent state for a Couchbase bucket are written to and updated as information is appended, updated and deleted. This process eventually leads to gaps within the data file (particularly when data is deleted) which can be reclaimed using a process called compaction.

The index files that are created each time a view is built are also written in a sequential format. Updated index information is appended to the file as updates to the stored information is indexed.

In both these cases, frequent compaction of the files on disk can help to reclaim disk space and reduce fragmentation.

Compaction process

How it works

Couchbase compacts views and data files. For database compaction, a new file is created into which the active (non-stale) information is written. Meanwhile, the existing database files stay in place and continue to be used for storing information and updating the index data. This process ensures that the database continues to be available while compaction takes place. Once compaction is completed, the old database is disabled and saved. Then any incoming updates continue in the newly created database files. The old database is then deleted from the system.

View compaction occurs in the same way. Couchbase creates a new index file for each active design document. Then Couchbase takes this new index file and writes active index information into it. Old index files are handled in the same way old data files are handled during compaction. Once compaction is complete, the old index files are deleted from the system.

How to use it

Compaction takes place as a background process while Couchbase Server is running. You do not need to shutdown or pause your database operation, and clients can continue to access and submit requests while the database is running. While compaction takes place in the background, you need to pay attention to certain factors.

Make sure you perform compaction…

- **… on every server:** Compaction operates on only a single server within your Couchbase Server cluster. You will need to perform compaction on each node in your cluster, on each database in your cluster.

- **… during off-peak hours:** The compaction process is both disk and CPU intensive. In heavy-write based databases where compaction is required, the compaction should be scheduled during off-peak hours (use auto-compact to schedule specific times).

If compaction isn't scheduled during off-peak hours, it can cause problems. Because the compaction process can take a long to complete on large and busy databases, it is possible for the compaction process to fail to complete properly while the database is still active. In extreme cases, this can lead to the compaction process never catching up with the database modifications, and eventually using up all the disk space. Schedule compaction during off-peak hours to prevent this!

- **… with adequate disk space:** Because compaction occurs by creating new files and updating the information, you may need as much as twice the disk space of your current database and index files for compaction to take place.

However, it is important to keep in mind that the exact amount of the disk space required depends on the level of fragmentation, the amount of dead data and the activity of the database, as changes during compaction will also need to be written to the updated data files.

Before compaction takes place, the disk space is checked. If the amount of available disk space is less than twice the current database size, the compaction process does not take place and a warning is issued in the log.

Compaction Behavior

- **Stop/Restart:** The compaction process can be stopped and restarted. However, you should be aware that the if the compaction process is stopped, further updates to database are completed, and then the compaction process is restarted, the updated database may not be a clean compacted version. This is because any changes to the portion of the database file that were processed before the compaction was canceled and restarted have already been processed.
- **Auto-compaction:** Auto-compaction automatically triggers the compaction process on your database. You can schedule specific hours when compaction can take place.
- **Compaction activity log:** Compaction activity is reported in the Couchbase Server log. You will see entries similar to following showing the compaction operation and duration:
- **Compaction activity log:** Compaction activity is reported in the Couchbase Server log. You can see the following items for compaction:
  - **Autocompaction** Indicates compaction cannot be performed because of inadequate disk space
  - **Manually triggered compaction**
  - Compaction completed successfully
  - Compaction failed
  - **Purge deletes compaction**
  - Compaction started/completed
  - Compaction failed

Auto-compaction configuration

Couchbase Server incorporates an automated compaction mechanism that can compact both data files and the view index files, based on triggers that measure the current fragmentation level within the database and view index data files.

Note

Spatial indexes are not automatically compacted. Spatial indexes must be compacted manually.

Auto-compaction can be configured in two ways:

- *Default Auto-Compaction* affects all the Couchbase Buckets within your Couchbase Server. If you set the default Auto-Compaction settings for your Couchbase server then auto-compaction is enabled for all Couchbase Buckets automatically.
- *Bucket Auto-Compaction* can be set on individual Couchbase Buckets. The bucket-level compaction always overrides any default auto-compaction settings, including if you have not configured any default auto-compaction

settings. You can choose to explicitly override the Couchbase Bucket specific settings when editing or creating a new Couchbase Bucket.

The available settings for both default Auto-Compaction and Couchbase Bucket specific settings are identical:

- **Database Fragmentation**

The primary setting is the percentage level within the database at which compaction occurs. The figure is expressed as a percentage of fragmentation for each item, and you can set the fragmentation level at which the compaction process will be triggered.

For example, if you set the fragmentation percentage at 10%, the moment the fragmentation level has been identified, the compaction process will be started, unless you have time limited auto-compaction. SeeTime Period.

- **View Fragmentation**

The View Fragmentation specifies the percentage of fragmentation within all the view index files at which compaction will be triggered, expressed as a percentage.

- **Time Period**

To prevent auto compaction taking place when your database is in heavy use, you can configure a time during which compaction is allowed. This is expressed as the hour and minute combination between which compaction occurs. For example, you could configure compaction to take place between 01:00 and 06:00.

If compaction is identified as required outside of these hours, compaction will be delayed until the specified time period is reached. The time period is applied every day while the Couchbase Server is active. The time period cannot be configured on a day-by-day basis.

- **Compaction abortion**

The compaction process can be configured so that if the time period during which compaction is allowed ends while the compaction process is still completing, the entire compaction process will be terminated. This option affects the compaction process:

If this option is enabled, and compaction is running, the process will be stopped. The files generated during the compaction process will be kept, and compaction will be restarted when the next time period is reached.

This can be a useful setting if want to ensure the performance of your Couchbase Server during a specified time period, as this will ensure that compaction is never running outside of the specified time period.

If this option is disabled and the compaction is running when the time period ends, compaction will continue suntil the process has been completed.

Using this option can be useful if you want to ensure that the compaction process completes.

- **Parallel Compaction**

By default, compaction operates sequentially, executing first on the database and then the Views if both are configured for auto-compaction. If you enable parallel compaction, both the databases and the views can be compacted at the same time. This requires more CPU and database activity for both to be processed simultaneously, but if you have CPU cores and disk I/O (for example, if the database and view index information is stored on different physical disk devices), the two can complete in a shorter time.

- **Metadata Purge Interval**

You can remove tombstones for expired and deleted items as part of the auto-compaction process. Tombstones are records containing the key and metadata for deleted and expired items and are used for eventually consistency between clusters and for views.

Configuration of auto-compaction is through Couchbase Web Console. Information on per-bucket settings is through the Couchbase Bucket create/edit screen. . You can also view and change these settings using the REST API,

Auto-compaction strategies

The exact fragmentation and scheduling settings for auto-compaction should be chosen carefully to ensure that your database performance and compaction performance meet your requirements.

You want to consider the following:

- You should monitor the compaction process to determine how long it takes to compact your database. This will help you identify and schedule a suitable time-period for auto-compaction to occur.
- Compaction affects the disk space usage of your database, but should not affect performance. Frequent compaction runs on a small database file are unlikely to cause problems, but frequent compaction on a large database file may impact the performance and disk usage.
- Compaction can be terminated at any time. This means that if you schedule compaction for a specific time period, but then require the additional resources being used for compaction you can terminate the compaction and restart during another off-peak period.
- Because compaction can be stopped and restarted it is possible to indirectly trigger an incremental compaction. For example, if you configure a one-hour compaction period, enable Compaction abortion, and compaction takes 4 hours to complete, compaction will incrementally take place over four days.
- When you have a large number of Couchbase buckets on which you want to use auto-compaction, you may want to schedule your auto-compaction time period for each bucket in a staggered fashion so that compaction on each bucket can take place within a it's own unique time period.

Related Links

Cluster management on page 135

Related topics

Settings on page 294
The **Settings** section provides configuration and information for the cluster, update notifications, auto failover, alerts, auto compaction, sample buckets, and account management.

Log on page 293
The Log section provides a built-in event log for Couchbase Server.

Data Buckets on page 259

Compaction REST API on page 494

Failing over nodes

Failing over a node means that Couchbase Server removes the node from a cluster and makes replicated data at other nodes available for client requests.

Because Couchbase Server provides data replication within a cluster, the cluster can handle failure of one or more nodes without affecting your ability to access the stored data. In the event of a node failure, you can manually initiate a `failover` status for the node in Web Console and resolve the issues.

Alternately you can configure Couchbase Server so it will *automatically* remove a failed node from a cluster and have the cluster operate in a degraded mode. If you choose this automatic option, the workload for functioning nodes that remain the cluster will increase. You will still need to address the node failure, return a functioning node to the cluster and then rebalance the cluster in order for the cluster to function as it did prior to node failure.

Whether you manually failover a node or have Couchbase Server perform automatic failover, you should determine the underlying cause for the failure. You should then set up functioning nodes, add the nodes, and then rebalance the cluster. Keep in mind the following guidelines on replacing or adding nodes when you cope with node failure and failover scenarios:

- If the node failed due to a hardware or system failure, you should add a new replacement node to the cluster and rebalance.
- If the node failed because of capacity problems in your cluster, you should replace the node but also add additional nodes to meet the capacity needs.
- If the node failure was transient in nature and the failed node functions once again, you can add the node back to the cluster.

Be aware that failover is a distinct operation compared to *removing/rebalancing* a node. Typically you remove a *functioning node* from a cluster for maintenance, or other reasons; in contrast you perform a failover for a node that does not function.

When you remove a functioning node from a cluster, you use Web Console to indicate the node will be removed, then you rebalance the cluster so that data requests for the node can be handled by other nodes. Since the node you want to remove still functions, it is able to handle data requests until the rebalance completes. At this point, other nodes in the cluster will handle data requests. There is therefore no disruption in data service or no loss of data that can occur when you remove a node then rebalance the cluster. If you need to remove a functioning node for administration purposes, you should use the remove and rebalance functionality not failover.

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

For more information about performing failover see the following resources:

- **Automated failover** will automatically mark a node as failed over if the node has been identified as unresponsive or unavailable. There are some deliberate limitations to the automated failover feature.
- **Initiating a failover** whether or not you use automatic or manual failover, you need to perform additional steps to bring a cluster into a fully functioning state.
- **Adding nodes after failover**. After you resolve the issue with the failed over node you can add the node back to your cluster.

Choosing a failover solution

Because node failover has the potential to reduce the performance of your cluster, you should consider how best to handle a failover situation. Using automated failover means that a cluster can fail over a node without user-intervention and without knowledge and identification of the issue that caused the node failure. It still requires you to initiate a rebalance in order to return the cluster to a healthy state.

If you choose manual failover to manage your cluster you need to monitor the cluster and identify when an issue occurs. If an issues does occur you then trigger a manual failover and rebalance operation. This approach requires more monitoring and manual intervention, there is also still a possibility that your cluster and data access may still degrade before you initiate failover and rebalance.

In the following sections the two alternatives and their issues are described in more detail.

Automated failover considerations

Automatically failing components in any distributed system can cause problems. If you cannot identify the cause of failure, and you do not understand the load that will be placed on the remaining system, then automated failover can cause more problems than it is designed to solve. Some of the situations that might lead to problems include:

- **Avoiding failover chain-reactions (Thundering herd)**

Imagine a scenario where a Couchbase Server cluster of five nodes is operating at 80–90% aggregate capacity in terms of network load. Everything is running well but at the limit of cluster capacity. Imagine a node fails and the software decides to automatically failover that node. It is unlikely that all of the remaining four nodes are be able to successfully handle the additional load.

The result is that the increased load could lead to another node failing and being automatically failed over. These failures can cascade and lead to the eventual loss of an entire cluster. Clearly having 1/5th of the requests not being serviced due to single node failure would be more desirable than none of the requests being serviced due to an entire cluster failure.

The solution in this case is to continue cluster operations with the single node failure, add a new server to the cluster to handle the missing capacity, mark the failed node for removal and then rebalance. This way there is a brief partial outage rather than an entire cluster being disabled.

One alternate preventative solution is to ensure there is excess capacity to handle unexpected node failures and allow replicas to take over.

- **Handling failovers with network partitions**

In case of network partition or split-brain where the failure of a network device causes a network to be split, Couchbase implements automatic failover with the following restrictions:

- Automatic failover requires a minimum of three (3) nodes per cluster. This prevents a 2-node cluster from having both nodes fail each other over in the face of a network partition and protects the data integrity and consistency.
- Automatic failover occurs only if exactly one (1) node is down. This prevents a network partition from causing two or more halves of a cluster from failing each other over and protects the data integrity and consistency.
- Automatic failover occurs only once before requiring administrative action. This prevents cascading failovers and subsequent performance and stability degradation. In many cases, it is better to not have access to a small part of the dataset rather than having a cluster continuously degrade itself to the point of being non-functional.
- Automatic failover implements a 30 second delay when a node fails before it performs an automatic failover. This prevents transient network issues or slowness from causing a node to be failed over when it shouldn't be.

If a network partition occurs, automatic failover occurs if and only if automatic failover is allowed by the specified restrictions. For example, if a single node is partitioned out of a cluster of five (5), it is automatically failed over. If more than one (1) node is partitioned off, autofailover does not occur. After that, administrative action is required for a reset. In the event that another node fails before the automatic failover is reset, no automatic failover occurs.

- **Handling misbehaving nodes**

There are cases where one node loses connectivity to the cluster or functions as if it has lost connectivity to the cluster. If you enable it to automatically failover the rest of the cluster, that node is able to create a cluster-of-one. The result for your cluster is a similar partition situation we described previously.

In this case you should make sure there is spare node capacity in your cluster and failover the node with network issues. If you determine there is not enough capacity, add a node to handle the capacity after your failover the node with issues.

Manual or monitored failover

Performing manual failover through monitoring can take two forms, either by human monitoring or by using a system external to the Couchbase Server cluster. An external monitoring system can monitor both the cluster and the node environment and make a more information-driven decision. If you choose a manual failover solution, there are also issues you should be aware of. Although automated failover has potential issues, choosing to use manual or monitored failover is not without potential problems.

- **Human intervention**

One option is to have a human operator respond to alerts and make a decision on what to do. Humans are uniquely capable of considering a wide range of data, observations and experiences to best resolve a situation. Many organizations disallow automated failover without human consideration of the implications. The drawback of using human intervention is that it will be slower to respond than using a computer-based monitoring system.

- **External monitoring**

Another option is to have a system monitoring the cluster via the Couchbase REST API. Such an external system is in a good position to failover nodes because it can take into account system components that are outside the scope of Couchbase Server.

For example monitoring software can observe that a network switch is failing and that there is a dependency on that switch by the Couchbase cluster. The system can determine that failing Couchbase Server nodes will not help the situation and will therefore not failover the node.

The monitoring system can also determine that components around Couchbase Server are functioning and that various nodes in the cluster are healthy. If the monitoring system determines the problem is only with a single node and remaining nodes in the cluster can support aggregate traffic, then the system may failover the node using the REST API or command-line tools.

Using automatic failover

There are a number of restrictions on automatic failover in Couchbase Server. This is to help prevent some issues that can occur when you use automatic failover.

- **Disabled by Default** Automatic failover is disabled by default. This prevents Couchbase Server from using automatic failover without you explicitly enabling it.
- **Minimum Nodes** Automatic failover is only available on clusters of at least three nodes.

If two or more nodes go down at the same time within a specified delay period, the automatic failover system will not failover any nodes.

- **Required Intervention** Automatic failover will only fail over one node before requiring human intervention. This is to prevent a chain reaction failure of all nodes in the cluster.
- **Failover Delay** There is a minimum 30 second delay before a node will be failed over. This time can be raised, but the software is hard coded to perform multiple pings of a node that may be down. This is to prevent failover of a functioning but slow node or to prevent network connection issues from triggering failover.

You can use the REST API to configure an email notification that will be sent by Couchbase Server if any node failures occur and node is automatically failed over.

Once an automatic failover has occurred, the Couchbase Cluster is relying on other nodes to serve replicated data. You should initiate a rebalance to return your cluster to a fully functioning state.

Resetting the Automatic failover counter

After a node has been automatically failed over, Couchbase Server increments an internal counter that indicates if a node has been failed over. This counter prevents the server from automatically failing over additional nodes until you identify the issue that caused the failover and resolve it. If the internal counter indicates a node has failed over, the server will no longer automatically failover additional nodes in the cluster. You will need to re-enable automatic failover in a cluster by resetting this counter.

Important

Reset the automatic failover only after the node issue is resolved, rebalance occurs, and the cluster is restored to a fully functioning state.

You can reset the counter using the REST API:

```
> curl -i -u cluster-username:cluster-password \
    http://localhost:8091/settings/autoFailover/resetCount
```

Initiating a node failover

If you need to remove a node from the cluster due to hardware or system failure, you need to indicate the failover status for that node. This causes Couchbase Server to use replicated data from other functioning nodes in the cluster.

Important

Before you indicate the failover for a node, read [Failing Over Nodes](#couchbase-admin-tasks-failover). Do not use failover to remove a functioning node from the cluster for administration or upgrade. This is because initiating a failover for a node activates replicated data at other nodes which reduces the overall capacity of the cluster. Data from the failover node that has not yet been replicated at other nodes or persisted on disk will be lost. For information about removing and adding a node, see [Performing a Rebalance, Adding a Node to a Cluster](../cb-admin/#couchbase-admin-tasks-addremove-rebalance).

You can provide the failover status for a node with two different methods:

- **Using the Web Console**

Go to the `Management -> Server Nodes` section of the Web Console. Find the node that you want to failover, and click the `Fail Over` button. You can only failover nodes that the cluster has identified as being Down.

Web Console will display a warning message.

Click `Fail Over` to indicate the node is failed over. You can also choose to `Cancel`.

- **Using the Command-line**

You can failover one or more nodes using the `failover` command in `couchbase-cli`. To failover the node, you must specify the IP address and port, if not the standard port for the node you want to failover. For example:

````
```
> couchbase-cli failover --cluster=localhost:8091\
    -u cluster-username -p cluster-password\
    --server-failover=192.168.0.72:8091
```
````

If successful this indicates the node is failed over.

After you specify that a node is failed over you should handle the cause of failure and get your cluster back to a fully functional state.

Handling a failover situation

Any time that you automatically or manually failover a node, the cluster capacity will be reduced. Once a node is failed over:

- The number of available nodes for each data bucket in your cluster will be reduced by one.
- Replicated data handled by the failover node will be enabled on other nodes in the cluster.
- Remaining nodes will have to handle all incoming requests for data.

After a node has been failed over, you should perform a rebalance operation. The rebalance operation will:

- Redistribute stored data across the remaining nodes within the cluster.
- Recreate replicated data for all buckets at remaining nodes.
- Return your cluster to the configured operational state.

You may decide to add one or more new nodes to the cluster after a failover to return the cluster to a fully functional state. Better yet you may choose to replace the failed node and add additional nodes to provide more capacity than before.

Adding back a failed over node

You can add a failed over node back to the cluster if you identify and fix the issue that caused node failure. After Couchbase Server marks a node as failed over, the data on disk at the node will remain. A failed over node will not longer be *synchronized* with the rest of the cluster; this means the node will no longer handle data request or receive replicated data.

When you add a failed over node back into a cluster, the cluster will treat it as if it is a new node. This means that you should rebalance after you add the node to the cluster. This also means that any data stored on disk at that node will be destroyed when you perform this rebalance.

Copy or Delete Data Files before Rejoining Cluster

Therefore, before you add a failed over node back to the cluster, it is best practice to move or delete the persisted data files before you add the node back into the cluster. If you want to keep the files you can copy or move the files to another location such as another disk or EBS volume. When you add a node back into the cluster and then rebalance, data files will be deleted, recreated and repopulated.

Related Links

Cluster management on page 135

Related topics

Rebalancing on page 158

Settings on page 294

The **Settings** section provides configuration and information for the cluster, update notifications, auto failover, alerts, auto compaction, sample buckets, and account management.

Managing auto-failover on page 437

View settings for email notifications on page 440

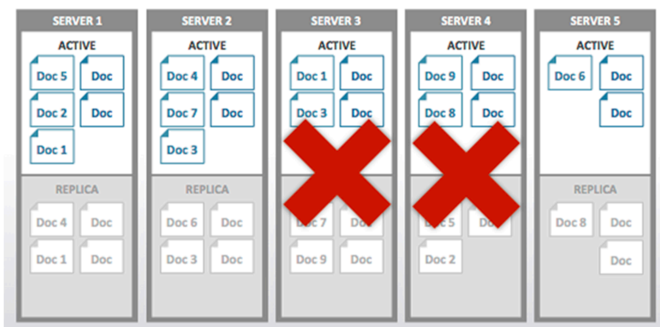
Data recovery from remote clusters

Data recovery from remote clusters requires an XDCR environment and adequate amount of memory and disk space to support the workload and recovered data.

If more nodes fail in a cluster than the number of replicas, data partitions in that cluster will no longer be available. For instance, if you have a four node cluster with one replica per node and two nodes fail, some data partitions will no longer be available. There are two solutions for this scenario:

- Recover data from disk. If you plan on recovering from disk, you may not be able to do so if the disk completely fails.
- Recover partitions from a remote cluster. You can use this second option when you have XDCR set up to replicate data to the second cluster. The requirement for using `cbrecovery` is that you need to set up a second cluster that will contain backup data.

The following shows a scenario where replica vBuckets are lost from a cluster due to multi-node failure:



Before you perform a recovery, make sure that your main cluster has an adequate amount of memory and disk space to support the workload as well as the data you recover. This means that even though you can recover data to a cluster with failed nodes, you should investigate what caused the node failures and also make sure your cluster has adequate capacity before you recover data. If you do add nodes be certain to rebalance only after you have

When you use `cbrecovery` it compares the data partitions from a main cluster with a backup cluster, then sends missing data partitions detected. If it fails, once you successfully restart `cbrecovery`, it will do a delta between clusters again and determine any missing partitions since the failure then resume restoring these partitions.

Failure Scenarios

Imagine the following happens when you have a four node cluster with one replica. Each node has 256 active and 256 replica vBuckets which total 1024 active and 1024 replica vBuckets:

1. When one node fails, some active and some replica vBuckets are no longer available in the cluster.
2. After you fail over this node, the corresponding replica vBuckets on other nodes will be put into an active state. At this point you have a full set of active vBuckets and a partial set of replica vBuckets in the cluster.
3. A second node fails. More active vBuckets will not be accessible.
4. You fail over the second node. At this point any missing active vBuckets that do not have corresponding replica vBuckets will be lost.

In this type of scenario you can use `cbrecovery` to get the missing vBuckets from your backup cluster. If you have multi-node failure on both your main and backup clusters you will experience data loss.

Recovery Scenarios for cbrecovery

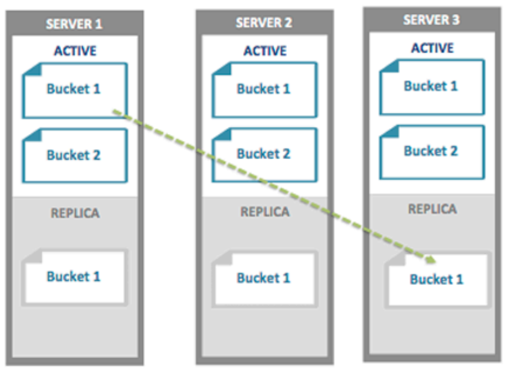
The following describes some different cluster setups so that you can better understand whether or not this approach will work in your failure scenario:

- Multiple Node Failure in Cluster. If multiple nodes fail in a cluster then some vBuckets may be unavailable. In this case if you have already setup XDCR with another cluster, you can recover those unavailable vBuckets from the other cluster.

- Bucket with Inadequate Replicas.

Single Bucket. In this case where we have only one bucket with zero replicas on all the nodes in a cluster. In this case when a node goes down in the cluster some of the partitions for that node will be unavailable. If we have XDCR set up for this cluster we can recover the missing partitions with `cbrecovery`.

Multi-Bucket. In this case, nodes in a cluster have multiple buckets and some buckets might have replicas and some do not. In the image below we have a cluster and all nodes have two buckets, Bucket1 and Bucket2. Bucket 1 has replicas but Bucket2 does not. In this case if one of the nodes goes down, since Bucket 1 has replicas, when we failover the node the replicas on other nodes will be activated. But for the bucket with no replicas some partitions will be unavailable and will require `cbrecovery` to recover data. In this same example if multiple nodes fail in the cluster, we need to perform vBucket recovery both buckets since both will have missing partitions.

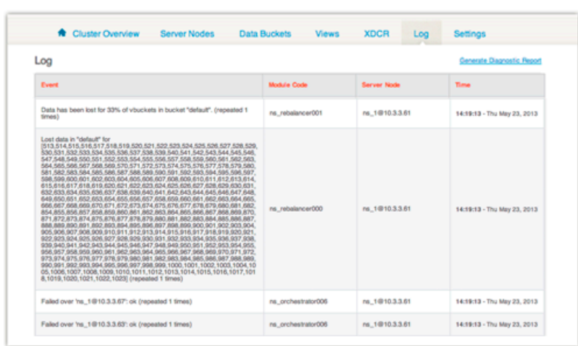


Handling the Recovery

Should you encounter node failure and have unavailable vBuckets, you should follow this process:

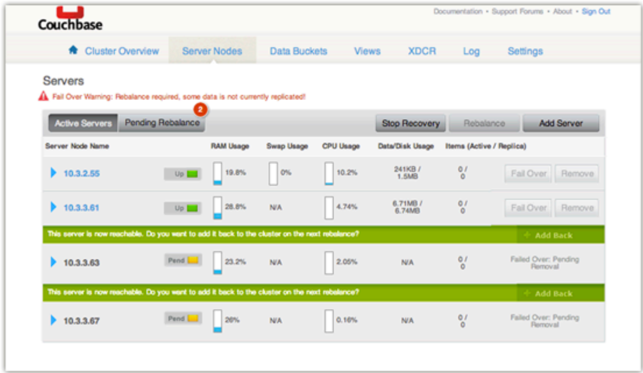
1. For each failed node, Click Fail Over under the Server Nodes tab in Web Console.

   After you click Fail Over, under Web Console | Log tab you will see whether data is unavailable and which vBuckets are unavailable. If you do not have enough replicas for the number of failed over nodes, some vBuckets will no longer be available:



2. Add new functioning nodes to replace the failed nodes.

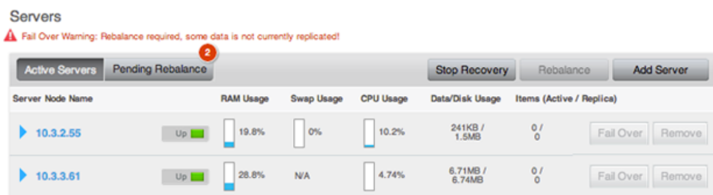
   Do not rebalance after you add new nodes to the cluster. Typically you do this after adding nodes to a cluster, but in this scenario the rebalance will destroy information about the missing vBuckets and you cannot recover them.



In this example we have two nodes that failed in a three-node cluster and we add a new node 10.3.3.61.

If you are certain your cluster can easily handle the workload and recovered data, you may choose to skip this step.

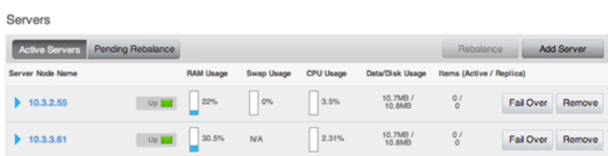
3. Run `cbrecovery` to recover data from your backup cluster. In the Server Panel, a Stop Recovery button appears.



After the recovery completes, this button disappears.

4. Rebalance your cluster.

   Once the recovery is done, you can rebalance your cluster, which will recreate replica vBuckets and evenly redistribute them across the cluster.



Recovery 'Dry-Run'

Before you recover vBuckets, you may want to preview a list of buckets no longer available in the cluster. Use this command and options:

```
   shell> ./cbrecovery http://Administrator:password@10.3.3.72:8091 http://
Administrator:password@10.3.3.61:8091 -n
```

Here we provide administrative credentials for the node in the cluster as well as the option `-n`. This will return a list of vBuckets in the remote secondary cluster which are no longer in the your first cluster. If there are any unavailable buckets in the cluster with failed nodes, you see output as follows:

```
   2013-04-29 18:16:54,384: MainThread Missing vbuckets to be recovered:
[{"node": "ns_1@10.3.3.61",
"vbuckets": [513, 514, 515, 516, 517, 518, 519, 520, 521, 522, 523, 524, 525,
 526,, 528, 529,
530, 531, 532, 533, 534, 535, 536, 537, 538, 539, 540, 541, 542, 543, 544,
 545,, 547, 548,
549, 550, 551, 552, 553, 554, 555, 556, 557, 558, 559, 560, 561, 562, 563,
 564, 565, 566, 567,
568, 569, 570, 571, 572,....
```

Where the `vbuckets` array contains all the vBuckets that are no longer available in the cluster. These are the bucket you can recover from the remotes cluster. To recover the vBuckets:

```
shell> ./cbrecovery http://Administrator:password@<From_IP>:8091 \
 http://Administrator:password@<To_IP>:8091 -B bucket_name
```

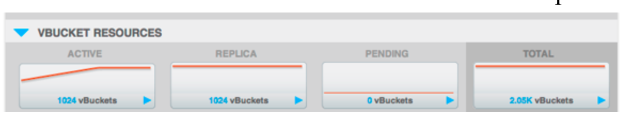
You can run the command on either the cluster with unavailable vBuckets or on the remote cluster, as long as you provide the hostname, port, and credentials for remote cluster and the cluster with missing vBuckets in that order. If you do not provide the parameter `-B` the tool assumes you will recover unavailable vBuckets for the default bucket.

Monitoring the Recovery Process

You can monitor the progress of recovery under the Data Buckets tab of Couchbase Web Console:

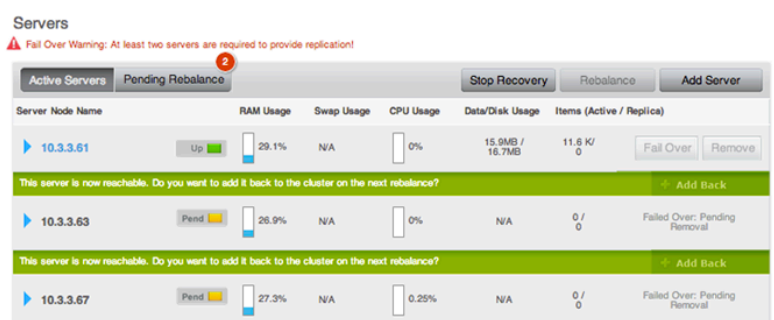
1. Click on the Data Buckets tab.
2. Select the data bucket you are recovering in the Data Buckets drop-down.
3. Click on the Summary drop-down to see more details about this data bucket. You see an increased number in the `items` level during recovery:

4. You can also see the number of active vBuckets increase as they are recovered until you reach 1024 vBuckets. Click on the vBucket Resources drop-down:



As this tool runs from the command line you can stop it at any time as you would any other command-line tool.

5. A `Stop Recovery` button appears in the Servers panels. If you click this button, you will stop the recovery process between clusters. Once the recovery process completes, this button will no longer appear and you will need to rebalance the cluster. If you are in Couchbase Web Console, you can also stop it in this panel:



6. After recovery completes, click on the Server Nodes tab then Rebalance to rebalance your cluster.

When `cbrecovery` finishes it will output a report in the console:

```
 Recovery :                    Total |     Per sec
 batch    :                     0000 |        14.5
 byte     :                     0000 |       156.0
 msg      :                     0000 |        15.6
4 vbuckets recovered with elapsed time 10.90 seconds
```

In this report `batch` is a group of internal operations performed by `cbrecovery`, `byte` indicates the total number of bytes recovered and `msg` is the number of documents recovered.

Related Links

Cluster management on page 135

Related topics

Failing over nodes on page 141

Failing over a node means that Couchbase Server removes the node from a cluster and makes replicated data at other nodes available for client requests.

Cross Datacenter Replication (XDCR) on page 241

Managing XDCR on page 169

Cross datacenter replication (XDCR) provides an easy method of replicating data from one cluster to another for disaster recovery as well as better data locality (getting data closer to its users).

Rebalancing on page 158

Backup and restore

The predominant method for backing up and restoring data is using the CLI `cbbackup` and `cbrestore` tools.

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 a number of methods for performing a backup:

- Using `cbbackup`

The `cbbackup` 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.

To restore, you need to use the `cbrestore`command.

- Using 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.

To restore, you need to use the file copy method.

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.

Best Practice

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.

Backing up using cbbackup

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 clusters do not need to match if you used `cbbackup` to store the information.

The `cbbackup` command will copy the data in each course from the source definition to a destination backup directory. The backup file format is unique to 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]
```

The `cbbackup` tool is located within the standard Couchbase command-line directory.

Be aware that `cbbackup` does not support external IP addresses. This means that if you install Couchbase Server with the default IP address, you cannot use an external hostname to access it.

Where the arguments are as described below:

- `[options]`

One or more options for the backup process. These are used to configure username and password information for connecting to the cluster, backup type selection, and bucket selection.

The primary options select what will be backed up by `cbbackup`, including:

```
* `--single-node`

  Only back up the single node identified by the source specification.

* `--bucket-source` or `-b`

  Backup only the specified bucket name.
```

- `[source]`

The source for the data, either a local data directory reference, or a remote node/cluster specification:

````
* `Local Directory Reference`

  A local directory specification is defined as a URL using the `couchstore-
files`
  protocol. For example:

    ```
    couchstore-files:///opt/couchbase/var/lib/couchbase/data/default
    ```

  Using this method you are specifically backing up the specified bucket data
 on a
  single node only. To backup an entire bucket data across a cluster, or all
 the
  data on a single node, you must use the cluster node specification. This
 method
  does not backup the design documents defined within the bucket.

* `cluster node`

  A node or node within a cluster, specified as a URL to the node or cluster
  service. For example:

    ```
    http://HOST:8091
    ```

  Or for distinction you can use the `couchbase` protocol prefix:

    ```
    couchbase://HOST:8091
````
````

```
  ```

 The administrator and password can also be combined with both forms of the
URL
 for authentication. If you have named data buckets other than the default
bucket
 which you want to backup, you will need to specify an administrative name
and
 password for the bucket:

  ```
  couchbase://Administrator:password@HOST:8091
  ```

 The combination of additional options specifies whether the supplied URL
refers
 to the entire cluster, a single node, or a single bucket (node or cluster).
The
 node and cluster can be remote (or local).

 This method also backs up the design documents used to define views and
indexes.
```

- `[backup_dir]`

The directory where the backup data files will be stored on the node on which the `cbbackup` is executed. This must be an absolute, explicit, directory, as the files will be stored directly within the specified directory; no additional directory structure is created to differentiate between the different components of the data backup.

The directory that you specify for the backup should either not exist, or exist and be empty with no other files. If the directory does not exist, it will be created, but only if the parent directory already exists.

The backup directory is always created on the local node, even if you are backing up a remote node or cluster. The backup files are stored locally in the backup directory specified.

Backups can take place on a live, running, cluster or node for the IP

Using this basic structure, you can backup a number of different combinations of data from your source cluster. Examples of the different combinations are provided below:

- **Backup all nodes and all buckets**

To backup an entire cluster, consisting of all the buckets and all the node data:

```
> cbbackup http://HOST:8091 /backups/backup-20120501 \
  -u Administrator -p password
  [####################] 100.0% (231726/231718 msgs)
bucket: default, msgs transferred...
       :                total |       last |    per sec
 batch :                 5298 |       5298 |      617.1
 byte  :             10247683 |   10247683 |  1193705.5
 msg   :               231726 |     231726 |    26992.7
done
  [####################] 100.0% (11458/11458 msgs)
bucket: loggin, msgs transferred...
       :                total |       last |    per sec
 batch :                 5943 |       5943 |    15731.0
 byte  :             11474121 |   11474121 | 30371673.5
 msg   :84 |84 |   643701.2
done
```

When backing up multiple buckets, a progress report, and summary report for the information transferred will be listed for each bucket backed up. The `msgs` count shows the number of documents backed up. The `byte` shows the overall size of the data document data.
```

The source specification in this case is the URL of one of the nodes in the cluster. The backup process will stream data directly from each node in order to create the backup content. The initial node is only used to obtain the cluster topology so that the data can be backed up.

A backup created in this way enables you to choose during restoration how you want to restore the information. You can choose to restore the entire dataset, or a single bucket, or a filtered selection of that information onto a cluster of any size or configuration.

- **Backup all nodes, single bucket**

To backup all the data for a single bucket, containing all of the information from the entire cluster:

```
> cbbackup http://HOST:8091 /backups/backup-20120501 \
  -u Administrator -p password \
  -b default
  [####################] 100.0% (231726/231718 msgs)
bucket: default, msgs transferred...
       :                total |       last |    per sec
 batch :                 5294 |       5294 |      617.0
 byte  :             10247683 |   10247683 |  1194346.7
 msg   :               231726 |     231726 |    27007.2
done
```

The `-b` option specifies the name of the bucket that you want to backup. If the bucket is a named bucket you will need to provide administrative name and password for that bucket.

To backup an entire cluster, you will need to run the same operation on each bucket within the cluster.

- **Backup single node, all buckets**

To backup all of the data stored on a single node across all of the different buckets:

```
> cbbackup http://HOST:8091 /backups/backup-20120501 \
  -u Administrator -p password \
  --single-node
```

Using this method, the source specification must specify the node that you want backup. To backup an entire cluster using this method, you should backup each node individually.

- **Backup single node, single bucket**

To backup the data from a single bucket on a single node:

```
> cbbackup http://HOST:8091 /backups/backup-20120501 \
  -u Administrator -p password \
  --single-node \
  -b default
```

Using this method, the source specification must be the node that you want to back up.

- **Backup single node, single bucket; backup files stored on same node**

To backup a single node and bucket, with the files stored on the same node as the source data, there are two methods available. One uses a node specification, the other uses a file store specification. Using the node specification:

```
> ssh USER@HOST
remote-> sudo su - couchbase
remote-> cbbackup http://127.0.0.1:8091 /mnt/backup-20120501 \
  -u Administrator -p password \
  --single-node \
  -b default
```

This method backups up the cluster data of a single bucket on the local node, storing the backup data in the local filesystem.

Using a file store reference (in place of a node reference) is faster because the data files can be copied directly from the source directory to the backup directory:

```
> ssh USER@HOST
remote-> sudo su - couchbase
remote-> cbbackup couchstore-files:///opt/couchbase/var/lib/couchbase/data/
default /mnt/backup-20120501
```

To backup the entire cluster using this method, you will need to backup each node, and each bucket, individually.

Choosing the right backup solution will depend on your requirements and your expected method for restoring the data to the cluster.

Filtering keys during backup

The `cbbackup` command includes support for filtering the keys that are backed up into the database files you create. 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 specification is in the form of a regular expression, and is performed on the client-side within the `cbbackup` tool. For example, to backup information from a bucket where the keys have a prefix of 'object':

```
> cbbackup http://HOST:8091 /backups/backup-20120501 \
  -u Administrator -p password \
  -b default \
  -k '^object.*'
```

The above will copy only the keys matching the specified prefix into the backup file. When the data is restored, only those keys that were recorded in the backup file will be restored.

Warning

The regular expression match is performed client side. This means that the entire bucket contents must be accessed by the `cbbackup` command and then discarded if the regular expression does not match.

Key-based regular expressions can also be used when restoring data. You can backup an entire bucket and restore selected keys during the restore process using `cbrestore`.

Backing up using file copies

You can also backup by using either `cbbackup` and specifying the local directory where the data is stored, or by copying the data files directly using `cp`, `tar` or similar.

For example, using `cbbackup` :

```
> cbbackup \
    couchstore-files:///opt/couchbase/var/lib/couchbase/data/default \
    /mnt/backup-20120501
```

The same backup operation using `cp` :

```
> cp -R /opt/couchbase/var/lib/couchbase/data/default \
      /mnt/copy-20120501
```

The limitation of backing up information in this way is that the data can only be restored to offline nodes in an identical cluster configuration, and where an identical vbucket map is in operation (you should also copy the `config.dat` configuration file from each node.

Restoring using cbrestore

When restoring a backup, you have to select the appropriate restore sequence based on the type of restore you are performing. The methods available to you when restoring a cluster are dependent on the method you used when backing up the cluster. If `cbbackup` was used to backup the bucket data, you can restore back to a cluster with the same or different configuration. This is because `cbbackup` stores information about the stored bucket data in a format that enables it to be restored back into a bucket on a new cluster. For all these scenarios you can use `cbrestore`.

If the information was backed up using a direct file copy, then you must restore the information back to an identical cluster.

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]
```

Where:

- `[options]`

Options specifying how the information should be restored into the cluster. Common options include:

```
* `--bucket-source`

  Specify the name of the bucket data to be read from the backup data that
 will be
  restored.

* `--bucket-destination`

  Specify the name of the bucket the data will be written to. If this option
 is
  not specified, the data will be written to a bucket with the same name as
 the
  source bucket.

* `--add`

  Use `--add` instead of `--set` in order to not overwrite existing items in
 the destination.
```

For information on all the options available when using the cbrestore tool.

- `[source]`

The backup directory specified to `cbbackup` where the backup data was stored.

- `[destination]`

The REST API URL of a node within the cluster where the information will be restored.

The `cbrestore` command restores only a single bucket of data at a time. If you have created a backup of an entire cluster (i.e. all buckets), then you must restore each bucket individually back to the cluster. All destination buckets must already exist; `cbrestore` does not create or configure destination buckets for you.

For example, to restore a single bucket of data to a cluster:

```
> cbrestore \
    /backups/backup-2012-05-10 \
```

```
    http://Administrator:password@HOST:8091 \
    --bucket-source=XXX
  [####################] 100.0% (231726/231726 msgs)
bucket: default, msgs transferred...
       :                total |       last |    per sec
 batch :                  232 |        232 |       33.1
 byte  :             10247683 |   10247683 |  1462020.7
 msg   :               231726 |     231726 |    33060.0
done
```

To restore the bucket data to a different bucket on the cluster:

```
> cbrestore \
    /backups/backup-2012-05-10 \
    http://Administrator:password@HOST:8091 \
    --bucket-source=XXX \
    --bucket-destination=YYY
  [####################] 100.0% (231726/231726 msgs)
bucket: default, msgs transferred...
       :                total |       last |    per sec
 batch :                  232 |        232 |       33.1
 byte  :             10247683 |   10247683 |  1462020.7
 msg   :               231726 |     231726 |    33060.0
done
```

The `msg` count in this case is the number of documents restored back to the bucket in the cluster.

Filtering keys during restore

The `cbrestore` command includes support for filtering the keys that are restored to the database from the files that were created during backup. This is in addition to the filtering support available during backup.

The specification is in the form of a regular expression supplied as an option to the `cbrestore` command. For example, to restore information to a bucket only where the keys have a prefix of ‘object’:

```
> cbrestore /backups/backup-20120501 http://HOST:8091 \
  -u Administrator -p password \
  -b default \
  -k '^object.*'
2013-02-18 10:39:09,476: w0 skipping msg with key: sales_7597_3783_6
...
2013-02-18 10:39:09,476: w0 skipping msg with key: sales_5575_3699_6
2013-02-18 10:39:09,476: w0 skipping msg with key: sales_7597_3840_6
  [                    ] 0.0% (0/231726 msgs)
bucket: default, msgs transferred...
       :                total |       last |    per sec
 batch :                    1 |          1 |        0.1
 byte  :                    0 |          0 |        0.0
 msg   :                    0 |          0 |        0.0
done
```

The above will copy only the keys matching the specified prefix into the `default` bucket. For each key skipped, an information message will be supplied. The remaining output shows the records transferred and summary as normal.

Restoring using file copies

To restore the information to the same cluster, with the same configuration, you must shutdown your entire cluster while you restore the data, and then restart the cluster again. You are replacing the entire cluster data and configuration with the backed up version of the data files, and then re-starting the cluster with the saved version of the cluster files.

Important

Make sure that any restoration of files also sets the proper ownership of those files to the couchbase user

When restoring data back in to the same cluster, then the following must be true before proceeding:

- The backup and restore must take between cluster using the same version of Couchbase Server.
- The cluster must contain the same number of nodes.
- Each node must have the IP address or hostname it was configured with when the cluster was backed up.
- You must restore all of the `config.dat` configuration files as well as all of the database files to their original locations.

The steps required to complete the restore process are:

1. Stop the Couchbase Server service on all nodes.
2. On each node, restore the database, `stats.json`, and configuration file (`config.dat`) from your backup copies for each node.
3. Restart the service on each node.

Backup and restore between Mac OS X and other platforms

Couchbase Server on Mac OS X uses a different number of configured vBuckets than the Linux and Windows installations. Because of this, backing up from Mac OS X and restoring to Linux or Windows, or vice versa, requires using the built-in Moxi server and the memcached protocol. Moxi will rehash the stored items into the appropriate bucket.

- **Backing up Mac OS X and restoring on Linux/Windows**

To backup the data from Mac OS X, you can use the standard `cbbackup` tool and options:

```
```
> cbbackup http://Administrator:password@mac:8091 /macbackup/today
```
```

To restore the data to a Linux/Windows cluster, you must connect to the Moxi port (11211) on one of the nodes within your destination cluster and use the Memcached protocol to restore the data. Moxi will rehash the information and distribute the data to the appropriate node within the cluster. For example:

```
```
> cbrestore /macbackup/today memcached://linux:11211 -b default -B default
```
```

If you have backed up multiple buckets from your Mac, you must restore to each bucket individually.

- **Backing Up Linux/Windows and restoring on Mac OS X**

To backup the data from Linux or Windows, you can use the standard `cbbackup` tool and options:

```
```
> cbbackup http://Administrator:password@linux:8091 /linuxbackup/today
```
```

To restore to the Mac OS X node or cluster, you must connect to the Moxi port (11211) and use the Memcached protocol to restore the data. Moxi will rehash the information and distribute the data to the appropriate node within the cluster. For example:

```
```
> cbrestore /linuxbackup/today memcached://mac:11211 -b default -B default
```
```

- **Transferring data directly**

You can use `cbtransfer` to perform the data move directly between Mac OS X and Linux/Windows clusters without creating the backup file, providing you correctly specify the use of the Moxi and Memcached protocol in the destination:

```
```
> cbtransfer http://linux:8091 memcached://mac:11211 -b default -B default
> cbtransfer http://mac:8091 memcached://linux:11211 -b default -B default
```
```

Important

These transfers will not transfer design documents, since they are using the Memcached protocol

- **Transferring design documents**

Because you are restoring data using the Memcached protocol, design documents are not restored. A possible workaround is to modify your backup directory. Using this method, you first delete the document data from the backup directory, and then use the standard restore process. This will restore only the design documents. For example:

```
```
> cbbackup http://Administrator:password@linux:8091 /linuxbackup/today
```
```

Remove or move the data files from the backup out of the way:

```
```
> mv /linuxbackup/today/bucket-default/* /tmp
```
```

Only the design document data will remain in the backup directory, you can now restore that information using `cbrestore` as normal:

```
```
> cbrestore /linuxbackup/today http://mac:8091 -b default -B default
```
```

Related Links

Cluster management on page 135

Related topics

Initial server setup on page 66

Command-line interface overview on page 353

cbbackup tool on page 357

The `cbbackup` tool creates a copy of data from an entire running cluster, an entire bucket, a single node, or a single bucket on a single functioning node. Your node or cluster needs to be functioning in order to create the backup. Couchbase Server will write a copy of data onto disk.

cbrestore tool on page 390

Rebalancing

As you store data into your Couchbase Server cluster, you may need to alter the number of nodes in your cluster to cope with changes in your application load, RAM, disk I/O and networking performance requirements.

Couchbase Server is designed to actively change the number of nodes configured within the cluster to cope with these requirements, all while the cluster is up and running and servicing application requests. The overall process is broken down into two stages; the addition and/or removal of nodes in the cluster, and the `rebalancing` of the information across the nodes.

The addition and removal process merely configures a new node into the cluster, or marks a node for removal from the cluster. No actual changes are made to the cluster or data when configuring new nodes or removing existing ones.

During the rebalance operation:

- Using the new Couchbase Server cluster structure, data is moved between the vBuckets on each node from the old structure. This process works by exchanging the data held in vBuckets on each node across the cluster. This has two effects:
  - Removes the data from machines being removed from the cluster. By totally removing the storage of data on these machines, it allows for each removed node to be taken out of the cluster without affecting the cluster operation.
  - Adds data and enables new nodes so that they can serve information to clients. By moving active data to the new nodes, they will be made responsible for the moved vBuckets and for servicing client requests.

- Rebalancing moves both the data stored in RAM, and the data stored on disk for each bucket, and for each node, within the cluster. The time taken for the move is dependent on the level of activity on the cluster and the amount of stored information.
- The cluster remains up, and continues to service and handle client requests. Updates and changes to the stored data during the migration process are tracked and will be updated and migrated with the data that existed when the rebalance was requested.
- The current vBucket map, used to identify which nodes in the cluster are responsible for handling client requests, is updated incrementally as each vBucket is moved. The updated vBucket map is communicated to Couchbase client libraries and enabled smart clients (such as Moxi), and allows clients to use the updated structure as the rebalance completes. This ensures that the new structure is used as soon as possible to help spread and even out the load during the rebalance operation.

Because the cluster stays up and active throughout the entire process, clients can continue to store and retrieve information and do not need to be aware that a rebalance operation is taking place.

There are four primary reasons that you perform a rebalance operation:

- Adding nodes to expand the size of the cluster.
- Removing nodes to reduce the size of the cluster.
- Reacting to a failover situation, where you need to bring the cluster back to a healthy state.
- You need to temporarily remove one or more nodes to perform a software, operating system or hardware upgrade.

Regardless of the reason for the rebalance, the purpose of the rebalance is migrate the cluster to a healthy state, where the configured nodes, buckets, and replicas match the current state of the cluster.

Related Links

Cluster management on page 135

Rebalance behind-the-scenes on page 160
The rebalance process is managed through a specific process called the orchestrator.

Rebalance effect on bucket types on page 160
The rebalance operation works across the cluster on both Couchbase and memcached buckets

Swap rebalance on page 161
Swap rebalance is an automatic feature that optimizes the movement of data when you are adding and removing the same number of nodes within the same operation.

Rebalancing a cluster on page 162
Rebalancing a cluster involves marking nodes to be added or removed from the cluster, and then starting the rebalance operation so that the data is moved around the cluster to reflect the new structure.

Rebalancing factors on page 162
Choosing when, why, and how to rebalance your cluster depends on the scenario.

Performing a rebalance on page 163
Once you have configured the nodes that you want to add or remove from your cluster, you must perform a rebalance operation.

Adding nodes on page 164

Removing nodes from clusters on page 167
Removing a node marks the node for removal from the cluster and completely disables the node from serving any requests across the cluster.

Common rebalancing questions on page 168
Common questions and answers about the rebalancing operation are addressed.

Related topics

Rebalancing nodes on page 433

Monitoring a rebalance

Server nodes REST API on page 444
A Couchbase Server instance, known as node, is a physical or virtual machine running Couchbase Server. Each node is as a member of a cluster.

Rebalance behind-the-scenes

The rebalance process is managed through a specific process called the orchestrator.

The orchestrator examines the current vBucket map and then combines that information with the node additions and removals in order to create a new vBucket map.

The orchestrator starts the process of moving the individual vBuckets from the current vBucket map to the new vBucket structure. The process is only started by the orchestrator - the nodes themselves are responsible for actually performing the movement of data between the nodes. The aim is to make the newly calculated vBucket map match the current situation.

Each vBucket is moved independently, and a number of vBuckets can be migrated simultaneously in parallel between the different nodes in the cluster. On each destination node, a process called `ebucketmigrator` is started, which uses the TAP system to request that all the data is transferred for a single vBucket, and that the new vBucket data will become the active vBucket once the migration has been completed.

While the vBucket migration process is taking place, clients are still sending data to the existing vBucket. This information is migrated along with the original data that existed before the migration was requested. Once the migration of the all the data has completed, the original vBucket is marked as disabled, and the new vBucket is enabled. This updates the vBucket map, which is communicated back to the connected clients which will now use the new location.

Changing vBucket moves with REST

The number of vBucket moves that occur during the rebalance operation can be modified. The default is one (1), that is, only one vBucket is moved at a time during the rebalance operation.

To change the number of vBucket moves, execute a curl POST command using the following syntax with the `/internalSettings` endpoint and `rebalanceMovesPerNode` option.

```
curl -X POST -u admin:password
 -d rebalanceMovesPerNode=1
 http://HOST:PORT/internalSettings
```

For example:

```
curl -X POST -u Administrator:password
 -d rebalanceMovesPerNode=14
 http://soursop-s11201.sc.couchbase.com:8091/internalSettings
```

Related Links

Rebalancing on page 158

Rebalance effect on bucket types

The rebalance operation works across the cluster on both Couchbase and memcached buckets

There are differences in the rebalance operation for Couchbase and memcached buckets due to the inherent differences of the two bucket types.

For Couchbase buckets:

- Data is rebalance across all the nodes in the cluster to match the new configuration.
- Updated vBucket map is communicated to clients as each vBucket is successfully moved.
- No data is lost, and there are no changes to the caching or availability of individual keys.

For memcached buckets:

- If new nodes are being added to the cluster, the new node is added to the cluster, and the node is added to the list of nodes supporting the memcached bucket data.
- If nodes are being removed from the cluster, the data stored on that node within the memcached bucket will be lost, and the node removed from the available list of nodes.

- In either case, the list of nodes handling the bucket data is automatically updated and communicated to the client nodes. Memcached buckets use the Ketama hashing algorithm which is designed to cope with server changes, but the change of server nodes may shift the hashing and invalidate some keys once the rebalance operation has completed.

Related Links

Rebalancing on page 158

Swap rebalance

Swap rebalance is an automatic feature that optimizes the movement of data when you are adding and removing the same number of nodes within the same operation.

The swap rebalance optimizes the rebalance operation by moving data directly from the nodes being removed to the nodes being added. This is more efficient than standard rebalancing which would normally move data across the entire cluster.

Swap rebalance occurs automatically if the number of nodes being added and removed are identical. For example, two nodes are marked to be removed and another two nodes are added to the cluster. There is no configuration or selection mechanism to force a swap rebalance. If a swap rebalance cannot take place, then a normal rebalance operation is used instead. When a swap rebalance takes place, the rebalance operates as follows:

- Data is moved directly from a node being removed to a node being added on a one-to-one basis. This eliminates the need to restructure the entire vBucket map.
- Active vBuckets are moved, one at a time, from a source node to a destination node.
- Replica vBuckets are created on the new node and populated with existing data before being activated as the live replica bucket. This ensures that if there is a failure during the rebalance operation, that your replicas are still in place.

For example, if you have a cluster with 20 nodes in it, and configure two nodes (X and Y) to be added, and two nodes to be removed (A and B):

- vBuckets from node A are moved to node X.
- vBuckets from node B are moved to node Y.

The benefits of swap rebalance are:

- Reduced rebalance duration. Since the move takes place directly from the nodes being removed to the nodes being added.
- Reduced load on the cluster during rebalance.
- Reduced network overhead during the rebalance.
- Reduced chance of a rebalance failure if a failover occurs during the rebalance operation, since replicas are created in tandem on the new hosts while the old host replicas still remain available.
- Because data on the nodes are swapped, rather than performing a full rebalance, the capacity of the cluster remains unchanged during the rebalance operation, helping to ensure performance and failover support.

The behavior of the cluster during a failover and rebalance operation with the swap rebalance functionality affects the following situations:

- Stopping a rebalance

  If rebalance fails, or has been deliberately stopped, the active and replica vBuckets that have been transitioned will be part of the active vBucket map. Any transfers still in progress will be canceled. Restarting the rebalance operation will continue the rebalance from where it left off.
- Adding back a failed node

  When a node has failed, removing it and adding a replacement node, or adding the node back, will be treated as swap rebalance.

Important: With swap rebalance functionality, after a node has failed over, either clean up and re-add the failed over node, or add a new node and perform a rebalance as normal. The rebalance is handled as a swap rebalance which minimize the data movements without affecting the overall capacity of the cluster.

Related Links

Rebalancing on page 158

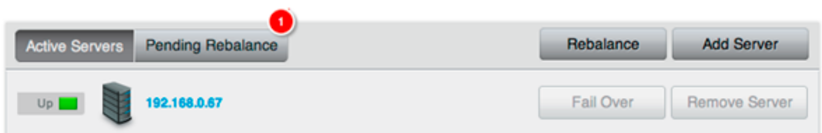
Rebalancing a cluster

Rebalancing a cluster involves marking nodes to be added or removed from the cluster, and then starting the rebalance operation so that the data is moved around the cluster to reflect the new structure.

Important: Until you complete a rebalance, avoid using the failover functionality since it may result in loss of data that has not yet been replicated.

In the event of a failover situation, a rebalance is required to bring the cluster back to a healthy state and re-enable the configured replicas.

The Couchbase web console indicates when the cluster requires a rebalance because the structure of the cluster has been changed, either through adding a node, removing a node, or due to a failover. The notification is through the count of the number of servers that require a rebalance. The following figure shows the `Manage Server Nodes` page.



To rebalance the cluster,click **Rebalance**.

Related Links

Rebalancing on page 158

Rebalancing factors

Choosing when, why, and how to rebalance your cluster depends on the scenario.

Choosing when each of situations applies is not always straightforward. In general, the followith typical triggers and indicators define when to change the node configuration and when to perform a rebalance.

When to expand your cluster

You can increase the size of your cluster by adding more nodes. Adding more nodes increases the available RAM, disk I/O and network bandwidth available to your client applications and helps to spread the load around more machines. There are a few different metrics and statistics that you can use on which to base your decision:

- **Increasing RAM capacity**

  One of the most important components in a Couchbase Server cluster is the amount of RAM available. RAM not only stores application data and supports the Couchbase Server caching layer, it is also actively used for other operations by the server, and a reduction in the overall available RAM may cause performance problems elsewhere.

  The following are common indicators for increasing your RAM capacity within your cluster:

  - If you see more disk fetches occurring, that means that your application is requesting more and more data from disk that is not available in RAM. Increasing the RAM in a cluster will allow it to store more data and therefore provide better performance to your application.
  - If you want to add more buckets to your Couchbase Server cluster you may need more RAM to do so. Adding nodes will increase the overall capacity of the system and then you can shrink any existing buckets in order to make room for new ones.
- **Increasing disk I/O throughput**

  By adding nodes to a Couchbase Server cluster, you will increase the aggregate amount of disk I/O that can be performed across the cluster. This is especially important in high-write environments, but can also be a factor when you need to read large amounts of data from the disk.
- **Increasing disk capacity**

You can either add more disk space to your current nodes or add more nodes to add aggregate disk space to the cluster.

- **Increasing network bandwidth**

  If you see that you are or are close to saturating the network bandwidth of your cluster, this is a very strong indicator of the need for more nodes. More nodes will cause the overall network bandwidth required to be spread out across additional nodes, which will reduce the individual bandwidth of each node.

When to shrink your cluster

Choosing to shrink a Couchbase cluster is a more subjective decision. It is usually based upon cost considerations, or a change in application requirements not requiring as large a cluster to support the required load.

When choosing whether to shrink a cluster:

- You should ensure you have enough capacity in the remaining nodes to support your dataset and application load. Removing nodes may have a significant detrimental effect on your cluster if there are not enough nodes.
- You should avoid removing multiple nodes at once if you are trying to determine the ideal cluster size. Instead, remove each node one at a time to understand the impact on the cluster as a whole.
- You should remove and rebalance a node, rather than using failover. When a node fails and is not coming back to the cluster, the failover functionality will promote its replica vBuckets to become active immediately. If a healthy node is failed over, there might be some data loss for the replication data that was in flight during that operation. Using the remove functionality will ensure that all data is properly replicated and continuously available.

When to rebalance your cluster

Once you decide to add or remove nodes, consider the folloiwng:

- If you're planning on adding and/or removing multiple nodes in a short period of time, it is best to add them all at once and then kick-off the rebalancing operation rather than rebalance after each addition. This will reduce the overall load placed on the system as well as the amount of data that needs to be moved.
- Choose a quiet time for adding nodes. While the rebalancing operation is meant to be performed online, it is not a "free" operation and will undoubtedly put increased load on the system as a whole in the form of disk IO, network bandwidth, CPU resources and RAM usage.
- Voluntary rebalancing (i.e. not part of a failover situation) should be performed during a period of low usage of the system. Rebalancing is a comparatively resource intensive operation as the data is redistributed around the cluster and you should avoid performing a rebalance during heavy usage periods to avoid having a detrimental affect on overall cluster performance.
- Rebalancing requires moving large amounts of data around the cluster. The more RAM that is available will allow the operating system to cache more disk access which will allow it to perform the rebalancing operation much faster. If there is not enough memory in your cluster the rebalancing may be very slow. It is recommended that you don't wait for your cluster to reach full capacity before adding new nodes and rebalancing.

Related Links

Rebalancing on page 158

Performing a rebalance

Once you have configured the nodes that you want to add or remove from your cluster, you must perform a rebalance operation.

A rebalance moves the data around the cluster so that the data is distributed across the entire cluster, removing and adding data to different nodes in the process.

If Couchbase Server identifies that a rebalance is required, either through explicit addition or removal, or through a failover, then the cluster is in a `pending rebalance` state. This does not affect the cluster operation, it merely indicates that a rebalance operation is required to move the cluster into its configured state. To start a rebalance:

Rebalancing via CLI

To initiate a rebalance using the `couchbase-cli` and the `rebalance` command:

```
> couchbase-cli rebalance -c 127.0.0.1:8091 -u Administrator -p Password
  INFO:
 rebalancing . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
  . . . . . . . . . . .
  SUCCESS: rebalanced cluster
```

You can also use this method to add and remove nodes and initiate the rebalance operation using a single command. You can specify nodes to be added using the `--server-add` option, and nodes to be removed using the `--server-remove`. You can use multiple options of each type. For example, to add two nodes, and remove two nodes, and immediately initiate a rebalance operation:

```
> couchbase-cli rebalance -c 127.0.0.1:8091 \
        -u Administrator -p Password \
        --server-add=192.168.0.72 \
        --server-add=192.168.0.73 \
        --server-remove=192.168.0.70 \
        --server-remove=192.168.0.69
```

The command-line provides an active view of the progress and will only return once the rebalance operation has either completed successfully, or in the event of a failure. The time taken for a rebalance operation depends on the number of servers, quantity of data, cluster performance and any existing cluster activity, and is therefore difficult to accurately predict or estimate. Throughout any rebalance operation, monitor the process to ensure that it completes successfully.

Stop the rebalance operation by using the `stop-rebalance` command to `couchbase-cli`.

Related Links

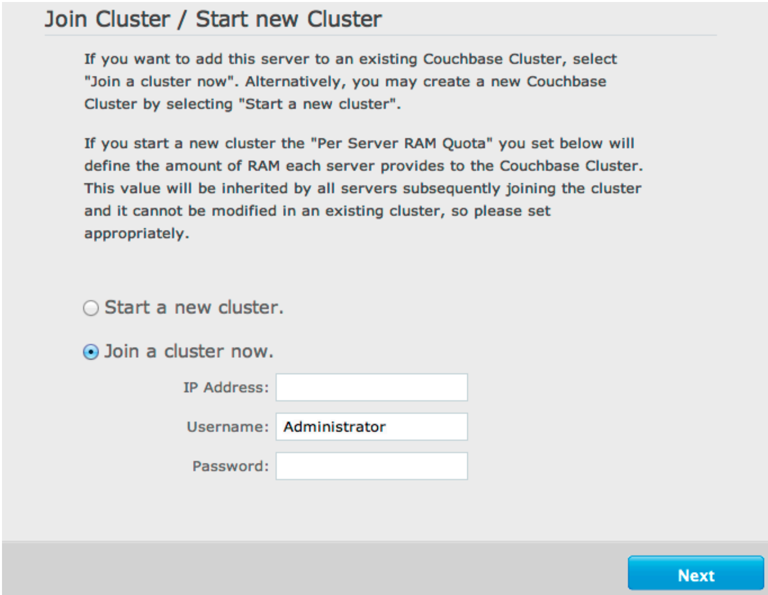
Rebalancing on page 158

Adding nodes

There are a number of methods available for adding a node to a cluster. The result is the same in each case, the node is marked to be added to the cluster, but the node is not an active member until you have performed a rebalance operation. Nodes can be added to a cluster via the UI, CLI, or REST API.

When you are performing the Setup of a new Couchbase Server installation, you have the option of joining the new node to an existing cluster.

During the first step, you can select the `Join a cluster now` radio button, as shown in the figure below:



You are prompted for three pieces of information:

```
* `IP Address`

  The IP address of any existing node within the cluster you want to join.

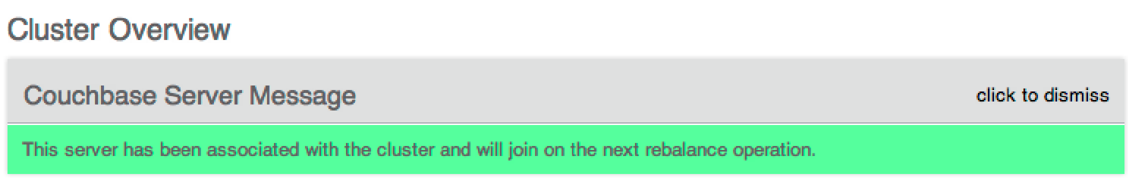
* `Username`

  The username of the administrator of the target cluster.

* `Password`

  The password of the administrator of the target cluster.
```

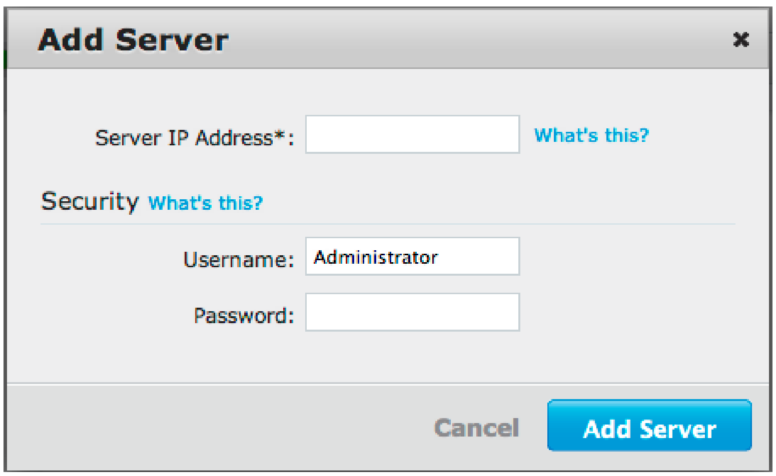
The node will be created as a new cluster, but the pending status of the node within the new cluster will be indicated on the Cluster Overview page, as seen in the example below:



Adding nodes via UI

You can add a new node to an existing cluster after installation by clicking the `Add Server` button within the `Manage Server Nodes` area of the Admin Console. You can see the button in the figure below.

You will be presented with a dialog box, as shown below. Couchbase Server should be installed, and should have been configured as per the normal setup procedures. You can also add a server that has previously been part of this or another cluster using this method. The Couchbase Server must be running.



You need to fill in the requested information:

```
* `Server IP Address`

  The IP address of the server that you want to add.

* `Username`

  The username of the administrator of the target node.

* `Password`

  The password of the administrator of the target node.
```

You will be provided with a warning notifying you that the operation is destructive on the destination server. Any data currently stored on the server will be deleted, and if the server is currently part of another cluster, it will be removed and marked as failed over in that cluster.

Once the information has been entered successfully, the node will be marked as ready to be added to the cluster, and the servers pending rebalance count will be updated.

Adding nodes via REST

With the REST API, you can add nodes to the cluster by providing the IP address, administrator username and password as part of the data payload. For example, using `curl` you could add a new node:

```
> curl -u cluster-username:cluster-password\
    localhost:8091/controller/addNode \
    -d "hostname=192.168.0.68&user=node-username&password=node-password"
```

Adding nodes via CLI

You can use the `couchbase-cli` command-line tool to add one or more nodes to an existing cluster. The new nodes must have Couchbase Server installed, and Couchbase Server must be running on each node.

To add, run the command:

````
```
> couchbase-cli server-add \
      --cluster=localhost:8091 \
      -u cluster-username -p cluster-password \
      --server-add=192.168.0.72:8091 \
      --server-add-username=node-username \
      --server-add-password=node-password
```
````

Where:

Parameter	Description
`--cluster`	The IP address of a node in the existing cluster.
`-u`	The username for the existing cluster.
`-p`	The password for the existing cluster.
`--server-add`	The IP address of the node to be added to the cluster.
`--server-add-username`	The username of the node to be added.
`--server-add-password`	The password of the node to be added.

If the add process is successful, you will see the following response:

```
SUCCESS: server-add 192.168.0.72:8091
```

If you receive a failure message, you will be notified of the type of failure.

You can add multiple nodes in one command by supplying multiple `--server-add` command-line options to the command.

Note: Once a server has been successfully added, the Couchbase Server cluster indicates that a rebalance is required to complete the operation.

You can cancel the addition of a node to a cluster without having to perform a rebalance operation. Canceling the operation removes the server from the cluster without having transferred or exchanged any data, since no rebalance operation took place. You can cancel the operation through the web interface.

Related Links

Rebalancing on page 158

Removing nodes from clusters

Removing a node marks the node for removal from the cluster and completely disables the node from serving any requests across the cluster.

Once a node is removed, a node is no longer part of the cluster in any way and can be switched off, or can be updated or upgraded.

Important

Before you remove a node from the cluster, ensure that you have the capacity within the remaining nodes of your cluster to handle your workload. For more information on the considerations, see Choosing when to shrink your cluster. For the best results, use swap rebalance to swap the node you want to remove out, and swap in a replacement node.

Note: Removing a node does not stop the node from servicing requests. Instead, it only marks the node ready for removal from the cluster. You must perform a rebalance operation to complete the removal process.

Like adding nodes, node can be removed either via the Web Console or CLI.

Web Console

You can remove a node from the cluster from within the `Manage Server Nodes` section of the Web Console, as shown in the figure below.

To remove a node, click the `Remove Server` button next to the node you want to remove. You will be provided with a warning to confirm that you want to remove the node. Click `Remove` to mark the node for removal.

Using the Command-line

You cannot mark a node for removal from the command-line without also initiating a rebalance operation. The `rebalance` command accepts one or more `--server-add` and/or `--server-remove` options. This adds or removes the server from the cluster, and immediately initiates a rebalance operation.

For example, to remove a node during a rebalance operation:

```
> couchbase-cli rebalance --cluster=127.0.0.1:8091 \
          -u Administrator -p Password \
          --server-remove=192.168.0.73
```

Related Links

Rebalancing on page 158

Common rebalancing questions

Common questions and answers about the rebalancing operation are addressed.

How long will rebalancing take?

Because the rebalancing operation moves data stored in RAM and on disk, and continues while the cluster is still servicing client requests, the time required to perform the rebalancing operation is unique to each cluster. Other factors, such as the size and number of objects, speed of the underlying disks used for storage, and the network bandwidth and capacity will also impact the rebalance speed.

Busy clusters may take a significant amount of time to complete the rebalance operation. Similarly, clusters with a large quantity of data to be moved between nodes on the cluster will also take some time for the operation to complete. A busy cluster with lots of data may take a significant amount of time to fully rebalance.

How many nodes can be added or removed?

Functionally there is no limit to the number of nodes that can be added or removed in one operation. However, from a practical level you should be conservative about the numbers of nodes being added or removed at one time.

When expanding your cluster, adding more nodes and performing fewer rebalances is the recommend practice.

When removing nodes, you should take care to ensure that you do not remove too many nodes and significantly reduce the capability and functionality of your cluster.

Remember as well that you can remove nodes, and add nodes, simultaneously. If you are planning on performing a number of addition and removals simultaneously, it is better to add and remove multiple nodes and perform one rebalance, than to perform a rebalance operation with each individual move.

If you are swapping out nodes for servicing, then you can use this method to keep the size and performance of your cluster constant.

Will cluster performance be affected during a rebalance?

By design, there should not be any significant impact on the performance of your application. However, it should be obvious that a rebalance operation implies a significant additional load on the nodes in your cluster, particularly the network and disk I/O performance as data is transferred between the nodes.

Ideally, you should perform a rebalance operation during the quiet periods to reduce the impact on your running applications.

Can I stop a rebalance operation?

The vBuckets within the cluster are moved individually. This means that you can stop a rebalance operation at any time. Only the vBuckets that have been fully migrated will have been made active. You can re-start the rebalance operation at any time to continue the process. Partially migrated vBuckets are not activated.

The one exception to this rule is when removing nodes from the cluster. Stopping the rebalance cancels their removal. You will need to mark these nodes again for removal before continuing the rebalance operation.

To ensure that the necessary clean up occurs, stopping a rebalance incurs a five minute grace period before the rebalance can be restarted. This ensures that the cluster is in a fixed state before rebalance is requested again.

Related Links

Rebalancing on page 158

Managing XDCR

Cross datacenter replication (XDCR) provides an easy method of replicating data from one cluster to another for disaster recovery as well as better data locality (getting data closer to its users).

Related Links

Cluster management on page 135

Configuring XDCR replications on page 169
Configuration of XDCR replications is done on a per-bucket basis.

Conflict resolution in XDCR on page 170
XDCR automatically performs conflict resolution for different document versions on source and destination clusters.

Securing data communication on page 171
To ensure security for the replicated information, configure a suitable VPN gateway between the two datacenters that encrypts the data between each route between datacenters.

Tuning XDCR performance on page 172
XDCR performance can be tuned via the `xdcrOptimisticReplicationThreshold` parameter.

Configuring bi-directional replication on page 174

Modifying XDCR settings on page 174
To modify XDCR advanced settings, use either the Couchbase Server CLI or REST API.

Related topics

XDCR REST API on page 483
Cross Datacenter Replication (XDCR) enables you to automatically replicate data between clusters and between data buckets.

XDCR CLI on page 372

Couchbase Elasticsearch Guide

Amazon Virtual Private Cloud FAQs

Configuring XDCR replications

Configuration of XDCR replications is done on a per-bucket basis.

Replications are configured from the **XDCR** tab of the Web Console. You configure replication on a bucket basis. To replicate data from all buckets in a cluster, individually configure replication for each bucket.

Before configuring XDCR:

- Configure all nodes within each cluster to communicate with all the nodes on the destination cluster. XDCR uses any node in a cluster to replicate between the two clusters.
- Ensure that all Couchbase Server versions and platforms match. For instance, if you want to replicate from a Linux-based cluster, you need to do so with another Linux-based cluster.
- When XDCR performs replication, it exchanges data between clusters over TCP/IP port 8092; Couchbase Server uses TCP/IP port 8091 to exchange cluster configuration information. If you are communicating with a destination cluster over a dedicated connection or the Internet you should ensure that all the nodes in the destination and source clusters can communicate with each other over ports 8091 and 8092.

`Ongoing Replications` are those replications that are currently configured and operating. You can monitor the current configuration, current status, and the last time a replication process was triggered for each configured replication.

Under the XDCR tab you can also configure `Remote Clusters` for XDCR; these are named destination clusters you can select when you configure replication. When you configure XDCR, the destination cluster reference should point to the IP address of one of the nodes in the destination cluster.

Before you set up replication via XDCR, you should be certain that a destination bucket already exists. If this bucket does not exist, replication via XDCR may not find some shards on the destination cluster; this will result in replication of only some data from the source bucket and will significantly delay replication. This would also require you to retry replication multiple times to get a source bucket to be fully replicated to a destination.

Therefore, make sure that you check that a destination bucket exists. The recommended approach is try to read on any key from the bucket. If you receive a 'key not found' error, or the document for the key, the bucket exists and is available to all nodes in a cluster. You can do this via a Couchbase SDK with any node in the cluster.

Related Links

Managing XDCR on page 169
Cross datacenter replication (XDCR) provides an easy method of replicating data from one cluster to another for disaster recovery as well as better data locality (getting data closer to its users).

Conflict resolution in XDCR

XDCR automatically performs conflict resolution for different document versions on source and destination clusters.

The algorithm is designed to consistently select the same document on either a source or destination cluster. For each stored document, XDCR perform checks of metadata to resolve conflicts. It checks the following:

- Numerical sequence, which is incremented on each mutation
- CAS value
- Document flags
- Expiration (TTL) value

If a document does not have the highest revision number, changes to this document will not be stored or replicated; instead the document with the highest score will take precedence on both clusters. Conflict resolution is automatic and does not require any manual correction or selection of documents.

By default XDCR fetches metadata twice from every document before it replicates the document at a destination cluster. XDCR fetches metadata on the source cluster and looks at the number of revisions for a document. It compares this number with the number of revisions on the destination cluster and the document with more revisions is considered the 'winner.'

If XDCR determines a document from a source cluster will win conflict resolution, it puts the document into the replication queue. If the document will lose conflict resolution because it has a lower number of mutations, XDCR will not put it into the replication queue. Once the document reaches the destination, this cluster will request metadata once again to confirm the document on the destination has not changed since the initial check. If the document from the source cluster is still the 'winner' it will be persisted onto disk at the destination. The destination cluster will discard the document version with the lowest number of mutations.

The key point is that the number of document mutations is the main factor that determines whether XDCR keeps a document version or not. This means that the document that has the most recent mutation may not be necessarily the one that wins conflict resolution. If both documents have the same number of mutations, XDCR selects a winner based on other document metadata. Precisely determining which document is the most recently changed is often difficult in a distributed system. The algorithm Couchbase Server uses does ensure that each cluster can independently reach a consistent decision on which document wins.

Related Links

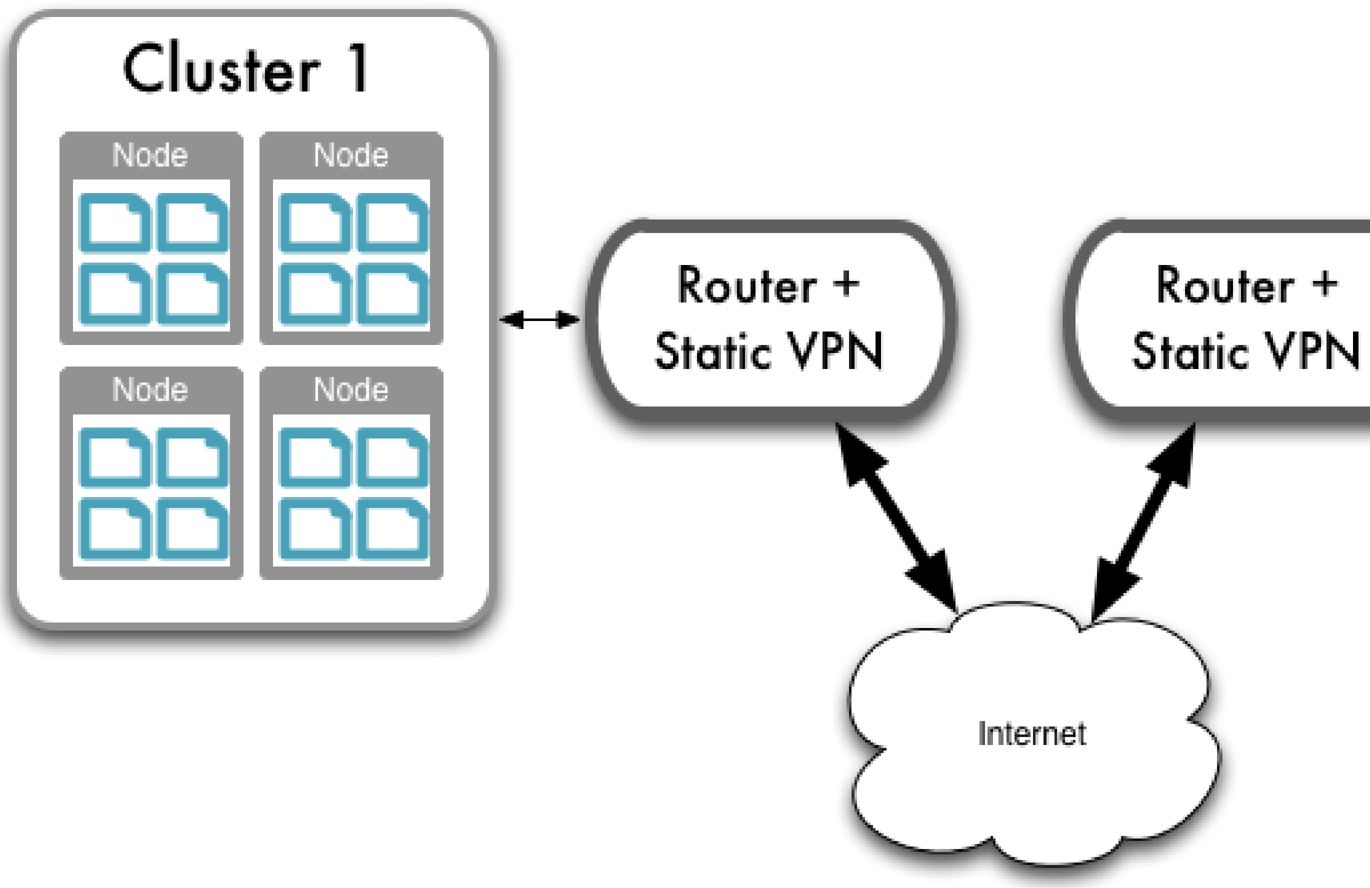
Managing XDCR on page 169
Cross datacenter replication (XDCR) provides an easy method of replicating data from one cluster to another for disaster recovery as well as better data locality (getting data closer to its users).

Securing data communication

To ensure security for the replicated information, configure a suitable VPN gateway between the two datacenters that encrypts the data between each route between datacenters.

When configuring XDCR across multiple clusters over public networks, the data is sent unencrypted across the public interface channel.

Within dedicated datacenters being used for Couchbase Server deployments, you can configure a point to point VPN connection using a static route between the two clusters:

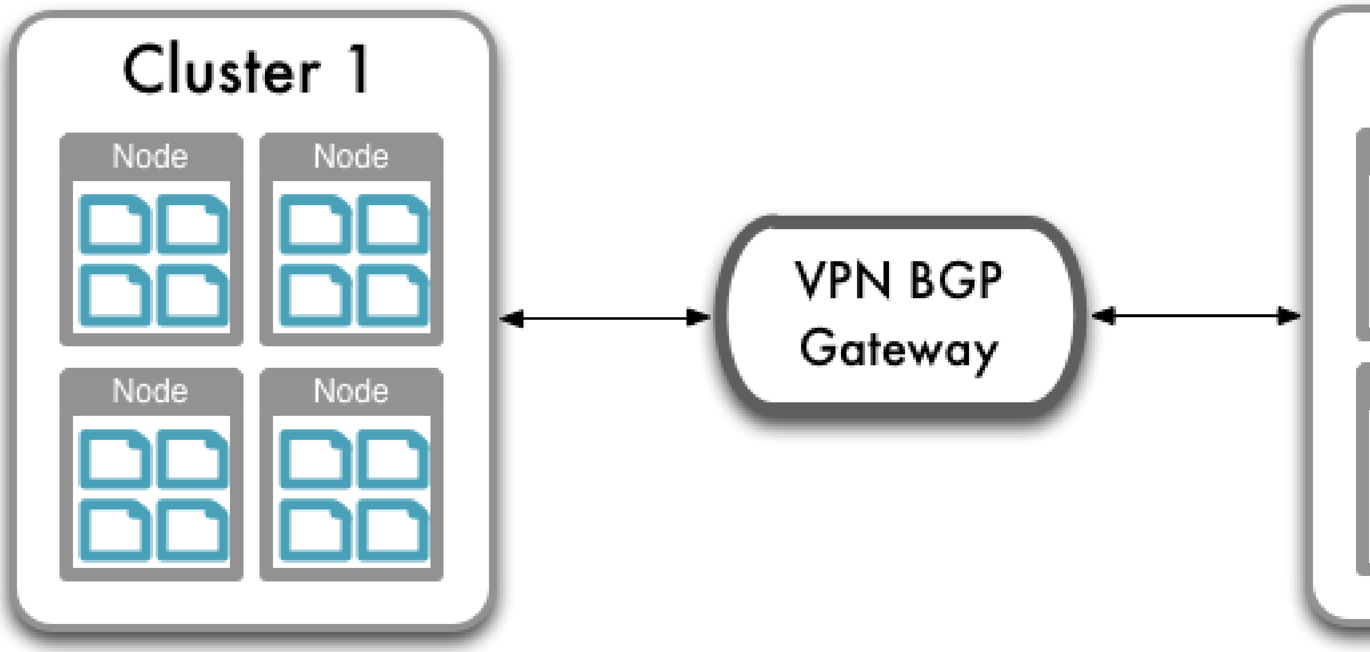


When using Amazon EC2 or other cloud deployment solutions, particularly when using different EC2 zones, there is no built-in VPN support between the different EC2 regional zones. However, there is VPN client support for your cluster within EC2 and Amazon VPC to allow communication to a dedicated VPN solution.

To support cluster to cluster VPN connectivity within EC2:

1. Configure a multi-point BGP VPN solution that can route multiple VPN connections.
2. Route the VPN connection from one EC2 cluster and region to the third-party BGP VPN router.

3. Route the VPN connection from the other region, using the BGP gateway to route between the two VPN connections.



Note: Configuration of these VPN routes and systems is dependent on your VPN solution.

For additional security, configure your security groups to allow traffic only on the required ports between the IP addresses for each cluster. To configure security groups, specify the inbound port and IP address range. You will also need to ensure that the security also includes the right port and IP addresses for the remainder of your cluster to allow communication between the nodes within the cluster.

Important: When configuring your VPN connection, be sure that you route and secure all the ports in use by the XDCR communication protocol, ports 8091 and 8092, on every node within the cluster at each destination.

Related Links

Managing XDCR on page 169
Cross datacenter replication (XDCR) provides an easy method of replicating data from one cluster to another for disaster recovery as well as better data locality (getting data closer to its users).

Tuning XDCR performance

XDCR performance can be tuned via the `xdcrOptimisticReplicationThreshold` parameter.

By default, XDCR gets metadata twice for documents over 256 bytes before it performs conflict resolution for at a destination cluster. If the document fails conflict resolution it will be discarded at the destination cluster.

When a document is smaller than the number of bytes provided as this parameter, XDCR immediately puts it into the replication queue without getting metadata on the source cluster. If the document is deleted on a source cluster, XDCR will no longer fetch metadata for the document before it sends this update to a destination cluster. Once a document reaches the destination cluster, XDCR will fetch the metadata 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 will frequently check metadata. This will increase latency during replication, it also means that it will get metadata 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.'

If you set this very high relative to document size, XDCR will fetch less metadata 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.

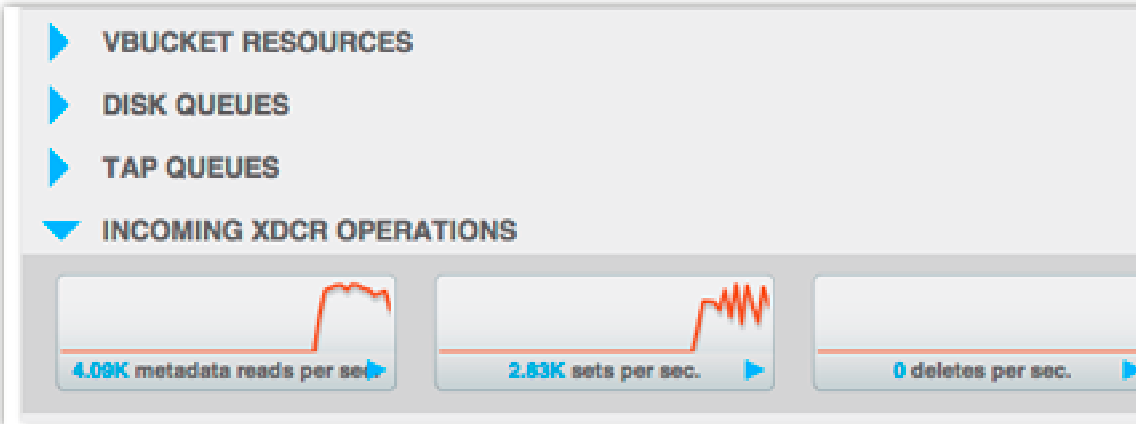
Note: DCR does not fetch metadata for documents that are deleted.

Changing the document threshold

You can change this setting with the REST API as one of the internal settings for XDCR.

Monitoring Optimistic Replication

The easiest way you can monitor the impact of this setting is in Couchbase Web Console. On the Data Buckets tab under Incoming XDCR Operations, you can compare `metadata reads per sec` to `sets per sec`.



If you set a low threshold relative to document size, `metadata reads per sec` will be roughly twice the value of `sets per sec`. If you set a high threshold relative to document size, this will virtually eliminate the first fetch of metadata and therefore `metadata reads per sec` will roughly equal `sets per sec`

The other option is to check the log files for XDCR, which you can find in `/opt/couchbase/var/lib/couchbase/logs` on the nodes for a source bucket. The log files following the naming convention `xdcr.1`, `xdcr.2` and so on. In the logs you will see a series of entries as follows:

```
out of all 11 docs, number of small docs (including dels: 2) is 4,
number of big docs is 7, threshold is 256 bytes,
after conflict resolution at target ("http://
Administrator:asdasd@127.0.0.1:9501/default
%2f3%3ba19c9d4e733a97fa7cb38daa4113d034/"),
out of all big 7 docs the number of docs we need to replicate is: 5;
total # of docs to be replicated is: 9, total latency: 142 ms
```

The first line means that 4 documents are under the threshold and XDCR checked metadata twice for all 7 documents and replicated 5 larger documents and 4 smaller documents. The amount of time to check and replicate all 11 documents was 142 milliseconds.

Related Links

Managing XDCR on page 169
Cross datacenter replication (XDCR) provides an easy method of replicating data from one cluster to another for disaster recovery as well as better data locality (getting data closer to its users).

Related topics

Viewing internal XDCR settings on page 489

Configuring bi-directional replication

Replication is unidirectional from one cluster to another. To configure bidirectional replication between two clusters, 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.

> **Note:** You do not need identical topologies for both clusters. You can have a different number of nodes in each cluster, RAM configuration, and persistence configuration.

To configure a bi-directional replication:

1. Create a replication from Cluster A to Cluster B on Cluster A.
2. Create a replication from Cluster B to Cluster A on Cluster B.
3. Configure the number of parallel replicators that run per node. The default number of parallel, active streams per node is 32 and is adjustable.

Related Links

Managing XDCR on page 169
Cross datacenter replication (XDCR) provides an easy method of replicating data from one cluster to another for disaster recovery as well as better data locality (getting data closer to its users).

Modifying XDCR settings

To modify XDCR advanced settings, use either the Couchbase Server CLI or REST API.

Besides Couchbase Web Console, you can use several Couchbase REST API endpoints to modify XDCR settings. Some of these settings are references used in XDCR and some of these settings will change XDCR behavior or performance:

For the XDCR retry interval, you can provide an environment variable or make a PUT request. By default if XDCR is unable to replicate for any reason like network failures, it will stop and try to reach the remote cluster every 30 seconds if the network is back, XDCR will resume replicating. You can change this default behavior by changing an environment variable or by changing the server parameter `xdcr_failure_restart_interval` with a PUT request:

> **Note:** If you are using XDCR on multiple nodes in cluster and want to change this setting throughout the cluster, you must perform this operation on every node in the cluster.

- By an environment variable:

```
export XDCR_FAILURE_RESTART_INTERVAL=60
```

- By server setting:

```
curl -X POST
  http://Administrator: asdasd@127.0.0.1:8091/diag/eval
  -d 'rpc:call(node(), ns_config, set, [xdcr_failure_restart_interval, 60]).'
```

You can put the system environment variable in a system configuration file on your nodes. When the server restarts, it loads this parameter. If both the environment variable and the server parameter are set, the value for the environment parameter will supersede.

Related Links

Managing XDCR on page 169

Cross datacenter replication (XDCR) provides an easy method of replicating data from one cluster to another for disaster recovery as well as better data locality (getting data closer to its users).

Monitoring

There are a number of different ways to monitor Couchbase servers including underlying processes, ports, and queueing.

Related Links

Couchbase Administration on page 89

Underlying server processes on page 176
There are several server processes that constantly run in Couchbase Server.

Port numbers and accessing different buckets on page 176

Monitoring startup (warmup) on page 177

Disk write queue on page 178
Disk writing is implemented as a 2-queue system: commit to DRAM and then queued to be written to disk

Couchbase Server statistics on page 179
Couchbase Server provides statistics at multiple levels throughout the cluster.

Couchbase Server Moxi statistics on page 181
Regular memcached clients can request statistics through the memcached stats command.

Underlying server processes

There are several server processes that constantly run in Couchbase Server.

These processes occur whether or not the server is actively handling reads/writes or handling other operations from a client application. Right after you start up a node, you may notice a spike in CPU utilization, and the utilization rate will plateau at some level greater than zero. The following describes the ongoing processes that are running on your node:

- **beam.smp on Linux: erl.exe on Windows**

These processes are responsible for monitoring and managing all other underlying server processes such as ongoing XDCR replications, cluster operations, and views.

There is a separate monitoring/babysitting process running on each node. The process is small and simple and therefore unlikely to crash due to lack of memory. It is responsible for spawning and monitoring the second, larger process for cluster management, XDCR and views. It also spawns and monitors the processes for Moxi and memcached. If any of these three processes fail, the monitoring process will re-spawn them.

The main benefit of this approach is that an Erlang VM crash will not cause the Moxi and memcached processes to also crash. You will also see two `beam.smp` or `erl.exe` processes running on Linux or Windows respectively.

The set of log files for this monitoring process is `ns_server.babysitter.log` which you can collect with `cbcollect_info`.

- **memcached** : This process is responsible for caching items in RAM and persisting them to disk.
- **moxi** : This process enables third-party memcached clients to connect to the server.

Related Links

Monitoring on page 176
There are a number of different ways to monitor Couchbase servers including underlying processes, ports, and queueing.

Related topics

cbcollect_info tool on page 376

Port numbers and accessing different buckets

In a Couchbase Server cluster, any communication (stats or data) to a port *other* than 11210 will result in the request going through a Moxi process. This means that any stats request will be aggregated across the cluster (and may produce some inconsistencies or confusion when looking at stats that are not “aggregatable”).

In general, it is best to run all your stat commands against port 11210 which will always give you the information for the specific node that you are sending the request to. It is a best practice to then aggregate the relevant data across nodes at a higher level (in your own script or monitoring system).

When you run the below commands (and all stats commands) without supplying a bucket name and/or password, they will return results for the default bucket and produce an error if one does not exist.

To access a bucket other than the default, you will need to supply the bucket name and/or password on the end of the command. Any bucket created on a dedicated port does not require a password.

The TCP/IP port allocation on Windows by default includes a restricted number of ports available for client communication.

Related Links

Monitoring on page 176
There are a number of different ways to monitor Couchbase servers including underlying processes, ports, and queueing.

Related topics

MSDN: Avoiding TCP/IP Port Exhaustion

Monitoring startup (warmup)

If a Couchbase Server node is starting up for the first time, it creates whatever DB files necessary and begin serving data immediately. However, if there is already data on disk (likely because the node rebooted or the service restarted) the node needs to read all of this data off of disk before it can begin serving data. This is called “warmup”. Depending on the size of data, this can take some time.

When starting up a node, there are a few statistics to monitor. Use the `cbstats` command to watch the warmup and item stats:

```
> cbstats localhost:11210 -b bucket_name -p bucket_password warmup | »
    egrep "warm|curr_items"
```

curr_items:	0
curr_items_tot:	15687
ep_warmed_up:	15687
ep_warmup:	false
ep_warmup_dups:	0
ep_warmup_oom:	0
ep_warmup_thread:	running
ep_warmup_time:	787

And when it is complete:

```
> cbstats localhost:11210 -b bucket_name -p bucket_password warmup | »
    egrep "warm|curr_items"
```

curr_items:	10000
curr_items_tot:	20000
ep_warmed_up:	20000
ep_warmup:	true
ep_warmup_dups:	0
ep_warmup_oom:	0

curr_items:	**10000**
ep_warmup_thread:	complete
ep_warmup_time	1400

Stat	Description
curr_items	The number of items currently active on this node. During warmup, this will be 0 until complete
curr_items_tot	The total number of items this node knows about (active and replica). During warmup, this will be increasing and should match ep_warmed_up
ep_warmed_up	The number of items retrieved from disk. During warmup, this should be increasing.
ep_warmup_dups	The number of duplicate items found on disk. Ideally should be 0, but a few is not a problem
ep_warmup_oom	How many times the warmup process received an Out of Memory response from the server while loading data into RAM
ep_warmup_thread	The status of the warmup thread. Can be either running or complete
ep_warmup_time	How long the warmup thread was running for. During warmup this number should be increasing, when complete it will tell you how long the process took

Related Links

Monitoring on page 176
There are a number of different ways to monitor Couchbase servers including underlying processes, ports, and queueing.

Related topics

Server warmup on page 110
Whenever the Couchbase server is restarted or data is restored to a server instance, the server undergoes a warmup process before data requests can be handled.

Disk write queue

Disk writing is implemented as a 2-queue system: commit to DRAM and then queued to be written to disk

Couchbase Server is a persistent database which means that part of monitoring the system is understanding how we interact with the disk subsystem.

Since Couchbase Server is an asynchronous system, any mutation operation is committed first to DRAM and then queued to be written to disk. The client is returned an acknowledgment almost immediately so that it can continue working. There is replication involved here too, but we're ignoring it for the purposes of this discussion.

Disk writing is implemented as a 2-queue system and are tracked by the stats. The first queue is where mutations are immediately placed. Whenever there are items in that queue, our "flusher" (disk writer) comes along and takes all the items off of that queue, places them into the other one and begins writing to disk. Since disk performance is so dramatically different than RAM, this allows us to continue accepting new writes while we are (possibly slowly) writing new ones to the disk.

The flusher will process 250k items a a time, then perform a disk commit and continue this cycle until its queue is drained. When it has completed everything in its queue, it will either grab the next group from the first queue or essentially sleep until there are more items to write.

Monitoring the disk write queue

There are basically two ways to monitor the disk queue, at a high-level from the Web UI or at a low-level from the individual node statistics.

1. From the Web UI, click on Monitor Data Buckets and select the particular bucket that you want to monitor.
2. Click “Configure View” in the top right corner and select the “Disk Write Queue” statistic. Closing this window shows that there is a new mini-graph.

This graph is showing the Disk Write Queue for all nodes in the cluster. To get a deeper view into this statistic, monitor each node individually using the ‘stats’ output. The statistics to watch are `ep_queue_size` (where new mutations are placed) and `flusher_todo` (the queue of items currently being written to disk).

Related Links

Monitoring on page 176
There are a number of different ways to monitor Couchbase servers including underlying processes, ports, and queueing.

Related topics

Server Nodes on page 252
The **Server Nodes** section shows statistics across the server nodes in the cluster.

Couchbase Server statistics on page 179
Couchbase Server provides statistics at multiple levels throughout the cluster.

Couchbase Server statistics

Couchbase Server provides statistics at multiple levels throughout the cluster.

The statistics used for regular monitoring, capacity planning and to identify the performance characteristics of your cluster deployment. The most visible statistics are those in the Web UI, but components such as the REST interface, the proxy and individual nodes have directly accessible statistics interfaces.

REST interface statistics

To interact with statistics provided by REST, use the Couchbase web console. This GUI gathers statistics via REST and displays them to your browser. The REST interface has a set of resources that provide access to the current and historic statistics the cluster gathers and stores.

Couchbase Server node statistics

Detailed stats documentation can be found in the repository.

Along with stats at the REST and UI level, individual nodes can also be queried for statistics either through a client which uses binary protocol or through the cbstats utility.

For example:

```
> cbstats localhost:11210 all
 auth_cmds:                    9
 auth_errors:                  0
 bucket_conns:                 10
 bytes_read:                   246378222
 bytes_written:                289715944
 cas_badval:                   0
 cas_hits:                     0
 cas_misses:                   0
 cmd_flush:                    0
 cmd_get:                      134250
 cmd_set:                      115750
…
```

The most commonly needed statistics are surfaced through the Web Console and have descriptions there and in the associated documentation. Software developers and system administrators wanting lower level information have it available through the stats interface.

There are seven commands available through the stats interface:

- `stats` (referred to as 'all')
- `dispatcher`
- `hash`
- `tap`
- `timings`
- `vkey`
- `reset`

stats command

This displays a large list of statistics related to the Couchbase process including the underlying engine (ep_* stats).

dispatcher command

This statistic will show what the dispatcher is currently doing:

```
dispatcher
     runtime: 45ms
       state: dispatcher_running
      status: running
        task: Running a flusher loop.
nio_dispatcher
       state: dispatcher_running
      status: idle
```

The first entry, dispatcher, monitors the process responsible for disk access. The second entry is a non-IO (non disk) dispatcher. There may also be a ro_dispatcher dispatcher present if the engine is allowing concurrent reads and writes. When a task is actually running on a given dispatcher, the "runtime" tells you how long the current task has been running. Newer versions will show you a log of recently run dispatcher jobs so you can see what's been happening.

Changing statistics collection

The default Couchbase Server statistics collection is set to collect every second. The tuning that is available for statistic collection is by collecting statistics less frequently.

Note: If statistic collection is changed from the default, the Couchbase service must be restarted.

To change statistic collection:

1. Log in as root or sudo and navigate to the directory where Couchbase is installed. For example: `/opt/couchbase/etc/couchbase/static_config`
2. Edit the static_config file.
3. Add the following parameter: `grab_stats_every_n_ticks, 10`, where 10 is the number of ticks. In the Couchbase environment one tick is one second (default). It is recommended that the statistics collection be more frequent (and accurate). However, assign an appropriate tick value for you environment.
4. Restart the Couchbase service.

After restarting the Couchbase service, the statistics collection rate is changed.

Changing the stats file location

The default stats file location is `/opt/couchbase/var/lib/couchbase/stats`, however, if you want to change the default stats file location, create a symlink location to the new directory.

Note: When creating a symlink, stop and restart the Couchbase service.

Related Links

Monitoring on page 176

There are a number of different ways to monitor Couchbase servers including underlying processes, ports, and queueing.

Related topics

REST API overview on page 426

cbstats tool on page 392

Detailed stats documentation

Couchbase Server Moxi statistics

Regular memcached clients can request statistics through the memcached stats command.

Moxi, as part of it's support of memcached protocol, has support for the memcached `stats` command. The stats command accepts optional arguments, and in the case of Moxi, there is a stats proxy sub-command. A detailed description of statistics available through Moxi can be found in the Moxi 1.8 Manual.

For example, one simple client one may use is the commonly available netcat (output elided with ellipses):

```
$ echo "stats proxy" | nc localhost 11211
STAT basic:version 1.6.0
STAT basic:nthreads 5
…
STAT proxy_main:conf_type dynamic
STAT proxy_main:behavior:cycle 0
STAT proxy_main:behavior:downstream_max 4
STAT proxy_main:behavior:downstream_conn_max 0
STAT proxy_main:behavior:downstream_weight 0
…
STAT proxy_main:stats:stat_configs 1
STAT proxy_main:stats:stat_config_fails 0
STAT proxy_main:stats:stat_proxy_starts 2
STAT proxy_main:stats:stat_proxy_start_fails 0
STAT proxy_main:stats:stat_proxy_existings 0
STAT proxy_main:stats:stat_proxy_shutdowns 0
STAT 11211:default:info:port 11211
STAT 11211:default:info:name default
…
STAT 11211:default:behavior:downstream_protocol 8
STAT 11211:default:behavior:downstream_timeout 0
STAT 11211:default:behavior:wait_queue_timeout 0
STAT 11211:default:behavior:time_stats 0
STAT 11211:default:behavior:connect_max_errors 0
STAT 11211:default:behavior:connect_retry_interval 0
STAT 11211:default:behavior:front_cache_max 200
STAT 11211:default:behavior:front_cache_lifespan 0
STAT 11211:default:behavior:front_cache_spec
STAT 11211:default:behavior:front_cache_unspec
STAT 11211:default:behavior:key_stats_max
STAT 11211:default:behavior:key_stats_lifespan 0
STAT 11211:default:behavior:key_stats_spec
STAT 11211:default:behavior:key_stats_unspec
STAT 11211:default:behavior:optimize_set
STAT 11211:default:behavior:usr default
…
STAT 11211:default:pstd_stats:num_upstream 1
STAT 11211:default:pstd_stats:tot_upstream 2
STAT 11211:default:pstd_stats:num_downstream_conn 1
STAT 11211:default:pstd_stats:tot_downstream_conn 1
STAT 11211:default:pstd_stats:tot_downstream_conn_acquired 1
STAT 11211:default:pstd_stats:tot_downstream_conn_released 1
STAT 11211:default:pstd_stats:tot_downstream_released 2
```

```
STAT 11211:default:pstd_stats:tot_downstream_reserved 1
STAT 11211:default:pstd_stats:tot_downstream_reserved_time 0
STAT 11211:default:pstd_stats:max_downstream_reserved_time 0
STAT 11211:default:pstd_stats:tot_downstream_freed 0
STAT 11211:default:pstd_stats:tot_downstream_quit_server 0
STAT 11211:default:pstd_stats:tot_downstream_max_reached 0
STAT 11211:default:pstd_stats:tot_downstream_create_failed 0
STAT 11211:default:pstd_stats:tot_downstream_connect 1
STAT 11211:default:pstd_stats:tot_downstream_connect_failed 0
STAT 11211:default:pstd_stats:tot_downstream_connect_timeout 0
STAT 11211:default:pstd_stats:tot_downstream_connect_interval 0
STAT 11211:default:pstd_stats:tot_downstream_connect_max_reached 0
…
END
```

Related Links

Monitoring on page 176
There are a number of different ways to monitor Couchbase servers including underlying processes, ports, and queueing.

Related topics

Moxi 1.8 Manual

Views and indexes

Views within Couchbase Server process the information stored in your Couchbase Server database, allowing you to index and query your data. A view creates an index on the stored information according to the format and structure defined within the view. The view consists of specific fields and information extracted from the objects stored in Couchbase. Views create indexes on your information allowing you to search and select information stored within Couchbase Server.

Views are eventually consistent compared to the underlying stored documents. Documents are included in views when the document data is persisted to disk, and documents with expiry times are removed from indexes only when the expiration pager operates to remove the document from the database.

Views can be used within Couchbase Server for a number of reasons, including:

- Indexing and querying data from your stored objects
- Producing lists of data on specific object types
- Producing tables and lists of information based on your stored data
- Extracting or filtering information from the database
- Calculating, summarizing or reducing the information on a collection of stored data

You can create multiple views and therefore multiple indexes and routes into the information stored in your database. By exposing specific fields from the stored information, views enable you to create and query the information stored within your Couchbase Server, perform queries and selection on the information, and paginate through the view output. The View Builder provides an interface for creating your views within the Couchbase Server Web Console. Views can be accessed using a suitable client library to retrieve matching records from the Couchbase Server database.

Related Links

Couchbase Administration on page 89

View basics on page 183
Views allow you to extract specific fields and information from data and create an index.

Views operation on page 184

Views and stored data on page 192

Development views on page 194

Production views on page 195

Writing views on page 196

Views in a schema-less database on page 212

Querying views on page 213

View and query pattern samples on page 222

Translating SQL to map/reduce on page 233

Writing geospatial views on page 238

Related topics

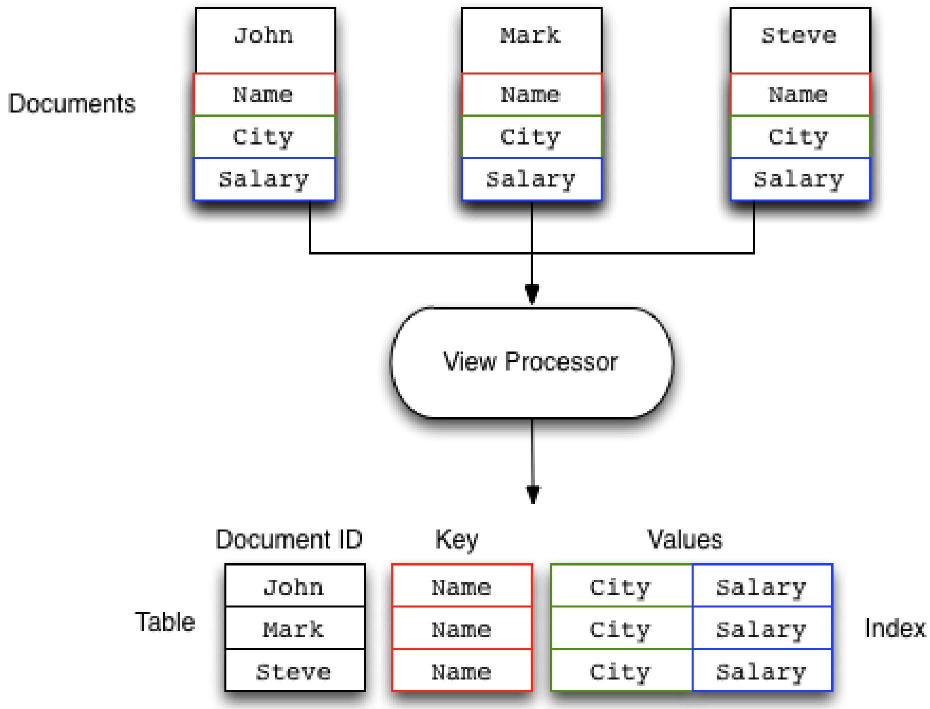
Managing design documents on page 475
Design documents are used to store one or more view definitions. Views can be defined within a design document and uploaded to the server through the REST API.

View basics

Views allow you to extract specific fields and information from data and create an index.

The purpose of a view is take the un-structured, or semi-structured, data stored within your Couchbase Server database, extract the fields and information that you want, and to produce an index of the selected information. Storing information in Couchbase Server using JSON makes the process of selecting individual fields for output easier. The resulting generated structure is a *view* on the stored data. The view that is created during this process allows you to iterate, select and query the information in your database from the raw data objects that have been stored.

A brief overview of this process is shown in the figure below.



In the above example, the view takes the Name, City and Salary fields from the stored documents and then creates a array of this information for each document in the view. A view is created by iterating over every single document within the Couchbase bucket and outputting the specified information. The resulting index is stored for future use and updated with new data stored when the view is accessed. The process is incremental and therefore has a low ongoing impact on performance. Creating a new view on an existing large dataset may take a long time to build, but updates to the data will be quick.

The view definition specifies the format and content of the information generated for each document in the database. Because the process relies on the fields of stored JSON, if the document is not JSON, or the requested field in the view does not exist, the information is ignored. This enables the view to be created, even if some documents have minor errors or lack the relevant fields altogether.

One of the benefits of a document database is the ability to change the format of documents stored in the database at any time, without requiring a wholesale change to applications or a costly schema update before doing so.

Related Links

Views and indexes on page 183

Views operation

All views within Couchbase operate as follows:

- Views are updated when the document data is persisted to disk. There is a delay between creating or updating the document, and the document being updated within the view.
- Documents that are stored with an expiry are not automatically removed until the background expiry process removes them from the database. This means that expired documents may still exist within the index.
- Views are scoped within a design document, with each design document part of a single bucket. A view can only access the information within the corresponding bucket.
- View names must be specified using one or more UTF–8 characters. You cannot have a blank view name. View names cannot have leading or trailing whitespace characters (space, tab, newline, or carriage-return).

- Document IDs that are not UTF–8 encodable are automatically filtered and not included in any view. The filtered documents are logged so that they can be identified.
- If you have a long view request, use POST instead of GET.
- Views can only access documents defined within their corresponding bucket. You cannot access or aggregate data from multiple buckets within a given view.
- Views are created as part of a design document, and each design document exists within the corresponding named bucket.
    - Each design document can have 0-n views.
    - Each bucket can contain 0-n design documents.
- All the views within a single design document are updated when the update to a single view is triggered. For example, a design document with three views will update all three views simultaneously when just one of these views is updated.
- Updates can be triggered in two ways:
    - At the point of access or query by using the `stale` parameter..
    - Automatically by Couchbase Server based on the number of updated documents, or the period since the last update. Automatic updates can be controlled either globally, or individually on each design document.
- Views are updated incrementally. The first time the view is accessed, all the documents within the bucket are processed through the map/reduce functions. Each new access to the view only processes the documents that have been added, updated, or deleted, since the last time the view index was updated.

In practice this means that views are entirely incremental in nature. Updates to views are typically quick as they only update changed documents. You should try to ensure that views are updated, using either the built-in automatic update system, through client-side triggering, or explicit updates within your application framework.

- Because of the incremental nature of the view update process, information is only ever appended to the index stored on disk. This helps ensure that the index is updated efficiently. Compaction (including auto-compaction) will optimize the index size on disk and optimize the index structure. An optimized index is more efficient to update and query.
- The entire view is recreated if the view definition has changed. Because this would have a detrimental effect on live data, only development views can be modified.

Views are organized by design document, and indexes are created according to the design document. Changing a single view in a design document with multiple views invalidates all the views (and stored indexes) within the design document, and all the corresponding views defined in that design document will need to be rebuilt. This will increase the I/O across the cluster while the index is rebuilt, in addition to the I/O required for any active production views.

- You can choose to update the result set from a view before you query it or after you query. Or you can choose to retrieve the existing result set from a view when you query the view. In this case the results are possibly out of date, or stale.
- The views engine creates an index is for each design document; this index contains the results for all the views within that design document.
- The index information stored on disk consists of the combination of both the key and value information defined within your view. The key and value data is stored in the index so that the information can be returned as quickly as possible, and so that views that include a reduce function can return the reduced information by extracting that data from the index.

Because the value and key information from the defined map function are stored in the index, the overall size of the index can be larger than the stored data if the emitted key/value information is larger than the original source document data.

How expiration impacts views

Be aware that Couchbase Server does lazy expiration, that is, expired items are flagged as deleted rather than being immediately erased. Couchbase Server has a maintenance process, called *expiry pager* that will periodically look through all information and erase expired items. This maintenance process will run every 60 minutes, but it can be configured to run at a different interval. Couchbase Server will immediately remove an item flagged for deletion the next time the item requested; the server will respond that the item does not exist to the requesting process.

The result set from a view *will contain* any items stored on disk that meet the requirements of your views function. Therefore information that has not yet been removed from disk may appear as part of a result set when you query a view.

Using Couchbase views, you can also perform *reduce functions* on data, which perform calculations or other aggregations of data. For instance if you want to count the instances of a type of object, you would use a reduce function. Once again, if an item is on disk, it will be included in any calculation performed by your reduce functions. Based on this behavior due to disk persistence, here are guidelines on handling expiration with views:

- **Detecting Expired Documents in Result Sets** : If you are using views for indexing items from Couchbase Server, items that have not yet been removed as part of the expiry pager maintenance process will be part of a result set returned by querying the view. To exclude these items from a result set you should use query parameter `include_docs` set to `true`. This parameter typically includes all JSON documents associated with the keys in a result set. For example, if you use the parameter `include_docs=true` Couchbase Server will return a result set with an additional `"doc"` object which contains the JSON or binary data for that key:

  ```
  {"total_rows":2,"rows":[
  {"id":"test","key":"test","value":null,"doc":{"meta":
  {"id":"test","rev":"4-0000003f04e86b040000000000000000","expiration":0,"flags":0},"jso
  {"testkey":"testvalue"}}},
  {"id":"test2","key":"test2","value":null,"doc":{"meta":
  {"id":"test2","rev":"3-0000004134bd596f50bce37d00000000","expiration":1354556285,"flag
  {"testkey":"testvalue"}}}
  ]
  }
  ```

For expired documents if you set `include_docs=true`, Couchbase Server will return a result set indicating the document does not exist anymore. Specifically, the key that had expired but had not yet been removed by the cleanup process will appear in the result set as a row where `"doc":null` :

````
```
{"total_rows":2,"rows":[
{"id":"test","key":"test","value":null,"doc":{"meta":
{"id":"test","rev":"4-0000003f04e86b040000000000000000","expiration":0,"flags":0},"json"
{"testkey":"testvalue"}}},
{"id":"test2","key":"test2","value":null,"doc":null}
]
}
```
````

- **Reduces and Expired Documents** : In some cases, you may want to perform a *reduce function* to perform aggregations and calculations on data in Couchbase Server. In this case, Couchbase Server takes pre-calculated values which are stored for an index and derives a final result. This also means that any expired items still on disk will be part of the reduction. This may not be an issue for your final result if the ratio of expired items is proportionately low compared to other items. For instance, if you have 10 expired scores still on disk for an average performed over 1 million players, there may be only a minimal level of difference in the final result. However, if you have 10 expired scores on disk for an average performed over 20 players, you would get very different result than the average you would expect.

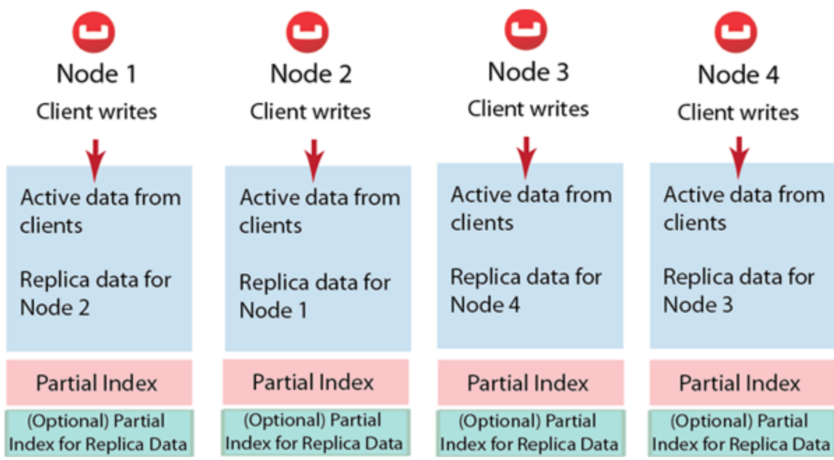
In this case, you may want to run the expiry pager process more frequently to ensure that items that have expired are not included in calculations used in the reduce function. We recommend an interval of 10 minutes for the expiry pager on each node of a cluster. Do note that this interval will have some slight impact on node performance as it will be performing cleanup more frequently on the node.

For more information about setting intervals for the maintenance process, refer to the Couchbase command line tool and review the examples on `exp_pager_stime`.

How views function in a cluster

Distributing data. If you familiar working with Couchbase Server you know that the server distributes data across different nodes in a cluster. This means that if you have four nodes in a cluster, on average each node will contain about 25% of active data. If you use views with Couchbase Server, the indexing process runs on all four nodes and the four nodes will contain roughly 25% of the results from indexing on disk. We refer to this index as a *partial index*, since it is an index based on a subset of data within a cluster. We show this in this partial index in the illustration below.

Replicating data and Indexes. Couchbase Server also provides data replication; this means that the server will replicate data from one node onto another node. In case the first node fails the second node can still handle requests for the data. To handle possible node failure, you can specify that Couchbase Server also replicate a partial index for replicated data. By default each node in a cluster will have a copy of each design document and view functions. If you make any changes to a views function, Couchbase Server will replicate this change to all nodes in the cluster. The sever will generate indexes from views within a single design document and store the indexes in a single file on each node in the cluster:



Couchbase Server can optionally create replica indexes on nodes that are contain replicated data; this is to prepare your cluster for a failover scenario. The server does not replicate index information from another node, instead each node creates an index for the replicated data it stores. The server recreates indexes using the replicated data on a node for each defined design document and view. By providing replica indexes the server enables you to still perform queries even in the event of node failure. You can specify whether Couchbase Server creates replica indexes or not when you create a data bucket.

Query Time within a Cluster

When you query a view and thereby trigger the indexing process, you send that request to a single node in the cluster. This node then distributes the request to all other nodes in the cluster. Depending on the parameter you send in your query, each node will either send the most current partial index at that node, will update the partial index and send it, or send the partial index and update it on disk. Couchbase Server will collect and collate these partial indexes and sent this aggregate result to a client.

To handle errors when you perform a query, you can configure how the cluster behaves when errors occur.

Queries During Rebalance or Failover

You can query an index during cluster rebalance and node failover operations. If you perform queries during rebalance or node failure, Couchbase Server will ensure that you receive the query results that you would expect from a node as if there were no rebalance or node failure.

During node rebalance, you will get the same results you would get as if the data were active data on a node and as if data were not being moved from one node to another. In other words, this feature ensures you get query results from a node during rebalance that are consistent with the query results you would have received from the node before rebalance started. This functionality operates by default in Couchbase Server, however you can optionally choose to disable it. Be aware that while this functionality, when enabled, will cause cluster rebalance to take more time;

however we do not recommend you disable this functionality in production without thorough testing otherwise you may observe inconsistent query results.

View performance

View performance includes the time taken to update the view, the time required for the view update to be accessed, and the time for the updated information to be returned, depend on different factors. Your file system cache, frequency of updates, and the time between updating document data and accessing (or updating) a view will all impact performance.

Some key notes and points are provided below:

- Index queries are always accessed from disk; indexes are not kept in RAM by Couchbase Server. However, frequently used indexes are likely to be stored in the filesystem cache used for caching information on disk. Increasing your filesystem cache, and reducing the RAM allocated to Couchbase Server from the total RAM available will increase the RAM available for the OS.
- The filesystem cache will play a role in the update of the index information process. Recently updated documents are likely to be stored in the filesystem cache. Requesting a view update immediately after an update operation will likely use information from the filesystem cache. The eventual persistence nature implies a small delay between updating a document, it being persisted, and then being updated within the index.

Keeping some RAM reserved for your operating system to allocate filesystem cache, or increasing the RAM allocated to filesystem cache, will help keep space available for index file caching.

- View indexes are stored, accessed, and updated, entirely independently of the document updating system. This means that index updates and retrieval is not dependent on having documents in memory to build the index information. Separate systems also mean that the performance when retrieving and accessing the cluster is not dependent on the document store.

Index updates and the stale parameter

Indexes are created by Couchbase Server based on the view definition, but updating of these indexes can be controlled at the point of data querying, rather than each time data is inserted. Whether the index is updated when queried can be controlled through the `stale` parameter.

Irrespective of the `stale` parameter, documents can only be indexed by the system once the document has been persisted to disk. If the document has not been persisted to disk, use of the `stale` will not force this process. You can use the `observe` operation to monitor when documents are persisted to disk and/or updated in the index.

Views can also be updated automatically according to a document change, or interval count. .

Three values for `stale` are supported:

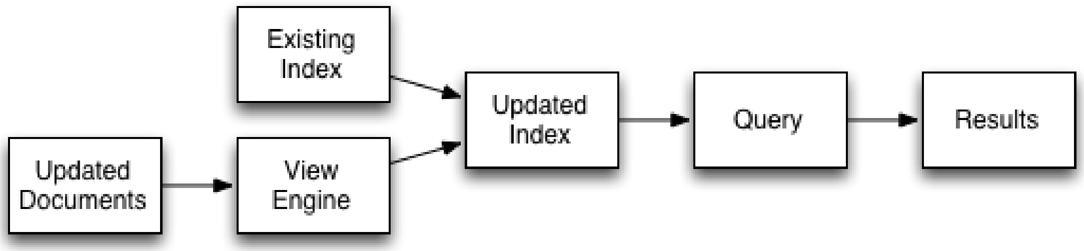
- **stale=ok**

The index is not updated. If an index exists for the given view, then the information in the current index is used as the basis for the query and the results are returned accordingly.

This setting results in the fastest response times to a given query, since the existing index will be used without being updated. However, this risks returning incomplete information if changes have been made to the database and these documents would otherwise be included in the given view.

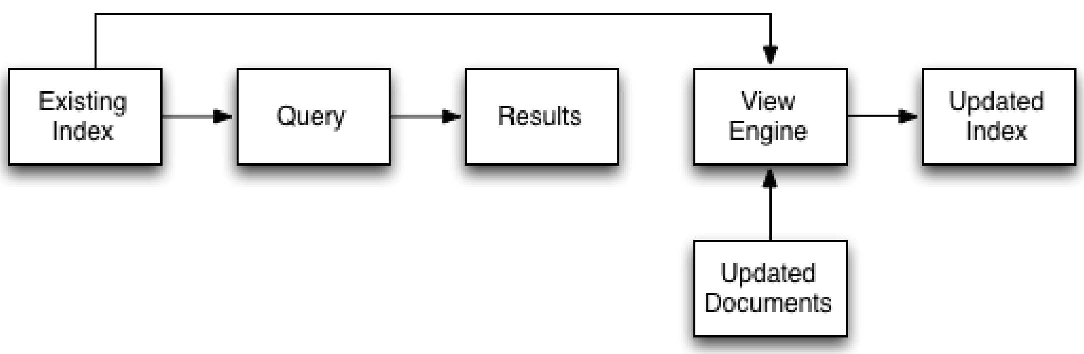
- **stale=false**

The index is updated before the query is executed. This ensures that any documents updated (and persisted to disk) are included in the view. The client will wait until the index has been updated before the query has executed, and therefore the response will be delayed until the updated index is available.



- **stale=update_after**

This is the default setting if no `stale` parameter is specified. The existing index is used as the basis of the query, but the index is marked for updating once the results have been returned to the client.



The indexing engine is an asynchronous process; this means querying an index may produce results you may not expect. For example, if you update a document, and then immediately run a query on that document you may not get the new information in the emitted view data. This is because the document updates have not yet been committed to disk, which is the point when the updates are indexed.

This also means that deleted documents may still appear in the index even after deletion because the deleted document has not yet been removed from the index.

For both scenarios, you should use an `observe` command from a client with the `persistto` argument to verify the persistent state for the document, then force an update of the view using `stale=false`. This will ensure that the document is correctly updated in the view index.

When you have multiple clients accessing an index, the index update process and results returned to clients depend on the parameters passed by each client and the sequence that the clients interact with the server.

- Situation 1

  1. Client 1 queries view with `stale=false`
  2. Client 1 waits until server updates the index
  3. Client 2 queries view with `stale=false` while re-indexing from Client 1 still in progress
  4. Client 2 will wait until existing index process triggered by Client 1 completes. Client 2 gets updated index.
- Situation 2

  1. Client 1 queries view with `stale=false`
  2. Client 1 waits until server updates the index

3. Client 2 queries view with `stale=ok` while re-indexing from Client 1 in progress
4. Client 2 will get the existing index

- Situation 3

1. Client 1 queries view with `stale=false`
2. Client 1 waits until server updates the index
3. Client 2 queries view with `stale=update_after`
4. If re-indexing from Client 1 not done, Client 2 gets the existing index. If re-indexing from Client 1 done, Client 2 gets this updated index and triggers re-indexing.

Index updates may be stacked if multiple clients request that the view is updated before the information is returned (`stale=false`). This ensures that multiple clients updating and querying the index data get the updated document and version of the view each time. For `stale=update_after` queries, no stacking is performed, since all updates occur after the query has been accessed.

Sequential accesses

1. Client 1 queries view with stale=ok
2. Client 2 queries view with stale=false
3. View gets updated
4. Client 1 queries a second time view with stale=ok
5. Client 1 gets the updated view version

The above scenario can cause problems when paginating over a number of records as the record sequence may change between individual queries.

Automated index updates

In addition to a configurable update interval, you can also update all indexes automatically in the background. You configure automated update through two parameters, the update time interval in seconds and the number of document changes that occur before the views engine updates an index. These two parameters are `updateInterval` and `updateMinChanges` :

- `updateInterval` : the time interval in milliseconds, default is 5000 milliseconds. At every `updateInterval` the views engine checks if the number of document mutations on disk is greater than `updateMinChanges`. If true, it triggers view update. The documents stored on disk potentially lag documents that are in-memory for tens of seconds.
- `updateMinChanges` : the number of document changes that occur before re-indexing occurs, default is 5000 changes.

The auto-update process only operates on full-set development and production indexes. Auto-update does not operate on partial set development indexes.

Irrespective of the automated update process, documents can only be indexed by the system once the document has been persisted to disk. If the document has not been persisted to disk, the automated update process will not force the unwritten data to be written to disk. You can use the `observe` operation to monitor when documents have been persisted to disk and/or updated in the index.

The updates are applied as follows:

- Active indexes, Production views

For all active, production views, indexes are automatically updated according to the update interval `updateInterval` and the number of document changes `updateMinChanges`.

If `updateMinChanges` is set to 0 (zero), then automatic updates are disabled for main indexes.

- Replica indexes

If replica indexes have been configured for a bucket, the index is automatically updated according to the document changes (`replicaUpdateMinChanges` ; default 5000) settings.

If `replicaUpdateMinChanges` is set to 0 (zero), then automatic updates are disabled for replica indexes.

The trigger level can be configured both globally and for individual design documents for all indexes using the REST API.

To obtain the current view update daemon settings, access a node within the cluster on the administration port using the URL `http://nodename:8091/settings/viewUpdateDaemon`:

```
GET http://Administrator:Password@nodename:8091/settings/viewUpdateDaemon
```

The request returns the JSON of the current update settings:

```
{
    "updateInterval":5000,
    "updateMinChanges":5000,
    "replicaUpdateMinChanges":5000
}
```

To update the settings, use `POST` with a data payload that includes the updated values. For example, to update the time interval to 10 seconds, and document changes to 7000 each:

```
POST http://nodename:8091/settings/viewUpdateDaemon
updateInterval=10000&updateMinChanges=7000
```

If successful, the return value is the JSON of the updated configuration.

To configure the `updateMinChanges` or `replicaUpdateMinChanges` values explicitly on individual design documents, specify the parameters within the `options` section of the design document. For example:

```
{
   "_id": "_design/myddoc",
   "views": {
      "view1": {
          "map": "function(doc, meta) { if (doc.value) { emit(doc.value,
 meta.id);} }"
      }
   },
   "options": {
       "updateMinChanges": 1000,
       "replicaUpdateMinChanges": 20000
   }
}
```

You can set this information when creating and updating design documents through the design document REST API. To perform this operation using the `curl` tool:

```
> curl -X POST -v -d 'updateInterval=7000&updateMinChanges=7000' \
    'http://Administrator:Password@192.168.0.72:8091/settings/
viewUpdateDaemon'
```

Partial-set development views are not automatically rebuilt, and during a rebalance operation, development views are not updated, even when when consistent views are enabled, as this relies on the automated update mechanism. Updating development views in this way would waste system resources.

Related Links

Views and indexes on page 183

Related topics

Querying views on page 213

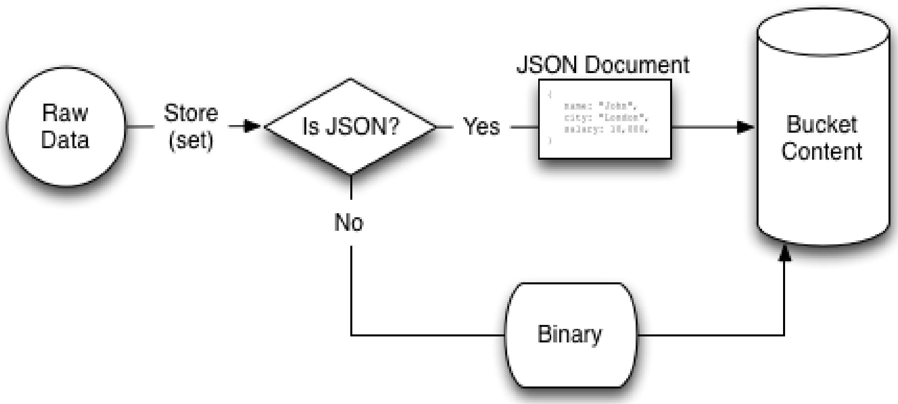
Compaction on page 138
Database and view compaction helps to reclaim disk space and reduce fragmentation.

Managing design documents on page 475
Design documents are used to store one or more view definitions. Views can be defined within a design document and uploaded to the server through the REST API.

Views and stored data

The view system relies on the information stored within your cluster being formatted as a JSON document. The formatting of the data in this form allows the individual fields of the data to be identified and used at the components of the index.

Information is stored into your Couchbase database the data stored is parsed, if the information can be identified as valid JSON then the information is tagged and identified in the database as valid JSON. If the information cannot be parsed as valid JSON then it is stored as a verbatim binary copy of the submitted data.



When retrieving the stored data, the format of the information depends on whether the data was tagged as valid JSON or not:

- **JSON**

Information identified as JSON data may not be returned in a format identical to that stored. The information will be semantically identical, in that the same fields, data and structure as submitted will be returned. Metadata information about the document is presented in a separate structure available during view processing.

The white space, field ordering may differ from the submitted version of the JSON document.

For example, the JSON document below, stored using the key `mykey` :

```
{
   "title" : "Fish Stew",
   "servings" : 4,
   "subtitle" : "Delicious with fresh bread"
}
```

May be returned within the view processor as:

```
{
    "servings": 4,
    "subtitle": "Delicious with fresh bread",
    "title": "Fish Stew"
}
```

- **Non-JSON**

Information not parsable as JSON will always be stored and returned as a binary copy of the information submitted to the database. If you store an image, for example, the data returned will be an identical binary copy of the stored image.

Non-JSON data is available as a base64 string during view processing. A non-JSON document can be identified by examining the `type` field of the metadata structure.

The significance of the returned structure can be seen when editing the view within the Web Console.

JSON basics

JSON is used because it is a lightweight, easily parsed, cross-platform data representation format. There are a multitude of libraries and tools designed to help developers work efficiently with data represented in JSON format, on every platform and every conceivable language and application framework, including, of course, most web browsers.

JSON supports the same basic types as supported by JavaScript, these are:

- Number (either integer or floating-point).

JavaScript supports a maximum numerical value of 2^{53}. If you are working with numbers larger than this from within your client library environment (for example, 64-bit numbers), you must store the value as a string.

- String — this should be enclosed by double-quotes and supports Unicode characters and backslash escaping. For example:

  ```
  "A String"
  ```
- Boolean — a `true` or `false` value. You can use these strings directly. For example:

  ```
  { "value": true}
  ```
- Array — a list of values enclosed in square brackets. For example:

  ```
  ["one", "two", "three"]
  ```
- Object — a set of key/value pairs (i.e. an associative array, or hash). The key must be a string, but the value can be any of the supported JSON values. For example:

  ```
  {
     "servings" : 4,
     "subtitle" : "Easy to make in advance, and then cook when ready",
     "cooktime" : 60,
     "title" : "Chicken Coriander"
  }
  ```

If the submitted data cannot be parsed as a JSON, the information will be stored as a binary object, not a JSON document.

Document metadata

During view processing, metadata about individual documents is exposed through a separate JSON object, `meta`, that can be optionally defined as the second argument to the `map()`. This metadata can be used to further identify and qualify the document being processed.

The `meta` structure contains the following fields and associated information:

- `id`

The ID or key of the stored data object. This is the same as the key used when writing the object to the Couchbase database.

- `rev`

An internal revision ID used internally to track the current revision of the information. The information contained within this field is not consistent or trackable and should not be used in client applications.

- `type`

The type of the data that has been stored. A valid JSON document will have the type `json`. Documents identified as binary data will have the type `base64`.

- `flags`

The numerical value of the flags set when the data was stored. The availability and value of the flags is dependent on the client library you are using to store your data. Internally the flags are stored as a 32-bit integer.

- `expiration`

The expiration value for the stored object. The stored expiration time is always stored as an absolute Unix epoch time value.

These additional fields are only exposed when processing the documents within the view server. These fields are not returned when you access the object through the Memcached/Couchbase protocol as part of the document.

Non-JSON data

All documents stored in Couchbase Server will return a JSON structure, however, only submitted information that could be parsed into a JSON document will be stored as a JSON document. If you store a value that cannot be parsed as a JSON document, the original binary data is stored. This can be identified during view processing by using the `meta` object supplied to the `map()` function.

Information that has been identified and stored as binary documents instead of JSON documents can still be indexed through the views system by creating an index on the key data. This can be particularly useful when the document key is significant. For example, if you store information using a prefix to the key to identify the record type, you can create document-type specific indexes.

Document storage and indexing sequence

The method of storage of information into the Couchbase Server affects how and when the indexing information is built, and when data written to the cluster is incorporated into the indexes. In addition, the indexing of data is also affected by the view system and the settings used when the view is accessed.

The basic storage and indexing sequence is:

1. A document is stored within the cluster. Initially the document is stored only in RAM.
2. The document is persisted to disk through the standard disk write queue mechanism.
3. Once the document has been persisted to disk, the document can be indexed by the view mechanism.

This sequence means that the view results are `eventually consistent` with what is stored in memory based on whether documents have been persisted to disk. It is possible to write a document to the cluster, and access the index, without the newly written document appearing in the generated view index.

Conversely, documents that have been stored with an expiry may continue to be included within the view index until the document has been removed from the database by the expiry pager.

Couchbase Server supports the Observe command, which enables the current state of a document and whether the document has been persisted to disk and/or whether it has been considered for inclusion in an index.

When accessing a view, the contents of the view are asynchronous to the stored documents. In addition, the creation and updating of the view is subject to the `stale` parameter. This controls how and when the view is updated when the view content is queried.

Related Links

Views and indexes on page 183

Related topics

Views operation on page 184

Development views

Due to the nature of the Couchbase cluster and because of the size of the datasets that can be stored across a cluster, the impact of view development needs to be controlled. Creating a view implies the creation of the index which could slow down the performance of your server while the index is being generated. However, views also need to be built and developed using the actively stored information.

To support both the creation and testing of views, and the deployment of views in production, Couchbase Server supports two different view types,Developmentviews andProductionviews. The two view types work identically, but have different purposes and restrictions placed upon their operation.

Development views are designed to be used while you are still selecting and designing your view definitions. While a view is in development mode, views operate with the following attributes:

- By default the development view works on only a subset of the stored information. You can, however, force the generation of a development view information on the full dataset.
- Development views use live data from the selected Couchbase bucket, enabling you to develop and refine your view in real-time on your production data.
- Development views are not automatically rebuilt, and during a rebalance operation, development views are not updated, even when when consistent views are enabled, as this relies on the automated update mechanism. Updating development views in this way would waste system resources.
- Development views are fully editable and modifiable during their lifetime. You can change and update the view definition for a development view at any time. During development of the view, you can view and edit stored document to help develop the view definition.
- Development views are accessed from client libraries through a different URL than production views, making it easy to determine the view type and information during development of your application.

Within the Web Console, the execution of a view by default occurs only over a subset of the full set of documents stored in the bucket. You can elect to run the View over the full set using the Web Console.

Because of the selection process, the reduced set of documents may not be fully representative of all the documents in the bucket. You should always check the view execution over the full set.

Related Links

Views and indexes on page 183

Production views

Due to the nature of the Couchbase cluster and because of the size of the datasets that can be stored across a cluster, the impact of view development needs to be controlled. Creating a view implies the creation of the index which could slow down the performance of your server while the index is being generated. However, views also need to be built and developed using the actively stored information.

To support both the creation and testing of views, and the deployment of views in production, Couchbase Server supports two different view types,Developmentviews andProductionviews. The two view types work identically, but have different purposes and restrictions placed upon their operation.

- **Production views**

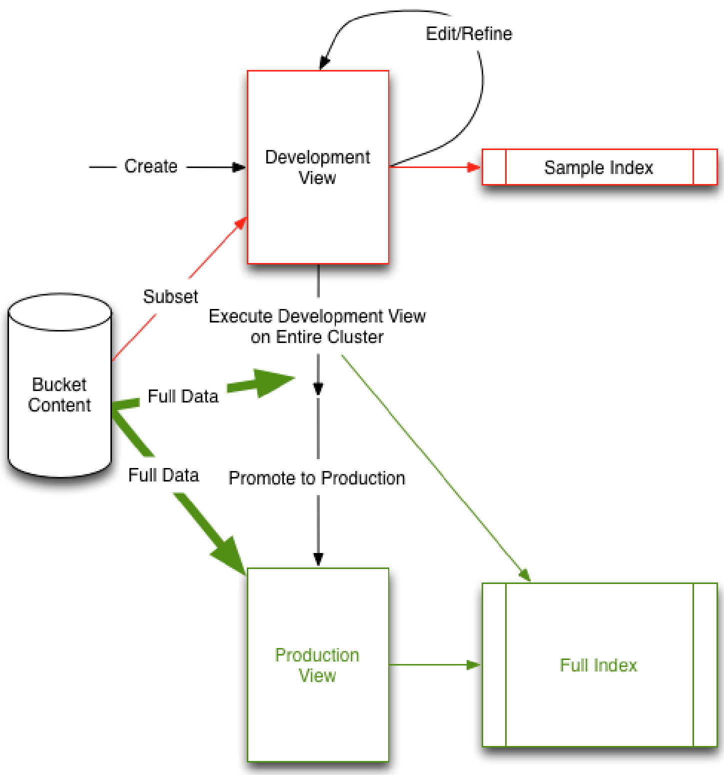
Production views are optimized for production use. A production view has the following attributes:

- Production views always operate on the full dataset for their respective bucket.
- Production views can either be created from the Web Console or through REST API. From the Web Console, you first create development views and then publish them as production views. Through REST API, you directly create the production views (and skip the initial development views).
- Production views cannot be modified through the UI. You can only access the information exposed through a production view. To make changes to a production view, it must be copied to a development view, edited, and re-published.

Views can be updated by the REST API, but updating a production design document immediately invalidates all of the views defined within it.

- Production views are accessed through a different URL to development views.

The support for the two different view types means that there is a typical work flow for view development, as shown in the figure below:



The above diagram features the following steps:

1. Create a development view and view the sample view output.
2. Refine and update your view definition to suit your needs, repeating the process until your view is complete. During this phase you can access your view from your client library and application to ensure it suits your needs.
3. Once the view definition is complete, apply your view to your entire Cluster dataset.
4. Push your development view into production. This moves the view from development into production, and renames the index (so that the index does not need to be rebuilt).
5. Start using your production view.

Individual views are created as part of a design document. Each design document can have multiple views, and each Couchbase bucket can have multiple design documents. You can therefore have both development and production views within the same bucket while you development different indexes on your data.

For information on publishing a view from development to production state.

Related Links

Views and indexes on page 183

Related topics

Views on page 278

The Views section allows you to manage your development and production views.

Writing views

The fundamentals of a view are straightforward. A view creates a perspective on the data stored in your Couchbase buckets in a format that can be used to represent the data in a specific way, define and filter the information, and provide a basis for searching or querying the data in the database based on the content. During the view creation process, you define the output structure, field order, content and any summary or grouping information desired in the view.

Views achieve this by defining an output structure that translates the stored JSON object data into a JSON array or object across two components, the key and the value. This definition is performed through the specification of two separate functions written in JavaScript. The view definition is divided into two parts, a map function and a reduce function:

- **Map function**

As the name suggests, the map function creates a mapping between the input data (the JSON objects stored in your database) and the data as you want it displayed in the results (output) of the view. Every document in the Couchbase bucket for the view is submitted to the `map()` function in each view once, and it is the output from the `map()` function that is used as the result of the view.

The `map()` function is supplied two arguments by the views processor. The first argument is the JSON document data. The optional second argument is the associated metadata for the document, such as the expiration, flags, and revision information.

The map function outputs zero or more 'rows' of information using an `emit()` function. Each call to the `emit()` function is equivalent to a row of data in the view result. The `emit()` function can be called multiple times within the single pass of the `map()` function. This functionality allows you to create views that may expose information stored in a compound format within a single stored JSON record, for example generating a row for each item in an array.

You can see this in the figure below, where the name, salary and city fields of the stored JSON documents are translated into a table (an array of fields) in the generated view content.

- **Reduce function**

The reduce function is used to summarize the content generated during the map phase. Reduce functions are optional in a view and do not have to be defined. When they exist, each row of output (from each `emit()` call in the corresponding `map()` function) is processed by the corresponding `reduce()` function.

If a reduce function is specified in the view definition it is automatically used. You can access a view without enabling the reduce function by disabling reduction (`reduce=false`) when the view is accessed.

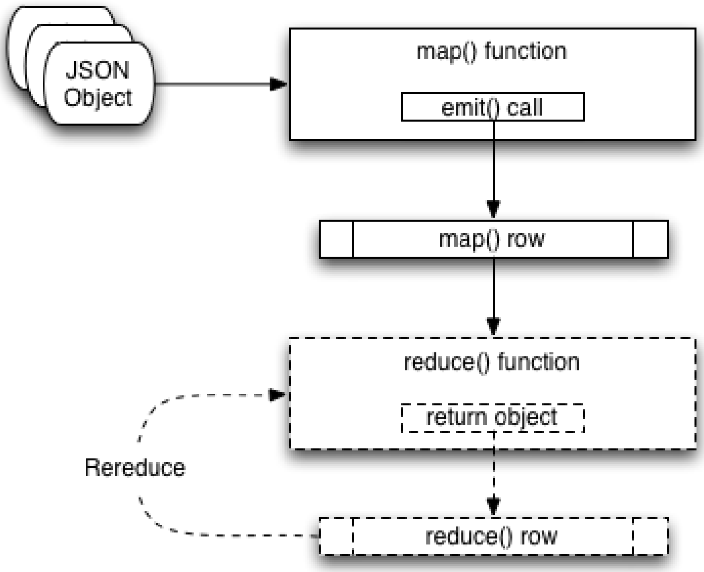
Typical uses for a reduce function are to produce a summarized count of the input data, or to provide sum or other calculations on the input data. For example, if the input data included employee and salary data, the reduce function could be used to produce a count of the people in a specific location, or the total of all the salaries for people in those locations.

The combination of the map and the reduce function produce the corresponding view. The two functions work together, with the map producing the initial material based on the content of each JSON document, and the reduce function summarizing the information generated during the map phase. The reduction process is selectable at the point of accessing the view, you can choose whether to the reduce the content or not, and, by using an array as the key, you can specifying the grouping of the reduce information.

Each row in the output of a view consists of the view key and the view value. When accessing a view using only the map function, the contents of the view key and value are those explicitly stated in the definition. In this mode the view will also always contain an `id` field which contains the document ID of the source record (i.e. the string used as the ID when storing the original data record).

When accessing a view employing both the map and reduce functions the key and value are derived from the output of the reduce function based on the input key and group level specified. A document ID is not automatically included because the document ID cannot be determined from reduced data where multiple records may have been merged into one. Examples of the different explicit and implicit values in views will be shown as the details of the two functions are discussed.

You can see an example of the view creation process in the figure below.



Because of the separation of the two elements, you can consider the two functions individually.

For information on how to write map functions, and how the output of the map function affects and supports searching.

View names must be specified using one or more UTF–8 characters. You cannot have a blank view name. View names cannot have leading or trailing whitespace characters (space, tab, newline, or carriage-return).

To create views, you can use either the Admin Console View editor, use the REST API for design documents, or use one of the client libraries that support view management.

Map functions

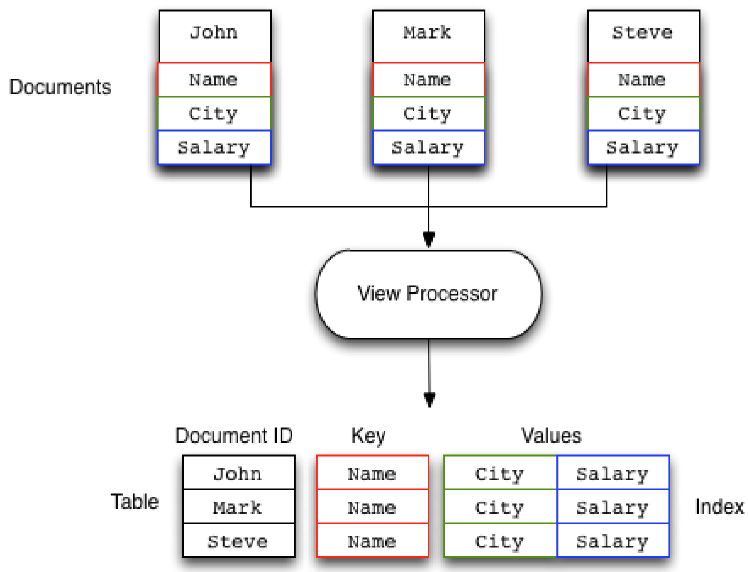
The map function is the most critical part of any view as it provides the logical mapping between the input fields of the individual objects stored within Couchbase to the information output when the view is accessed.

Through this mapping process, the map function and the view provide:

- The output format and structure of the view on the bucket.
- Structure and information used to query and select individual documents using the view information.
- Sorting of the view results.
- Input information for summarizing and reducing the view content.

Applications access views through the REST API, or through a Couchbase client library. All client libraries provide a method for submitting a query into the view system and obtaining and processing the results.

The basic operation of the map function can be seen in the figure below.



In this example, a map function is taking the Name, City, and Salary fields from the JSON documents stored in the Couchbase bucket and mapping them to a table of these fields. The map function which produces this output might look like this:

```
function(doc, meta)
{
  emit(doc.name, [doc.city, doc.salary]);
}
```

When the view is generated the `map()` function is supplied two arguments for each stored document, `doc` and `meta` :

- `doc`

The stored document from the Couchbase bucket, either the JSON or binary content. Content type can be identified by accessing the `type` field of the `meta` argument object.

- `meta`

The metadata for the stored document, containing expiry time, document ID, revision and other information.

Every document in the Couchbase bucket is submitted to the `map()` function in turn. After the view is created, only the documents created or changed since the last update need to be processed by the view. View indexes and updates are materialized when the view is accessed. Any documents added or changed since the last access of the view will be submitted to the `map()` function again so that the view is updated to reflect the current state of the data bucket.

Within the `map()` function itself you can perform any formatting, calculation or other detail. To generate the view information, you use calls to the `emit()` function. Each call to the `emit()` function outputs a single row or record in the generated view content.

The `emit()` function accepts two arguments, the key and the value for each record in the generated view:

- *key*

The emitted key is used by Couchbase Server both for sorting and querying the content in the database.

The key can be formatted in a variety of ways, including as a string or compound value (such as an array or JSON object). The content and structure of the key is important, because it is through the emitted key structure that information is selected within the view.

All views are output in a sorted order according to the content and structure of the key. Keys using a numeric value are sorted numerically, for strings, UTF–8 is used. Keys can also support compound values such as arrays and hashes.

The key content is used for querying by using a combination of this sorting process and the specification of either an explicit key or key range within the query specification. For example, if a view outputs the `RECIPE TITLE` field as a key, you could obtain all the records matching 'Lasagne' by specifying that only the keys matching 'Lasagne' are returned.

- *value*

The value is the information that you want to output in each view row. The value can be anything, including both static data, fields from your JSON objects, and calculated values or strings based on the content of your JSON objects.

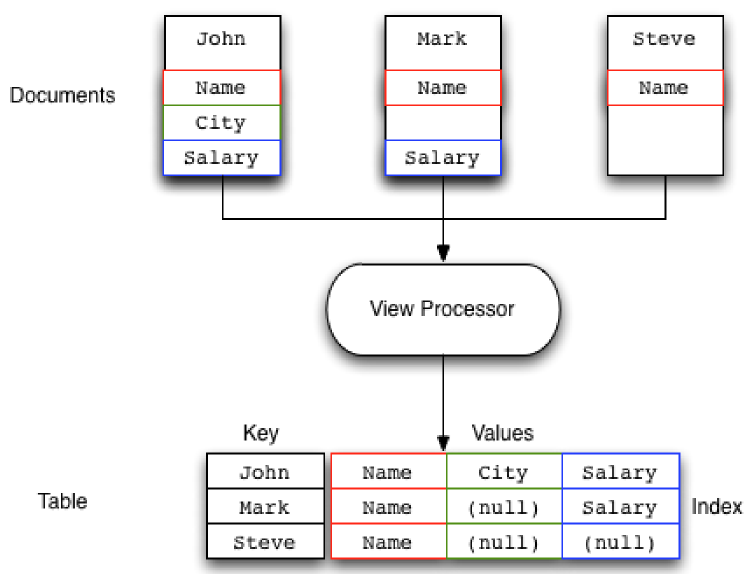
The content of the value is important when performing a reduction, since it is the value that is used during reduction, particularly with the built-in reduction functions. For example, when outputting sales data, you might put the `SALESMAN` into the emitted key, and put the sales amounts into the value. The built-in `_sum` function will then total up the content of the corresponding value for each unique key.

The format of both key and value is up to you. You can format these as single values, strings, or compound values such as arrays or JSON. The structure of the key is important because you must specify keys in the same format as they were generated in the view specification.

The `emit()` function can be called multiple times in a single map function, with each call outputting a single row in the generated view. This can be useful when you want to supporting querying information in the database based on a compound field. For a sample view definition and selection criteria.

Views and map generation are also very forgiving. If you elect to output fields in from the source JSON objects that do not exist, they will simply be replaced with a `null` value, rather than generating an error.

For example, in the view below, some of the source records do contain all of the fields in the specified view. The result in the view result is just the `null` entry for that field in the value output.



You should check that the field or data source exists during the map processing before emitting the data.

To better understand how the map function works to output different types of information and retrieve it, see View and Query Pattern Samples.

Reduce functions

Often the information that you are searching or reporting on needs to be summarized or reduced. There are a number of different occasions when this can be useful. For example, if you want to obtain a count of all the items of a particular type, such as comments, recipes matching an ingredient, or blog entries against a keyword.

When using a reduce function in your view, the value that you specify in the call to `emit()` is replaced with the value generated by the reduce function. This is because the value specified by `emit()` is used as one of the input parameters to the reduce function. The reduce function is designed to reduce a group of values emitted by the corresponding `map()` function.

Alternatively, reduce can be used for performing sums, for example totalling all the invoice values for a single client, or totalling up the preparation and cooking times in a recipe. Any calculation that can be performed on a group of the emitted data.

In each of the above cases, the raw data is the information from one or more rows of information produced by a call to `emit()`. The input data, each record generated by the `emit()` call, is reduced and grouped together to produce a new record in the output.

The grouping is performed based on the value of the emitted key, with the rows of information generated during the map phase being reduced and collated according to the uniqueness of the emitted key.

When using a reduce function the reduction is applied as follows:

- For each record of input, the corresponding reduce function is applied on the row, and the return value from the reduce function is the resulting row.

For example, using the built-in `_sum` reduce function, the `value` in each case would be totaled based on the emitted key:

````
```
{
   "rows" : [
      {"value" : 13000, "id" : "James", "key" : "James" },
      {"value" : 20000, "id" : "James", "key" : "James" },
      {"value" : 5000,  "id" : "Adam",  "key" : "Adam"  },
      {"value" : 8000,  "id" : "Adam",  "key" : "Adam"  },
      {"value" : 10000, "id" : "John",  "key" : "John"  },
      {"value" : 34000, "id" : "John",  "key" : "John"  }
   ]
}
```
````

Using the unique key of the name, the data generated by the map above would be reduced, using the key as the collator, to the produce the following output:

```
{
   "rows" : [
      {"value" : 33000, "key" : "James" },
      {"value" : 13000, "key" : "Adam"  },
      {"value" : 44000, "key" : "John"  },
   ]
}
```

In each case the values for the common keys (John, Adam, James), have been totalled, and the six input rows reduced to the 3 rows shown here.

- Results are grouped on the key from the call to `emit()` if grouping is selected during query time. As shown in the previous example, the reduction operates by the taking the key as the group value as using this as the basis of the reduction.
- If you use an array as the key, and have selected the output to be grouped during querying you can specify the level of the reduction function, which is analogous to the element of the array on which the data should be grouped.

The view definition is flexible. You can select whether the reduce function is applied when the view is accessed. This means that you can access both the reduced and unreduced (map-only) content of the same view. You do not need to create different views to access the two different types of data.

Whenever the reduce function is called, the generated view content contains the same key and value fields for each row, but the key is the selected group (or an array of the group elements according to the group level), and the value is the computed reduction value.

Couchbase includes the following built-in reduce functions:

- `_count`
- `_sum`
- `_stats`.

Note: You can also write your own custom reduction functions.

The reduce function also has a final additional benefit. The results of the computed reduction are stored in the index along with the rest of the view information. This means that when accessing a view with the reduce function enabled, the information comes directly from the index content. This results in a very low impact on the Couchbase Server to the query (the value is not computed at runtime), and results in very fast query times, even when accessing information based on a range-based query.

The `reduce()` function is designed to reduce and summarize the data emitted during the `map()` phase of the process. It should only be used to summarize the data, and not to transform the output information or concatenate the information into a single structure.

When using a composite structure, the size limit on the composite structure within the `reduce()` function is 64KB.

Built-in _count

The `_count` function provides a simple count of the input rows from the `map()` function, using the keys and group level to provide a count of the correlated items. The values generated during the `map()` stage are ignored.

For example, using the input:

```
{
   "rows" : [
      {"value" : 13000, "id" : "James", "key" : ["James", "Paris"] },
      {"value" : 20000, "id" : "James", "key" : ["James", "Tokyo"] },
      {"value" : 5000,  "id" : "James", "key" : ["James", "Paris"] },
      {"value" : 7000,  "id" : "Adam",  "key" : ["Adam",  "London"] },
      {"value" : 19000, "id" : "Adam",  "key" : ["Adam",  "Paris"] },
      {"value" : 17000, "id" : "Adam",  "key" : ["Adam",  "Tokyo"] },
      {"value" : 22000, "id" : "John",  "key" : ["John",  "Paris"] },
      {"value" : 3000,  "id" : "John",  "key" : ["John",  "London"] },
      {"value" : 7000,  "id" : "John",  "key" : ["John",  "London"] },
    ]
}
```

Enabling the `reduce()` function and using a group level of 1 would produce:

```
{
   "rows" : [
      {"value" : 3, "key" : ["Adam" ] },
      {"value" : 3, "key" : ["James"] },
      {"value" : 3, "key" : ["John" ] }
   ]
}
```

The reduction has produce a new result set with the key as an array based on the first element of the array from the map output. The value is the count of the number of records collated by the first element.

Using a group level of 2 would generate the following:

```
{
   "rows" : [
      {"value" : 1, "key" : ["Adam", "London"] },
      {"value" : 1, "key" : ["Adam", "Paris" ] },
      {"value" : 1, "key" : ["Adam", "Tokyo" ] },
```

```
      {"value" : 2, "key" : ["James","Paris" ] },
      {"value" : 1, "key" : ["James","Tokyo" ] },
      {"value" : 2, "key" : ["John", "London"] },
      {"value" : 1, "key" : ["John", "Paris" ] }
   ]
}
```

Now the counts are for the keys matching both the first two elements of the map output.

Built-in _sum

The built-in `_sum` function sums the values from the `map()` function call, this time summing up the information in the value for each row. The information can either be a single number or during a rereduce an array of numbers.

The input values must be a number, not a string-representation of a number. The entire map/reduce will fail if the reduce input is not in the correct format. You should use the `parseInt()` or `parseFloat()` function calls within your `map()` function stage to ensure that the input data is a number.

For example, using the same sales source data, accessing the group level 1 view would produce the total sales for each salesman:

```
{
   "rows" : [
      {"value" : 43000, "key" : [ "Adam"  ] },
      {"value" : 38000, "key" : [ "James" ] },
      {"value" : 32000, "key" : [ "John"  ] }
   ]
}
```

Using a group level of 2 you get the information summarized by salesman and city:

```
{
   "rows" : [
      {"value" : 7000,  "key" : [ "Adam",  "London" ] },
      {"value" : 19000, "key" : [ "Adam",  "Paris"  ] },
      {"value" : 17000, "key" : [ "Adam",  "Tokyo"  ] },
      {"value" : 18000, "key" : [ "James", "Paris"  ] },
      {"value" : 20000, "key" : [ "James", "Tokyo"  ] },
      {"value" : 10000, "key" : [ "John",  "London" ] },
      {"value" : 22000, "key" : [ "John",  "Paris"  ] }
   ]
}
```

Built-in _stats

The built-in `_stats` reduce function produces statistical calculations for the input data. As with the `_sum` function, the corresponding value in the emit call should be a number. The generated statistics include the sum, count, minimum (`min`), maximum (`max`) and sum squared (`sumsqr`) of the input rows.

Using the sales data, a slightly truncated output at group level one would be:

```
{
   "rows" : [
      {
         "value" : {
            "count" : 3,
            "min" : 7000,
            "sumsqr" : 699000000,
            "max" : 19000,
            "sum" : 43000
         },
         "key" : [
            "Adam"
         ]
      },
```

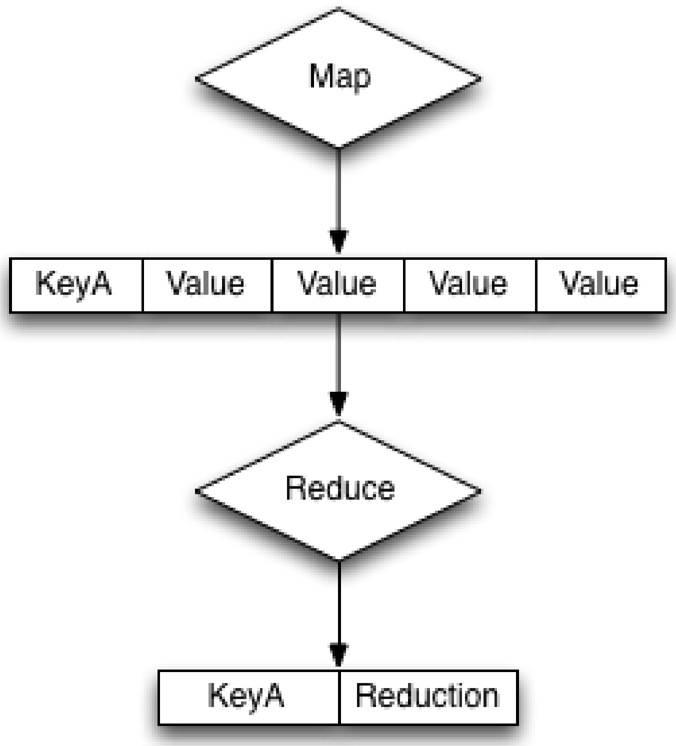
```
        {
            "value" : {
                "count" : 3,
                "min" : 5000,
                "sumsqr" : 594000000,
                "max" : 20000,
                "sum" : 38000
            },
            "key" : [
                "James"
            ]
        },
        {
            "value" : {
                "count" : 3,
                "min" : 3000,
                "sumsqr" : 542000000,
                "max" : 22000,
                "sum" : 32000
            },
            "key" : [
                "John"
            ]
        }
    ]
}
```

The same fields in the output value are provided for each of the reduced output rows.

Writing custom reduce functions

The `reduce()` function has to work slightly differently to the `map()` function. In the primary form, a `reduce()` function must convert the data supplied to it from the corresponding `map()` function.

The core structure of the reduce function execution is shown the figure below.



The base format of the `reduce()` function is as follows:

```
function(key, values, rereduce) {
…

return retval;
}
```

The reduce function is supplied three arguments:

- `key`

The `key` is the unique key derived from the `map()` function and the `group_level` parameter.

- `values`

The `values` argument is an array of all of the values that match a particular key. For example, if the same key is output three times, `data` will be an array of three items containing, with each item containing the value output by the `emit()` function.

- `rereduce`

The `rereduce` indicates whether the function is being called as part of a re-reduce, that is, the reduce function being called again to further reduce the input data.

When `rereduce` is false:

```
* The supplied `key` argument will be an array where the first argument is the
  `key` as emitted by the map function, and the `id` is the document ID that
  generated the key.

* The values is an array of values where each element of the array matches the
  corresponding element within the array of `keys`.
```

When `rereduce` is true:

```
* `key` will be null.
```

```
* `values` will be an array of values as returned by a previous `reduce()`
  function.
```

The function should return the reduced version of the information by calling the `return()` function. The format of the return value should match the format required for the specified key.

Re-writing the built-in reduce functions

Using this model as a template, it is possible to write the full implementation of the built-in functions `_sum` and `_count` when working with the sales data and the standard `map()` function below:

```
function(doc, meta)
{
  emit(meta.id, null);
}
```

The `_count` function returns a count of all the records for a given key. Since argument for the reduce function contains an array of all the values for a given key, the length of the array needs to be returned in the `reduce()` function:

```
function(key, values, rereduce) {
   if (rereduce) {
       var result = 0;
       for (var i = 0; i < values.length; i++) {
           result += values[i];
       }
       return result;
   } else {
       return values.length;
   }
}
```

To explicitly write the equivalent of the built-in `_sum` reduce function, the sum of supplied array of values needs to be returned:

```
function(key, values, rereduce) {
  var sum = 0;
  for(i=0; i < values.length; i++) {
    sum = sum + values[i];
  }
  return(sum);
}
```

In the above function, the array of data values is iterated over and added up, with the final value being returned.

Handling re-reduce

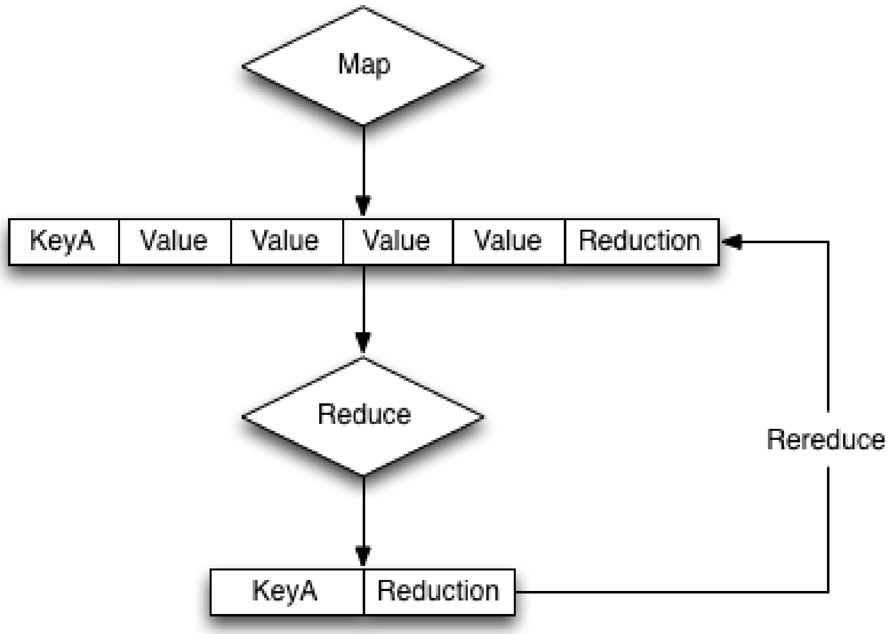
For `reduce()` functions, they should be both transparent and standalone. For example, the `_sum` function did not rely on global variables or parsing of existing data, and didn't need to call itself, hence it is also transparent.

In order to handle incremental map/reduce functionality (i.e. updating an existing view), each function must also be able to handle and consume the functions own output. This is because in an incremental situation, the function must be handle both the new records, and previously computed reductions.

This can be explicitly written as follows:

```
f(keys, values) = f(keys, [ f(keys, values) ])
```

This can been seen graphically in the illustration below, where previous reductions are included within the array of information are re-supplied to the reduce function as an element of the array of values supplied to the reduce function.



That is, the input of a reduce function can be not only the raw data from the map phase, but also the output of a previous reduce phase. This is called `rereduce`, and can be identified by the third argument to the `reduce()`. When the `rereduce` argument is true, both the `key` and `values` arguments are arrays, with the corresponding element in each containing the relevant key and value. I.e., `key[1]` is the key related to the value of `value[1]`.

An example of this can be seen by considering an expanded version of the `sum` function showing the supplied values for the first iteration of the view index building:

```
function('James', [ 13000,20000,5000 ]) {...}
```

When a document with the 'James' key is added to the database, and the view operation is called again to perform an incremental update, the equivalent call is:

```
function('James', [ 19000, function('James', [ 13000,20000,5000 ]) ]) { ... }
```

In reality, the incremental call is supplied the previously computed value, and the newly emitted value from the new document:

```
function('James', [ 19000, 38000 ]) { ... }
```

Fortunately, the simplicity of the structure for `sum` means that the function both expects an array of numbers, and returns a number, so these can easily be recombined.

If writing more complex reductions, where a compound key is output, the `reduce()` function must be able to handle processing an argument of the previous reduction as the compound value in addition to the data generated by the `map()` phase. For example, to generate a compound output showing both the total and count of values, a suitable `reduce()` function could be written like this:

```
function(key, values, rereduce) {
  var result = {total: 0, count: 0};
  for(i=0; i < values.length; i++) {
    if(rereduce) {
        result.total = result.total + values[i].total;
        result.count = result.count + values[i].count;
    } else {
        result.total = sum(values);
        result.count = values.length;
    }
  }
  return(result);
```

```
}
```

Each element of the array supplied to the function is checked using the built-in `typeof` function to identify whether the element was an object (as output by a previous reduce), or a number (from the map phase), and then updates the return value accordingly.

Using the sample sales data, and group level of two, the output from a reduced view may look like this:

```
{"rows":[
{"key":["Adam", "London"],"value":{"total":7000,  "count":1}},
{"key":["Adam", "Paris"], "value":{"total":19000, "count":1}},
{"key":["Adam", "Tokyo"], "value":{"total":17000, "count":1}},
{"key":["James","Paris"], "value":{"total":118000,"count":3}},
{"key":["James","Tokyo"], "value":{"total":20000, "count":1}},
{"key":["John", "London"],"value":{"total":10000, "count":2}},
{"key":["John", "Paris"], "value":{"total":22000, "count":1}}
]
}
```

Reduce functions must be written to cope with this scenario in order to cope with the incremental nature of the view and index building. If this is not handled correctly, the index will fail to be built correctly.

The `reduce()` function is designed to reduce and summarize the data emitted during the `map()` phase of the process. It should only be used to summarize the data, and not to transform the output information or concatenate the information into a single structure.

When using a composite structure, the size limit on the composite structure within the `reduce()` function is 64KB.

Views on non-JSON data

If the data stored within your buckets is not JSON formatted or JSON in nature, then the information is stored in the database as an attachment to a JSON document returned by the core database layer.

This does not mean that you cannot create views on the information, but it does limit the information that you can output with your view to the information exposed by the document key used to store the information.

At the most basic level, this means that you can still do range queries on the key information. For example:

```
function(doc, meta)
{
    emit(meta.id, null);
}
```

You can now perform range queries by using the emitted key data and an appropriate `startkey` and `endkey` value.

If you use a structured format for your keys, for example using a prefix for the data type, or separators used to identify different elements, then your view function can output this information explicitly in the view. For example, if you use a key structure where the document ID is defined as a series of values that are colon separated:

```
OBJECTYPE:APPNAME:OBJECTID
```

You can parse this information within the JavaScript map/reduce query to output each item individually. For example:

```
function(doc, meta)
{
    values = meta.id.split(':',3);
    emit([values[0], values[1], values[2]], null);
}
```

The above function will output a view that consists of a key containing the object type, application name, and unique object ID. You can query the view to obtain all entries of a specific object type using:

```
startkey=['monster', null, null]&endkey=['monster','\u0000' ,'\u0000']
```

Built-in utility functions

Couchbase Server incorporates different utility function beyond the core JavaScript functionality that can be used within `map()` and `reduce()` functions where relevant.

- `dateToArray(date)`

Converts a JavaScript Date object or a valid date string such as “2012–07–30T23:58:22.193Z” into an array of individual date components. For example, the previous string would be converted into a JavaScript array:

```
```
[2012, 7, 30, 23, 58, 22]
```
```

The function can be particularly useful when building views using dates as the key where the use of a reduce function is being used for counting or rollup.

Currently, the function works only on UTC values. Timezones are not supported.

- `decodeBase64(doc)`

Converts a binary (base64) encoded value stored in the database into a string. This can be useful if you want to output or parse the contents of a document that has not been identified as a valid JSON value.

- `sum(array)`

When supplied with an array containing numerical values, each value is summed and the resulting total is returned.

For example:

```
sum([12,34,56,78])
```

View writing best practice

Although you are free to write views matching your data, you should keep in mind the performance and storage implications of creating and organizing the different design document and view definitions.

You should keep the following in mind while developing and deploying your views:

- **Quantity of Views per Design Document**

Because the index for each map/reduce combination within each view within a given design document is updated at the same time, avoid declaring too many views within the same design document. For example, if you have a design document with five different views, all five views will be updated simultaneously, even if only one of the views is accessed.

This can result in increase view index generation times, especially for frequently accessed views. Instead, move frequently used views out to a separate design document.

The exact number of views per design document should be determined from a combination of the update frequency requirements on the included views and grouping of the view definitions. For example, if you have a view that needs to be updated with a high frequency (for example, comments on a blog post), and another view that needs to be updated less frequently (e.g. top blogposts), separate the views into two design documents so that the comments view can be updated frequently, and independently, of the other view.

You can always configure the updating of the view through the use of the `stale` parameter. You can also configure different automated view update times for individual design documents

- **Modifying Existing Views**

If you modify an existing view definition, or are executing a full build on a development view, the entire view will need to be recreated. In addition, all the views defined within the same design document will also be recreated.

Rebuilding all the views within a single design document is an expensive operation in terms of I/O and CPU requirements, as each document will need to be parsed by each views `map()` and `reduce()` functions, with the resulting index stored on disk.

This process of rebuilding will occur across all the nodes within the cluster and increases the overall disk I/O and CPU requirements until the view has been recreated. This process will take place in addition to any production design documents and views that also need to be kept up to date.

- **Don't Include Document ID**

The document ID is automatically output by the view system when the view is accessed. When accessing a view without reduce enabled you can always determine the document ID of the document that generated the row. You should not include the document ID (from `meta.id`) in your key or value data.

- **Check Document Fields**

Fields and attributes from source documentation in `map()` or `reduce()` functions should be checked before their value is checked or compared. This can cause issues because the view definitions in a design document are processed at the same time. A common cause of runtime errors in views is missing or invalid field and attribute checking.

The most common issue is a field within a null object being accessed. This generates a runtime error that will cause execution of all views within the design document to fail. To address this problem, you should check for the existence of a given object before it is used, or the content value is checked. For example, the following view will fail if the `doc.ingredient` object does not exist, because accessing the `length` attribute on a null object will fail:

```
function(doc, meta)
{
    emit(doc.ingredient.ingredtext, null);
}
```

Adding a check for the parent object before calling `emit()` ensures that the function is not called unless the field in the source document exists:

```
function(doc, meta)
{
  if (doc.ingredient)
  {
     emit(doc.ingredient.ingredtext, null);
  }
}
```

The same check should be performed when comparing values within the `if` statement.

This test should be performed on all objects where you are checking the attributes or child values (for example, indices of an array).

- **View Size, Disk Storage and I/O**

Within the map function, the information declared within your `emit()` statement is included in the view index data and stored on disk. Outputting this information will have the following effects on your indexes:

```
* *Increased index size on disk* — More detailed or complex key/value
 combinations
  in generated views will result in more information being stored on disk.

* *Increased disk I/O* — in order to process and store the information on
 disk,
  and retrieve the data when the view is queried. A larger more complex key/
value
  definition in your view will increase the overall disk I/O required both to
  update and read the data back.
```

The result is that the index can be quite large, and in some cases, the size of the index can exceed the size of the original source data by a significant factor if multiple views are created, or you include large portions or the entire document data in the view output.

For example, if each view contains the entire document as part of the value, and you define ten views, the size of your index files will be more than 10 times the size of the original data on which the view was created. With a 500-byte document and 1 million documents, the view index would be approximately 5GB with only 500MB of source data.

- **Including Value Data in Views**

Views store both the key and value emitted by the `emit()`. To ensure the highest performance, views should only emit the minimum key data required to search and select information. The value output by `emit()` should only be used when you need the data to be used within a `reduce()`.

You can obtain the document value by using the core Couchbase API to get individual documents or documents in bulk. Some SDKs can perform this operation for you automatically.

Using this model will also prevent issues where the emitted view data may be inconsistent with the document state and your view is emitting value data from the document which is no longer stored in the document itself.

For views that are not going to be used with reduce, you should output a null value:

````
```
function(doc, meta)
    {
    if(doc.type == 'object')
    emit(doc.experience, null);
    }
```
````

This will create an optimized view containing only the information required, ensuring the highest performance when updating the view, and smaller disk usage.

- **Don't Include Entire Documents in View output**

A view index should be designed to provide base information and through the implicitly returned document ID point to the source document. It is bad practice to include the entire document within your view output.

You can always access the full document data through the client libraries by later requesting the individual document data. This is typically much faster than including the full document data in the view index, and enables you to optimize the index performance without sacrificing the ability to load the full document data.

For example, the following is an example of a bad view:

````
```
function(doc, meta)
    {
    if(doc.type == 'object')
    emit(doc.experience, doc);
    }
```
````

The above view may have significant performance and index size effects.

This will include the full document content in the index.

Instead, the view should be defined as:

````
```
function(doc, meta)
    {
    if(doc.type == 'object')
    emit(doc.experience, null);
    }
```
````

You can then either access the document data individually through the client libraries, or by using the built-in client library option to separately obtain the document data.

- **Using Document Types**

If you are using a document type (by using a field in the stored JSON to indicate the document structure), be aware that on a large database this can mean that the view function is called to update the index for document types that are not being updated or added to the index.

For example, within a database storing game objects with a standard list of objects, and the users that interact with them, you might use a field in the JSON to indicate 'object' or 'player'. With a view that outputs information when the document is an object:

```
```
function(doc, meta)
{
  emit(doc.experience, null);
}
```
```

If only players are added to the bucket, the map/reduce functions to update this view will be executed when the view is updated, even though no new objects are being added to the database. Over time, this can add a significant overhead to the view building process.

In a database organization like this, it can be easier from an application perspective to use separate buckets for the objects and players, and therefore completely separate view index update and structure without requiring to check the document type during progressing.

- **Use Built-in Reduce Functions**

Where possible, use one of the supplied built-in reduce functions, `_sum`, `_count`](#couchbase-views-writing-reduce-count), `_stats`](#couchbase-views-writing-reduce-stats).

These functions are highly optimized. Using a custom reduce function requires additional processing and may impose additional build time on the production of the index.

Related Links

Views and indexes on page 183

Related topics

Views operation on page 184

View and query pattern samples on page 222

Views and stored data on page 192

Writing views on page 196

Managing design documents on page 475

Design documents are used to store one or more view definitions. Views can be defined within a design document and uploaded to the server through the REST API.

Couchbase SDKs

Views in a schema-less database

One of the primary advantages of the document-based storage and the use of map/reduce views for querying the data is that the structure of the stored documents does not need to be predeclared, or even consistent across multiple documents.

Instead, the view can cope with and determine the structure of the incoming documents that are stored in the database, and the view can then reformat and restructure this data during the map/reduce stage. This simplifies the storage of information, both in the initial format, and over time, as the format and structure of the documents can change over time.

For example, you could start storing name information using the following JSON structure:

```
{
   "email" : "mc@example.org",
   "name" : "Martin Brown"
}
```

A view can be defined that outputs the email and name:

```
function(doc, meta)
{
    emit([doc.name, doc.email], null);
}
```

This generates an index containing the name and email information. Over time, the application is adjusted to store the first and last names separately:

```
{
   "email" : "mc@example.org",
   "firstname" : "Martin",
   "lastname" : "Brown"
}
```

The view can be modified to cope with both the older and newer document types, while still emitting a consistent view:

```
function(doc, meta)
{
  if (doc.name && (doc.name != null))
  {
    emit([doc.name, doc.email], null);
  }
  else
  {
    emit([doc.firstname + " " + doc.lastname, doc.email], null);
  }
}
```

The schema-less nature and view definitions allows for a flexible document structure, and an evolving one, without requiring either an initial schema description, or explicit schema updates when the format of the information changes.

Related Links

Views and indexes on page 183

Querying views

In order to query a view, the view definition must include a suitable map function that uses the `emit()` function to generate each row of information. The content of the key that is generated by the `emit()` provides the information on which you can select the data from your view.

The key can be used when querying a view as the selection mechanism, either by using an:

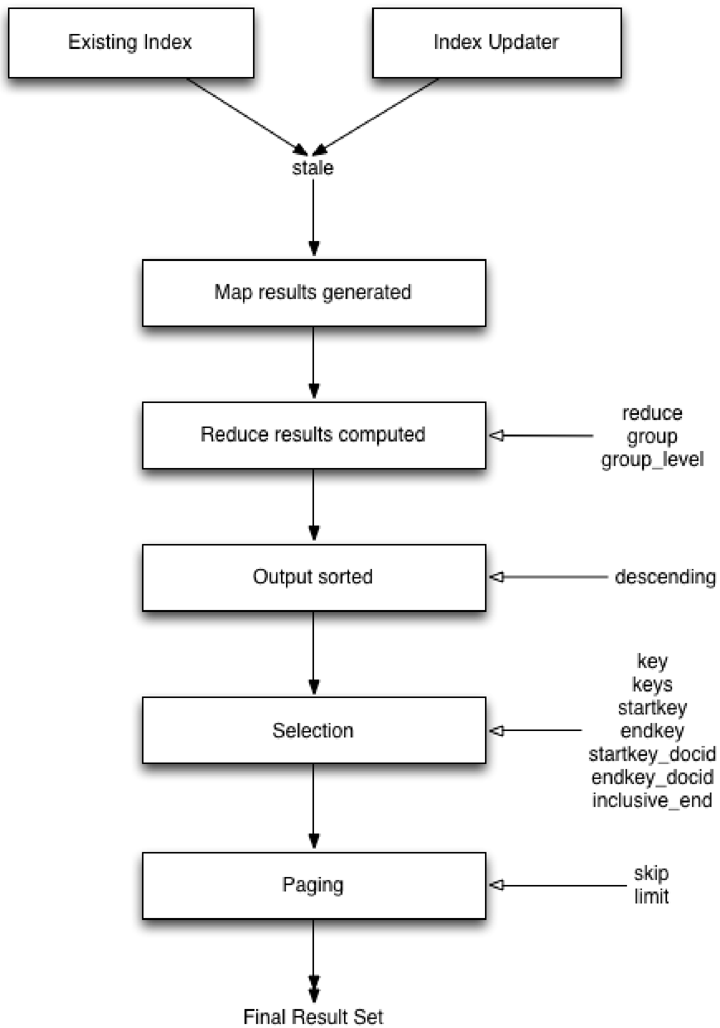
- *explicit key* — show all the records matching the exact structure of the supplied key.
- *list of keys* — show all the records matching the exact structure of each of the supplied keys (effectively showing keya or keyb or keyc).
- *range of keys* — show all the records starting with keya and stopping on the last instance of keyb.

When querying the view results, a number of parameters can be used to select, limit, order and otherwise control the execution of the view and the information that is returned.

When a view is accessed without specifying any parameters, the view will produce results matching the following:

- Full view specification, i.e. all documents are potentially output according to the view definition.
- Limited to 10 items within the Admin Console, unlimited through the REST API.
- Reduce function used if defined in the view.
- Items sorted in ascending order (using UTF–8 comparison for strings, natural number order)

View results and the parameters operate and interact in a specific order. The interaction directly affects how queries are written and data is selected.



The core arguments and selection systems are the same through both the REST API interface, and the client libraries. The setting of these values differs between different client libraries, but the argument names and expected and supported values are the same across all environments.

Querying using the REST API

Querying can be performed through the REST API endpoint. The REST API supports and operates using the core HTTP protocol, and this is the same system used by the client libraries to obtain the view data.

Selecting information

Couchbase Server supports a number of mechanisms for selecting information returned by the view. Key selection is made after the view results (including the reduction function) are executed, and after the items in the view output have been sorted.

When specifying keys to the selection mechanism, the key must be expressed in the form of a JSON value. For example, when specifying a single key, a string must be quoted (“string”).

When specifying the key selection through a parameter, the keys must match the format of the keys emitted by the view. Compound keys, for example where an array or hash has been used in the emitted key structure, the supplied selection value should also be an array or a hash.

The following selection types are supported:

- **Explicit Key**

An explicit key can be specified using the parameter `key`. The view query will only return results where the key in the view output, and the value supplied to the `key` parameter match identically.

For example, if you supply the value “tomato” only records matching *exactly* “tomato” will be selected and returned. Keys with values such as “tomatoes” will not be returned.

- **Key List**

A list of keys to be output can be specified by supplying an array of values using the `keys` parameter. In this instance, each item in the specified array will be used as explicit match to the view result key, with each array value being combined with a logical `or`.

For example, if the value specified to the `keys` parameter was `["tomato","avocado"]`, then all results with a key of ‘tomato’ *or* ‘avocado’ will be returned.

When using this query option, the output results are not sorted by key. This is because key sorting of these values would require collating and sorting all the rows before returning the requested information.

In the event of using a compound key, each compound key must be specified in the query. For example:

````
```
keys=[["tomato",20],["avocado",20]]
```
````

- **Key Range**

A key range, consisting of a `startkey` and `endkey`. These options can be used individually, or together, as follows:

```
* `startkey` only

  Output does not start until the first occurrence of `startkey`, or a value
  greater than the specified value, is seen. Output will then continue until
 the
  end of the view.

* `endkey` only

  Output starts with the first view result, and continues until the last
  occurrence of `endkey`, or until the emitted value is greater than the
 computed
  lexical value of `endkey`.

* `startkey` and `endkey`

  Output of values does not start until `startkey` is seen, and stops when the
  last occurrence of `endkey` is identified.
```

When using `endkey`, the `inclusive_end` option specifies whether output stops after the last occurrence of the specified `endkey` (the default). If set to false, output stops on the last result before the specified `endkey` is seen.

The matching algorithm works on partial values, which can be used to an advantage when searching for ranges of keys.

Selecting compound information by key or keys

If you are generating a compound key within your view, for example when outputting a date split into individually year, month, day elements, then the selection value must exactly match the format and size of your compound key. The value of `key` or `keys` must exactly match the output key structure.

For example, with the view data:

```
{"total_rows":5693,"rows":[
{"id":"1310653019.12667","key":[2011,7,14,14,16,59],"value":null},
{"id":"1310662045.29534","key":[2011,7,14,16,47,25],"value":null},
```

```
{"id":"1310668923.16667","key":[2011,7,14,18,42,3],"value":null},
{"id":"1310675373.9877","key":[2011,7,14,20,29,33],"value":null},
{"id":"1310684917.60772","key":[2011,7,14,23,8,37],"value":null},
{"id":"1310693478.30841","key":[2011,7,15,1,31,18],"value":null},
{"id":"1310694625.02857","key":[2011,7,15,1,50,25],"value":null},
{"id":"1310705375.53361","key":[2011,7,15,4,49,35],"value":null},
{"id":"1310715999.09958","key":[2011,7,15,7,46,39],"value":null},
{"id":"1310716023.73212","key":[2011,7,15,7,47,3],"value":null}
]
}
```

Using the `key` selection mechanism you must specify the entire key value, i.e.:

```
?key=[2011,7,15,7,47,3]
```

If you specify a value, such as only the date:

```
?key=[2011,7,15]
```

The view will return no records, since there is no exact key match. Instead, you must use a range that encompasses the information range you want to output:

```
?startkey=[2011,7,15,0,0,0]&endkey=[2011,7,15,99,99,99]
```

This will output all records within the specified range for the specified date.

Partial selection and key ranges

Matching of the key value has a precedence from right to left for the key value and the supplied `startkey` and/or `endkey`. Partial strings may therefore be specified and return specific information.

For example, given the view data:

```
"a",
 "aa",
 "bb",
 "bbb",
 "c",
 "cc",
 "ccc"
 "dddd"
```

Specifying a `startkey` parameter with the value “aa” will return the last seven records, including “aa”:

```
"aa",
 "bb",
 "bbb",
 "c",
 "cc",
 "ccc",
 "dddd"
```

Specifying a partial string to `startkey` will trigger output of the selected values as soon as the first value or value greater than the specified value is identified. For strings, this partial match (from left to right) is identified. For example, specifying a `startkey` of “d” will return:

```
"dddd"
```

This is because the first match is identified as soon as the a key from a view row matches the supplied `startkey` value *from left to right*. The supplied single character matches the first character of the view output.

When comparing larger strings and compound values the same matching algorithm is used. For example, searching a database of ingredients and specifying a `startkey` of “almond” will return all the ingredients, including “almond”, “almonds”, and “almond essence”.

To match all of the records for a given word or value across the entire range, you can use the null value in the `endkey` parameter. For example, to search for all records that start only with the word “almond”, you specify a `startkey` of “almond”, and an endkey of “almond\u02ad” (i.e. with the last Latin character at the end). If you are using Unicode strings, you may want to use “\uefff”.

```
startkey="almond"&endkey="almond\u02ad"
```

The precedence in this example is that output starts when ‘almond’ is seen, and stops when the emitted data is lexically greater than the supplied `endkey`. Although a record with the value “almond\02ad” will never be seen, the emitted data will eventually be lexically greater than “almond\02ad” and output will stop.

In effect, a range specified in this way acts as a prefix with all the data being output that match the specified prefix.

Partial selection with compound keys

Compound keys, such as arrays or hashes, can also be specified in the view output, and the matching precedence can be used to provide complex selection ranges. For example, if time data is emitted in the following format:

```
[year,month,day,hour,minute]
```

Then precise date (and time) ranges can be selected by specifying the date and time in the generated data. For example, to get information between 1st April 2011, 00:00 and 30th September 2011, 23:59:

```
?startkey=[2011,4,1,0,0]&endkey=[2011,9,30,23,59]
```

The flexible structure and nature of the `startkey` and `endkey` values enable selection through a variety of range specifications. For example, you can obtain all of the data from the beginning of the year until the 5th March using:

```
?startkey=[2011]&endkey=[2011,3,5,23,59]
```

You can also examine data from a specific date through to the end of the month:

```
?startkey=[2011,3,16]&endkey=[2011,3,99]
```

In the above example, the value for the `day` element of the array is an impossible value, but the matching algorithm will identify when the emitted value is lexically greater than the supplied `endkey` value, and information selected for output will be stopped.

A limitation of this structure is that it is not possible to ignore the earlier array values. For example, to select information from 10am to 2pm each day, you cannot use this parameter set:

```
?startkey=[null,null,null,10,0]&endkey=[null,null,null,14,0]
```

In addition, because selection is made by a outputting a range of values based on the start and end key, you cannot specify range values for the date portion of the query:

```
?startkey=[0,0,0,10,0]&endkey=[9999,99,99,14,0]
```

This will instead output all the values from the first day at 10am to the last day at 2pm.

Pagination

Pagination over results can be achieved by using the `skip` and `limit` parameters. For example, to get the first 10 records from the view:

```
?limit=10
```

The next ten records can obtained by specifying:

```
?skip=10&limit=10
```

On the server, the `skip` option works by executing the query and literally iterating over the specified number of output records specified by `skip`, then returning the remainder of the data up until the specified `limit` records are reached, if the `limit` parameter is specified.

When paginating with larger values for `skip`, the overhead for iterating over the records can be significant. A better solution is to track the document id output by the first query (with the `limit` parameter). You can then use `startkey_docid` to specify the last document ID seen, skip over that record, and output the next ten records.

Therefore, the paging sequence is, for the first query:

```
?startkey="carrots"&limit=10
```

Record the last document ID in the generated output, then use:

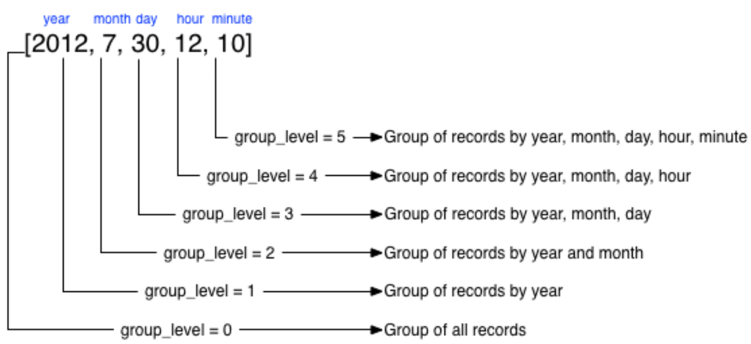
```
?startkey="carrots"&startkey_docid=DOCID&skip=1&limit=10
```

When using `startkey_docid` you must specify the `startkey` parameter to specify the information being searched for. By using the `startkey_docid` parameter, Couchbase Server skips through the B-Tree index to the specified document ID. This is much faster than the skip/limit example shown above.

Grouping in queries

If you have specified an array as your compound key within your view, then you can specify the group level to be applied to the query output when using a `reduce()`.

When grouping is enabled, the view output is grouped according to the key array, and you can specify the level within the defined array that the information is grouped by. You do this by specifying the index within the array by which you want the output grouped using the `group_level` parameter.



The `group_level` parameter specifies the array index (starting at 1) at which you want the grouping occur, and generate a unique value based on this value that is used to identify all the items in the view output that include this unique value:

- A group level of `0` groups by the entire dataset (as if no array exists).
- A group level of `1` groups the content by the unique value of the first element in the view key array. For example, when outputting a date split by year, month, day, hour, minute, each unique year will be output.
- A group level of `2` groups the content by the unique value of the first and second elements in the array. With a date, this outputs each unique year and month, including all records with that year and month into each group.
- A group level of `3` groups the content by the unique value of the first three elements of the view key array. In a date this outputs each unique date (year, month, day) grouping all items according to these first three elements.

The grouping will work for any output structure where you have output an compound key using an array as the output value for the key.

Selection when grouping

When using grouping and selection using the `key`, `keys`, or `startkey` / `endkey` parameters, the query value should match at least the format (and element count) of the group level that is being queried.

For example, using the following `map()` function to output information by date as an array:

```
function(doc, meta)
{
  emit([doc.year, doc.mon, doc.day], doc.logtype);
}
```

If you specify a `group_level` of `2` then you must specify a key using at least the year and month information. For example, you can specify an explicit key, such as `[2012,8]` :

```
?group=true&group_level=2&key=[2012,8]
```

You can query it for a range:

```
?group=true&group_level=2&startkey=[2012,2]&endkey=[2012,8]
```

You can also specify a year, month and day, while still grouping at a higher level. For example, to group by year/month while selecting by specific dates:

```
?group=true&group_level=2&startkey=[2012,2,15]&endkey=[2012,8,10]
```

Specifying compound keys that are shorter than the specified group level may output unexpected results due to the selection mechanism and the way `startkey` and `endkey` are used to start and stop the selection of output rows.

Ordering

All view results are automatically output sorted, with the sorting based on the content of the key in the output view. Views are sorted using a specific sorting format, with the basic order for all basic and compound follows as follows:

- `null`
- `false`
- `true`
- Numbers
- Text (case sensitive, lowercase first, UTF–8 order)
- Arrays (according to the values of each element, in order)
- Objects (according to the values of keys, in key order)

The natural sorting is therefore by default close to natural sorting order both alphabetically (A-Z) and numerically (0–9).

There is no collation or foreign language support. Sorting is always according to the above rules based on UTF–8 values.

You can alter the direction of the sorting (reverse, highest to lowest numerically, Z-A alphabetically) by using the `descending` option. When set to true, this reverses the order of the view results, ordered by their key.

Because selection is made after sorting the view results, if you configure the results to be sorted in descending order and you are selecting information using a key range, then you must also reverse the `startkey` and `endkey` parameters. For example, if you query ingredients where the start key is ‘tomato’ and the end key is ‘zucchini’, for example:

```
?startkey="tomato"&endkey="zucchini"
```

The selection will operate, returning information when the first key matches ‘tomato’ and stopping on the last key that matches ‘zucchini’.

If the return order is reversed:

```
?descending=true&startkey="tomato"&endkey="zucchini"
```

The query will return only entries matching ‘tomato’. This is because the order will be reversed, ‘zucchini’ will appear first, and it is only when the results contain ‘tomato’ that any information is returned.

To get all the entries that match, the `startkey` and `endkey` values must also be reversed:

```
?descending=true&startkey="zucchini"&endkey="tomato"
```

The above selection will start generating results when ‘zucchini’ is identified in the key, and stop returning results when ‘tomato’ is identified in the key.

View output and selection are case sensitive. Specifying the key ‘Apple’ will not return ‘apple’ or ‘APPLE’ or other case differences. Normalizing the view output and query input to all lowercase or upper case will simplify the process by eliminating the case differences.

Understanding letter ordering in views

Couchbase Server uses a Unicode collation algorithm to order letters, so you should be aware of how this functions. Most developers are typically used to Byte order, such as that found in ASCII and which is used in most programming languages for ordering strings during string comparisons.

The following shows the order of precedence used in Byte order, such as ASCII:

```
123456890 < A-Z < a-z
```

This means any items that start with integers will appear before any items with letters; any items that beginning with capital letters will appear before items in lower case letters. This means the item named "Apple" will appear before "apple" and the item "Zebra" will appear before "apple". Compare this with the order of precedence used in Unicode collation, which is used in Couchbase Server:

```
123456790 < aAbBcCdDeEfFgGhH...
```

Notice again that items that start with integers will appear before any items with letters. However, in this case, the lowercase and then uppercase of the same letter are grouped together. This means that that if "apple" will appear before "Apple" and would also appear before "Zebra." In addition, be aware that with accented characters will follow this ordering:

```
a < á < A < Á < b
```

This means that all items starting with "a" *and accented variants of the letter* will occur before "A" and any accented variants of "A."

Ordering Example

In Byte order, keys in an index would appear as follows:

```
"ABC123" < "ABC223" < "abc123" < "abc223" < "abcd23" < "bbc123" < "bbcd23"
```

The same items will be ordered this way by Couchbase Server under Unicode collation:

```
"abc123" < "ABC123" < "abc223" < "ABC223" < "abcd23" < "bbc123" < "bbcd23"
```

This is particularly important for you to understand if you query Couchbase Server with a `startkey` and `endkey` to get back a range of results. The items you would retrieve under Byte order are different compared to Unicode collation.

Ordering and Query Example

This following example demonstrates Unicode collation in Couchbase Server and the impact on query results returned with a `startkey` and `endkey`. It is based on the `beer-sample` database provided with Couchbase Server.

Imagine you want to retrieve all breweries with names starting with uppercase Y. Your query parameters would appear as follows:

```
startkey="Y"&endkey="z"
```

If you want breweries starting with lowercase y *or* uppercase Y, you would provides a query as follows:

```
startkey="y"&endkey="z"
```

This will return all names with lower case Y and items up to, but not including lowercase z, thereby including uppercase Y as well. To retrieve the names of breweries starting with lowercase y only, you would terminate your range with capital Y:

```
startkey="y"&endkey="Y"
```

As it happens, the sample database does not contain any results because there are no beers in it which start with lowercase `y`.

Error control

There are a number of parameters that can be used to help control errors and responses during a view query.

- `on_error`

The `on_error` parameter specifies whether the view results will be terminated on the first error from a node, or whether individual nodes can fail and other nodes return information.

When returning the information generated by a view request, the default response is for any raised error to be included as part of the JSON response, but for the view process to continue. This allows for individual nodes within the Couchbase cluster to timeout or fail, while still generating the requested view information.

In this instance, the error is included as part of the JSON returned:

```
{
   "errors" : [
      {
         "from" : "http://192.168.1.80:9503/_view_merge/?stale=false",
         "reason" : "req_timedout"
      },
      {
         "from" : "http://192.168.1.80:9502/_view_merge/?stale=false",
         "reason" : "req_timedout"
      },
      {
         "from" : "http://192.168.1.80:9501/_view_merge/?stale=false",
         "reason" : "req_timedout"
      }
   ],
   "rows" : [
      {
         "value" : 333280,
         "key" : null
      }
   ]
}
```

You can alter this behavior by using the `on_error` argument. The default value is `continue`. If you set this value to `stop` then the view response will cease the moment an error occurs. The returned JSON will contain the error information for the node that returned the first error. For example:

````
```
{
   "errors" : [
      {
         "from" : "http://192.168.1.80:9501/_view_merge/?stale=false",
         "reason" : "req_timedout"
      }
   ],
   "rows" : [
      {
         "value" : 333280,
         "key" : null
      }
   ]
}
```
````

Related Links

Views and indexes on page 183

Related topics

Writing views on page 196

View and query pattern samples on page 222

Views REST API on page 475

Unicode Technical Standard #10

ICU User Guide, Customization, Default Options

View and query pattern samples

Building views and querying the indexes they generate is a combined process based both on the document structure and the view definition. Writing an effective view to query your data may require changing or altering your document structure, or creating a more complex view in order to allow the specific selection of the data through the querying mechanism.

For background and examples, the following selections provide a number of different scenarios and examples have been built to demonstrate the document structures, views and querying parameters required for different situations.

General advice

There are some general points and advice for writing all views that apply irrespective of the document structure, query format, or view content.

- Do not assume the field will exist in all documents.

Fields may be missing from your document, or may only be supported in specific document types. Use an `if` test to identify problems. For example:

~~~~
```
if (document.firstname)...
```
~~~~

- View output is case sensitive.

The value emitted by the `emit()` function is case sensitive. Emitting a field value of ‘Martin’ but specifying a `key` value of ‘martin’ will not match the data. Emitted data, and the key selection values, should be normalized to eliminate potential problems. For example:

~~~~
```
emit(doc.firstname.toLowerCase(),null);
```
~~~~

- Number formatting

Numbers within JavaScript may inadvertently be converted and output as strings. To ensure that data is correctly formatted, the value should be explicitly converted. For example:

~~~~
```
emit(parseInt(doc.value,10),null);
```
~~~~

The `parseInt()` built-in function will convert a supplied value to an integer. The `parseFloat()` function can be used for floating-point numbers.

Validating document type

If your dataset includes documents that may be either JSON or binary, then you do not want to create a view that outputs individual fields for non-JSON documents. You can fix this by using a view that checks the metadata `type` field before outputting the JSON view information:

```
function(doc,meta) {
    if (meta.type == "json") {
        emit(doc.firstname.toLowerCase(),null);
    }
}
```

In the above example, the `emit()` function will only be called on a valid JSON document. Non-JSON documents will be ignored and not included in the view output.

Document ID (primary) index

To create a ‘primary key’ index, i.e. an index that contains a list of every document within the database, with the document ID as the key, you can create a simple view:

```
function(doc,meta)
{
  emit(meta.id,null);
}
```

This enables you to iterate over the documents stored in the database.

This will provide you with a view that outputs the document ID of every document in the bucket using the document ID as the key.

The view can be useful for obtaining groups or ranges of documents based on the document ID, for example to get documents with a specific ID prefix:

```
?startkey="object"&endkey="object\u0000"
```

Or to obtain a list of objects within a given range:

```
?startkey="object100"&endkey="object199"
```

For all views, the document ID is automatically included as part of the view response. But the without including the document ID within the key emitted by the view, it cannot be used as a search or querying mechanism.

Secondary index

The simplest form of view is to create an index against a single field from the documents stored in your database.

For example, given the document structure:

```
{
    "firstname": "Martin",
    "lastname": "Brown"
}
```

A view to support queries on the `firstname` field could be defined as follows:

```
function(doc, meta)
{
  if (doc.firstname)
  {
     emit(doc.firstname.toLowerCase(),null);
  }
}
```

The view works as follows for each document:

- Only outputs a record if the document contains a `firstname` field.
- Converts the content of the `firstname` field to lowercase.

Queries can now be specified by supplying a string converted to lowercase. For example:

```
?key="martin"
```

Will return all documents where the `firstname` field contains 'Martin', regardless of the document field capitalization.

Using expiration metadata

The metadata object makes it very easy to create and update different views on your data using information outside of the main document data. For example, you can use the expiration field within a view to get the list of recently active sessions in a system.

Using the following `map()` function, which uses the expiration as part of the emitted data.

```
function(doc, meta)
{
  if (doc.type && doc.type == "session")
  {
    emit(meta.expiration, doc.nickname)
```

```
  }
}
```

If you have sessions which are saved with a TTL, this will allow you to give a view of who was recently active on the service.

Emitting multiple rows

The `emit()` function is used to create a record of information for the view during the map phase, but it can be called multiple times within that map phase to allowing querying over more than one source of information from each stored document.

An example of this is when the source documents contain an array of information. For example, within a recipe document, the list of ingredients is exposed as an array of objects. By iterating over the ingredients, an index of ingredients can be created and then used to find recipes by ingredient.

```
{
    "title": "Fried chilli potatoes",
    "preptime": "5"
    "servings": "4",
    "totaltime": "10",
    "subtitle": "A new way with chips.",
    "cooktime": "5",
    "ingredients": [
        {
            "ingredtext": "chilli powder",
            "ingredient": "chilli powder",
            "meastext": "3-6 tsp"
        },
        {
            "ingredtext": "potatoes, peeled and cut into wedges",
            "ingredient": "potatoes",
            "meastext": "900 g"
        },
        {
            "ingredtext": "vegetable oil for deep frying",
            "ingredient": "vegetable oil for deep frying",
            "meastext": ""
        }
    ],
}
```

The view can be created using the following `map()` function:

```
function(doc, meta)
{
  if (doc.ingredients)
  {
    for (i=0; i < doc.ingredients.length; i++)
    {
        emit(doc.ingredients[i].ingredient, null);
    }
  }
}
```

To query for a specific ingredient, specify the ingredient as a key:

```
?key="carrot"
```

The `keys` parameter can also be used in this situation to look for recipes that contain multiple ingredients. For example, to look for recipes that contain either “potatoes” or “chilli powder” you would use:

```
?keys=["potatoes","chilli powder"]
```

This will produce a list of any document containing either ingredient. A simple count of the document IDs by the client can determine which recipes contain all three.

The output can also be combined. For example, to look for recipes that contain carrots and can be cooked in less than 20 minutes, the view can be rewritten as:

```
function(doc, meta)
{
  if (doc.ingredients)
  {
    for (i=0; i < doc.ingredients.length; i++)
    {
      if (doc.ingredients[i].ingredtext &amp;&amp; doc.totaltime)
      {
        emit([doc.ingredients[i].ingredtext, parseInt(doc.totaltime,10)],
 null);
      }
    }
  }
}
```

In this map function, an array is output that generates both the ingredient name, and the total cooking time for the recipe. To perform the original query, carrot recipes requiring less than 20 minutes to cook:

```
?startkey=["carrot",0]&endkey=["carrot",20]
```

This generates the following view:

```
{"total_rows":26471,"rows":[
{"id":"Mangoandcarrotsmoothie","key":["carrots",5],"value":null},
{"id":"Cheeseandapplecoleslaw","key":["carrots",15],"value":null}
]
}
```

Date and time selection

For date and time selection, consideration must be given to how the data will need to be selected when retrieving the information. This is particularly true when you want to perform log roll-up or statistical collection by using a reduce function to count or quantify instances of a particular event over time.

Examples of this in action include querying data over a specific range, on specific day or date combinations, or specific time periods. Within a traditional relational database it is possible to perform an extraction of a specific date or date range by storing the information in the table as a date type.

Within a map/reduce, the effect can be simulated by exposing the date into the individual components at the level of detail that you require. For example, to obtain a report that counts individual log types over a period identifiable to individual days, you can use the following `map()` function:

```
function(doc, meta) {
    emit([doc.year, doc.mon, doc.day, doc.logtype], null);
}
```

By incorporating the full date into the key, the view provides the ability to search for specific dates and specific ranges. By modifying the view content you can simplify this process further. For example, if only searches by year/month are required for a specific application, the day can be omitted.

And with the corresponding `reduce()` built-in of `_count`, you can perform a number of different queries. Without any form of data selection, for example, you can use the `group_level` parameter to summarize down as far as individual day, month, and year. Additionally, because the date is explicitly output, information can be selected over a specific range, such as a specific month:

```
endkey=[2010,9,30]&group_level=4&startkey=[2010,9,0]
```

Here the explicit date has been specified as the start and end key. The `group_level` is required to specify roll-up by the date and log type.

This will generate information similar to this:

```
{"rows":[
```

```
{"key":[2010,9,1,"error"],"value":5},
{"key":[2010,9,1,"warning"],"value":10},
{"key":[2010,9,2,"error"],"value":8},
{"key":[2010,9,2,"warning"],"value":9},
{"key":[2010,9,3,"error"],"value":16},
{"key":[2010,9,3,"warning"],"value":8},
{"key":[2010,9,4,"error"],"value":15},
{"key":[2010,9,4,"warning"],"value":11},
{"key":[2010,9,5,"error"],"value":6},
{"key":[2010,9,5,"warning"],"value":12}
]
}
```

Additional granularity, for example down to minutes or seconds, can be achieved by adding those as further arguments to the map function:

```
function(doc, meta)
{
    emit([doc.year, doc.mon, doc.day, doc.hour, doc.min, doc.logtype], null);
}
```

The same trick can also be used to output based on other criteria. For example, by day of the week, week number of the year or even by period:

```
function(doc, meta) {
  if (doc.mon)
  {
    var quarter = parseInt((doc.mon - 1)/3,10)+1;

    emit([doc.year, quarter, doc.logtype], null);
  }
}
```

To get more complex information, for example a count of individual log types for a given date, you can combine the `map()` and `reduce()` stages to provide the collation.

For example, by using the following `map()` function we can output and collate by day, month, or year as before, and with data selection at the date level.

```
function(doc, meta) {
    emit([doc.year, doc.mon, doc.day], doc.logtype);
}
```

For convenience, you may wish to use the `dateToArray()` function, which converts a date object or string into an array. For example, if the date has been stored within the document as a single field:

```
function(doc, meta) {
    emit(dateToArray(doc.date), doc.logtype);
}
```

For more information, see `dateToArray()`.

Using the following `reduce()` function, data can be collated for each individual logtype for each day within a single record of output.

```
function(key, values, rereduce)
{
  var response = {"warning" : 0, "error": 0, "fatal" : 0 };
  for(i=0; i<values.length; i++)
  {
    if (rereduce)
    {
      response.warning = response.warning + values[i].warning;
      response.error = response.error + values[i].error;
      response.fatal = response.fatal + values[i].fatal;
    }
    else
```

```
    {
      if (values[i] == "warning")
      {
        response.warning++;
      }
      if (values[i] == "error" )
      {
        response.error++;
      }
      if (values[i] == "fatal" )
      {
        response.fatal++;
      }
    }
  }
  return response;
}
```

When queried using a `group_level` of two (by month), the following output is produced:

```
{"rows":[
{"key":[2010,7], "value":{"warning":4,"error":2,"fatal":0}},
{"key":[2010,8], "value":{"warning":4,"error":3,"fatal":0}},
{"key":[2010,9], "value":{"warning":4,"error":6,"fatal":0}},
{"key":[2010,10],"value":{"warning":7,"error":6,"fatal":0}},
{"key":[2010,11],"value":{"warning":5,"error":8,"fatal":0}},
{"key":[2010,12],"value":{"warning":2,"error":2,"fatal":0}},
{"key":[2011,1], "value":{"warning":5,"error":1,"fatal":0}},
{"key":[2011,2], "value":{"warning":3,"error":5,"fatal":0}},
{"key":[2011,3], "value":{"warning":4,"error":4,"fatal":0}},
{"key":[2011,4], "value":{"warning":3,"error":6,"fatal":0}}
]
}
```

The input includes a count for each of the error types for each month. Note that because the key output includes the year, month and date, the view also supports explicit querying while still supporting grouping and roll-up across the specified group. For example, to show information from 15th November 2010 to 30th April 2011 using the following query:

```
?endkey=[2011,4,30]&group_level=2&startkey=[2010,11,15]
```

Which generates the following output:

```
{"rows":[
{"key":[2010,11],"value":{"warning":1,"error":8,"fatal":0}},
{"key":[2010,12],"value":{"warning":3,"error":4,"fatal":0}},
{"key":[2011,1],"value":{"warning":8,"error":2,"fatal":0}},
{"key":[2011,2],"value":{"warning":4,"error":7,"fatal":0}},
{"key":[2011,3],"value":{"warning":4,"error":4,"fatal":0}},
{"key":[2011,4],"value":{"warning":5,"error":7,"fatal":0}}
]
}
```

Keep in mind that you can create multiple views to provide different views and queries on your document data. In the above example, you could create individual views for the limited datatypes of logtype to create a `warningsbydate` view.

Selective record output

If you are storing different document types within the same bucket, then you may want to ensure that you generate views only on a specific record type within the `map()` phase. This can be achieved by using an `if` statement to select the record.

For example, if you are storing blog ‘posts’ and ‘comments’ within the same bucket, then a view on the blog posts could be created using the following map:

```
function(doc, meta) {
    if (doc.title && doc.type && doc.date &&
        doc.author && doc.type == 'post')
    {
        emit(doc.title, [doc.date, doc.author]);
    }
}
```

The same solution can also be used if you want to create a view over a specific range or value of documents while still allowing specific querying structures. For example, to filter all the records from the statistics logging system over a date range that are of the type error you could use the following `map()` function:

```
function(doc, meta) {
    if (doc.logtype == 'error')
    {
       emit([doc.year, doc.mon, doc.day],null);
    }
}
```

The same solution can also be used for specific complex query types. For example, all the recipes that can be cooked in under 30 minutes, made with a specific ingredient:

```
function(doc, meta)
{
  if (doc.totaltime &amp;&amp; doc.totaltime <= 20)
  {
    if (doc.ingredients) {
      for (i=0; i < doc.ingredients.length; i++)
      {
        if (doc.ingredients[i].ingredtext)
        {
          emit(doc.ingredients[i].ingredtext, null);
        }
      }
    }
  }
}
```

The above function allows for much quicker and simpler selection of recipes by using a query and the `key` parameter, instead of having to work out the range that may be required to select recipes when the cooking time and ingredients are generated by the view.

These selections are application specific, but by producing different views for a range of appropriate values, for example 30, 60, or 90 minutes, recipe selection can be much easier at the expense of updating additional view indexes.

Sorting on reduce values

The sorting algorithm within the view system outputs information ordered by the generated key within the view, and therefore it operates before any reduction takes place. Unfortunately, it is not possible to sort the output order of the view on computed reduce values, as there is no post-processing on the generated view information.

To sort based on reduce values, you must access the view content with reduction enabled from a client, and perform the sorting within the client application.

Solutions for simulating joins

Joins between data, even when the documents being examined are contained within the same bucket, are not possible directly within the view system. However, you can simulate this by making use of a common field used for linking when outputting the view information. For example, consider a blog post system that supports two different record types, ‘blogpost’ and ‘blogcomment’. The basic format for ‘blogpost’ is:

```
{
    "type" : "post",
```

```
    "title" : "Blog post"
    "categories" : [...],
    "author" : "Blog author"
    ...
}
```

The corresponding comment record includes the blog post ID within the document structure:

```
{
    "type" : "comment",
    "post_id" : "post_3454"
    "author" : "Comment author",
    "created_at" : 123498235
...
}
```

To output a blog post and all the comment records that relate to the blog post, you can use the following view:

```
function(doc, meta)
{
    if (doc.post_id && doc.type && doc.type == "post")
    {
        emit([doc.post_id, null], null);
    }
    else if (doc.post_id && doc.created_at && doc.type && doc.type ==
 "comment")
    {
        emit([doc.post_id, doc.created_at], null);
    }
}
```

The view makes use of the sorting algorithm when using arrays as the view key. For a blog post record, the document ID will be output will a null second value in the array, and the blog post record will therefore appear first in the sorted output from the view. For a comment record, the first value will be the blog post ID, which will cause it to be sorted in line with the corresponding parent post record, while the second value of the array is the date the comment was created, allowing sorting of the child comments.

For example:

```
{"rows":[
{"key":["post_219",null],       "value":{...}},
{"key":["post_219",1239875435],"value":{...}},
{"key":["post_219",1239875467],"value":{...}},
]
}
```

Another alternative is to make use of a multi-get operation within your client through the main Couchbase SDK interface, which should load the data from cache. This allows you to structure your data with the blog post containing an array of the of the child comment records. For example, the blog post structure might be:

```
{
    "type" : "post",
    "title" : "Blog post"
    "categories" : [...],
    "author" : "Blog author",
    "comments": ["comment_2298","comment_457","comment_4857"],
    ...
}
```

To obtain the blog post information and the corresponding comments, create a view to find the blog post record, and then make a second call within your client SDK to get all the comment records from the Couchbase Server cache.

Simulating transactions

Couchbase Server does not support transactions, but the effect can be simulated by writing a suitable document and view definition that produces the effect while still only requiring a single document update to be applied.

For example, consider a typical banking application, the document structure could be as follows:

```
{
   "account" : "James",
   "value" : 100
}
```

A corresponding record for another account:

```
{
   "account" : "Alice",
   "value" : 200
}
```

To get the balance of each account, the following `map()` :

```
function(doc, meta) {
    if (doc.account && doc.value)
    {
      emit(doc.account,doc.value);
    }
}
```

The `reduce()` function can use the built-in `_sum` function.

When queried, using a `group_level` of 1, the balance of the accounts is displayed:

```
{"rows":[
{"key":"Alice","value":200},
{"key":"James","value":100}
]
}
```

Money in an account can be updated just by adding another record into the system with the account name and value. For example, adding the record:

```
{
   "account" : "James",
   "value" : 50
}
```

Re-querying the view produces an updated balance for each account:

```
{"rows":[
{"key":"Alice","value":200},
{"key":"James","value":150}
]
}
```

However, if Alice wants to transfer $100 to James, two record updates are required:

1. A record that records an update to Alice's account to reduce the value by 100.
2. A record that records an update to James's account to increase the value by 100.

Unfortunately, the integrity of the transaction could be compromised in the event of a problem between step 1 and step 2. Alice's account may be deducted, without updates James' record.

To simulate this operation while creating (or updating) only one record, a combination of a transaction record and a view must be used. The transaction record looks like this:

```
{
     "fromacct" : "Alice",
     "toacct" : "James",
     "value" : 100
}
```

The above records the movement of money from one account to another. The view can now be updated to handle a transaction record and output a row through `emit()` to update the value for each account.

```
function(doc, meta)
{
  if (doc.fromacct)
  {
    emit(doc.fromacct, -doc.value);
    emit(doc.toacct, doc.value);
  }
  else
  {
    emit(doc.account, doc.value);
  }
}
```

The above `map()` effectively generates two fake rows, one row subtracts the amount from the source account, and adds the amount to the destination account. The resulting view then uses the `reduce()` function to sum up the transaction records for each account to arrive at a final balance:

```
{"rows":[
{"key":"Alice","value":100},
{"key":"James","value":250}
]
}
```

Throughout the process, only one record has been created, and therefore transient problems with that record update can be captured without corrupting or upsetting the existing stored data.

Simulating multi-phase transactions

The technique in Simulating Transactions works if your data will allow the use of a view to effectively roll-up the changes into a single operation. However, if your data and document structure do not allow it then you can use a multi-phase transaction process to perform the operation in a number of distinct stages.

This method is not reliant on views, but the document structure and update make it easy to find out if there are 'hanging' or trailing transactions that need to be processed without additional document updates. Using views and the Observe operation to monitor changes could lead to long wait times during the transaction process while the view index is updated.

To employ this method, you use a similar transaction record as in the previous example, but use the transaction record to record each stage of the update process.

Start with the same two account records:

```
{
   "type" : "account",
   "account" : "James",
   "value" : 100,
   "transactions" : []
}
```

The record explicitly contains a `transactions` field which contains an array of all the currently active transactions on this record.

The corresponding record for the other account:

```
{
   "type" : "account",
   "account" : "Alice",
   "value" : 200,
   "transactions" : []
}
```

Now perform the following operations in sequence:

1. Create a new transaction record that records the transaction information:

```
{ "type" : "transaction", "fromacct" : "Alice", "toacct" : "James",
"value" : 100, "status" : "waiting" }
```

The core of the transaction record is the same, the difference is the use of a `status` field which will be used to monitor the progress of the transaction.

Record the ID of the transaction, for example, `transact_20120717163`.

2. Set the value of the `status` field in the transaction document to 'pending':

```
{ "type" : "transaction", "fromacct" : "Alice", "toacct" : "James",
"value" : 100, "status" : "pending" }
```

3. Find all transaction records in the `pending` state using a suitable view:

```
function(doc, meta) { if (doc.type && doc.status && doc.type ==
"transaction" && doc.status == "pending" ) { emit([doc.fromacct,doc.toacct],
doc.value); } }
```

4. Update the record identified in `toacct` with the transaction information, ensuring that the transaction is not already pending:

```
{ "type" : "account", "account" : "Alice", "value" : 100, "transactions" :
["transact_20120717163"] }
```

Repeat on the other account:

```
{ "type" : "account", "account" : "James", "value" : 200, "transactions" :
["transact_20120717163"] }
```

5. Update the transaction record to mark that the records have been updated:

```
{ "type" : "transaction", "fromacct" : "Alice", "toacct" : "James",
"value" : 100, "status" : "committed" }
```

6. Find all transaction records in the `committed` state using a suitable view:

```
function(doc, meta) { if (doc.type && doc.status && doc.type ==
"transaction" && doc.status == "committed" ) { emit([doc.fromacct,
doc.toacct], doc.value); } }
```

Update the source account record noted in the transaction and remove the transaction ID:

```
{ "type" : "account", "account" : "Alice", "value" : 100, "transactions" :
[] }
```

Repeat on the other account:

```
{ "type" : "account", "account" : "James", "value" : 200, "transactions" :
[] }
```

7. Update the transaction record state to 'done'. This will remove the transaction from the two views used to identify unapplied, or uncommitted transactions.

Within this process, although there are multiple steps required, you can identify at each step whether a particular operation has taken place or not.

For example, if the transaction record is marked as 'pending', but the corresponding account records do not contain the transaction ID, then the record still needs to be updated. Since the account record can be updated using a single atomic operation, it is easy to determine if the record has been updated or not.

The result is that any sweep process that accesses the views defined in each step can determine whether the record needs updating. Equally, if an operation fails, a record of the transaction, and whether the update operation has been applied, also exists, allowing the changes to be reversed and backed out.

Related Links

Views and indexes on page 183

Translating SQL to map/reduce

```
SELECT fieldlist FROM table \
    WHERE condition \
    GROUP BY groupfield \
    ORDER BY orderfield \
    LIMIT limitcount OFFSET offsetcount
```

The different elements within the source statement affect how a view is written in the following ways:

- `SELECT fieldlist`

The field list within the SQL statement affects either the corresponding key or value within the `map()` function, depending on whether you are also selecting or reducing your data.

- `FROM table`

There are no table compartments within Couchbase Server and you cannot perform views across more than one bucket boundary. However, if you are using a `type` field within your documents to identify different record types, then you may want to use the `map()` function to make a selection.

- `WHERE condition`

The `map()` function and the data generated into the view key directly affect how you can query, and therefore how selection of records takes place.

- `ORDER BY orderfield`

The order of record output within a view is directly controlled by the key specified during the `map()` function phase of the view generation.

- `LIMIT limitcount OFFSET offsetcount`

There are a number of different paging strategies available within the map/reduce and views mechanism.

- `GROUP BY groupfield`

Grouping within SQL is handled within views through the use of the `reduce()` function.

The interaction between the view `map()` function, `reduce()` function, selection parameters and other miscellaneous parameters according to the table below:

SQL Statement Fragment	View Key	View Value	`map()` Function	`reduce()` Function	Selection Parameters	Other Parameters
SELECT fields	Yes	Yes	Yes	No: with `GROUP BY` and `SUM()` or `COUNT()` functions only	No	No
FROM table	No	No	Yes	No	No	No
WHERE clause	Yes	No	Yes	No	Yes	No
ORDER BY field	Yes	No	Yes	No	No	`descending`
LIMIT x OFFSET y	No	No	No	No	No	`limit`, `skip`
GROUP BY field	Yes	Yes	Yes	Yes	No	No

Within SQL, the basic query structure can be used for a multitude of different queries. For example, the same ' `SELECT fieldlist FROM table WHERE xxxx` can be used with a number of different clauses.

Within map/reduce and Couchbase Server, multiple views may be needed to be created to handled different query types. For example, performing a query on all the blog posts on a specific date will need a very different view definition than one needed to support selection by the author.

Translating SQL SELECT to map/reduce

The field selection within an SQL query can be translated into a corresponding view definition, either by adding the fields to the emitted key (if the value is also used for selection in a `WHERE` clause), or into the emitted value, if the data is separate from the required query parameters.

For example, to get the sales data by country from each stored document using the following `map()` function:

```
function(doc, meta) {
  emit([doc.city, doc.sales], null);
}
```

If you want to output information that can be used within a reduce function, this should be specified in the value generated by each `emit()` call. For example, to reduce the sales figures the above `map()` function could be rewritten as:

```
function(doc, meta) {
  emit(doc.city, doc.sales);
}
```

In essence this does not produce significantly different output (albeit with a simplified key), but the information can now be reduced using the numerical value.

If you want to output data or field values completely separate to the query values, then these fields can be explicitly output within the value portion of the view. For example:

```
function(doc, meta) {
  emit(doc.city, [doc.name, doc.sales]);
}
```

If the entire document for each item is required, load the document data after the view has been requested through the client library. For more information on this parameter and the performance impact.

Within a `SELECT` statement it is common practice to include the primary key for a given record in the output. Within a view this is not normally required, since the document ID that generated each row is always included within the view output.

Translating SQL WHERE to map/reduce

The `WHERE` clause within an SQL statement forms the selection criteria for choosing individual records. Within a view, the ability to query the data is controlled by the content and structure of the `key` generated by the `map()` function.

In general, for each `WHERE` clause you need to include the corresponding field in the key of the generated view, and then use the `key`, `keys` or `startkey` / `endkey` combinations to indicate the data you want to select.. The complexity occurs when you need to perform queries on multiple fields. There are a number of different strategies that you can use for this.

The simplest way is to decide whether you want to be able to select a specific combination, or whether you want to perform range or multiple selections. For example, using our recipe database, if you want to select recipes that use the ingredient 'carrot' and have a cooking time of exactly 20 minutes, then you can specify these two fields in the `map()` function:

```
function(doc, meta)
{
  if (doc.ingredients)
  {
    for(i=0; i < doc.ingredients.length; i++)
```

```
    {
      emit([doc.ingredients[i].ingredient, doc.totaltime], null);
    }
  }
}
```

Then the query is an array of the two selection values:

```
?key=["carrot",20]
```

This is equivalent to the SQL query:

```
SELECT recipeid FROM recipe JOIN ingredients on ingredients.recipeid =
 recipe.recipeid
    WHERE ingredient = 'carrot' AND totaltime = 20
```

If, however, you want to perform a query that selects recipes containing carrots that can be prepared in less than 20 minutes, a range query is possible with the same `map()` function:

```
?startkey=["carrot",0]&endkey=["carrot",20]
```

This works because of the sorting mechanism in a view, which outputs in the information sequentially, fortunately nicely sorted with carrots first and a sequential number.

More complex queries though are more difficult. What if you want to select recipes with carrots and rice, still preparable in under 20 minutes?

A standard `map()` function like that above wont work. A range query on both ingredients will list all the ingredients between the two. There are a number of solutions available to you. First, the easiest way to handle the timing selection is to create a view that explicitly selects recipes prepared within the specified time. I.E:

```
function(doc, meta)
{
  if (doc.totaltime <= 20)
  {
    ...
  }
}
```

Although this approach seems to severely limit your queries, remember you can create multiple views, so you could create one for 10 mins, one for 20, one for 30, or whatever intervals you select. It’s unlikely that anyone will really want to select recipes that can be prepared in 17 minutes, so such granular selection is overkill.

The multiple ingredients is more difficult to solve. One way is to use the client to perform two queries and merge the data. For example, the `map()` function:

```
function(doc, meta)
{
  if (doc.totaltime &amp;&amp; doc.totaltime <= 20)
  {
    if (doc.ingredients)
    {
      for(i=0; i < doc.ingredients.length; i++)
      {
        emit(doc.ingredients[i].ingredient, null);
      }
    }
  }
}
```

Two queries, one for each ingredient can easily be merged by performing a comparison and count on the document ID output by each view.

The alternative is to output the ingredients twice within a nested loop, like this:

```
function(doc, meta)
{
  if (doc.totaltime &amp;&amp; doc.totaltime <= 20)
```

```
  {
    if (doc.ingredients)
    {
      for (i=0; i < doc.ingredients.length; i++)
      {
        for (j=0; j < doc.ingredients.length; j++)
        {
          emit([doc.ingredients[i].ingredient, doc.ingredients[j].ingredient],
 null);
        }
      }
    }
  }
}
```

Now you can perform an explicit query on both ingredients:

```
?key=["carrot","rice"]
```

If you really want to support flexible cooking times, then you can also add the cooking time:

```
function(doc, meta)
{
  if (doc.ingredients)
  {
    for (i=0; i < doc.ingredients.length; i++)
    {
      for (j=0; j < doc.ingredients.length; j++)
      {
        emit([doc.ingredients[i].ingredient, doc.ingredients[j].ingredient,
 recipe.totaltime], null);
      }
    }
  }
}
```

And now you can support a ranged query on the cooking time with the two ingredient selection:

```
?startkey=["carrot","rice",0]&key=["carrot","rice",20]
```

This would be equivalent to:

```
SELECT recipeid FROM recipe JOIN ingredients on ingredients.recipeid =
 recipe.recipeid
    WHERE (ingredient = 'carrot' OR ingredient = 'rice') AND totaltime = 20
```

Translating SQL ORDER BY to map/reduce

The `ORDER BY` clause within SQL controls the order of the records that are output. Ordering within a view is controlled by the value of the key. However, the key also controls and supports the querying mechanism.

In `SELECT` statements where there is no explicit `WHERE` clause, the emitted key can entirely support the sorting you want. For example, to sort by the city and salesman name, the following `map()` will achieve the required sorting:

```
function(doc, meta)
{
   emit([doc.city, doc.name], null)
}
```

If you need to query on a value, and that query specification is part of the order sequence then you can use the format above. For example, if the query basis is city, then you can extract all the records for ‘London’ using the above view and a suitable range query:

```
?endkey=["London\u0fff"]&startkey=["London"]
```

However, if you want to query the view by the salesman name, you need to reverse the field order in the `emit()` statement:

```
function(doc, meta)
{
   emit([doc.name,doc.city],null)
}
```

Now you can search for a name while still getting the information in city order.

The order the output can be reversed (equivalent to `ORDER BY field DESC`) by using the `descending` query parameter.

Translating SQL GROUP BY to map/reduce

The `GROUP BY` parameter within SQL provides summary information for a group of matching records according to the specified fields, often for use with a numeric field for a sum or total value, or count operation.

For example:

```
SELECT name,city,SUM(sales) FROM sales GROUP BY name,city
```

This query groups the information by the two fields 'name' and 'city' and produces a sum total of these values. To translate this into a map/reduce function within Couchbase Server:

- From the list of selected fields, identify the field used for the calculation. These will need to be exposed within the value emitted by the `map()` function.
- Identify the list of fields in the `GROUP BY` clause. These will need to be output within the key of the `map()` function.
- Identify the grouping function, for example `SUM()` or `COUNT()`. You will need to use the equivalent built-in function, or a custom function, within the `reduce()` function of the view.

For example, in the above case, the corresponding map function can be written as `map()`:

```
function(doc, meta)
{
   emit([doc.name,doc.city],doc.sales);
}
```

This outputs the name and city as the key, and the sales as the value. Because the `SUM()` function is used, the built-in `reduce()` function `_sum` can be used.

An example of this map/reduce combination can be seen `_sum`.

More complex grouping operations may require a custom reduce function.

Translating SQL LIMIT and OFFSET

Within SQL, the `LIMIT` and `OFFSET` clauses to a given query are used as a paging mechanism. For example, you might use:

```
SELECT recipeid,title FROM recipes LIMIT 100
```

To get the first 100 rows from the database, and then use the `OFFSET` to get the subsequent groups of records:

```
SELECT recipeid,title FROM recipes LIMIT 100 OFFSET 100
```

With Couchbase Server, the `limit` and `skip` parameters when supplied to the query provide the same basic functionality:

```
?limit=100&skip=100
```

Performance for high values of skip can be affected.

Related Links

Views and indexes on page 183
View and query pattern samples on page 222
Querying views on page 213
Writing views on page 196

Writing geospatial views

Geospatial support was introduced as an *experimental* feature in Couchbase Server. This feature is currently unsupported and is provided only for the purposes of demonstration and testing.

GeoCouch adds two-dimensional spatial index support to Couchbase. Spatial support enables you to record geometry data into the bucket and then perform queries which return information based on whether the recorded geometries existing within a given two-dimensional range such as a bounding box. This can be used in spatial queries and in particular geolocationary queries where you want to find entries based on your location or region.

The GeoCouch support is provided through updated index support and modifications to the view engine to provide advanced geospatial queries.

Adding geometry data

GeoCouch supports the storage of any geometry information using the GeoJSON specification. The format of the storage of the point data is arbitrary with the geometry type being supported during the view index generation.

For example, you can use two-dimensional geometries for storing simple location data. You can add these to your Couchbase documents using any field name. The convention is to use a single field with two-element array with the point location, but you can also use two separate fields or compound structures as it is the view that compiles the information into the geospatial index.

For example, to populate a bucket with city location information, the document sent to the bucket could be formatted like that below:

```
{
"loc" : [-122.270833, 37.804444],
"title" : "Oakland"
}
```

Views and queries

The GeoCouch extension uses the standard Couchbase indexing system to build a two-dimensional index from the point data within the bucket. The format of the index information is based on the GeoJSON specification.

To create a geospatial index, use the `emit()` function to output a GeoJSON Point value containing the coordinates of the point you are describing. For example, the following function will create a geospatial index on the earlier spatial record example.

```
function(doc, meta)
{
  if (doc.loc)
  {
     emit(
          {
             type: "Point",
             coordinates: doc.loc,
          },
          [meta.id, doc.loc]);
  }
}
```

The key in the spatial view index can be any valid GeoJSON geometry value, including points, multipoints, linestrings, polygons and geometry collections.

The view `map()` function should be placed into a design document using the `spatial` prefix to indicate the nature of the view definition. For example, the following design document includes the above function as the view `points`

```
{
   "spatial" : {
      "points" : "function(doc, meta) { if (doc.loc) { emit({ type: \"Point\",
 coordinates: [doc.loc[0], doc.loc[1]]}, [meta.id, doc.loc]);}}",
   }
```

```
}
```

To execute the geospatial query you use the design document format using the embedded spatial indexing. For example, if the design document is called `main` within the bucket `places`, the URL will be `http://localhost:8092/places/_design/main/_spatial/points`.

Spatial queries include support for a number of additional arguments to the view request. The full list is provided in the following summary table.

Get Spatial Name	Description
Method	`GET /bucket/_design/design-doc/_spatial/spatial-name`
Request Data	None
Response Data	JSON of the documents returned by the view
Authentication Required	no
	Query Arguments
`bbox`	Specify the bounding box for a spatial query
	Parameters : string; optional
`limit`	Limit the number of the returned documents to the specified number
	Parameters : numeric; optional
`skip`	Skip this number of records before starting to return the results
	Parameters : numeric; optional
`stale`	Allow the results from a stale view to be used
	Parameters : string; optional
	Supported Values
	`false` : Force update of the view index before results are returned
	`ok` : Allow stale views
	`update_after` : Allow stale view, update view after access

Bounding Box Queries If you do not supply a bounding box, the full dataset is returned. When querying a spatial index you can use the bounding box to specify the boundaries of the query lookup on a given value. The specification should be in the form of a comma-separated list of the coordinates to use during the query.

These coordinates are specified using the GeoJSON format, so the first two numbers are the lower left coordinates, and the last two numbers are the upper right coordinates.

For example, using the above design document:

```
GET http://localhost:8092/places/_design/main/_spatial/points?bbox=0,0,180,90
Content-Type: application/json
```

Returns the following information:

```
{
   "update_seq" : 3,
   "rows" : [
      {
         "value" : [
```

```
            "oakland",
            [
               10.898333,
               48.371667
            ]
         ],
         "bbox" : [
            10.898333,
            48.371667,
            10.898333,
            48.371667
         ],
         "id" : "augsburg"
      }
   ]
}
```

Note: The return data includes the value specified in the design document view function, and the bounding box of each individual matching document. If the spatial index includes the `bbox` bounding box property as part of the specification, then this information will be output in place of the automatically calculated version.

Related Links

Views and indexes on page 183

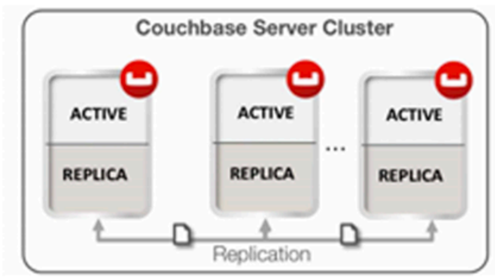
Related topics

GeoJSON

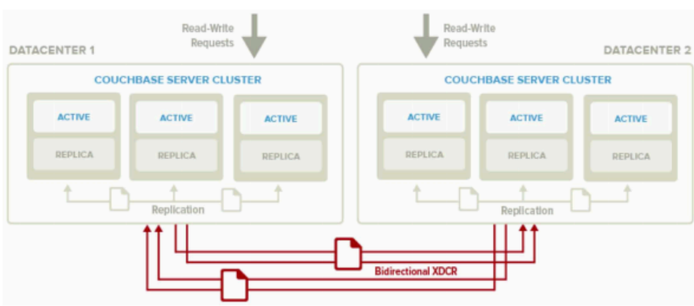
Cross Datacenter Replication (XDCR)

Couchbase Server supports cross datacenter replication (XDCR), providing an easy way to replicate data from one cluster to another for disaster recovery as well as better data locality (getting data closer to its users).

Couchbase Server provides support for both intra-cluster replication and cross datacenter replication (XDCR). Intra-cluster replication is the process of replicating data on multiple servers within a cluster in order to provide data redundancy should one or more servers crash. Data in Couchbase Server is distributed uniformly across all the servers in a cluster, with each server holding active and replica documents. When a new document is added to Couchbase Server, in addition to being persisted, it is also replicated to other servers within the cluster (this is configurable up to three replicas). If a server goes down, failover promotes replica data to active:



Cross datacenter replication in Couchbase Server involves replicating active data to multiple, geographically diverse datacenters either for disaster recovery or to bring data closer to its users for faster data access, as shown in below:



You can also see that XDCR and intra-cluster replication occurs simultaneously. Intra-cluster replication is taking place within the clusters at both Datacenter 1 and Datacenter 2, while at the same time XDCR is replicating documents across datacenters. Both datacenters are serving read and write requests from the application.

Related Links

Couchbase Administration on page 89

XDCR use cases on page 242

XDCR architecture on page 242

XDCR basic topologies on page 243

XDCR advanced topologies on page 244

XDCR replication via memcached protocol on page 245
XDCR can replicate data through the memcached protocol at a destination cluster.

XDCR and network or system outages on page 246
XDCR is resilient to intermittent network failures.

XDCR document handling on page 246
XDCR does not replicate views and view indexes.

XDCR flush requests on page 247
Flush requests to delete the entire contents of bucket are not replicated to the remote cluster.

XDCR stream management on page 247
New XDCR stream creation must occur a period of time after creating a bucket or after deleting a XDCR stream.

XDCR data encryption on page 247
The cross data center (XDCR) data security feature provides secure cross data center replication using Secure Socket Layer (SSL) data encryption.

XDCR use cases

Disaster Recovery. Disaster can strike your datacenter at any time – often with little or no warning. With active-active cross datacenter replication in Couchbase Server, applications can read and write to any geo-location ensuring availability of data 24x365 even if an entire datacenter goes down.

Bringing Data Closer to Users. Interactive web applications demand low latency response times to deliver an awesome application experience. The best way to reduce latency is to bring relevant data closer to the user. For example, in online advertising, sub-millisecond latency is needed to make optimized decisions about real-time ad placements. XDCR can be used to bring post-processed user profile data closer to the user for low latency data access.

Data Replication for Development and Test Needs. Developers and testers often need to simulate production-like environments for troubleshooting or to produce a more reliable test. By using cross datacenter replication, you can create test clusters that host subset of your production data so that you can test code changes without interrupting production processing or risking data loss.

Related Links

Cross Datacenter Replication (XDCR) on page 241

XDCR architecture

There are a number of key elements in Couchbase Server's XDCR architecture including:

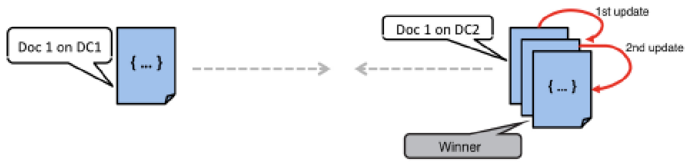
Continuous Replication. XDCR in Couchbase Server provides continuous replication across geographically distributed datacenters. Data mutations are replicated to the destination cluster after they are written to disk. There are multiple data streams (32 by default) that are shuffled across all shards (called vBuckets in Couchbase Server) on the source cluster to move data in parallel to the destination cluster. The vBucket list is shuffled so that replication is evenly load balanced across all the servers in the cluster. The clusters scale horizontally, more the servers, more the replication streams, faster the replication rate.

Cluster Aware. XDCR is cluster topology aware. The source and destination clusters could have different number of servers. If a server in the source or destination cluster goes down, XDCR is able to get the updated cluster topology information and continue replicating data to available servers in the destination cluster.

Push based connection resilient replication. XDCR in Couchbase Server is push-based replication. The source cluster regularly checkpoints the replication queue per vBucket and keeps track of what data the destination cluster last received. If the replication process is interrupted for example due to a server crash or intermittent network connection failures, it is not required to restart replication from the beginning. Instead, once the replication link is restored, replication can continue from the last checkpoint seen by the destination cluster.

Efficient. For the sake of efficiency, Couchbase Server is able to de-duplicate information that is waiting to be stored on disk. For instance, if there are three changes to the same document in Couchbase Server, and these three changes are waiting in queue to be persisted, only the last version of the document is stored on disk and later gets pushed into the XDCR queue to be replicated.

Active-Active Conflict Resolution. Within a cluster, Couchbase Server provides strong consistency at the document level. On the other hand, XDCR also provides eventual consistency across clusters. Built-in conflict resolution will pick the same "winner" on both the clusters if the same document was mutated on both the clusters. If a conflict occurs, the document with the most updates will be considered the "winner." If the same document is updated the same number of times on the source and destination, additional metadata such as numerical sequence, CAS value, document flags and expiration TTL value are used to pick the "winner." XDCR applies the same rule across clusters to make sure document consistency is maintained:



As shown in above, bidirectional replication is set up between Datacenter 1 and Datacenter 2 and both the clusters start off with the same JSON document (Doc 1). In addition, two additional updates to Doc 1 happen on Datacenter 2. In the case of a conflict, Doc 1 on Datacenter 2 is chosen as the winner because it has seen more updates.

Related Links

Cross Datacenter Replication (XDCR) on page 241

Related topics

XDCR REST API on page 483

Cross Datacenter Replication (XDCR) enables you to automatically replicate data between clusters and between data buckets.

XDCR basic topologies

XDCR can be configured to support a variety of different topologies; the most common are unidirectional and bidirectional.

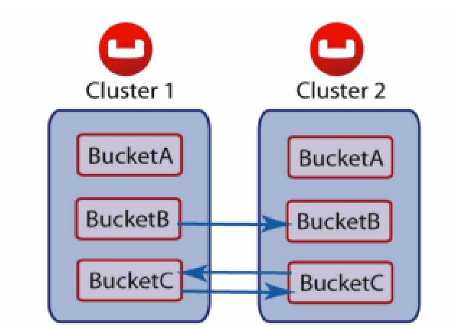
Unidirectional Replication is one-way replication, where active data gets replicated from the source cluster to the destination cluster. You may use unidirectional replication when you want to create an active offsite backup, replicating data from one cluster to a backup cluster.

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.

In both topologies, data changes on the source cluster are replicated to the destination cluster only after they are persisted to disk. You can also have more than two datacenters and replicate data between all of them.

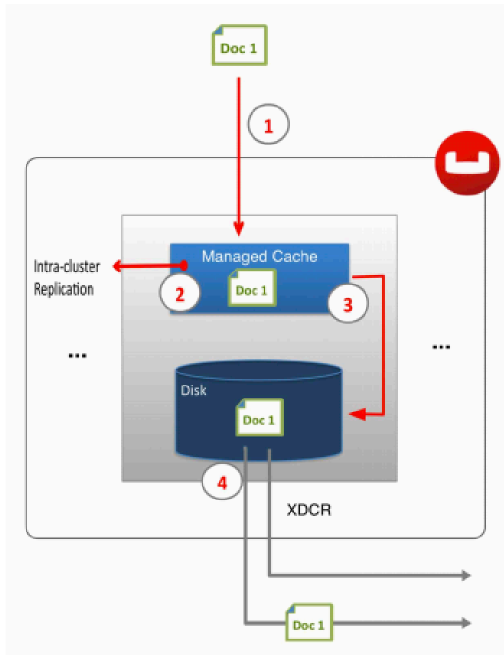
XDCR can be setup on a per bucket basis. A bucket is a logical container for documents in Couchbase Server. Depending on your application requirements, you might want to replicate only a subset of the data in Couchbase Server between two clusters. With XDCR you can selectively pick which buckets to replicate between two clusters in a unidirectional or bidirectional fashion. As shown in Figure 3, there is no XDCR between Bucket A (Cluster 1) and Bucket A (Cluster 2). Unidirectional XDCR is setup between Bucket B (Cluster 1) and Bucket B (Cluster 2). There is bidirectional XDCR between Bucket C (Cluster 1) and Bucket C (Cluster 2):

Cross datacenter replication in Couchbase Server involves replicating active data to multiple, geographically diverse datacenters either for disaster recovery or to bring data closer to its users for faster data access, as shown in below:



As shown above, after the document is stored in Couchbase Server and before XDCR replicates a document to other datacenters, a couple of things happen within each Couchbase Server node.

1. Each server in a Couchbase cluster has a managed cache. When an application stores a document in Couchbase Server it is written into the managed cache.
2. The document is added into the intra-cluster replication queue to be replicated to other servers within the cluster.
3. The document is added into the disk write queue to be asynchronously persisted to disk. The document is persisted to disk after the disk-write queue is flushed.
4. After the documents are persisted to disk, XDCR pushes the replica documents to other clusters. On the destination cluster, replica documents received will be stored in cache. This means that replica data on the destination cluster can undergo low latency read/write operations:

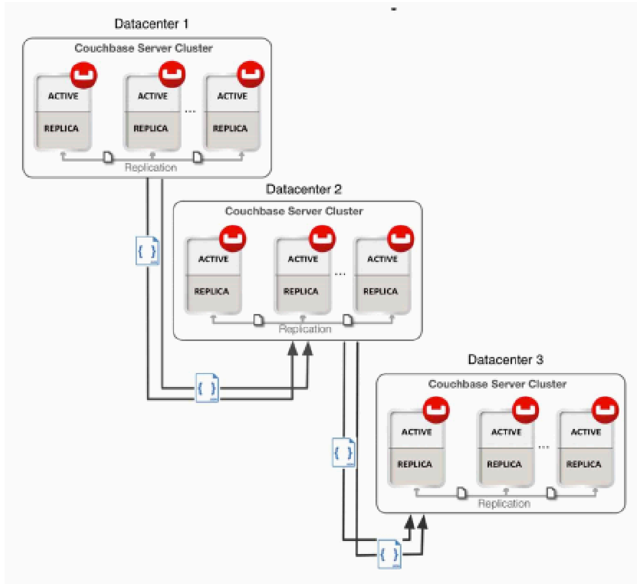


Related Links

Cross Datacenter Replication (XDCR) on page 241

XDCR advanced topologies

By combining unidirectional and bidirectional topologies, you have the flexibility to create several complex topologies such as the chain and propagation topology.



In the image below there is one bidirectional replication link between Datacenter 1 and Datacenter 2 and two unidirectional replication links between Datacenter 2 and Datacenters 3 and 4. Propagation replication can be useful in a scenario when you want to setup a replication scheme between two regional offices and several other local offices. Data between the regional offices is replicated bidirectionally between Datacenter 1 and Datacenter 2. Data changes in the local offices (Datacenters 3 and 4) are pushed to the regional office using unidirectional replication:

Related Links

Cross Datacenter Replication (XDCR) on page 241

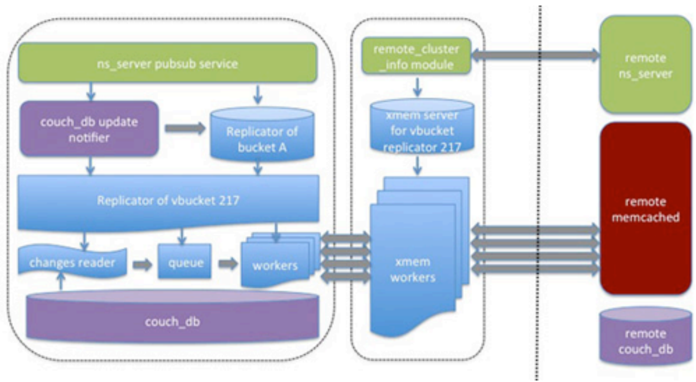
XDCR replication via memcached protocol

XDCR can replicate data through the memcached protocol at a destination cluster.

This mode utilizes highly efficient memcached protocol on the destination cluster for replicating changes. The new mode of XDCR increases XDCR throughput, reducing the CPU usage at destination cluster and also improves XDCR scalability.

In earlier versions of Couchbase Server only the REST protocol could be used for replication. On a source cluster a work process batched multiple mutations and sent the batch to a destination cluster using a REST interface. The REST interface at the destination node unpacked the batch of mutations and sent each mutation via a single memcached command. The destination cluster then stored mutations in RAM. This process is known as *CAPI mode XDCR* as it relies on the REST API known as CAPI.

This second mode available for XDCR is known as *XMEM mode XDCR* which bypasses the REST interface and replicates mutations via the memcached protocol at the destination cluster:



In this mode, every replication process at a source cluster delivers mutations directly via the memcached protocol on the remote cluster. This additional mode does not impact current XDCR architecture, rather it is implemented completely within the data communication layer used in XDCR. Any external XDCR interface remains the same. The benefit of using this mode is performance by increasing XDCR throughput, improving XDCR scalability, and reducing CPU usage at destination clusters during replication.

XDCR can be configured to operate via the new XMEM mode, which is the default or with CAPI mode. To change the replication mode, change the setting for `xdcr_replication_mode` via the Web Console or REST API.

Related Links

Cross Datacenter Replication (XDCR) on page 241

Related topics

XDCR REST API on page 483
Cross Datacenter Replication (XDCR) enables you to automatically replicate data between clusters and between data buckets.

XDCR and network or system outages

XDCR is resilient to intermittent network failures.

In the event that the destination cluster is unavailable due to a network interruption, XDCR pauses replication and then retries the connection to the cluster every 30 seconds. Once XDCR can successfully reconnect with a destination cluster, it resumes replication. In the event of a more prolonged network failure where the destination cluster is unavailable for more than 30 seconds, a source cluster continues polling the destination cluster which may result in numerous errors over time.

Related Links

Cross Datacenter Replication (XDCR) on page 241

XDCR document handling

XDCR does not replicate views and view indexes.

To replicate views and view indexes, manually exchange view definitions between clusters and re-generate the index on the destination cluster.

Non UTF–8 encodable document IDs on the source cluster are automatically filtered out and logged. The IDs are not transferred to the remote cluster. If there are any non UTF–8 keys, the warning output, `xdcr_error.*` displays in the log files along with a list of all non-UTF–8 keys found by XDCR.

Related Links

Cross Datacenter Replication (XDCR) on page 241

XDCR flush requests

Flush requests to delete the entire contents of bucket are not replicated to the remote cluster.

Performing a flush operation will only delete data on the local cluster. Flush is disabled if there is an active outbound replica stream configured.

Important: When replicating to or from a bucket, do not flush that bucket on the source or destination cluster. Flushing causes the vBucket state becomes temporarily unaccessible and results in a "not_found" error. The error suspends replication.

Related Links

Cross Datacenter Replication (XDCR) on page 241

XDCR stream management

New XDCR stream creation must occur a period of time after creating a bucket or after deleting a XDCR stream.

XDCR stream management Under the following circumstances, a period of time should pass (depending on the CPU load) before creating new XDCR streams:

- After creating a bucket
- After deleting an old XDCR stream

If a new XDCR stream is created immediately after a bucket has been created, a `db_not_found` error may occur. When a bucket is created, a period of time passes before the buckets are available. If XDCR tries to replicate to or from the vBucket too soon, a `db_not_found` error occurs. The same situation applies when other clients are “talking” to a bucket.

If a new XDCR stream is created immediately after an old XDCR stream is deleted, an Erlang `eaddrinuse` error occurs. This is related to the Erlang implementation of the TCP/IP protocol. After an Erlang process releases a socket, the socket stays in TIME_WAIT for a while before a new Erlang process can reuse it. If the new XDCR stream is created too quickly, vBucket replicators may encounter the eaddrinuse error and XDCR may not be able to fully start.

Note: The TIME_WAIT interval may be tunable from the operating system. If so, try lowering the interval time.

Related Links

Cross Datacenter Replication (XDCR) on page 241

XDCR data encryption

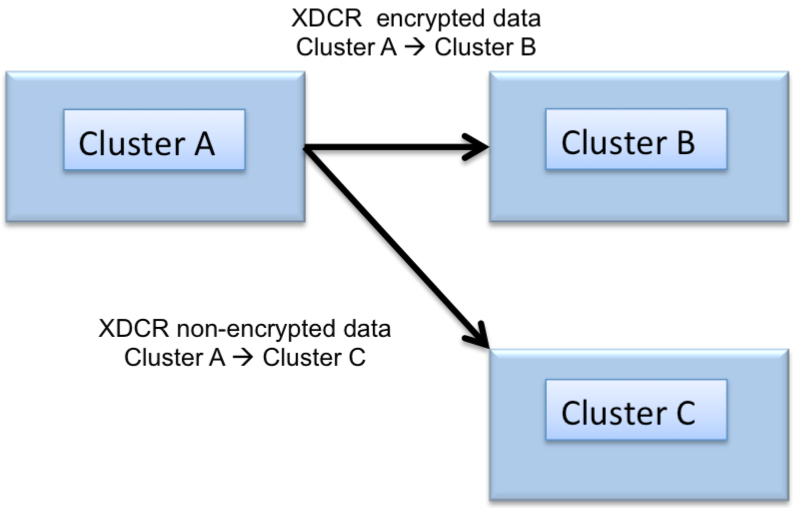
The cross data center (XDCR) data security feature provides secure cross data center replication using Secure Socket Layer (SSL) data encryption.

The data replicated between clusters can be encrypted in both uni-directional and bi-directional topologies.

By default, XDCR traffic to a destination cluster is sent in clear text that is unencrypted. In this case, when XDCR traffic occurs across multiple clusters over public networks, it is recommended that a VPN gateway be configured between the two data centers to encrypt the data between each route.

With the XDCR data encryption feature, the XDCR traffic from the source cluster is secured by enabling the XDCR encryption option, providing the destination cluster’s certificate, and then replicating. The certificate is a self-signed certificate used by SSL to initiate secure sessions.

Data encryption is established between the source and destination clusters. Since data encryption is established at the cluster level, all buckets that are selected for replicated on the destination cluster are data encrypted. For buckets that need to be replicated without data encryption, establish a second XDCR destination cluster without XDCR data encryption enabled.



Important: Both data encrypted and non-encrypted replication can not occur between the same XDCR source and destination cluster. For example, if Cluster A (source) has data encryption enabled to Cluster B (destination), then Cluster A (source) cannot also have non-encryption (data encryption is not enabled) to Cluster B (destination).

Related Links

Cross Datacenter Replication (XDCR) on page 241

Related topics

Managing XDCR data encryption on page 289

Couchbase web console

The Couchbase web console is the main tool for managing the Couchbase environment.

The web console provides the following tabs:

- Cluster Overview - A quick guide to the status of your Couchbase cluster.
- Server Nodes - To show active nodes, node configuration, node activity and performance, and cluster statistics. Provides node failover, node removal.
- Data Buckets - To create data buckets, edit bucket settings, and view bucket statistics.
- Views - To create and manage view functions for indexing and querying data including managing documents.
- Log - To display errors and problems.
- Settings - To provide configuration and information for the cluster, update notifications, auto failover, alerts, auto compaction, sample buckets, and account management.

Related Links

Couchbase Administration on page 89

Cluster Overview on page 249
Cluster Overview is the home page and provides an overview of your cluster health, including RAM and disk usage and activity.

Server Nodes on page 252
The **Server Nodes** section shows statistics across the server nodes in the cluster.

Data Buckets on page 259

Views on page 278
The Views section allows you to manage your development and production views.

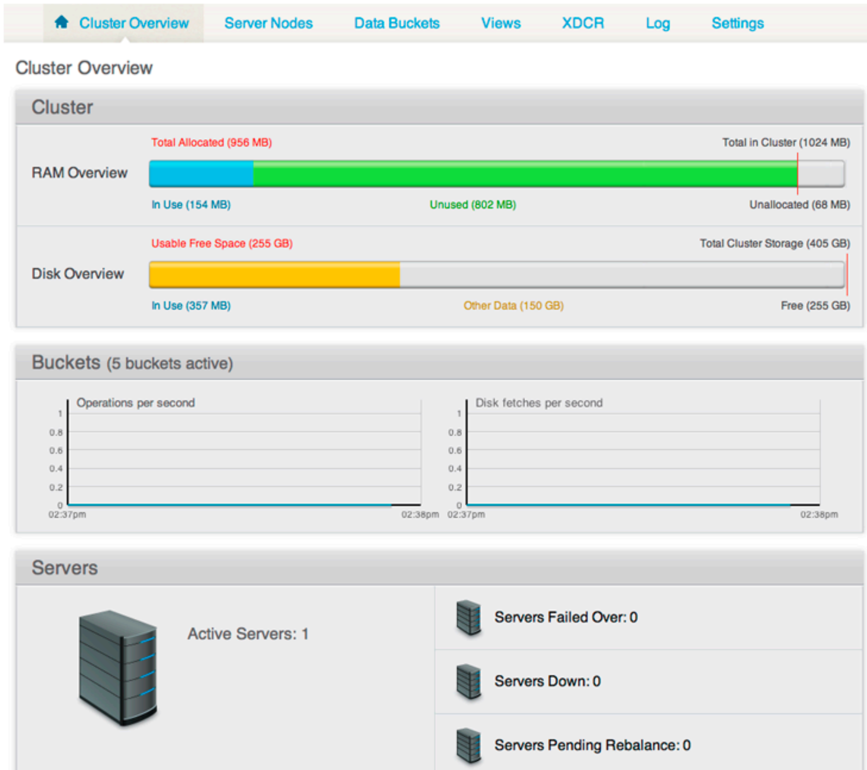
XDCR on page 283
The XDCR panel is used to create a remote cluster reference and specify replication.

Log on page 293
The Log section provides a built-in event log for Couchbase Server.

Settings on page 294
The **Settings** section provides configuration and information for the cluster, update notifications, auto failover, alerts, auto compaction, sample buckets, and account management.

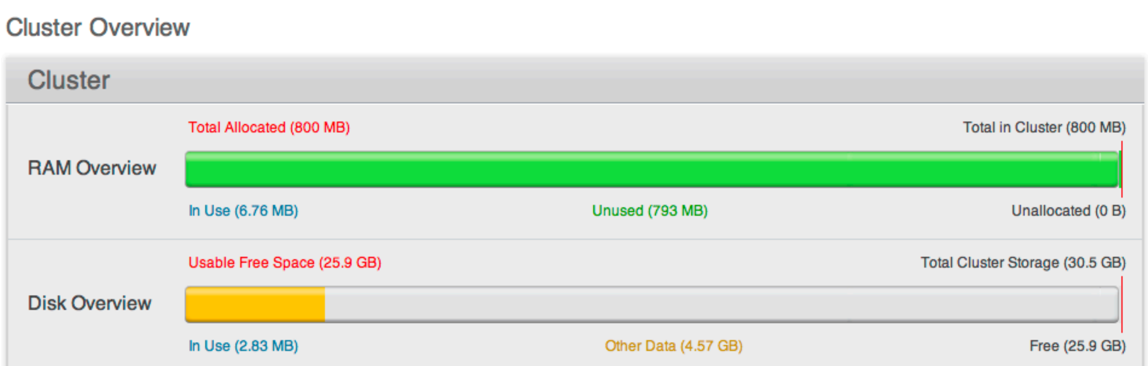
Cluster Overview

Cluster Overview is the home page and provides an overview of your cluster health, including RAM and disk usage and activity.



Viewing cluster overview

The Cluster section provides information on the RAM and disk usage information for your cluster.



For the RAM information you are provided with a graphical representation of your RAM situation, including:

- `Total in Cluster`

Total RAM configured within the cluster. This is the total amount of memory configured for all the servers within the cluster.

- `Total Allocated`

The amount of RAM allocated to data buckets within your cluster.

- `Unallocated`

The amount of RAM not allocated to data buckets within your cluster.

- `In Use`

The amount of memory across all buckets that is actually in use (i.e. data is actively being stored).

- `Unused`

The amount of memory that is unused (available) for storing data.

The `Disk Overview` section provides similar summary information for disk storage space across your cluster.

- `Total Cluster Storage`

Total amount of disk storage available across your entire cluster for storing data.

- `Usable Free Space`

The amount of usable space for storing information on disk. This figure shows the amount of space available on the configured path after non-Couchbase files have been taken into account.

- `Other Data`

The quantity of disk space in use by data other than Couchbase information.

- `In Use`

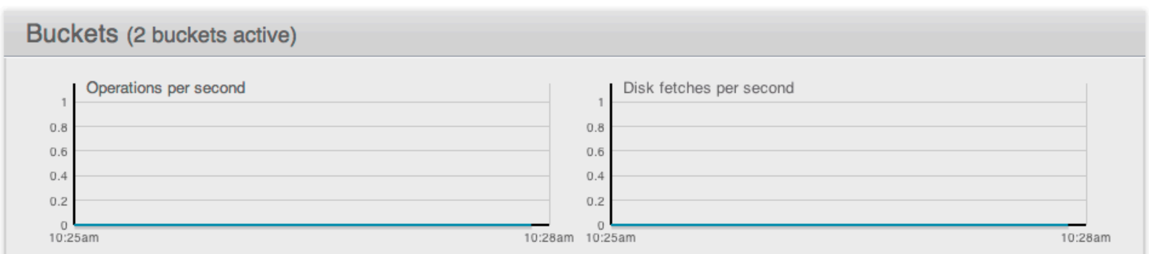
The amount of disk space being used to actively store information on disk.

- `Free`

The free space available for storing objects on disk.

Viewing buckets

The `Buckets` section provides two graphs showing the `Operations per second` and `Disk fetches per second`.

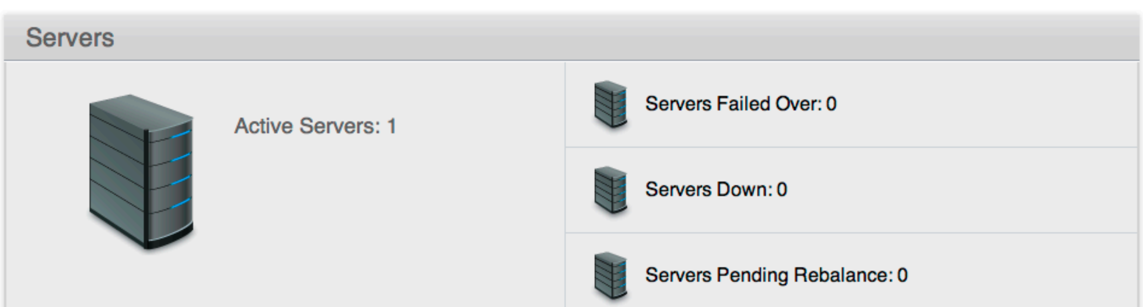


The `Operations per second` provides information on the level of activity on the cluster in terms of storing or retrieving objects from the data store.

The `Disk fetches per second` indicates how frequently Couchbase is having to go to disk to retrieve information instead of using the information stored in RAM.

Viewing servers

The `Servers` section indicates overall server information for the cluster:



- `Active Servers` is the number of active servers within the current cluster configuration.

- `Servers Failed Over` is the number of servers that have failed over due to an issue that should be investigated.
- `Servers Down` shows the number of servers that are down and not-contactable.
- `Servers Pending Rebalance` shows the number of servers that are currently waiting to be rebalanced after joining a cluster or being reactivated after failover.

Related Links

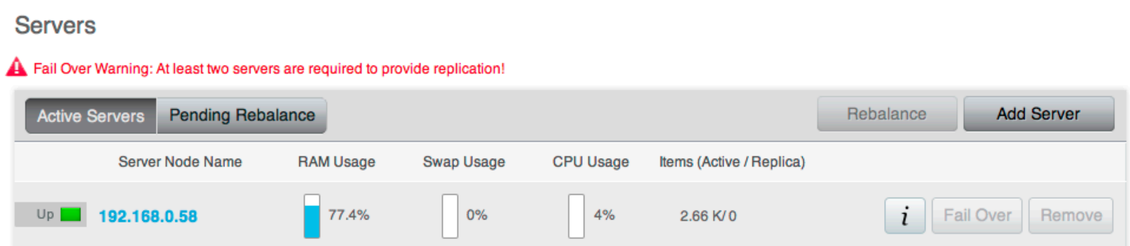
Couchbase web console on page 249
The Couchbase web console is the main tool for managing the Couchbase environment.

Server Nodes

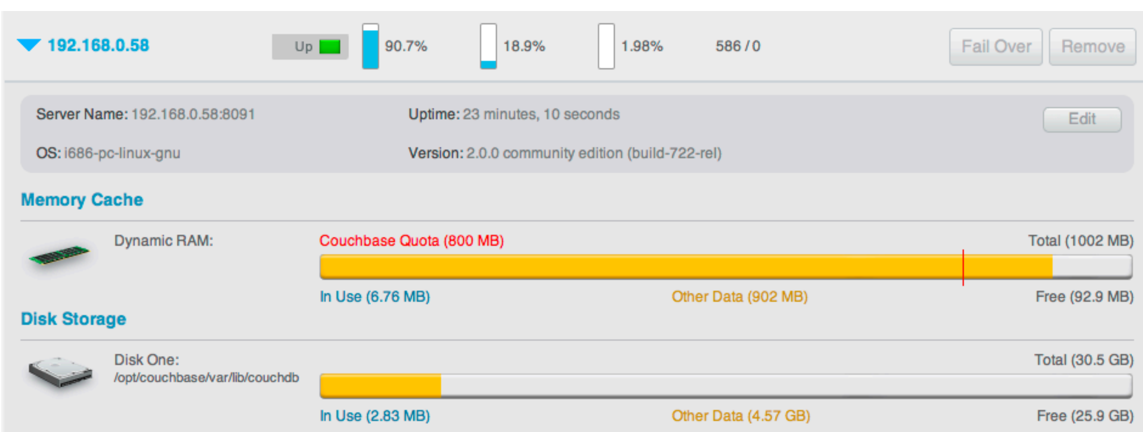
The **Server Nodes** section shows statistics across the server nodes in the cluster.

In addition to monitoring buckets over all the nodes within the cluster, Couchbase Server also includes support for monitoring the statistics for an individual node.

The Server Nodes monitoring overview shows summary data for the Swap Usage, RAM Usage, CPU Usage and Active Items across all the nodes in your cluster.



Clicking the triangle next to a server displays server node specific information, including the IP address, OS, Couchbase version and Memory and Disk allocation information.



The detail display shows the following information:

- **Node information**
  - Server Name - The server IP address and port number used to communicated with this sever.
  - Uptime - The uptime of the Couchbase Server process. This displays how long Couchbase Server has been running as a node, not the uptime for the server.
  - OS - The operating system identifier, showing the platform, environment, operating system and operating system derivative.
  - Version - The version number of the Couchbase Server installed and running on this node.
- **Memory cache**

The Memory Cache section shows you the information about memory usage, both for Couchbase Server and for the server as a whole. You can use this to compare RAM usage within Couchbase Server to the overall available RAM. The specific details tracked are:

  - Couchbase Quota - Shows the amount of RAM in the server allocated specifically to Couchbase Server.
  - In Use - Shows the amount of RAM currently in use by stored data by Couchbase Server.
  - Other Data - Shows the RAM used by other processes on the server.
  - Free - Shows the amount of free RAM out of the total RAM available on the server.
  - Total - Shows the total amount of free RAM on the server available for all processes.

- **Disk Storage**

  This section displays the amount of disk storage available and configured for Couchbase. Information will be displayed for each configured disk.

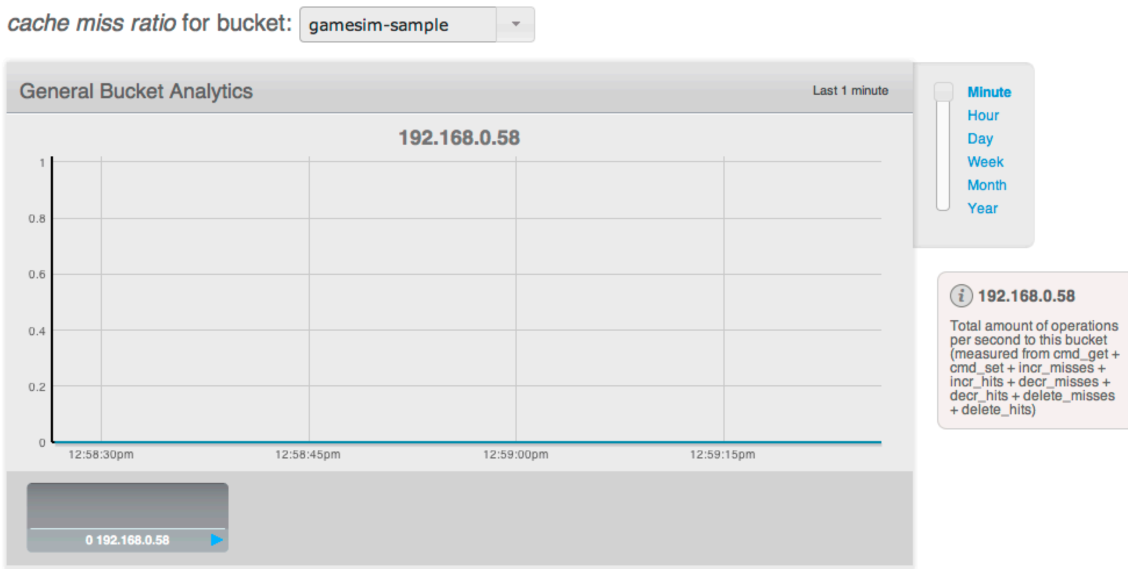
  - In Use - Shows the amount of disk space currently used to stored data for Couchbase Server.
  - Other Data - Shows the disk space used by other files on the configured device, not controlled by Couchbase Server.
  - Free - Shows the amount of free disk storage on the server out of the total disk space available.
  - Total - Shows the total disk size for the configured storage device.

Selecting a server from the list shows the server-specific version of the Bucket Monitoring overview, showing server-specific performance information.

The graphs specific to the server are:

- **swap usage** - Amount of swap space in use on this server.
- **free RAM** - Amount of RAM available on this server.
- **CPU utilization** - Percentage of CPU utilized across all cores on the selected server.
- **connection count** - Number of connections to this server of all types for client, proxy, TAP requests and internal statistics.

By clicking on the blue triangle against an individual statistic within the server monitoring display, you can optionally select to view the information for a specific bucket-statistic on an individual server, instead of across the entire cluster.



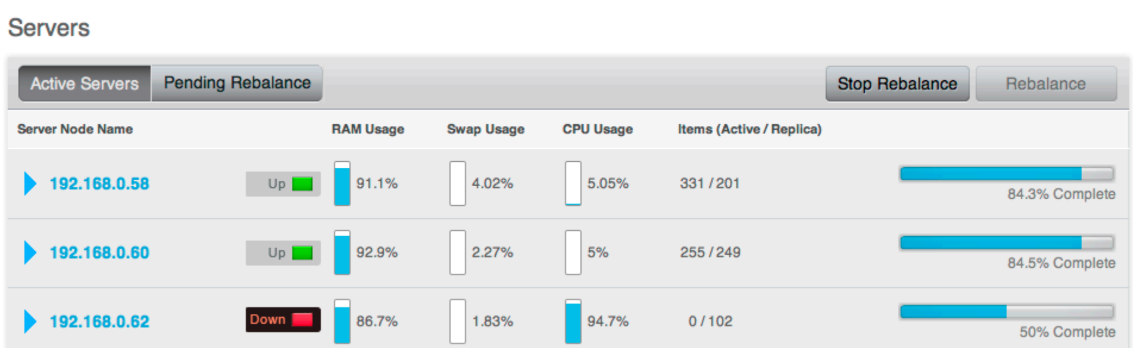
Understanding server states

Couchbase Server nodes can be in a number of different states depending on their current activity and availability. The displayed states are:

- **Up**

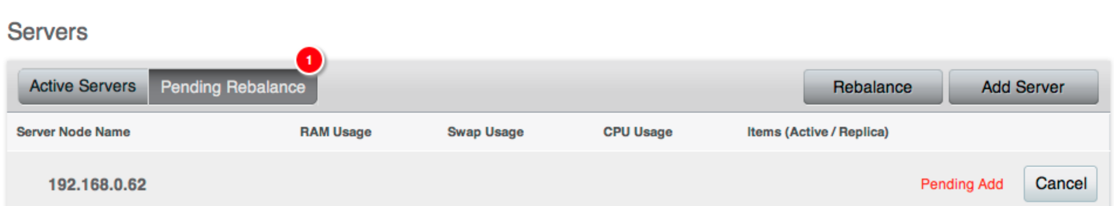
  Host is up, replicating data between nodes and servicing requests from clients.
- **Down**

  Host is down, not replicating data between nodes and not servicing requests from clients.



- **Pend**

  Host is up and currently filling RAM with data, but is not servicing requests from clients. Client access will be supported once the RAM has been pre-filled with information.



You can monitor the current server status using both the `Manage: Server Nodes` and `Monitor: Server Nodes` screens within the Web Console.

Related Links

Couchbase web console on page 249
The Couchbase web console is the main tool for managing the Couchbase environment.

Managing Rack Awareness on page 255
The Rack Awareness feature (Enterprise Edition) allows logical groupings of servers on a cluster where each server group physically belongs to a rack or availability zone. This feature provides the ability to specify that active and corresponding replica partitions be created on servers that are part of a separate rack or zone.

Related topics

Data Buckets on page 259

Managing Rack Awareness

The Rack Awareness feature (Enterprise Edition) allows logical groupings of servers on a cluster where each server group physically belongs to a rack or availability zone. This feature provides the ability to specify that active and corresponding replica partitions be created on servers that are part of a separate rack or zone.

This section describes how to manage server groups through the Web Console. Server and server groups can also be managed throught the Couchbase command-line interface (CLI) and REST API.

Note: By default, when a Couchbase cluster is initialized, Group 1 is created.

- Upgrade all servers in the cluster to version 2.5 or higher and to Couchbase Enterprise Edition
- Configure at least two server groups.
- Configure all of the servers to use server groups.
- Configure each server group to have the same number of serves (recommended).

The servers and server groups are displayed from the Server Nodes tab. Server groups are edited and created by clicking on Server Groups

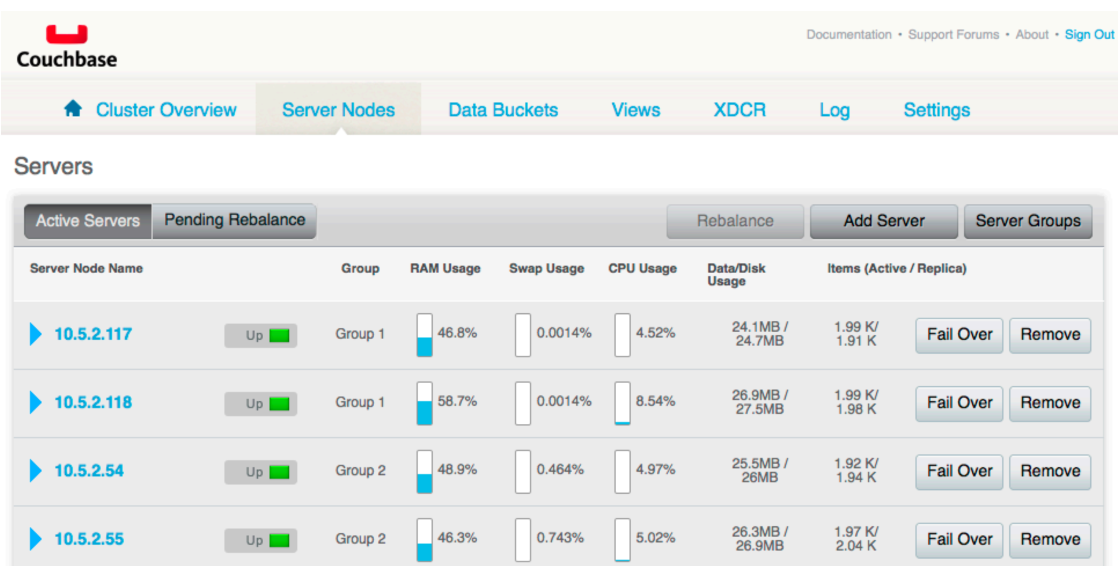


Figure 4: Server Nodes tab

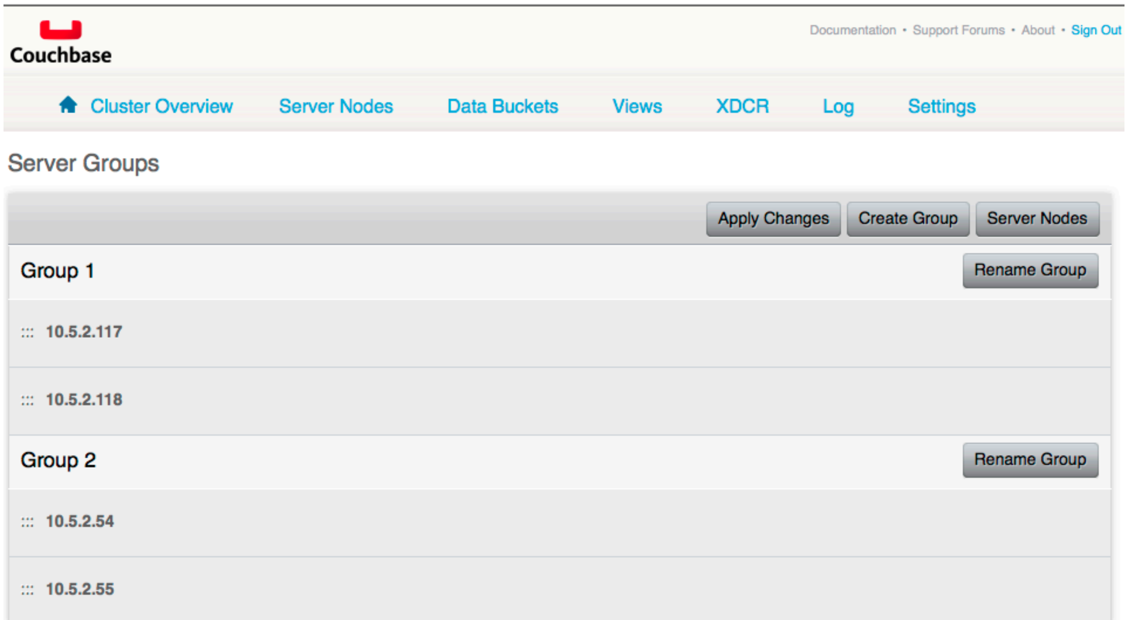


Figure 5: Server Groups

Related Links

Server Nodes on page 252
The **Server Nodes** section shows statistics across the server nodes in the cluster.

Creating server groups on page 256
Server groups are created from the web console by selecting the Server Nodes tab and clicking on Server Groups.

Deleting server groups on page 257
Server groups are deleted by removing all nodes from the server group and then deleting the server group.

Moving servers between server groups on page 257
Servers are moved between server groups from the Server Groups section.

Adding servers to server groups on page 258
Server are added to server groups from the Server Nodes tab.

Removing servers from server groups on page 258
Servers are removed from server groups group.

Renaming server groups on page 258
Server groups are rename from the web console by selecting the Server Nodes tab and clicking on Server Groups.

couchbase-cli tool on page 361
Rack Awareness REST API on page 452
The Rack Awareness feature allows logical groupings of servers on a cluster where each server group physically belongs to a rack or availability zone.

Creating server groups

Server groups are created from the web console by selecting the Server Nodes tab and clicking on Server Groups.

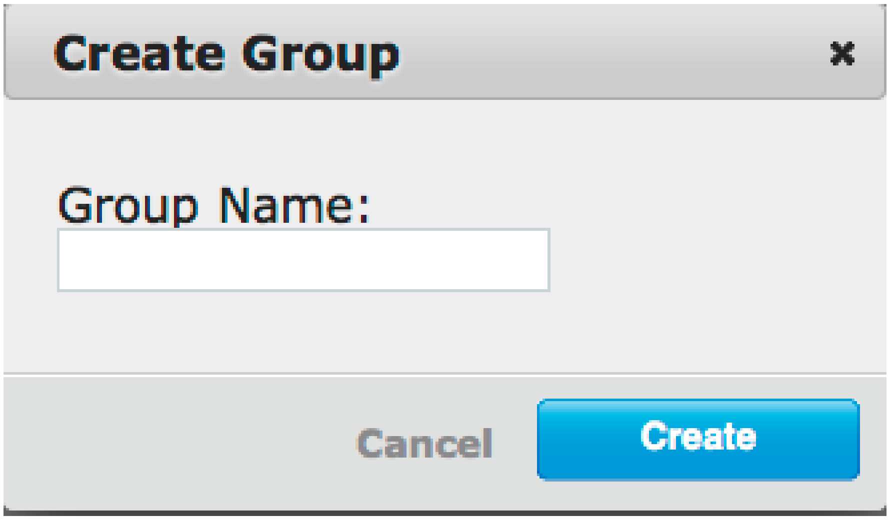


Figure 6: Create server group

1. From the Server Nodes tab, click Server Groups.
2. Click Create Group and provide a group name to the Add Group pop-up.
3. Click Create.

Related Links

Managing Rack Awareness on page 255
The Rack Awareness feature (Enterprise Edition) allows logical groupings of servers on a cluster where each server group physically belongs to a rack or availability zone. This feature provides the ability to specify that active and corresponding replica partitions be created on servers that are part of a separate rack or zone.

Deleting server groups

Server groups are deleted by removing all nodes from the server group and then deleting the server group.

1. From the Server Nodes tab, click Remove to remove all nodes from the server group.
2. From the Server Nodes tab, click Server Groups.
3. Click on "This group is empty, click to delete." which is displayed if the server group is empty.
4. Click Delete from the Removing pop-up.

Related Links

Managing Rack Awareness on page 255
The Rack Awareness feature (Enterprise Edition) allows logical groupings of servers on a cluster where each server group physically belongs to a rack or availability zone. This feature provides the ability to specify that active and corresponding replica partitions be created on servers that are part of a separate rack or zone.

Moving servers between server groups

Servers are moved between server groups from the Server Groups section.

Figure 7: Apply changes to server group

1. From the Server Nodes tab, click Server Groups
2. Drag and drop the server from one group to another.
3. Click Apply Changes.
4. From the Server Nodes tab, click Rebalance

Related Links

Managing Rack Awareness on page 255
The Rack Awareness feature (Enterprise Edition) allows logical groupings of servers on a cluster where each server group physically belongs to a rack or availability zone. This feature provides the ability to specify that active and corresponding replica partitions be created on servers that are part of a separate rack or zone.

Adding servers to server groups

Server are added to server groups from the Server Nodes tab.

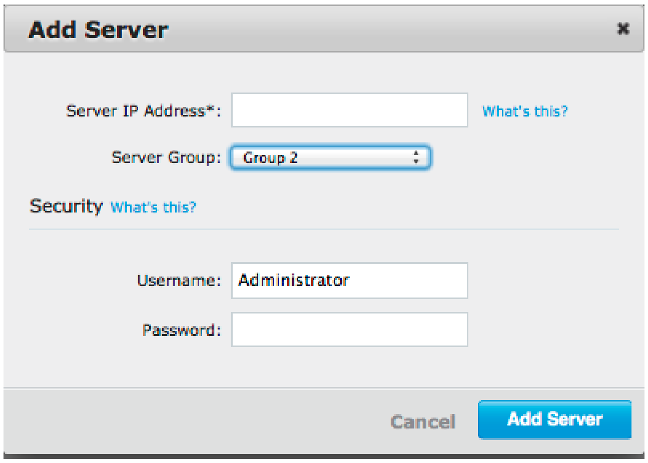


Figure 8: Add server to server group

1. From the Server Nodes tab, click Add Server
2. Provide the Server IP Address, select a server group from the drop down menu, and provide the administrator username and password for the server being added.
3. Click Add Server
4. From the Server Nodes tab, click Rebalance

Related Links

Managing Rack Awareness on page 255
The Rack Awareness feature (Enterprise Edition) allows logical groupings of servers on a cluster where each server group physically belongs to a rack or availability zone. This feature provides the ability to specify that active and corresponding replica partitions be created on servers that are part of a separate rack or zone.

Removing servers from server groups

Servers are removed from server groups group.

1. From the Server Nodes tab, click Remove for the server that you want to delete.
2. Click Remove from the confirmation pop-up.
3. From the Server Nodes tab, click Rebalance.

Related Links

Managing Rack Awareness on page 255
The Rack Awareness feature (Enterprise Edition) allows logical groupings of servers on a cluster where each server group physically belongs to a rack or availability zone. This feature provides the ability to specify that active and corresponding replica partitions be created on servers that are part of a separate rack or zone.

Renaming server groups

Server groups are rename from the web console by selecting the Server Nodes tab and clicking on Server Groups.

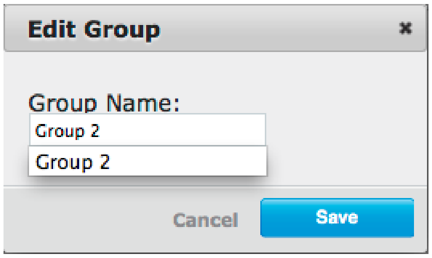


Figure 9: Edit server group

1. From the Server Nodes tab, click Server Groups.
2. Click Edit Group
3. Change the group name and click Save.

Related Links

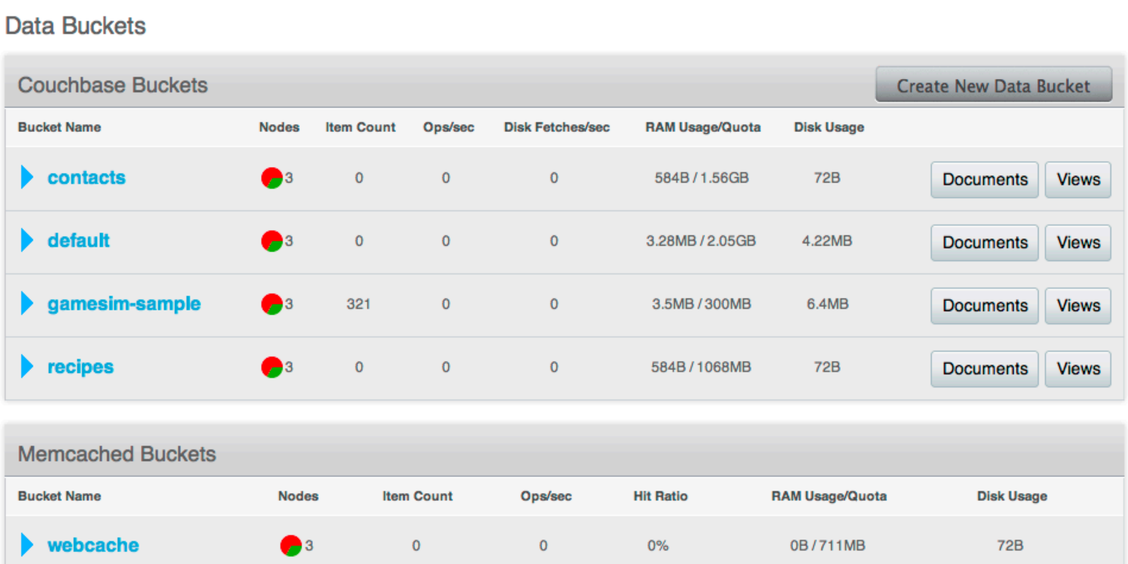
Managing Rack Awareness on page 255
The Rack Awareness feature (Enterprise Edition) allows logical groupings of servers on a cluster where each server group physically belongs to a rack or availability zone. This feature provides the ability to specify that active and corresponding replica partitions be created on servers that are part of a separate rack or zone.

Data Buckets

Couchbase Server provides a range of statistics and settings through the `Data Buckets` and `Server Nodes`. These show overview and detailed information so that administrators can better understand the current state of individual nodes and the cluster as a whole.

The `Data Buckets` page displays a list of all the configured buckets on your system (of both Couchbase and memcached types). The page provides a quick overview of your cluster health from the perspective of the configured buckets, rather than whole cluster or individual servers.

The information is shown in the form of a table, as seen in the figure below.



The list of buckets are separated by the bucket type. For each bucket, the following information is provided in each column:

- **Bucket name** is the given name for the bucket. Clicking on the bucket name takes you to the individual bucket statistics page.
- **RAM Usage/Quota** shows the amount of RAM used (for active objects) against the configure bucket size.

- **Disk Usage** shows the amount of disk space in use for active object data storage.
- **Item Count** indicates the number of objects stored in the bucket.
- `Ops/sec` shows the number of operations per second for this data bucket.
- **Disk Fetches/sec** shows the number of operations required to fetch items from disk.
- Clicking the **Bucket Name** opens the basic bucket information summary.
- Clicking the **Documents** button will take you to a list of objects identified as parseable documents.
- The **Views** button allows you to create and manage views on your stored objects.

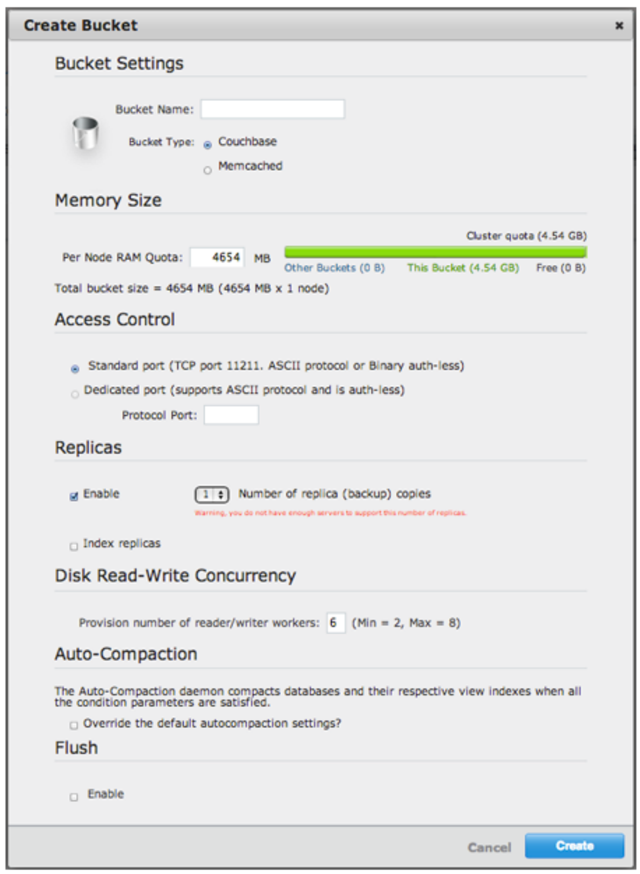
To create a new data bucket, click the **Create New Data Bucket**.

Creating and editing data buckets

When creating a new data bucket, or editing an existing one, you will be presented with the bucket configuration screen. From here you can set the memory size, access control and other settings, depending on whether you are editing or creating a new bucket, and the bucket type.

You can create a new bucket in Couchbase Web Console under the Data Buckets tab.

1. Click Data Buckets | Create New Data Bucket. You see the `Create Bucket` panel, as follows:



2. Select a name for the new bucket.

   The bucket name can only contain characters in range A-Z, a-z, 0–9 as well as underscore, period, dash and percent symbols.

   Tip: Create a named bucket specifically for your application. Any default bucket you initially set up with Couchbase Server should not be used for storing live application data. The default bucket you create when you first install Couchbase Server should be used only for testing.

3. Select a Bucket Type, either Memcached or Couchbase.

   The options that appear in this panel differ based on the bucket type.

For `Couchbase` bucket type:

- **Memory Size**

The amount of available RAM on this server which should be allocated to the bucket. Note that the allocation is the amount of memory that will be allocated for this bucket on each node, not the total size of the bucket across all nodes.

- **Replicas**

For Couchbase buckets you can enable data replication so that the data is copied to other nodes in a cluster. You can configure up to three replicas per bucket. If you set this to one, you need to have a minimum of two nodes in your cluster and so forth. If a node in a cluster fails, after you perform failover, the replicated data will be made available on a functioning node. This provides continuous cluster operations in spite of machine failure.

You can disable replication by deselecting the `Enable` checkbox.

You can disable replication by setting the number of replica copies to zero (0).

To configure replicas, Select a number in `Number of replica (backup) copies` drop-down list.

To enable replica indexes, Select the `Index replicas` checkbox. Couchbase Server can also create replicas of indexes. This ensures that indexes do not need to be rebuilt in the event of a node failure. This will increase network load as the index information is replicated along with the data.

- **Disk Read-Write Concurrency**

Multiple readers and writers are supported to persist data onto disk. For earlier versions of Couchbase Server, each server instance had only single disk reader and writer threads. By default this is set to three total threads per data bucket, with two reader threads and one writer thread for the bucket.

For now, leave this setting at the default. In the future, when you create new data buckets you can update this setting.

- **Flush**

To enable the operation for a bucket, click the `Enable` checkbox. Enable or disable support for the Flush command, which deletes all the data in an a bucket. The default is for the flush operation to be disabled.

For `Memcached` bucket type:

- **Memory Size**

The bucket is configured with a per-node amount of memory. Total bucket memory will change as nodes are added/removed.

Warning: Changing the size of a memcached bucket will erase all the data in the bucket and recreate it, resulting in loss of all stored data for existing buckets.

- **Auto-Compaction**

Both data and index information stored on disk can become fragmented. Compaction rebuilds the stored data on index to reduce the fragmentation of the data.

You can opt to override the default auto compaction settings for this individual bucket. Default settings are configured through the `Settings` menu. If you override the default autocompaction settings, you can configure the same parameters, but the limits will affect only this bucket.

For either bucket type provide these two settings in the Create Bucket panel:

- **Access Control**

  The access control configures the port clients use to communicate with the data bucket, and whether the bucket requires a password.

  To use the TCP standard port (11211), the first bucket you create can use this port without requiring SASL authentication. For each subsequent bucket, you must specify the password to be used for SASL authentication, and client communication must be made using the binary protocol.

To use a dedicated port, select the dedicate port radio button and enter the port number you want to use. Using a dedicated port supports both the text and binary client protocols, and does not require authentication.

Note: When defining a port on a bucket, the server automatically starts up a copy of Moxi on the servers, and exposes it on that port. This supports the ASCII memcached protocol. However, Couchbase strongly recommend against using Moxi in this way. If needed, a client-side Moxi should be installed on the application servers and have it connect to this bucket (whether it is “port” or “password” doesn’t matter).

When defining a password on a bucket, it requires a client that supports the binary memcached protocol with SASL (all Couchbase client libraries and client-side Moxi provide this support). Defining a password on a bucket is the recommended approach.

- **Flush**

  Enable or disable support for the Flush command, which deletes all the data in an a bucket. The default is for the flush operation to be disabled. To enable the operation for a bucket, select the Enable checkbox.
- Click **Create**.

Editing Couchbase buckets

You can edit a number of settings for an existing Couchbase bucket in Couchbase Web Console:

- **Access Control**, including the standard port/password or custom port settings.
- **Memory Size** can be modified providing you have unallocated space within your Cluster configuration. You can reduce the amount of memory allocated to a bucket if that space is not already in use.
- **Auto-Compaction** settings, including enabling the override of the default auto-compaction settings, and bucket-specific auto-compaction.
- **Flush** support. You can enable or disable support for the Flush command.

The bucket name cannot be modified. To delete the configured bucket entirely, click the **Delete** button.

Editing Memcached buckets

For Memcached buckets, you can modify the following settings when editing an existing bucket:

- `Access Control`, including the standard port/password or custom port settings.
- `Memory Size` can be modified providing you have unallocated RAM quota within your Cluster configuration. You can reduce the amount of memory allocated to a bucket if that space is not already in use.

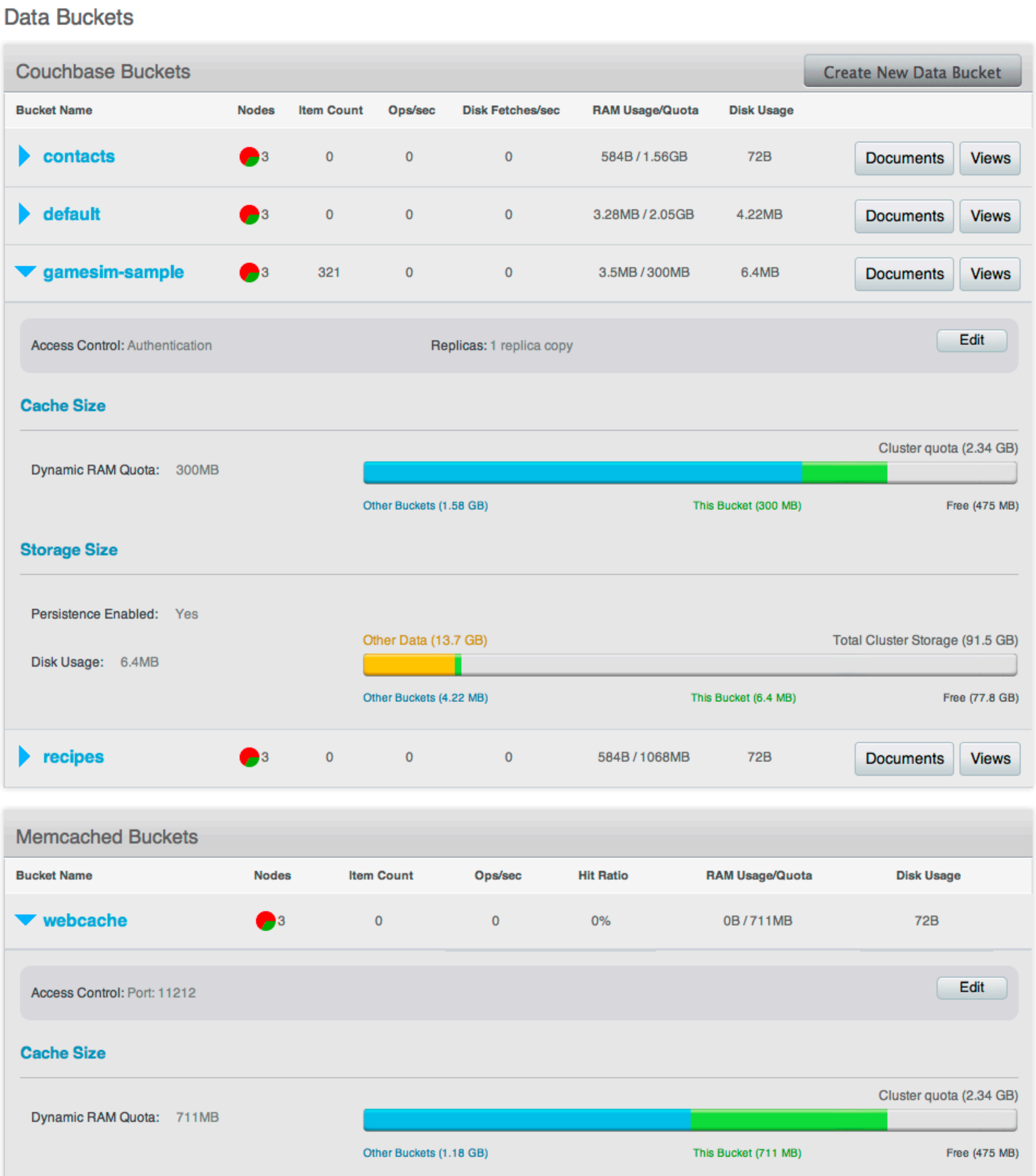
You can delete the bucket entirely by clicking the **Delete** button.

You can empty a Memcached bucket of all the cached information that it stores by using the `Flush` button.

Warning: Using the `Flush` button removes all the objects stored in the Memcached bucket. Using this button on active Memcached buckets may delete important information.

Bucket information

You can obtain basic information about the status of your data buckets by clicking on the drop-down next to the bucket name under the `Data Buckets` page. The bucket information shows memory size, access, and replica information for the bucket, as shown in the figure below.



You can edit the bucket information by clicking the `Edit` button within the bucket information display.

Related Links

Couchbase web console on page 249
The Couchbase web console is the main tool for managing the Couchbase environment.

Using multi-readers and writers on page 264
Multiple readers and writers are supported to increase disk I/O throughput.

Monitoring statistics on page 266
Within the **Data Bucket** tab, information and statistics about bucketsand nodes is displayed for the entire Couchbase Server cluster. The information is aggregated from all the server nodes within the configured cluster for the selected bucket.

Managing documents on page 276

The Document Viewer and Editor enables you to browse, view, and edit individual documents stored in Couchbase Server buckets.

Using multi-readers and writers

Multiple readers and writers are supported to increase disk I/O throughput.

Multiple readers and writers are supported to persist data onto disk. By default, this is set to three total workers per data bucket, with two reader workers and one writer worker for the bucket. This feature helps increase disk I/O throughput. If disk utilization is below the optimal level, increase the setting to improve disk utilization. If disk utilization is near the maximum and heavy I/O contention occurs, decrease this setting. By default, three total readers and writers are allocated.

Related Links

Data Buckets on page 259

Specifying read-write for new buckets on page 264
The number of readers and writer are typically set when creating a data bucket: **Data Buckets > Create New Data Bucket > Disk Read-Write Concurrency**.

Specifying read-write for existing buckets on page 265
The number of readers and writers can be changed for existing data buckets: **Data Buckets > Data bucket drop-down > Disk Read-Write Concurrency**.

Viewing the impact of read-write changes on page 266
Through the Web Console, the impact of reader/writer changes for a bucket is viewable via that bucket analytics: **Data Buckets > Bucket Name > DISK QUEUES**.

Related topics

REST API overview on page 426

Buckets REST API on page 456
The bucket management and configuration REST API endpoints provide a fine level of control over the individual buckets in the cluster, their configuration, and specific operations.

Changing bucket read-write threads on page 472
To change the disk readers and writers setting, use the `POST` operation with the `/pools/default/bucket/bucket_name` URI.

Read-write thread stats on page 418
The `cbstats` tools provides the status of read-write threads for buckets.

Handling server warmup on page 135
Couchbase server warmup behavior can be modified changing the access scanner and warmup threshold settings via the `cbepctl` tool.

Specifying read-write for new buckets

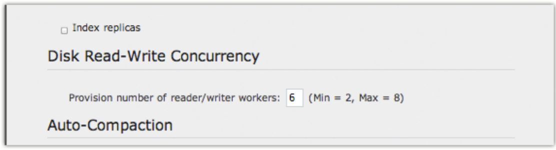
The number of readers and writer are typically set when creating a data bucket: **Data Buckets > Create New Data Bucket > Disk Read-Write Concurrency**.

The number of readers and writers are typically specified (default: 3) when a new bucket is created. Although, the read-write setting can also be modified on existing buckets.

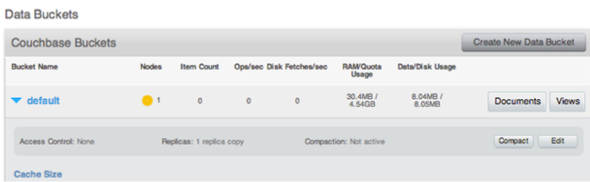
This default bucket is now ready to receive and serve requests. If named bucket is created, a similar status indicator displays next to the named bucket.

1. Under Data Buckets, click **Create New Data Bucket**.

   A Configure Bucket panel appears where you can provide settings for the new bucket.
2. Under **Disk I/O Optimization**, specify the bucket disk I/O priority. Low (default) sets three (3) reader/writers and High allocates eight (8).



3. Provide other bucket-level settings of your choice.
4. Click **Create**.
   The new bucket displays with a yellow indicator while in warmup phase and a green indicator after warmup is complete:



Related Links

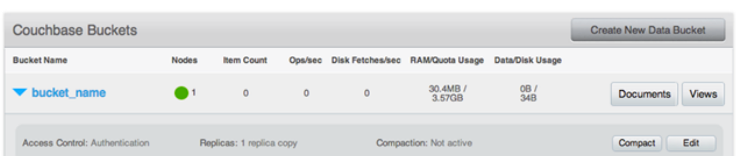
Using multi-readers and writers on page 264
Multiple readers and writers are supported to increase disk I/O throughput.

Specifying read-write for existing buckets

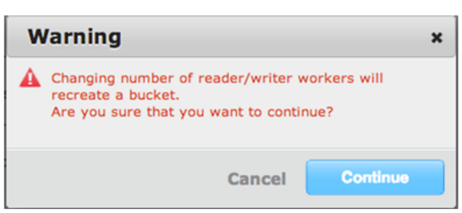
The number of readers and writers can be changed for existing data buckets: **Data Buckets > Data bucket drop-down > Disk Read-Write Concurrency**.

After a bucket has been created, the reader/writer setting can be changed. After changing the setting for a bucket, the bucket is re-started and goes through server warmup before the bucket becomes available.

1. Click the **Data Buckets** tab.
   A table with all data buckets in your cluster appears.
2. Click the drop-down next to your data bucket.
   General information about the bucket appears as well as controls for the bucket.



3. Click **Edit**.
   A Configure Bucket panel appears where the reader-writer setting is changed.
4. Under **Disk I/O Optimization**, specify the bucket disk I/O priority. Low (default) sets three (3) reader/writers and High allocates eight (8).
5. Click **Save**.
   A warning appears indicating that this change recreates the data bucket.



6. Click **Continue**.

The Data Buckets tab appears and the named bucket displays with a yellow or green indicator. A yellow indicator means that the bucket is recreated and is warming up. A green indicates means that the bucket has completed warmup. At this point, the bucket is ready to receive and serve requests.

Related Links

Using multi-readers and writers on page 264
Multiple readers and writers are supported to increase disk I/O throughput.

Viewing the impact of read-write changes

Through the Web Console, the impact of reader/writer changes for a bucket is viewable via that bucket analytics: **Data Buckets > Bucket Name > DISK QUEUES**.

A change to the number of readers/writers is reflected in the change in the active and replica *fill rate*, *drain rate*, and *average age*.

1. Click the **Data Buckets** tab.
2. Click on the bucket name (rather than the expander icon) to view the analytics for the bucket.
3. Expand the **DISK QUEUE** section to view active, replica, pending, and total summary.

   Additionally, the number of items, fill rate, drain rate, and average age is summarized and displayed on a per server basis.

Related Links

Using multi-readers and writers on page 264
Multiple readers and writers are supported to increase disk I/O throughput.

Monitoring statistics

Within the **Data Bucket** tab, information and statistics about bucketsand nodes is displayed for the entire Couchbase Server cluster. The information is aggregated from all the server nodes within the configured cluster for the selected bucket.

The following functionality is available through this display, and is common to all the graphs and statistics display within the web console.

- `Bucket Selection`

The `Data Buckets` selection list allows you to select which of the buckets configured on your cluster is to be used as the basis for the graph display. The statistics shown are aggregated over the whole cluster for the selected bucket.

- `Server Selection`

The `Server Selection` option enables you to limit the display to an individual server or entire cluster. The individual node selection displays information for the node. The all server nodes selection displays information for the entire cluster.

- `Interval Selection`

The `Interval Selection` at the top of the main graph changes interval display for all graphs displayed on the page. For example, selecting `Minute` shows information for the last minute, continuously updating.

As the selected interval increases, the amount of statistical data displayed will depend on how long your cluster has been running.

- `Statistic Selection`

All of the graphs within the display update simultaneously. Clicking on any of the smaller graphs will promote that graph to be displayed as the main graph for the page.

- `Individual Server Selection`

Clicking the blue triangle next to any of the smaller statistics graphs enables you to show the selected statistic individual for each server within the cluster, instead of aggregating the information for the entire cluster.

Individual bucket monitoring

Bucket monitoring within the Couchbase Web Console has been updated to show additional detailed information. The following statistic groups are available for Couchbase bucket types.

- **Summary**

The summary section provides a quick overview of the cluster activity.

- **vBucket Resources**

This section provides detailed information on the vBucket resources across the cluster, including the active, replica and pending operations.

- **Disk Queues**

Disk queues show the activity on the backend disk storage used for persistence within a data bucket. The information displayed shows the active, replica and pending activity.

- **TAP Queues**

The TAP queues section provides information on the activity within the TAP queues across replication, rebalancing and client activity.

- **XDCR Destination**

The XDCR Destination section show you statistical information about the Cross Datacenter Replication (XDCR), if XDCR has been configured.

- **View Stats**

The View Stats section allows you to monitor the statistics for each production view configured within the bucket or system.

- **Top Keys**

This shows a list of the top 10 most actively used keys within the selected data bucket.

For Memcached bucket types, the Memcached statistic summary is provided.

Bucket monitoring — summary statistics

The summary section is designed to provide a quick overview of the cluster activity. Each graph (or selected graph) shows information based on the currently selected bucket.

The following graph types are available:

ops per second	The total number of operations per second on this bucket.
cache miss ratio	Ratio of reads per second to this bucket which required a read from disk rather than RAM.
creates per second	Number of new items created in this bucket per second.
updates per second	Number of existing items updated in this bucket per second.
XDCR ops per sec	Number of XDCR related operations per second for this bucket.
disk reads per sec	Number of reads per second from disk for this bucket.
temp OOM per sec	Number of temporary out of memory conditions per second.
gets per second	Number of get operations per second.
sets per second	Number of set operations per second.
deletes per second	Number of delete operations per second.
items	Number of items (documents) stored in the bucket.
disk write queue	Size of the disk write queue.
docs data size	Size of the stored document data.
docs total disk size	Size of the persisted stored document data on disk.
doc fragmentation %	Document fragmentation of persisted data as stored on disk.
XDC replication queue	Size of the XDCR replication queue.

total disk size	Total size of the information for this bucket as stored on disk, including persisted and view index data.
views data size	Size of the view data information.
views total disk size	Size of the view index information as stored on disk.
views fragmentation %	Percentage of fragmentation for a given view index.
view reads per second	Number of view reads per second.
memory used	Amount of memory used for storing the information in this bucket.
high water mark	High water mark for this bucket (based on the configured bucket RAM quota).
low water mark	Low water mark for this bucket (based on the configured bucket RAM quota).
disk update time	Time required to update data on disk.

Monitoring vBucket resources

The vBucket statistics provide information for all vBucket types within the cluster across three different states. Within the statistic display the table of statistics is organized in four columns, showing the Active, Replica and Pending states for each individual statistic. The final column provides the total value for each statistic.

The Active column displays the information for vBuckets within the Active state. The Replica column displays the statistics for vBuckets within the Replica state (i.e. currently being replicated). The Pending columns shows statistics for vBuckets in the Pending state, i.e. while data is being exchanged during rebalancing.

These states are shared across all the following statistics. For example, the graph `new items per sec` within the `Active` state column displays the number of new items per second created within the vBuckets that are in the active state.

The individual statistics, one for each state, shown are:

- `vBuckets`

The number of vBuckets within the specified state.

- `items`

Number of items within the vBucket of the specified state.

- `resident %`

Percentage of items within the vBuckets of the specified state that are resident (in RAM).

- `new items per sec.`

Number of new items created in vBuckets within the specified state. Note that new items per second is not valid for the Pending state.

- `ejections per second`

Number of items ejected per second within the vBuckets of the specified state.

- `user data in RAM`

Size of user data within vBuckets of the specified state that are resident in RAM.

- `metadata in RAM`

Size of item metadata within the vBuckets of the specified state that are resident in RAM.

Monitoring disk queues

The Disk Queues statistics section displays the information for data being placed into the disk queue. Disk queues are used within Couchbase Server to store the information written to RAM on disk for persistence. Information is displayed for each of the disk queue states, Active, Replica and Pending.

The Active column displays the information for the Disk Queues within the Active state. The Replica column displays the statistics for the Disk Queues within the Replica state (i.e. currently being replicated). The Pending columns shows statistics for the disk Queues in the Pending state, i.e. while data is being exchanged during rebalancing.

These states are shared across all the following statistics. For example, the graph `fill rate` within the `Replica` state column displays the number of items being put into the replica disk queue for the selected bucket.

The displayed statistics are:

- `items`

The number of items waiting to be written to disk for this bucket for this state.

- `fill rate`

The number of items per second being added to the disk queue for the corresponding state.

- `drain rate`

Number of items actually written to disk from the disk queue for the corresponding state.

- `average age`

The average age of items (in seconds) within the disk queue for the specified state.

Monitoring TAP queues

The TAP queues statistics are designed to show information about the TAP queue activity, both internally, between cluster nodes and clients. The statistics information is therefore organized as a table with columns showing the statistics for TAP queues used for replication, rebalancing and clients.

The statistics in this section are detailed below:

- `TAP senders`

Number of TAP queues in this bucket for internal (replica), rebalancing or client connections.

- `items`

Number of items in the corresponding TAP queue for this bucket.

- `drain rate`

Number of items per second being sent over the corresponding TAP queue connections to this bucket.

- `back-off rate`

Number of back-offs per second sent when sending data through the corresponding TAP connection to this bucket.

- `backfill remaining`

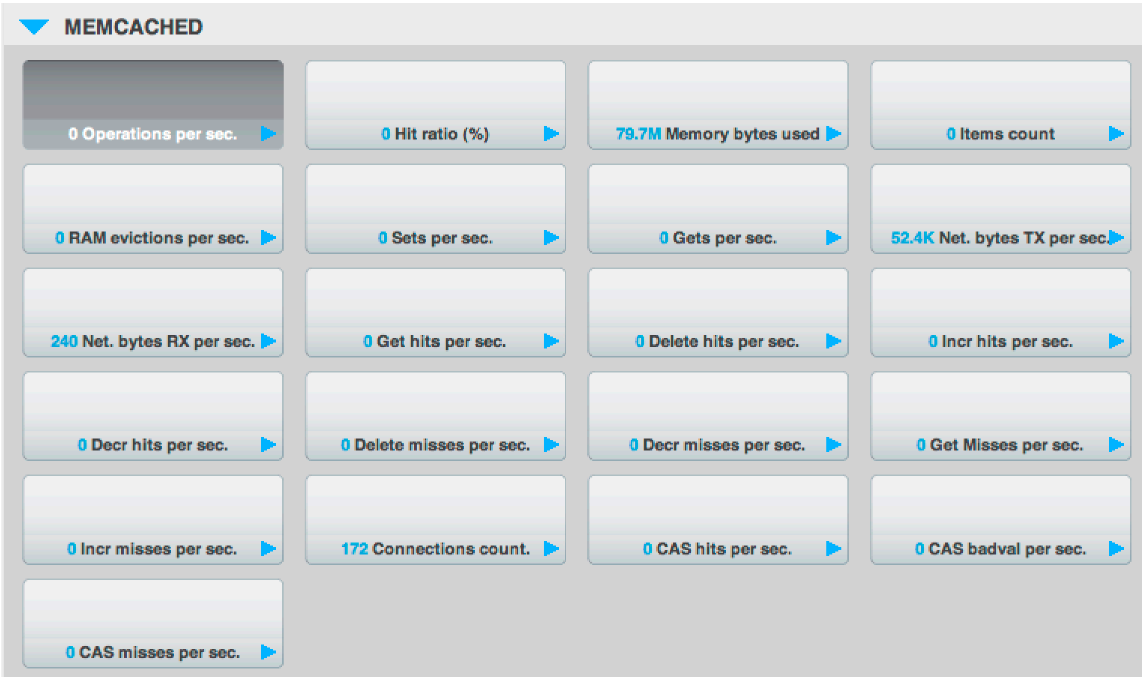
Number of items in the backfill queue for the corresponding TAP connection for this bucket.

- `remaining on disk`

Number of items still on disk that need to be loaded in order to service the TAP connection to this bucket.

Memcached buckets

For Memcached buckets, Web Console displays a separate group of statistics:



The Memcached statistics are:

- `Operations per sec.`

Total operations per second serviced by this bucket

- `Hit Ratio %`

Percentage of get requests served with data from this bucket

- `Memory bytes used`

Total amount of RAM used by this bucket

- `Items count`

Number of items stored in this bucket

- `RAM evictions per sec.`

Number of items per second evicted from this bucket

- `Sets per sec.`

Number of set operations serviced by this bucket

- `Gets per sec.`

Number of get operations serviced by this bucket

- `Net. bytes TX per sec`

Number of bytes per second sent from this bucket

- `Net. bytes RX per sec.`

Number of bytes per second sent into this bucket

- `Get hits per sec.`

Number of get operations per second for data that this bucket contains

- `Delete hits per sec.`

Number of delete operations per second for data that this bucket contains

- `Incr hits per sec.`

Number of increment operations per second for data that this bucket contains

- `Decr hits per sec.`

Number of decrement operations per second for data that this bucket contains

- `Delete misses per sec.`

Number of delete operations per second for data that this bucket does not contain

- `Decr misses per sec.`

Number of decr operations per second for data that this bucket does not contain

- `Get Misses per sec.`

Number of get operations per second for data that this bucket does not contain

- `Incr misses per sec.`

Number of increment operations per second for data that this bucket does not contain

- `CAS hits per sec.`

Number of CAS operations per second for data that this bucket contains

- `CAS badval per sec.`

Number of CAS operations per second using an incorrect CAS ID for data that this bucket contains

- `CAS misses per sec.`

Number of CAS operations per second for data that this bucket does not contain

Monitoring outgoing XDCR

The Outgoing XDCR shows the XDCR operations that are supporting cross datacenter replication from the current cluster to a destination cluster.

You can monitor the current status for all active replications in the `Ongoing Replications` section under the XDCR tab:

The `Ongoing Replications` section shows the following information:

Column	Description
Bucket	The source bucket on the current cluster that is being replicated.

Column	Description
From	Source cluster name.
To	Destination cluster name.
Status	Current status of replications.
When	Indicates when replication occurs.

The `Status` column indicates the current state of the replication configuration. Possible include:

- **Starting Up**

The replication process has just started, and the clusters are determining what data needs to be sent from the originating cluster to the destination cluster.

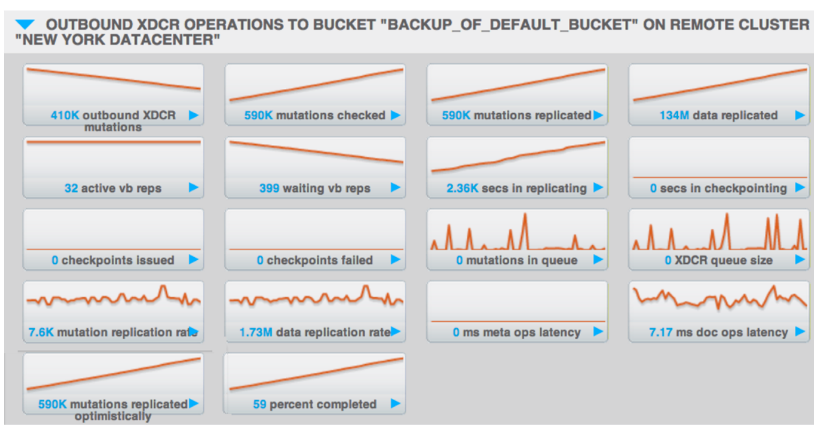
- **Replicating**

The bucket is currently being replicated and changes to the data stored on the originating cluster are being sent to the destination cluster.

- **Failed**

Replication to the destination cluster has failed. The destination cluster cannot be reached. The replication configuration may need to be deleted and recreated.

Under the `Data Buckets` tab you can click on a named Couchbase bucket and find more statistics about replication for that bucket. Couchbase Web Console displays statistics for the particular bucket; on this page you can find two drop-down areas called in the `Outgoing XDCR` and `Incoming XDCR Operations`. Both provides statistics about ongoing replication for the particular bucket. Under the `Outgoing XDCR` panel if you have multiple replication streams you will see statistics for each stream.



The statistics shown are:

- `outbound XDCR mutation`

Number of changes in the queue waiting to be sent to the destination cluster.

- `mutations checked`

Number of document mutations checked on source cluster.

- `mutations replicated`

Number of document mutations replicated to the destination cluster.

- `data replicated`

Size of data replicated in bytes.

- `active vb reps`

Number of parallel, active vBucket replicators. Each vBucket has one replicator which can be active or waiting. By default you can only have 32 parallel active replicators at once per node. Once an active replicator finishes, it will pass a token to a waiting replicator.

- `waiting vb reps`

Number of vBucket replicators that are waiting for a token to replicate.

- `secs in replicating`

Total seconds elapsed for data replication for all vBuckets in a cluster.

- `secs in checkpointing`

Time working in seconds including wait time for replication.

- `checkpoints issued`

Total number of checkpoints issued in replication queue. By default active vBucket replicators issue a checkpoint every 30 minutes to keep track of replication progress.

- `checkpoints failed`

Number of checkpoints failed during replication. This can happen due to timeouts, due to network issues or if a destination cluster cannot persist quickly enough.

- `mutations in queue`

Number of document mutations waiting in replication queue.

- `XDCR queue size`

Amount of memory used by mutations waiting in replication queue. In bytes.

- `mutation replication rate`

Number of mutations replicated to destination cluster per second.

- `data replication rate`

Bytes replicated to destination per second.

- `ms meta ops latency`

Weighted average time for requesting document metadata. In milliseconds.

- `mutations replicated optimistically`

Total number of mutations replicated with optimistic XDCR.

- `ms docs ops latency`

Weighted average time for sending mutations to destination cluster. In milliseconds.

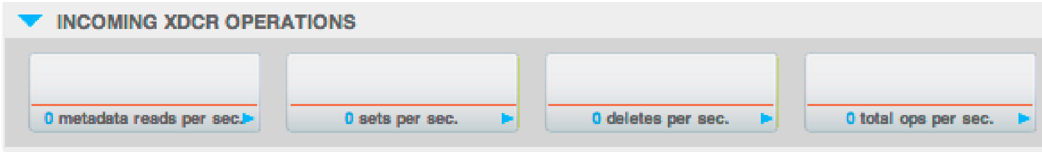
- `percent completed`

Percent of total mutations checked for metadata.

Be aware that if you use an earlier version of Couchbase Server, such as Couchbase Server 2.0, only the first three statistics appear and have the labels **changes queue, documents checked, and documents replicated** respectively. You can also get XDCR statistics using the Couchbase REST API. All of the statistics in Web Console are based on statistics via the REST API or values derived from them.

Monitoring incoming XDCR

The Incoming XDCR section shows the XDCR operations that are coming into to the current cluster from a remote cluster.



The statistics shown are:

- `metadata reads per sec.`

Number of documents XDCR scans for metadata per second. XDCR uses this information for conflict resolution.

- `sets per sec.`

Set operations per second for incoming XDCR data.

- `deletes per sec.`

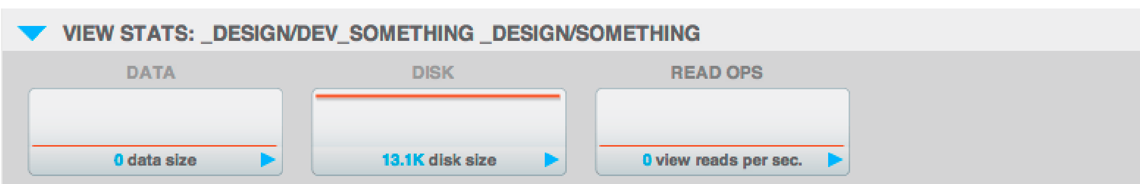
Delete operations per second as a result of the incoming XDCR data stream.

- `total ops per sec.`

Total of all the operations per second.

Monitoring view statistics

The View statistics show information about individual design documents within the selected bucket. One block of stats will be shown for each production-level design document.



The statistics shown are:

- `data size`

Size of the data required for this design document.

- `disk size`

Size of the stored index as stored on disk.

- `view reads per sec.`

Number of read operations per second for this view.

Related Links

Data Buckets on page 259

Related topics

Getting bucket statistics on page 458

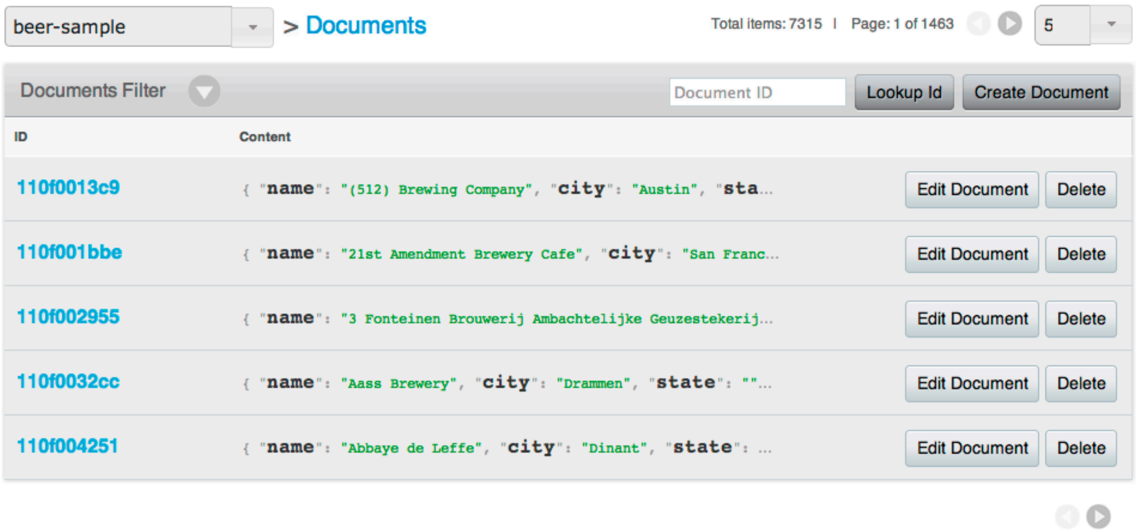
To retrieve bucket statistics, use the `GET` operation with the `/pools/default/buckets/bucket_name/stats` URI.

Getting XDCR stats via REST on page 490

Managing documents

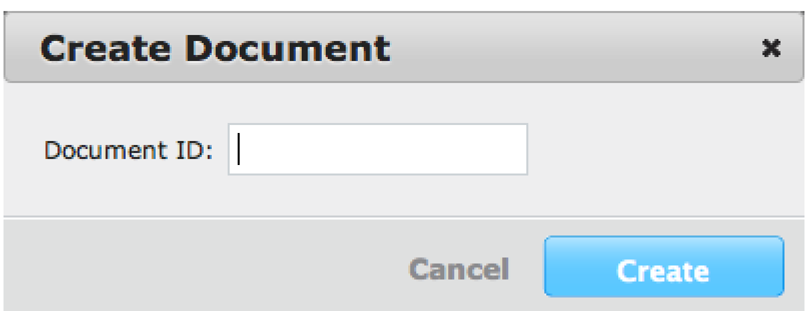
The Document Viewer and Editor enables you to browse, view, and edit individual documents stored in Couchbase Server buckets.

To get to the Documents editor, click on the **Documents** button within the Data Buckets view. The opens a list of available documents. Only a selection of the available documents are displayed rather than all documents. The maximum size of editable documents is 2.5 KB.



Select a different Bucket by using the bucket selection popup on the left. Page through the list of documents shown by using the navigation arrows on the right. To jump to a specific document ID, enter the ID in the box provided and click `Lookup Id`. To edit an existing document, click the **Edit Document** button. To delete the document from the bucket, click **Delete**.

To create a new document, click the **Create Document** button. This opens a prompt to specify the document Id of the created document.



Once the document Id has been set, the document editor displays. The document editor is also opened when the document ID within the document list is selected. To edit the contents of the document, use the textbox to modify the JSON of the stored document.

Within the document editor, click **Delete** to delete the current document, **Save As...** copies the currently displayed information and create a new document with the document Id you specify. The **Save** saves the current document and return you to the list of documents.

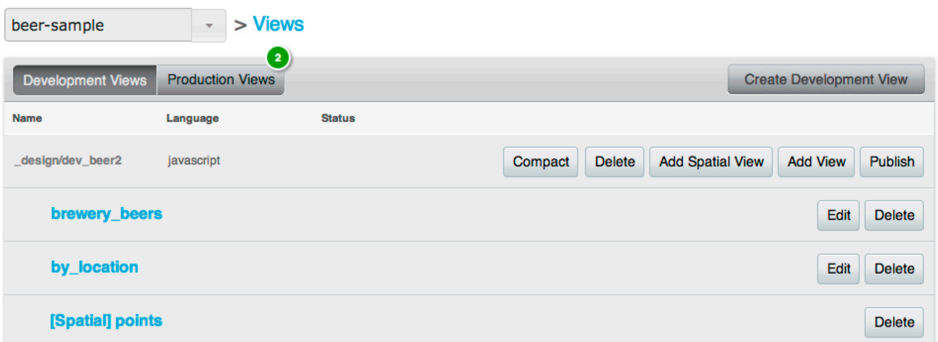
Related Links

Data Buckets on page 259

Views

The Views section allows you to manage your development and production views.

The Views Editor is available within the Couchbase web console. You can access the View Editor either by clicking the `Views` for a given data bucket within the `Data Buckets` display, or by selecting the `Views` page from the main navigation panel.



The individual elements of this interface are:

- The pop-up, at the top-left, provides the selection of the data bucket where you are viewing or editing a view.
- The `Create Development View` enables you to create a new view either within the current design document, or within a new document.
- You can switch between Production Views and Development Views.
- The final section provides a list of the design documents, and within each document, each defined view.

When viewing `Development Views`, you can perform the following actions:

```
* `Compact` the view index with an associated design document. This will
 compact
  the view index and recover space used to store the view index on disk.

* `Delete` a design document. This deletes all of the views defined within the
  design document.

* `Add Spatial View` creates a new spatial view within the corresponding
 design
  document. See [Creating and Editing Views](#couchbase-views-editor-
createedit).

* `Add View` creates a new view within the corresponding design document. See
  [Creating and Editing Views](#couchbase-views-editor-createedit).

* `Publish` your design document (and all of the defined views) as a
 production
  design document. See [Publishing Views](#couchbase-views-editor-publishing).

* For each individual view listed:

   * `Edit`, or clicking the view name

     Opens the view editor for the current view name, see [Creating and
 Editing
```

```
     Views](#couchbase-views-editor-createedit).

   * `Delete`

     Deletes an individual view.
```

When viewing `Production Views` you can perform the following operations on each design document:

```
* `Compact` the view index with an associated design document. This will
 compact
  the view index and recover space used to store the view index on disk.

* `Delete` a design document. This will delete all of the views defined within
 the
  design document.

* `Copy to Dev` copies the view definition to the development area of the view
  editor. This enables you edit the view definition. Once you have finished
 making
  changes, using the `Publish` button will then overwrite the existing view
  definition.

* For each individual view:

   * By clicking the view name, or the `Show` button, execute and examine the
 results
     of a production view. See [Getting View Results](#couchbase-views-editor-
view)
     for more information.
```

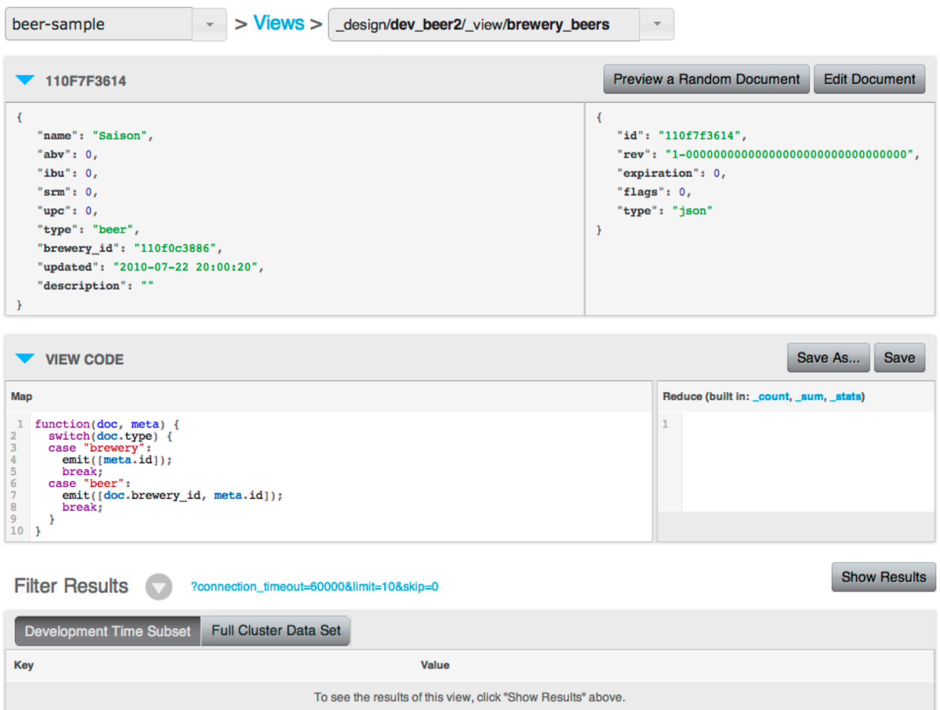
Creating and editing views

You can create a new design document and/or view by clicking the `Create Development View` button within the `Views` section of the Web Console. If you are creating a new design document and view you will be prompted to supply both the design document and view name.

To create a new view as part of an existing design document, click the `Add View` button against the corresponding design document.

View names must be specified using one or more UTF–8 characters. You cannot have a blank view name. View names cannot have leading or trailing whitespace characters (space, tab, newline, or carriage-return).

If you create a new view, or have selected a Development view, you can create and edit the `map()` and `reduce()` functions. Within a development view, the results shown for the view are executed either over a small subset of the full document set (which is quicker and places less load on the system), or the full data set.



>

The top portion of the interface provides navigation between the available design documents and views.

The `Sample Document` section allows you to view a random document from the database to help you write your view functions and so that you can compare the document content with the generated view output. Clicking the `Preview a Random Document` will randomly select a document from the database. Clicking **Edit Document** takes you to the Views editor.

Documents stored in the database that are identified as Non-JSON may be displayed as binary, or text-encoded binary, within the UI.

Document metadata is displayed in a separate box on the right hand side of the associated document. This shows the metadata for the displayed document, as supplied to the `map()` as the second argument to the function. For more information on writing views and creating the `map()` and `reduce()` functions.

With the View Code section, you should enter the function that you want to use for the `map()` and `reduce()` portions of the view. The map function is required, the reduce function is optional. When creating a new view a basic `map()` function will be provided. You can modify this function to output the information in your view that you require.

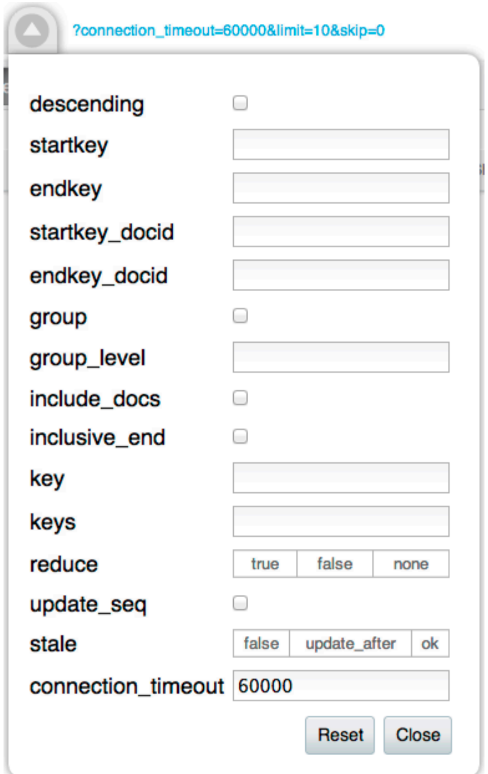
Once you have edited your `map()` and `reduce()` functions, you must use the `Save` button to save the view definition.

The design document will be validated before it is created or updated in the system. The validation checks for valid JavaScript and for the use of valid built-in reduce functions. Any validation failure is reported as an error.

You can also save the modified version of your view as a new view using the `Save As...` button.

The lower section of the window will show you the list of documents that would be generated by the view. You can use the `Show Results` to execute the view.

To execute a view and get a sample of the output generated by the view operation, click the `Show Results` button. This will create the index and show the view output within the table below. You can configure the different parameters by clicking the arrow next to `Filter Results`. This shows the view selection criteria, as seen in the figure below.



Clicking on the `Filter Results` query string opens a new window containing the raw, JSON formatted, version of the View results.

By default, Views during the development stage are executed only over a subset of the full document set. This is indicated by the `Development Time Subset` button. You can execute the view over the full document set by selecting `Full Cluster Data Set`. Because this executes the view in real-time on the data set, the time required to build the view may be considerable. Progress for building the view is shown at the top of the window.

If you have edited either the `map()` or `reduce()` portions of your view definition, you *must* save the definition. The `Show Results` button will remain greyed out until the view definition has been saved.

You can also filter the results and the output using the built-in filter system. This filter provides similar options that are available to clients for filtering results.

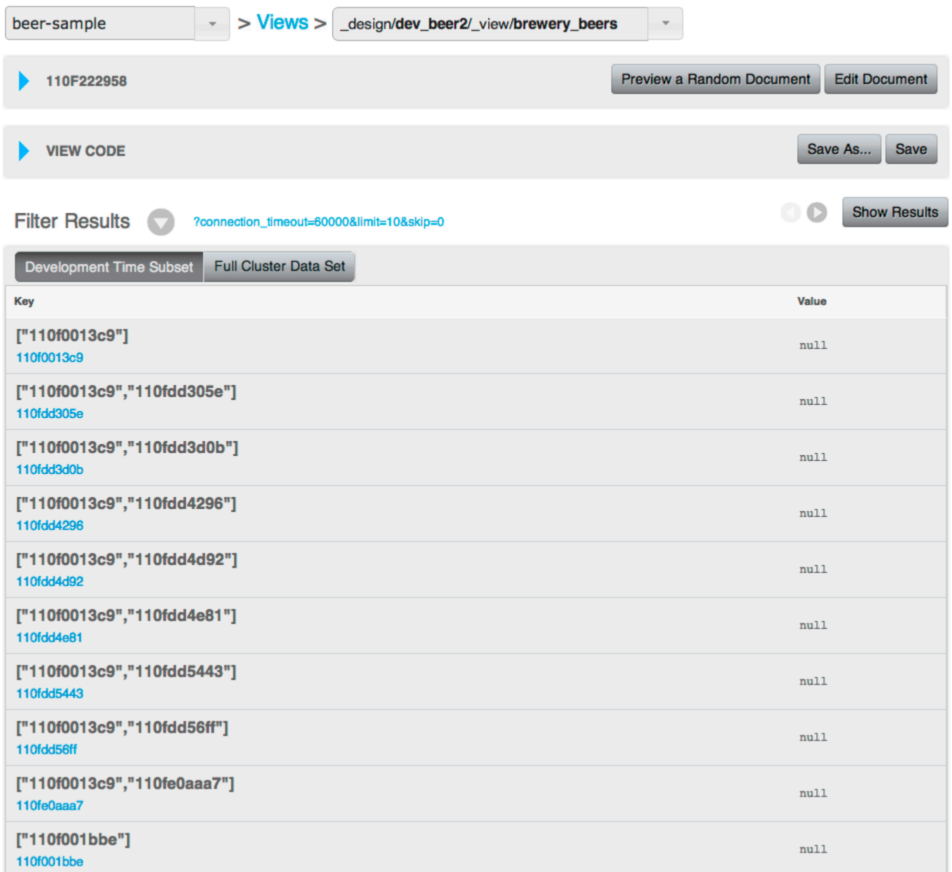
Publishing views

Publishing a view moves the view definition from the Development view to a Production View. Production views cannot be edited. The act of publishing a view and moving the view from the development to the production view will overwrite a view the same name on the production side. To edit a Production view, you copy the view from production to development, edit the view definition, and then publish the updated version of the view back to the production side.

Getting view results

Once a view has been published to be a production view, you can examine and manipulate the results of the view from within the web console view interface. This makes it easy to study the output of a view without using a suitable client library to obtain the information.

To examine the output of a view, click icon next to the view name within the view list. This will present you with a view similar to that shown in the figure below.



The top portion of the interface provides navigation between the available design documents and views.

The `Sample Document` section allows you to view a random document from the database so that you can compare the document content with the generated view output. Clicking the `Preview a Random Document` will randomly select a document from the database. If you know the ID of a document that you want to examine, enter the document ID in the box, and click the `Lookup Id` button to load the specified document.

To examine the function that generate the view information, use the `View Code` section of the display. This will show the configured map and reduce functions.

The lower portion of the window will show you the list of documents generated by the view. You can use the `Show Results` to execute the view.

The `Filter Results` interface allows you to query and filter the view results by selecting the sort order, key range, or document range, and view result limits and offsets.

To specify the filter results, click on the pop-up triangle next to `Filter Results`. You can delete existing filters, and add new filters using the embedded selection windows. Click `Show Results` when you have finished selecting filter values. The filter values you specify are identical to those available when querying from a standard client library.

Due to the nature of range queries, a special character may be added to query specifications when viewing document ranges. The character may not show up in all web browsers, and may instead appear instead as an invisible, but selectable, character.

Related Links

Couchbase web console on page 249
The Couchbase web console is the main tool for managing the Couchbase environment.

Related topics

Views and indexes on page 183

Writing views on page 196

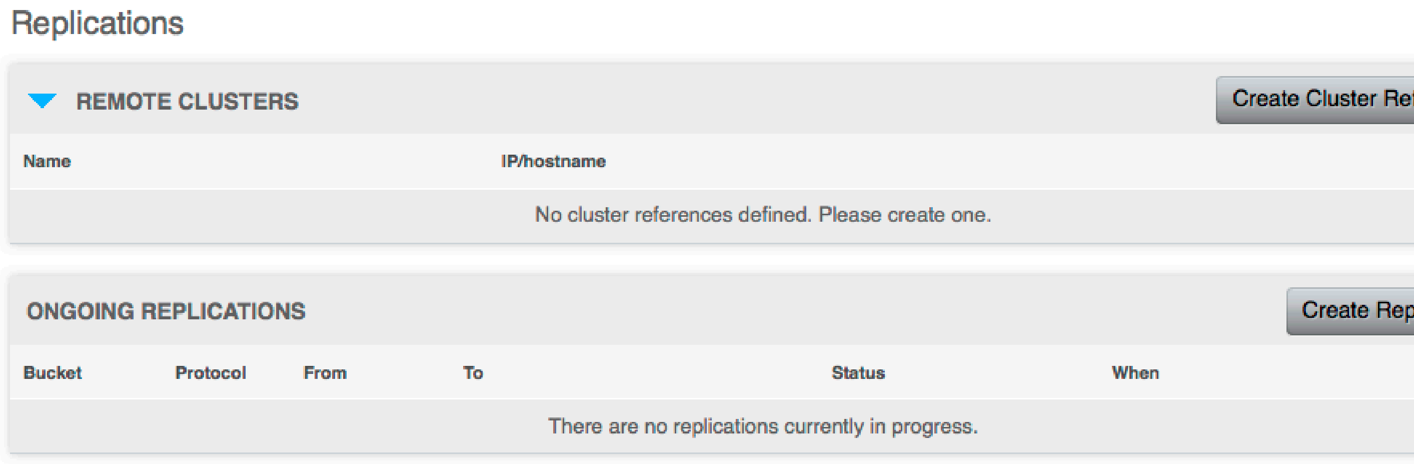
Managing design documents on page 475

Design documents are used to store one or more view definitions. Views can be defined within a design document and uploaded to the server through the REST API.

Views REST API on page 475

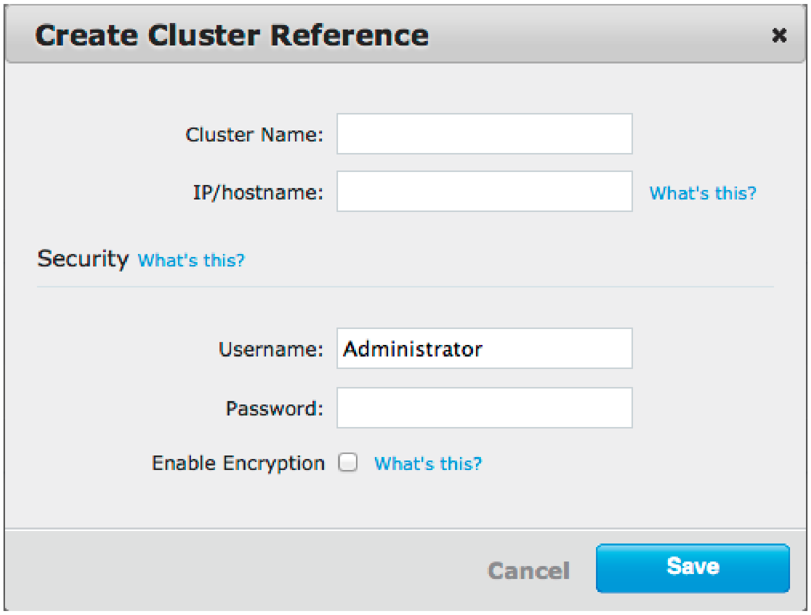
XDCR

The XDCR panel is used to create a remote cluster reference and specify replication.



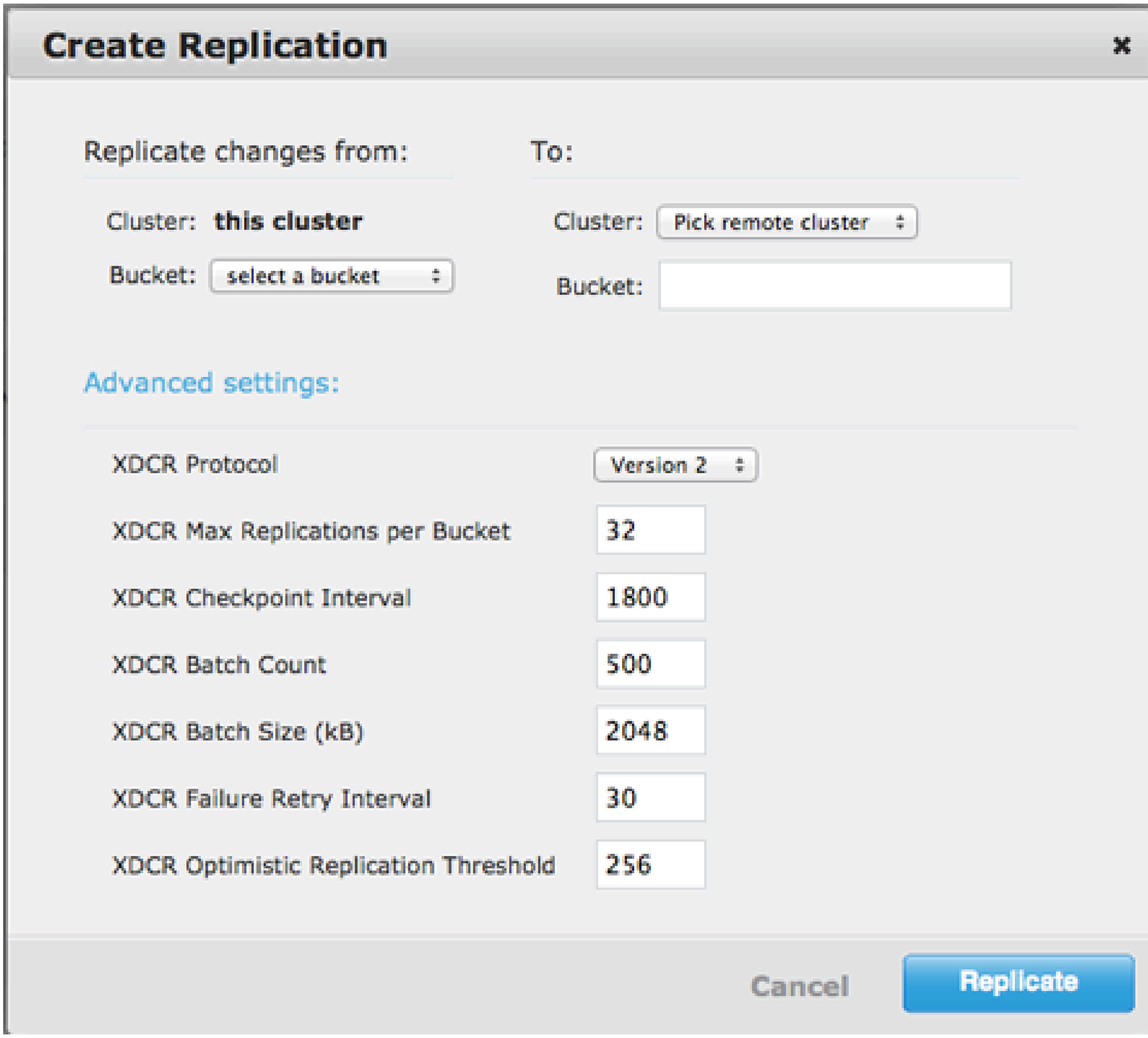
Create cluster reference

To create a cluster reference, provide the cluster name, IP or hostname, administrator username and password and whether or not to enable encryption (Enterprise Edition only).



Create replication

To create a replication, provide the bucket to replicaed on the remote cluster, the remote cluster and bucket, and modify any advanced XDCR settings.



Related Links

Couchbase web console on page 249
The Couchbase web console is the main tool for managing the Couchbase environment.

Creating replications on page 285
After references to the source and destination clusters are created, replication can be created between the clusters.

Specifying a destination cluster on page 285
The destination cluster for XDCR replication is specified via **XDCR > Remote Cluster > Create Cluster Reference**.

Specifying XDCR settings on page 286
When creating a new replication, advanced settings are modified (from the default) via **XDCR > Ongoing Replications > Create Replication**.

Managing XDCR data encryption on page 289

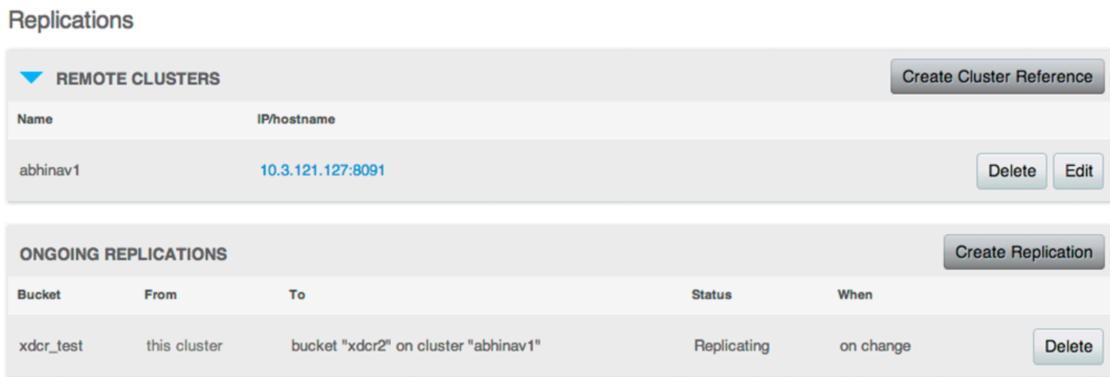
Monitoring replication status on page 293
Replication status is monitored via the **XDCR** and **Data Buckets** tabs.

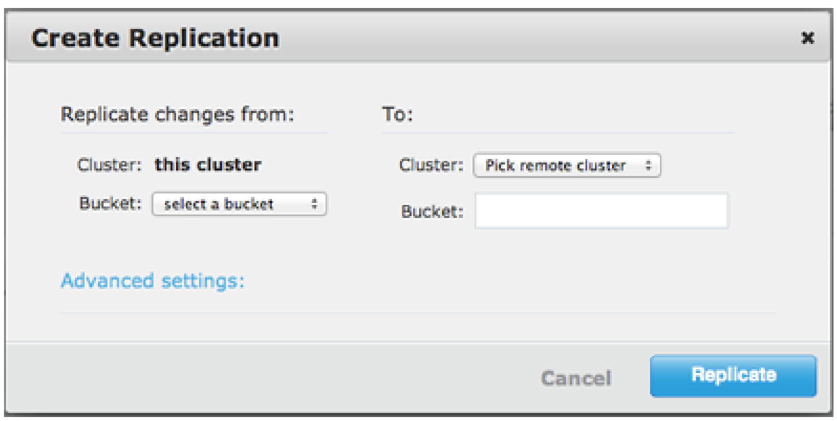
Canceling replication on page 293
To cancel the replication, delete the active replication.

Creating replications

After references to the source and destination clusters are created, replication can be created between the clusters.

After replication has been configured and started, view current status and list of replications in the **Ongoing Replications** section.





1. Click **Create Replication** to configure a new XDCR replication. A panel appears where you can configure a new replication from source to destination cluster.
2. In the **Replicate changes from** section select a from the current cluster that is to be replicated. This is your source bucket.
3. In the **To** section, select a destination cluster and enter a bucket name from the destination cluster:
4. Click the `Replicate` button to start the replication process.

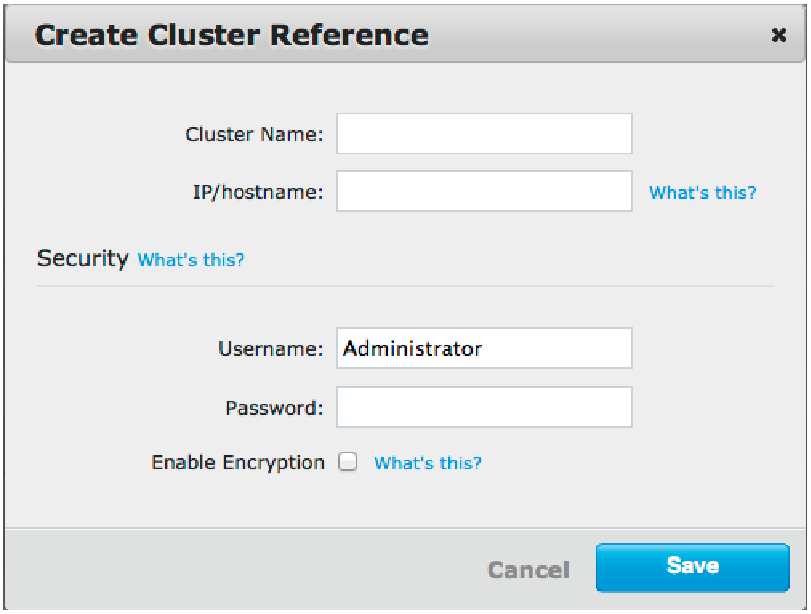
Related Links

XDCR on page 283
The XDCR panel is used to create a remote cluster reference and specify replication.

Specifying a destination cluster

The destination cluster for XDCR replication is specified via **XDCR > Remote Cluster > Create Cluster Reference**.

To create a uni-directional replication (from cluster A to cluster B):



1. Verify that a destination bucket exists on the cluster where you will be replicating.

   To check that a destination bucket exists, issue a REST API request using the following syntax, GET HTTP method, and URI path:

   ```
   curl GET -u Admin:password http://ip.for.destination.cluster:8091/pools/
   default/buckets
   ```

2. Navigate to the XDCR section, **XDCR > Remote Cluster section > Create Cluster Reference**.
3. Click the **Create Cluster Reference** button.
4. Provide the following information for identifying and accessing the destination cluster.
   - Cluster Name
   - IP address or hostname of a node in the destination cluster
   - Administrator username and password for the destination cluster
   - Enable Encryption - If selected, XDCR data encryption occurs using SSL.
5. Click **Save** to store new reference to the destination cluster. This cluster information is now be available when configuring replication for the source cluster.

Related Links

XDCR on page 283
The XDCR panel is used to create a remote cluster reference and specify replication.

Specifying XDCR settings

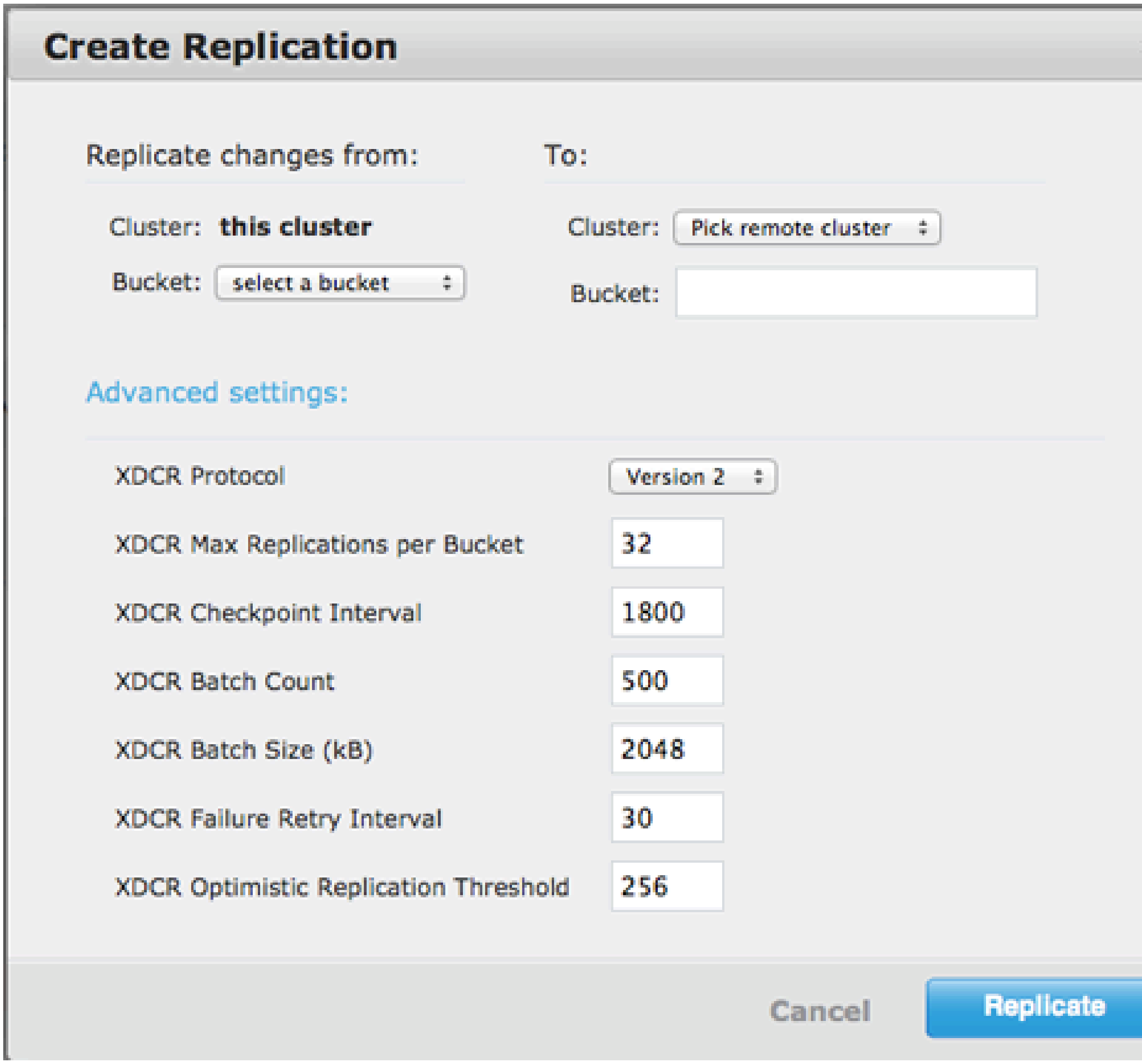
When creating a new replication, advanced settings are modified (from the default) via **XDCR > Ongoing Replications > Create Replication**.

To change the replication protocol for an existing XDCR replication, delete the replication and then re-create the replication with your preference.

> **Note:** How you adjust these variables differs based on what whether you want to perform unidirectional or bidirectional replication between clusters. Other factors for consideration include intensity of read/write operations on your clusters, the rate of disk persistence on your destination cluster, and your system environment. Changing these parameters impacts cluster performance and XDCR replication performance.

1. Navigate to **XDCR > Ongoing Replications > Create Replication**.

2. In the **Create Replication** panel, click **Advanced Settings**.



3. Under **Advanced settings**, choose an XDCR Protocol version. The XDCR protocol defaults to version 2.
   - Version 1 uses the REST protocol for replication. This increases XDCR throughput at destination clusters. If the Elasticsearch plug-in used, choose version 1 because it 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.
4. Change the XDCR settings. These settings plus additional internal settings can be modified via the REST API.
5. Click **Replicate**.
6. After creating the replication or updating the setting, view or edit them by clicking **Settings** in Outgoing Replications.

Related Links

XDCR on page 283
The XDCR panel is used to create a remote cluster reference and specify replication.

XDCR advanced settings on page 288
XDCR advanced settings are internal settings that are available for configuration.

Related topics

XDCR advanced settings on page 288
XDCR advanced settings are internal settings that are available for configuration.

XDCR REST API on page 483
Cross Datacenter Replication (XDCR) enables you to automatically replicate data between clusters and between data buckets.

XDCR advanced settings

XDCR advanced settings are internal settings that are available for configuration.

Advanced settings that can be updated include:

XDCR Max Replications per Bucket	Maximum concurrent replications per bucket, 8 to 256. This controls the number of parallel replication streams per node. If you are running your cluster on hardware with high-performance CPUs, you can increase this value to improve replication speed.
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. This time interval determines the volume of data that will be replicated via XDCR should replication need to restart. The greater this value, the longer amount of time transpires for XDCR queues to grow. For example, if you set this to 10 minutes and a network error occurs, when XDCR restarts replication, 10 minutes of items will have accrued for replication. 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. Note however that this can have a negative impact on the destination cluster if you are performing bidirectional replication between two clusters and the destination already handles a significant volume of reads/writes.
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. 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. Note however that this can have a negative impact on the destination cluster if you are performing bidirectional replication between two clusters and the destination already handles a significant volume of reads/writes.
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 is compressed 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 uncompressed document to a destination cluster. This option improves XDCR latency.

Related Links

Specifying XDCR settings on page 286
When creating a new replication, advanced settings are modified (from the default) via **XDCR > Ongoing Replications > Create Replication**.

Managing XDCR data encryption

The cross data center (XDCR) data security feature (Enterprise Edition only) provides secure cross data center replication using Secure Socket Layer (SSL) data encryption. The data replicated between clusters can be encrypted in both uni-directional and bi-directional replications.

XDCR data encryption prerequisites

- Couchbase servers on both source and destination clusters must have Couchbase 2.5 Enterprise Edition and above installed.
- The source cluster must use the destination cluster's certificate. The certificate is a self-signed certificate used by SSL to initiate secure sessions.
- The reserved ports for XDCR data encryption must be available.

Important

Ensure that the Secure Socket Layer (SSL) reserved ports are available prior to using XDCR data encryption. Otherwise, XDCR data encryption is unavailable.

With XDCR data encryption, the following ports are reserved:

Port	Description
11214	Incoming SSL Proxy
11215	Internal Outgoing SSL Proxy
18091	Internal REST HTTPS for SSL
18092	Internal CAPI HTTPS for SSL

To enable XDCR data security

To enable XDCR data security using SSL and create replication:

1. On the destination cluster, navigate to Settings > Cluster and copy the certificate.
   - (Optional) To regenerate the existing destination certificate, click **Regenerate** before copying.
2. On the source cluster, select the XDCR tab.
3. On the Remote Clusters panel, click **Create Cluster Reference** to verify or create the cluster reference.
4. Select the **Enable Encryption** box and paste the certificate in the provided area and click **Save**.
5. On the Ongoing Replications panel, click **Create Replication**, provide the cluster and bucket information, and click **Replicate**.

To change XDCR data encryption

In some situations (such as updating SSL data security), the SSL certificate is regenerated and the XDCR data encryption is updated. To change XDCR data encryption:

1. On the destination cluster, navigate to Settings > Cluster.
2. Click **Regenerate** and copy the certificate.
3. On the source cluster, select the XDCR tab.
4. On the **Remote Clusters** panel, for the destination cluster, click **Edit**.
5. Paste the regenerated certificate in the provided area and click **Save**.

Anytime a destination cluster’s certificate is regenerated, the corresponding source cluster(s) must be updated with the regenerated certificate.

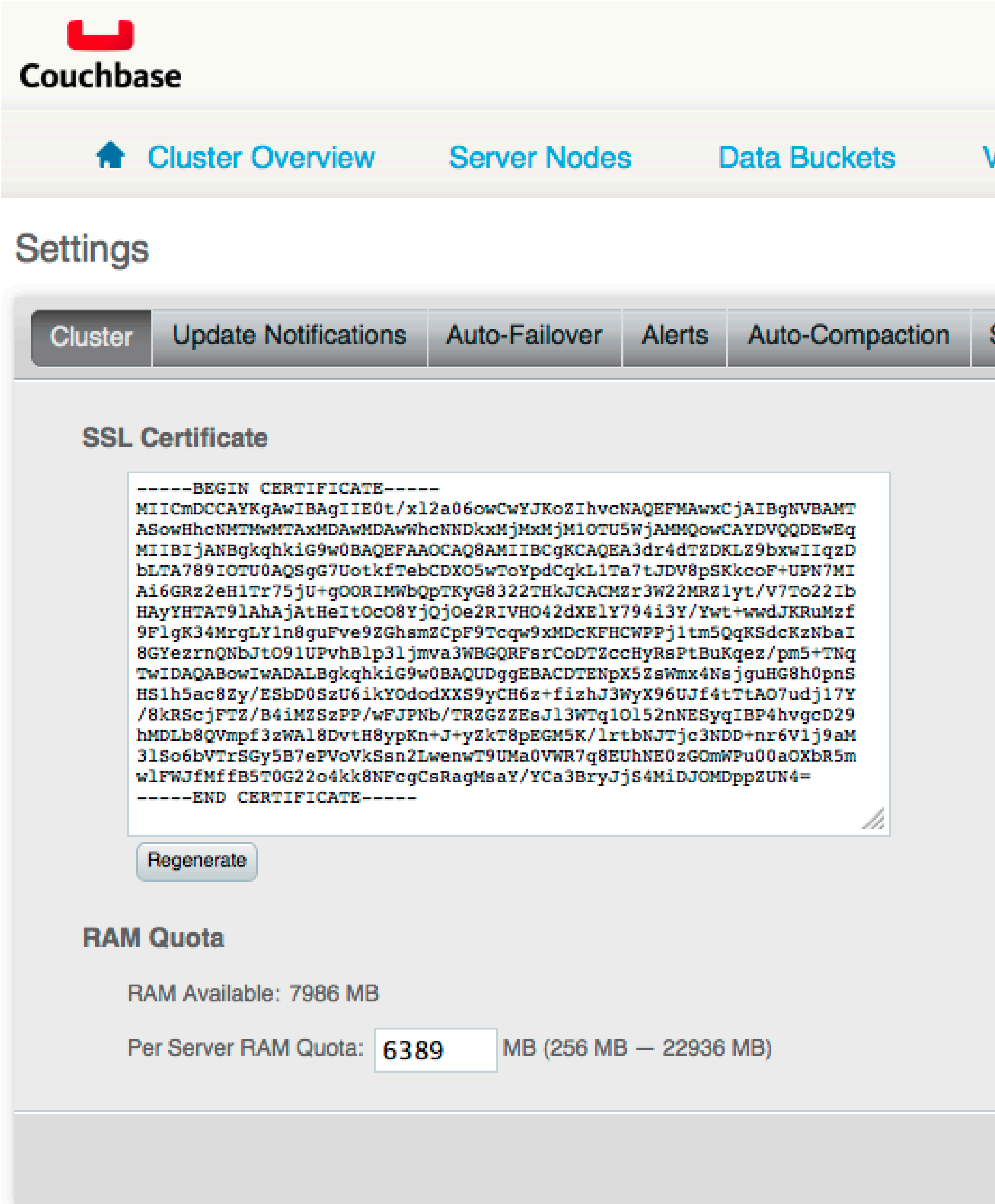
For example, if source clusters A, B, and C use XDCR data encryption to replicate to destination cluster D, each of the source clusters must be updated whenever the certificate on the destination cluster D is regenerated (changed).

Important

If a destination cluster's certificate is regenerated and the source cluster(s) are not updated with the new certificate, replication stops.

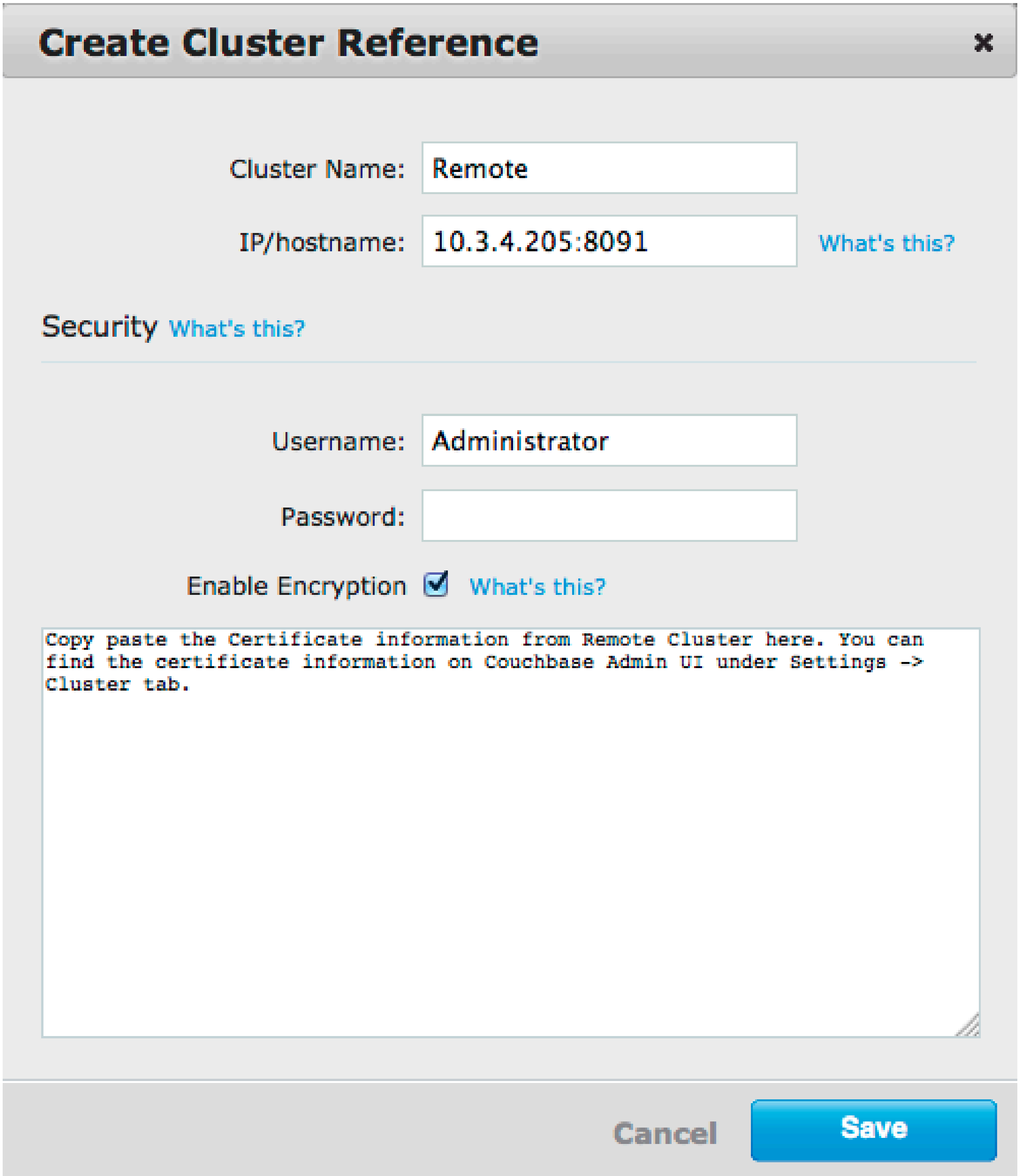
SSL certificate

The following is an example of an SSL certificate and where the certificate is obtained on the cluster.


Couchbase
Cluster Overview
Server Nodes
Data Buckets
Settings
Cluster
Update Notifications
Auto-Failover
Alerts
Auto-Compaction
SSL Certificate
-----BEGIN CERTIFICATE-----
MIICmDCCAYKgAwIBAgIIE0t/xl2a06owCwYJKoZIhvcNAQEFMAwxCjAIBgNVBAMT
ASowHhcNMTMwMTAxMDAwMDAwWhcNNDkxMjMxMjM1OTU5WjAMMQowCAYDVQQDEwEq
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA3dr4dTZDKLZ9bxwIIqzD
bLTA789IOTU0AQSgG7UotkfTebCDXO5wToYpdCqkL1Ta7tJDV8pSKkcoF+UPN7MI
Ai6GRz2eH1Tr75jU+gOORIMWbQpTKyG8322THkJCACMZr3W22MRZ1yt/V7To22Ib
HAyYHTAT9lAhAjAtHeItOcO8YjQjOe2RIVHO42dXElY794i3Y/Ywt+wwdJKRuMzf
9FlgK34MrgLY1n8guFve9ZGhsmZCpF9Tcqw9xMDcKFHCWPPj1tm5QqKSdcKzNbaI
8GYezrnQNbJtO91UPvhBlp3ljmva3WBGQRFsrCoDTZccHyRsPtBuKqez/pm5+TNq
TwIDAQABowIwADALBgkqhkiG9w0BAQUDggEBACDTENpX5ZsWmx4NsjguHG8h0pnS
HS1h5ac8Zy/ESbD0SzU6ikYOdodXXS9yCH6z+fizhJ3WyX96UJf4tTtAO7udj17Y
/8kRScjFTZ/B4iMZSzPP/wFJPNb/TRZGZZEsJl3WTq1Ol52nNESyqIBP4hvgcD29
hMDLb8QVmpf3zWAl8DvtH8ypKn+J+yZkT8pEGM5K/lrtbNJTjc3NDD+nr6V1j9aM
3lSo6bVTrSGy5B7ePVoVkSsn2LwenwT9UMa0VWR7q8EUhNE0zGOmWPu00aOXbR5m
wlFWJfMffB5T0G22o4kk8NFcgCsRagMsaY/YCa3BryJjS4MiDJOMDppZUN4=
-----END CERTIFICATE-----
Regenerate
RAM Quota
RAM Available: 7986 MB
Per Server RAM Quota:
6389
MB (256 MB — 22936 MB)

Create Cluster Reference

The following is an example of the Create Cluster Reference pop-up.



XDCR data security error messages

When creating the cluster reference, if the SSL certificates are not the same on the destination and source clusters, the following error message displays:

```
Attention - Got certificate mismatch while trying to send https request to
 HOST:18091
```

If the SSL certificates become mismatched (for example, if the certificate on the destination cluster is regenerated and the source cluster is not updated with the new certificate), vBucket replication stops and the following error message displays:

```
Error replicating vbucket <bucketNumber>. Please see logs for details.
```

Related Links

XDCR on page 283
The XDCR panel is used to create a remote cluster reference and specify replication.

Monitoring replication status

Replication status is monitored via the **XDCR** and **Data Buckets** tabs.

The following Couchbase Web Console areas contains information about replication via XDCR:

- The XDCR tab.
- The outgoing XDCR section under the Data Buckets tab.

The Couchbase Web Console displays replication from the cluster it belongs to. Therefore, when you view the console from a particular cluster, it will display any replications configured, or replications in progress for that particular source cluster. If you want to view information about replications at a destination cluster, you need to open the console at that cluster. Therefore, when configuring bidirectional, use the web consoles that belong to the source and destination clusters to monitor both clusters.

Any errors that occur during replication appear in the XDCR errors panel. The following example shows the errors that occur if replication streams from XDCR fail due to the missing vBuckets:

Related Links

XDCR on page 283
The XDCR panel is used to create a remote cluster reference and specify replication.

Canceling replication

To cancel the replication, delete the active replication.

Canceled replications that were terminated while the replication was still active are displayed within the **Past Replications** section of **Replications**.

1. From the XDCR section, click **Delete** next to the active replication that is to be canceled.
2. Confirm the deletion of the configured replication. Once the replication has been stopped, replication ceases on the originating cluster on a document boundary.

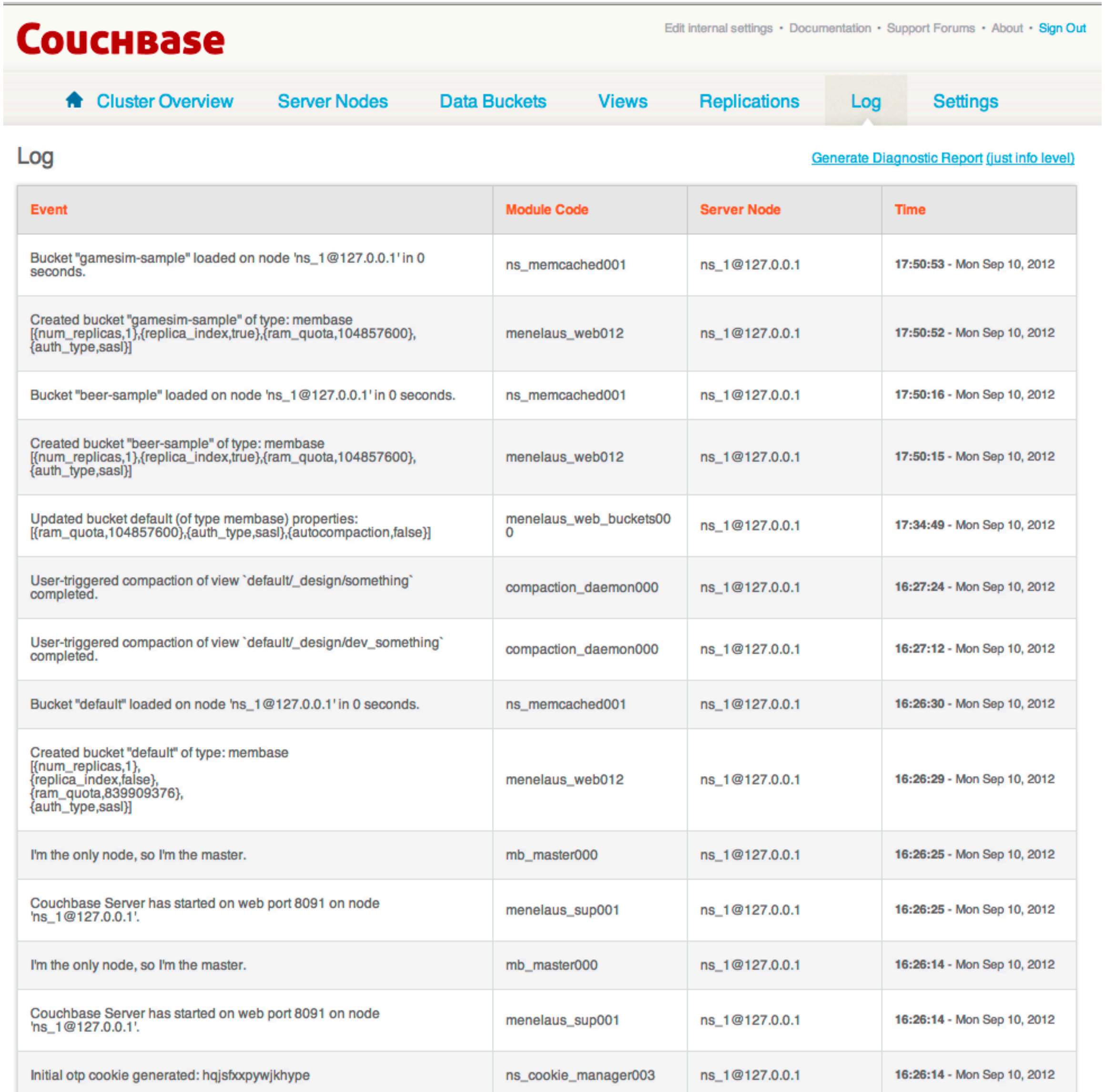
Related Links

XDCR on page 283
The XDCR panel is used to create a remote cluster reference and specify replication.

Log

The Log section provides a built-in event log for Couchbase Server.

The event log allows you to identify activity and errors within the Couchbase cluster.



Event	Module Code	Server Node	Time
Bucket "gamesim-sample" loaded on node 'ns_1@127.0.0.1' in 0 seconds.	ns_memcached001	ns_1@127.0.0.1	17:50:53 - Mon Sep 10, 2012
Created bucket "gamesim-sample" of type: membase [{num_replicas,1},{replica_index,true},{ram_quota,104857600},{auth_type,sasl}]	menelaus_web012	ns_1@127.0.0.1	17:50:52 - Mon Sep 10, 2012
Bucket "beer-sample" loaded on node 'ns_1@127.0.0.1' in 0 seconds.	ns_memcached001	ns_1@127.0.0.1	17:50:16 - Mon Sep 10, 2012
Created bucket "beer-sample" of type: membase [{num_replicas,1},{replica_index,true},{ram_quota,104857600},{auth_type,sasl}]	menelaus_web012	ns_1@127.0.0.1	17:50:15 - Mon Sep 10, 2012
Updated bucket default (of type membase) properties: [{ram_quota,104857600},{auth_type,sasl},{autocompaction,false}]	menelaus_web_buckets000	ns_1@127.0.0.1	17:34:49 - Mon Sep 10, 2012
User-triggered compaction of view `default/_design/something` completed.	compaction_daemon000	ns_1@127.0.0.1	16:27:24 - Mon Sep 10, 2012
User-triggered compaction of view `default/_design/dev_something` completed.	compaction_daemon000	ns_1@127.0.0.1	16:27:12 - Mon Sep 10, 2012
Bucket "default" loaded on node 'ns_1@127.0.0.1' in 0 seconds.	ns_memcached001	ns_1@127.0.0.1	16:26:30 - Mon Sep 10, 2012
Created bucket "default" of type: membase [{num_replicas,1}, {replica_index,false}, {ram_quota,839909376}, {auth_type,sasl}]	menelaus_web012	ns_1@127.0.0.1	16:26:29 - Mon Sep 10, 2012
I'm the only node, so I'm the master.	mb_master000	ns_1@127.0.0.1	16:26:25 - Mon Sep 10, 2012
Couchbase Server has started on web port 8091 on node 'ns_1@127.0.0.1'.	menelaus_sup001	ns_1@127.0.0.1	16:26:25 - Mon Sep 10, 2012
I'm the only node, so I'm the master.	mb_master000	ns_1@127.0.0.1	16:26:14 - Mon Sep 10, 2012
Couchbase Server has started on web port 8091 on node 'ns_1@127.0.0.1'.	menelaus_sup001	ns_1@127.0.0.1	16:26:14 - Mon Sep 10, 2012
Initial otp cookie generated: hqjsfxxpywjkhype	ns_cookie_manager003	ns_1@127.0.0.1	16:26:14 - Mon Sep 10, 2012

Related Links

Couchbase web console on page 249
The Couchbase web console is the main tool for managing the Couchbase environment.

Settings

The **Settings** section provides configuration and information for the cluster, update notifications, auto failover, alerts, auto compaction, sample buckets, and account management.

The `Settings` interface sets the global settings for your Couchbase Server instance.

Related Links

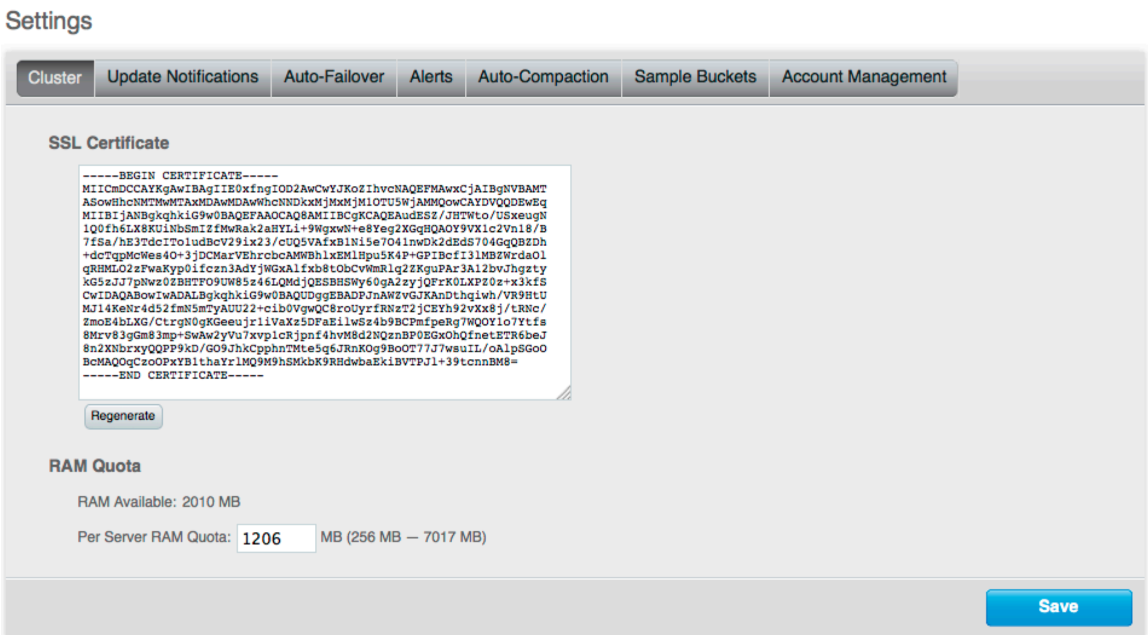
Couchbase web console on page 249
The Couchbase web console is the main tool for managing the Couchbase environment.

Cluster tab on page 295
Update Notifications tab on page 295
Auto-Failover tab on page 296
Alerts tab on page 297

Auto-Compaction tab on page 300
Sample Buckets tab on page 301
Account Management tab on page 302

Cluster tab

Cluster settings show the available RAM on your cluster and the per server RAM quota. The Per Server RAM Quota is adjustable. In addition, if you have the Enterprise Edition, a Couchbase Server self-signed SSL certification is provided to set up secure communication in an XDCR environment. The SSL certificate can be regenerated.



Related Links

Settings on page 294
The **Settings** section provides configuration and information for the cluster, update notifications, auto failover, alerts, auto compaction, sample buckets, and account management.

Related topics

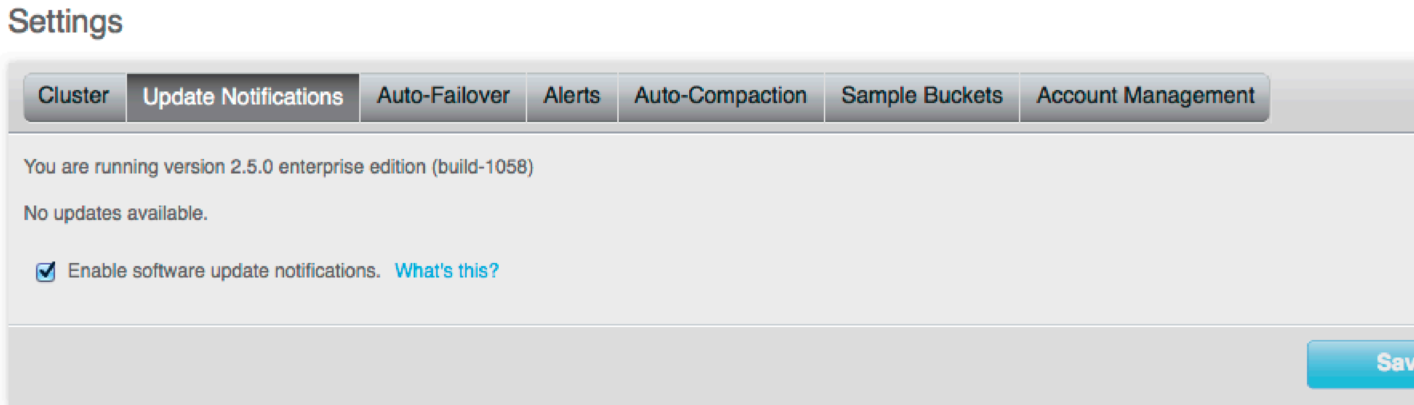
RAM quotas on page 99
RAM is allocated to Couchbase Server in the following configurable quantities: *Server Quota* and *Bucket Quota*.

XDCR data encryption on page 247
The cross data center (XDCR) data security feature provides secure cross data center replication using Secure Socket Layer (SSL) data encryption.

Update Notifications tab

You can enable or disable Update Notifications by checking the `Enable software update notifications` checkbox within the `Update Notifications` screen. Once you have changed the option, click `Save` to record the change.

If update notifications are disabled, then the Update Notifications screen only notifies you of your currently installed version, and no alert is provided.



During installation you can select to enable the Update Notification function. Update notifications allow a client accessing the Couchbase Web Console to determine whether a newer version of Couchbase Server is available for download.

If `Enable software update notifications` is selected, the web console communicates with Couchbase servers to confirm the version number of your Couchbase installation. During this process, the client submits the following information to the Couchbase server:

- The current version of your Couchbase Server installation. When a new version of Couchbase Server becomes available, you are provided with notification of the new version and information on where to download the new version.
- Basic information about the size and configuration of your Couchbase cluster. This information is used to help Couchbase prioritize development efforts.

You can enable/disable software update notifications. The process occurs within the browser accessing the web console, not within the server itself. No further configuration or internet access is required on the server to enable this functionality. Providing the client accessing the Couchbase server console has internet access, the information can be communicated to the Couchbase servers.

The update notification process the information anonymously, and the data cannot be tracked. The information is only used to provide you with update notification and to provide information that will help improve future development process for Couchbase Server and related products.

Note: If the browser or computer that you are using to connect to your Couchbase Server web console does not have Internet access, the update notification system does not work.

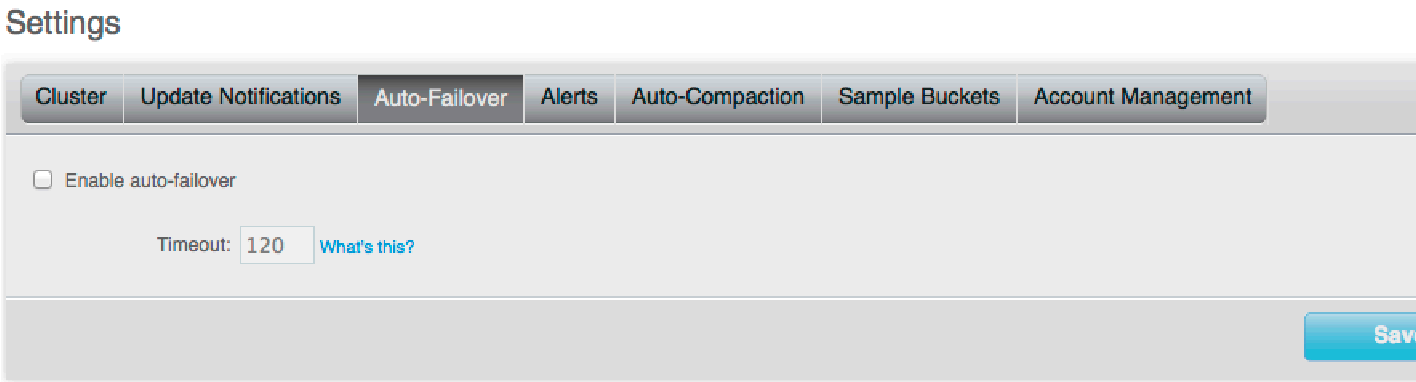
Related Links

Settings on page 294
The **Settings** section provides configuration and information for the cluster, update notifications, auto failover, alerts, auto compaction, sample buckets, and account management.

Auto-Failover tab

The Auto-Failover settings enable auto-failover. The timeout before the auto-failover process is started when a cluster node failure is detected.

To enable Auto-Failover, check the `Enable auto-failover` checkbox. To set the delay in seconds before auto-failover is started, enter the number of seconds in the `Timeout` box. The default timeout is 120 seconds.



Related Links

Settings on page 294
The **Settings** section provides configuration and information for the cluster, update notifications, auto failover, alerts, auto compaction, sample buckets, and account management.

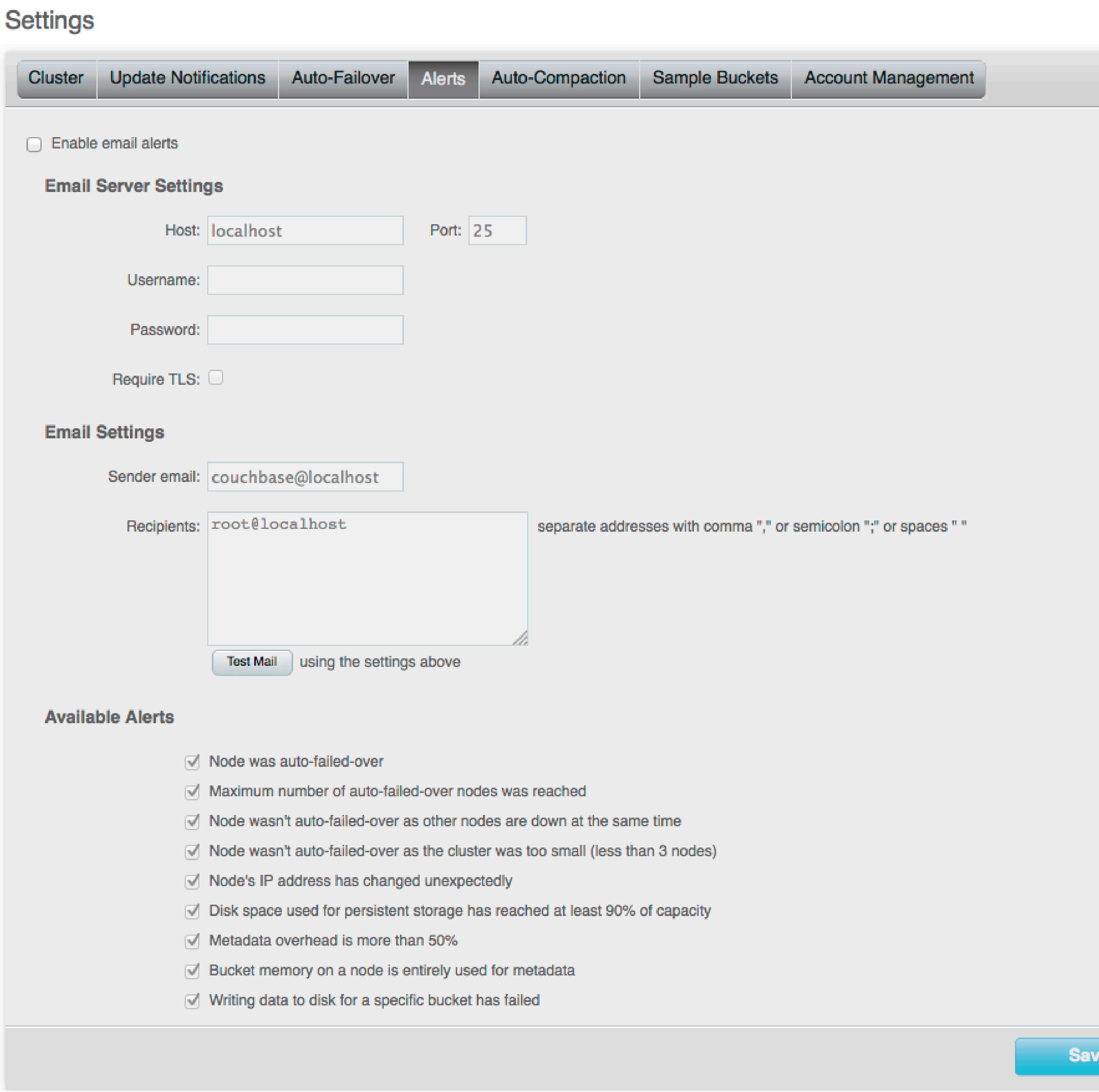
Related topics

Failing over nodes on page 141
Failing over a node means that Couchbase Server removes the node from a cluster and makes replicated data at other nodes available for client requests.

Alerts tab

You can enable email alerts to be raised when a significant error occurs on your Couchbase Server cluster. The email alert system works by sending email directly to a configured SMTP server. Each alert email is send to the list of configured email recipients. This is used to highlight specific issues and problems that you should be aware of and may need to check to ensure the health of your Couchbase cluster. Alerts are provided as a popup within the web console.

Select **Enable email alerts** to configure email alerts including the server settings and recipient information. Email alerts are raised for the errors selected in the Available Alerts section.



Email Server Settings

The available settings are:

Table 11: Email Server settings

Options	Description
Host	The hostname for the SMTP server that will be used to send the email.
Port	The TCP/IP port to be used to communicate with the SMTP server. The default is the standard SMTP port 25.

Options	Description
Username	For email servers that require a username and password to send email, the username for authentication.
Password	For email servers that require a username and password to send email, the password for authentication.
Require TSL	Enable Transport Layer Security (TLS) when sending the email through the designated server.

Email Settings

Table 12: Email settings

Option	Description
Sender email	The email address from which the email will be identified as being sent from. This email address should be one that is valid as a sender address for the SMTP server that you specify.
Recipients	A list of the recipients of each alert message. To specify more than one recipien, separate each address by a space, comma, or semicolon.
Test Mail	Click **Test Mail** to send a test email to confirm the settings and configuration of the email server and recipients.

Available Alerts

You can enable individual alert messages that can be sent by using the series of checkboxes. The supported alerts are:

Table 13: Available alerts

Alert	Description
Node was auto-failovered	The sending node has been auto-failovered.
Maximum number of auto-failovered nodes was reached	The auto-failover system stops auto-failover when the maximum number of spare nodes available has been reached.
Node wasn't auto-failovered as other nodes are down at the same time	Auto-failover does not take place if there are no spare nodes within the current cluster.
Node wasn't auto-failovered as the cluster was too small (less than 3 nodes)	You cannot support auto-failover with less than 3 nodes.
Node's IP address has changed unexpectedly	The IP address of the node has changed, which may indicate a network interface, operating system, or other network or system failure.
Disk space used for persistent storage has reach at least 90% of capacity	The disk device configured for storage of persistent data is nearing full capacity.
Metadata overhead is more than 50%	The amount of data required to store the metadata information for your dataset is now greater than 50% of the available RAM.
Bucket memory on a node is entirely used for metadata	All the available RAM on a node is being used to store the metadata for the objects stored. This means that there is no memory available for caching values,. With no memory left for storing metadata, further requests to store data will also fail.

Alert	Description
Writing data to disk for a specific bucket has failed	The disk or device used for persisting data has failed to store persistent data for a bucket.

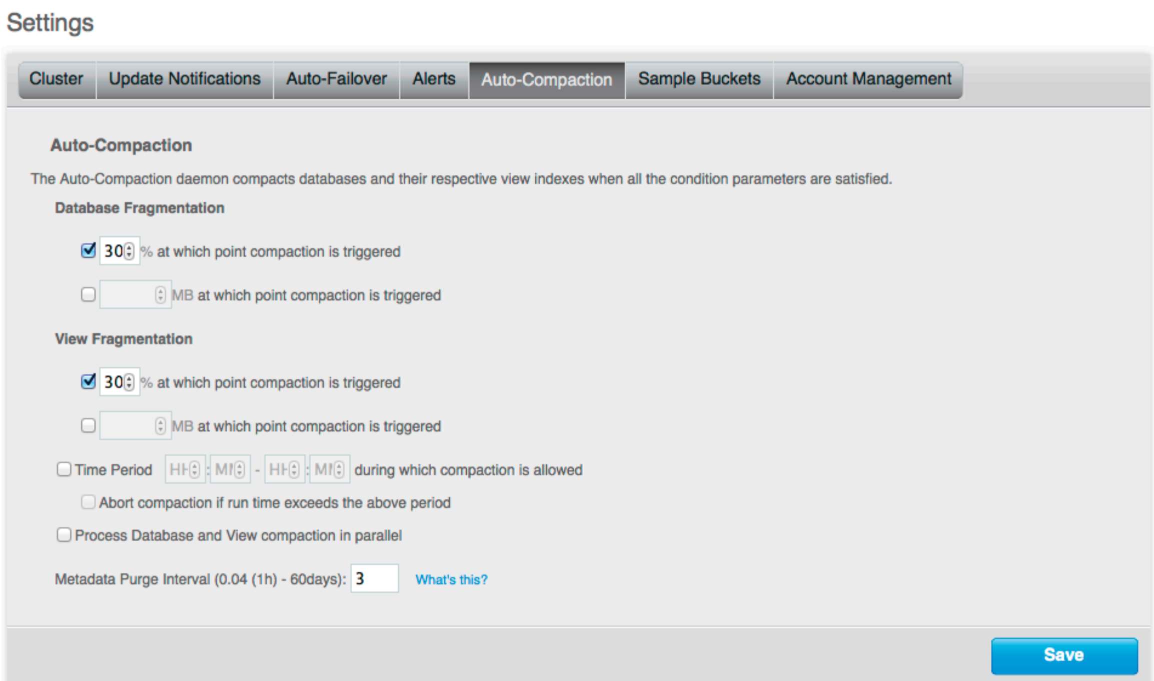
Related Links

Settings on page 294

The **Settings** section provides configuration and information for the cluster, update notifications, auto failover, alerts, auto compaction, sample buckets, and account management.

Auto-Compaction tab

The **Auto-Compaction** tab configures the default auto-compaction settings for all the databases. These can be overridden using per-bucket settings available when creating or editing data buckets. You can provide a purge interval to remove the key and metadata for items that have been deleted or are expired. This is known as 'tombstone purging'.



The Auto-Compaction tab sets the following default parameters:

Table 14: Auto-compaction parameters

Parameter	Description
Database Fragmentation	If checked, you must specify either the percentage of fragmentation at which database compaction will be triggered, or the database size at which compaction will be triggered. You can also configure both trigger parameters.
View Fragmentation	If checked, you must specify either the percentage of fragmentation at which database compaction will be triggered, or the view size at which compaction will be triggered. You can also configure both trigger parameters.
Time Period	If checked, you must specify the start hour and minute, and end hour and minute of the time period when compaction is allowed to occur.
Abort if run time exceeds the above period	If checked, if database compaction is running when the configured time period ends, the compaction process will be terminated.

Parameter	Description
Process Database and View compaction in parallel	If enabled, database and view compaction will be executed simultaneously, implying a heavier processing and disk I/O load during the compaction process.
Metadata Purge Interval	Defaults to three days. Tombstones are records of expired or deleted items and they include the key and metadata. Tombstones are used in Couchbase Server to provide eventual consistency of data between clusters. The auto-compaction process waits this number of days before it permanently deletes tombstones for expired or deleted items. If you set this value too low, you may see more inconsistent results in views queries such as deleted items in a result set. You may also see inconsistent items in clusters with XDCR set up between the clusters. If you set this value too high, it will delay the server from reclaiming disk space.

Related Links

Settings on page 294
The **Settings** section provides configuration and information for the cluster, update notifications, auto failover, alerts, auto compaction, sample buckets, and account management.

Related topics

Tombstone purging on page 104
Tombstones are records of expired or deleted items that include item keys and metadata.

Compaction on page 138
Database and view compaction helps to reclaim disk space and reduce fragmentation.

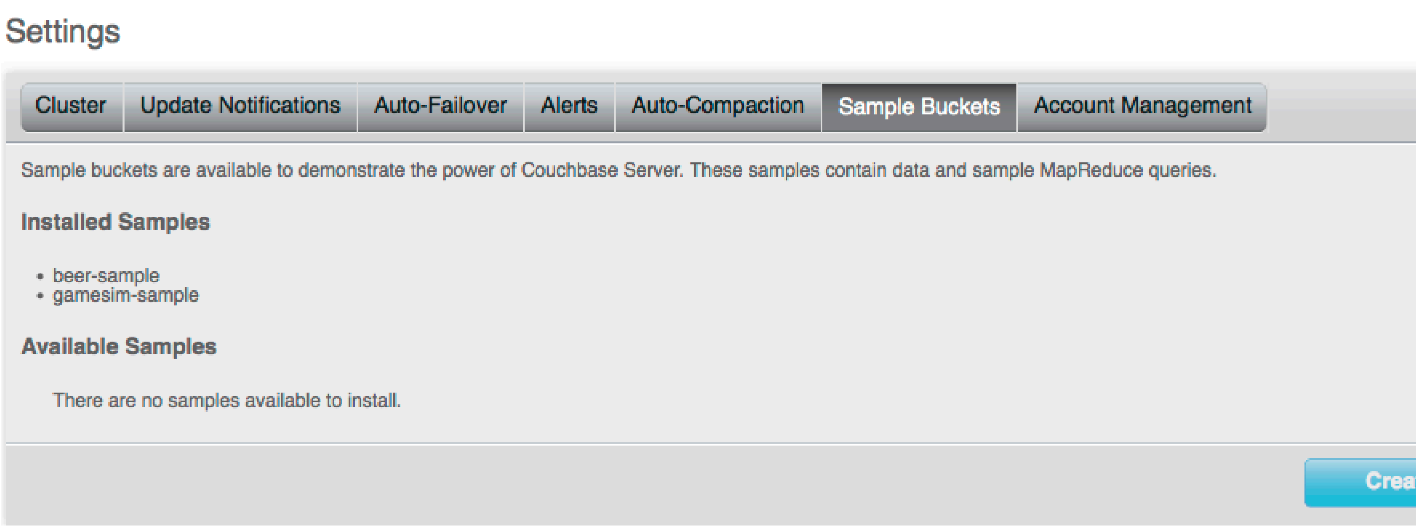
Buckets REST API on page 456
The bucket management and configuration REST API endpoints provide a fine level of control over the individual buckets in the cluster, their configuration, and specific operations.

Compaction REST API on page 494

Sample Buckets tab

The **Sample Buckets** tab enables you to install the sample bucket data if the data has not already been loaded in the system. If the sample bucket data was not loaded during setup, select the sample buckets that you want to load using the checkboxes, and click **Create**.

If the sample bucket data has already been loaded, it is listed under the Installed Samples section of the page.

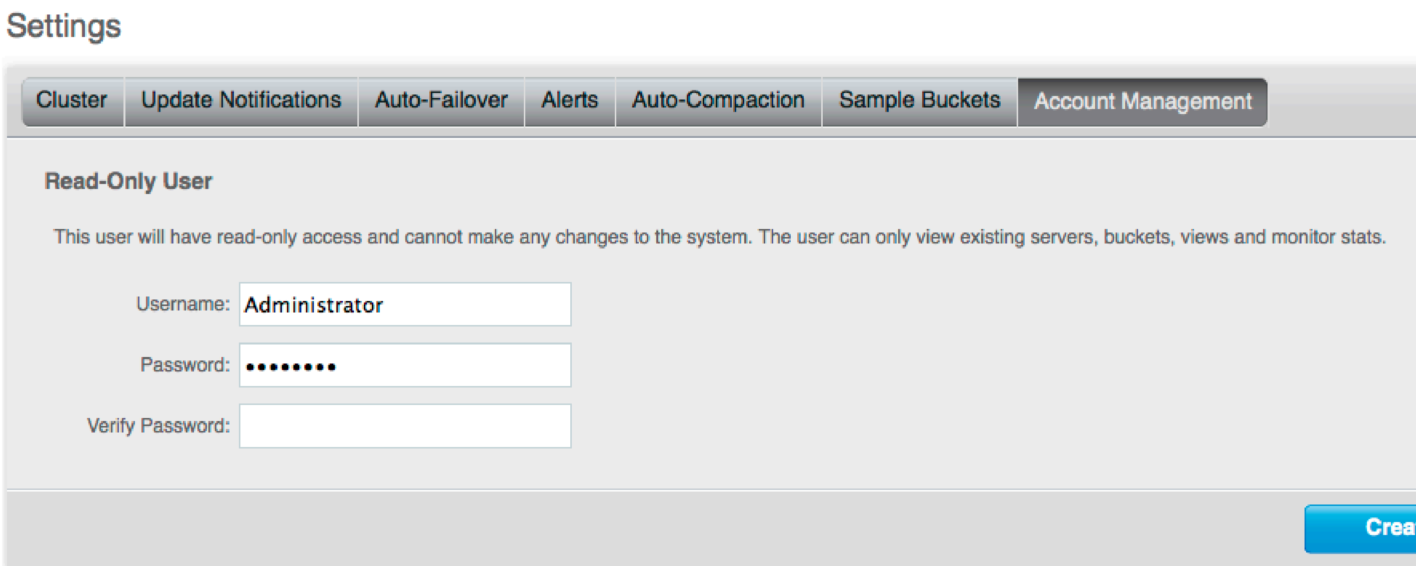


Related Links

Settings on page 294
The **Settings** section provides configuration and information for the cluster, update notifications, auto failover, alerts, auto compaction, sample buckets, and account management.

Account Management tab

Account management settings allows you to set up and modify the read-only user's user name and password. This user has read-only access and cannot make any changes to the system. The user can only view existing servers, buckets, views and monitor stats.



Related Links

Settings on page 294
The **Settings** section provides configuration and information for the cluster, update notifications, auto failover, alerts, auto compaction, sample buckets, and account management.

Creating a read-only user on page 302
One non-administrative user can be created with read-only access for the Web Console and REST API.

Creating a read-only user

One non-administrative user can be created with read-only access for the Web Console and REST API.

A read-only user cannot create buckets, edit buckets, add nodes to clusters, change XDCR settings, create views or see any stored data. Any REST API calls which require an administrator fail and return an error for this user.

In the Couchbase web console, a read-only user can view:

- Cluster Overview.
- Design documents and view definitions but cannot query views.
- Bucket summaries including Cache Size and Storage Size, but cannot view documents.
- List of XDCR replications and remote clusters.
- Logged events under the Log tab, but the user cannot Generate Diagnostic Report.
- Settings for a cluster.

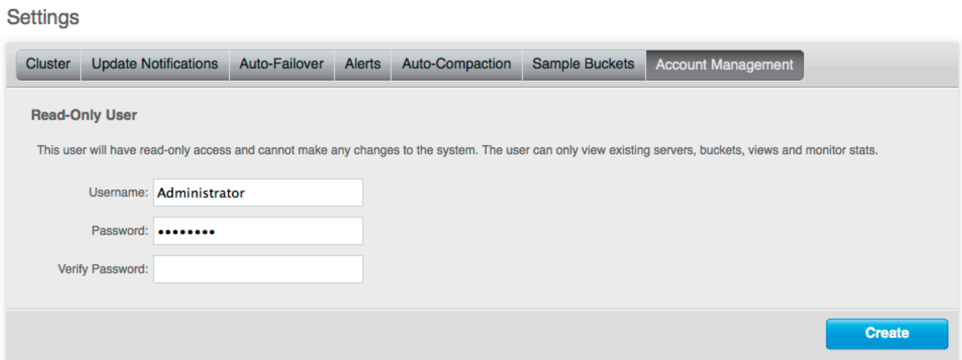
Note:

If a read-only user performs a REST POST or DELETE request that changes cluster, bucket, XDCR, or node settings, the server sends an HTTP 401 error:

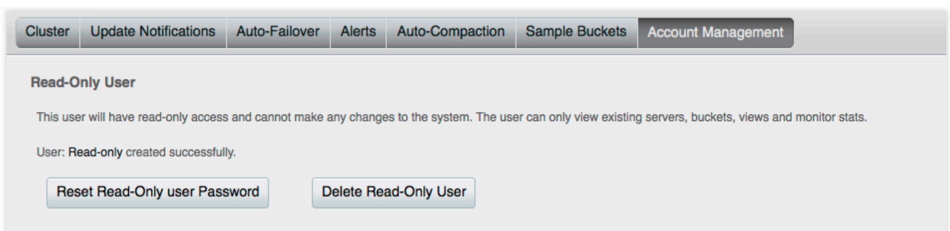
```
HTTP/1.1 401 Unauthorized WWW-Authenticate: Basic realm="Couchbase Server
 Admin / REST"
....
```

Tip: The read-only user cannot set up a Couchbase SDK to connect to the server. All SDKs require that a client connect with bucket-level credentials.

1. In the Couchbase Web Console, click Settings.
   A panel appears with several different sub-tabs.



2. Click Account Management. A panel appears where you can add a read-only user.
3. Enter a Username, Password and verify the password.
4. Click Create.
   The panel refreshes and has options for resetting the read-only user password or deleting the user.



Related Links

Account Management tab on page 302

Related topics

Users REST API on page 499

Couchbase, All Client Libraries

FAQs

Topic paragraph

What clients do I use with Couchbase?

Couchbase Server is compatible with existing memcached clients. If a memcached client already exists, just point it at couchbase. Regular testing is done with spymemcached (Java client), libmemcached, and fauna (Ruby client).

What is a vBucket?

A vBucket is conceptually a computed subset of all possible mapping keys. vBuckets are mapped to servers statically and have a consistent key through vBucket computations. The number of vBuckets in a cluster remains constant regardless of server topology which means that a key always maps to the same vBucket given the same hash.

What is a TAP stream?

A TAP stream is a when a client requests a stream of item updates from the server. That is, as other clients are requesting item mutations (for example, SET's and DELETE's), a TAP stream client can "wire-tap" the server to receive a stream of item change notifications. When a TAP stream client starts its connection, it may also optionally request a stream of all items stored in the server, even if no other clients are making any item changes. On the TAP stream connection setup options, a TAP stream client may request to receive just current items stored in the server (all items until "now"), or all item changes from now onward into in the future, or both.

Which ports does Couchbase Server need?

- 4369 - Erlang port mapper (epmd)
- 8091 - GUI and REST interface
- 8092 - Couchbase API port
- 11209, 11210, 11211, 11214, 11215 - Bucket and proxy ports
- 18091, 18092 - Internal REST and CAPI HTTPS for SSL
- 21100 to 21199 - Inclusive for dynamic cluster communication

What hardware and platforms does Couchbase Server support?

- Red Hat and CentOS versions 5, update 2 and higher
- Ubuntu 9
- Windows Server 2008
- Mac OS X (Community support only)
- 32-bit and 64-bit versions

How can I get Couchbase on a different OS?

The Couchbase source code is quite portable and is known to have been built on several other UNIX and Linux based OSs.

Can I query Couchbase by something other than the key name?

Not directly. It's possible to build these kinds of solutions atop TAP. For instance, it is possible to stream out the data, process it with Cascading, then create indexes in Elasticsearch.

What is the maximum item size in Couchbase>	The default item size for Couchbase buckets is 20 MBytes. The default item size for memcached buckets is 1 MByte.
How do I the change password?	With the command-line tool (CLI), use `couchbase-cli cluster-init`: `couchbase-cli cluster-init -c cluster_IP:8091 -u current_username-p current password --cluster-init-username=new_username --cluster-init-password=new_password`
How do I change the per-node RAM quota?	With the command-line tool (CLI), use `couchbase-cli`: `couchbase-cli cluster-init -c cluster_IP:8091 -u username-p password --cluster-init-ramsize=RAM_in_M`
How do I change the disk path?	With the command-line tool (CLI), use `node-init`: `couchbase-cli node-init -c cluster_IP:8091 -u username-p password--node-init-data-path=/tmp`
Why are some clients getting different results than others for the same requests?	This should never happen in a correctly configured Ccouchbase cluster, since Couchbase ensures a consistent view of all data in a cluster. However, if some clients can't reach all the nodes in a cluster (due to firewall or routing rules, for example), it is possible for the same key to end up on more than one cluster node, resulting in inconsistent duplication. Always ensure that all cluster nodes are reachable from every smart client or client-side moxi host.

Related Links

Couchbase Administration on page 89
Client libraries
memcached and vBuckets (by Dustin Sallings)
TAP interface (by Trond Norbye)
Couchbase consolidated downloads
Cascading

Sample buckets

Couchbase provides sample buckets to familiarize yourself with the Couchbase Server.

Related Links

Couchbase Administration on page 89
Beer sample bucket on page 306
Game Simulation sample bucket on page 309

Beer sample bucket

The beer sample data demonstrates a combination of the document structure used to describe different items, including references between objects, and also includes a number of sample views that show the view structure and layout.

The primary document type is the 'beer' document:

```
{
   "name": "Piranha Pale Ale",
   "abv": 5.7,
   "ibu": 0,
   "srm": 0,
   "upc": 0,
   "type": "beer",
   "brewery_id": "110f04166d",
   "updated": "2010-07-22 20:00:20",
   "description": "",
   "style": "American-Style Pale Ale",
   "category": "North American Ale"
}
```

Beer documents contain core information about different beers, including the name, alcohol by volume (`abv`) and categorization data.

Individual beer documents are related to brewery documents using the `brewery_id` field, which holds the information about a specific brewery for the beer:

```
{
   "name": "Commonwealth Brewing #1",
   "city": "Boston",
   "state": "Massachusetts",
   "code": "",
   "country": "United States",
   "phone": "",
   "website": "",
   "type": "brewery",
   "updated": "2010-07-22 20:00:20",
   "description": "",
   "address": [
   ],
   "geo": {
       "accuracy": "APPROXIMATE",
       "lat": 42.3584,
       "lng": -71.0598
   }
}
```

The brewery record includes basic contact and address information for the brewery, and contains a spatial record consisting of the latitude and longitude of the brewery location.

To demonstrate the view functionality in Couchbase Server, three views are defined.

brewery_beers view

The `brewery_beers` view outputs a composite list of breweries and beers they brew by using the view output format to create a ‘fake’ join. This outputs the brewery ID for brewery document types, and the brewery ID and beer ID for beer document types:

```
function(doc, meta) {
  switch(doc.type) {
  case "brewery":
    emit([meta.id]);
    break;
  case "beer":
    if (doc.brewery_id) {
      emit([doc.brewery_id, meta.id]);
    }
    break;
  }
}
```

The raw JSON output from the view:


```
{
   "total_rows" : 7315,
   "rows" : [
      {
         "value" : null,
         "id" : "110f0013c9",
         "key" : [
            "110f0013c9"
         ]
      },
      {
         "value" : null,
         "id" : "110fdd305e",
         "key" : [
            "110f0013c9",
            "110fdd305e"
         ]
      },
      {
         "value" : null,
         "id" : "110fdd3d0b",
         "key" : [
            "110f0013c9",
            "110fdd3d0b"
         ]
      },
…
      {
         "value" : null,
         "id" : "110fdd56ff",
         "key" : [
            "110f0013c9",
            "110fdd56ff"
         ]
      },
      {
         "value" : null,
         "id" : "110fe0aaa7",
         "key" : [
            "110f0013c9",
            "110fe0aaa7"
         ]
      },
      {
```

```
         "value" : null,
         "id" : "110f001bbe",
         "key" : [
            "110f001bbe"
         ]
      }
   ]
}
```


The output could be combined with the corresponding brewery and beer data to provide a list of the beers at each brewery.

by_location view

Outputs the brewery location, accounting for missing fields in the source data. The output creates information either by country, by country and state, or by country, state and city.

```
function (doc, meta) {
  if (doc.country, doc.state, doc.city) {
    emit([doc.country, doc.state, doc.city], 1);
  } else if (doc.country, doc.state) {
    emit([doc.country, doc.state], 1);
  } else if (doc.country) {
    emit([doc.country], 1);
  }
}
```

The view also includes the built-in `_count` function for the reduce portion of the view. Without using the reduce, the information outputs the raw location information:


```
{
   "total_rows" : 1413,
   "rows" : [
      {
         "value" : 1,
         "id" : "110f0b267e",
         "key" : [
            "Argentina",
            "",
            "Mendoza"
         ]
      },
      {
         "value" : 1,
         "id" : "110f035200",
         "key" : [
            "Argentina",
            "Buenos Aires",
            "San Martin"
         ]
      },
…
      {
         "value" : 1,
         "id" : "110f2701b3",
         "key" : [
            "Australia",
            "New South Wales",
            "Sydney"
         ]
      },
      {
         "value" : 1,
         "id" : "110f21eea3",
```

```
         "key" : [
            "Australia",
            "NSW",
            "Picton"
         ]
      },
      {
         "value" : 1,
         "id" : "110f117f97",
         "key" : [
            "Australia",
            "Queensland",
            "Sanctuary Cove"
         ]
      }
   ]
}
```

With the `reduce()` enabled, grouping can be used to report the number of breweries by the country, state, or city. For example, using a grouping level of two, the information outputs the country and state counts:

```
{"rows":[
{"key":["Argentina",""],"value":1},
{"key":["Argentina","Buenos Aires"],"value":1},
{"key":["Aruba"],"value":1},
{"key":["Australia"],"value":1},
{"key":["Australia","New South Wales"],"value":4},
{"key":["Australia","NSW"],"value":1},
{"key":["Australia","Queensland"],"value":1},
{"key":["Australia","South Australia"],"value":2},
{"key":["Australia","Victoria"],"value":2},
{"key":["Australia","WA"],"value":1}
]
}
```

Related Links

Sample buckets on page 306

Related topics

View and query pattern samples on page 222

Game Simulation sample bucket

The Game Simulation sample bucket is designed to showcase a typical gaming application that combines records showing individual gamers, game objects and how this information can be merged together and then reported on using views.

For example, a typical game player record looks like the one below:

```
{
    "experience": 14248,
    "hitpoints": 23832,
    "jsonType": "player",
    "level": 141,
    "loggedIn": true,
    "name": "Aaron1",
    "uuid": "78edf902-7dd2-49a4-99b4-1c94ee286a33"
}
```

A game object, in this case an Axe, is shown below:

```
{
   "jsonType" : "item",
   "name" : "Axe_14e3ad7b-8469-444e-8057-ac5aefcdf89e",
   "ownerId" : "Benjamin2",
```

```
   "uuid" : "14e3ad7b-8469-444e-8057-ac5aefcdf89e"
}
```

In this example, you can see how the game object has been connected to an individual user through the `ownerId` field of the item JSON.

Monsters within the game are similarly defined through another JSON object type:

```
{
    "experienceWhenKilled": 91,
    "hitpoints": 3990,
    "itemProbability": 0.19239324085462631,
    "jsonType": "monster",
    "name": "Wild-man9",
    "uuid": "f72b98c2-e84b-4b17-9e2a-bcec52b0ce1c"
}
```

For each of the three records, the `jsonType` field is used to define the type of the object being stored.

leaderboard view

The `leaderboard` view is designed to generate a list of the players and their current score:

```
function (doc) {
  if (doc.jsonType == "player") {
  emit(doc.experience, null);
  }
}
```

The view looks for records with a `jsonType` of “player”, and then outputs the `experience` field of each player record. Because the output from views is naturally sorted by the key value, the output of the view will be a sorted list of the players by their score. For example:

```
{
   "total_rows" : 81,
   "rows" : [
      {
         "value" : null,
         "id" : "Bob0",
         "key" : 1
      },
      {
         "value" : null,
         "id" : "Dustin2",
         "key" : 1
      },
…
      {
         "value" : null,
         "id" : "Frank0",
         "key" : 26
      }
   ]
}
```

To get the top 10 highest scores (and ergo players), you can send a request that reverses the sort order (by using `descending=true`, for example:

```
http://127.0.0.1:8092/gamesim-sample/_design/dev_players/_view/leaderboard?
descending=true&connection_timeout=60000&limit=10&skip=0
```

Which generates the following:

```
{
   "total_rows" : 81,
   "rows" : [
      {
```

```
            "value" : null,
            "id" : "Tony0",
            "key" : 23308
         },
         {
            "value" : null,
            "id" : "Sharon0",
            "key" : 20241
         },
         {
            "value" : null,
            "id" : "Damien0",
            "key" : 20190
         },
…
         {
            "value" : null,
            "id" : "Srini0",
            "key" :9
         },
         {
            "value" : null,
            "id" : "Aliaksey1",
            "key" : 17263
         }
   ]
}
```


playerlist view

The `playerlist` view creates a list of all the players by using a map function that looks for “player” records.

```
function (doc, meta) {
  if (doc.jsonType == "player") {
    emit(meta.id, null);
  }
}
```

This outputs a list of players in the format:


```
{
   "total_rows" : 81,
   "rows" : [
      {
         "value" : null,
         "id" : "Aaron0",
         "key" : "Aaron0"
      },
      {
         "value" : null,
         "id" : "Aaron1",
         "key" : "Aaron1"
      },
      {
         "value" : null,
         "id" : "Aaron2",
         "key" : "Aaron2"
      },
      {
         "value" : null,
         "id" : "Aliaksey0",
         "key" : "Aliaksey0"
      },
      {
```

```
            "value" : null,
            "id" : "Aliaksey1",
            "key" : "Aliaksey1"
        }
    ]
}
```

Related Links

Sample buckets on page 306

Troubleshooting

Troubleshooting covers general tips, common errors, log information, and other issues.

When troubleshooting your Couchbase Server deployment there are a number of different approaches available to you.

Related Links

Couchbase Administration on page 89

Common errors on page 313
Common errors encountered include issues when starting Couchbase server for the first time.

General tips on page 314
General tips include various initial diagnostics activities.

Logs and logging on page 315
Couchbase Server creates a number of different log files depending on the component of the system that produce the error, and the level and severity of the problem being reported.

Reporting issues on page 317
A description on what to include when reporting an issue (JIRA).

Beam.smp on page 318
Beam.smp uses excessive memory on Linux

Blocked indexer on page 318
Indexer shows no progress for long periods of time.

Server issues on page 321
Basic action associated with critical and informational issues.

Incorrect or missing data (server issue) on page 321
Data missing in query response or it's wrong (potentially due to server issues)

Incorrect or missing data (user issue) on page 332
Data missing in query response or it's wrong (user issue)

Design document aliases on page 333

Expired documents issue on page 335
Expired documents have their associated key-value pairs returned in queries with `stale=false`.

Index filesystem structure on page 335
A description of the index filesystem struction.

Index results for a single node on page 336
A specific URI is used to get index results for a single node.

Debugging replica index on page 337
Description of how to test and verify that the replica index is working.

Debugging stale=false queries on page 338
Debugging stale=false queries for missing/unexpected data

Timeout errors on page 343
Timeout errors when querying a view with `stale=false.`

total_rows values are too high on page 345
There are cases where the `total_rows` value is higher than expected.

Wrong documents or rows issue on page 346
The wrong documents or rows are returned when querying with `include_docs=true`.

Common errors

Common errors encountered include issues when starting Couchbase server for the first time.

This page will attempt to describe and resolve some common errors that are encountered when using Couchbase. It will be a living document as new problems and/or resolutions are discovered.

Problems Starting Couchbase Server for the first time

If you are having problems starting Couchbase Server on Linux for the first time, there are two very common causes of this that are actually quite related. When the `/etc/init.d/couchbase-server` script runs, it tries to set the file descriptor limit and core file size limit:

```
> ulimit -n 10240 ulimit -c unlimited
```

Depending on the defaults of your system, this may or may not be allowed. If Couchbase Server is failing to start, you can look through the logs and pick out one or both of these messages:

```
ns_log: logging ns_port_server:0:Port server memcached on node
 'ns_1@127.0.0.1' exited with status 71. »
Restarting. Messages: failed to set rlimit for open files. »
Try running as root or requesting smaller maxconns value.
```

Alternatively, you may additionally see or optionally see:

```
ns_port_server:0:info:message - Port server memcached on node 'ns_1@127.0.0.1'
 exited with status 71. »
Restarting. Messages: failed to ensure corefile creation
```

The resolution to these is to edit the /etc/security/limits.conf file and add these entries:

```
couchbase hard nofile 10240
couchbase hard core unlimited
```

Related Links

Troubleshooting on page 313
Troubleshooting covers general tips, common errors, log information, and other issues.

General tips

General tips include various initial diagnostics activities.

The following are some general tips that may be useful before performing any more detailed investigations:

- Try pinging the node.
- Try connecting to the Couchbase Server Web Console on the node.
- Try to use telnet to connect to the various ports that Couchbase Server uses.
- Try reloading the web page.
- Check firewall settings (if any) on the node. Make sure there isn't a firewall between you and the node. On a Windows system, for example, the Windows firewall might be blocking the ports (Control Panel > Windows Firewall).
- Make sure that the documented ports are open between nodes and make sure the data operation ports are available to clients.
- Check your browser's security settings.
- Check any other security software installed on your system, such as antivirus programs.
- Generate a Diagnostic Report for use by Couchbase Technical Support to help determine what the problem is. There are two ways of collecting this information:
    - Click `Generate Diagnostic Report` on the Log page to obtain a snapshot of your system's configuration and log information for deeper analysis. You must send this file to Couchbase.

- Run the `cbcollect_info` on each node within your cluster. To run, you must specify the name of the file to be generated:

```
> cbcollect_info nodename.zip
```

This will create a Zip file with the specified name. You must run each command individually on each node within the cluster. You can then send each file to Couchbase for analysis.

Related Links

Troubleshooting on page 313
Troubleshooting covers general tips, common errors, log information, and other issues.

Related topics

cbcollect_info tool on page 376

Network ports on page 51
Couchbase Server specific network ports for communication between server components and for communicating with the clients that access the data stored in the Couchbase cluster.

Logs and logging

Couchbase Server creates a number of different log files depending on the component of the system that produce the error, and the level and severity of the problem being reported.

Platform	Location
Linux	`/opt/couchbase/var/lib/couchbase/logs`
Windows	`C:\Program Files\Couchbase\Server\var\lib\couchbase\logs` Assumes default installation location
Mac OS X	`/Users/couchbase/Library/Application Support/Couchbase/var/lib/couchbase/logs`

Individual log files are automatically numbered, with the number suffix incremented for each new log, with a maximum of 20 files per log. Individual log file sizes are limited to 10MB by default.

File	Log Contents
`couchdb`	Errors relating to the couchdb subsystem that supports views, indexes and related REST API issues
`debug`	Debug level error messages related to the core server management subsystem, excluding information included in the `couchdb`, `xdcr` and `stats` logs.
`info`	Information level error messages related to the core server management subsystem, excluding information included in the `couchdb`, `xdcr` and `stats` logs.
`error`	Error level messages for all subsystems excluding `xdcr`.
`xcdr_error`	XDCR error messages.
`xdcr`	XDCR information messages.
`mapreduce_errors`	JavaScript and other view-processing errors are reported in this file.
`views`	Errors relating to the integration between the view system and the core server subsystem.
`stats`	Contains periodic reports of the core statistics.

File	Log Contents
`memcached.log`	Contains information relating to the core memcache component, including vBucket and replica and rebalance data streams requests.

Each log file group will also include a `.idx` and `.siz` file which holds meta information about the log file group. These files are automatically updated by the logging system.

Changing log file location

The default file log location is /opt/couchbase/var/lib/couchbase/logs, however, if you want to change the default log location to a different directory, change the log file configuration option.

Note

To implement a log file location change (from the default), you must be log in as either root or sudo and the Couchbase service must be restarted.

To change the log file configuration:

1. Log in as root or sudo and navigate to the directory where you installed Couchbase. For example: `/opt/couchbase/etc/couchbase/static_config`
2. Edit the static_config file and change the `error_logger_mf_dir` variable to a different directory. For example: `{error_logger_mf_dir, "/home/user/cb/opt/couchbase/var/lib/couchbase/logs"}`
3. Restart the Couchbase service. After restarting the Couchbase service, all subsequent logs will be in the new directory.

Changing logging levels

The default logging level for all log files are set to debug except for couchdb, which is set to info. If you want to change the default logging level, modify the logging level configuration options.

The configuration change can be performed in one of the following ways:

- persistent
- dynamic (on the fly, without restarting).

Changing logging levels to be persistent

Logging levels can be changed so that the changes are persistent, that is, the changes continue to be implemented should a Couchbase Server reboot occur.

Note: To implement logging level changes, the Couchbase service must be restarted.

To change logging levels to be persistent:

1. Log in as root or sudo and navigate to the directory where you installed Couchbase. For example: `/opt/couchbase/etc/couchbase/static_config`
2. Edit the **static_config** file and change the desired log component. For example, parameters with the `loglevel_` prefix set the logging level.
3. Restart the Couchbase service.

After restarting the Couchbase service, logging levels for that component will be changed.

Changing logging levels dynamically

If logging levels are changed dynamically and if a Couchbase server reboot occurs, then the changed logging levels revert to the default.

To change logging levels dynamically, execute a curl POST command using the following syntax:

```
curl -X POST -u adminName:adminPassword
  HOST:PORT/diag/eval
  -d 'ale:set_loglevel(<log_component>,<logging_level>).'
```

Where:

Log_component	The default log level (except couchdb) is debug. For example, ns_server. The available loggers are ns_server, couchdb, user, Menelaus, ns_doctor, stats, rebalance, cluster, views, mapreduce_errors , xdcr and error_logger
Logging_level	The available log levels are debug, info, warning, and error.

```
curl -X POST -u Administrator:password
 http://127.0.0.1:8091/diag/eval
 -d 'ale:set_loglevel(ns_server,error).
```

Related Links

Troubleshooting on page 313
Troubleshooting covers general tips, common errors, log information, and other issues.

Reporting issues

A description on what to include when reporting an issue (JIRA).

When reporting issues to Couchbase, you should always add the following information to JIRA issues:

- Environment description (package installation? cluster_run? build number? OS)
- All the steps necessary to reproduce (if applicable)
- Show the full content of all the design documents
- Describe how your documents are structured (all same structure, different structures?)
- If you generated the data with any tool, mention its name and all the parameters given to it (full command line)
- Show what queries you were doing (include all query parameters, full URL), use curl with option -v and show the full output, example:

```
> curl -v 'http://localhost:9500/default/_design/test/_view/view1?
limit=10&stale=false'
* About to connect() to localhost port 9500 (#0)
*   Trying ::1... Connection refused
*   Trying 127.0.0.1... connected
* Connected to localhost (127.0.0.1) port 9500 (#0)
> GET /default/_design/test/_view/view1 HTTP/1.1
> User-Agent: curl/7.21.4 (universal-apple-darwin11.0) libcurl/7.21.4
 OpenSSL/0.9.8r zlib/1.2.5
> Host: localhost:9500
> Accept: */*
>
< HTTP/1.1 200 OK
< Transfer-Encoding: chunked
< Server: MochiWeb/1.0 (Any of you quaids got a smint?)
< Date: Tue, 21 Aug 2012 14:43:06 GMT
< Content-Type: text/plain;charset=utf-8
< Cache-Control: must-revalidate
<
{"total_rows":2,"rows":[
{"id":"doc1","key":"doc1","value":111111},
{"id":"doc2","key":"doc2","value":222222}
]
}
```

```
* Connection #0 to host localhost left intact
* Closing connection #0
```

- Repeat the query with different values for the stale parameter and show the output
- Attach logs from all nodes in the cluster
- Try all view related operations, including design document creation/update/deletion, from the command line. The goal here to isolate UI problems from the view engine.
- If you suspect the indexer is stuck, blocked, etc, please use curl against the `_active_tasks` API to confirm that, the goal again is to isolate UI issues from view-engine issues. Example:

```
> curl -s 'http://localhost:9500/_active_tasks' | json_xs
[
   {
      "indexer_type" : "main",
      "started_on" : 1345645088,
      "progress" : 43,
      "initial_build" : true,
      "updated_on" : 1345645157,
      "total_changes" : 250000,
      "design_documents" : [
         "_design/test"
      ],
      "pid" : "<0.5948.0>",
      "changes_done" : 109383,
      "signature" : "4995c136d926bdaf94fbe183dbf5d5aa",
      "type" : "indexer",
      "set" : "default"
   }
]
```

Note that the `started_on` and `update_on` fields are UNIX timestamps. There are tools (even online) and programming language APIs (Perl, Python, etc) to convert them into human readable form, including date and time. Note that the `_active_tasks` API contains information per node, so you'll have to query `_active_tasks` or every node in the cluster to verify if progress is stuck, etc.

Related Links

Troubleshooting on page 313
Troubleshooting covers general tips, common errors, log information, and other issues.

Related topics

couchbase.com/issues

Beam.smp

Beam.smp uses excessive memory on Linux

On Linux, if XDCR Max Replications per Bucket are set to a value in the higher limit (such as 128), then beam.sm uses excessive memory. Solution: Reset to 32 or lower.

Related Links

Troubleshooting on page 313
Troubleshooting covers general tips, common errors, log information, and other issues.

Blocked indexer

Indexer shows no progress for long periods of time.

Each design document maps to one indexer, so when the indexer runs it updates all views defined in the corresponding design document. Indexing takes resources (CPU, disk IO, memory), therefore Couchbase Server limits the maximum number of indexers that can run in parallel. There are 2 configuration parameters to specify the limit, one for regular (main/active) indexers and other for replica indexers (more on this in a later section). The default for the former is 4 and for the later is 2. They can be queried like this:

```
> curl -s 'http://Administrator:asdasd@localhost:8091/settings/
maxParallelIndexers'
{"globalValue":4,"nodes":{"n_0@192.168.1.80":4}}
```

`maxParallelIndexers` is for main indexes and `maxParallelReplicaIndexers` is for replica indexes. When there are more design documents (indexers) than maxParallelIndexers, some indexers are blocked until there's a free slot, and the rule is simple as first-come-first-served. These slots are controlled by 2 barriers processes, one for main indexes, and the other for replica indexes. Their current state can be seen from `_active_tasks` (per node), for example when there's no indexing happening:

```
> curl -s 'http://localhost:9500/_active_tasks' | json_xs
[
 {
     "waiting" : 0,
         "started_on" : 1345642656,
         "pid" : "<0.234.0>",
         "type" : "couch_main_index_barrier",
         "running" : 0,
         "limit" : 4,
         "updated_on" : 1345642656
         },
 {
     "waiting" : 0,
         "started_on" : 1345642656,
         "pid" : "<0.235.0>",
         "type" : "couch_replica_index_barrier",
         "running" : 0,
         "limit" : 2,
         "updated_on" : 1345642656
         }
 ]
```

The `waiting` fields tells us how many indexers are blocked, waiting for their turn to run. Queries with `stale=false` have to wait for the indexer to be started (if not already), unblocked and to finish, which can lead to a long time when there are many design documents in the system. Also take into account that the indexer for a particular design document might be running for one node but it might be blocked in another node - when it's blocked it's not necessarily blocked in all nodes of the cluster nor when it's running is necessarily running in all nodes of the cluster - you verify this by querying _active_tasks for each node (this API is not meant for direct user consumption, just for developers and debugging/troubleshooting).

Through `_active_tasks` (remember, it's per node, so check it for every node in the cluster), you can see which indexers are running and which are blocked. Here follows an example where we have 5 design documents (indexers) and `>maxParallelIndexers` is 4:

```
> curl -s 'http://localhost:9500/_active_tasks' | json_xs
[
   {
      "waiting" : 1,
      "started_on" : 1345644651,
      "pid" :  "<0.234.0>",
      "type" :  "couch_main_index_barrier",
      "running" : 4,
      "limit" : 4,
      "updated_on" : 1345644923
   },
   {
      "waiting" : 0,
      "started_on" : 1345644651,
      "pid" :  "<0.235.0>",
      "type" :  "couch_replica_index_barrier",
      "running" : 0,
      "limit" : 2,
      "updated_on" : 1345644651
   },
```

```
{
   "indexer_type" : "main",
   "started_on" : 1345644923,
   "updated_on" : 1345644923,
   "design_documents" : [
      "_design/test"
   ],
   "pid" :  "<0.4706.0>",
   "signature" : "4995c136d926bdaf94fbe183dbf5d5aa",
   "type" :  "blocked_indexer",
   "set" :  "default"
},
{
   "indexer_type" : "main",
   "started_on" : 1345644923,
   "progress" : 0,
   "initial_build" : true,
   "updated_on" : 1345644923,
   "total_changes" : 250000,
   "design_documents" : [
      "_design/test4"
   ],
   "pid" :  "<0.4715.0>",
   "changes_done" : 0,
   "signature" : "15e1f576bc85e3e321e28dc883c90077",
   "type" :  "indexer",
   "set" :  "default"
},
{
   "indexer_type" : "main",
   "started_on" : 1345644923,
   "progress" : 0,
   "initial_build" : true,
   "updated_on" : 1345644923,
   "total_changes" : 250000,
   "design_documents" : [
      "_design/test3"
   ],
   "pid" :  "<0.4719.0>",
   "changes_done" : 0,
   "signature" : "018b83ca22e53e14d723ea858ba97168",
   "type" :  "indexer",
   "set" :  "default"
},
{
   "indexer_type" : "main",
   "started_on" : 1345644923,
   "progress" : 0,
   "initial_build" : true,
   "updated_on" : 1345644923,
   "total_changes" : 250000,
   "design_documents" : [
      "_design/test2"
   ],
   "pid" :  "<0.4722.0>",
   "changes_done" : 0,
   "signature" : "440b0b3ded9d68abb559d58b9fda3e0a",
   "type" :  "indexer",
   "set" : "default"
},
{
   "indexer_type" : "main",
   "started_on" : 1345644923,
   "progress" : 0,
```

```
      "initial_build" : true,
      "updated_on" : 1345644923,
      "total_changes" : 250000,
      "design_documents" : [
         "_design/test7"
      ],
      "pid" :  "<0.4725.0>",
      "changes_done" : 0,
      "signature" : "fd2bdf6191e61af6e801e3137e2f1102",
      "type" :  "indexer",
      "set" :  "default"
   }
]
```

The indexer for design document _design/test is represented by a task with a `type` field of `blocked_indexer`, while other indexers have a task with type `indexer`, meaning they're running. The task with type `couch_main_index_barrier` confirms this by telling us there are currently 4 indexers running and 1 waiting for its turn. When an indexer is allowed to execute, its active task with type `blocked_indexer` is replaced by a new one with type `indexer`.

Related Links

Troubleshooting on page 313
Troubleshooting covers general tips, common errors, log information, and other issues.

Server issues

Basic action associated with critical and informational issues.

The following table outlines some specific areas to check when experiencing different problems:

Severity	Issue	Suggested Action(s)
Critical	Couchbase Server does not start up.	Check that the service is running.
		Check error logs.
		Try restarting the service.
Critical	A server is not responding.	Check that the service is running.
		Check error logs.
		Try restarting the service.
Critical	A server is down.	Try restarting the server.
		Use the command-line interface to check connectivity.
Informational	Bucket authentication failure.	Check the properties of the bucket that you are attempting to connect to.

The primary source for run-time logging information is the Couchbase Server Web Console. Run-time logs are automatically set up and started during the installation process. However, the Couchbase Server gives you access to lower-level logging details if needed for diagnostic and troubleshooting purposes. Log files are stored in a binary format in the logs directory under the Couchbase installation directory. You must use `browse_logs` to extract the log contents from the binary format to a text file.

Related Links

Troubleshooting on page 313
Troubleshooting covers general tips, common errors, log information, and other issues.

Incorrect or missing data (server issue)

Data missing in query response or it's wrong (potentially due to server issues)

Sometimes, especially between releases for development builds, it's possible results are missing due to issues in some component of Couchbase Server. This section describes how to do some debugging to identify which components, or at least to identify which components are not at fault.

Before proceeding, it needs to be mentioned that each vbucket is physically represented by a CouchDB database (generated by couchstore component) which corresponds to exactly 1 file in the filesystem, example from a development environment using 16 vbuckets only (for example simplicity), 4 nodes and without replicas enabled:

```
> tree ns_server/couch/0/
ns_server/couch/0/
 ???
_replicator.couch.1
 ???
_users.couch.1
 ??? default
     ??? 0.couch.1
     ??? 1.couch.1
     ??? 2.couch.1
     ??? 3.couch.1
     ??? master.couch.1
     ??? stats.json

 1 directory, 8 files

> tree ns_server/couch/1/
ns_server/couch/1/
 ???
_replicator.couch.1
 ???
_users.couch.1
 ??? default
     ??? 4.couch.1
     ??? 5.couch.1
     ??? 6.couch.1
     ??? 7.couch.1
     ??? master.couch.1
     ??? stats.json
     ??? stats.json.old

 1 directory, 9 files

> tree ns_server/couch/2/
ns_server/couch/2/
 ???
_replicator.couch.1
 ???
_users.couch.1
 ??? default
     ??? 10.couch.1
     ??? 11.couch.1
     ??? 8.couch.1
     ??? 9.couch.1
     ??? master.couch.1
     ??? stats.json
     ??? stats.json.old

 1 directory, 9 files

> tree ns_server/couch/3/
ns_server/couch/3/
 ???
_replicator.couch.1
 ???
_users.couch.1
```

```
 ??? default
     ??? 12.couch.1
     ??? 13.couch.1
     ??? 14.couch.1
     ??? 15.couch.1
     ??? master.couch.1
     ??? stats.json
     ??? stats.json.old

 1 directory, 9 files
```

For this particular example, because there are `no replicas enabled` (ran `./cluster_connect -n 4 -r 0`), each node only has database files for the vbuckets it's responsible for (active vbuckets). The numeric suffix in each database filename, starts at 1 when the database file is created and it gets incremented, by 1, every time the vbucket is compacted. If replication is enabled, for example you ran `./cluster_connect -n 4 -r 1`, then each node will have vbucket database files for the vbuckets it's responsible for (active vbuckets) and for some replica vbuckets, example:

```
> tree ns_server/couch/0/

ns_server/couch/0/
 ???
_replicator.couch.1
 ???
_users.couch.1
 ??? default
     ??? 0.couch.1
     ??? 1.couch.1
     ??? 12.couch.1
     ??? 2.couch.1
     ??? 3.couch.1
     ??? 4.couch.1
     ??? 5.couch.1
     ??? 8.couch.1
     ??? master.couch.1
     ??? stats.json

 1 directory, 12 files

> tree ns_server/couch/1/

ns_server/couch/1/
 ???
_replicator.couch.1
 ???
_users.couch.1
 ??? default
     ??? 0.couch.1
     ??? 1.couch.1
     ??? 13.couch.1
     ??? 4.couch.1
     ??? 5.couch.1
     ??? 6.couch.1
     ??? 7.couch.1
     ??? 9.couch.1
     ??? master.couch.1
     ??? stats.json

 1 directory, 12 files

> tree ns_server/couch/2/

ns_server/couch/2/
 ???
```

```
_replicator.couch.1
 ???
_users.couch.1
 ??? default
     ??? 10.couch.1
     ??? 11.couch.1
     ??? 14.couch.1
     ??? 15.couch.1
     ??? 2.couch.1
     ??? 6.couch.1
     ??? 8.couch.1
     ??? 9.couch.1
     ??? master.couch.1
     ??? stats.json

 1 directory, 12 files

> tree ns_server/couch/3/
ns_server/couch/3/
 ???
_replicator.couch.1
 ???
_users.couch.1
 ??? default
     ??? 10.couch.1
     ??? 11.couch.1
     ??? 12.couch.1
     ??? 13.couch.1
     ??? 14.couch.1
     ??? 15.couch.1
     ??? 3.couch.1
     ??? 7.couch.1
     ??? master.couch.1
     ??? stats.json

 1 directory, 12 files
```

You can figure out which vbucket are active in each node, by querying the following URL:

```
> curl -s http://localhost:8091/pools/default/buckets |
  json_xs
 [
    {
       "quota" :
{
          "rawRAM" : 268435456,
          "ram"
: 1073741824
       },
       "localRandomKeyUri" : "/pools/default/buckets/default/localRandomKey",
       "bucketCapabilitiesVer" : "",
       "authType"
: "sasl",
       "uuid" :
  "89dd5c64504f4a9414a2d3bcf9630d15",
       "replicaNumber" : 1,
       "vBucketServerMap" : {
          "vBucketMap" : [
             [
                0,
                1
             ],
             [
                0,
```

```
            1
         ],
         [
            0,
            2
         ],
         [
            0,
            3
         ],
         [
            1,
            0
         ],
         [
            1,
            0
         ],
         [
            1,
            2
         ],
         [
            1,
            3
         ],
         [
            2,
            0
         ],
         [
            2,
            1
         ],
         [
            2,
            3
         ],
         [
            2,
            3
         ],
         [
            3,
            0
         ],
         [
            3,
            1
         ],
         [
            3,
            2
         ],
         [
            3,
            2
         ]
      ],
      "numReplicas" : 1,
      "hashAlgorithm" : "CRC",
      "serverList" : [
         "192.168.1.81:12000",
         "192.168.1.82:12002",
```

```
              "192.168.1.83:12004",
              "192.168.1.84:12006"
           ]
       },

(....)
 ]
```

The field to look at is named `vBucketServerMap`, and it contains two important sub-fields, named `vBucketMap` and `serverList`, which we use to find out which nodes are responsible for which vbuckets (active vbuckets).

Looking at these 2 fields, we can do the following active and replica vbucket to node mapping:

- vbuckets 0, 1, 2 and 3 are active at node 192.168.1.81:12000, and vbuckets 4, 5, 8 and 12 are replicas at that same node
- vbuckets 4, 5, 6 and 7 are active at node 192.168.1.82:12002, and vbuckets 0, 1, 9 and 13 are replicas at that same node
- vbuckets 8, 9, 10 and 11 are active at node 192.168.1.83:12004, and vbuckets 2, 6, 14 and 15 are replicas at that same node
- vbuckets 12, 13, 14 and 15 are active at node 192.168.1.84:12006, and vbucket 3, 7, 11 and 10

the value of `vBucketMap` is an array of arrays of 2 elements. Each sub-array corresponds to a vbucket, so the first one is related to vbucket 0, second one to vbucket 1, etc, and the last one to vbucket 15. Each sub-array element is an index (starting at 0) into the `serverList` array. First element of each sub-array tells us which node (server) has the corresponding vbucket marked as active, while the second element tells us which server has this vbucket marked as replica.

If the replication factor is greater than 1 (N > 1), then each sub-array will have N + 1 elements, where first one is always index of server/node that has that vbucket active and the remaining elements are the indexes of the servers having the first, second, third, etc replicas of that vbucket.

After knowing which vbuckets are active in each node, we can use some tools such as `couch_dbinfo` and `couch_dbdump` to analyze active vbucket database files. Before looking at those tools, lets first know what database sequence numbers are.

When a CouchDB database (remember, each corresponds to a vbucket) is created, its update_seq (update sequence number) is 0. When a document is created, updated or deleted, its current sequence number is incremented by 1. So all the following sequence of actions result in the final sequence number of 5:

1. Create document doc1, create document doc2, create document doc3, create document doc4, create document doc5
2. Create document doc1, update document doc1, update document doc1, update document doc1, delete document doc1
3. Create document doc1, delete document doc1, create document doc2, update document doc2, update document doc2
4. Create document doc1, create document doc2, create document doc3, create document doc4, update document doc2
5. etc…

You can see the current update_seq of a vbucket database file, amongst other information, with the `couch_dbinfo` command line tool, example with vbucket 0, active in the first node:

```
> ./install/bin/couch_dbinfo ns_server/couch/0/default/0.couch.1
 DB Info
  (ns_server/couch/0/default/0.couch.1)
    file format version: 10
    update_seq: 31250
    doc count: 31250
    deleted doc count: 0
    data size: 3.76 MB
    B-tree size: 1.66 MB
    total disk size: 5.48 MB
```

After updating all the documents in that vbucket database, the update_seq doubled:

```
> ./install/bin/couch_dbinfo ns_server/couch/0/default/0.couch.1
DB Info
 (ns_server/couch/0/default/0.couch.1)
   file format version: 10
   update_seq:00
   doc count: 31250
   deleted doc count: 0
   data size: 3.76 MB
   B-tree size: 1.75 MB
   total disk size: 10.50 MB
```

An important detail, if not obvious, is that with each vbucket database sequence number one and only one document ID is associated to it. At any time, there's only one update sequence number associated with a document ID, and it's always the most recent. We can verify this with the `couch_dbdump` command line tool. Take the following example, where we only have 2 documents, document with ID doc1 and document with ID doc2:

```
> ./install/bin/couch_dbdump ns_server/couch/0/default/0.couch.1
Doc seq: 1
     id: doc1
     rev: 1
     content_meta: 0
     cas: 130763975746, expiry: 0, flags: 0
     data: {"value": 1}
Total docs: 1
```

On an empty vbucket 0 database, we created document with ID `doc1`, which has a JSON value of `{"value": 1}`. This document is now associated with update sequence number 1. Next we create another document, with ID *doc2* and JSON value `{"value": 2}`, and the output of `couch_dbdump` is:

```
> ./install/bin/couch_dbdump ns_server/couch/0/default/0.couch.1
Doc seq: 1
     id: doc1
     rev: 1
     content_meta: 0
     cas: 130763975746, expiry: 0, flags: 0
     data: {"value": 1}
Doc seq: 2
     id: doc2
     rev: 1
     content_meta: 0
     cas: 176314689876, expiry: 0, flags: 0
     data: {"value": 2}
Total docs: 2
```

Document `doc2` got associated to vbucket 0 database update sequence number 2. Next, we update document `doc1` with a new JSON value of `{"value": 1111}`, and `couch_dbdump` tells us:

```
> ./install/bin/couch_dbdump ns_server/couch/0/default/0.couch.1
Doc seq: 2
     id: doc2
     rev: 1
     content_meta: 0
     cas: 176314689876, expiry: 0, flags: 0
     data: {"value": 2}
Doc seq: 3
     id: doc1
     rev: 2
     content_meta: 0
     cas: 201537725466, expiry: 0, flags: 0
     data: {"value": 1111}

Total docs: 2
```

So, document `doc1` is now associated with update sequence number 3. Note that it's no longer associated with sequence number 1, because the update was the most recent operation against that document (remember, only 3 operations are possible: create, update or delete). The database no longer has a record for sequence number 1 as well. After this, we update document `doc2` with JSON value `{"value": 2222}`, and we get the following output from `couch_dbdump`:

```
> ./install/bin/couch_dbdump ns_server/couch/0/default/0.couch.1
Doc seq: 3
     id: doc1
     rev: 2
     content_meta: 0
     cas: 201537725466, expiry: 0, flags: 0
     data: {"value": 1111}
Doc seq: 4
     id: doc2
     rev: 2
     content_meta: 0
     cas: 213993873979, expiry: 0, flags:   0
     data: {"value": 2222}

Total docs: 2
```

Document `doc2` is now associated with sequence number 4, and sequence number 2 no longer has a record in the database file. Finally we deleted document `doc1`, and then we get:

```
> ./install/bin/couch_dbdump ns_server/couch/0/default/0.couch.1
Doc seq: 4
     id: doc2
     rev: 2
     content_meta: 0
     cas: 213993873979, expiry: 0, flags: 0
     data: {"value": 2222}
Doc seq: 5
     id: doc1
     rev: 3
     content_meta: 3
     cas: 201537725467, expiry: 0, flags: 0
     doc deleted
     could not read document body: document not found

Total docs: 2
```

Note that document deletes don't really delete documents from the database files, instead they flag the document has deleted and remove its JSON (or binary) value. Document `doc1` is now associated with sequence number 5 and the record for its previously associated sequence number 3, is removed from the vbucket 0 database file. This allows for example, indexes to know they have to delete all Key-Value pairs previously emitted by a map function for a document that was deleted - if there weren't any update sequence number associated with the delete operation, indexes would have no way to know if documents were deleted or not.

These details of sequence numbers and document operations are what allow indexes to be updated incrementally in Couchbase Server (and Apache CouchDB as well).

In Couchbase Server, indexes store in their header (state) the last update_seq seen for each vbucket database. Put it simply, whenever an index build/update finishes, it stores in its header the last update_seq processed for each vbucket database. Vbucket databases have states too in indexes, and these states do not necessarily match the vbucket states in the server. For the goals of this wiki page, it only matters to mention that view requests with `stale=false` will be blocked only if the currently stored update_seq of any active vbucket in the index header is smaller than the current update_seq of the corresponding vbucket database - if this is true for at least one active vbucket, an index update is scheduled immediately (if not already running) and when it finishes it will unblock the request. Requests with `stale=false` will not be blocked if the update_seq of vbuckets in the index with other states (passive, cleanup, replica) are smaller than the current update_seq of the corresponding vbucket databases - the reason for this is that queries only see rows produced for documents that live in the active vbuckets.

We can see that states of vbuckets in the index, and the update_seqs in the index, by querying the following URL (example for 16 vbuckets only, for the sake of simplicity):

```
> curl -s 'http://localhost:9500/_set_view/default/_design/dev_test2/_info' |
 json_xs
{
   "unindexable_partitions" : {},
   "passive_partitions" : [],
   "compact_running" : false,
   "cleanup_partitions" : [],
   "replica_group_info" : {
      "unindexable_partitions" : {},
      "passive_partitions" : [
         4,
         5,
         8,
         12
      ],
      "compact_running" : false,
      "cleanup_partitions" : [],
      "active_partitions" : [],
      "pending_transition" : null,
      "db_set_message_queue_len" : 0,
      "out_of_sync_db_set_partitions" : false,
      "expected_partition_seqs" : {
         "8" :00,
         "4" :00,
         "12" :00,
         "5" :00
      },
      "updater_running" : false,
      "partition_seqs" : {
         "8" :00,
         "4" :00,
         "12" :00,
         "5" :00
      },
      "stats" : {
         "update_history" : [
            {
               "deleted_ids" : 0,
               "inserted_kvs" : 38382,
               "inserted_ids" : 12794,
               "deleted_kvs" : 38382,
               "cleanup_kv_count" : 0,
               "blocked_time" : 1.5e-05,
               "indexing_time" : 3.861918
            }
         ],
         "updater_cleanups" : 0,
         "compaction_history" : [
            {
               "cleanup_kv_count" : 0,
               "duration" : 1.955801
            },
            {
               "cleanup_kv_count" : 0,
               "duration" : 2.443478
            },
            {
               "cleanup_kv_count" : 0,
               "duration" : 4.956397
            },
            {
```

```
                "cleanup_kv_count" : 0,
                "duration" : 9.522231
             }
          ],
          "full_updates" : 1,
          "waiting_clients" : 0,
          "compactions" : 4,
          "cleanups" : 0,
          "partial_updates" : 0,
          "stopped_updates" : 0,
          "cleanup_history" : [],
          "cleanup_interruptions" : 0
       },
       "initial_build" : false,
       "update_seqs" : {
          "8" :00,
          "4" :00,
          "12" :00,
          "5" :00
       },
       "partition_seqs_up_to_date" : true,
       "updater_state" : "not_running",
       "data_size" : 5740951,
       "cleanup_running" : false,
       "signature" : "440b0b3ded9d68abb559d58b9fda3e0a",
       "max_number_partitions" : 16,
       "disk_size" : 5742779
    },
    "active_partitions" : [
       0,
       1,
       2,
       3
    ],
    "pending_transition" : null,
    "db_set_message_queue_len" : 0,
    "out_of_sync_db_set_partitions" : false,
    "replicas_on_transfer" : [],
    "expected_partition_seqs" : {
       "1" :00,
       "3" :00,
       "0" :00,
       "2" :00
    },
    "updater_running" : false,
    "partition_seqs" : {
       "1" :00,
       "3" :00,
       "0" :00,
       "2" :00
    },
    "stats" : {
       "update_history" : [],
       "updater_cleanups" : 0,
       "compaction_history" : [],
       "full_updates" : 0,
       "waiting_clients" : 0,
       "compactions" : 0,
       "cleanups" : 0,
       "partial_updates" : 0,
       "stopped_updates" : 0,
       "cleanup_history" : [],
       "cleanup_interruptions" : 0
    },
```

```
   "initial_build" :   false,
   "replica_partitions" : [
      4,
      5,
      8,
      12
   ],
   "update_seqs" : {
      "1" : 31250,
      "3" : 31250,
      "0" : 31250,
      "2" : 31250
   },
   "partition_seqs_up_to_date" : true,
   "updater_state" :   "not_running",
   "data_size" : 5717080,
   "cleanup_running" : false,
   "signature" :   "440b0b3ded9d68abb559d58b9fda3e0a",
   "max_number_partitions" : 16,
   "disk_size" : 5726395
}
```


The output gives us several fields useful to diagnose issues in the server. The field `replica_group_info` can be ignored for the goals of this wiki (would only be useful during a failover), the information it contains is similar to the top level information, which is the one for the main/principal index, which is the one we care about during steady state and during rebalance.

Some of the top level fields and their meaning:

- `active_partitions` - this is a list with the ID of all the vbuckets marked as active in the index.
- `passive_partitions` - this is a list with the ID of all vbuckets marked as passive in the index.
- `cleanup_partitions` - this is a list with the ID of all vBuckets marked as cleanup in the index.
- `compact_running` - true if index compaction is ongoing, false otherwise.
- `updater_running` - true if index build/update is ongoing, false otherwise.
- `update_seqs` - this tells us what up to which vbucket database update_seqs the index reflects data, keys are vbucket IDs and values are update_seqs. The update_seqs here are always smaller or equal then the values in `partition_seqs` and `expected_partition_seqs`. If the value of any update_seq here is smaller than the corresponding value in `partition_seqs` or `expected_partition_seqs`, than it means the index is not up to date (it's stale), and a subsequent query with `stale=false` will be blocked and spawn an index update (if not already running).
- `partition_seqs` - this tells us what are the current update_seqs for each vbucket database. If any update_seq value here is greater than the corresponding value in `update_seqs`, we can say the index is not up to date (it's stale). See the description above for `update_seqs`.
- `expected_partition_seqs` - this should normally tells us exactly the same as `partition_seqs` (see above). Index processes have an optimization where they monitor vbucket database updates and track their current update_seqs, so that when the index needs to know them, it doesn't need to consult them from the databases (expensive, from a performance perspective). The update_seqs in this field are obtained by consulting each database file. If they don't match the corresponding values in `partition_seqs`, then we can say there's an issue in the view-engine.
- `unindexable_partitions` - this field should be non-empty only during rebalance. Vbuckets that are in this meta state "unindexable" means that index updates will ignore these vbuckets. Transitions to and from this state are used by ns_server for consistent views during rebalance. When not in rebalance, this field should always be empty, if not, then there's a issue somewhere. The value for this field, when non-empty, is an object whose keys are vbucket IDs and values are update_seqs.

Using the information given by this URL (remember, it's on a per node basis), to check the vbucket states and indexed update_seqs, together with the tools `couch_dbinfo` and `couch_dbdump` (against all active vbucket database files), one can debug where (which component) a problem is. For example, it's useful to find if it's the indexes that are not indexing latest data/updates/processing deletes, or if the memcached/ep-engine layer is not persisting data/updates to disk or if there's some issue in couchstore (component which writes to database files) that causes it to not write data or write incorrect data to the database file.

An example where using these tools and the information from the URL /_set_view/bucketname/_design/ddocid/_info was very important to find which component was misbehaving. In this case Tommie was able to identify that the problem was in ep-engine.

Related Links

Troubleshooting on page 313
Troubleshooting covers general tips, common errors, log information, and other issues.

Related topics

http://www.couchbase.com/issues/browse/MB–5534

Incorrect or missing data (user issue)

Data missing in query response or it's wrong (user issue)

For example, you defined a view with a `_stats` reduce function. You query your view, and keep getting empty results all the time, for example:

```
> curl -s 'http://localhost:9500/default/_design/dev_test3/_view/view1?
full_set=true'
{"rows":[
]
}
```

You repeat this query over and over for several minutes or even hours, and you always get an empty result set.

Try to query the view with `stale=false`, and you get:

```
> curl -s 'http://localhost:9500/default/_design/dev_test3/_view/view1?
full_set=true&stale=false'
{"rows":[
],
"errors":[
{"from":"local","reason":"Builtin _stats function
 requires map values to be numbers"},
{"from":"http://192.168.1.80:9502/_view_merge/?stale=false","reason":"Builtin
 _stats function requires map values to be
 numbers"},
{"from":"http://192.168.1.80:9501/_view_merge/?stale=false","reason":"Builtin
 _stats function requires map values to be
 numbers"},
{"from":"http://192.168.1.80:9503/_view_merge/?stale=false","reason":"Builtin
 _stats function requires map values to be
 numbers"}
]
}
```

Then looking at the design document, you see it could never work, as values are not numbers:

```
{
   "views":
   {
       "view1": {
           "map": "function(doc, meta) { emit(meta.id, meta.id); }",
           "reduce": "_stats"
       }
   }
}
```

One important question to answer is, why do you see the errors when querying with `stale=false` but do not see them when querying with `stale=update_after` (default) or `stale=ok` ? The answer is simple:

1. `stale=false` means: trigger an index update/build, and wait until it that update/build finishes, then start streaming the view results. For this example, index build/update failed, so the client gets an error, describing why it failed, from all nodes where it failed.
2. `stale=update_after` means start streaming the index contents immediately and after trigger an index update (if index is not up to date already), so query responses won't see indexing errors as they do for the `stale=false` scenario. For this particular example, the error happened during the initial index build, so the index was empty when the view queries arrived in the system, whence the empty result set.
3. `stale=ok` is very similar to (2), except it doesn't trigger index updates.

Finally, index build/update errors, related to user Map/Reduce functions, can be found in a dedicated log file that exists per node and has a file name matching `mapreduce_errors.#`. For example, from node 1, the file `*mapreduce_errors.1` contained:

```
[mapreduce_errors:error,2012-08-20T16:18:36.250,n_0@192.168.1.80:<0.2096.1>]
 Bucket `default`, main group `_design/dev_test3`,
 error executing reduce
function for view `view1'
   reason:                Builtin _stats function requires map values to be
numbers
```

Related Links

Troubleshooting on page 313
Troubleshooting covers general tips, common errors, log information, and other issues.

Design document aliases

When 2 or more design documents have exactly the same map and reduce functions (but different IDs of course), they get the same signature. This means that both point to the same index files, and it's exactly this feature that allows publishing development design documents into production, which consists of creating a copy of the development design document (ID matches _design/dev_foobar) with an ID not containing the `dev_` prefix and then deleting the original development document, which ensure the index files are preserved after deleting the development design document. It's also possible to have multiple "production" aliases for the same production design document. The view engine itself has no notion of development and production design documents, this is a notion only at the UI and cluster layers, which exploits the design document signatures/aliases feature.

The following example shows this property.

We create 2 identical design documents, only their IDs differ:

```
> curl -H 'Content-Type: application/json' \
    -X PUT 'http://localhost:9500/default/_design/ddoc1' \
    -d '{ "views": {"view1": {"map": "function(doc, meta) { emit(doc.level,
 meta.id); }"}}}'
{"ok":true,"id":"_design/ddoc1"}

> curl -H 'Content-Type: application/json' \
    -X PUT 'http://localhost:9500/default/_design/ddoc2'
    -d '{ "views": {"view1": {"map": "function(doc, meta) { emit(doc.level,
 meta.id); }"}}}'
{"ok":true,"id":"_design/ddoc2"}
```

Next we query view1 from _design/ddoc1 with `stale=false`, and get:

```
> curl -s 'http://localhost:9500/default/_design/ddoc1/_view/view1?
limit=10&stale=false'
{"total_rows":1000000,"rows":[
{"id":"0000025","key":1,"value":"0000025"},
{"id":"0000136","key":1,"value":"0000136"},
{"id":"0000158","key":1,"value":"0000158"},
{"id":"0000205","key":1,"value":"0000205"},
{"id":"0000208","key":1,"value":"0000208"},
{"id":"0000404","key":1,"value":"0000404"},
{"id":"0000464","key":1,"value":"0000464"},
```

```
{"id":"0000496","key":1,"value":"0000496"},
{"id":"0000604","key":1,"value":"0000604"},
{"id":"0000626","key":1,"value":"0000626"}
]
}
```

If immediately after you query view1 from _design/ddoc2 with `stale=ok`, you'll get exactly the same results, because both design documents are aliases, they share the same signature:

```
> curl -s 'http://localhost:9500/default/_design/ddoc2/_view/view1?
limit=10&stale=ok'
{"total_rows":1000000,"rows":[
{"id":"0000025","key":1,"value":"0000025"},
{"id":"0000136","key":1,"value":"0000136"},
{"id":"0000158","key":1,"value":"0000158"},
{"id":"0000205","key":1,"value":"0000205"},
{"id":"0000208","key":1,"value":"0000208"},
{"id":"0000404","key":1,"value":"0000404"},
{"id":"0000464","key":1,"value":"0000464"},
{"id":"0000496","key":1,"value":"0000496"},
{"id":"0000604","key":1,"value":"0000604"},
{"id":"0000626","key":1,"value":"0000626"}
]
}
```

If you look into the data directory, there's only one main index file and one replica index file:

```
> tree couch/0/\@indexes
couch/0/@indexes
 ??? default
     ???
main_1909e1541626269ef88c7107f5123feb.view.1
     ???
replica_1909e1541626269ef88c7107f5123feb.view.1
     ???
tmp_1909e1541626269ef88c7107f5123feb_main

 2 directories, 2 files
```

Also, while the indexer is running, if you query `_active_tasks` for a node, you'll see one single indexer task, which lists both design documents in the `design_documents` array field:

```
> curl -s http://localhost:9500/_active_tasks | json_xs
[
   {
      "waiting" : 0,
      "started_on" : 1345662986,
      "pid" :   "<0.234.0>",
      "type" :   "couch_main_index_barrier",
      "running" : 1,
      "limit" : 4,
      "updated_on" : 1345663590
   },
   {
      "waiting" : 0,
      "started_on" : 1345662986,
      "pid" : "<0.235.0>",
      "type" :   "couch_replica_index_barrier",
      "running" : 0,
      "limit" : 2,
      "updated_on" : 1345662986
   },
   {
      "indexer_type" : "main",
```

```
      "started_on" : 1345663590,
      "progress" : 75,
      "initial_build" : true,
      "updated_on" : 1345663634,
      "total_changes" : 250000,
      "design_documents" : [
         "_design/ddoc1",
         "_design/ddoc2"
      ],
      "pid" :   "<0.6567.0>",
      "changes_done" : 189635,
      "signature" : "1909e1541626269ef88c7107f5123feb",
      "type" :   "indexer",
      "set" :   "default"
   }
]
```

Related Links

Troubleshooting on page 313
Troubleshooting covers general tips, common errors, log information, and other issues.

Related topics

Views and indexes on page 183

Expired documents issue

Expired documents have their associated key-value pairs returned in queries with `stale=false`.

See Couchbase issuee MB-6219.

Related Links

Troubleshooting on page 313
Troubleshooting covers general tips, common errors, log information, and other issues.

Related topics

http://www.couchbase.com/issues/browse/MB–6219

Index filesystem structure

A description of the index filesystem struction.

All index files live within a subdirectory of the data directory named `@indexes`. Within this subdirectory, there's a subdirectory for each bucket (which matches exactly the bucket name).

Any index file has the form `<type>_<hexadecimal_signature>.view.N` Each component's meaning is:

- `type` - the index type, can be main (active vbuckets data) or replica (replica vbuckets data)
- `hexadecimal_signature` - this is the hexadecimal form of an MD5 hash computed over the map/reduce functions of a design document, when these functions change, a new index is created. It's possible to have multiple versions of the same design document alive (different signatures). This happens for a short period, for example a client does a `stale=false` request to an index (1 index == 1 design document), which triggers an index build/update and before this update/build finishes, the design document is updated (with different map/reduce functions). The initial version of the index will remain alive until all currently blocked clients on it are served. In the meanwhile new query requests are redirected to the latest (second) version of the index, always. This is what makes it possible to have multiple versions of the same design document index files at any point in time (however for short periods).
- `N` - when an index file is created N is 1, always. Every time the index file is compacted, N is incremented by 1. This is similar to what happens for vbucket database files Data missing in query response or it's wrong (potentially due to server issues)).

For each design document, there's also a subdirectory named like `tmp_<hexadecimal_signature>_<type>`. This is a directory containing temporary files used for the initial index build (and soon for incremental optimizations).

Files within this directory have a name formed by the design document signature and a generated UUID. These files are periodically deleted when they're not useful anymore.

All views defined within a design document are backed by a btree data structure, and they all live inside the same index file. Therefore for each design document, independently of the number of views it defines, there's 2 files, one for main data and the other for replica data.

Example:

```
> tree couch/0/\@indexes/
couch/0/@indexes/
 ??? default
     ???
main_018b83ca22e53e14d723ea858ba97168.view.1
     ???
main_15e1f576bc85e3e321e28dc883c90077.view.1
     ???
main_440b0b3ded9d68abb559d58b9fda3e0a.view.1
     ???
main_4995c136d926bdaf94fbe183dbf5d5aa.view.1
     ???
main_fd2bdf6191e61af6e801e3137e2f1102.view.1
     ???
replica_018b83ca22e53e14d723ea858ba97168.view.1
     ???
replica_15e1f576bc85e3e321e28dc883c90077.view.1
     ???
replica_440b0b3ded9d68abb559d58b9fda3e0a.view.1
     ???
replica_4995c136d926bdaf94fbe183dbf5d5aa.view.1
     ???
replica_fd2bdf6191e61af6e801e3137e2f1102.view.1
     ???
tmp_018b83ca22e53e14d723ea858ba97168_main
     ???
tmp_15e1f576bc85e3e321e28dc883c90077_main
     ???
tmp_440b0b3ded9d68abb559d58b9fda3e0a_main
     ???
tmp_4995c136d926bdaf94fbe183dbf5d5aa_main
     ???
tmp_fd2bdf6191e61af6e801e3137e2f1102_main

 6 directories, 10 files
```

Related Links

Troubleshooting on page 313
Troubleshooting covers general tips, common errors, log information, and other issues.

Related topics

Views and indexes on page 183

Index results for a single node

A specific URI is used to get index results for a single node.

There's a special URI which allows to get index results only from the targeted node. It is used only for development and debugging, not meant to be public. The following is an example where two different nodes are queried from a four node cluster.

```
> curl -s 'http://192.168.1.80:9500/_set_view/default/_design/ddoc2/_view/
view1?limit=4'
{"total_rows":250000,"offset":0,"rows":[
{"id":"0000136","key":1,"value":"0000136"},
{"id":"0000205","key":1,"value":"0000205"},
```

```
{"id":"0000716","key":1,"value":"0000716"},
{"id":"0000719","key":1,"value":"0000719"}
]}
> curl -s 'http://192.168.1.80:9500/_set_view/default/_design/ddoc2/_view/
view1?limit=4'
{"total_rows":250000,"offset":0,"rows":[
{"id":"0000025","key":1,"value":"0000025"},
{"id":"0000158","key":1,"value":"0000158"},
{"id":"0000208","key":1,"value":"0000208"},
{"id":"0000404","key":1,"value":"0000404"}
]}
```

Note: For this special API, the default value of the stale parameter is `stale=false`, while for the public, documented API the default is `stale=update_after`.

Related Links

Troubleshooting on page 313

Troubleshooting covers general tips, common errors, log information, and other issues.

Debugging replica index

Description of how to test and verify that the replica index is working.

It's not easy to test/verify from the outside that the replica index is working. Remember, replica index is optional, and it's just an optimization for faster `stale=false` queries after rebalance - it doesn't cope with correctness of the results.

There's a non-public query parameter named `_type` used only for debugging and testing. Its default value is `main`, and the other possible value is `replica`. Here follows an example of querying the main (default) and replica indexes on a 2 nodes cluster (for sake of simplicity), querying the main (normal) index gives:

```
> curl -s 'http://localhost:9500/default/_design/test/_view/view1?
limit=20&stale=false&debug=true'
{"total_rows":20000,"rows":[
{"id":"0017131","key":2,"partition":43,"node":"http://192.168.1.80:9501/
_view_merge/","value":"0017131"},
{"id":"0000225","key":10,"partition":33,"node":"http://192.168.1.80:9501/
_view_merge/","value":"0000225"},
{"id":"0005986","key":15,"partition":34,"node":"http://192.168.1.80:9501/
_view_merge/","value":"0005986"},
{"id":"0015579","key":17,"partition":27,"node":"local","value":"0015579"},
{"id":"0018530","key":17,"partition":34,"node":"http://192.168.1.80:9501/
_view_merge/","value":"0018530"},
{"id":"0006210","key":23,"partition":2,"node":"local","value":"0006210"},
{"id":"0006866","key":25,"partition":18,"node":"local","value":"0006866"},
{"id":"0019349","key":29,"partition":21,"node":"local","value":"0019349"},
{"id":"0004415","key":39,"partition":63,"node":"http://192.168.1.80:9501/
_view_merge/","value":"0004415"},
{"id":"0018181","key":48,"partition":5,"node":"local","value":"0018181"},
{"id":"0004737","key":49,"partition":1,"node":"local","value":"0004737"},
{"id":"0014722","key":51,"partition":2,"node":"local","value":"0014722"},
{"id":"0003686","key":54,"partition":38,"node":"http://192.168.1.80:9501/
_view_merge/","value":"0003686"},
{"id":"0004656","key":65,"partition":48,"node":"http://192.168.1.80:9501/
_view_merge/","value":"0004656"},
{"id":"0012234","key":65,"partition":10,"node":"local","value":"0012234"},
{"id":"0001610","key":71,"partition":10,"node":"local","value":"0001610"},
{"id":"0015940","key":83,"partition":4,"node":"local","value":"0015940"},
{"id":"0010662","key":87,"partition":38,"node":"http://192.168.1.80:9501/
_view_merge/","value":"0010662"},
{"id":"0015913","key":88,"partition":41,"node":"http://192.168.1.80:9501/
_view_merge/","value":"0015913"},
{"id":"0019606","key":90,"partition":22,"node":"local","value":"0019606"}
],
```

Note that the `debug=true` parameter, for map views, add 2 row fields, `partition` which is the vbucket ID where the document that produced this row (emitted by the map function) lives, and `node` which tells from which node in the cluster the row came (value “local” for the node which received the query, an URL otherwise).

Now, doing the same query but against the replica index (`_type=replica`) gives:

```
> curl -s 'http://localhost:9500/default/_design/test/_view/view1?
limit=20&stale=false&_type=replica&debug=true'
{"total_rows":20000,"rows":[
{"id":"0017131","key":2,"partition":43,"node":"local","value":"0017131"},
{"id":"0000225","key":10,"partition":33,"node":"local","value":"0000225"},
{"id":"0005986","key":15,"partition":34,"node":"local","value":"0005986"},
{"id":"0015579","key":17,"partition":27,"node":"http://192.168.1.80:9501/
_view_merge/","value":"0015579"},
{"id":"0018530","key":17,"partition":34,"node":"local","value":"0018530"},
{"id":"0006210","key":23,"partition":2,"node":"http://192.168.1.80:9501/
_view_merge/","value":"0006210"},
{"id":"0006866","key":25,"partition":18,"node":"http://192.168.1.80:9501/
_view_merge/","value":"0006866"},
{"id":"0019349","key":29,"partition":21,"node":"http://192.168.1.80:9501/
_view_merge/","value":"0019349"},
{"id":"0004415","key":39,"partition":63,"node":"local","value":"0004415"},
{"id":"0018181","key":48,"partition":5,"node":"http://192.168.1.80:9501/
_view_merge/","value":"0018181"},
{"id":"0004737","key":49,"partition":1,"node":"http://192.168.1.80:9501/
_view_merge/","value":"0004737"},
{"id":"0014722","key":51,"partition":2,"node":"http://192.168.1.80:9501/
_view_merge/","value":"0014722"},
{"id":"0003686","key":54,"partition":38,"node":"local","value":"0003686"},
{"id":"0004656","key":65,"partition":48,"node":"local","value":"0004656"},
{"id":"0012234","key":65,"partition":10,"node":"http://192.168.1.80:9501/
_view_merge/","value":"0012234"},
{"id":"0001610","key":71,"partition":10,"node":"http://192.168.1.80:9501/
_view_merge/","value":"0001610"},
{"id":"0015940","key":83,"partition":4,"node":"http://192.168.1.80:9501/
_view_merge/","value":"0015940"},
{"id":"0010662","key":87,"partition":38,"node":"local","value":"0010662"},
{"id":"0015913","key":88,"partition":41,"node":"local","value":"0015913"},
{"id":"0019606","key":90,"partition":22,"node":"http://192.168.1.80:9501/
_view_merge/","value":"0019606"}
],
```

Note that you get exactly the same results (id, key and value for each row). Looking at the row field `node`, you can see there’s a duality when compared to the results we got from the main index, which is very easy to understand for the simple case of a 2 nodes cluster.

Related Links

Troubleshooting on page 313
Troubleshooting covers general tips, common errors, log information, and other issues.

Related topics

Incorrect or missing data (server issue) on page 321
Data missing in query response or it’s wrong (potentially due to server issues)

Debugging stale=false queries

Debugging stale=false queries for missing/unexpected data

The query parameter `debug=true` can be used to debug queries with `stale=false` that are not returning all expected data or return unexpected data. This is particularly useful when clients issue a `stale=false` query right after being unblocked by a memcached OBSERVE command. .

Here follows an example of how to debug this sort of issues on a simple scenario where there's only 16 vbuckets (instead of 1024) and 2 nodes. The tools `couchdb_dump` and `couchdb_info` (from the couchstore git project) are used to help analyze this type of issues (available under `install/bin` directory).

Querying a view with `debug=true` will add an extra field, named `debug_info` in the view response. This field has one entry per node in the cluster (if no errors happened, like down/timed out nodes for example). Example:

```
> curl -s 'http://localhost:9500/default/_design/test/_view/view1?
stale=false&limit=5&debug=true' | json_xs
 {
    "debug_info" : {
       "local" : {
          "main_group" : {
             "passive_partitions" : [],
             "wanted_partitions" : [
                0,
                1,
                2,
                3,
                4,
                5,
                6,
                7
             ],
             "wanted_seqs" : {
                "0002" :00,
                "0001" :00,
                "0006" :00,
                "0005" :00,
                "0004" :00,
                "0000" :00,
                "0007" :00,
                "0003" :00
             },
             "indexable_seqs" : {
                "0002" :00,
                "0001" :00,
                "0006" :00,
                "0005" :00,
                "0004" :00,
                "0000" :00,
                "0007" :00,
                "0003" :00
             },
             "cleanup_partitions" : [],
             "stats" : {
                "update_history" : [
                   {
                      "deleted_ids" : 0,
                      "inserted_kvs" :00,
                      "inserted_ids" :00,
                      "deleted_kvs" : 0,
                      "cleanup_kv_count" : 0,
                      "blocked_time" : 0.000258,
                      "indexing_time" : 103.222201
                   }
                ],
                "updater_cleanups" : 0,
                "compaction_history" : [],
                "full_updates" : 1,
                "accesses" : 1,
                "cleanups" : 0,
                "compactions" : 0,
                "partial_updates" : 0,
```

```
            "stopped_updates" : 0,
            "cleanup_history" : [],
            "update_errors" : 0,
            "cleanup_stops" : 0
         },
         "active_partitions" : [
            0,
            1,
            2,
            3,
            4,
            5,
            6,
            7
         ],
         "pending_transition" : null,
         "unindexeable_seqs" : {},
         "replica_partitions" : [
            8,
            9,
            10,
            11,
            12,
            13,
            14,
            15
         ],
         "original_active_partitions" : [
            0,
            1,
            2,
            3,
            4,
            5,
            6,
            7
         ],
         "original_passive_partitions" : [],
         "replicas_on_transfer" : []
      }
   },
   "http://10.17.30.98:9501/_view_merge/" :   {
      "main_group" : {
         "passive_partitions" : [],
         "wanted_partitions" : [
            8,
            9,
            10,
            11,
            12,
            13,
            14,
            15
         ],
         "wanted_seqs" : {
            "0008" :00,
            "0009" :00,
            "0011" :00,
            "0012" :00,
            "0015" :00,
            "0013" :00,
            "0014" :00,
            "0010" :00
         },
```

```
        "indexable_seqs" : {
           "0008" :00,
           "0009" :00,
           "0011" :00,
           "0012" :00,
           "0015" :00,
           "0013" :00,
           "0014" :00,
           "0010" :00
        },
        "cleanup_partitions" : [],
        "stats" : {
           "update_history" : [
              {
                 "deleted_ids" : 0,
                 "inserted_kvs" :00,
                 "inserted_ids" :00,
                 "deleted_kvs" : 0,
                 "cleanup_kv_count" : 0,
                 "blocked_time" : 0.000356,
                 "indexing_time" : 103.651148
              }
           ],
           "updater_cleanups" : 0,
           "compaction_history" : [],
           "full_updates" : 1,
           "accesses" : 1,
           "cleanups" : 0,
           "compactions" : 0,
           "partial_updates" : 0,
           "stopped_updates" : 0,
           "cleanup_history" : [],
           "update_errors" : 0,
           "cleanup_stops" : 0
        },
        "active_partitions" : [
           8,
           9,
           10,
           11,
           12,
           13,
           14,
           15
        ],
        "pending_transition" : null,
        "unindexeable_seqs" : {},
        "replica_partitions" : [
           0,
           1,
           2,
           3,
           4,
           5,
           6,
           7
        ],
        "original_active_partitions" : [
           8,
           9,
           10,
           11,
           12,
           13,
```

```
                14,
                15
            ],
            "original_passive_partitions" : [],
            "replicas_on_transfer" : []
        }
    }
},
"total_rows" : 1000000,
"rows" : [
    {
        "value" : {
            "ratio" : 1.8,
            "type" : "warrior",
            "category" : "orc"
        },
        "id" : "0000014",
        "node" : "http://10.17.30.98:9501/_view_merge/",
        "partition" : 14,
        "key" : 1
    },
    {
        "value" : {
            "ratio" : 1.8,
            "type" : "warrior",
            "category" : "orc"
        },
        "id" : "0000017",
        "node" : "local",
        "partition" : 1,
        "key" : 1
    },
    {
        "value" : {
            "ratio" : 1.8,
            "type" : "priest",
            "category" : "human"
        },
        "id" : "0000053",
        "node" : "local",
        "partition" : 5,
        "key" : 1
    },
    {
        "value" : {
            "ratio" : 1.8,
            "type" : "priest",
            "category" : "orc"
        },
        "id" : "0000095",
        "node" : "http://10.17.30.98:9501/_view_merge/",
        "partition" : 15,
        "key" : 1
    },
    {
        "value" : {
            "ratio" : 1.8,
            "type" : "warrior",
            "category" : "elf"
        },
        "id" : "0000151",
        "node" : "local",
        "partition" : 7,
        "key" : 1
```

```
        }
    ]
}
```

For each node, there are 2 particular fields of interest when debugging `stale=false` queries that apparently miss some data:

- `wanted_seqs` - This field has an object (dictionary) value where keys are vbucket IDs and values are vbucket database sequence numbers for an explanation of sequence numbers). This field tells us the sequence number of each vbucket database file (at the corresponding node) at the moment the query arrived at the server (all these vbuckets are `active vbuckets`).
- `indexable_seqs` - This field has an object (dictionary) value where keys are vbucket IDs and values are vbucket database sequence numbers. This field tells us, for each active vbucket database, up to which sequence the index has processed/indexed documents (remember, each vbucket database sequence number is associated with 1, and only 1, document).

For queries with `stale=false`, all the sequences in `indexable_seqs` must be greater or equal then the sequences in `wanted_seqs` - otherwise the `stale=false` option can be considered broken. What happens behind the scenes is, at each node, when the query request arrives, the value for `wanted_seqs` is computed (by asking each active vbucket database for its current sequence number), and if any sequence is greater than the corresponding entry in `indexable_seqs` (stored in the index), the client is blocked, the indexer is started to update the index, the client is unblocked when the indexer finishes updating the index, and finally the server starts streaming rows to the client - note that at this point, all sequences in `indexable_seqs` are necessarily greater or equal then the corresponding sequences in `wanted_sequences`, otherwise the `stale=false` implementation is broken.

Related Links

Troubleshooting on page 313

Troubleshooting covers general tips, common errors, log information, and other issues.

Related topics

MB–7161

Timeout errors

Timeout errors when querying a view with `stale=false`.

When querying a view with `stale=false`, you get often timeout errors for one or more nodes. These nodes are nodes that did not receive the original query request, for example you query node 1, and you get timeout errors for nodes 2, 3 and 4 as in the example below (view with reduce function _count):

```
> curl -s 'http://localhost:9500/default/_design/dev_test2/_view/view2?
full_set=true&stale=false'
{"rows":[
  {"key":null,"value":125184}
],
"errors":[
  {"from":"http://192.168.1.80:9503/_view_merge/?
stale=false","reason":"timeout"},
  {"from":"http://192.168.1.80:9501/_view_merge/?
stale=false","reason":"timeout"},
  {"from":"http://192.168.1.80:9502/_view_merge/?
stale=false","reason":"timeout"}
]
}
```

The problem here is that by default, for queries with `stale=false` (full consistency), the view merging node (node which receive the query request, node 1 in this example) waits up to 60000 milliseconds (1 minute) to receive partial view results from each other node in the cluster. If it waits for more than 1 minute for results from a remote node, it stops waiting for results from that node and a timeout error entry is added to the final response. A `stale=false` request blocks a client, or the view merger node as in this example, until the index is up to date, so these timeouts can happen frequently.

If you look at the logs from those nodes you got a timeout error, you'll see the index build/update took more than 60 seconds, example from node 2:

```
[couchdb:info,2012-08-20T15:21:13.150,n_1@192.168.1.80:<0.6234.0>:couch_log:info:39]
 Set view
  `default`, main group `_design/dev_test2`, updater finished
 Indexing time: 93.734 seconds
 Blocked time:  10.040 seconds
 Inserted IDs:  124960
 Deleted IDs:   0
 Inserted KVs:  374880
 Deleted KVs:   0
 Cleaned KVs:   0
```

In this case, node 2 took 103.774 seconds to update the index.

In order to avoid those timeouts, you can pass a large connection_timeout in the view query URL, example:

```
> time curl -s
 'http://localhost:9500/default/_design/dev_test2/_view/view2?
full_set=true&stale=false&connection_timeout=999999999'
{"rows":[
{"key":null,"value":2000000}
]
}
real  2m44.867s
user   0m0.007s
sys    0m0.007s
```

And in the logs of nodes 1, 2, 3 and 4, respectively you'll see something like this:

```
node 1, view merger node
```

```
[couchdb:info,2012-08-20T16:10:02.887,n_0@192.168.1.80:<0.27674.0>:couch_log:info:39]
 Set view
  `default`, main group `_design/dev_test2`, updater
finished
 Indexing time: 155.549
seconds
 Blocked time:  0.000 seconds
 Inserted IDs:96
 Deleted IDs:   0
 Inserted KVs:  1500288
 Deleted KVs:   0
 Cleaned KVs:   0
```

```
node 2
```

```
[couchdb:info,2012-08-20T16:10:28.457,n_1@192.168.1.80:<0.6071.0>:couch_log:info:39]
 Set view
  `default`, main group `_design/dev_test2`, updater
finished
 Indexing time: 163.555
seconds
 Blocked time:  0.000 seconds
 Inserted IDs:  499968
 Deleted IDs:   0
 Inserted KVs:  1499904
 Deleted KVs:   0
 Cleaned KVs:   0
```

```
node 3
```

```
[couchdb:info,2012-08-20T16:10:29.710,n_2@192.168.1.80:<0.6063.0>:couch_log:info:39]
 Set view
  `default`, main group `_design/dev_test2`, updater
finished
```

```
 Indexing time: 164.808
seconds
 Blocked time:  0.000 seconds
 Inserted IDs:  499968
 Deleted IDs:   0
 Inserted KVs:  1499904
 Deleted KVs:   0
 Cleaned KVs:   0
```

node 4

```
[couchdb:info,2012-08-20T16:10:26.686,n_3@192.168.1.80:<0.6063.0>:couch_log:info:39]
 Set view
  `default`, main group `_design/dev_test2`, updater
finished
 Indexing time: 161.786
seconds
 Blocked time:  0.000 seconds
 Inserted IDs:  499968
 Deleted IDs:   0
 Inserted KVs:  1499904
 Deleted KVs:   0
 Cleaned KVs:   0
```

Related Links

Troubleshooting on page 313
Troubleshooting covers general tips, common errors, log information, and other issues.

total_rows values are too high

There are cases where the `total_rows` value is higher than expected.

In some scenarios, it's expected to see queries returning a `total_rows` field with a value higher than the maximum rows they can return (map view queries without an explicit `limit`, `skip`, `startkey` or `endkey`).

The expected scenarios are during rebalance, and immediately after a failover for a finite period of time.

This happens because in these scenarios some vbuckets are marked for cleanup in the indexes, temporarily marked as passive, or data is being transferred from the replica index to the main index (after a failover). While the rows originated from those vbuckets are never returned to queries, they contribute to the reduction value of every view btree, and this value is what is used for the `total_rows` field in map view query responses (it's simply a counter with total number of Key-Value pairs per view).

Ensuring that `total_rows` always reflected the number of rows originated from documents in active vbuckets would be very expensive, severely impacting performance. For example, we would need to maintain a different value in the btree reductions which would map vbucket IDs to row counts:

```
{"0":56, "1": 2452435, ..., "1023": 432236}
```

This would significantly reduce the btrees branching factor, making them much more deep, using more disk space and taking more time to compute reductions on inserts/updates/deletes.

To know if there are vbuckets under cleanup, vbuckets in passive state or vbuckets being transferred from the replica index to main index (on failover), one can query the following URL:

```
> curl -s 'http://localhost:9500/_set_view/default/_design/dev_test2/_info' |
 json_xs
{
   "passive_partitions" : [1, 2, 3],
   "cleanup_partitions" : [],
   "replicas_on_transfer" : [1, 2, 3],
   (....)
}
```

Note that the example above intentionally hides all non-relevant fields. If any of the fields above is a non-empty list, than `total_rows` for a view may be higher than expected, that is, we're under one of those expected scenarios mentioned above. In steady state all of the above fields are empty lists.

Related Links

Troubleshooting on page 313
Troubleshooting covers general tips, common errors, log information, and other issues.

Wrong documents or rows issue

The wrong documents or rows are returned when querying with `include_docs=true`.

Imagine you have the following design document:

```
{
     "meta": {"id": "_design/test"},
     "views":
     {
         "view1": {
             "map": "function(doc, meta) { emit(meta.id,  doc.value); }"
         }
     }
 }
```

And the bucket only has 2 documents, document `doc1` with JSON value `{"value": 1}`, and document `doc2` with JSON value `{"value": 2}`, you query the view initially with `stale=false` and `include_docs=true` and get:

```
> curl -s 'http://localhost:9500/default/_design/test/_view/view1?
include_docs=true&stale=false' | json_xs
 {
    "total_rows" :
2,
    "rows" :
[
       {
          "value" : 1,
          "doc"
: {
             "json" : {
                "value" : 1
             },
             "meta" : {
                "flags" : 0,
                "expiration" : 0,
                "rev" : "1-000000367916708a0000000000000000",
                "id" : "doc1"
             }
          },
          "id"
: "doc1",
          "key"
: "doc1"
       },
       {
          "value" : 2,
          "doc"
: {
             "json" : {
                "value" : 2
             },
             "meta" : {
                "flags" : 0,
                "expiration" : 0,
```

```
                "rev" : "1-00000037b8a32e420000000000000000",
                "id" : "doc2"
             }
          },
          "id"
: "doc2",
          "key"
: "doc2"
       }
    ]
 }
```

Later on you update both documents, such that document `doc1` has the JSON value `{"value": 111111}` and document `doc2` has the JSON value `{"value": 222222}`. You then query the view with `stale=update_after` (default) or `stale=ok` and get:

```
> curl -s 'http://localhost:9500/default/_design/test/_view/view1?
include_docs=true' | json_xs
 {
    "total_rows" :
2,
    "rows" :
[
       {
          "value" : 1,
          "doc"
: {
             "json" : {
                "value" : 111111
             },
             "meta" : {
                "flags" : 0,
                "expiration" : 0,
                "rev" : "2-0000006657aeed6e0000000000000000",
                "id" : "doc1"
             }
          },
          "id"
: "doc1",
          "key"
: "doc1"
       },
       {
          "value" : 2,
          "doc"
: {
             "json" : {
                "value" : 222222
             },
             "meta" : {
                "flags" : 0,
                "expiration" : 0,
                "rev" : "2-00000067e3ee42620000000000000000",
                "id" : "doc2"
             }
          },
          "id"
: "doc2",
          "key"
: "doc2"
       }
    ]
 }
```

The documents included in each row don't match the value field of each row, that is, the documents included are the latest (updated) versions but the index row values still reflect the previous (first) version of the documents.

Why this behavior? Well, `include_docs=true` works at query time, for each row, to fetch from disk the latest revision of each document. There's no way to include a previous revision of a document. Previous revisions are not accessible through the latest vbucket databases MVCC snapshots, and it's not possible to find efficiently from which previous MVCC snapshots of a vbucket database a specific revision of a document is located. Further, vbucket database compaction removes all previous MVCC snapshots (document revisions). In short, this is a deliberate design limit of the database engine.

The only way to ensure full consistency here is to include the documents themselves in the values emitted by the map function. Queries with `stale=false` are not 100% reliable either, as just after the index is updated and while rows are being streamed from disk to the client, document updates and deletes can still happen, resulting in the same behavior as in the given example.

Related Links

Troubleshooting on page 313
Troubleshooting covers general tips, common errors, log information, and other issues.

Related topics

http://en.wikipedia.org/wiki/Multiversion_concurrency_control

Deprecated items

The following items are deprecated, will be deprecated, or are unsupported.

Platforms

The following operating systems were or will be deprecated.

Table 15: Deprecated operating systems

Operating system	Description	Deprecated version	Unsupported version
Linux	32-bit operating systems (CentOS, Ubuntu, RHEL) will not be supported.	2.5.x	3.0.0
Windows	32-bit operating systems will only be supported for development purposes. 32-bit production systems will not be supported.	2.5.x	3.0.0
CentOS 5	CentOS 5 will not be supported after Couchbase Server version 3.0.0.	3.0.0	3.0.x
Ubuntu 10.04	Ubuntu 10.04 will not be supported after Couchbase Server version 3.0.0.	3.0.0	3.0.x

REST API

The following REST API URI was or will be deprecated. To obtain information about nodes in a Couchbase cluster, use the `/pools/default/buckets/default` URI.

Table 16: Deprecated REST API URI

REST API	URI	Description	Deprecated version	Unsupported version
Server nodes	/pools/nodes	URI for obtaining information about nodes in a Couchbase cluster.	3.0.0	3.0.x

CLI tools and parameters

The following tools parameters are deprecated, removed, or not supported.

Table 17: Deprecated tools

Tool	Description	Deprecated version	Unsupported version
cbadm-online-restore	Removed	2.0	2.0
cbadm-online-update	Removed	2.0	2.0
cbadm_tap-registration	Removed	2.0	2.0
cbbackup-incremental	Removed	2.0	2.0
cbbackup-merge-incremental	Removed	2.0	2.0
cbdbconvert	Removed	2.0	2.0
cbdbmaint	Removed	2.0	2.0

Tool	Description	Deprecated version	Unsupported version
cbdbupgrade	Removed	2.0	2.0
cbflushctl	Replaced by cbepctl	2.0	2.0
tap.py		1.8	1.8
cbclusterstats	Replaced by cbstats	1.8	1.8.1
membase	Replaced by couchbase-cli	1.8	1.8.1
mbadm-online-restore	Replaced by cbadm-online-restore	1.8	1.8.1
mbadm-online-update	Replaced by cbadm-online-update	1.8	1.8.1
mbadm-tap-registration	Replaced by cbadm_tap-registration	1.8	1.8.1
mbbackup-incremental	Replaced by cbbackup-incremental	1.8	1.8.1
mbbackup-merge-incremental	Replaced by cbbackup-merge-incremental	1.8	1.8.1
mbbackup	Replaced by cbbackup	1.8	1.8.1
mbbrowse_logs	Replaced by cbbrowse_logs	1.8	1.8.1
mbcollect_info	Replaced by cbcollect_info	1.8	1.8.1
mbdbconvert	Replaced by cbdbconvert	1.8	1.8.1
mbdbmaint	Replaced by cbdbmaint	1.8	1.8.1
mbdbupgrade	Replaced by cbdbupgrade	1.8	1.8.1
mbdumpconfig.escript	Replaced by cbdumpconfig.escript	1.8	1.8.1
mbenable_core_dumps.sh	Replaced by cbenable_core_dumps.sh	1.8	1.8.1
mbflushctl	Replaced by cbflushctl	1.8	1.8.1
mbrestore	Replaced by cbrestore	1.8	1.8.1
mbstats	Replaced by cbstats	1.8	1.8.1
mbupgrade	Replaced by cbupgrade	1.8	1.8.1
mbvbucketctl	Replaced by cbbucketctl	1.8	1.8.1

The following CLI parameter is deprecated.

Table 18: Deprecated CLI parameter

Tool	Parameter	Description	Deprecated version	Unsupported version
cbepctl	flush_param flushall_enabled	The flushall_enabled parameter is deprecated.	2.2	3.x

The following CLI tools are visible but not supported by Couchbase Technical Support. These tools are for Couchbase internal use only.

- cbbrowse_logs
- cbdump-config
- cbenable_core_dumps.sh
- couch_compact
- couch_dbdump

- couch_dbinfo

Miscellaneous

The following items are deprecated or not supported.

- The `_all_docs` view is not supported
- RightScale Server non-Chef templates are deprecated as of Couchbase Server 2.2. Couchbase provides RightScale Server templates based on Chef.

The `_all_docs` view is not supported

Related Links

Couchbase Administration on page 89

Couchbase CLI

The Couchbase Server 3.0 command-line interface (CLI) provides the following topics:

Related Links

The `cbbackup` tool creates a copy of data from an entire running cluster, an entire bucket, a single node, or a single bucket on a single functioning node. Your node or cluster needs to be functioning in order to create the backup. Couchbase Server will write a copy of data onto disk.

Command-line interface overview

Couchbase Server includes a number of command-line tools that can be used to manage and monitor a Couchbase Server cluster or server. All operations are mapped to their appropriate REST API call (where available).

There are a number of command-line tools that perform different functions and operations, these are described individually within the following sections. Tools can be located in a number of directories, dependent on the tool in question in each case.

Command line tools and availability

As of Couchbase Server 2.0, the following publicly available tools have been renamed, consolidated or removed. This is to provide better usability, and reduce the number of commands required to manage Couchbase Server:

By default, the command-line tools are installed into the following locations on each platform:

Operating System	Directory Locations
Linux	`/opt/couchbase/bin`, `/opt/couchbase/bin/install`, `/opt/couchbase/bin/tools`, `/opt/couchbase/bin/tools/unsupported`
Windows	`C:\Program Files\couchbase\server\bin`, `C:\Program Files\couchbase\server\bin\install`, and `C:\Program Files\couchbase\server\bin\tools`.
Mac OS X	`/Applications/Couchbase Server.app/Contents/Resources/couchbase-core/bin`

Unsupported tools

The following are tools that are visible in Couchbase Server 2.0 installation; however the tools are unsupported. This means they are meant for Couchbase internal use and will not be supported by Couchbase Technical Support:

- `cbbrowse_logs`
- `cbdump-config`
- `cbenable_core_dumps.sh`
- `couch_compact`
- `couch_dbdump`
- `couch_dbinfo`
- `memslap`

Deprecated and removed tools

The following are tools that existed in previous versions but have been deprecated and removed as of Couchbase Server 1.8:

Tool	Server Versions	Description/Status
`tap.py`	1.8	Deprecated in 1.8.
`cbclusterstats`	1.8	Deprecated in 1.8. Replaced by `cbstats` in 1.8.
`membase`	1.7	Deprecated in 1.8. Replaced by `couchbase-cli` in 1.8.1

Tool	Server Versions	Description/Status
`mbadm-online-restore`	1.7	Deprecated in 1.8. Replaced by `cbadm-online-restore` in 1.8.1
`membase`	1.7	Deprecated in 1.8, replaced by `couchbase-cli`
`mbadm-online-restore`	1.7	Deprecated in 1.8, replaced by `cbadm-online-restore`
`mbadm-online-update`	1.7	Deprecated in 1.8, replaced by `cbadm-online-update`
`mbadm-tap-registration`	1.7	Deprecated in 1.8, replaced by `cbadm-tap-registration`
`mbbackup-incremental`	1.7	Deprecated in 1.8, replaced by `cbbackup-incremental`
`mbbackup-merge-incremental`	1.7	Deprecated in 1.8, replaced by `cbbackup-merge-incremental`
`mbbackup`	1.7	Deprecated in 1.8, replaced by `cbbackup`
`mbbrowse_logs`	1.7	Deprecated in 1.8, replaced by `cbbrowse_logs`
`mbcollect_info`	1.7	Deprecated in 1.8, replaced by `cbcollect_info`
`mbdbconvert`	1.7	Deprecated in 1.8, replaced by `cbdbconvert`
`mbdbmaint`	1.7	Deprecated in 1.8, replaced by `cbdbmaint`
`mbdbupgrade`	1.7	Deprecated in 1.8, replaced by `cbdbupgrade`
`mbdumpconfig.escript`	1.7	Deprecated in 1.8, replaced by `cbdumpconfig.escript`
`mbenable_core_dumps.sh`	1.7	Deprecated in 1.8, replaced by `cbenable_core_dumps.sh`
`mbflushctl`	1.7	Deprecated in 1.8, replaced by `cbflushctl`
`mbrestore`	1.7	Deprecated in 1.8, replaced by `cbrestore`
`mbstats`	1.7	Deprecated in 1.8, replaced by `cbstats`
`mbupgrade`	1.7	Deprecated in 1.8, replaced by `cbupgrade`
`mbvbucketctl`	1.7	Deprecated in 1.8, replaced by `cbvbucketctl`

Related Links

Couchbase CLI on page 352

Related topics

REST API overview on page 426

cbanalyze-core tool

Helper script to parse and analyze core dump from a Couchbase node. Depending upon your platform, this tool is at the following locations:

Operating System	Location
Linux	`/opt/couchbase/bin/tools/`
Windows	Not Available on this platform.
Mac OS X	`/Applications/Couchbase Server.app/Contents/Resources/couchbase-core/bin/tools/`

Related Links

Couchbase CLI on page 352

cbbackup tool

The `cbbackup` tool creates a copy of data from an entire running cluster, an entire bucket, a single node, or a single bucket on a single functioning node. Your node or cluster needs to be functioning in order to create the backup. Couchbase Server will write a copy of data onto disk.

`cbbackup`, `cbrestore` and `cbtransfer` do not communicate with external IP addresses for server nodes outside of a cluster. They can only communicate with nodes from a node list obtained within a cluster. You should perform backup, restore, or transfer to data from a node within a Couchbase cluster. This also means that if you install Couchbase Server with the default IP address, you cannot use an external hostname to access it.

Depending upon your platform, this tool is the following directories:

Linux	`/opt/couchbase/bin/cbbackup`
Windows	`C:\Program Files\Couchbase\Server\bin\cbbackup`
Mac OS X	`/Applications/Couchbase Server.app/Contents/Resources/couchbase-core/bin/cbbackup`

The format of the `cbbackup` command is:

```
cbbackup [options] [source] [destination]
```

Where:

- `[options]`

Same options available for `cbtransfer` tool.

- `[source]`

Source for the backup. This can be either a URL of a node when backing up a single node or the cluster, or a URL specifying a directory where the data for a single bucket is located.

- `[destination]`

The destination directory for the backup files to be stored. Either the directory must exist, and be empty, or the directory will be created. The parent directory must exist.

This tool has several different options which you can use to:

- Backup all buckets in an entire cluster,
- Backup one named bucket in a cluster,
- Backup all buckets on a node in a cluster,
- Backup one named buckets on a specified node,

All command options for `cbbackup` are the same options available for `cbtransfer`. For a list of standard and special-use options, see `cbtransfer` yool.

You can backup an entire cluster, which includes all of the data buckets and data at all nodes. This will also include all design documents; do note however that you will need to rebuild any indexes after you restore the data. To backup an entire cluster and all buckets for that cluster:

```
> cbbackup http://HOST:8091 ~/backups \
          -u Administrator -p password
```

Where `~/backups` is the directory where you want to store the data. When you perform this operation, be aware that cbbackup will create the following directory structure and files in the `~/backups` directory assuming you have two buckets in your cluster named `my_name` and `sasl` and two nodes `N1` and `N2` :

```
~/backups
        bucket-my_name
```

```
            N1
            N2
        bucket-sasl
            N1
            N2
```

Where `bucket-my_name` and `bucket-sasl` are directories containing data files and where `N1` and `N2` are two sets of data files for each node in the cluster. To backup a single bucket in a cluster:

```
> cbbackup http://HOST:8091 /backups/backup-20120501 \
  -u Administrator -p password \
  -b default
```

In this case `-b default` specifies you want to backup data from the default bucket in a cluster. You could also provide any other given bucket in the cluster that you want to backup. To backup all the data stored in multiple buckets from a single node which access the buckets:

```
> cbbackup http://HOST:8091 /backups/ \
  -u Administrator -p password \
  --single-node
```

This is an example of how to backup data from a single bucket on a single node follows:

```
> cbbackup http://HOST:8091 /backups \
  -u Administrator -p password \
  --single-node \
  -b bucket_name
```

This example shows you how you can specify keys that are backed up using the `- k` option. For example, to backup all keys from a bucket with the prefix ‘object’:

```
> cbbackup http://HOST:8091 /backups/backup-20120501 \
  -u Administrator -p password \
  -b bucket_name \
  -k '^object.*'
```

For more information on using `cbbackup` scenarios when you may want to use it and best practices for backup and restore of data with Couchbase Server.

Backing Up Design Documents Only

You can backup only design documents from a cluster or bucket with the option, `design_doc_only=1`. You can later restore the design documents only with `cbrestore`:

```
> ./cbbackup http://10.5.2.30:8091 ~/backup -x design_doc_only=1 -b
 bucket_name

transfer design doc only. bucket msgs will be skipped.
done
```

Where you provide the hostname and port for a node in the cluster. This will make a backup copy of all design documents from `bucket_name` and store this as `design.json` in the directory `~/backup/bucket_name`. If you do not provide a named bucket it will backup design documents for all buckets in the cluster. In this example we did a backup of two design documents on a node and our file will appear as follows:

```
[
   {
      "controllers":{
         "compact":"/pools/default/buckets/default/ddocs/_design%2Fddoc1/
controller/compactView",
         "setUpdateMinChanges":"/pools/default/buckets/default/ddocs/_design
%2Fddoc1/controller/setUpdateMinChanges"
      },
      "doc":{
         "json":{
            "views":{
               "view1":{
```

```
                  "map":"function(doc){emit(doc.key,doc.key_num);}"
               },
               "view2":{
                  "map":"function(doc,meta){emit(meta.id,doc.key);}"
               }
            }
         },
         "meta":{
            "rev":"1-6f9bfe0a",
            "id":"_design/ddoc1"
         }
      }
   },
   {
      "controllers":{
         "compact":"/pools/default/buckets/default/ddocs/_design%2Fddoc2/
controller/compactView",
         "setUpdateMinChanges":"/pools/default/buckets/default/ddocs/_design
%2Fddoc2/controller/setUpdateMinChanges"
      },
      "doc":{
         "json":{
            "views":{
               "dothis":{
                  "map":"function (doc, meta) {\n  emit(meta.id, null);\n}"
               }
            }
         },
         "meta":{
            "rev":"1-4b533871",
            "id":"_design/ddoc2"
         }
      }
   },
   {
      "controllers":{
         "compact":"/pools/default/buckets/default/ddocs/_design%2Fdev_ddoc2/
controller/compactView",
         "setUpdateMinChanges":"/pools/default/buckets/default/ddocs/_design
%2Fdev_ddoc2/controller/setUpdateMinChanges"
      },
      "doc":{
         "json":{
            "views":{
               "dothat":{
                  "map":"function (doc, meta) {\n  emit(meta.id, null);\n}"
               }
            }
         },
         "meta":{
            "rev":"1-a8b6f59b",
            "id":"_design/dev_ddoc2"
         }
      }
   }
]
```


Using cbbackup from Couchbase Server 2.0 with 1.8.x

You can use `cbbackup` 2.x to backup data from a Couchbase 1.8.x cluster, including 1.8. To do so you use the same command options you use when you backup a 2.0 cluster except you provide it the hostname and port for the 1.8.x cluster. You do not need to even install Couchbase Server 2.0 in order to use `cbbackup 2.x` to backup Couchbase Server 1.8.x. After you get the tool, go to the directory where you cloned the tool and perform the command. For instance:

```
./cbbackup http://1.8_host_name:port ~/backup -u Administrator -p password
```

This creates a backup of all buckets in the 1.8 cluster at `~/backups` on the physical machine where you run `cbbackup`. So if you want to make the backup on the machine containing the 1.8.x data bucket, you should copy the tool on that machine. As in the case where you perform backup with Couchbase 2.0, you can use `cbbackup 2.0` options to backup all buckets in a cluster, backup a named bucket, backup the default bucket, or backup the data buckets associated with a single node.

Note: The `cbrestore` tool can also be used to restore backup data onto a 1.8.x cluster.

Related Links

Couchbase CLI on page 352

Related topics

Using hostnames on page 69

couchbase-cli tool on page 361

cbrestore tool on page 390

cbtransfer tool on page 420

Couchbase command-line tools GitHub repository

couchbase-cli tool

The couchbase-cli tool is located in the following paths, depending upon the platform. This tool can perform operations on an entire cluster, on a bucket shared across an entire cluster, or on a single node in a cluster. For instance, if this tool is used to create a data bucket, all nodes in the cluster have access the bucket.

Note

Many of these same settings can be performed using the REST API.

Operating System	Directory Locations
Linux	`/opt/couchbase/bin/couchbase-cli`
Windows	`C:\Program Files\Couchbase\Server\bin\couchbase-cli.exe`
Mac OS X	`/Applications/Couchbase Server.app/Contents/Resources/couchbase-core/bin/couchbase-cli`

This tool provides access to various management operations for Couchbase Server clusters, nodes and buckets. The basic usage format is:

```
couchbase-cli COMMAND [BUCKET_NAME] CLUSTER [OPTIONS]
```

Where:

- `COMMAND` is a command listed below.
- `CLUSTER` is a cluster specification. The following shows both short and long form syntax:

```
// Short form
    -c HOST[:PORT]
// Long form
    --cluster=HOST[:PORT]
```

- `OPTIONS` are zero or more options as follows:

Option	Description
`-u USERNAME, --user=USERNAME`	Admin username of the cluster
`-p PASSWORD, --password=PASSWORD`	Admin password of the cluster
`-o KIND, --output=KIND`	Type of document: JSON or standard
`-d, --debug`	Output debug information

Related Links

Couchbase CLI on page 352

couchbase-cli commands on page 362

Buckets CLI on page 367
Buckets are managed with the `couchbase-cli` tool and the `bucket-*` commands.

Rack-Zone Awareness CLI on page 369
Rack-Zone Awareness allows logical groupings of servers on a cluster where each server group physically belongs to a rack or availability zone.

Server nodes CLI on page 371

XDCR CLI on page 372

couchbase-cli commands

Command	Description
`server-list`	List all servers in a cluster
`server-info`	Show details on one server
`server-add`	Add one or more servers to the cluster
`server-readd`	Readds a server that was failed over
`group-manage`	Manages server groups (Enterprise Edition only)
`rebalance`	Start a cluster rebalancing
`rebalance-stop`	Stop current cluster rebalancing
`rebalance-status`	Show status of current cluster rebalancing
`failover`	Failover one or more servers
`cluster-init`	Set the username,password and port of the cluster
`cluster-edit`	Modify cluster settings
`node-init`	Set node specific parameters
`bucket-list`	List all buckets in a cluster
`bucket-create`	Add a new bucket to the cluster
`bucket-edit`	Modify an existing bucket
`bucket-delete`	Delete an existing bucket
`bucket-flush`	Flush all data from disk for a given bucket
`bucket-compact`	Compact database and index data
`setting-compaction`	Set auto compaction settings
`setting-notification`	Set notifications.
`setting-alert`	Email alert settings
`setting-autofailover`	Set auto failover settings
`setting-xdcr`	Set XDCR-related configuration which affect behavior.
`xdcr-setup`	Set up XDCR replication.
`xdcr-replicate`	Create and run replication via XDCR
`help show longer`	usage/help and examples

couchbase-cli command options

The following are options which can be used with their respective commands. Administration — `couchbase-cli` Tool commands options:

server-list option

server-list options	Description
–group-name=GROUPNAME	Displays all server in a server group (Enterprise Edition only)

server-add options

server-add options	Description
`--server-add=HOST[:PORT]`	Server to add to cluster
`--server-add-username=USERNAME`	Admin username for the server to be added
`--server-add-password=PASSWORD`	Admin password for the server to be added
`--group-name=GROUPNAME`	Server group where the server is to be added (Enterprise Edition only)

server-readd options

server-readd options	Description
`--server-add=HOST[:PORT]`	Server to re-add to cluster
`--server-add-username=USERNAME`	Admin username for the server to be added
`--server-add-password=PASSWORD`	Admin password for the server to be added
`--group-name=GROUPNAME`	Server group where the server is to be added (Enterprise Edition only)

group-manage options (Enterprise Edition only)

group-manage options	Description
`--group-name=GROUPNAME`	Server group name
`--list`	Shows the server groups and the server assigned to each server group
`--create`	Creates a server group.
`--delete`	Removes an empty server group.
`--rename=NEWGROUPNAME`	Renames an existing server group.
`--add-servers="HOST:PORT;HOST:PORT"`	Adds servers to a group
`--move-servers="HOST:PORT;HOST:PORT"`	Moves a list of server from a group
`--from-group=GROUPNAME`	Moves one or more servers from a group.
`--to-group=GROUPNAME`	Moves one or more server to a group

rebalance options

rebalance options	Description
`--server-add*`	See server-add OPTIONS
`--server-remove=HOST[:PORT]`	The server to remove from cluster

failover option

failover option	Description
`--server-failover=HOST[:PORT]`	Server to failover

cluster-* options

cluster-* options	Description
`--cluster-username=USER`	New admin username
`--cluster-password=PASSWORD`	New admin password
`--cluster-port=PORT`	New cluster REST/http port
`--cluster-ramsize=RAMSIZEMB`	Per node RAM quota in MB

node-init options

node-init options	Description
`--node-init-data-path=PATH`	Per node path to store data
`--node-init-index-path=PATH`	Per node path to store index

bucket-* options

bucket-* options	Description
`--bucket=BUCKETNAME`	Named bucket to act on
`--bucket-type=TYPE`	Bucket type, either memcached or couchbase
`--bucket-port=PORT`	Supports ASCII protocol and does not require authentication
`--bucket-password=PASSWORD`	Standard port, exclusive with bucket-port
`--bucket-ramsize=RAMSIZEMB`	Bucket RAM quota in MB
`--bucket-replica=COUNT`	Replication count
`--enable-flush=[0|1]`	Enable/disable flush
`--enable-index-replica=[0|1]`	Enable/disable index replicas
`--wait`	Wait for bucket create to be complete before returning
`--force`	Force command execution without asking for confirmation
`--data-only`	Compact database data only
`--view-only`	Compact view data only

setting-compaction options

setting-compaction options	Description
`--compaction-db-percentage=PERCENTAGE`	Percentage of disk fragmentation when database compaction is triggered
`--compaction-db-size=SIZE[MB]`	Size of disk fragmentation when database compaction is triggered
`--compaction-view-percentage=PERCENTAGE`	Percentage of disk fragmentation when views compaction is triggered
`--compaction-view-size=SIZE[MB]`	Size of disk fragmentation when views compaction is triggered
`--compaction-period-from=HH:MM`	Enable compaction from this time onwards

setting-compaction options	Description
`--compaction-period-to=HH:MM`	Stop enabling compaction at this time
`--enable-compaction-abort=[0|1]`	Allow compaction to abort when time expires
`--enable-compaction-parallel=[0|1]`	Allow parallel compaction processes for database and view

setting-alert and notification options

setting-alert options	Description
`--enable-email-alert=[0|1]`	Allow email alert
`--email-recipients=RECIPIENT`	Email recipents, separate addresses with, or ;
`--email-sender=SENDER`	Sender email address
`--email-user=USER`	Email server username
`--email-password=PWD`	Email server password
`--email-host=HOST`	Email server hostname
`--email-port=PORT`	Email server port
`--enable-email-encrypt=[0|1]`	Email encryption with 0 the default for no encryption
`--alert-auto-failover-node`	Node was failed over via autofailover
`--alert-auto-failover-max-reached`	Maximum number of auto failover nodes reached
`--alert-auto-failover-node-down`	Node not auto failed-over as other nodes are down at the same time
`--alert-auto-failover-cluster-small`	Node not auto failed-over as cluster was too small
`--alert-ip-changed`	Node ip address changed unexpectedly
`--alert-disk-space`	Disk space used for persistent storage has reached at least 90% capacity
`--alert-meta-overhead`	Metadata overhead is more than 50% of RAM for node
`--alert-meta-oom`	Bucket memory on a node is entirely used for metadata
`--alert-write-failed`	Writing data to disk for a specific bucket has failed

setting-notification option	Description
`--enable-notification=[0|1]`	Allow notifications

setting-autofailover options

setting-autofailover options	Description
`--enable-auto-failover=[0|1]`	Allow auto failover
`--auto-failover-timeout=TIMEOUT (>=30)`	Specify amount of node timeout that triggers auto failover

setting-xdcr options

setting-xdcr options	Description
`--max-concurrent-reps=[32]`	Maximum concurrent replicators per bucket, 8 to 256.
`--checkpoint-interval=[1800]`	Intervals between checkpoints, 60 to 14400 seconds.
`--worker-batch-size=[500]`	Doc batch size, 500 to 10000.
`--doc-batch-size=[2048]KB`	Document batching size, 10 to 100000 KB
`--failure-restart-interval=[30]`	Interval for restarting failed xdcr, 1 to 300 seconds
`--optimistic-replication-threshold=[256]`	Document body size threshold (bytes) to trigger optimistic replication

xdcr-setup options

xdcr-setup options	Description
`--create`	Create a new xdcr configuration
`--edit`	Modify existed xdcr configuration
`--delete`	Delete existing xdcr configuration
`--xdcr-cluster-name=CLUSTERNAME`	Remote cluster name
`--xdcr-hostname=HOSTNAME`	Remote host name to connect to
`--xdcr-username=USERNAME`	Remote cluster admin username
`--xdcr-password=PASSWORD`	Remote cluster admin password
`--xdcr-demand-encryption=[0|1]`	Enables data encryption using Secure Socket Layer (SSL). **1** (one) enables data encryption. Default: 0 (Enterprise Edition only)
`--xdcr-certificate=CERTIFICATE`	Specifies the pem-encoded certificate. The certificate is required for XDCR data encryption. Specify the full path for the location of the pem-encoded certificate file on the source cluster. (Enterprise Edition only)

xdcr-replicate options

xdcr-replicate options	Description
`--create`	Create and start a new replication
`--delete`	Stop and cancel a replication
`--xdcr-from-bucket=BUCKET`	Source bucket name to replicate from
`--xdcr-clucter-name=CLUSTERNAME`	Remote cluster to replicate to
`--xdcr-to-bucket=BUCKETNAME`	Remote bucket to replicate to
`--xdcr-replication-mode= PROTOCOL`	Select REST protocol or memcached for replication. `xmem` indicates memcached while `capi` indicates REST protocol.

ssl-manage options

ssl-manage options	Description
`--regenerate-cert=CERTIFICATE`	Regenerates a self-signed certificate on the destination cluster. Specify the full path for the location of the pem-encoded certificate file. For example, `--regenerate-cert=./new.pem`. (Enterprise Edition only)
`--retrieve-cert=CERTIFICATE`	Retrieves the self-signed certificate from the destination cluster to the source cluster. Specify a local location (full path) and file name for the pem-encoded certificate. For example, `--retrieve-cert=./newCert.pem`. (Enterprise Edition only)

Related Links

couchbase-cli tool on page 361

Buckets CLI

Buckets are managed with the `couchbase-cli` tool and the `bucket-*` commands.

This section provides examples for listing, creating, modifying, flushing, and compacting buckets.

Listing bucketss

To list buckets in a cluster:

```
couchbase-cli bucket-list -c 192.168.0.1:8091
```

Creating buckets

To create a new dedicated port couchbase bucket:

```
couchbase-cli bucket-create -c 192.168.0.1:8091 \
       --bucket=test_bucket \
       --bucket-type=couchbase \
       --bucket-port=11222 \
       --bucket-ramsize=200 \
       --bucket-replica=1
```

To create a couchbase bucket and wait for bucket ready:

```
couchbase-cli bucket-create -c 192.168.0.1:8091 \
       --bucket=test_bucket \
       --bucket-type=couchbase \
       --bucket-port=11222 \
       --bucket-ramsize=200 \
       --bucket-replica=1 \
       --wait
```

To create a new sasl memcached bucket:

```
couchbase-cli bucket-create -c 192.168.0.1:8091 \
       --bucket=test_bucket \
       --bucket-type=memcached \
       --bucket-password=password \
       --bucket-ramsize=200 \
       --enable-flush=1 \
       --enable-index-replica=1
```

Modifying buckets

To modify a dedicated port bucket:

```
couchbase-cli bucket-edit -c 192.168.0.1:8091 \
       --bucket=test_bucket \
       --bucket-port=11222 \
       --bucket-ramsize=400 \
       --enable-flush=1
```

Deleting buckets

To delete a bucket:

```
couchbase-cli bucket-delete -c 192.168.0.1:8091 \
       --bucket=test_bucket
```

Flushing buckets

Flushing buckets involves:

1. Enable the flush buckets option.
2. Flush the bucket.

To enable the flush bucket option:

When you want to flush a data bucket you must first enable this option then actually issue the command to flush the data bucket.

Note: We do not advise that you enable this option if your data bucket is in a production environment. Be aware that this is one of the preferred methods for enabling data bucket flush.

You can enable this option when you actually create the data bucket, or when you edit the bucket properties:

```
// To enable, set bucket flush to 1. Default:0
// To enable bucket flush when creating a bucket:

couchbase-cli bucket-create [bucket_name] [cluster_admin:pass] --enable-
flush=[0|1]

// To enable bucket flush when editing the bucket properties:

couchbase-cli bucket-edit [bucket_name] [cluster_admin:pass] --enable-
flush=[0|1]
```

After you enable this option, you can then flush the data bucket.

To flush a bucket:

After you explicitly enable data bucket flush, flush the data from the bucket. Flushing a bucket is data destructive. Client applications using this are advised to double check with the end user before sending such a request. You can control and limit the ability to flush individual buckets by setting the `flushEnabled` parameter on a bucket in Couchbase Web Console or via `couchbase-cli` as described in the previous section.

Syntax

```
> couchbase-cli bucket-flush [cluster_admin:pass] [bucket_name OPTIONS]
```

By default, this command confirms whether or not you truly want to flush the data bucket. You can optionally call this command with the `--force` option to flush data without confirmation.

Example

```
couchbase-cli bucket-flush -c 192.168.0.1:8091 \
       --force
```

Compacting buckets

To compact a bucket for both data and view:

```
couchbase-cli bucket-compact -c 192.168.0.1:8091 \
        --bucket=test_bucket
```

To compact a bucket for data only:

```
couchbase-cli bucket-compact -c 192.168.0.1:8091 \
        --bucket=test_bucket \
        --data-only
```

To compact a bucket for view only:

```
couchbase-cli bucket-compact -c 192.168.0.1:8091 \
        --bucket=test_bucket \
        --view-only
```

Related Links

couchbase-cli tool on page 361

Rack-Zone Awareness CLI

Rack-Zone Awareness allows logical groupings of servers on a cluster where each server group physically belongs to a rack or availability zone.

The Rack-Zone Awareness feature provides the ability to specify that active and corresponding replica partitions be created on servers that are part of a separate rack or zone. To enable Rack-Zone Awareness, all servers in a cluster must be upgraded.

Note: The Rack Awareness feature with its server group capability is an Enterprise Edition feature.

To configure servers into groups, use the `couchbase-cli` tool with the `group-manage` command.

General syntax with `group-manage`:

```
couchbase-cli group-manage -c HOST:PORT
    -u USERNAME -p PASSWORD
    [OPTIONS]
```

`-c HOST:PORT or --cluster=HOST:PORT`	Cluster location
`-u USERNAME or --username=USERNAME`	Administrator username for the cluster
`-p PASSWORD or --password=PASSWORD`	Administrator password for the cluster
`--list`	Shows the server groups and the servers assigned to each server group.
`--create --group-name=groupName`	Creates a server group .
`--delete --group-name=groupName`	Removes an empty server group.
`--rename=newGroupName --group-name=oldGroupName`	Renames an existing server group.

--group-name=groupName --add-servers="HOST:PORT;HOST:PORT"	Adds servers to a group.
--from-group=groupName --to-group=groupName --move-servers="HOST:PORT;HOST:PORT"	Moves one or more servers from one group to another.

Creating server groups

In the following example, a server group is created.

Note: The `-create --group-name` command may fail when an exclamation (!) is present inside the group name.

```
couchbase-cli group-manage -c 192.168.0.1:8091
   -u myAdminName
   -p myAdminPassword
   --create --group-name=myGroupName
```

Adding servers to server groups

In the following example, two servers are added to a server group using the `group-manage` command.

```
couchbase-cli group-manage -c 192.168.0.1:8091
   -u myAdminName
   -p myAdminPassword
   --group-name=myNewGroup
   --add-servers="10.1.1.1:8091;10.1.1.2:8091"
```

In the following example, a server is added to the server group using the `server-add` command.

Note: The `couchbase-cli group-manage` command is the preferred method of adding servers to server group.

If the `--group-name` option is not specified with the `server-add` command, the server is added to the default group.

```
couchbase-cli server-add -c 192.168.0.1:8091
   --server-add=192.168.0.2:8091
   --server-add-username=Administrator
   --server-add-password=password
   --group-name=groupName
```

Moving servers from server groups

In the following example, two servers are moved from one server group to another using the `group-manage` command.

```
couchbase-cli group-manage -c 192.168.0.1:8091
   -u myAdminName
   -p myAdminPassword
   --from-group=myFirstGroup
   --to-group=mySecondGroup
   --move-servers="10.1.1.1:8091;10.1.1.2:8091"
```

Related Links

couchbase-cli tool on page 361

Server nodes CLI

To set a data path for an unprovisioned cluster:

```
couchbase-cli node-init -c 192.168.0.1:8091 \
       --node-init-data-path=/tmp/data \
       --node-init-index-path=/tmp/index
```

To list servers in a cluster:

```
couchbase-cli server-list -c 192.168.0.1:8091
```

Retrieving server information

```
couchbase-cli server-info -c 192.168.0.1:8091
```

Adding nodes to clusters

The following example adds a node to a cluster but does not rebalance:

```
couchbase-cli server-add -c 192.168.0.1:8091 \
    --server-add=192.168.0.2:8091 \
    --server-add-username=Administrator \
    --server-add-password=password
```

The following example adds a node to a cluster and rebalances:

```
couchbase-cli rebalance -c 192.168.0.1:8091 \
    --server-add=192.168.0.2:8091 \
    --server-add-username=Administrator \
    --server-add-password=password
```

Removing nodes

The following example removes a node from a cluster and rebalances:

```
couchbase-cli rebalance -c 192.168.0.1:8091 \
    --server-remove=192.168.0.2:8091
```

The following example removes and adds nodes from/to a cluster and rebalances:

```
couchbase-cli rebalance -c 192.168.0.1:8091 \
    --server-remove=192.168.0.2 \
    --server-add=192.168.0.4 \
    --server-add-username=Administrator \
    --server-add-password=password
```

Stopping rebalance

The following example stops the current rebalancing:

```
couchbase-cli rebalance-stop -c 192.168.0.1:8091
```

Setting cluster parameters

The following example sets the username, password, port and ram quota:

```
couchbase-cli cluster-init -c 192.168.0.1:8091 \
    --cluster-init-username=Administrator \
    --cluster-init-password=password \
    --cluster-init-port=8080 \
    --cluster-init-ramsize=300
```

The following example changes the cluster username, password, port and ram quota:

```
    couchbase-cli cluster-edit -c 192.168.0.1:8091 \
       --cluster-username=Administrator \
       --cluster-password=password \
       --cluster-port=8080 \
       --cluster-ramsize=300
```

Changing data paths

The following example changes the data path:

```
     couchbase-cli node-init -c 192.168.0.1:8091 \
       --node-init-data-path=/tmp
```

Related Links

couchbase-cli tool on page 361

XDCR CLI

To create a XDCR remote cluster:

```
    couchbase-cli xdcr-setup -c 192.168.0.1:8091 \
        --create \
        --xdcr-cluster-name=test \
        --xdcr-hostname=10.1.2.3:8091 \
        --xdcr-username=Administrator \
        --xdcr-password=password
```

To delete a XDCR remote cluster:

```
    couchbase-cli xdcr-delete -c 192.168.0.1:8091 \
        --xdcr-cluster-name=test
```

Managing XDCR replication streams

To start a replication stream:

```
    couchbase-cli xdcr-replicate -c 192.168.0.1:8091 \
        --create \
        --xdcr-cluster-name=test \
        --xdcr-from-bucket=default \
        --xdcr-to-bucket=default1
```

To delete a replication stream:

```
    couchbase-cli xdcr-replicate -c 192.168.0.1:8091 \
        --delete \
        --xdcr-replicator=f4eb540d74c43fd3ac6d4b7910c8c92f/default/default
```

Managing remote clusters

To create a remote cluster reference:

In the following example the remote cluster is “RemoteCluster”.

```
couchbase-cli xdcr-setup -c 10.3.121.121:8091 -u Administrator  -p password \
    --create
    --xdcr-cluster-name=RemoteCluster
    --xdcr-hostname=10.3.121.123:8091
    --xdcr-username=Administrator
    --xdcr-password=password
```

To set a XDCR protocol:

An XDCR protocol for the mode of replication can be specified for XDCR.

To change a XDCR replication protocol for an existing XDCR replication:

If you change want the replication protocol for an existing XDCR replication:

1. Delete the replication.
2. Re-create the replication with your preference.

First we create a destination cluster reference named "RemoteCluster":

```
couchbase-cli xdcr-setup -c hostname_:8091 -u Administrator  -p password \
  --create --xdcr-cluster-name=RemoteCluster  --xdcr-
hostname=10.3.121.123:8091 \
  --xdcr-username=Administrator  --xdcr-password=password
```

Upon success, we get this response:

```
    SUCCESS: init RemoteCluster
```

Now you can start replication to the remote cluster using memcached protocol as the existing default:

```
couchbase-cli xdcr-replicate -c host_name:8091 -u Administrator -p password
        --xdcr-cluster-name RemoteCluster
        --xdcr-from-bucket default
        --xdcr-to-bucket backup
```

To explicitly set the protocol to memcached:

```
couchbase-cli xdcr-replicate -c host_name:8091 -u Administrator -p password \
    --xdcr-cluster-name RemoteCluster
    --xdcr-from-bucket default
    --xdcr-to-bucket backup
    --xdcr-replication-mode xmem
```

To set the protocol to CAPI:

```
couchbase-cli xdcr-replicate -c host_name:8091 -u Administrator -p password \
    --xdcr-cluster-name RemoteCluster
    --xdcr-from-bucket default
    --xdcr-to-bucket backup
    --xdcr-replication-mode capi
```

If there is already an existing replication for a bucket, you get an error when you try to start the replication again with any new settings:

```
 couchbase-cli xdcr-replicate -c 10.3.121.121:8091 -u Administrator -p
 password
    --xdcr-cluster-name RemoteCluster
    --xdcr-from-bucket default
    --xdcr-to-bucket backup
    --xdcr-replication-mode capi
```

Results in the following error:

```
ERROR: unable to create replication (400) Bad Request
        {u'errors': {u'_': u'Replication to the same remote cluster and bucket
 already exists'}}
ERROR: Replication to the same remote cluster and bucket already exists
```

Managing XDCR data encryption

The Couchbase Server command line interface (CLI) enables XDCR data encryption (Enterprise Edition only) when an XDCR cluster reference is created or modified. The CLI provides the `couchbase-cli` tool and the `xdcr-setup` command. The option `--xdcr-demand-encryption=1` enables XDCR data encryption `-xdcr-certificate=CERTIFICATE` provides the SSL certificate for data security.

To setup XDCR with SSL data encryption:

1. Retrieve the certificate from the destination cluster.
2. Create or modify the XDCR configuration to allow data encryption and provide the SSL certificate.
3. Define the replication.

To configure XDCR with SSL data encryption, the `xdcr-setup` command is used.

Syntax

```
couchbase-cli xdcr-setup -c localHost:port -u localAdmin -p localPassword
  --create --xdcr-cluster-name=remoteClustername
  --xdcr-hostname=remoteHost:port
  --xdcr-username=remoteAdmin --xdcr-password=remotePassword
  --xdcr-demand-encryption=[0|1]   // 1 to enable, 0 to disable (default)
  --xdcr-certificate=<localPath>/<certFile>.pem
```

Example

```
couchbase-cli xdcr-setup -c 10.3.4.186:8091 -u localAdmin -p localPassword
  --create --xdcr-cluster-name=Omaha
  --xdcr-hostname=10.3.4.187:8091
  --xdcr-username=Peyton --xdcr-password=Manning
  --xdcr-demand-encryption=1
  --xdcr-certificate=./new.pem
```

Results

The following is an example of results for a successful XDCR configuration.

```
SUCCESS: init/edit test
<<replication reference created>>
```

To disable XDCR data encryption, execute `couchbase-cli xdcr-setup` with `--xdcr-demand-encryption=0`.

Example

```
couchbase-cli xdcr-setup -c 10.3.4.186:8091 -u localAdmin -p localPassword
  --create --xdcr-cluster-name=Omaha
  --xdcr-hostname=10.3.4.187:8091
  --xdcr-username=Peyton --xdcr-password=Manning
  --xdcr-demand-encryption=0
```

Managing SSL certificates

Retrieving an SSL certificate for XDCR data encryption, should be done in a secure manner, such as with `ssh` and `scp`. For example:

1. Use a secure method to log in to a node on the destination cluster. For example: `ssh`.
2. Retrieve the certificate with the `couchbase-cli ssl-manage` command.
3. Use a secure method to transfer the certificate from the destination cluster to the source cluster. For example: `scp`.
4. Proceed with setting up XDCR with SSL data encryption.

The `couchbase-cli ssl-manage` command provides the following options for regenerating and retrieving certificates.

`--regenerate-cert=CERTIFICATE`	Regenerates a self-signed certificate on the destination cluster. Specify the full path for the location of the pem-encoded certificate file. For example, `--regenerate-cert=./new.pem`.
`--retrieve-cert=CERTIFICATE`	Retrieves the self-signed certificate from the destination cluster to the source cluster. Specify a local location (full path) and file name for the pem-encoded certificate. For example, `--retrieve-cert=./newCert.pem`.

To retrieve an existing self-signed certificate, use the `ssl-manage` command.

Syntax

```
couchbase-cli ssl-manage -c localHost:port
```

```
  -u Administrator -p password
  --retrieve-cert=./<newCert>.pem
```

Example

```
couchbase-cli ssl-manage -c 10.3.4.187:8091
  -u Administrator -p password
  --retrieve-cert=./newCert.pem
```

Results

The following is an example of results for a successful retrieval of the certificate:

```
SUCCESS: retrieve certificate to './newCert.pem'
Certificate matches what seen on GUI
```

To regenerate a self-signed certificate, use the `couchbase-cli ssl-manage` command.

Syntax

```
couchbase-cli ssl-manage
  -c remoteHost:port
  -u adminName -p adminPassword
  --regenerate-cert=CERTIFICATE
```

Example

The following is an example of the CLI commands and options for regenerating a self-signed certificate with the `ssl-manage` command:

```
couchbase-cli ssl-manage
  -c 10.3.4.187:8091
  -u Administrator -p password
  --regenerate-cert=./new.pem
```

Results

The following is an example of results for a successful regeneration of the certification:

```
SUCCESS: regenerate certificate to './new.pem'
```

To retrieve an existing self-signed certificate, the `ssl-manage` command is used.

Syntax

```
couchbase-cli ssl-manage -c localHost:port
  -u Administrator -p password
  --retrieve-cert=./<newCert>.pem
```

Example

```
couchbase-cli ssl-manage -c 10.3.4.187:8091
  -u Administrator -p password
  --retrieve-cert=./newCert.pem
```

Results

The following is an example of results for a successful retrieval of the certificate:

```
SUCCESS: retrieve certificate to './newCert.pem'
Certificate matches what seen on GUI
```

Related Links

couchbase-cli tool on page 361

Related topics

Cross Datacenter Replication (XDCR) on page 241

Managing XDCR data encryption on page 289

Managing XDCR on page 169

Cross datacenter replication (XDCR) provides an easy method of replicating data from one cluster to another for disaster recovery as well as better data locality (getting data closer to its users).

cbcollect_info tool

This is one of the most important diagnostic tools used by Couchbase technical support teams; this command-line tool provides detailed statistics for a specific node. The tool is at the following locations, depending upon your platform:

Operating System	Location
Linux	`/opt/couchbase/bin/cbcollect_info`
Windows	`C:\Program Files\Couchbase\Server\bin\cbcollect_info`
Mac OS X	`/Applications/Couchbase Server.app/Contents/Resources/couchbase-core/bin/cbcollect_info`

Be aware that this tool is a per-node operation. If you want to perform this operation for an entire cluster, you will need to perform the command for every node that exists for that cluster.

You will need a root account to run this command and collect all the server information needed. There are internal server files and directories that this tool accesses which require root privileges.

To use this command, you remotely connect to the machine which contains your Couchbase Server then issue the command with options. You typically run this command under the direction of technical support at Couchbase and it will generate a large.zip file. This archive will contain several different files which contain performance statistics and extracts from server logs. The following describes usage, where `output_file` is the name of the.zip file you will create and send to Couchbase technical support:

```
cbcollect_info hostname:port output_file

Options:
  -h, --help  show this help message and exit
  -v          increase verbosity level
```

If you choose the verbosity option, `-v` debugging information for `cbcollect_info` will be also output to your console. When you run `cbcollect_info`, it will gather statistics from an individual node in the cluster.

This command will collect information from an individual Couchbase Server node. If you are experiencing problems with multiple nodes in a cluster, you may need to run it on all nodes in a cluster.

The tool will create the following.log files in your named archive:

couchbase.log	OS-level information about a node.
ns_server.couchdb.log	Information about the persistence layer for a node.
ns_server.debug.log	Debug-level information for the cluster management component of this node.
ns_server.error.log	Error-level information for the cluster management component of this node.
ns_server.info.log	Info-level entries for the cluster management component of this node.
ns_server.views.log	Includes information about indexing, time taken for indexing, queries which have been run, and other statistics about views.
stats.log	The results from multiple `cbstats` options run for the node.

After you finish running the tool, you should upload the archive and send it to Couchbase technical support:

```
> curl --upload-file file_name https://s3.amazonaws.com/
customers.couchbase.com/company_name/
```

Where `file_name` is the name of your archive, and `company_name` is the name of your organization. After you have uploaded the archive, please contact Couchbase technical support.

Related Links

Couchbase CLI on page 352

Related topics

cbstats tool on page 392

Working with Couchbase Customer Support

cbdocloader tool

You can use this tool to load a group of JSON documents in a given directory, or in a single.zip file. This is the underlying tool used during your initial Couchbase Server install which will optionally install two sample databases provided by Couchbase. You can find this tool in the following locations, depending upon your platform:

Operating System	Location
Linux	`/opt/couchbase/bin/tools/`
Windows	`C:\Program Files\Couchbase\Server\bin\tools\`
Mac OS X	`/Applications/Couchbase Server.app/Contents/Resources/couchbase-core/bin/tools/`

When you load documents as well as any associated design documents for views, you should use a directory structure similar to the following:

```
/design_docs    // which contains all the design docs for views.
/docs           // which contains all the raw json data files. This can
 contain other sub directories too.
```

All JSON files that you want to upload contain well-formatted JSON. Any file names should exclude spaces. If you want to upload JSON documents and design documents into Couchbase Server, be aware that the design documents will be uploaded after all JSON documents. The following are command options for `cbdocloader` :

```
-n HOST[:PORT], --node=HOST[:PORT] Default port is 8091

-u USERNAME, --user=USERNAME REST username of the cluster. It can be specified
 in environment variable REST_USERNAME.

-p PASSWORD, --password=PASSWORD REST password of the cluster. It can be
 specified in environment variable REST_PASSWORD.

-b BUCKETNAME, --bucket=BUCKETNAME Specific bucket name. Default is default
 bucket. Bucket will be created if it does not exist.

-s QUOTA, RAM quota for the bucket. Unit is MB. Default is 100MB.

-h --help Show this help message and exit
```

The following is an example of uploading JSON from a.zip file:

```
./cbdocloader  -n localhost:8091 -u Administrator -p password -b mybucket ../
samples/gamesim.zip
```

Be aware that there are typically three types of errors that can occur: 1) the files are not well-formatted, 2) credentials are incorrect, or 3) the RAM quota for a new bucket to contain the JSON is too large given the current quota for Couchbase Server.

Related Links

Couchbase CLI on page 352

cbepctl tool

The `cbepctl` command enables you to control many of the configuration, RAM and disk parameters of a running cluster. This tool is for controlling the vBucket states on a Couchbase Server node. It is also responsible for controlling the configuration, memory and disk persistence behavior. This tool was formerly provided as the separate tools, `cbvbucketctl` and `cbflushctl` in Couchbase 1.8.

Changes to the cluster configuration using `cbepctl` are not persisted over a cluster restart.

Operating system	Location
Linux	`/opt/couchbase/bin/cbepctl`
Windows	`C:\Program Files\Couchbase\Server\bin\cbepctl.exe`
Mac OS X	`/Applications/Couchbase Server.app/Contents/Resources/couchbase-core/bin/cbepctl`

Be aware that this tool is a per-node, per-bucket operation. That means that if you want to perform this operation, you must specify the IP address of a node in the cluster and a named bucket. If you do not provided a named bucket, the server will apply the setting to any default bucket that exists at the specified node. If you want to perform this operation for an entire cluster, you will need to perform the command for every node/bucket combination that exists for that cluster.

```
cbepctl host:11210 -b bucket_name -p bucket_password start
cbepctl host:11210 -b bucket_name -p bucket_password stop
cbepctl host:11210 -b bucket_name -p bucket_password set type param value
```

For this command, `host` is the IP address for your Couchbase cluster, or node in the cluster. The port will always be the standard port used for cluster-wide stats and is at `11210`. You also provide the named bucket and the password for the named bucket. After this you provide command options and authentication.

You can use the following command options to manage persistence:

Option	Description
stop	Stop persistence
start	Start persistence
drain	Wait until queues are drained
set	To set `checkpoint_param`, `flush_param`, and `tap_param`. This changes how or when persistence occurs.

You can use the following command options, combined with the parameters to set `checkpoint_param`, `flush_param`, and `tap_param`. These changes the behavior of persistence in Couchbase Server.

cbepctl set checkpoint_param

The command options for `checkpoint_param` are:

Parameter	Description
chk_max_items	Max number of items allowed in a checkpoint.
chk_period	Time bound (in sec.) on a checkpoint.
item_num_based_new_chk	True if a new checkpoint can be created based on. the number of items in the open checkpoint.

Parameter	Description
keep_closed_chks	True if we want to keep closed checkpoints in memory, as long as the current memory usage is below high water mark.
max_checkpoints	Max number of checkpoints allowed per vbucket.

cbepctl set flush_param

The complete list of options for `flush_param` are:

Parameter	Description
alog_sleep_time	Access scanner interval (minute)
alog_task_time	Access scanner next task time (UTC)
bg_fetch_delay	Delay before executing a bg fetch (test feature).
couch_response_timeout	timeout in receiving a response from CouchDB.
exp_pager_stime	Expiry Pager interval. Time interval that Couchbase Server waits before it performs cleanup and removal of expired items from disk.
pager_active_vb_pcnt	Percentage of active vBuckets items among all ejected items by item pager.
max_size	Maximum memory used by the server.
mem_high_wat	Low water mark.
mem_low_wat	High water mark.
mutation_mem_threshold	Amount of RAM that can be consumed in that caching layer before clients start receiving temporary out of memory messages.
timing_log	Path to log detailed timing stats.
warmup_min_memory_threshold	Memory threshold (%) during warmup to enable traffic.
warmup_min_item_threshold	Item number threshold (%) during warmup to enable traffic.
klog_compactor_queue_cap	queue cap to throttle the log compactor.
klog_max_log_size	maximum size of a mutation log file allowed.
klog_max_entry_ratio	max ratio of # of items logged to # of unique items.
pager_unbiased_period	Period after last access scanner run during which item pager preserve working set.
queue_age_cap	Maximum queue age before flushing data.
max_txn_size	Maximum number of items in a flusher transaction.
min_data_age	Minimum data age before flushing data.

Note: The `cbepctl` tool is a per-node, per-bucket operation. That means that if you want to perform this operation, you must specify the IP address of a node in the cluster and a named bucket. If you do not provided a named bucket, the server will apply the setting to any default bucket that exists at the specified node. If you want to perform this operation for an entire cluster, you will need to perform the command for every node/bucket combination that exists for that cluster.

cbepctl set tap_param

Parameter	Description
tap_keepalive	Seconds to hold a named tap connection.
tap_throttle_queue_cap	Maximum disk write queue size to throttle tap streams (infinite means no cap).
tap_throttle_cap_pcnt	Percentage of total items in write queue at which we throttle tap input.
tap_throttle_threshold	Percentage of memory in use to throttle tap streams

Related Links

Couchbase CLI on page 352
Changing thresholds for ejection on page 381
Changing access log settings on page 382
Changing disk cleanup interval on page 383
Changing disk write queue quotas on page 383
Changing setting for out of memory errors on page 384

Related topics

Buckets CLI on page 367
Buckets are managed with the `couchbase-cli` tool and the `bucket-*` commands.

Flushing buckets on page 471
To flush all buckets, use the `POST` operation with the `/pools/default/buckets/default/controller/doFlush` URI.

Data Buckets on page 259

Changing thresholds for ejection

The item pager process ejects items from RAM when too much space is being taken up in RAM. Ejection means that documents are removed from RAM but the key and metadata remain.

If the amount of RAM used by items reaches the high water mark (upper threshold), both active and replica data are ejected until the memory usage (amount of RAM consumed) reaches the low water mark (lower threshold).

The server determines that items are not frequently used based on a not-recently-used (NRU) boolean. There a few settings you can adjust to change server behavior during the ejection process. In general, we do not recommend you change ejection defaults for Couchbase Server unless you are required to do so.

Be aware that this tool is a per-node, per-bucket operation. To perform this operation, the IP address of a node in the cluster and a named bucket must be specified. If a named bucket is not provided, the server applies the setting to any default bucket that exists at the specified node. To perform this operation for an entire cluster, perform the command for every node/bucket combination that exists for that cluster.

For technical information about the ejection process, the role of NRU and server processes related to ejection.

Setting the low water mark

This represents the lower threshold of RAM to be consumed on a node. The item pager stops ejecting items once the low water mark is reached. To change this percentage amount of RAM, use the `cpepctl` tool:

```
>    ./cbepctl 10.5.2.31:11210 -b bucket_name -p bucket_password set
 flush_param mem_low_wat 70
```

Setting the high water mark

This represents the amount of RAM consumed by items that must be breached before infrequently used active and replica items are ejected. To change this amount, use the `cbepctl` tool. In the following example, the high water mark is set to 80% of RAM for a specific data bucket on a given node.

This means that items in RAM on this node can consume up to 80% of RAM before the item pager begins ejecting items.

```
>    ./cbepctl 10.5.2.31:11210 -b bucket_name -p bucket_password set
 flush_param mem_high_wat 80
```

Setting percentage of ejected items

Based on the NRU algorithm, the server ejects active and replica data from a node. By default, the server is configured to 40% active items and 60% replica data from a node.

To change default percentage for ejecting active and replica items, use the `cbepctl` tool. The following example increases the percentage of active items that can be ejected from a node to 50%.

```
>    ./cbepctl 10.5.2.31:11210 -b bucket_name -p bucket_password set
 flush_param pager_active_vb_pcnt 50
```

Be aware of potential performance implications when making this change. It may seem more desirable to eject as many replica items as possible and limit the amount of active data that can be ejected. In doing so, you will be able to maintain as much active data from a source node as possible, and maintain incoming requests to that node. However, if you have the server eject a very large percentage of replica data, should a node fail, the replica data is not immediately available. In that case, the items are retrieved from disk and put back into RAM. Once in RAM, the request can be fulfilled. Couchbase recommends that you do not change these defaults.

For technical information about the ejection process, the role of NRU and server processes related to ejection.

Related Links

cbepctl tool on page 379

Related topics

Ejection and working set management on page 107

Working set management refers to the ejection of data from RAM. Couchbase Server ejects data to create free space and to maintain system performance.

Changing access log settings

Couchbase Server has an optimized disk warmup. Couchbase Server pre-fetches a list of most-frequently accessed keys and fetches these documents first. The server runs a periodic scanner process which determines which keys are most frequently-used. The `cbepctl flush_param` command is used to change the initial time and the interval for the process. You may want to do this, for example, if you have a peak time for your application when you want the keys used during this time to be quickly available after server restart.

Note: If you want to change this setting for an entire Couchbase cluster, you will need to perform this command on per-node and per-bucket in the cluster. By default, any setting you change with `cbepctl` will only be for the named bucket at the specific node you provide in the command. This means that if you have a data bucket that is shared by two nodes, you will nonetheless need to issue this command twice and provide the different host names and ports for each node and the bucket name. Similarly, if you have two data buckets for one node, you need to issue the command twice and provide the two data bucket names. If you do not specify a named bucket, it will apply to the default bucket or return an error if a default bucket does not exist.

By default the scanner process runs once every 24 hours with a default initial start time of 2:00 AM UTC. This means after you install a new Couchbase Server 2.0 instance or restart the server, by default the scanner will run every 24-hour time period at 2:00 AM GMT and then 2:00 PM GMT by default. To change the time interval when the access scanner process runs to every 20 minutes:

```
> ./cbepctl hostname:port -b bucket_name -p bucket_password set flush_param
 alog_sleep_time 20
```

To change the initial time that the access scanner process runs from the default of 2:00 AM UTC:

```
> ./cbepctl hostname:port -b bucket_name -p bucket_password set flush_param
 alog_task_time 23
```

In this example we set the initial time to 11:00 PM UTC.

Important: Be aware that this tool is a per-node, per-bucket operation. This means that in order to perform this operation, you must specify the IP address of a node in the cluster and a named bucket. If you do not provided a named bucket, the server applies the setting to any default bucket that exists at the specified node. In order to perform this operation for an entire cluster, perform the command for every node/bucket combination that exists for that cluster.

Related Links

cbepctl tool on page 379

Changing disk cleanup interval

One of the most important use cases for the `cbepctl flush_param` is the set the time interval for disk cleanup. Couchbase Server does lazy expiration, that is, expired items are flagged as deleted rather than being immediately erased. Couchbase Server has a maintenance process that will periodically look through all information and erase expired items. This maintenance process will run every 60 minutes, but it can be configured to run at a different interval. For example, the following options will set the cleanup process to run every 10 minutes:

```
./cbepctl localhost:11210 -b bucket_name -p bucket_password set flush_param
 exp_pager_stime 600
```

Be aware that this tool is a per-node, per-bucket operation. That means that if you want to perform this operation, you must specify the IP address of a node in the cluster and a named bucket. If you do not provided a named bucket, the server will apply the setting to any default bucket that exists at the specified node. If you want to perform this operation for an entire cluster, you will need to perform the command for every node/bucket combination that exists for that cluster.

Here we specify 600 seconds, or 10 minutes as the interval Couchbase Server waits before it tries to remove expired items from disk.

Related Links

cbepctl tool on page 379

Changing disk write queue quotas

One of the specific uses of `cbepctl` is to the change the default maximum items for a disk write queue. This impacts replication of data that occurs between source and destination nodes within a cluster. Both data that a node receives from client applications, and replicated items that it receives are placed on a disk write queue. If there are too many items waiting in the disk write queue at any given destination, Couchbase Server will reduce the rate of data that is sent to a destination. This is process is also known as *backoff*.

By default, when a disk write queue contains one million items, a Couchbase node will reduce the rate it sends out data to be replicated. You can change this setting to be the greater of 10% of the items at a destination node or a number you specify. For instance:

```
> ./cbepctl 10.5.2.31:11210 -b bucket_name -p bucket_password set tap_param
 tap_throttle_queue_cap 2000000
```

Be aware that this tool is a per-node, per-bucket operation. That means that if you want to perform this operation, you must specify the IP address of a node in the cluster and a named bucket. If you do not provided a named bucket, the server will apply the setting to any default bucket that exists at the specified node. If you want to perform this operation for an entire cluster, you will need to perform the command for every node/bucket combination that exists for that cluster.

In this example we specify that a replica node send a request to backoff when it has two million items or 10% of all items, whichever is greater. You will see a response similar to the following:

```
setting param: tap_throttle_queue_cap 2000000
```

In this next example, we change the default percentage used to manage the replication stream. If the items in a disk write queue reach the greater of this percentage or a specified number of items, replication requests will slow down:

```
> ./cbepctl 10.5.2.31:11210 -b bucket_name -p bucket_password set tap_param
 tap_throttle_cap_pcnt 15
```

In this example, we set the threshold to 15% of all items at a replica node. When a disk write queue on a replica node reaches this point, it will request replication backoff. For more information about replicas, replication and backoff from replication. The other command options for `tap_param` are:

Parameter	Description
tap_keepalive	Seconds to hold a named tap connection.
tap_throttle_queue_cap	Max disk write queue size when tap streams will put into a temporary, 5-second pause. 'Infinite' means there is no cap.
tap_throttle_cap_pcnt	Maximum items in disk write queue as percentage of all items on a node. At this point tap streams will put into a temporary, 5-second pause.
tap_throttle_threshold	Percentage of memory in use when tap streams will be put into a temporary, 5-second pause.

Related Links

cbepctl tool on page 379

Related topics

Replicas and replication on page 111

Replicas are copies of data that are proved on another node in a cluster.

Changing setting for out of memory errors

By default, Couchbase Server will send clients a temporary out of memory error if RAM is 95% consumed and only 5% RAM remains for overhead. We do not suggest you change this default to a higher value; however you may choose to reduce this value if you think you need more RAM available for system overhead such as disk queue or for server data structures. To change this value:

```
>./cbepctl 10.5.2.31:11210 -b bucket_name -p bucket_password set flush_param
 mutation_mem_threshold 65
```

Be aware that this tool is a per-node, per-bucket operation. That means that if you want to perform this operation, you must specify the IP address of a node in the cluster and a named bucket. If you do not provided a named bucket, the server will apply the setting to any default bucket that exists at the specified node. If you want to perform this operation for an entire cluster, you will need to perform the command for every node/bucket combination that exists for that cluster.

In this example we reduce the threshold to 65% of RAM. This setting must be updated on a per-node, per-bucket basis, meaning you need to provide the specific node and named bucket to update this setting. To update it for an entire cluster, you will need to issue the command for every combination of node and named bucket that exists in the cluster.

Related Links

cbepctl tool on page 379

cbhealthchecker tool

The `cbhealthchecker` tool generates a health report named *Cluster Health Check Report* for a Couchbase cluster. The report provides data that helps administrators, developers, and testers determine whether a cluster is healthy, has issues that must be addressed soon to prevent future problems, or has issues that must be addressed immediately.

The tool retrieves data from the Couchbase Server monitoring system, aggregates it over a time scale, analyzes the statistics against thresholds, and generates a report. Unlike other command line tools such as `cbstats` and `cbtransfer` that use the TAP protocol to obtain data from the monitoring system, `cbhealthchecker` obtains data by using the REST API and the memcached protocol. For more information about the statistics provided by Couchbase Server.

You can generate reports on the following time scales: minute, hour, day, week, month, and year. The tool outputs an HTML file, a text file, and a JSON file. Each file contains the same information — the only difference between them is the format of the information. All `cbhealthchecker` output is stored in a `reports` folder. The tool does not delete any files from the folder. You can delete files manually if the `reports` folder becomes too large. The path to the output files is displayed when the run finishes.

`cbhealthchecker` is automatically installed with Couchbase Server. You can find the tool in the following locations, depending upon your platform:

Operating System	Location
Linux	`/opt/couchbase/bin/`
Windows	`C:\Program Files\Couchbase\Server\bin\`
Mac OS X	`/Applications/Couchbase Server.app/Contents/Resources/couchbase-core/bin/`

The format of the `cbhealthchecker` command is:

```
cbhealthchecker CLUSTER USERNAME PASSWORD OPTIONS
```

Option	Syntax	Description
CLUSTER	`-c HOST[:PORT] --cluster=HOST[:PORT]`	Hostname and port of a node in the cluster. The default port is 8091.
USERNAME	`-u USERNAME --user=USERNAME`	Admin username of the cluster.
PASSWORD	`-p PASSWORD --password=PASSWORD`	Admin password of the cluster.
OPTIONS	`-b BUCKETNAME --bucket=BUCKETNAME`	Specific bucket on which to report. The default is all buckets.
	`-i FILENAME --input=FILENAME`	Generate an analysis report from an input JSON file.
	`-o FILENAME --output=FILENAME`	File name for the HTML report. The default output file name is the report time stamp, for example: `2013-07-26_13-26-23.html`.
	`-h --help`	Show the help message and exit.
	`-s SCALE --scale=SCALE`	Time span (scale) for the statistics: minute, hour, day, week, month or year. The default time span is day.

Option	Syntax	Description
	`-j --jsononly`	Collect data and output only a JSON file. When you use this option, the analysis report is not generated.

Sample Commands

The following command runs a report on all buckets in the cluster for the past day:

```
./cbhealthchecker -c 10.3.1.10:8091 -u Administrator -p password

bucket: default
  node: 10.3.1.10 11210
  node: 10.3.1.11 11210
................................

The run finished successfully.
Please find html output at '/opt/couchbase/bin/
reports/2013-07-23_16-29-02.html'
and text output at '/opt/couchbase/bin/reports/2013-07-23_16-29-02.txt'.
```

The following command runs a report on all buckets in the cluster for the past month:

```
./cbhealthchecker -c 10.3.1.10:8091 -u Administrator -p password -s month

The run finished successfully.
Please find html output at '/opt/couchbase/bin/
reports/2013-07-26_13-26-23.html'
and text output at '/opt/couchbase/bin/reports/2013-07-26_13-26-23.txt'.
```

The following command runs a report on only the `beer-sample` bucket for the past year and outputs the HTML report to a file named `beer-health-report.html`.

```
./cbhealthchecker -c 10.3.1.10:8091 -u Administrator -p password -o beer-
health-report.html \
      -b beer-sample -s year

The run finished successfully.
Please find html output at '/opt/couchbase/bin/reports/beer-health-
report.html'
and text output at '/opt/couchbase/bin/reports/2013-07-26_15-57-11.txt'.
```

The following command generates only the statistics and outputs them in a JSON file:

```
./cbhealthchecker -c 10.3.1.10:8091 -u Administrator -p password -j

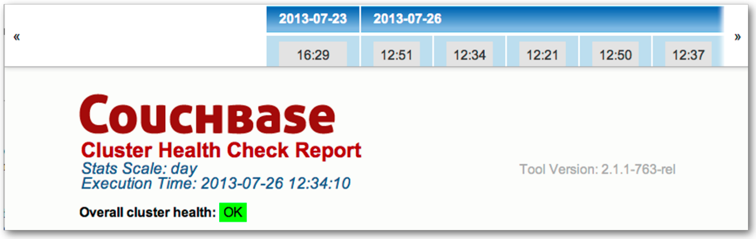
The run finished successfully.
Please find collected stats at '/opt/couchbase/bin/
reports/2013-07-26_13-30-36.json'.
```

HTML Report

You can view the HTML report in any web browser. If you copy the report to another location, be sure to copy all the files in the reports folder to ensure that the report is displayed correctly by the browser. When you have multiple HTML reports in the folder, you can use the tabs at the top of the page to display a particular report. (If the tabs do not function in your browser, try using Firefox.)

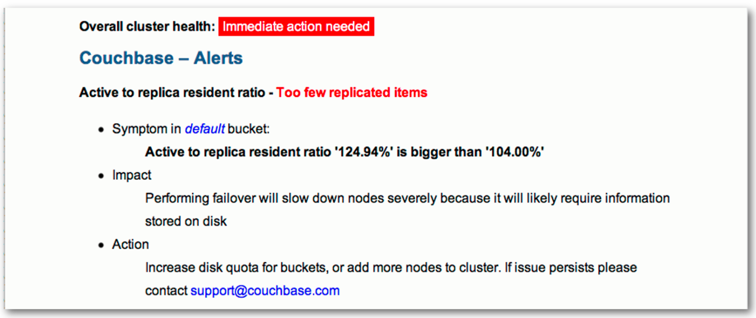
Throughout the report, normal health statuses are highlighted in green, warnings are highlighted in yellow, and conditions that require immediate action are highlighted in red. When viewing the report, you can hover your mouse over each statistic to display a message that describes how the statistic is calculated.

The report begins with a header that lists the statistics scale, the date and time the report was run, and an assessment of the overall health of the cluster. The following figure shows the report header:



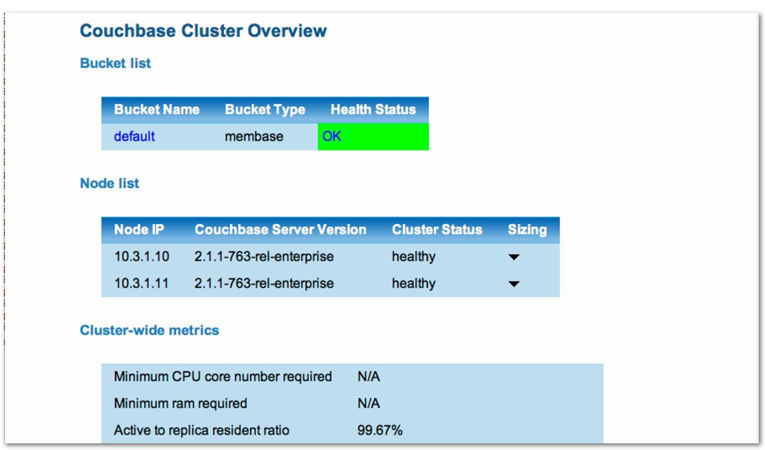
- The body of the report is divided into several sections:Couchbase — Alerts

The alerts section contains a list of urgent issues that require immediate attention. For each issue, the report lists the symptoms detected, the impact of the issue, and the recommended corrective action to take. This section appears in the report only when urgent issues are detected. The following figure shows a portion of the alerts section of a report:



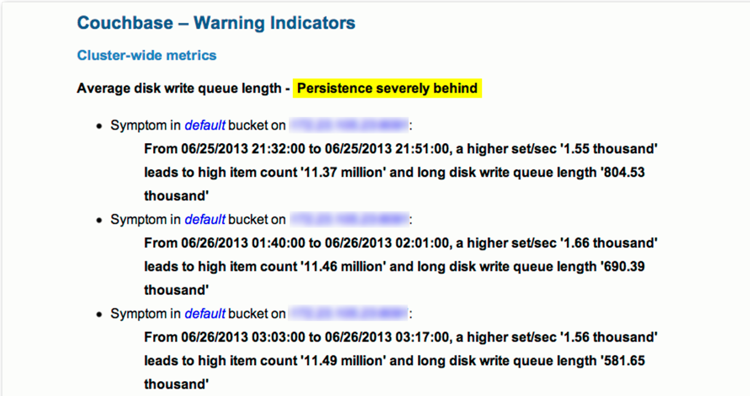
- Couchbase Cluster Overview

The cluster overview section contains cluster-wide metrics and metrics for each bucket and node in the cluster. This section appears in all reports. The following figure shows a portion of the cluster overview section of a report:



- Couchbase — Warning Indicators

The warning indicators section contains a list of issues that require attention. For each issue, the report lists the symptoms detected, the impact of the issue, and the recommended corrective action to take. This section appears in the report only when warning indicators are detected. The following figure shows a portion of the warning indicators section of a report:



Related Links

Couchbase CLI on page 352

Related topics

Statistics and monitoring on page 117

A complete set of statical and monitoring information are provided through the Web Console, CLI, and REST API.

TAP on page 116

The TAP protocol is an internal part of the Couchbase Server system and is used to exchange data throughout the system.

cbreset_password tool

You use this tool to reset an administrative or read-only password. You can find this tool in the following locations, depending upon your platform:

Operating System	Location
Linux	`/opt/couchbase/bin/tools/`
Windows	`C:\Program Files\Couchbase\Server\bin\tools\`
Mac OS X	`/Applications/Couchbase Server.app/Contents/Resources/couchbase-core/bin/tools/`

To reset the administrative password:

```
./cbreset_password hostname:port
```

This will result in output as follows:

```
Please enter the new administrative password (or <Enter> for system generated
 password):
```

Enter a password of six characters or more or you can have the system generate one for you. After you enter a password or accept a generated one, the system will prompt you for confirmation:

```
Running this command will reset administrative password.
Do you really want to do it? (yes/no)yes
```

Upon success you will see this output:

```
Resetting administrative password...
Password for user Administrator was successfully replaced. New password is
 Uxye76FJ
```

There are a few possible errors from this command:

```
{error,<<"The password must be at least six characters.">>}

{error,<<"Failed to reset administrative password. Node is not
 initialized.">>}
```

The first one indicates you have not provided a password of adequate length. The second one indicates that Couchbase Server is not yet configured and running.

Related Links

Couchbase CLI on page 352

cbrestore tool

The `cbrestore` tool restores data from a file to an entire cluster or to a single bucket in the cluster. Items that had been written to file on disk will be restored to RAM.

`cbbackup`, `cbrestore` and `cbtransfer` do not communicate with external IP addresses for server nodes outside of a cluster. They can only communicate with nodes from a node list obtained within a cluster. You should perform backup, restore, or transfer to data from a node within a Couchbase cluster. This also means that if you install Couchbase Server with the default IP address, you cannot use an external hostname to access it.

The tool is in the following locations, depending on your platform:

Linux	**`/opt/couchbase/bin/cbrestore`**
Windows	`C:\Program Files\Couchbase\Server\bin\cbrestore`
Mac OS X	`/Applications/Couchbase Server.app/Contents/Resources/couchbase-core/bin/cbrestore`

The format of the `cbrestore` command is:

```
cbrestore [options] [host:ip] [source] [destination]
```

Where:

- `[options]`

Command options for `cbrestore` are the same options for `cbtransfer`.

- `[host:ip]`

Hostname and port for a node in cluster.

- `[source]`

Source bucket name for the backup data. This is in the directory created by `cbbackup` when you performed the backup.

- `[destination]`

The destination bucket for the restored information. This is a bucket in an existing cluster. If you restore the data to a single node in a cluster, provide the hostname and port for the node you want to restore to. If you restore an entire data bucket, provide the URL of one of the nodes within the cluster.

All command options for `cbrestore` are the same options available for `cbtransfer`.

Using cbrestore for design documents

You can restore design documents to a server node with the option, `design_doc_only=1`. You can restore from a backup file you create with `cbbackup`.

```
> ./cbrestore ~/backup http://10.3.1.10:8091 -x design_doc_only=1 -b a_bucket
 -B my_bucket

transfer design doc only. bucket msgs will be skipped.
done
```

This will restore design documents from the backup file `~/backup/a_bucket` to the destination bucket `my_bucket` in a cluster. If you backed up more than one source bucket, you will need to perform this command more than once. For instance, imagine you did a backup for a cluster with two data buckets and have the backup files `~/backup/bucket_one/design.json` and `~/backup/bucket_two/design.json` :

```
> ./cbrestore ~/backup http://10.3.1.10:8091 -x design_doc_only=1 -b
 bucket_one -B my_bucket

> ./cbrestore ~/backup http://10.3.1.10:8091 -x design_doc_only=1 -b
 bucket_two -B my_bucket
```

This will restore design documents in both backup files to a bucket in your cluster named `my_bucket` After you restore the design documents you can see them in Couchbase Web Console under the Views tab.

Using cbrestore for 1.8.x

You can use `cbrestore` to backup data from a Couchbase 1.8.x cluster, including 1.8. To do so you use the same command options you use when you backup a 2.0 cluster except you provide it the hostname and port for the 1.8.x cluster. You do not need to even install Couchbase Server 2.0 in order to use `cbrestore` 2.0 to backup Couchbase Server 1.8.x. . After you get the tool, go to the directory where you cloned the tool and perform the command. For instance:

```
./cbrestore ~/backup http://10.3.3.11:8091 -u Administrator -p password -B
 saslbucket_destination -b saslbucket_source
```

This restores all data in the `bucket-saslbucket_source` directory under `~/backups` on the physical machine where you run `cbbackup`. It will restore this data into a bucket named `saslbucket_destination` in the cluster with the node host:port of `10.3.3.11:8091`.

Be aware that if you are trying to restore data to a different cluster, that you should make sure that cluster should have the same number of vBuckets as the cluster that you backed up. If you attempt to restore data from a cluster to a cluster with a different number of vBuckets, it will fail when you use the default port of `8091`. The default number of vBuckets for Couchbase 2.0 is 1024; in earlier versions of Couchbase, you may have a different number of vBuckets. If you do want to restore data to a cluster with a different number of vBuckets, you should perform this command with port `11211`, which will accommodate the difference in vBuckets:

```
cbrestore /backups/backup-42 memcached://HOST:11211 \
    --bucket-source=sessions --bucket-destination=sessions2
```

Related Links

Couchbase CLI on page 352

Related topics

Couchbase command-line tools GitHub repository

Using hostnames on page 69

Views on page 278
The Views section allows you to manage your development and production views.

cbbackup tool on page 357
The `cbbackup` tool creates a copy of data from an entire running cluster, an entire bucket, a single node, or a single bucket on a single functioning node. Your node or cluster needs to be functioning in order to create the backup. Couchbase Server will write a copy of data onto disk.

cbtransfer tool on page 420

cbstats tool

You use the `cbstats` tool to get node- and cluster-level statistics about performance and items in storage. The tool can be found in the following locations, depending on your platform:

Linux	/opt/couchbase/bin/cbstats
Windows	C:\Program Files\Couchbase\Server\bin \cbstats.exe
Mac OS X	/Applications/Couchbase Server.app/ Contents/Resources/couchbase-core/bin/ cbstats

Be aware that this tool is a per-node, per-bucket operation. That means that if you want to perform this operation, you must specify the IP address of a node in the cluster and a named bucket. If you do not provided a named bucket, the server will apply the setting to any default bucket that exists at the specified node. If you want to perform this operation for an entire cluster, you will need to perform the command for every node/bucket combination that exists for that cluster.

Use this tool to get the couchbase node statistics. The general format for the command is:

```
> cbstats <IP>:11210 <command> -b <bucket_name> [-p <bucket_password>]
```

Where `BUCKET_HOST` is the hostname and port (`HOSTNAME[:PORT]`) combination for a Couchbase bucket, and `username` and `password` are the authentication for the named bucket. `COMMAND` (and `[options]`) are one of the follow options:

```
all
allocator
checkpoint [vbid]
dispatcher [logs]
hash [detail]
items
kvstore
kvtimings
raw argument
reset
slabs
tap [username password]
tapagg
timings
vkey keyname vbid
```

From these options, `all` and `timings` will be the main ones you will use to understand cluster or node performance. The other options are used by Couchbase internally and to help resolve customer support incidents.

For example, the `cbstats` output can be used with other command-line tools to sort and filter the data.

```
> watch --diff "cbstats \
    ip-10-12-19-81:11210 -b bucket_name -p bucket_password all | egrep 'item|
mem|flusher|ep_queue|bg|eje|resi|warm'"
```

Related Links

Couchbase CLI on page 352
Available stats on page 393
Checkpoint stats on page 394
Dispatcher stats and job logs on page 395
General form on page 395
Getting server timings on page 396
Getting TAP information on page 397

Related topics

Couchbase Server statistics on page 179
Couchbase Server provides statistics at multiple levels throughout the cluster.

Available stats

The following histograms are available from “timings” in the general form to describe when time was spent doing various things:

Stat	Description
bg_wait	bg fetches waiting in the dispatcher queue
bg_load	bg fetches waiting for disk
bg_tap_wait	tap bg fetches waiting in the dispatcher queue
bg_tap_load	tap bg fetches waiting for disk
pending_ops	client connections blocked for operations
	in pending vbuckets
storage_age	Analogous to ep_storage_age in main stats
data_age	Analogous to ep_data_age in main stats
get_cmd	servicing get requests
arith_cmd	servicing incr/decr requests
get_stats_cmd	servicing get_stats requests
get_vb_cmd	servicing vbucket status requests
set_vb_cmd	servicing vbucket set state commands
del_vb_cmd	servicing vbucket deletion commands
chk_persistence_cmd	waiting for checkpoint persistence
tap_vb_set	servicing tap vbucket set state commands
tap_vb_reset	servicing tap vbucket reset commands
tap_mutation	servicing tap mutations
notify_io	waking blocked connections

Stat	Description
paged_out_time	time (in seconds) objects are non-resident
disk_insert	waiting for disk to store a new item
disk_update	waiting for disk to modify an existing item
disk_del	waiting for disk to delete an item
disk_vb_del	waiting for disk to delete a vbucket
disk_commit	waiting for a commit after a batch of updates
disk_vbstate_snapshot	Time spent persisting vbucket state changes
item_alloc_sizes	Item allocation size counters (in bytes)

Related Links

cbstats tool on page 392

Checkpoint stats

Checkpoint stats provide detailed information on per-vbucket checkpoint data structure.

Like Hash stats, requesting these stats has some impact on performance. Therefore, please do not poll them from the server frequently. Each stat is prefixed with =vb_= followed by a number, a colon, and then each stat name.

Stat	Description
cursor_name:cursor_checkpoint_id	Checkpoint ID at which the cursor is
	name ‘cursor_name’ is pointing now
open_checkpoint_id	ID of the current open checkpoint
num_tap_cursors	Number of referencing TAP cursors
num_checkpoint_items	Number of total items in a checkpoint
	data structure
num_open_checkpoint_items	Number of items in the open checkpoint
num_checkpoints	Number of checkpoints in a checkpoint
	data structure
num_items_for_persistence	Number of items remaining for persistence
checkpoint_extension	True if the open checkpoint is in the
	extension mode
state	The state of the vbucket this checkpoint
	contains data for
last_closed_checkpoint_id	The last closed checkpoint number
persisted_checkpoint_id	The last persisted checkpoint number

Related Links

cbstats tool on page 392

Dispatcher stats and job logs

This provides the stats from AUX dispatcher and non-IO dispatcher, and from all the reader and writer threads running for the specific bucket. Along with stats, the job logs for each of the dispatchers and worker threads is also made available.

The following stats are available for the workers and dispatchers:

Stat	Description
state	Threads's current status: running, sleeping etc.
runtime	The amount of time since the thread started running
task	The activity/job the thread is involved with at the moment

The following stats are for individual job logs:

Stat	Description
starttime	The timestamp when the job started
runtime	Time it took for the job to run
task	The activity/job the thread ran during that time

Related Links

cbstats tool on page 392

General form

As this data is multi-dimensional, some parsing may be required for machine processing. It's somewhat human readable, but the =stats= script mentioned in the Getting Started section above will do fancier formatting for you.

The following sample statistics show that =disk_insert= took 8–16µs 9,488 times, 16–32µs 290 times, and so on.

```
STAT disk_insert_8,16 9488
STAT disk_insert_16,32 290
STAT disk_insert_32,64 73
STAT disk_insert_64,128 86
STAT disk_insert_128,256 48
STAT disk_insert_256,512 2
STAT disk_insert_512,1024 12
STAT disk_insert_1024,2048 1
```

The same statistics displayed through the =stats= CLI tool look like the following:

```
disk_insert (10008 total)
8us - 16us : ( 94.80%) 9488 ############################################
16us - 32us : ( 97.70%) 290 #
32us - 64us : ( 98.43%) 73
64us - 128us : ( 99.29%) 86
128us - 256us : ( 99.77%) 4
256us - 512us : ( 99.79%) 2
512us - 1ms : ( 99.91%) 12
1ms - 2ms : ( 99.92%) 1
```

Related Links

cbstats tool on page 392

Getting server timings

Be aware that this tool is a per-node, per-bucket operation. That means that if you want to perform this operation, you must specify the IP address of a node in the cluster and a named bucket. If you do not provided a named bucket, the server will apply the setting to any default bucket that exists at the specified node. If you want to perform this operation for an entire cluster, you will need to perform the command for every node/bucket combination that exists for that cluster.

The following is sample output from `cbstats timings`:

```
disk_insert (10008 total)
   8us - 16us    : ( 94.80%)8 ###########################################
   16us - 32us   : ( 97.70%)  290 #
   32us - 64us   : ( 98.43%)   73
   64us - 128us  : ( 99.29%)   86
   128us - 256us : ( 99.77%)   48
   256us - 512us : ( 99.79%)    2
   512us - 1ms   : ( 99.91%)   12
   1ms - 2ms     : ( 99.92%)    1
disk_commit (1 total)
    0 - 1s        : (100.00%) 1
 #############################################################
disk_vbstate_snapshot (2 total)
    4s - 8s       : (100.00%) 2
 #############################################################
get_stats_cmd (1535 total)
    ....
set_vb_cmd (1024 total)
    4us - 8us     : ( 97.95%) 1003
 ########################################################
    8us - 16us    : ( 98.83%)    9
    ....
```

The first statistic tells you that `disk_insert` took 8–16µs8 times, 16–32µs 290 times, and so forth.

The following are the possible return values provided by `cbstats timings`. The return values provided by this command depend on what has actually occurred on a data bucket:

bg_load	Background fetches waiting for disk
bg_wait	Background fetches waiting in the dispatcher queue
data_age	Age of data written to disk
disk_commit	Time waiting for a commit after a batch of updates
disk_del	Wait for disk to delete an item
disk_insert	Wait for disk to store a new item
disk_vbstate_snapshot	Time spent persisting vbucket state changes
disk_update	Wait time for disk to modify an existing item
get_cmd	Servicing get requests
get_stats_cmd	Servicing get_stats requests
set_vb_cmd	Servicing vbucket set state commands
item_alloc_sizes	Item allocation size counters (in bytes)
notify_io	Time for waking blocked connections
storage_age	Time since most recently persisted item was initially queued for storage.
tap_mutation	Time spent servicing tap mutations

Related Links

cbstats tool on page 392

Getting TAP information

Couchbase Server uses an internal protocol known as TAP to stream information about data changes between cluster nodes. Couchbase Server uses the TAP protocol during 1) rebalance, 2) replication at other cluster nodes, and 3) persistence of items to disk.

Be aware that this tool is a per-node, per-bucket operation. That means that if you want to perform this operation, you must specify the IP address of a node in the cluster and a named bucket. If you do not provided a named bucket, the server will apply the setting to any default bucket that exists at the specified node. If you want to perform this operation for an entire cluster, you will need to perform the command for every node/bucket combination that exists for that cluster.

The following statistics will be output in response to a `cbstats tap` request:

ep_tap_total_queue	Sum of tap queue sizes on the current tap queues
ep_tap_total_fetched	Sum of all tap messages sent
ep_tap_bg_max_pending	The maximum number of background jobs a tap connection may have
ep_tap_bg_fetched	Number of tap disk fetches
ep_tap_bg_fetch_requeued	Number of times a tap background fetch task is requeued.
ep_tap_fg_fetched	Number of tap memory fetches
ep_tap_deletes	Number of tap deletion messages sent
ep_tap_throttled	Number of tap messages refused due to throttling.
ep_tap_keepalive	How long to keep tap connection state after client disconnect.
ep_tap_count	Number of tap connections.
ep_tap_bg_num_samples	The number of tap background fetch samples included in the average
ep_tap_bg_min_wait	The shortest time (µs) for a tap item before it is serviced by the dispatcher
ep_tap_bg_max_wait	The longest time (µs) for a tap item before it is serviced by the dispatcher
ep_tap_bg_wait_avg	The average wait time (µs) for a tap item before it is serviced by the dispatcher
ep_tap_bg_min_load	The shortest time (µs) for a tap item to be loaded from the persistence layer
ep_tap_bg_max_load	The longest time (µs) for a tap item to be loaded from the persistence layer
ep_tap_bg_load_avg	The average time (µs) for a tap item to be loaded from the persistence layer
ep_tap_noop_interval	The number of secs between a no-op is added to an idle connection
ep_tap_backoff_period	The number of seconds the tap connection should back off after receiving ETMPFAIL

ep_tap_total_queue	Sum of tap queue sizes on the current tap queues
ep_tap_queue_fill	Total enqueued items
ep_tap_queue_drain	Total drained items
ep_tap_queue_backoff	Total back-off items
ep_tap_queue_backfill	Number of backfill remaining
ep_tap_queue_itemondisk	Number of items remaining on disk
ep_tap_throttle_threshold	Percentage of memory in use before we throttle tap streams
ep_tap_throttle_queue_cap	Disk write queue cap to throttle tap streams

You use the `cbstats tapagg` to get statistics from named tap connections which are logically grouped and aggregated together by prefixes.

For example, if all of your tap connections started with `rebalance_` or `replication_`, you could call `cbstats tapagg _` to request stats grouped by the prefix starting with _. This would return a set of statistics for `rebalance` and a set for `replication`. The following are possible values returned by `cbstats tapagg`:

[prefix]:count	Number of connections matching this prefix
[prefix]:qlen	Total length of queues with this prefix
[prefix]:backfill_remaining	Number of items needing to be backfilled
[prefix]:backoff	Total number of backoff events
[prefix]:drain	Total number of items drained
[prefix]:fill	Total number of items filled
[prefix]:itemondisk	Number of items remaining on disk
[prefix]:total_backlog_size	Number of remaining items for replication

Related Links

cbstats tool on page 392

Getting warmup information

Be aware that this tool is a per-node, per-bucket operation. That means that if you want to perform this operation, you must specify the IP address of a node in the cluster and a named bucket. If you do not provided a named bucket, the server will apply the setting to any default bucket that exists at the specified node. If you want to perform this operation for an entire cluster, you will need to perform the command for every node/bucket combination that exists for that cluster.

You can use `cbstats` to get information about server warmup, including the status of warmup and whether warmup is enabled. The following are two alternates to filter for the information:

```
cbstats hostname:port -b bucket_name -p bucket_password | grep 'warmup'

cbstats hostname:port -b bucket_name -p bucket_password raw warmup
```

- **ep_warmup_thread** - Indicates if the warmup has completed. Returns “running” or “complete”.
- **ep_warmup_state** - Indicates the current progress of the warmup:
  - Initial - Start warmup processes.
  - EstimateDatabaseItemCount - Estimating database item count.
  - KeyDump - Begin loading keys and metadata, but not documents, into RAM.

- CheckForAccessLog - Determine if an access log is available. This log indicates which keys have been frequently read or written.
- LoadingAccessLog - Load information from access log.
- LoadingData - The server is loading data first for keys listed in the access log, or if no log available, based on keys found during the 'Key Dump' phase.
- Done - The server is ready to handle read and write requests.

High-level warmup statistics that are available are as follows:

Name	Description	Value Type
ep_warmup_dups	Number of failures due to duplicate keys	Integer
ep_warmup_estimated_key_count	Estimated number of keys in database	Integer (DEFAULT = "unknown")
ep_warmup_estimated_value_count	Estimated number of key data to read based on the access log	Integer (DEFAULT = "unknown")
ep_warmup_keys_time	Total time spent by loading persisted keys	Integer
ep_warmup_min_items_threshold	Enable data traffic after loading this percentage of key data	Integer
ep_warmup_min_memory_threshold	Enable data traffic after filling this % of memory	Integer (%)
ep_warmup_oom	Number of out of memory failures during warmup	Integer
ep_warmup_state	What is current warmup state	String, refer to *WarmupStateTable*
ep_warmup_thread	Is warmup running?	String ("running", "complete")
ep_warmup_time	Total time spent by loading data (warmup)	Integer (microseconds)

There are also additional lower-level, detailed statistics returned by passing the keyword "warmup" for the command. For instance:

```
cbstats hostname:port -b bucket_name -p bucket_password raw warmup
```

The additional lower-level stats are as follows. Note that some of these items are also available as higher-level summary statistics about warmup:

Name	Description	Value Type
ep_warmup	Is warmup enabled?	String ("enabled")
ep_warmup_key_count	How many keys have been loaded?	Integer
ep_warmup_value_count	How many key values (data) have been loaded?	Integer
ep_warmup_dups	Number of failures due to duplicate keys	Integer
ep_warmup_estimated_key_count	Estimated number of keys in database	Integer (DEFAULT = "unknown")
ep_warmup_estimated_value_count	Estimated number of key data to read based on the access log	Integer (DEFAULT = "unknown")
ep_warmup_keys_time	Total time spent by loading persisted keys	Integer

Name	Description	Value Type
ep_warmup_min_items_threshold	Enable data traffic after loading this percentage of key data	Integer
ep_warmup_min_memory_threshold	Enable data traffic after filling this % of memory	Integer (%)
ep_warmup_oom	Number of out of memory failures during warmup	Integer
ep_warmup_state	What is current warmup state	String, refer to *WarmupStateTable*
ep_warmup_thread	Is warmup running?	String (“running”, “complete”)
ep_warmup_time	Total time spent by loading data (warmup)	Integer (microseconds)

Related Links

cbstats tool on page 392

Hash stats

Hash stats provide information on your vbucket hash tables.

Requesting these stats does affect performance, so don’t do it too regularly, but it’s useful for debugging certain types of performance issues. For example, if your hash table is tuned to have too few buckets for the data load within it, the =max_depth= will be too large and performance will suffer.

Stat	Description
avg_count	The average number of items per vbucket
avg_max	The average max depth of a vbucket hash table
avg_min	The average min depth of a vbucket hash table
largest_max	The largest hash table depth of in all vbuckets
largest_min	The largest minimum hash table depth of all vbuckets
max_count	The largest number of items in a vbucket
min_count	The smallest number of items in a vbucket
total_counts	The total number of items in all vbuckets

It is also possible to get more detailed hash tables stats by using ‘hash detail’. This will print per-vbucket stats.

Each stat is prefixed with =vb_= followed by a number, a colon, then the individual stat name.

For example, the stat representing the size of the hash table for vbucket 0 is =vb_0:size=.

Stat	Description
state	The current state of this vbucket
size	Number of hash buckets
locks	Number of locks covering hash table operations
min_depth	Minimum number of items found in a bucket
max_depth	Maximum number of items found in a bucket
reported	Number of items this hash table reports having
counted	Number of items found while walking the table

Stat	Description
resized	Number of times the hash table resized
mem_size	Running sum of memory used by each item
mem_size_counted	Counted sum of current memory used by each item

Related Links

cbstats tool on page 392

Stats key and Vkey

Stat	Description	K/V
key_cas	The keys current cas value	KV
key_data_age	How long the key has waited for its	KV
	value to be persisted (0 if clean)	KV
key_exptime	Expiration time from the epoch	KV
key_flags	Flags for this key	KV
key_is_dirty	If the value is not yet persisted	KV
key_last_modified_time	Last updated time	KV
key_valid	See description below	V
key_vb_state	The vbucket state of this key	KV

`key_valid=` can have the following responses:

- this_is_a_bug - Some case we didn't take care of.
- dirty - The value in memory has not been persisted yet.
- length_mismatch - The key length in memory doesn't match the length on disk.
- data_mismatch - The data in memory doesn't match the data on disk.
- flags_mismatch - The flags in memory don't match the flags on disk.
- valid - The key is both on disk and in memory
- ram_but_not_disk - The value doesn't exist yet on disk.
- item_deleted - The item has been deleted.

Related Links

cbstats tool on page 392

KV store stats

These provide various low-level stats and timings from the underlying KV storage system and useful to understand various states of the storage system.

The following stats are available for all database engine:

Stat	Description
open	Number of database open operations
close	Number of database close operations
readTime	Time spent in read operations
readSize	Size of data in read operations

Stat	Description
writeTime	Time spent in write operations
writeSize	Size of data in write operations
delete	Time spent in delete() calls

The following stats are available for the CouchStore database engine:

Stat	Description
backend_type	Type of backend database engine
commit	Time spent in CouchStore commit operation
commitRetry	Time spent in retry of commit operation
numLoadedVb	Number of Vbuckets loaded into memory
numCommitRetry	Number of commit retry
lastCommDocs	Number of docs in the last commit
failure_set	Number of failed set operation
failure_get	Number of failed get operation
failure_vbset	Number of failed vbucket set operation
save_documents	Time spent in CouchStore save documents operation

Related Links

cbstats tool on page 392

Memory stats

This provides various memory-related stats including the stats from tcmalloc. Note that tcmalloc stats are not available on some operating systems (e.g., Windows) that do not support tcmalloc.

Stat	Description
mem_used (deprecated)	Engine’s total memory usage
bytes	Engine’s total memory usage
ep_kv_size	Memory used to store item metadata, keys and values, no matter the vbucket’s state. If an item’s value is ejected, this stat will be decremented by the size of the item’s value.
ep_value_size	Memory used to store values for resident keys
ep_overhead	Extra memory used by transient data like persistence queue, replication queues, checkpoints, etc
ep_max_size	Max amount of data allowed in memory

Stat	Description
ep_mem_low_wat	Low water mark for auto-evictions
ep_mem_high_wat	High water mark for auto-evictions
ep_oom_errors	Number of times unrecoverable OOMs happened while processing operations
ep_tmp_oom_errors	Number of times temporary OOMs happened while processing operations
ep_mem_tracker_enabled	If smart memory tracking is enabled
tcmalloc_allocated_bytes	Engine's total memory usage reported from tcmalloc
tcmalloc_heap_size	Bytes of system memory reserved by tcmalloc
tcmalloc_free_bytes	Number of bytes in free, mapped pages in page heap
tcmalloc_unmapped_bytes	Number of bytes in free, unmapped pages in page heap. These are bytes that have been released back to OS
tcmalloc_max_thread_cache_bytes	A limit to how much memory TCMalloc dedicates for small objects
tcmalloc_current_thread_cache_bytes	A measure of some of the memory TCMalloc is using for small objects

Related Links

cbstats tool on page 392

Pending vBucket stats

Stat	Description
vb_pending_num	Number of pending vBuckets
vb_pending_curr_items	Number of in memory items
vb_pending_num_non_resident	Number of non-resident items
vb_pending_perc_mem_resident	% memory resident
vb_pending_eject	Number of times item values got ejected
vb_pending_expired	Number of times an item was expired
vb_pending_ht_memory	Memory overhead of the hashtable
vb_pending_itm_memory	Total item memory
vb_pending_meta_data_memory	Total metadata memory
vb_pending_ops_create	Number of create operations
vb_pending_ops_update	Number of update operations

Stat	Description
vb_pending_ops_delete	Number of delete operations
vb_pending_ops_reject	Number of rejected operations
vb_pending_queue_size	Pending items in disk queue
vb_pending_queue_memory	Memory used for disk queue
vb_pending_queue_age	Sum of disk queue item age in milliseconds
vb_pending_queue_pending	Total bytes of pending writes
vb_pending_queue_fill	Total enqueued items
vb_pending_queue_drain	Total drained items

Related Links

cbstats tool on page 392

Replica vBucket stats

Stat	Description
vb_replica_num	Number of replica vBuckets
vb_replica_curr_items	Number of in memory items
vb_replica_num_non_resident	Number of non-resident items
vb_replica_perc_mem_resident	% memory resident
vb_replica_eject	Number of times item values got ejected
vb_replica_expired	Number of times an item was expired
vb_replica_ht_memory	Memory overhead of the hashtable
vb_replica_itm_memory	Total item memory
vb_replica_meta_data_memory	Total metadata memory
vb_replica_ops_create	Number of create operations
vb_replica_ops_update	Number of update operations
vb_replica_ops_delete	Number of delete operations
vb_replica_ops_reject	Number of rejected operations
vb_replica_queue_size	Replica items in disk queue
vb_replica_queue_memory	Memory used for disk queue
vb_replica_queue_age	Sum of disk queue item age in milliseconds
vb_replica_queue_pending	Total bytes of pending writes
vb_replica_queue_fill	Total enqueued items
vb_replica_queue_drain	Total drained items

Related Links

cbstats tool on page 392

Stats reset

Resets the list of stats below.

Reset Stats
ep_bg_load
ep_bg_wait
ep_bg_max_load
ep_bg_min_load
ep_bg_max_wait
ep_bg_min_wait
ep_commit_time
ep_flush_duration
ep_flush_duration_highwat
ep_io_num_read
ep_io_num_write
ep_io_read_bytes
ep_io_write_bytes
ep_items_rm_from_checkpoints
ep_num_eject_failures
ep_num_pager_runs
ep_num_not_my_vbuckets
ep_num_value_ejects
ep_pending_ops_max
ep_pending_ops_max_duration
ep_pending_ops_total
ep_storage_age
ep_storage_age_highwat
ep_too_old
ep_too_young
ep_tap_bg_load_avg
ep_tap_bg_max_load
ep_tap_bg_max_wait
ep_tap_bg_min_load
ep_tap_bg_min_wait
ep_tap_bg_wait_avg
ep_tap_throttled
ep_tap_total_fetched
ep_vbucket_del_max_walltime
pending_ops

Reset Histograms stats
bg_load
bg_wait
bg_tap_load
bg_tap_wait
chk_persistence_cmd
data_age
del_vb_cmd
disk_insert
disk_update
disk_del
disk_vb_del
disk_commit
get_stats_cmd
item_alloc_sizes
get_vb_cmd
notify_io
pending_ops
set_vb_cmd
storage_age
tap_mutation
tap_vb_reset
tap_vb_set

Related Links

cbstats tool on page 392

Tap stats

Stat	Description
ep_tap_ack_grace_period	The amount of time to wait for a tap acks before disconnecting
ep_tap_ack_interval	The amount of messages a tap producer should send before requesting an ack
ep_tap_ack_window_size	The maximum amount of ack requests that can be sent before the consumer sends a response ack. When the window is full the tap stream is paused
ep_tap_queue_backfillremaining	Number of items needing to be backfilled

Stat	Description
ep_tap_total_backlog_size	Number of remaining items for replication
ep_tap_total_queue	Sum of tap queue sizes on the current tap queues
ep_tap_total_fetched	Sum of all tap messages sent
ep_tap_bg_max_pending	The maximum number of bg jobs a tap connection may have
ep_tap_bg_fetched	Number of tap disk fetches
ep_tap_bg_fetch_requeued	Number of times a tap bg fetch task is requeued
ep_tap_fg_fetched	Number of tap memory fetches
ep_tap_deletes	Number of tap deletion messages sent
ep_tap_throttled	Number of tap messages refused due to throttling
ep_tap_count	Number of tap connections
ep_tap_bg_num_samples	The number of tap bg fetch samples included in the avg
ep_tap_bg_min_wait	The shortest time (µs) for a tap item before it is serviced by the dispatcher
ep_tap_bg_max_wait	The longest time (µs) for a tap item before it is serviced by the dispatcher
ep_tap_bg_wait_avg	The average wait time (µs) for a tap item before it is serviced by the dispatcher
ep_tap_bg_min_load	The shortest time (µs) for a tap item to be loaded from the persistence layer
ep_tap_bg_max_load	The longest time (µs) for a tap item to be loaded from the persistence layer
ep_tap_bg_load_avg	The average time (µs) for a tap item to be loaded from the persistence layer
ep_tap_noop_interval	The number of secs between a noop is added to an idle connection
ep_tap_backoff_period	The number of seconds the tap connection should back off after receiving ETMPFAIL
ep_tap_queue_fill	Total enqueued items
ep_tap_queue_drain	Total drained items
ep_tap_queue_backoff	Total back-off items
ep_tap_queue_backfill	Number of backfill remaining

Stat	Description
ep_tap_queue_itemondisk	Number of items remaining on disk
ep_tap_throttle_threshold	Percentage of memory in use before we
	throttle tap streams
ep_tap_throttle_queue_cap	Disk write queue cap to throttle
	tap streams

Per Tap client stats

Each stat begins with =ep_tapq:= followed by a unique /client_id/ and another colon. For example, if your client is named, =slave1=, the =qlen= stat would be =ep_tapq:slave1:qlen=.

Stat	Description	P/C
type	The kind of tap connection (producer or	PC
	consumer)	
created	Creation time for the tap connection	PC
supports_ack	true if the connection use acks	PC
connected	true if this client is connected	PC
disconnects	Number of disconnects from this client	PC
reserved	true if the tap stream is reserved	P
suspended	true if the tap stream is suspended	P
qlen	Queue size for the given client_id	P
qlen_high_pri	High priority tap queue items	P
qlen_low_pri	Low priority tap queue items	P
vb_filters	Size of connection vbucket filter set	P
vb_filter	The content of the vbucket filter	P
rec_fetched	Tap messages sent to the client	P
rec_skipped	Number of messages skipped due to	P
	tap reconnect with a different filter	P
idle	True if this connection is idle	P
has_queued_item	True if there are any remaining items	P
	from hash table or disk	
bg_result_size	Number of ready background results	P
bg_jobs_issued	Number of background jobs started	P
bg_jobs_completed	Number of background jobs completed	P
flags	Connection flags set by the client	P
pending_disconnect	true if we’re hanging up on this client	P
paused	true if this client is blocked	P

Stat	Description	P/C
pending_backfill	true if we're still backfilling keys	P
	for this connection	P
pending_disk_backfill	true if we're still backfilling keys	P
	from disk for this connection	P
backfill_completed	true if all items from backfill is	P
	successfully transmitted to the client	P
backfill_start_timestamp	Timestamp of backfill start	P
reconnects	Number of reconnects from this client	P
backfill_age	The age of the start of the backfill	P
ack_seqno	The current tap ACK sequence number	P
recv_ack_seqno	Last receive tap ACK sequence number	P
ack_log_size	Tap ACK backlog size	P
ack_window_full	true if our tap ACK window is full	P
seqno_ack_requested	The seqno of the ack message that the	P
	producer is wants to get a response for	
expires	When this ACK backlog expires	P
queue_memory	Memory used for tap queue	P
queue_fill	Total queued items	P
queue_drain	Total drained items	P
queue_backoff	Total back-off items	P
queue_backfillremaining	Number of backfill remaining	P
queue_itemondisk	Number of items remaining on disk	P
total_backlog_size	Num of remaining items for replication	P
total_noops	Number of NOOP messages sent	P
num_checkpoint_end	Number of chkpoint end operations	C
num_checkpoint_end_failed	Number of chkpoint end operations failed	C
num_checkpoint_start	Number of chkpoint end operations	C
num_checkpoint_start_failed	Number of chkpoint end operations failed	C
num_delete	Number of delete operations	C
num_delete_failed	Number of failed delete operations	C
num_flush	Number of flush operations	C

Stat	Description	P/C
num_flush_failed	Number of failed flush operations	C
num_mutation	Number of mutation operations	C
num_mutation_failed	Number of failed mutation operations	C
num_opaque	Number of opaque operation	C
num_opaque_failed	Number of failed opaque operations	C
num_vbucket_set	Number of vbucket set operations	C
num_vbucket_set_failed	Number of failed vbucket set operations	C
num_unknown	Number of unknown operations	C

Tap aggregated stats

Aggregated tap stats allow named tap connections to be logically grouped and aggregated together by prefixes.

For example, if all of your tap connections started with =rebalance= *or =replication=*, you could call =stats tapagg *= to request stats grouped by everything before the first* == character, giving you a set for =rebalance= and a set for =replication=.

Results

Stat	Description
[prefix]:count	Number of connections matching this prefix
[prefix]:qlen	Total length of queues with this prefix
[prefix]:backfill_remaining	Number of items needing to be backfilled
[prefix]:backoff	Total number of backoff events
[prefix]:drain	Total number of items drained
[prefix]:fill	Total number of items filled
[prefix]:itemondisk	Number of items remaining on disk
[prefix]:total_backlog_size	Num of remaining items for replication

Related Links

cbstats tool on page 392

Timing stats

Timing stats provide histogram data from high resolution timers over various operations within the system.

Related Links

cbstats tool on page 392

Toplevel stats

The following provides the stats that are created by `cbstats`:

Stat	Description
uuid	The unique identifier for the bucket
ep_version	Version number of ep_engine

Stat	Description
ep_storage_age	Seconds since most recently stored object was initially queued
ep_storage_age_highwat	ep_storage_age high water mark
ep_startup_time	System-generated engine startup time
ep_max_txn_size	Max number of updates per transaction
ep_data_age	Seconds since most recently stored object was modified
ep_data_age_highwat	ep_data_age high water mark
ep_too_young	Number of times an object was not stored due to being too young
ep_too_old	Number of times an object was stored after being dirty too long
ep_total_enqueued	Total number of items queued for persistence
ep_total_new_items	Total number of persisted new items
ep_total_del_items	Total number of persisted deletions
ep_total_persisted	Total number of items persisted
ep_item_flush_failed	Number of times an item failed to flush due to storage errors
ep_item_commit_failed	Number of times a transaction failed to commit due to storage errors
ep_item_begin_failed	Number of times a transaction failed to start due to storage errors
ep_expired_access	Number of times an item was expired on application access.
ep_expired_pager	Number of times an item was expired by ep engine item pager
ep_item_flush_expired	Number of times an item is not flushed due to the expiry of the item
ep_queue_size	Number of items queued for storage
ep_flusher_todo	Number of items currently being written
ep_flusher_state	Current state of the flusher thread
ep_commit_num	Total number of write commits
ep_commit_time	Number of milliseconds of most recent commit

Stat	Description
ep_commit_time_total	Cumulative milliseconds spent committing
ep_vbucket_del	Number of vbucket deletion events
ep_vbucket_del_fail	Number of failed vbucket deletion events
ep_vbucket_del_max_walltime	Max wall time (µs) spent by deleting a vbucket
ep_vbucket_del_avg_walltime	Avg wall time (µs) spent by deleting a vbucket
ep_flush_duration_total	Cumulative seconds spent flushing
ep_flush_all	True if disk flush_all is scheduled
ep_num_ops_get_meta	Number of getMeta operations
ep_num_ops_set_meta	Number of setWithMeta operations
ep_num_ops_del_meta	Number of delWithMeta operations
ep_num_ops_set_meta_res_failed	Number of setWithMeta ops that failed conflict resolution
ep_num_ops_del_meta_res_failed	Number of delWithMeta ops that failed conflict resolution
ep_num_ops_set_ret_meta	Number of setRetMeta operations
ep_num_ops_del_ret_meta	Number of delRetMeta operations
curr_items	Num items in active vbuckets (temp + live)
curr_temp_items	Num temp items in active vbuckets
curr_items_tot	Num current items including those not active (replica, dead and pending states)
ep_kv_size	Memory used to store item metadata, keys and values, no matter the vbucket's state. If an item's value is ejected, this stats will be decremented by the size of the item's value.
ep_value_size	Memory used to store values for resident keys
ep_overhead	Extra memory used by transient data like persistence queues, replication

Stat	Description
	queues, checkpoints, etc
ep_mem_low_wat	Low water mark for auto-evictions
ep_mem_high_wat	High water mark for auto-evictions
ep_total_cache_size	The total byte size of all items, no matter the vbucket’s state, no matter if an item’s value is ejected
ep_oom_errors	Number of times unrecoverable OOMs happened while processing operations
ep_tmp_oom_errors	Number of times temporary OOMs happened while processing operations
ep_mem_tracker_enabled	True if memory usage tracker is enabled
ep_bg_fetched	Number of items fetched from disk
ep_bg_meta_fetched	Number of meta items fetched from disk
ep_bg_remaining_jobs	Number of remaining bg fetch jobs
ep_max_bg_remaining_jobs	Max number of remaining bg fetch jobs that we have seen in the queue so far
ep_tap_bg_fetched	Number of tap disk fetches
ep_tap_bg_fetch_requeued	Number of times a tap bg fetch task is requeued
ep_num_pager_runs	Number of times we ran pager loops to seek additional memory
ep_num_expiry_pager_runs	Number of times we ran expiry pager loops to purge expired items from memory/disk
ep_num_access_scanner_runs	Number of times we ran access scanner to snapshot working set
ep_access_scanner_num_items	Number of items that last access scanner task swept to access log.
ep_access_scanner_task_time	Time of the next access scanner task (GMT)
ep_access_scanner_last_runtime	Number of seconds that last access scanner task took to complete.
ep_items_rm_from_checkpoints	Number of items removed from closed unreferenced checkpoints
ep_num_value_ejects	Number of times item values got

Stat	Description
	ejected from memory to disk
ep_num_eject_failures	Number of items that could not be ejected
ep_num_not_my_vbuckets	Number of times Not My VBucket exception happened during runtime
ep_tap_keepalive	Tap keepalive time
ep_dbname	DB path
ep_io_num_read	Number of io read operations
ep_io_num_write	Number of io write operations
ep_io_read_bytes	Number of bytes read (key + values)
ep_io_write_bytes	Number of bytes written (key + values)
ep_pending_ops	Number of ops awaiting pending vbuckets
ep_pending_ops_total	Total blocked pending ops since reset
ep_pending_ops_max	Max ops seen awaiting 1 pending vbucket
ep_pending_ops_max_duration	Max time (µs) used waiting on pending vbuckets
ep_bg_num_samples	The number of samples included in the average
ep_bg_min_wait	The shortest time (µs) in the wait queue
ep_bg_max_wait	The longest time (µs) in the wait queue
ep_bg_wait_avg	The average wait time (µs) for an item before it's serviced by the dispatcher
ep_bg_min_load	The shortest load time (µs)
ep_bg_max_load	The longest load time (µs)
ep_bg_load_avg	The average time (µs) for an item to be loaded from the persistence layer
ep_num_non_resident	The number of non-resident items
ep_bg_wait	The total elapse time for the wait queue
ep_bg_load	The total elapse time for items to be loaded from the persistence layer
ep_allow_data_loss_during_shutdown	Whether data loss is allowed during

Stat	Description
	server shutdown
ep_alog_block_size	Access log block size
ep_alog_path	Path to the access log
ep_alog_sleep_time	Interval between access scanner runs in minutes
ep_alog_task_time	Hour in GMT time when access scanner task is scheduled to run
ep_backend	The backend that is being used for data persistence
ep_bg_fetch_delay	The amount of time to wait before doing a background fetch
ep_chk_max_items	The number of items allowed in a checkpoint before a new one is created
ep_chk_period	The maximum lifetime of a checkpoint before a new one is created
ep_chk_persistence_remains	Number of remaining vbuckets for checkpoint persistence
ep_chk_persistence_timeout	Timeout for vbucket checkpoint persistence
ep_chk_remover_stime	The time interval for purging closed checkpoints from memory
ep_config_file	The location of the ep-engine config file
ep_couch_bucket	The name of this bucket
ep_couch_host	The hostname that the CouchDB views server is listening on
ep_couch_port	The port the CouchDB views server is listening on
ep_couch_reconnect_sleeptime	The amount of time to wait before reconnecting to CouchDB
ep_couch_response_timeout	Length of time to wait for a response from CouchDB before reconnecting
ep_data_traffic_enabled	Whether or not data traffic is enabled for this bucket
ep_degraded_mode	True if the engine is either warming up or data traffic is disabled

Stat	Description
ep_exp_pager_stime	The time interval for purging expired items from memory
ep_expiry_window	Expiry window to not persist an object that is expired
ep_failpartialwarmup	True if we want kill the bucket if warmup fails
ep_flushall_enabled	True if this bucket allows the use of the flush_all command
ep_getl_default_timeout	The default getl lock duration
ep_getl_max_timeout	The maximum getl lock duration
ep_ht_locks	The amount of locks per vb hashtable
ep_ht_size	The initial size of each vb hashtable
ep_item_num_based_new_chk	True if the number of items in the current checkpoint plays a role in a new checkpoint creation
ep_keep_closed_chks	True if we want to keep the closed checkpoints for each vbucket unless the memory usage is above high water mark
ep_max_checkpoints	The maximum amount of checkpoints that can be in memory per vbucket
ep_max_item_size	The maximum value size
ep_max_size	The maximum amount of memory this bucket can use
ep_max_vbuckets	The maximum amount of vbuckets that can exist in this bucket
ep_mutation_mem_threshold	The ratio of total memory available that we should start sending temp oom or oom message when hitting
ep_pager_active_vb_pcnt	Active vbuckets paging percentage
ep_tap_ack_grace_period	The amount of time to wait for a tap acks before disconnecting
ep_tap_ack_initial_sequence_number	The initial sequence number for a tap ack when a tap stream is created
ep_tap_ack_interval	The amount of messages a tap producer should send before requesting an ack

Stat	Description
ep_tap_ack_window_size	The maximum amount of ack requests that can be sent before the consumer sends a response ack. When the window is full the tap stream is paused.
ep_tap_backfill_resident	The resident ratio for deciding how to do backfill. If under the ratio we schedule full disk backfill. If above the ratio then we do bg fetches for non-resident items.
ep_tap_backlog_limit	The maximum amount of backfill items that can be in memory waiting to be sent to the tap consumer
ep_tap_backoff_period	The number of seconds the tap connection
ep_tap_bg_fetch_requeued	Number of times a tap bg fetch task is requeued
ep_tap_bg_max_pending	The maximum number of bg jobs a tap connection may have
ep_tap_noop_interval	Number of seconds between a noop is sent on an idle connection
ep_tap_requeue_sleep_time	The amount of time to wait before a failed tap item is requeued
ep_tap_throttle_cap_pcnt	Percentage of total items in write queue at which we throttle tap input
ep_tap_throttle_queue_cap	Max size of a write queue to throttle incoming tap input
ep_tap_throttle_threshold	Percentage of max mem at which we begin NAKing tap input
ep_uncommitted_items	The amount of items that have not been written to disk
ep_vb0	Whether vbucket 0 should be created by default
ep_waitforwarmup	True if we should wait for the warmup process to complete before enabling traffic
ep_warmup	Shows if warmup is enabled / disabled

Stat	Description
ep_warmup_batch_size	The size of each batch loaded during warmup
ep_warmup_dups	Number of Duplicate items encountered during warmup
ep_warmup_min_items_threshold	Percentage of total items warmed up before we enable traffic
ep_warmup_min_memory_threshold	Percentage of max mem warmed up before we enable traffic
ep_warmup_oom	The amount of oom errors that occurred during warmup
ep_warmup_thread	The status of the warmup thread
ep_warmup_time	The amount of time warmup took

Related Links

cbstats tool on page 392

vBucket total stats

Stat	Description
ep_vb_total	Total vBuckets (count)
curr_items_tot	Total number of items
curr_items	Number of active items in memory
curr_temp_items	Number of temporary items in memory
vb_dead_num	Number of dead vBuckets
ep_diskqueue_items	Total items in disk queue
ep_diskqueue_memory	Total memory used in disk queue
ep_diskqueue_fill	Total enqueued items on disk queue
ep_diskqueue_drain	Total drained items on disk queue
ep_diskqueue_pending	Total bytes of pending writes
ep_vb_snapshot_total	Total VB state snapshots persisted in disk
ep_meta_data_memory	Total memory used by meta data

Related Links

cbstats tool on page 392

Read-write thread stats

The `cbstats` tools provides the status of read-write threads for buckets.

Example

View the status of this setting with `cbstats` tools and the `workload` parameter.

```
/opt/couchbase/bin/cbstats hostname:11210 -b bucket_name raw
 workload

 ep_workload:num_readers: 3
 ep_workload:num_shards:  3
 ep_workload:num_writers: 2
 ep_workload:policy:      Optimized for read data access
```

This result example indicates three reader threads and two writer threads on `bucket_name` in the cluster at `hostname:11210`. The vBucket map for the data bucket is grouped into multiple shards, where one read worker accesses one of the shards. In this example, there is one reader for each of the three shards. This report also shows that read data access is optimized because there are more reader threads than writer threads for the bucket.

Related Links

cbstats tool on page 392

Warmup

Stats `_warmup_` shows statistics related to warmup logic

Stat	Description
ep_warmup	Shows if warmup is enabled / disabled
ep_warmup_estimated_key_count	Estimated number of keys in database
ep_warmup_estimated_value_count	Estimated number of values in database
ep_warmup_state	The current state of the warmup thread
ep_warmup_thread	Warmup thread status
ep_warmup_key_count	Number of keys warmed up
ep_warmup_value_count	Number of values warmed up
ep_warmup_dups	Duplicates encountered during warmup
ep_warmup_oom	OOMs encountered during warmup
ep_warmup_item_expired	Number of items expired during warmup
ep_warmup_time	Time (µs) spent by warming data
ep_warmup_keys_time	Time (µs) spent by warming keys
ep_warmup_mutation_log	Number of keys present in mutation log
ep_warmup_access_log	Number of keys present in access log
ep_warmup_min_items_threshold	Percentage of total items warmed up
	before we enable traffic
ep_warmup_min_memory_threshold	Percentage of max mem warmed up before
	we enable traffic
ep_warmup_estimate_time	Estimated time in microseconds to do warmup

Related Links

cbstats tool on page 392

cbtransfer tool

You use this tool to transfer data and design documents between two clusters or from a file to a cluster. With this tool you can also create a copy of data from a node that no longer running. This tool is the underlying, generic data transfer tool that `cbbackup` and `cbrestore` are built upon. It is a lightweight extract-transform-load (ETL) tool that can move data from a source to a destination. The source and destination parameters are similar to URLs or file paths.

Note

`cbbackup`, `cbrestore` and `cbtransfer` do not communicate with external IP addresses for server nodes outside of a cluster. They can only communicate with nodes from a node list obtained within a cluster. You should perform backup, restore, or transfer to data from a node within a Couchbase cluster. This also means that if you install Couchbase Server with the default IP address, you cannot use an external hostname to access it. For general information about hostnames for the server, see [Using Hostnames with Couchbase Server](../cb-install/#couchbase-getting-started-hostnames).

The tool is at the following locations:

Linux	**`/opt/couchbase/bin/`**
Windows	`C:\Program Files\Couchbase\Server\bin\`
Mac OS X	`/Applications/Couchbase Server.app/Contents/Resources/couchbase-core/bin/`

The following is the syntax and examples for this command:

```
> ./cbtransfer [options] source destination


Examples:
  cbtransfer http://SOURCE:8091 /backups/backup-42
  cbtransfer /backups/backup-42 http://DEST:8091
  cbtransfer /backups/backup-42 couchbase://DEST:8091
  cbtransfer http://SOURCE:8091 http://DEST:8091
  cbtransfer 1.8_COUCHBASE_BUCKET_MASTER_DB_SQLITE_FILE http://DEST:8091
  cbtransfer file.csv http://DEST:8091
```

The following are the standard command options which you can also view with `cbtransfer -h`:

-h, –help	**Command help**
–add	Use –add instead of –set in order to not overwrite existing items in the destination
-b BUCKET_SOURCE	Single named bucket from source cluster to transfer
-B BUCKET_DESTINATION, –bucket-destination=BUCKET_DESTINATION	Single named bucket on destination cluster which receives transfer. This allows you to transfer to a bucket with a different name as your source bucket. If you do not provide defaults to the same name as the bucket-source
-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 cluster or server node

-h, –help	Command help
-t THREADS, –threads=THREADS	Number of concurrent workers threads performing the transfer. Defaults to 4.
-v, –verbose	Verbose logging; provide more verbosity
-x EXTRA, –extra=EXTRA	Provide extra, uncommon config parameters
–single-node	Transfer from a single server node in a source cluster. This single server node is a source node URL
–source-vbucket-state=SOURCE_VBUCKET_STATE	Only transfer from source vbuckets in this state, such as 'active' (default) or 'replica'. Must be used with Couchbase cluster as source.
–destination-vbucket-state=DESTINATION_VBUCKET_STATE	Only transfer to destination vbuckets in this state, such as 'active' (default) or 'replica'. Must be used with Couchbase cluster as destination.
–destination-operation=DESTINATION_OPERATION	Perform this operation on transfer. "set" will override an existing document, 'add' will not override, 'get' will load all keys transferred from a source cluster into the caching layer at the destination.
`/path/to/filename`	Export a.csv file from the server or import a.csv file to the server.

The following are extra, specialized command options you use in this form `cbtransfer -x [EXTRA OPTIONS]`:

batch_max_bytes=400000	Transfer this # of bytes per batch.
batch_max_size=1000	Transfer this # of documents per batch
cbb_max_mb=100000	Split backup file on destination cluster if it exceeds MB
max_retry=10	Max number of sequential retries if transfer fails
nmv_retry=1	0 or 1, where 1 retries transfer after a NOT_MY_VBUCKET message. Default of 1.
recv_min_bytes=4096	Amount of bytes for every TCP/IP batch transferred
report=5	Number batches transferred before updating progress bar in console
report_full=2000	Number batches transferred before emitting progress information in console
try_xwm=1	As of 2.1, transfer documents with metadata. 1 is default. 0 should only be used if you transfer from 1.8.x to 1.8.x.
data_only=0	For value 1, only transfer data from a backup file or cluster.
design_doc_only=0	For value 1, transfer design documents only from a backup file or cluster. Defaults to 0.

The most important way you can use this tool is to transfer data from a Couchbase node that is no longer running to a cluster that is running:

```
./cbtransfer \
       couchstore-files://COUCHSTORE_BUCKET_DIR \
       couchbase://HOST:PORT \
       --bucket-destination=DESTINATION_BUCKET
```

```
./cbtransfer \
       couchstore-files:///opt/couchbase/var/lib/couchbase/data/default \
       couchbase://10.5.3.121:8091 \
       --bucket-destination=foo
```

Upon success, the tool will output as follows:

```
[####################] 100.0% (10000/10000 msgs)
bucket: bucket_name, msgs transferred...
      : total | last | per sec
batch : 1088 | 1088 | 554.8
byte : 5783385 | 5783385 | 3502156.4
msg : 10000 | 10000 | 5230.9
done
```

This shows we successfully transferred 10000 total documents in batch size of 1088 documents each. This next examples shows how you can send all the data from a node to standard output:

```
> ./cbtransfer http://10.5.2.37:8091/ stdout:
```

Will produce a output as follows:

```
set pymc40 0 0 10
0000000000
set pymc16 0 0 10
0000000000
set pymc9 0 0 10
0000000000
set pymc53 0 0 10
0000000000
set pymc34 0 0 10
0000000000
```

Note: Couchbase Server stores all data from a bucket, node or cluster, but not the associated design documents. To do so, you should explicitly use `cbbackup` to store the information and `cbrestore` to read it back into memory.

Exporting and Importing CSV Files

You can import and export well-formed.csv files with `cbtransfer`. This will import data into Couchbase Server as documents and will export documents from the server into comma-separated values. This does not include any design documents associated with a bucket in the cluster.

For example imagine you have records as follows in the default bucket in a cluster:

```
re-fdeea652a89ec3e9,
0,
0,
4271152681275955,
"{""key"":""re-fdeea652a89ec3e9"",
 ""key_num"":4112,
 ""name"":""fdee c3e"",
 ""email"":""fdee@ea.com"",
 ""city"":""a65"",
 ""country"":""2a"",
 ""realm"":""89"",
 ""coins"":650.06,
 ""category"":1,
 ""achievements"":[77, 149, 239, 37, 76],""body"":""xc4ca4238a0b923820d
 .......
""}"
......
```

Where `re-fdeea652a89ec3e9` is the document ID, 0 are flags, 0 is the expiration and the CAS value is `4271152681275955`. The actual value in this example is the hash starting with `"{""key"".......` To export these items to a.csv file perform this command:

```
./cbtransfer http://[hostname]:[port] csv:./data.csv -b default -u
 Administrator -p password
```

Will transfer all items from the default bucket, `-b default` available at the node `http://localhost:8091` and put the items into the `/data.csv` file. If you provide another named bucket for the `-b` option, it will export items from that named bucket. You will need to provide credentials for the cluster when you export items from a bucket in the cluster. You will see output similar to that in other `cbtransfer` scenarios:

```
[####################] 100.0% (10000/10000 msgs)
bucket: default, msgs transferred...
       : total | last | per sec
 batch : 1053 | 1053 | 550.8
 byte : 4783385 | 4783385 | 2502156.4
 msg : 10000 | 10000 | 5230.9
2013-05-08 23:26:45,107: mt warning: cannot save bucket design on a CSV
 destination
done
```

This shows we transferred 1053 batches of data at 550.8 batches per second. The tool outputs “cannot save bucket design….” to indicate that no design documents were exported. To import information from a.csv file to a named bucket in a cluster:

```
./cbtransfer /data.csv http://[hostname]:[port] -B bucket_name -u
 Administrator -p password
```

If your.csv is not correctly formatted you will see the following error during import:

```
w0 error: fails to read from csv file, .....
```

Transferring Design Documents Only

You can transfer design documents from one cluster to another one with the option, `design_doc_only=1` :

```
> ./cbtransfer http://10.5.2.30:8091 http://10.3.1.10:8091 -x
 design_doc_only=1 -b bucket_one -B bucket_two
transfer design doc only. bucket msgs will be skipped.
done
```

This will transfer all design documents associated with `bucket_one` to `bucket_two` on the cluster with node `http://10.3.1.10:8091`. In Couchbase Web Console you can see this updated design documents when you click on the View tab and select `bucket_two` in the drop-down.

Related Links

Couchbase CLI on page 352

cbworkloadgen tool

Tool that generates random data and perform read/writes for Couchbase Server. This is useful for testing your Couchbase node.

Operating System	Location
Linux	`/opt/couchbase/bin/tools/`
Windows	`C:\Program Files\Couchbase\Server\bin\tools\`
Mac OS X	`/Applications/Couchbase Server.app/Contents/Resources/couchbase-core/bin/tools/`

The following is the standard command format:

```
cbworkloadgen Usage:
cbworkloadgen -n host:port -u [username] -p [password]

Options are as follows:

-r [number] // % of workload will be writes, remainder will be reads
--ratio-sets=[number] // 95% of workload will be writes, 5% will be reads
-i [number]    // number of inserted items
-l // loop forever until interrupted by user
-t // set number of concurrent threads
-v // verbose mode
```

For example, to generate workload on a given Couchbase node and open port on that node:

```
> ./cbworkloadgen -n 10.17.30.161:8091 -u Administrator -p password
```

Will produce a result similar to the following if successful:

```
[####################] 100.0% (10527/10526 msgs)
bucket: default, msgs transferred...
       :                total |       last |    per sec
 batch :                   11 |         11 |        2.2
 byte  :               105270 |     105270 |    21497.9
 msg   :                10527 |      10527 |     2149.8
done
```

When you check the data bucket you will see 10000 new items of with random keys and values such as the following item:

```
pymc0    "MDAwMDAwMDAwMA=="
```

Related Links

Couchbase CLI on page 352

Couchbase REST API

The Couchbase Server 3.0 REST API provides the following topics:

Related Links

REST API overview on page 426

Cluster REST API on page 429

Server nodes REST API on page 444
A Couchbase Server instance, known as node, is a physical or virtual machine running Couchbase Server. Each node is as a member of a cluster.

Rack Awareness REST API on page 452
The Rack Awareness feature allows logical groupings of servers on a cluster where each server group physically belongs to a rack or availability zone.

Buckets REST API on page 456
The bucket management and configuration REST API endpoints provide a fine level of control over the individual buckets in the cluster, their configuration, and specific operations.

Views REST API on page 475

XDCR REST API on page 483
Cross Datacenter Replication (XDCR) enables you to automatically replicate data between clusters and between data buckets.

Compaction REST API on page 494

Logs REST API on page 497

Users REST API on page 499

REST API overview

The Couchbase REST API enables you to manage a Couchbase Server deployment as well as perform operations such as storing design documents and querying for results. The REST API conforms to Representational State Transfer (REST) constraints, in other words, the REST API follows a **RESTful** architecture.

Use the REST API to manage clusters, server nodes, and buckets, and to retrieve run-time statistics within your Couchbase Server deployment. If you want to develop your own Couchbase-compatible SDK, you will also use the REST API within your library to handle views. Views enable you to index and query data based on functions that you define.

Tip: The REST API should **not** be used to read or write data to the server. Data operations, such as `set` and `get` for example, are handled by Couchbase SDKs.

In addition, the Couchbase Web Console uses many of the same REST API endpoints that are used for a REST API request. This is especially for administrative tasks such as creating a new bucket, adding a node to a cluster, or changing cluster settings.

Please provide RESTful requests; you will not receive any handling instructions, resource descriptions, nor should you presume any conventions for URI structure for resources represented. The URIs in the REST API may have a specific URI or may even appear as RPC or some other architectural style using HTTP operations and semantics.

In other words, build your request starting from Couchbase Cluster URIs, and be aware that URIs for resources may change from version to version. Also note that the hierarchies shown here enable your reuse of requests, since they follow a similar pattern for accessing different parts of the system.

The REST API is built on a number of basic principles:

- **JSON Responses**

  The Couchbase Management REST API returns many responses as JavaScript Object Notation (JSON). On that note, you may find it convenient to read responses in a JSON reader. Some responses may have an empty body, but indicate the response with standard HTTP codes. For more information, see RFC 4627 at www.json.org.
- **HTTP Basic Access Authentication**

  The Couchbase Management REST API uses HTTP basic authentication. The browser-based Web Console and Command-line Interface also use HTTP basic authentication.
- **Versatile Server Nodes**

  All server nodes in a cluster share the same properties and can handle any requests made via the REST API.; you can make a REST API request on any node in a cluster you want to access. If the server node cannot service a request directly, due to lack of access to state or some other information, it will forward the request to the appropriate server node, retrieve the results, and send the results back to the client.

In order to use the REST API you should be aware of the different terms and concepts discussed in the following sections.

Types of resources

There are a number of different resources within the Couchbase Server and these resources require a different URI/RESTful-endpoint in order to perform an operations:

- Server nodes

  A Couchbase Server instance, also known as ‘node’, is a physical or virtual machine running Couchbase Server. Each node is as a member of a cluster.
- Cluster/Pool

  A cluster is a group of one or more nodes; it is a collection of physical resources that are grouped together and provide services and a management interface. A single default cluster exists for every deployment of Couchbase Server. A node, or instance of Couchbase Server, is a member of a cluster. Couchbase Server collects run-time statistics for clusters, maintaining an overall pool-level data view of counters and periodic metrics of the overall system. The Couchbase Management REST API can be used to retrieve historic statistics for a cluster.

- Buckets

  A bucket is a logical grouping of data within a cluster. It provides a name space for all the related data in an application; therefore you can use the same key in two different buckets and they are treated as unique items by Couchbase Server.

  Couchbase Server collects run-time statistics for buckets, maintaining an overall bucket-level data view of counters and periodic metrics of the overall system. Buckets are categorized by storage type: 1) memcached buckets are for in-memory, RAM-based information, and 2) Couchbase buckets, which are for persisted data.
- Views

  Views enable you to index and query data based on logic you specify. You can also use views to perform calculations and aggregations, such as statistics, for items in Couchbase Server.
- Cross datacenter replication (XDCR)

  Cross Datacenter Replication (XDCR) is new functionality as of Couchbase Server 2.0. It enables you to automatically replicate data between clusters and between data buckets. There are two major benefits of using XDCR as part of your Couchbase Server implementation: 1) enables you to restore data from one Couchbase cluster to another cluster after system failure. 2) provide copies of data on clusters that are physically closer to your end users.

HTTP request headers

The following HTTP request headers are used to create your request:

Header	Supported Values	Description of Use	Required
Accept	Comma-delimited list of media types or media type patterns.	Indicates to the server what media type(s) this client is prepared to accept.	Recommended
Authorization	`Basic` plus username and password (per RFC 2617).	Identifies the authorized user making this request.	No, unless secured
Content-Length	Body Length (in bytes)	Describes the size of the message body.	Yes, on requests that contain a message body.
Content-Type	Content type	Describes the representation and syntax of the request message body.	Yes, on requests that contain a message body.
Host	Origin hostname	Required to allow support of multiple origin hosts at a single IP address.	All requests
X-YYYYY-Client-Specification-Version	String	Declares the specification version of the YYYYY API that this client was programmed against.	No

HTTP status codes

The Couchbase Server will return one of the following HTTP status codes in response to your REST API request:

HTTP Status	Description
200 OK	Successful request and an HTTP response body returns. If this creates a new resource with a URI, the 200 status

HTTP Status	Description
	will also have a location header containing the canonical URI for the newly created resource.
201 Created	Request to create a new resource is successful, but no HTTP response body returns. The URI for the newly created resource returns with the status code.
202 Accepted	The request is accepted for processing, but processing is not complete. Per HTTP/1.1, the response, if any, SHOULD include an indication of the request's current status, and either a pointer to a status monitor or some estimate of when the request will be fulfilled.
204 No Content	The server fulfilled the request, but does not need to return a response body.
400 Bad Request	The request could not be processed because it contains missing or invalid information, such as validation error on an input field, a missing required value, and so on.
401 Unauthorized	The credentials provided with this request are missing or invalid.
403 Forbidden	The server recognized the given credentials, but you do not possess proper access to perform this request.
404 Not Found	URI you provided in a request does not exist.
405 Method Not Allowed	The HTTP verb specified in the request (DELETE, GET, HEAD, POST, PUT) is not supported for this URI.
406 Not Acceptable	The resource identified by this request cannot create a response corresponding to one of the media types in the Accept header of the request.
409 Conflict	A create or update request could not be completed, because it would cause a conflict in the current state of the resources supported by the server. For example, an attempt to create a new resource with a unique identifier already assigned to some existing resource.
500 Internal Server Error	The server encountered an unexpected condition which prevented it from fulfilling the request.
501 Not Implemented	The server does not currently support the functionality required to fulfill the request.
503 Service Unavailable	The server is currently unable to handle the request due to temporary overloading or maintenance of the server.

Related Links

Couchbase REST API on page 425

Related topics

Cross Datacenter Replication (XDCR) on page 241

Views and indexes on page 183

couchbase-cli tool on page 361

Couchbase SDKs

http://www.ietf.org/rfc/rfc4627.txt

www.json.org

Cluster REST API

Cluster operations through the Couchbase REST API include managing server nodes, viewing cluster details, viewing cluster information, and managing auto-failover.

Related Links

Couchbase REST API on page 425
Retrieving cluster information on page 429
Viewing cluster details on page 430
Adding nodes to clusters on page 432
Joining nodes into clusters on page 432
Rebalancing nodes on page 433
Removing nodes from clusters on page 436
Managing internal cluster settings on page 436
Managing auto-failover on page 437
Disabling consistent query results on rebalance on page 439
View settings for email notifications on page 440
Setting maximum buckets for clusters on page 441
Setting maximum parallel indexers on page 442

Retrieving cluster information

One of the first ways to discover the URI endpoints for the REST API is to find the clusters available. For this you provide the Couchbase Server IP address, port number, and append '/pools'.

Example Request

```
curl -u admin:password http://localhost:8091/pools
```

Replace the *admin*, *password*, and *localhost* values in the above example with your actual values.

As a raw HTTP request:

```
GET /pools
Host: localhost:8091
Authorization: Basic xxxxxxxxxxxxxxxxxxx
Accept: application/json
X-memcachekv-Store-Client-Specification-Version: 0.1
```

The corresponding HTTP response contains a JSON document describing the cluster configuration:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: nnn


{"pools": [
            {
            "name":"default",
            "uri":"/pools/default",
            "streamingUri":"/poolsStreaming/default"
            }
          ],
          "isAdminCreds":false,
          "uuid":"c25913df-59a2-4886-858c-7119d42e36ab",
          "implementationVersion":"1.8.1-927-rel-enterprise",
          "componentsVersion":
             {
             "ale":"8cffe61",
```

```
                "os_mon":"2.2.6",
                "mnesia":"4.4.19",
                "inets":"5.6",
                "kernel":"2.14.4",
                "sasl":"2.1.9.4",
                "ns_server":"1.8.1-927-rel-enterprise",
                "stdlib":"1.17.4"}
 }
```

Couchbase Server returns only one cluster per group of systems and the cluster will typically have a default name.

Couchbase Server returns the build number for the server in `implementation_version`, the specifications supported are in the `componentsVersion`. While this node can only be a member of one cluster, there is flexibility which allows for any given node to be aware of other pools.

The Client-Specification-Version is optional in the request, but advised. It allows for implementations to adjust representation and state transitions to the client, if backward compatibility is desirable.

Related Links

Cluster REST API on page 429

Viewing cluster details

At the highest level, the response for this request describes a cluster, as mentioned previously. The response contains a number of properties which define attributes of the cluster and *controllers* which enable you to make certain requests of the cluster.

Warning: Since buckets could be renamed and there is no way to determine the name for the default bucket for a cluster, the system attempts to connect non-SASL, non-proxied to a bucket named "default". If it does not exist, Couchbase Server drops the connection.

Do not rely on the node list returned by this request to connect to a Couchbase Server. Instead, issue an HTTP Get call to the bucket to get the node list for that specific bucket.

```
GET /pools/default
Host: localhost:8091
Authorization: Basic xxxxxxxxxxxxxxxxxxx
Accept: application/json
X-memcachekv-Store-Client-Specification-Version: 0.1
```

```

HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: nnn
{
    "name":"default",
    "nodes":[{

        "hostname":"10.0.1.20",

        "status":"healthy",
        "uptime":"14",
        "version":"1.6.0",
        "os":"i386-apple-darwin9.8.0",
        "memoryTotal":3584844000.0,
        "memoryFree":74972000,
        "mcdMemoryReserved":64,
        "mcdMemoryAllocated":48,
        "ports":{

            "proxy":11213,
            "direct":11212
        },
        "otpNode":"ns_1@localhost",
        "otpCookie":"fsekryjfoeygvgcd",
```

```
        "clusterMembership":"active"
    }],
    "storageTotals":{

        "ram":{
            "total":2032558080,
            "used":1641816064

        },
        "hdd":{
            "total":239315349504.0,
            "used": 229742735523.0
        }
    },
    "buckets":{

        "uri":"/pools/default/buckets"
    },
    "controllers":{
        "ejectNode":{
            "uri":"/pools/default/controller/ejectNode"
        },
        "addNode":{


            "uri":"/controller/addNode"
        },
        "rebalance":{


            "uri":"/controller/rebalance"
        },
        "failover":{


            "uri":"/controller/failOver"
        },
        "reAddNode":{


            "uri":"/controller/reAddNode"
        },
        "stopRebalance":{


            "uri":"/controller/stopRebalance"
        }
    },
    "rebalanceProgress":{


        "uri":"/pools/default/rebalanceProgress"
    },
    "balanced": true,
    "etag":"asdas123",
    "initStatus":

    "stats":{
        "uri":"/pools/default/stats"
    }
}
```


The controllers in this list all accept parameters as `x-www-form-urlencoded`, and perform the following functions:

Function	Description
ejectNode	Eject a node from the cluster. Required parameter: “otpNode”, the node to be ejected.
addNode	Add a node to this cluster. Required parameters: “hostname”, “user” and “password”. Username and password are for the Administrator for this node.
rebalance	Rebalance the existing cluster. This controller requires both “knownNodes” and “ejectedNodes”. This allows a client to state the existing known nodes and which nodes should be removed from the cluster in a single operation. To ensure no cluster state changes have occurred since a client last got a list of nodes, both the known nodes and the node to be ejected must be supplied. If the list does not match the set of nodes, the request will fail with an HTTP 400 indicating a mismatch. Note rebalance progress is available via the rebalanceProgress uri.
failover	Failover the vBuckets from a given node to the nodes which have replicas of data for those vBuckets. The “otpNode” parameter is required and specifies the node to be failed over.
reAddNode	The “otpNode” parameter is required and specifies the node to be re-added.
stopRebalance	Stop any rebalance operation currently running. This takes no parameters.

Related Links

Cluster REST API on page 429

Adding nodes to clusters

This is a REST request made to a Couchbase cluster to add a given node to the cluster. You add a new node with the at the RESTful endpoint `server_ip:port/controller/addNode`. Provide an administrative username and password as parameters:

```
curl -u admin:password \
10.2.2.60:8091/controller/addNode \
-d "hostname=10.2.2.64&user=admin&password=password"
```

Here we create a request to the cluster at 10.2.2.60:8091 to add a given node by using method, `controller/addNode` and by providing the IP address for the node as well as credentials. Replace the *admin*, *password*, *10.2.2.60*, and *10.2.2.64* values in the above example with your actual values.

If successful, Couchbase Server will respond:

```
HTTP/1.1 200 OK
{"otpNode":"ns_1@10.4.2.6"}
```

Related Links

Cluster REST API on page 429

Joining nodes into clusters

This is a REST request made to an individual Couchbase node to add that node to a given cluster. You cannot merge two clusters together into a single cluster using the REST API, however, you can add a single node to an existing cluster. You will need to provide several parameters to add a node to a cluster.

The following arguments are required:

Argument	Description
clusterMemberHostIp	Hostname or IP address to a member of the cluster the node receiving this POST will be joining.
clusterMemberPort	Port number for the RESTful interface to the system. If your cluster requires credentials, provide the administrator username and password.

Curl example

```
curl -u admin:password -d clusterMemberHostIp=192.168.0.1 \
-d clusterMemberPort=8091 \
-d user=admin -d password=password
http://localhost:8091/node/controller/doJoinCluster
```

Replace the *admin*, *password*, and *192.168.0.1* values in the above example with your actual values.

HTTP request

```
POST /node/controller/doJoinCluster
Host: localhost:8091
Authorization: Basic xxxxxxxxxxxx
Accept: */*
Content-Length: xxxxxxxxxx
Content-Type: application/x-www-form-urlencoded
clusterMemberHostIp=192.168.0.1&clusterMemberPort=8091&user=admin&password=admin123
```

Response

```
200 OK with Location header pointing to pool details of pool just joined -
 successful join
400 Bad Request - missing parameters, etc.
401 Unauthorized - credentials required, but not supplied
403 Forbidden bad credentials - invalid credentials
```

Related Links

Cluster REST API on page 429

Rebalancing nodes

To start a rebalance process through the REST API you must supply two arguments containing the list of nodes that have been marked to be ejected, and the list of nodes that are known within the cluster.

Initiating a rebalance

You can obtain the information about ejected and known node by getting the current node configuration.. This is to ensure that the client making the REST API request is aware of the current cluster configuration. Nodes should have been previously added or marked for removal as appropriate.

The information must be supplied via the `ejectedNodes` and `knownNodes` parameters as a `POST` operation to the `/controller/rebalance` endpoint.

Example

```
curl -v -X -u admin:password POST 'http://192.168.0.77:8091/controller/
rebalance' \
-d 'ejectedNodes=&knownNodes=ns_1%40192.168.0.77%2Cns_1%40192.168.0.56'
```

Replace the *admin*, *password*, *192.168.0.77*, and *192.168.0.56* values in the above example with your actual values.

The corresponding raw HTTP request:

```
POST /controller/rebalance HTTP/1.1
Authorization: Basic QWRtaW5pc3RyYXRvcjpUYW1zaW4=
User-Agent: curl/7.24.0 (x86_64-apple-darwin12.0) libcurl/7.24.0
 OpenSSL/0.9.8r zlib/1.2.5
Host: 192.168.0.77:8091
Accept: */*
Content-Length: 63
Content-Type: application/x-www-form-urlencoded
```

The response will be 200 (OK) if the operation was successfully submitted.

If the wrong node information has been submitted, JSON with the mismatch error will be returned:

```
{"mismatch":1}
```

Getting rebalance progress

There are two endpoints for rebalance progress. One is a general request which outputs high-level percentage completion at `/pools/default/rebalanceProgress`. The second possible endpoint is one corresponds to the detailed rebalance report available in Web Console.

Example

This first request returns a JSON structure containing the current progress information:

```
curl -u admin:password '192.168.0.77:8091/pools/default/rebalanceProgress'
```

Replace the *admin*, *password*, *localhost*, and *192.168.0.77* values in the above example with your actual values.

As a pure REST API call, it appears as follows:

```
GET /pools/default/rebalanceProgress HTTP/1.1
Authorization: Basic QWRtaW5pc3RyYXRvcjpUYW1zaW4=
User-Agent: curl/7.24.0 (x86_64-apple-darwin12.0) libcurl/7.24.0
 OpenSSL/0.9.8r zlib/1.2.5
Host: 192.168.0.77:8091
Accept: */*
```

The response data packet contains a JSON structure showing the rebalance progress for each node. The progress figure is provided as a percentage (shown as a floating point value between 0 and 1).

```
{
    "status":"running",
    "ns_1@192.168.0.56":{"progress":0.2734375},
    "ns_1@192.168.0.77":{"progress":0.09114583333333337}
}
```

For more details about the rebalance, use the following request:

```
curl -u admin:password 'http://localhost:8091/pools/default/tasks'
```

Replace the *admin*, *password*, and *localhost* values in the above example with your actual values.

```
GET /pools/default/rebalanceProgress HTTP/1.1
Authorization: Basic QWRtaW5pc3RyYXRvcjpUYW1zaW4=
User-Agent: curl/7.24.0 (x86_64-apple-darwin12.0) libcurl/7.24.0
 OpenSSL/0.9.8r zlib/1.2.5
Host: 192.168.0.77:8091
Accept: */*
```

The response data packet contains a JSON structure showing detailed progress:

```
{
  type: "rebalance",
  recommendedRefreshPeriod: 0.25,
  status: "running",
  progress: 9.049479166666668,
  perNode: {
```

```
    ns_1@10.3.3.61: {
      progress: 13.4765625
    },
    ns_1@10.3.2.55: {
      progress: 4.6223958333333375
    }
  },
  detailedProgress: {
    bucket: "default",
    bucketNumber: 1,
    bucketsCount: 1,
    perNode: {
      ns_1@10.3.3.61: {
        ingoing: {
          docsTotal: 0,
          docsTransferred: 0,
          activeVBucketsLeft: 0,
          replicaVBucketsLeft: 0
        },
        outgoing: {
          docsTotal: 512,
          docsTransferred: 69,
          activeVBucketsLeft: 443,
          replicaVBucketsLeft: 511
        }
      },
      ns_1@10.3.2.55: {
        ingoing: {
          docsTotal: 512,
          docsTransferred: 69,
          activeVBucketsLeft: 443,
          replicaVBucketsLeft: 0
        },
        outgoing: {
          docsTotal: 0,
          docsTransferred: 0,
          activeVBucketsLeft: 0,
          replicaVBucketsLeft: 443
        }
      }
    }
  }
}
```


This shows percentage complete for each individual node undergoing rebalance. For each specific node, it provides the current number of docs transferred and other items. For details and definitions of these items.

If you rebalance fails, you will see this response:


```
[
  {
    "type": "rebalance",
    "status": "notRunning",
    "errorMessage": "Rebalance failed. See logs for detailed reason. You can
 try rebalance again."
  }
]
```


Adjusting rebalance during compaction

If you perform a rebalance while a node is undergoing index compaction, you may experience delays in rebalance. The REST API parameter, `rebalanceMovesBeforeCompaction`, is used to improve rebalance performance. If you do make this selection, you will reduce the performance of index compaction which can result in larger index file size.

Example

To make this request:

```
curl -X POST -u admin:password 'http://localhost:8091/internalSettings'
    -d 'rebalanceMovesBeforeCompaction=256'
```

Replace the *admin*, *password*, *localhost*, and *256* values in the above example with your actual values.

This needs to be made as POST request to the `/internalSettings` endpoint. By default this setting is 64 which specifies the number of vBuckets which will moved per node until all vBucket movements pauses. After this pause the system triggers index compaction. Index compaction will not be performed while vBuckets are being moved, so if you specify a larger value, it means that the server will spend less time compacting the index, which will result in larger index files that take up more disk space.

Related Links

Cluster REST API on page 429

Related topics

Monitoring a rebalance

Server nodes REST API on page 444
A Couchbase Server instance, known as node, is a physical or virtual machine running Couchbase Server. Each node is as a member of a cluster.

Provisioning nodes on page 446
Provisioning refers to creating a new cluster or adding a node to a cluster.

Rebalancing on page 158

Removing nodes from clusters

When a node is temporarily or permanently down, you may want to remove it from a cluster:

```
curl -u admin:password -d otpNode=ns_1@192.168.0.107 \
http://192.168.0.106:8091/controller/ejectNode
```

Replace the *admin*, *password*, *192.168.0.107*, and *192.168.0.106* values in the above example with your actual values.

```
POST /controller/ejectNode
Host: localhost:8091
Authorization: Basic xxxxxxxxxxxx
Accept: */*
Content-Length: xxxxxxxxxx
Content-Type: application/x-www-form-urlencoded
otpNode=ns_1@192.168.0.1
```

```
200 OK - node ejected
400 Error, the node to be ejected does not exist
401 Unauthorized - Credentials were not supplied and are required
403 Forbidden - Credentials were supplied and are incorrect
```

Related Links

Cluster REST API on page 429

Managing internal cluster settings

You can set a number of internal settings the number of maximum number of supported buckets supported by the cluster. To get the current setting of the number of parallel indexers, use a `GET` request.

Get Internal Settings	Description
Method	`GET /internalSettings`
Request Data	None
Response Data	JSON of current internal settings

Get Internal Settings	Description
Authentication Required	no
Return Codes	
200	Settings returned

For example:

```
GET http://127.0.0.1:8091/internalSettings
```

This returns a JSON structure of the current settings:

```
{
  "indexAwareRebalanceDisabled":false,
  "rebalanceIndexWaitingDisabled":false,
  "rebalanceIndexPausingDisabled":false,
  "maxParallelIndexers":4,
  "maxParallelReplicaIndexers":2,
  "maxBucketCount":20
}
```

To set a configuration value, `POST` to the URL a payload containing the updated values.

Set Configuration Value	Description
Method	`POST /settings/maxParallelIndexers`
Request Data	None
Response Data	JSON of the global and node-specific parallel indexer configuration
Authentication Required	yes
Payload Arguments	
`globalValue`	Required parameter. Numeric. Sets the global number of parallel indexers. Minimum of 1, maximum 1024.
Return Codes	
400	globalValue not specified or invalid

For example, to update the maximum number of buckets:

```
curl -v -X POST http://admin:password@localhost:8091/internalSettings \
-d maxBucketCount=20
```

Replace the *admin*, *password*, *localhost*, and *20* values in the above example with your actual values.

Related Links

Cluster REST API on page 429

Managing auto-failover

This section provides information about retrieving, enabling, disabling and resetting auto-failover.

Related Links

Cluster REST API on page 429
Retrieving auto-failover settings on page 438
Enabling and disabling auto-failover on page 438
Resetting auto-failover on page 439

Retrieving auto-failover settings

Use this request to retrieve any auto-failover settings for a cluster. Auto-failover is a global setting for all clusters. You need to be authenticated to read this value. Example:

```
curl -u admin:password http://localhost:8091/settings/autoFailover
```

Replace the *admin*, *password*, and *localhost* values in the above example with your actual values.

If successful Couchbase Server returns any auto-failover settings for the cluster:

```
{"enabled":false,"timeout":30,"count":0}
```

The following parameters and settings appear:

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

Possible errors include:

```
HTTP/1.1 401 Unauthorized
This endpoint isn't available yet.
```

```
GET /settings/autoFailover HTTP/1.1
Host: localhost:8091
Authorization: Basic YWRtaW46YWRtaW4=
Accept: */*
```

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: nnn
{ "enabled": false, "timeout": 30, "count": 0 }
```

Related Links

Managing auto-failover on page 437

Enabling and disabling auto-failover

This is a global setting you apply to all clusters. You need to be authenticated to change this value. An example of this request:

```
curl "http://localhost:8091/settings/autoFailover" \
    -i -u admin:password -d 'enabled=true&timeout=600'
```

Replace the *admin*, *password*, *localhost*, and *600* values in the above example with your actual values.

Possible parameters are:

- `enabled` (true|false) (required): Indicates whether Couchbase Server will perform auto-failover for the cluster or not.
- `timeout` (integer that is greater than or equal to 30) (required; optional when enabled=false): The number of seconds a node must be down before Couchbase Server performs auto-failover on the node.

```
POST /settings/autoFailover HTTP/1.1
Host: localhost:8091
Content-Type: application/x-www-form-urlencoded
Authorization: Basic YWRtaW46YWRtaW4=
Content-Length: 14
enabled=true&timeout=60
```

```
HTTP/1.1 200 OK
```

The possible errors include:

```
400 Bad Request, The value of "enabled" must be true or false.
```

```
400 Bad Request, The value of "timeout" must be a positive integer bigger or
 equal to 30.
401 Unauthorized
This endpoint isn't available yet.
```

Related Links

Managing auto-failover on page 437

Resetting auto-failover

This resets the number of nodes that Couchbase Server has automatically failed-over. You can send a request to set the auto-failover number to 0. This is a global setting for all clusters. You need to be authenticated to change this value. No parameters are required:

```
curl -X POST -i -u admin:password \
    http://localhost:8091/settings/autoFailover/resetCount
```

Replace the *admin*, *password*, and *localhost* values in the above example with your actual values.

```
POST /settings/autoFailover/resetCount HTTP/1.1
Host: localhost:8091
Content-Type: application/x-www-form-urlencoded
Authorization: Basic YWRtaW46YWRtaW4=
```

```
HTTP/1.1 200 OK
```

Possible errors include:

```
This endpoint isn't available yet.
401 Unauthorized
```

Related Links

Managing auto-failover on page 437

Disabling consistent query results on rebalance

If you perform queries during rebalance, this new feature will ensure that you receive the query results that you would expect from a node as if it is not being rebalanced. During node rebalance, you will get the same results you would get as if the data were on an original node and as if data were not being moved from one node to another. In other words, this new feature ensures you get query results from a new node during rebalance that are consistent with the query results you would have received from the node before rebalance started.

By default this functionality is enabled; although it is possible to disable this functionality via the REST API, under certain circumstances described below.

Note: Be aware that rebalance may take significantly more time if you have implemented views for indexing and querying. While this functionality is enabled by default, if rebalance time becomes a critical factor for your application, you can disable this feature via the REST API.

We do not recommend you disable this functionality for applications in production without thorough testing. To do so may lead to unpredictable query results during rebalance.

To disable this feature, provide a request similar to the following:

```
curl -v -u admin:password -X POST http://localhost:8091/internalSettings \
-d indexAwareRebalanceDisabled=true
```

Replace the *admin*, *password*, and *localhost* values in the above example with your actual values.

If successful Couchbase Server will send a response:

```
HTTP/1.1 200 OK
Content-Type: application/json
```

For more information about views and how they function within a cluster.

Related Links

Cluster REST API on page 429

Related topics

Views operation on page 184

View settings for email notifications

The response to this request will specify whether you have email alerts set, and which events will trigger emails. This is a global setting for all clusters. You need to be authenticated to read this value:

```
curl -u admin:password http://localhost:8091/settings/alerts
```

Replace the *admin*, *password*, and *localhost* values in the above example with your actual values.

```
GET /settings/alerts HTTP/1.1
Host: localhost:8091
Authorization: Basic YWRtaW46YWRtaW4= Accept: */*
```

```
{
    "recipients": ["root@localhost"],
    "sender":"couchbase@localhost",
    "enabled":true,
    "emailServer":
{"user":"","pass":"","host":"localhost","port":25,"encrypt":false},
    "alerts":
      ["auto_failover_node",
      "auto_failover_maximum_reached",
      "auto_failover_other_nodes_down",
      "auto_failover_cluster_too_small"]
  }
```

Possible errors include:

```
This endpoint isn't available yet.
```

Enabling and disabling email notifications

This is a global setting for all clusters. You need to be authenticated to change this value. If this is enabled, Couchbase Server sends an email when certain events occur. Only events related to auto-failover trigger notification:

```
curl -i -u admin:password \
    -d
 'enabled=true&sender=couchbase@localhost&recipients=admin@localhost,membi@localhost&ema
 http://localhost:8091/settings/alerts
```

Replace the *admin*, *password*, *localhost*, *couchbase@localhost*, *admin@localhost*, *membi@localhost*, *25*, and *false* values in the above example with your actual values.

Possible parameters include:

- `enabled` : (true|false) (required). Whether to enable or disable email notifications
- `sender` (string) (optional, default: couchbase@localhost). Email address of the sender.
- `recipients` (string) (required). A comma separated list of recipients of the of the emails.
- `emailHost` (string) (optional, default: localhost). Host address of the SMTP server
- `emailPort` (integer) (optional, default: 25). Port of the SMTP server
- `emailEncrypt` (true|false) (optional, default: false). Whether you want to use TLS or not
- `emailUser` (string) (optional, default: ""): Username for the SMTP server
- `emailPass` (string) (optional, default: ""): Password for the SMTP server
- `alerts` (string) (optional, default: auto_failover_node, auto_failover_maximum_reached, auto_failover_other_nodes_down, auto_failover_cluster_too_small). Comma separated list of alerts that should cause an email to be sent. Possible values are: auto_failover_node, auto_failover_maximum_reached, auto_failover_other_nodes_down, auto_failover_cluster_too_small.

```
POST /settings/alerts HTTP/1.1
```

```
Host: localhost:8091
Content-Type: application/x-www-form-urlencoded
Authorization: Basic YWRtaW46YWRtaW4=
Content-Length: 14
 enabled=true&sender=couchbase@localhost&recipients=admin@localhost,membi@localhost&emailHost=localhost&e
```

```
HTTP/1.1 200 OK
```

Possible HTTP errors include:

```
400 Bad Request
401 Unauthorized
JSON object ({"errors": {"key": "error"}}) with errors.
```

- Possible errors returned in a JSON document include:
- alerts: alerts contained invalid keys. Valid keys are: [list_of_keys].
- email_encrypt: emailEncrypt must be either true or false.
- email_port: emailPort must be a positive integer less than 65536.
- enabled: enabled must be either true or false.
- recipients: recipients must be a comma separated list of valid email addresses.
- sender: sender must be a valid email address.
- general: No valid parameters given.

Sending test emails

This is a global setting for all clusters. You need to be authenticated to change this value. In response to this request, Couchbase Server sends a test email with the current configurations. This request uses the same parameters used in setting alerts and additionally an email subject and body.

```
    curl -i -u admin:password http://localhost:8091/settings/alerts/testEmail
 \
    -d 'subject=Test+email+from+Couchbase& \
    body=This+email+was+sent+to+you+to+test+the+email+alert+email+server
+settings.&enabled=true& \
    recipients=vmx%40localhost&sender=couchbase%40localhost& \
    emailUser=&emailPass=&emailHost=localhost&emailPort=25&emailEncrypt=false&
 \
    alerts=auto_failover_node%2Cauto_failover_maximum_reached
%2Cauto_failover_other_nodes_down%2Cauto_failover_cluster_too_small'
```

Replace the *admin*, *password*, *localhost*, *vmx%40localhost*, *couchbase%40localhost*, *25*, and *false* values in the above example with your actual values.

```
POST /settings/alerts/sendTestEmail HTTP/1.1
Host: localhost:8091
Content-Type: application/x-www-form-urlencoded
Authorization: Basic YWRtaW46YWRtaW4=
```

200 OK

Possible errors include:

```
400 Bad Request: Unknown macro: {"error"} 401 Unauthorized
This endpoint isn't available yet.
```

Related Links

Cluster REST API on page 429

Setting maximum buckets for clusters

By default the maximum number of buckets recommended for a Couchbase Cluster is ten. This is a safety mechanism to ensure that a cluster does not have resource and CPU overuse due to too many buckets. This limit is configurable using the REST API.

The Couchbase REST API has changed to enable you to change the default maximum number of buckets used in a Couchbase cluster. The maximum allowed buckets in this request is 128, however the suggested maximum number of buckets is ten per cluster. The following illustrates the endpoint and parameters used:

```
curl -X POST -u admin:password -d maxBucketCount=6 http://localhost:8091/
internalSettings
```

Replace the *admin*, *password*, *localhost*, and *6* values in the above example with your actual values.

For this request you need to provide administrative credentials for the cluster. The following HTTP request will be sent:

```
About to connect() to 127.0.0.1 port 8091 (#0)
Trying 127.0.0.1...
connected
Connected to 127.0.0.1 (127.0.0.1) port 8091 (#0)
Server auth using Basic with user 'Administrator'
POST /internalSettings HTTP/1.1
```

If Couchbase Server successfully changes the bucket limit for the cluster, you will get a HTTP 200 response:

```
HTTP/1.1 200 OK
Server: Couchbase Server 2.0.0r_501_gb614829
Pragma: no-cache
Date: Wed, 31 Oct 2012 21:21:48 GMT
Content-Type: application/json
Content-Length: 2
Cache-Control: no-cache
```

If you provide an invalid number, such as 0, a negative number, or an amount over 128 buckets, you will get this error message:

```
["Unexpected server error, request logged."]
```

Related Links

Cluster REST API on page 429

Setting maximum parallel indexers

You can set the number of parallel indexers that will be used on each node when view indexes are updated. To get the current setting of the number of parallel indexers, use a `GET` request.

Get Maximum Parallel Indexers	Description
Method	`GET /settings/maxParallelIndexers`
Request Data	None
Response Data	JSON of the global and node-specific parallel indexer configuration
Authentication Required	no

For example:

```
GET http://127.0.0.1:8091/settings/maxParallelIndexers
```

This returns a JSON structure of the current settings, providing both the globally configured value, and individual node configuration:

```
{
   "globalValue" : 4,
   "nodes" : {
      "ns_1@127.0.0.1" : 4
   }
}
```

To set the value, `POST` to the URL specifying a URL-encoded value to the `globalValue` argument.

Set Maximum Parallel Indexers	Description
Method	`POST /settings/maxParallelIndexers`
Request Data	None
Response Data	JSON of the global and node-specific parallel indexer configuration
Authentication Required	yes
Payload Arguments	
`globalValue`	Required parameter. Numeric. Sets the global number of parallel indexers. Minimum of 1, maximum 1024.
Return Codes	
400	globalValue not specified or invalid

Related Links

Cluster REST API on page 429

Server nodes REST API

A Couchbase Server instance, known as node, is a physical or virtual machine running Couchbase Server. Each node is as a member of a cluster.

HTTP method and URI

```
GET /pools/nodes
```

Curl syntax and example

```
curl -u admin:password http://localhost:port/pools/nodes
```

To view information about nodes that exist in a Couchbase Cluster, you use this request:

```
curl -u admin:password http://10.5.2.118:8091/pools/nodes
```

Response

Couchbase server returns this response in JSON:


```
      {"storageTotals":
                    {
                    "ram":
                    {
                    "quotaUsed":10246684672.0,
                    "usedByData":68584936,
                    "total":12396216320.0,
                    "quotaTotal":10246684672.0,
                    "used":4347842560.0},
                    "hdd":
                          {"usedByData":2560504,
                          "total":112654917632.0,
                          "quotaTotal":112654917632.0,
                          "used":10138942586.0,
                          "free":102515975046.0}
                     },
                     "name":"nodes",
                     "alerts":[],
                     "alertsSilenceURL":"/controller/resetAlerts?token=0",
                     "nodes":
                            [{"systemStats":
                                      {
                                      "cpu_utilization_rate":2.5,
                                      "swap_total":6140452864.0,
                                      "swap_used":0
                                      },
                             "interestingStats":
                                      {
                                      "curr_items":0,
                                      "curr_items_tot":0,
                                      "vb_replica_curr_items":0
                                      },
                              "uptime":"5782",
                              "memoryTotal":6198108160.0,
                              "memoryFree":3777110016.0,
                              "mcdMemoryReserved":4728,
                              "mcdMemoryAllocated":4728,
                              "clusterMembership":"active",
                              "status":"healthy",
                              "hostname":"10.4.2.5:8091",
                              "clusterCompatibility":1,
```

```
                              "version":"1.8.1-937-rel-community",
                              "os":"x86_64-unknown-linux-gnu",
                              "ports":
                                      {
                                      "proxy":11211,
                                      "direct":11210
                                      }
                            .......

                                  }],
                       "buckets":
                                {"uri":"/pools/nodes/buckets?v=80502896" },
                                "controllers":{"addNode":{"uri":"/controller/
addNode"},
                                "rebalance":{"uri":"/controller/rebalance"},
                                "failOver":{"uri":"/controller/failOver"},
                                "reAddNode":{"uri":"/controller/reAddNode"},
                                "ejectNode":{"uri":"/controller/ejectNode"},
                                "testWorkload":{"uri":"/pools/nodes/
controller/testWorkload"}},
                                "balanced":true,
                                "failoverWarnings":
["failoverNeeded","softNodesNeeded"],
                                "rebalanceStatus":"none",
                                "rebalanceProgressUri":"/pools/nodes/
rebalanceProgress",
                                "stopRebalanceUri":"/controller/
stopRebalance",
                                "nodeStatusesUri":"/nodeStatuses",
                                "stats":{"uri":"/pools/nodes/stats"},
                                "counters":
{"rebalance_success":1,"rebalance_start":1},
                                "stopRebalanceIsSafe":true}
```

Related Links

Couchbase REST API on page 425

Provisioning nodes on page 446
Provisioning refers to creating a new cluster or adding a node to a cluster.

Failing over nodes on page 446
Failing over a node indicates that the node is no longer available in a cluster and replicated data on another node should be made available to clients.

Setting hostnames on page 447
Hostnames must be specified prior to being added to a cluster.

Setting usernames and passwords on page 447
Usernames and passwords can be set at any time, however, it is typically the last step twhen adding a node into a new cluster.

Setting memory quota on page 448
The memory quota configures how much RAM to be allocated to Couchbase for every node within the cluster.

Setting index paths on page 449
The path for the index files can be configured with `POST /nodes/self/controller/settings`.

Retrieving statistics on page 450
To retrieve statistics for a node, first retrieve a list of nodes

Related topics

Cluster REST API on page 429

Using hostnames on page 69

Provisioning nodes

Provisioning refers to creating a new cluster or adding a node to a cluster.

Process for provisioning a node.

To provision a node:

- Create a new node by installing a new Couchbase Server.
- Configure disk path for the node.
- Optionally configure memory quota for each node within the cluster.

  Any nodes you add to a cluster will inherit the configured memory quota. The default memory quota for the first node in a cluster is 60% of the physical RAM.
- Add the node to your existing cluster.

Whether you are adding a node to an existing cluster or starting a new cluster, the node's disk path must be configured. Your next steps depends on whether you create a new cluster or add a node to an existing cluster. If you create a new cluster, secure it by providing an administrative username and password. If you add a node to an existing cluster, obtain the URI and credentials to use the REST API with that cluster.

Related Links

Server nodes REST API on page 444
A Couchbase Server instance, known as node, is a physical or virtual machine running Couchbase Server. Each node is as a member of a cluster.

Failing over nodes

Failing over a node indicates that the node is no longer available in a cluster and replicated data on another node should be made available to clients.

HTTP method and URI

This endpoint along with the `otpNode` parameter (internal node name) allows you to failover a specific node.

```
POST /controller/failOver
```

HTTP request

The HTTP request is similar to the following:

```
        POST /controller/failOver HTTP/1.1
        Authorization: Basic
```

Curl syntax and example

```
curl -v -X POST
        -u admin:password http://localhost:port/controller/failOver -d
 otpNode=node@hostname
```

The following examples fails over server node `10.3.3.63`.

```
curl -v -X POST
        -u admin:password http://localhost:8091/controller/failOver -d
 otpNode=ns_2@10.3.3.63
```

Response codes

Response codes	Description
200	OK

Response codes	Description
400	Bad Request JSON: The RAM Quota value is too small.
401	Unauthorized

The following example is a successful response:

```
HTTP/1.1 200 OK
```

The following example is an unsuccessful response, for example, if the node that you want to failover does not exist in the cluster.

```
HTTP/1.1 400
```

Related Links

Server nodes REST API on page 444
A Couchbase Server instance, known as node, is a physical or virtual machine running Couchbase Server. Each node is as a member of a cluster.

Setting hostnames

Hostnames must be specified prior to being added to a cluster.

A hostnames are provided when you install a Couchbase Server node, when you add it to an existing cluster for online upgrade, or via a REST API call. If a node restarts, any hostname that you established is used. A hostname cannot be provided for a node that is already part of a Couchbase cluster.

Response

If you attempt to specify a hostname once the node is in the cluster, the server rejects the request and returns the following:

```
error 400 reason: unknown ["Renaming is disallowed
        for nodes that are already part of a cluster"]
```

Related Links

Server nodes REST API on page 444
A Couchbase Server instance, known as node, is a physical or virtual machine running Couchbase Server. Each node is as a member of a cluster.

Setting usernames and passwords

Usernames and passwords can be set at any time, however, it is typically the last step twhen adding a node into a new cluster.

HTTP method and URI

The response indicates the new base URI if the parameters are valid. Clients generally send a new request for cluster information based on this response.

```
      POST /settings/web
```

HTTP request example

The raw HTTP request:

```
POST /settings/web HTTP/1.1
Host: localhost:8091
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Authorization: Basic YWRtaW46YWRtaW4=
Content-Length: xx
username=Administrator&password=letmein&port=8091
```

HTTP response

The corresponding HTTP response data:

```
HTTP/1.1 200 OK
Content-Type: application/json
Server: Couchbase Server 2.0
Pragma: no-cache
Date: Mon, 09 Aug 2010 18:50:00 GMT
Content-Type: application/json
Content-Length: 39
Cache-Control: no-cache no-store max-age=0
{"newBaseUri":"http://localhost:8091/"}
```

Note: The port number must be specified when username/password is updated.

Curl example

For example, using `curl`:

```
curl -u admin:password -d username=Administrator \
-d password=letmein \
-d port=8091 \
http://localhost:8091/settings/web
```

Related Links

Server nodes REST API on page 444
A Couchbase Server instance, known as node, is a physical or virtual machine running Couchbase Server. Each node is as a member of a cluster.

Setting memory quota

The memory quota configures how much RAM to be allocated to Couchbase for every node within the cluster.

HTTP method and URI

```
POST /pools/default
```

Set Memory	Description
Method	`POST /pools/default`
Request Data	Payload with memory quota setting
Response Data	Empty
Authentication Required	yes

HTTP request example

As a raw HTTP request:

```
POST /pools/default HTTP/1.1
Host: localhost:8091
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Authorization: Basic YWRtaW46YWRtaW4=
Content-Length: xx
memoryQuota=400
```

Curl syntax and example

```
curl -X POST -u admin:password -d memoryQuota=quotaNumber
      http://localhost:port/pools/default
```

The following example sets the memory quota for a cluster at 400MB:

```
curl -X POST -u admin:password -d memoryQuota=400 http://localhost:8091/pools/
default
```

Response codes

Response codes	Description
200	OK
400	Bad Request JSON: The RAM Quota value is too small.
401	Unauthorized

The following is an example HTTP response code:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 0
```

Related Links

Server nodes REST API on page 444
A Couchbase Server instance, known as node, is a physical or virtual machine running Couchbase Server. Each node is as a member of a cluster.

Setting index paths

The path for the index files can be configured with `POST /nodes/self/controller/settings`.

HTTP method and URI

The path for the index files can be configured with the `index_path`.

```
        POST /nodes/self/controller/settings
```

HTTP request

As a raw HTTP request:

```
POST /nodes/self/controller/settings HTTP/1.1
Host: localhost:8091
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Authorization: Basic YWRtaW46YWRtaW4=
Content-Length: xx path=/var/tmp/test
```

HTTP response

The HTTP response contains the response code and optional error message:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 0
```

Note: If you try to set the data path at this endpoint, you will receive the following error:

```
ERROR: unable to init 10.3.4.23 (400) Bad Request
{u'error': u'Changing data of nodes that are part of provisioned cluster
 is not supported'}
```

Curl example

```
curl -X POST -u admin:password \
    -d index_path=/var/tmp/text-index \
    http://localhost:8091/nodes/self/controller/settings
```

Related Links

Server nodes REST API on page 444
A Couchbase Server instance, known as node, is a physical or virtual machine running Couchbase Server. Each node is as a member of a cluster.

Retrieving statistics

To retrieve statistics for a node, first retrieve a list of nodes

HTTP method and URI

```
        GET /pools/default/buckets/default/nodes/host:port/stats
```

To obtain statistics for a node:

1. Retrieve a list of nodes in a cluster.
2. Send the statistics request using the IP address and port for a node in the cluster.

HTTP request

The curl request for a node list sends the following HTTP request:

```
GET /pools/default/buckets/default/nodes HTTP/1.1
User-Agent: curl/7.21.4 (universal-apple-darwin11.0) libcurl/7.21.4
 OpenSSL/0.9.8r zlib/1.2.5
Host: 10.5.2.118:8091
Accept: */*
```

The curl request for a node's statistics sends the following HTTP request:

```
GET /pools/default/buckets/default/nodes/10.4.2.4%3A8091/stats HTTP/1.1
User-Agent: curl/7.21.4 (universal-apple-darwin11.0) libcurl/7.21.4
 OpenSSL/0.9.8r zlib/1.2.5
Host: 10.4.2.4:8091
Accept: */*
```

Curl syntax

To retrieve a list of nodes:

```
curl -u admin:password http://localhost:port/pools/default/buckets/default/
nodes
```

To retrieve statistics about a node:

```
curl -u admin:password http://host:port/pools/default/buckets/default/nodes/
localhost%3Aport/stats
```

Curl example

The following example retrieves a list of nodes from a cluster:

```
curl -u admin:password http://10.5.2.118:8091/pools/default/buckets/default/
nodes
```

If Couchbase Server successfully handles the request, a response similar to the following displays:

```
{"servers":[
```

```
  {"hostname":"10.5.2.118:8091",
  "uri":"/pools/default/buckets/default/nodes/10.5.2.118%3A8091",
  "stats":
      {"uri":"/pools/default/buckets/default/nodes/10.5.2.118%3A8091/stats"}}
    ....
```

You can then make a REST request to the specific IP address and port of given node shown in the response and add `/stats` as the endpoint:

```
curl -u admin:password http://10.5.2.118:8091/pools/default/buckets/default/
nodes/10.5.2.118%3A8091/stats
```

Response

The following statistics returned are for the individual bucket associated with that node.

```
{"hostname":"10.5.2.118:8091","hot_keys":[{"name":"[2012-11-05::3:47:01]"
....
"samplesCount":60,"isPersistent":true,"lastTStamp":1352922180718,"interval":1000}}
```

Related Links

Server nodes REST API on page 444

A Couchbase Server instance, known as node, is a physical or virtual machine running Couchbase Server. Each node is as a member of a cluster.

Rack Awareness REST API

The Rack Awareness feature allows logical groupings of servers on a cluster where each server group physically belongs to a rack or availability zone.

This feature provides the ability to specify that active and corresponding replica partitions be created on servers that are part of a separate rack or zone. For purposes of the server group REST API, racks or availability zones are represented as flat space of server groups with group names. To enable Rack Awareness, all servers in a cluster must be upgraded to use the Rack Awareness feature.

Note: The Rack Awareness feature with its server group capability is an Enterprise Edition feature.

The Server groups REST API provides the following capability:

- Creates server groups
- Edits server groups
- Deletes server groups
- Assigns servers to server groups.

HTTP Methods

The following summarizes the HTTP methods used for managing server groups.

HTTP method	URI path	Description
GET	/pools/default/serverGroups	Retrieves information about a server group.
POST	/pools/default/serverGroups	Creates a server group with a specific name.
PUT	/pools/default/serverGroups/<:uuid>	Updates the server group information.
PUT	/pools/default/serverGroups?rev=<:number>	Updates a server's group memberships.
DELETE	/pools/default/serverGroups/<:uuid>	Deletes a specific server group.

Retrieving server group information

GET /pools/default/serverGroups retrieves information about server groups. Provides group information, "groups": [(<groupInfo>)+], where each server group has unique URIs and UUIDs.

Syntax

```
curl -X GET -u <administrator>:<password>
   http://<host>:<port>/pools/default/serverGroups
```

Example

```
curl -X GET -u Admin:myPassword
   http://192.168.0.1:8091/pools/default/serverGroups
```

Returns

```
{"groups":
   [
      {
      "name":"<groupName>",
      "uri": "/pools/default/serverGroups?rev=<integer>",
      "addNodeURI":"/pools/default/serverGroups/0",
      "nodes":[(<nodeInfo>)+]
      }
```

```
    ]
}
```

Group info	Description
`"groups": [(<groupInfo>)+]`	Information about server groups.
`"name":"<groupName>"`	Specifies the name of the group. If the group name has a space, for example, Group A, use double quotes (for example, `"Group A"`). If the name does not have spaces (for example, GroupA) double quotes are not required.
`"uri":"/pools/default/serverGroups?rev=<integer>"`	Specifies the URI path and revision integer.
`"uri":"/pools/default/serverGroups/<:uuid>"`	Specifies the URI path and UUID string.
`"addNodeURI":"/pools/default/serverGroups/<:uuid>/addNode`	Specifies the URI path and UUID string for adding servers to a server group.
`"nodes": [(<nodeInfo>+)]`	Information about the servers.

Creating server groups

`POST /pools/default/serverGroups` creates a server group with a specific name. In the following example, Group A is created. If the group name has a space, for example, Group A, use double quotes; for example, `"Group A"`.

Syntax

```
curl -X POST -u <administrator>:<password>
  http://<host>:<port>/pools/default/serverGroups
  -d name="<groupName>"
```

Example

```
curl -X POST -u Admin:myPassword
  http://192.168.0.1:8091/pools/default/serverGroups
  -d name="Group A"
```

Renaming server groups

`PUT /pools/default/serverGroups/<:uuid>` renames the server group. Find the UUID for the server group by using GET, add the UUID to the URI path, and specify a new group name. In this example, Group A is renamed to Group B. The UUID for the server group is located in the full URI information for that server group. The UUID remains the same for the server group after changing the name.

For example, the UUID for Group A is located in the following group information:

```
"name":"Group A",
"uri":"/pools/default/serverGroups/246b5de857e100dbfd8b6dee0406420a"
```

Syntax

```
curl -X PUT -u <administrator>:<password>
  http://<host>:<port>/pools/default/serverGroups/<uuid>
  -d name="<newGroupName>"
```

Example

```
curl -X PUT -u Admin:myPassword
  http://192.168.0.1:8091/pools/default/
serverGroups/246b5de857e100dbfd8b6dee0406420a
  -d name="Group B"
```

Deleting server groups

`DELETE /pools/default/serverGroups/<:uuid>` deletes a specific server group. The server group must be empty for a successful request. In the following example, the UUID is the same UUID used in the renaming example.

Syntax

```
curl -X DELETE -u <administrator>:<password>
  http://<host>:<port>/pools/default/serverGroups/<uuid>
```

Example

```
curl -X DELETE -u Admin:myPassword
  http://192.168.0.1:8091/pools/default/
serverGroups/246b5de857e100dbfd8b6dee0406420a
```

Adding servers to server groups

`POST /pools/default/serverGroups/<:uuid>/addNode` adds a server to a cluster and assigns it to the specified server group.

Syntax

```
curl -X POST -dhostname=<host>:<port>
  -u <administrator>:<password>
  http://<host>:<port>/pools/default/serverGroups/<uuid>/addNode
```

Example

```
curl -X POST -dhostname=192.168.0.2:8091
  -u Admin:myPassword
  http://192.168.0.1:8091/pools/default/
serverGroups/246b5de857e100dbfd8b6dee0406420a/addNode
```

The server group's UUID is in the group information

```
"name":"Group 2",
"uri":"/pools/default/serverGroups/d55339548767ceb51b241c61e3b9f036",
"addNodeURI":"/pools/default/serverGroups/d55339548767ceb51b241c61e3b9f036/
addNode",
```

Updating server group memberships

`PUT /pools/default/serverGroups?rev=<:number>` updates the server's group memberships. In the following examples, the group name is optional. If the group name is provided, it *must* match the current group name. All servers must be mentioned and *all* groups must be mentioned. The URI is used to identify the group.

This request only allows moving servers between server groups. It does not allow server group renaming or removal. In this example, the servers for Group 2 are moved to Group 1.

The following is the group information that is needed to update the server and server group memberships:

```
{
   "groups": [( { ("name": <groupName:string>,)?
   "uri": "/pools/default/serverGroups/"<uuid>,
   "nodes": [(<otpNode>)*]
}
```

Syntax

```
curl -d @<inputFile> -X PUT
  -u <administrator>:<password>
  http://<host>:<port>/pools/default/serverGroups?rev=<number>
```

Example

In this example, a JSON file is used.

```
curl -d@file.json -X PUT
  http://Administrator:asdasd@192.168.0.1:8091/pools/default/serverGroups?
rev=120137811
```

Example

In this example, the JSON data is provided on the command line.

```
curl -v -X PUT
  -u Administrator:password
  http://192.168.171.144:8091/pools/default/serverGroups?rev=28418649
  -d '{"groups":
      [{"nodes": [{"otpNode": "ns_1@192.168.171.144"},
                  {"otpNode": "ns_1@192.168.171.145"}],
        "name": "Group 1",
        "uri": "/pools/default/serverGroups/0"},
       {"nodes": [],
        "name": "Group 2",
        "uri": "/pools/default/
serverGroups/3ca074a8456e1d4940cfa3b7badc1e22"}] }'
```

Note: The PUT request is transactional. The request either succeeds completely or fails without impact. If all nodes or groups are not passed, a generic error message: "Bad input" occurs and the server group is removed.

Related Links

Couchbase REST API on page 425

Buckets REST API

The bucket management and configuration REST API endpoints provide a fine level of control over the individual buckets in the cluster, their configuration, and specific operations.

HTTP method	URI path	Description
GET	/pools/default/buckets	Retrieves all bucket and bucket operations information from a cluster.
GET	/pools/default/buckets/default	Retrieves information for a single bucket associated with a cluser.
GET	/pools/default/buckets/bucket_name/stats	Retrieves bucket statistics for a specific bucket.
POST	/pools/default/buckets	Creates a new Couchbase bucket.
DELETE	/pools/default/buckets/bucket_name	Deletes a specific bucket.
POST	/pools/default/buckets/default/controller/doFlush	Flushes a specific bucket.

Related Links

Couchbase REST API on page 425

Getting single bucket information on page 457
To retrieve information about existing buckets and the default bucket, use the `GET` operation with the `/pools/default/buckets/default` URI.

Getting bucket configuration on page 458
To retrieve information about an existing bucket, use the `GET` operation with the `/pools/default/buckets/bucketName` URI.

Getting bucket statistics on page 458
To retrieve bucket statistics, use the `GET` operation with the `/pools/default/buckets/bucket_name/stats` URI.

Getting the bucket streaming URI on page 464
To retrieve the streaming URI, use the `GET` operation with the `/pools/default/buckets/default` URI.

Viewing buckets and bucket operations on page 466
To retrieve all bucket information for a cluster use the `GET` operation with the `/pools/default/buckets` URI.

Creating and editing buckets on page 468
To create and edit buckets, use the `POST` operation with the `/pools/default/bucket` URI.

Deleting buckets on page 470
To delete buckets, use the `DELETE` operation with the `/pools/default/buckets/default` URI.

Flushing buckets on page 471
To flush all buckets, use the `POST` operation with the `/pools/default/buckets/default/controller/doFlush` URI.

Changing bucket authentication on page 471
To change bucket authentication use the `POST` operation with the `/pools/default/buckets/acache authType` URI.

Changing bucket read-write threads on page 472
To change the disk readers and writers setting, use the `POST` operation with the `/pools/default/bucket/bucket_name` URI.

Increasing bucket memory quota on page 473
To increase bucket memory quota, use the `POST` operation with the `/pools/default/buckets/newBucket` URI and `ramQuotaMB` option.

Modifying bucket parameters on page 473
To modify bucket parameters, use the `POST` operation with the `/pools/default/buckets/bucketName` URI with the bucket name as the REST API endpoint.

Getting single bucket information

To retrieve information about existing buckets and the default bucket, use the `GET` operation with the `/pools/default/buckets/default` URI.

To retrieve information for a single bucket associated with a cluster, you make this request, where the last default can be replaced with the name of a specific bucket, if you have named buckets:

```
curl -u admin:password \
    http://localhost:8091/pools/default/buckets/default
```

Replace the *admin*, *password*, and *localhost* values in the above example with actual values.

Couchbase Server returns a large JSON document with bucket information including internal vBucket information:

```
{
    "name":"default",
    "bucketType":"membase",
    "authType":"sasl",
    "saslPassword":"",
    "proxyPort":0,
    "uri":"/pools/default/buckets/default",
    "streamingUri":"/pools/default/bucketsStreaming/default",
    "flushCacheUri":"/pools/default/buckets/default/controller/doFlush",
    "nodes":[
        {
            "systemStats":
            {
                "cpu_utilization_rate":1.5151515151515151,
                "swap_total":6140452864.0,
                "swap_used":0
            },

                ......

            "replicaNumber":1,
            "quota":
            {
                "ram":10246684672.0,
                "rawRAM":5123342336.0
            },
            "basicStats":
            {
                "quotaPercentUsed":0.5281477251650123,
                "opsPerSec":0,"diskFetches":0,
                "itemCount":0,
                "diskUsed":7518856,
                "memUsed":54117632
            }
        }
    ]
}
```

```
GET http://10.4.2.5:8091/pools/default/buckets/default?_=1340926633052
```

```
HTTP/1.1 200 OK
```

Related Links

Buckets REST API on page 456

The bucket management and configuration REST API endpoints provide a fine level of control over the individual buckets in the cluster, their configuration, and specific operations.

Getting bucket configuration

To retrieve information about an existing bucket, use the `GET` operation with the `/pools/default/buckets/bucketName` URI.

The main REST API bucket endpoint is ends with the bucket name.

Clients MUST use the nodes list from the bucket, not the pool to indicate which are the appropriate nodes to connect to.

```
GET /pools/default/buckets/bucketName
```

```
HTTP/1.1 200 OK
Content-Type: application/com.couchbase.store+json
Content-Length: nnn
{
    "name" : "Another bucket",
    "bucketRules" :
    {
        "cacheRange" :
        {
            "min" : 1,
            "max" : 599
        },
        "replicationFactor" : 2
    }
    "nodes" : [
        {
            "hostname" : "10.0.1.20",
            "uri" : "/addresses/10.0.1.20",
            "status" : "healthy",
            "ports" :
            {
                "routing" : 11211,
                "kvcache" :1
            }
        },
        {
            "hostname" : "10.0.1.21",
            "uri" : "/addresses/10.0.1.21",
            "status" : "healthy",
            "ports" :
            {
                "routing" : 11211,
                "kvcache" :1
            }
        }
    ]
}
```

Related Links

Buckets REST API on page 456
The bucket management and configuration REST API endpoints provide a fine level of control over the individual buckets in the cluster, their configuration, and specific operations.

Getting bucket statistics

To retrieve bucket statistics, use the `GET` operation with the `/pools/default/buckets/bucket_name/stats` URI.

You can use the REST API to get statistics at the bucket level from Couchbase Server. Your request URL should be taken from stats.uri property of a bucket response. By default this request returns stats samples for the last minute and for heavily used keys. You use provide additional query parameters in a request to get a more detailed level of information:

zoom	Provides a statistical sampling for that bucket stats at a particular interval (minute | hour | day | week | month | year). For example zoom level of minute will provide bucket statistics from the past minute, a zoom level of day will provide bucket statistics for the past day, and so on. If you provide no zoom level, the server returns samples from the past minute.
haveTStamp	Requests statistics from this timestamp until now. The timestamp is specified as UNIX epoch time. To get a timestamp for a timeframe, make a REST request to the endpoint with a zoom level.

The following is a sample request to the endpoint with no parameters:

```
curl -u admin:password http://localhost:8091/pools/default/buckets/
bucket_name/stats
```

Replace the *admin*, *password*, *localhost*, and *bucket_name* values in the above example with your actual values.

The actual request appears as follows:

```
GET /pools/default/buckets/<bucket name>/stats
Host: localhost:8091
Authorization: Basic xxxxxxxxxxxxxxxxxxx
Accept: application/json X-memcachekv-Store-Client-Specification-Version: 0.1
```

Results:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: nnn
{
    "op": {
        "samples": {
            "hit_ratio": [
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
            ],
            "ep_cache_miss_rate": [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0,
                                    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0,
                                    0 ],


                .....


            "samplesCount": 60,
            "isPersistent": true,
            "lastTStamp":513777166.0,
            "interval": 1000
        },
        "hot_keys": [
```

```
            {
                "name": "48697",
                "ops": 0.0009276437847866419
            },
            {
                "name": "8487",
                "ops": 0.0009276437847866419
            },
            {
                "name": "77262",
                "ops": 0.0009276437847866419
            },
            {
                "name": "58495",
                "ops": 0.0009276437847866419
            },
            {
                "name": "21003",
                "ops": 0.0009276437847866419
            },
            {
                "name": "26850",
                "ops": 0.0009276437847866419
            },
            {
                "name": "73717",
                "ops": 0.0009276437847866419
            },
            {
                "name": "86218",
                "ops": 0.0009276437847866419
            },
            {
                "name": "80344",
                "ops": 0.0009276437847866419
            },
            {
                "name": "83457",
                "ops": 0.0009276437847866419
            }
        ]
    }
}
```


The following are sample requests at this endpoint with optional parameters. Replace the *admin*, *password*, *localhost*, *bucket_name*, and *1376963720000* values in the examples with actual values.

This following example retrieves sample statistics from a bucket for the last minute.

```
curl -X GET -u admin:password  -d zoom=minute http://localhost:8091/pools/
default/buckets/bucket_name/stats
```

The following example retrieves sample statistics from a bucket for the past day.

```
curl -X GET -u admin:password  -d zoom=day http://localhost:8091/pools/
default/buckets/bucket_name/stats
```

The following example retrieves sample statistics from a bucket for the last month.

```
curl -X GET -u admin:password  -d zoom=month http://localhost:8091/pools/
default/buckets/bucket_name/stats
```

The following example sample statistics from a bucket from the timestamp until the server receives the REST request.

```
curl -X GET -u admin:password  -d zoom=hour&haveTStamp=1376963720000 http://
localhost:8091/pools/default/buckets/bucket_name/stats
```

Sample output for each of these requests appears in the same format and with the same fields. Depending on the level of bucket activity, there may be more detail for each field or less. For the sake of brevity we have omitted sample output for each category.


```
{
  "hot_keys": [],
  "op": {
    "interval": 1000,
    "lastTStamp": 1376963580000,
    "isPersistent": true,
    "samplesCount": 1440,
    "samples": {
      "timestamp": [1376955060000, 1376955120000, 1376955180000,
 1376955240000, ... ],
      "xdc_ops": [0, 0, 0, 0, ... ],
      "vb_total_queue_age": [0, 0, 0, 0, ... ],
      "vb_replica_queue_size": [0, 0, 0, 0, ... ],
      "vb_replica_queue_fill": [0, 0, 0, 0, ... ],
      "vb_replica_queue_drain": [0, 0, 0, 0, ... ],
      "vb_replica_queue_age": [0, 0, 0, 0, ... ],
      "vb_replica_ops_update": [0, 0, 0, 0, ... ],
      "vb_replica_ops_create": [0, 0, 0, 0, ... ],
      "vb_replica_num_non_resident": [0, 0, 0, 0, ... ],
      "vb_replica_num": [0, 0, 0, 0, ... ],
      "vb_replica_meta_data_memory": [0, 0, 0, 0, ... ],
      "vb_replica_itm_memory": [0, 0, 0, 0, ... ],
      "vb_replica_eject": [0, 0, 0, 0, ... ],
      "vb_replica_curr_items": [0, 0, 0, 0, ... ],
      "vb_pending_queue_size": [0, 0, 0, 0, ... ],
      "vb_pending_queue_fill": [0, 0, 0, 0, ... ],
      "vb_pending_queue_drain": [0, 0, 0, 0, ... ],
      "vb_pending_queue_age": [0, 0, 0, 0, ... ],
      "vb_pending_ops_update": [0, 0, 0, 0, ... ],
      "vb_pending_ops_create": [0, 0, 0, 0, ... ],
      "vb_pending_num_non_resident": [0, 0, 0, 0, ... ],
      "vb_pending_num": [0, 0, 0, 0, ... ],
      "vb_pending_meta_data_memory": [0, 0, 0, 0, ... ],
      "vb_pending_itm_memory": [0, 0, 0, 0, ... ],
      "vb_pending_eject": [0, 0, 0, 0, ... ],
      "vb_pending_curr_items": [0, 0, 0, 0, ... ],
      "vb_active_queue_size": [0, 0, 0, 0, ... ],
      "vb_active_queue_fill": [0, 0, 0, 0, ... ],
      "vb_active_queue_drain": [0, 0, 0, 0, ... ],
      "vb_active_queue_age": [0, 0, 0, 0, ... ],
      "vb_active_ops_update": [0, 0, 0, 0, ... ],
      "vb_active_ops_create": [0, 0, 0, 0, ... ],
      "vb_active_num_non_resident": [0, 0, 0, 0, ... ],
      "vb_active_num": [1024, 1024, 1024, 1024, ... ],
      "vb_active_meta_data_memory": [0, 0, 0, 0, ... ],
      "vb_active_itm_memory": [0, 0, 0, 0, ... ],
      "vb_active_eject": [0, 0, 0, 0, ... ],
      "ep_ops_create": [0, 0, 0, 0, ... ],
      "ep_oom_errors": [0, 0, 0, 0, ... ],
      "ep_num_value_ejects": [0, 0, 0, 0, ... ],
      "ep_num_ops_set_ret_meta": [0, 0, 0, 0, ... ],
      "ep_num_ops_set_meta": [0, 0, 0, 0, ... ],
      "ep_num_ops_get_meta": [0, 0, 0, 0, ... ],
      "ep_num_ops_del_ret_meta": [0, 0, 0, 0, ... ],
```

```
    "ep_num_ops_del_meta": [0, 0, 0, 0, ... ],
    "ep_num_non_resident": [0, 0, 0, 0, ... ],
    "ep_meta_data_memory": [0, 0, 0, 0, ... ],
    "ep_mem_low_wat": [402653184, 402653184, 402653184, 402653184, ... ],
    "ep_mem_high_wat": [456340275, 456340275, 456340275, 456340275, ... ],
    "ep_max_data_size": [536870912, 536870912, 536870912, 536870912, ... ],
    "ep_kv_size": [0, 0, 0, 0, ... ],
    "ep_item_commit_failed": [0, 0, 0, 0, ... ],
    "ep_flusher_todo": [0, 0, 0, 0, ... ],
    "ep_diskqueue_items": [0, 0, 0, 0, ... ],
    "ep_diskqueue_fill": [0, 0, 0, 0, ... ],
    "ep_diskqueue_drain": [0, 0, 0, 0, ... ],
    "ep_bg_fetched": [0, 0, 0, 0, ... ],
    "disk_write_queue": [0, 0, 0, 0, ... ],
    "disk_update_total": [0, 0, 0, 0, ... ],
    "disk_update_count": [0, 0, 0, 0, ... ],
    "disk_commit_total": [0, 0, 0, 0, ... ],
    "disk_commit_count": [0, 0, 0, 0, ... ],
    "delete_misses": [0, 0, 0, 0, ... ],
    "delete_hits": [0, 0, 0, 0, ... ],
    "decr_misses": [0, 0, 0, 0, ... ],
    "decr_hits": [0, 0, 0, 0, ... ],
    "curr_items_tot": [0, 0, 0, 0, ... ],
    "curr_items": [0, 0, 0, 0, ... ],
    "curr_connections": [9, 9, 9, 9, ... ],
    "avg_bg_wait_time": [0, 0, 0, 0, ... ],
    "avg_disk_commit_time": [0, 0, 0, 0, ... ],
    "avg_disk_update_time": [0, 0, 0, 0, ... ],
    "vb_pending_resident_items_ratio": [0, 0, 0, 0, ... ],
    "vb_replica_resident_items_ratio": [0, 0, 0, 0, ... ],
    "vb_active_resident_items_ratio": [0, 0, 0, 0, ... ],
    "vb_avg_total_queue_age": [0, 0, 0, 0, ... ],
    "vb_avg_pending_queue_age": [0, 0, 0, 0, ... ],
    "couch_total_disk_size": [8442535, 8449358, 8449392, 8449392, ... ],
    "couch_docs_fragmentation": [0, 0, 0, 0, ... ],
    "couch_views_fragmentation": [0, 0, 0, 0, ... ],
    "hit_ratio": [0, 0, 0, 0, ... ],
    "ep_cache_miss_rate": [0, 0, 0, 0, ... ],
    "ep_resident_items_rate": [100, 100, 100, 100, ... ],
    "vb_avg_active_queue_age": [0, 0, 0, 0, ... ],
    "vb_avg_replica_queue_age": [0, 0, 0, 0, ... ],
    "bg_wait_count": [0, 0, 0, 0, ... ],
    "bg_wait_total": [0, 0, 0, 0, ... ],
    "bytes_read": [103.5379762658911, 103.53627151841438,
103.53627262555834, 103.53739884434893, ... ],
    "bytes_written": [20793.105529503482, 20800.99759272974,
20802.109356966503, 20803.59949917707, ... ],
    "cas_badval": [0, 0, 0, 0, ... ],
    "cas_hits": [0, 0, 0, 0, ... ],
    "cas_misses": [0, 0, 0, 0, ... ],
    "cmd_get": [0, 0, 0, 0, ... ],
    "cmd_set": [0, 0, 0, 0, ... ],
    "couch_docs_actual_disk_size": [8442535, 8449358, 8449392,
8449392, ... ],
    "couch_docs_data_size": [8435712, 8435712, 8435712, 8435712, ... ],
    "couch_docs_disk_size": [8435712, 8435712, 8435712, 8435712, ... ],
    "couch_views_actual_disk_size": [0, 0, 0, 0, ... ],
    "couch_views_data_size": [0, 0, 0, 0, ... ],
    "couch_views_disk_size": [0, 0, 0, 0, ... ],
    "couch_views_ops": [0, 0, 0, 0, ... ],
    "ep_ops_update": [0, 0, 0, 0, ... ],
    "ep_overhead": [27347928, 27347928, 27347928, 27347928, ... ],
    "ep_queue_size": [0, 0, 0, 0, ... ],
    "ep_tap_rebalance_count": [0, 0, 0, 0, ... ],
```

```
            "ep_tap_rebalance_qlen": [0, 0, 0, 0, ... ],
            "ep_tap_rebalance_queue_backfillremaining": [0, 0, 0, 0, ... ],
            "ep_tap_rebalance_queue_backoff": [0, 0, 0, 0, ... ],
            "ep_tap_rebalance_queue_drain": [0, 0, 0, 0, ... ],
            "ep_tap_rebalance_queue_fill": [0, 0, 0, 0, ... ],
            "ep_tap_rebalance_queue_itemondisk": [0, 0, 0, 0, ... ],
            "ep_tap_rebalance_total_backlog_size": [0, 0, 0, 0, ... ],
            "ep_tap_replica_count": [0, 0, 0, 0, ... ],
            "ep_tap_replica_qlen": [0, 0, 0, 0, ... ],
            "ep_tap_replica_queue_backfillremaining": [0, 0, 0, 0, ... ],
            "ep_tap_replica_queue_backoff": [0, 0, 0, 0, ... ],
            "ep_tap_replica_queue_drain": [0, 0, 0, 0, ... ],
            "ep_tap_replica_queue_fill": [0, 0, 0, 0, ... ],
            "ep_tap_replica_queue_itemondisk": [0, 0, 0, 0, ... ],
            "ep_tap_replica_total_backlog_size": [0, 0, 0, 0, ... ],
            "ep_tap_total_count": [0, 0, 0, 0, ... ],
            "ep_tap_total_qlen": [0, 0, 0, 0, ... ],
            "ep_tap_total_queue_backfillremaining": [0, 0, 0, 0, ... ],
            "ep_tap_total_queue_backoff": [0, 0, 0, 0, ... ],
            "ep_tap_total_queue_drain": [0, 0, 0, 0, ... ],
            "ep_tap_total_queue_fill": [0, 0, 0, 0, ... ],
            "ep_tap_total_queue_itemondisk": [0, 0, 0, 0, ... ],
            "ep_tap_total_total_backlog_size": [0, 0, 0, 0, ... ],
            "ep_tap_user_count": [0, 0, 0, 0, ... ],
            "ep_tap_user_qlen": [0, 0, 0, 0, ... ],
            "ep_tap_user_queue_backfillremaining": [0, 0, 0, 0, ... ],
            "ep_tap_user_queue_backoff": [0, 0, 0, 0, ... ],
            "ep_tap_user_queue_drain": [0, 0, 0, 0, ... ],
            "ep_tap_user_queue_fill": [0, 0, 0, 0, ... ],
            "ep_tap_user_queue_itemondisk": [0, 0, 0, 0, ... ],
            "ep_tap_user_total_backlog_size": [0, 0, 0, 0, ... ],
            "ep_tmp_oom_errors": [0, 0, 0, 0, ... ],
            "ep_vb_total": [1024, 1024, 1024, 1024, ... ],
            "evictions": [0, 0, 0, 0, ... ],
            "get_hits": [0, 0, 0, 0, ... ],
            "get_misses": [0, 0, 0, 0, ... ],
            "incr_hits": [0, 0, 0, 0, ... ],
            "incr_misses": [0, 0, 0, 0, ... ],
            "mem_used": [27347928, 27347928, 27347928, 27347928, ... ],
            "misses": [0, 0, 0, 0, ... ],
            "ops": [0, 0, 0, 0, ... ],
            "replication_active_vbreps": [0, 0, 0, 0, ... ],
            "replication_bandwidth_usage": [0, 0, 0, 0, ... ],
            "replication_changes_left": [0, 0, 0, 0, ... ],
            "replication_commit_time": [0, 0, 0, 0, ... ],
            "replication_data_replicated": [0, 0, 0, 0, ... ],
            "replication_docs_checked": [0, 0, 0, 0, ... ],
            "replication_docs_latency_aggr": [0, 0, 0, 0, ... ],
            "replication_docs_latency_wt": [0, 0, 0, 0, ... ],
            "replication_docs_rep_queue": [0, 0, 0, 0, ... ],
            "replication_docs_written": [0, 0, 0, 0, ... ],
            "replication_meta_latency_aggr": [0, 0, 0, 0, ... ],
            "replication_meta_latency_wt": [0, 0, 0, 0, ... ],
            "replication_num_checkpoints": [0, 0, 0, 0, ... ],
            "replication_num_failedckpts": [0, 0, 0, 0, ... ],
            "replication_rate_replication": [0, 0, 0, 0, ... ],
            "replication_size_rep_queue": [0, 0, 0, 0, ... ],
            "replication_waiting_vbreps": [0, 0, 0, 0, ... ],
            "replication_work_time": [0, 0, 0, 0, ... ]
        }
    }
}
```


Related Links

Buckets REST API on page 456
The bucket management and configuration REST API endpoints provide a fine level of control over the individual buckets in the cluster, their configuration, and specific operations.

Getting the bucket streaming URI

To retrieve the streaming URI, use the `GET` operation with the `/pools/default/buckets/default` URI.

The individual bucket request is exactly the same as what would be obtained from the item in the array for the entire buckets list. The streamingUri is exactly the same except it streams HTTP chunks using chunked encoding. A response of "\n\n\n\n" delimits chunks. This will likely be converted to a "zero chunk" in a future release of this API, and thus the behavior of the streamingUri should be considered evolving.

```
GET /pools/default/buckets/default
Host: localhost:8091
Authorization: Basic xxxxxxxxxxxxxxxxxxx
Accept: application/json
X-memcachekv-Store-Client-Specification-Version: 0.1
```

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: nnn
{
  "name": "default",
  "bucketType": "couchbase",
  "authType": "sasl",
  "saslPassword": "",
  "proxyPort": 0,
  "uri": "/pools/default/buckets/default",
  "streamingUri": "/pools/default/bucketsStreaming/default",
  "flushCacheUri": "/pools/default/buckets/default/controller/doFlush",
  "nodes": [
    {
      "uptime": "308",
      "memoryTotal": 3940818944.0,
      "memoryFree": 1608724480,
      "mcdMemoryReserved": 3006,
      "mcdMemoryAllocated": 3006,
      "replication": 1.0,
      "clusterMembership": "active",
      "status": "healthy",
      "hostname": "172.25.0.2:8091",
      "clusterCompatibility": 1,
      "version": "1.6.4r_107_g49a149d",
      "os": "i486-pc-linux-gnu",
      "ports": {
        "proxy": 11211,
        "direct": 11210
      }
    },
    {
      "uptime": "308",
      "memoryTotal": 3940818944.0,
      "memoryFree": 1608724480,
      "mcdMemoryReserved": 3006,
      "mcdMemoryAllocated": 3006,
      "replication": 1.0,
      "clusterMembership": "active",
      "status": "healthy",
      "hostname": "172.25.0.3:8091",
      "clusterCompatibility": 1,
      "version": "1.6.4r_107_g49a149d",
```

```
      "os": "i486-pc-linux-gnu",
      "ports": {
        "proxy": 11211,
        "direct": 11210
      }
    },
    {
      "uptime": "308",
      "memoryTotal": 3940818944.0,
      "memoryFree": 1608597504,
      "mcdMemoryReserved": 3006,
      "mcdMemoryAllocated": 3006,
      "replication": 1.0,
      "clusterMembership": "active",
      "status": "healthy",
      "hostname": "172.25.0.4:8091",
      "clusterCompatibility": 1,
      "version": "1.6.4r_107_g49a149d",
      "os": "i486-pc-linux-gnu",
      "ports": {
        "proxy": 11211,
        "direct": 11210
      }
    }
  ],
  "stats": {
    "uri": "/pools/default/buckets/default/stats"
  },
  "nodeLocator": "vbucket",
  "vBucketServerMap": {
    "hashAlgorithm": "CRC",
    "numReplicas": 1,
    "serverList": [
      "172.25.0.2:11210",
      "172.25.0.3:11210",
      "172.25.0.4:11210"
    ],
    "vBucketMap": [
      [1,0],
      [2,0],
      [1,2],
      [2,1],
      [1,2],
      [0,2],
      [0,1],
      [0,1]
    ]
  },
  "replicaNumber": 1,
  "quota": {
    "ram": 1887436800,
    "rawRAM":145600
  },
  "basicStats": {
    "quotaPercentUsed": 14.706055058373344,
    "opsPerSec": 0,
    "diskFetches": 0,
    "itemCount": 65125,
    "diskUsed": 139132928,
    "memUsed": 277567495
  }
}
```


Related Links

Buckets REST API on page 456
The bucket management and configuration REST API endpoints provide a fine level of control over the individual buckets in the cluster, their configuration, and specific operations.

Viewing buckets and bucket operations

To retrieve all bucket information for a cluster use the `GET` operation with the `/pools/default/buckets` URI.

If you create your own SDK for Couchbase, you can use either the proxy path or the direct path to connect to Couchbase Server. If your SDK uses the direct path, your SDK will not be insulated from most reconfiguration changes to the bucket. This means your SDK will need to either poll the bucket's URI or connect to the streamingUri to receive updates when the bucket configuration changes. Bucket configuration can happen for instance, when nodes are added, removed, or if a node fails.

To retrieve information for all bucket for cluster:

```
curl -u admin:password http://localhost:8091/pools/default/buckets
```

Replace the *admin*, *password*, and *localhost* values in the above example with your actual values.

```
GET /pools/default/buckets
Host: localhost:8091
Authorization: Basic xxxxxxxxxxxxxxxxxxx
Accept: application/json
X-memcachekv-Store-Client-Specification-Version: 0.1
```

```
HTTP/1.1 200 OK
Server: Couchbase Server 1.6.0
Pragma: no-cache
Date: Wed, 03 Nov 2010 18:12:19 GMT
Content-Type: application/json
Content-Length: nnn
Cache-Control: no-cache no-store max-age=0
[
    {
        "name": "default",
        "bucketType": "couchbase",
        "authType": "sasl",
        "saslPassword": "",
        "proxyPort": 0,
        "uri": "/pools/default/buckets/default",
        "streamingUri": "/pools/default/bucketsStreaming/default",
        "flushCacheUri": "/pools/default/buckets/default/controller/doFlush",
        "nodes": [
            {
                "uptime": "784657",
                "memoryTotal": 8453197824.0,
                "memoryFree": 1191157760,
                "mcdMemoryReserved": 6449,
                "mcdMemoryAllocated": 6449,
                "clusterMembership": "active",
                "status": "unhealthy",
                "hostname": "10.1.15.148:8091",
                "version": "1.6.0",
                "os": "windows",
                "ports": {
                    "proxy": 11211,
                    "direct": 11210
                }
            }
        ],
        "stats": {
            "uri": "/pools/default/buckets/default/stats"
        },
        "nodeLocator": "vbucket",
```

```
        "vBucketServerMap": {
            "hashAlgorithm": "CRC",
            "numReplicas": 1,
            "serverList": [
                "192.168.1.2:11210"
            ],
      "vBucketMap": [ [ 0, -1 ], [ 0, -1 ], [ 0, -1 ], [ 0, -1 ], [ 0, -1 ],
 [ 0, -1 ]]
  },
        "replicaNumber": 1,
        "quota": {
            "ram": 104857600,
            "rawRAM": 104857600
        },
        "basicStats": {
            "quotaPercentUsed": 24.360397338867188,
            "opsPerSec": 0,
            "diskFetches": 0,
            "itemCount": 0,
            "diskUsed": 0,
            "memUsed": 25543728
        }
    },
    {
        "name": "test-application",
        "bucketType": "memcached",
        "authType": "sasl",
        "saslPassword": "",
        "proxyPort": 0,
        "uri": "/pools/default/buckets/test-application",
        "streamingUri": "/pools/default/bucketsStreaming/test-application",
        "flushCacheUri": "/pools/default/buckets/test-application/controller/
doFlush",
        "nodes": [
            {
                "uptime": "784657",
                "memoryTotal": 8453197824.0,
                "memoryFree": 1191157760,
                "mcdMemoryReserved": 6449,
                "mcdMemoryAllocated": 6449,
                "clusterMembership": "active",
                "status": "healthy",
                "hostname": "192.168.1.2:8091",
                "version": "1.6.0",
                "os": "windows",
                "ports": {
                    "proxy": 11211,
                    "direct": 11210
                }
            }
        ],
        "stats": {
            "uri": "/pools/default/buckets/test-application/stats"
        },
        "nodeLocator": "ketama",
        "replicaNumber": 0,
        "quota": {
            "ram": 67108864,
            "rawRAM": 67108864
        },
        "basicStats": {
            "quotaPercentUsed": 4.064150154590607,
            "opsPerSec": 0,
            "hitRatio": 0,
```

```
            "itemCount": 1385,
            "diskUsed": 0,
            "memUsed": 2727405
        }
    }
]
```


Related Links

Buckets REST API on page 456
The bucket management and configuration REST API endpoints provide a fine level of control over the individual buckets in the cluster, their configuration, and specific operations.

Creating and editing buckets

To create and edit buckets, use the `POST` operation with the `/pools/default/bucket` URI.

Buckets are created and edited with a `POST` sent to the REST URI endpoint for buckets in a cluster. This can be used to create either a Couchbase or a Memcached type bucket. Bucket names cannot have a leading underscore.

- Method and URI - `POST /pools/default/buckets`
- Request Data - List of payload parameters for the new bucket
- Response Data - JSON of the bucket confirmation or an error condition
- Authentication Required - yes

This endpoint is also used to get a list of buckets that exist for a cluster.

Important: When editing bucket properties, be sure to specify all bucket properties. If a bucket property is not specified (whether or not you are changing the existing value), Couchbase Server may reset the property to the default. Even if you do not intend to change a certain property, re-specify the existing value to avoid this behavior.

The REST API returns a successful response when preliminary files for a data bucket are created on one node. However, if a multi-node cluster is implemented, bucket creation may not have completed for all nodes when a response is sent. Therefore, it is possible that the bucket is not available for operations immediately after this REST call successful returns.

To verify that a bucket is available, try to read a key from the bucket. If a ‘key not found’ error is received or the document for the key is returned, then the bucket exists and is available to all nodes in a cluster. Key requests can be issued via a Couchbase SDK with any node in the cluster. See the *Couchbase Developer Guide* for more information.

Payload Arguments	Description
`authType`	Required parameter. Type of authorization to be enabled for the new bucket as a string. Defaults to blank password if not specified. “sasl” enables authentication. “none” disables authentication.
`bucketType`	Required parameter. Type of bucket to be created. String value. “memcached” configures as Memcached bucket. “couchbase” configures as Couchbase bucket
`flushEnabled`	Optional parameter. Enables the ‘flush all’ functionality on the specified bucket. Boolean. 1 enables flush all support, 0 disables flush all support. Defaults to 0.
`name`	Required parameter. Name for new bucket.
`parallelDBAndViewCompaction`	Optional parameter. String value. Indicates whether database and view files on disk can be compacted simultaneously. Defaults to “false.”
`proxyPort`	Required parameter. Numeric. Proxy port on which the bucket communicates. Must be a valid network port which is not already in use. You must provide a valid port number if the authorization type is not SASL.

Payload Arguments	Description
`ramQuotaMB`	Required parameter. RAM Quota for new bucket in MB. Numeric. The minimum you can specify is 100, and the maximum can only be as great as the memory quota established for the node. If other buckets are associated with a node, RAM Quota can only be as large as the amount memory remaining for the node, accounting for the other bucket memory quota.
`replicaIndex`	Optional parameter. Boolean. 1 enable replica indexes for replica bucket data while 0 disables. Default of 1.
`replicaNumber`	Optional parameter. Numeric. Number of replicas to be configured for this bucket. Required parameter when creating a Couchbase bucket. Default 1, minimum 0, maximum 3.
`saslPassword`	Optional Parameter. String. Password for SASL authentication. Required if SASL authentication has been enabled.
`threadsNumber`	Optional Parameter. Integer from 2 to 8. Change the number of concurrent readers and writers for the data bucket.
Return Codes	
202	Accepted
204	Bad Request JSON with errors in the form of {"errors": {.... }} name: Bucket with given name already exists ramQuotaMB: RAM Quota is too large or too small replicaNumber: Must be specified and must be a non-negative integer proxyPort: port is invalid, port is already in use
404	Object Not Found

When you create a bucket you must provide the `authType` parameter:

- If you set `authType` to `none`, then you must specify a proxyPort number.
- If you set `authType` to `sasl`, then you may optionally provide a `saslPassword` parameter.

The `ramQuotaMB` parameter specifies how much memory, in megabytes, you want to allocate to each node for the bucket. The minimum supported value is 100MB.

- If the items stored in a memcached bucket take space beyond the `ramQuotaMB`, Couchbase Sever typically will evict items on least-requested-item basis. Couchbase Server may evict other infrequently used items depending on object size, or whether or not an item is being referenced.
- In the case of Couchbase buckets, the system may return temporary failures if the `ramQuotaMB` is reached. The system will try to keep 25% of the available ramQuotaMB free for new items by ejecting old items from occupying memory. In the event these items are later requested, they will be retrieved from disk.

In the following example, replace the *admin*, *password*, *localhost*, *newbucket*, *200*, *2*, and *11215* values with your actual values.

```
curl -X POST -u admin:password -d name=newbucket -d ramQuotaMB=200 -d
 authType=none \
-d replicaNumber=2 -d proxyPort=11215 http://localhost:8091/pools/default/
buckets
```

The parameters for configuring the bucket are provided as payload data, with each parameter and value provided as a key/value pair, separated by an ampersand.

The HTTP request should include the parameters setting in the payload of the `POST` request:

```
POST /pools/default/buckets
HTTP/1.1
Host: localhost:8091
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
```

```
Authorization: Basic YWRtaW46YWRtaW4=
Content-Length: xx
name=newbucket&ramQuotaMB=20&authType=none&replicaNumber=2&proxyPort=11215
```

If the bucket creation was successful, HTTP response 202 (Accepted) will be returned with empty content.

```
202 Accepted
```

If the bucket could not be created, because the parameter was missing or incorrect, HTTP response 400 will be returned, with a JSON payload containing the error reason.

Related Links

Buckets REST API on page 456
The bucket management and configuration REST API endpoints provide a fine level of control over the individual buckets in the cluster, their configuration, and specific operations.

Related topics

Multiple readers and writers on page 103
Multi-threaded readers and writers provide multiple processes to simultaneously read and write data on disk. Simultaneous reads and writes increases disk speed and improves the read rate from disk.

Deleting buckets

To delete buckets, use the `DELETE` operation with the `/pools/default/buckets/default` URI.

Method	**`DELETE /pools/default/buckets/bucket_name`**
Request Data	None
Response Data	None
Authentication Required	yes
	Return Codes
200	OK Bucket Deleted on all nodes
401	Unauthorized
404	Object Not Found
500	Bucket could not be deleted on all nodes
503	Buckets cannot be deleted during a rebalance

Warning: This operation is data destructive.The service makes no attempt to double check with the user. It simply moves forward. Clients applications using this are advised to double check with the end user before sending such a request.

```
DELETE /pools/default/buckets/default
Host: localhost:8091
Authorization: Basic xxxxxxxxxxxxxxxxxxx
```

Bucket deletion is a synchronous operation but because the cluster may include a number of nodes, they may not all be able to delete the bucket. If all the nodes delete the bucket within the standard timeout of 30 seconds, `200` will be returned. If the bucket cannot be deleted on all nodes within the 30 second timeout, a `500` is returned.

Further requests to delete the bucket will return a `404` error. Creating a new bucket with the same name may return an error that the bucket is still being deleted.

Related Links

Buckets REST API on page 456
The bucket management and configuration REST API endpoints provide a fine level of control over the individual buckets in the cluster, their configuration, and specific operations.

Flushing buckets

To flush all buckets, use the `POST` operation with the `/pools/default/buckets/default/controller/doFlush` URI.

The `doFlush` operation empties the contents of the specified bucket, deleting all stored data. The operation only succeeds if flush is enabled on configured bucket. The format of the request is the URL of the REST endpoint using the `POST` HTTP operation:

Warning: This operation is data destructive. The service makes no attempt to confirm or double check the request. Client applications using this are advised to double check with the end user before sending such a request. You can control and limit the ability to flush individual buckets by setting the `flushEnabled` parameter on a bucket in Couchbase Web Console or via `cbepctl flush_param`.

```
http://localhost:8091/pools/default/buckets/default/controller/doFlush
```

For example, using `curl` :

```
curl -X POST 'http://admin:password@localhost:8091/pools/default/buckets/
default/controller/doFlush'
```

Replace the *admin*, *password*, and *localhost* values in the above example with your actual values.

Equivalent HTTP protocol request:

```
POST /pools/default/buckets/default/controller/doFlush
Host: localhost:8091
Authorization: Basic xxxxxxxxxxxxxxxxxxx
```

Parameters and payload data are ignored, but the request must including the authorization header if the system has been secured.

If flushing is disable for the specified bucket, a 400 response will be returned with the bucket status:

```
{"_":"Flush is disabled for the bucket"}
```

If the flush is successful, the HTTP response code is `200` :

```
HTTP/1.1 200 OK
```

Important: The flush request may lead to significant disk activity as the data in the bucket is deleted from the database. The high disk utilization may affect the performance of your server until the data has been successfully deleted.

Note: The flush request is not transmitted over XDCR replication configurations; the remote bucket will not be flushed.

Couchbase Server returns a HTTP 404 response if the URI is invalid or if it does not correspond to an active bucket in the system.

```
404 Not Found
```

Related Links

Buckets REST API on page 456
The bucket management and configuration REST API endpoints provide a fine level of control over the individual buckets in the cluster, their configuration, and specific operations.

Changing bucket authentication

To change bucket authentication use the `POST` operation with the `/pools/default/buckets/acache authType` URI.

Changing a bucket from port-based authentication to SASL authentication is achieved by changing the active bucket configuration.

Note: When changing the active bucket configuration, specify the existing configuration parameters and the changed authentication parameters in the request.

`POST authType` is used with the `/pools/default/buckets/acache` URI.

```
curl -X POST -u admin:password -d ramQuotaMB=130 -d authType=sasl \
  -d saslPassword=letmein \
  http://localhost:8091/pools/default/buckets/acache
```

Replace the *admin*, *password*, *localhost*, *130*, *letmein*, and *acache* values in the above example with your actual values.

Related Links

Buckets REST API on page 456
The bucket management and configuration REST API endpoints provide a fine level of control over the individual buckets in the cluster, their configuration, and specific operations.

Changing bucket read-write threads

To change the disk readers and writers setting, use the `POST` operation with the `/pools/default/bucket/bucket_name` URI.

POST /pools/default/buckets/bucket_name	threadsNumber

View bucket read-write threads

The number of read-write threads are included with the data bucket properties via a REST call:

```
curl -x GET -u Admin:password http://localhost:8091/pools/default/buckets/
bucket_name
```

This provides information about the named bucket as a JSON response, including the total number of threads:

```
{"name":"bucket_name","bucketType":"couchbase"
....
"replicaNumber":1,
"threadsNumber":5,
....
}
```

Change bucket read-write threads

To change the bucket read-write thread setting, use the `threadsNumber` parameter with a value from two to eight. The following is an example REST call:

```
curl -X POST -u Admin:password http://10.3.3.72:8091/pools/default/buckets/
bucket_name \
  -d ramQuotaMB=4000 -d threadsNumber=3 -v
```

Request

You see the following request via HTTP:

```
About to connect() to 10.3.3.72 port 8091 (#0)
 Trying 10.3.3.72... connected
 Connected to 10.3.3.72 (10.3.3.72) port 8091 (#0)
 Server auth using Basic with user 'Administrator'
 POST /pools/default/buckets/bucket_name HTTP/1.1
 ....
```

Successful response:

```
HTTP/1.1 200 OK
....
```

Unsuccessful response

If an invalid number of threads is provides, a response similar to the following occurs:

```
HTTP/1.1 400 Bad Request
....
{"errors":{"threadsNumber":"The number of threads can't be greater
 than 8"},"
```

Related Links

Buckets REST API on page 456
The bucket management and configuration REST API endpoints provide a fine level of control over the individual buckets in the cluster, their configuration, and specific operations.

Increasing bucket memory quota

To increase bucket memory quota, use the `POST` operation with the `/pools/default/buckets/newBucket` URI and `ramQuotaMB` option.

You can increase and decrease a bucket's ramQuotaMB from its current level. However, while increasing will do no harm, decreasing should be done with proper sizing. Decreasing the bucket's ramQuotaMB lowers the watermark, and some items may be unexpectedly ejected if the ramQuotaMB is set too low.

Note: There are some known issues with changing the ramQuotaMB for memcached bucket types.

Example of a request:

```
curl -X POST -u admin:password -d ramQuotaMB=25 -d authType=none \
  -d proxyPort=11215 http://localhost:8091/pools/default/buckets/newBucket
```

Replace the *admin*, *password*, *localhost*, *25*, *11215*, and *newBucket* values in the above example with your actual values.

A 202 response indicates that the quota will be changed asynchronously throughout the servers in the cluster.

```
HTTP/1.1 202 OK
Server: Couchbase Server 1.6.0
Pragma: no-cache
Date: Wed, 29 Sep 2010 20:01:37 GMT
Content-Length: 0
Cache-Control: no-cache no-store max-age=0
```

Related Links

Buckets REST API on page 456
The bucket management and configuration REST API endpoints provide a fine level of control over the individual buckets in the cluster, their configuration, and specific operations.

Modifying bucket parameters

To modify bucket parameters, use the `POST` operation with the `/pools/default/buckets/bucketName` URI with the bucket name as the REST API endpoint.

You can modify existing bucket parameters by posting the updated parameters used to create the bucket to the bucket's URI. Do not omit a parameter in your request since this is equivalent to not setting it in many cases. We recommend you do a request to get current bucket settings, make modifications as needed and then make your POST request to the bucket URI.

For example, to edit the bucket `customer` :

```
curl -v -X POST -u admin:password -d name=customer \
-d flushEnabled=0 -d replicaNumber=1 -d authType=none \
-d ramQuotaMB=200 -d proxyPort=11212 \
 http://localhost:8091/pools/default/buckets/customer
```

Replace the *admin*, *password*, *localhost*, *customer*, *0*, *1*, *200*, *11212*, and *customer* values in the above example with your actual values.

If the request is successful, HTTP response 200 will be returned with an empty data content.

Warning: The bucket name cannot be changed via the REST API.

Related Links

Buckets REST API on page 456
The bucket management and configuration REST API endpoints provide a fine level of control over the individual buckets in the cluster, their configuration, and specific operations.

Views REST API

You can index and query JSON documents using views. Views are functions written in JavaScript that can serve several purposes in your application. You can use them to: find all the documents in your database, create a copy of data in a document and present it in a specific order, create an index to efficiently find documents by a particular value or by a particular structure in the document, represent relationships between documents, and perform calculations on data contained in documents.

You store view functions in a design document as JSON and can use the REST API to manage your design documents.

Related Links

Couchbase REST API on page 425

Managing design documents on page 475
Design documents are used to store one or more view definitions. Views can be defined within a design document and uploaded to the server through the REST API.

Querying views on page 479

Views requests on page 481

Managing design documents

Design documents are used to store one or more view definitions. Views can be defined within a design document and uploaded to the server through the REST API.

Storing design documents

To create a new design document with one or more views, you can upload the corresponding design document using the REST API with the definition in place. The format of this command is as shown in the table below:

Put Design Document	Description
Method	`PUT /bucket/_design/design-doc`
Request Data	Design document definition (JSON)
Response Data	Success and stored design document ID
Authentication Required	optional
Return Codes	
201	Document created successfully.
401	The item requested was not available using the supplied authorization, or authorization was not supplied.

When creating a design document through the REST API, we recommend that you create a development (`dev`) view. We recommend that you create a dev design document and views first, and then check the output of the configured views in your design document. To create a dev view you *must* explicitly use the `dev_` prefix for the design document name.

For example, using `curl`, you can create a design document, `byfield`, by creating a text file (with the name `byfield.ddoc`) with the design document content using the following command:

```
> curl -X PUT -H 'Content-Type: application/json' \
   http://user:password@localhost:8092/sales/_design/dev_byfield' \
   -d @byfield.ddoc
```

In the above example:

- `-X PUT`

Indicates that an HTTP PUT operation is requested.

- `-H 'Content-Type: application/json'`

Specifies the HTTP header information. Couchbase Server requires the information to be sent and identified as the `application/json` datatype. Information not supplied with the content-type set in this manner will be rejected.

- `http://user:password@localhost:8092/sales/_design/dev_byfield'`

The URL, including authentication information, of the bucket where you want the design document uploaded. The `user` and `password` should either be the Administration privileges, or for SASL protected buckets, the bucket name and bucket password. If the bucket does not have a password, then the authentication information is not required.

The view being accessed in this case is a development view. To create a development view, you *must* use the `dev_` prefix to the view name.

As a `PUT` command, the URL is also significant, in that the location designates the name of the design document. In the example, the URL includes the name of the bucket (`sales`) and the name of the design document that will be created `dev_byfield`.

- `-d @byfield.ddoc`

Specifies that the data payload should be loaded from the file `byfield.ddoc`.

If successful, the HTTP response code will be 201 (created). The returned JSON will contain the field `ok` and the ID of the design document created:

```
{
    "ok":true,
    "id":"_design/dev_byfield"
}
```

The design document will be validated before it is created or updated in the system. The validation checks for valid JavaScript and for the use of valid built-in reduce functions. Any validation failure is reported as an error.

In the event of an error, the returned JSON will include the field `error` with a short description, and the field `reason` with a longer description of the problem.

The format of the design document should include all the views defined in the design document, incorporating both the map and reduce functions for each named view. For example:

```
{"views":{"byloc":{"map":"function (doc, meta) {\n  if (meta.type == \"json
\") {\n    emit(doc.city, doc.sales);\n  } else {\n    emit([\"blob\"]);\n
  }\n}"}}}
```

Formatted, the design document looks like this:

```
{
   "views" : {
      "byloc" : {
         "map" : "function (doc, meta) {\n  if (meta.type == \"json\") {\n
 emit(doc.city, doc.sales);\n  } else {\n    emit([\"blob\"]);\n  }\n}"
      }
   }
}
```

The top-level `views` field lists one or more view definitions (the `byloc` view in this example), and for each view, a corresponding `map()` function.

Retrieving design documents

To obtain an existing design document from a given bucket, you need to access the design document from the corresponding bucket using a `GET` request, as detailed in the table below.

Get Design Document	Description
Method	`GET /bucket/_design/design-doc`
Request Data	Design document definition (JSON)
Response Data	Success and stored design document ID
Authentication Required	optional
Return Codes	
200	Request completed successfully.
401	The item requested was not available using the supplied authorization, or authorization was not supplied.
404	The requested content could not be found. The returned content will include further information, as a JSON object, if available.

To get back all the design documents with views defined on a bucket, the use following URI path with the GET request. In addition to get specific design documents back, the name of the design document can be specified to retrieve it.

```
"ddocs": {
        "uri": "/pools/default/buckets/default/ddocs" // To obtain design docs
 for this bucket
    }
```

For example, to get the existing design document from the bucket `sales` for the design document `byfield`:

```
> curl -X GET \
    -H 'Content-Type: application/json' \
    'http://user:password@192.168.0.77:8092/sales/_design/dev_byfield
```

Through `curl` this will download the design document to the file `dev_byfield` filename.

If the bucket does not have a password, you can omit the authentication information. If the view does not exist you will get an error:

```
{
   "error":"not_found",
   "reason":"missing"
}
```

The HTTP response header will include a JSON document containing the metadata about the design document being accessed. The information is returned within the `X-Couchbase-Meta` header of the returned data. You can obtain this information by using the `-v` option to the `curl`.

For example:

```
&gt; curl -v -X GET \
   -H 'Content-Type: application/json' \
   'http://user:password@192.168.0.77:8092/sales/_design/
* About to connect() to 192.168.0.77 port 8092 (#0)
*   Trying 192.168.0.77...
  % Total    % Received % Xferd  Average Speed   Time    Time     Time
 Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--
 0* connected
* Connected to 192.168.0.77 (192.168.0.77) port 8092 (#0)
* Server auth using Basic with user 'Administrator'
> GET /sales/_design/something HTTP/1.1
> Authorization: Basic QWRtaW5pc3RyYXRvcjpUYW1zaW4=
```

```
> User-Agent: curl/7.24.0 (x86_64-apple-darwin12.0) libcurl/7.24.0
 OpenSSL/0.9.8r zlib/1.2.5
> Host: 192.168.0.77:8092
> Accept: */*
> Content-Type: application/json
>
< HTTP/1.1 200 OK
< X-Couchbase-Meta: {"id":"_design/
dev_sample","rev":"5-2785ea87","type":"json"}
< Server: MochiWeb/1.0 (Any of you quaids got a smint?)
< Date: Mon, 13 Aug 2012 10:45:46 GMT
< Content-Type: application/json
< Content-Length: 159
< Cache-Control: must-revalidate
<
{ [data not shown]
100   159  100   159    0     0  41930      0 --:--:-- --:--:-- --:--:-- 53000
* Connection #0 to host 192.168.0.77 left intact
* Closing connection #0
```

The metadata matches the corresponding metadata for a data document.

Deleting design documents

To delete a design document, you use the `DELETE` HTTP request with the URL of the corresponding design document. The summary information for this request is shown in the table below:

Delete Design Document	Description
Method	`DELETE /bucket/_design/design-doc`
Request Data	Design document definition (JSON)
Response Data	Success and confirmed design document ID
Authentication Required	optional
Return Codes	
200	Request completed successfully.
401	The item requested was not available using the supplied authorization, or authorization was not supplied.
404	The requested content could not be found. The returned content will include further information, as a JSON object, if available.

Deleting a design document immediately invalidates the design document and all views and indexes associated with it. The indexes and stored data on disk are removed in the background.

For example, to delete the previously created design document using `curl` :

```
> curl -v -X DELETE -H 'Content-Type: application/json' \
    'http://Administrator:Password@192.168.0.77:8092/default/_design/
dev_byfield'
```

When the design document has been successfully removed, the JSON returned indicates successful completion, and confirmation of the design document removed:

```
{"ok":true,"id":"_design/dev_byfield"}
```

Error conditions will be returned if the authorization is incorrect, or the specified design document cannot be found.

Related Links

Views REST API on page 475

Querying views

Querying can be performed through the REST API endpoint. The REST API supports and operates using the core HTTP protocol, and this is the same system used by the client libraries to obtain the view data.

Use the REST API to query a view by accessing any node within the Couchbase Server cluster on port 8092. For example:

```
GET http://localhost:8092/bucketname/_design/designdocname/_view/viewname
```

Where:

- `bucketname` is the name of the bucket.
- `designdocname` is the name of the design document that contains the view.

For views defined within the development context, the `designdocname` is prefixed with `dev_`. For example, the design document `beer` is accessible as a development view using `dev_beer`.

Production views are accessible using their name only.

- `viewname` is the name of the corresponding view within the design document.

When accessing a view stored within an SASL password-protected bucket, you must include the bucket name and bucket password within the URL of the request:

```
GET http://bucketname:password@localhost:8092/bucketname/_design/
designdocname/_view/viewname
```

Additional arguments to the URL request can be used to select information from the view, and provide limit, sorting and other options. For example, to output only ten items:

```
GET http://localhost:8092/bucketname/_design/designdocname/_view/viewname?
limit=10
```

The formatting of the URL follows the HTTP specification. The first argument should be separated from the base URL using a question mark (`?`). Additional arguments should be separated using an ampersand (`&`). Special characters should be quoted or escaped according to the HTTP standard rules.

The additional supported arguments are detailed in the following table

Get View Name	Description
Method	`GET /bucket/_design/design-doc/_view/view-name`
Request Data	None
Response Data	JSON of the rows returned by the view
Authentication Required	no
Query Arguments	
`descending`	Return the documents in descending by key order
	Parameters : boolean; optional
`endkey`	Stop returning records when the specified key is reached. Key must be specified as a JSON value.
	Parameters : string; optional
`endkey_docid`	Stop returning records when the specified document ID is reached
	Parameters : string; optional
`full_set`	Use the full cluster data set (development views only).
	Parameters : boolean; optional

Get View Name	Description
`group`	Group the results using the reduce function to a group or single row
	Parameters : boolean; optional
`group_level`	Specify the group level to be used
	Parameters : numeric; optional
`inclusive_end`	Specifies whether the specified end key should be included in the result
	Parameters : boolean; optional
`key`	Return only documents that match the specified key. Key must be specified as a JSON value.
	Parameters : string; optional
`keys`	Return only documents that match each of keys specified within the given array. Key must be specified as a JSON value. Sorting is not applied when using this option.
	Parameters : array; optional
`limit`	Limit the number of the returned documents to the specified number
	Parameters : numeric; optional
`on_error`	Sets the response in the event of an error
	Parameters : string; optional
	Supported Values
	`continue` : Continue to generate view information in the event of an error, including the error information in the view response stream.
	`stop` : Stop immediately when an error condition occurs. No further view information will be returned.
`reduce`	Use the reduction function
	Parameters : boolean; optional
`skip`	Skip this number of records before starting to return the results
	Parameters : numeric; optional
`stale`	Allow the results from a stale view to be used
	Parameters : string; optional
	Supported Values :
	`false` : Force a view update before returning data
	`ok` : Allow stale views
	`update_after` : Allow stale view, update view after it has been accessed
`startkey`	Return records with a value equal to or greater than the specified key. Key must be specified as a JSON value.
	Parameters : string; optional
`startkey_docid`	Return records starting with the specified document ID
	Parameters : string; optional

The output from a view will be a JSON structure containing information about the number of rows in the view, and the individual view information.

An example of the View result is shown below:

```
{
  "total_rows": 576,
  "rows" : [
      {"value" : 13000, "id" : "James", "key" : ["James", "Paris"] },
      {"value" : 20000, "id" : "James", "key" : ["James", "Tokyo"] },
      {"value" : 5000,  "id" : "James", "key" : ["James", "Paris"] },
…
    ]
}
```

The JSON returned consists of two fields:

- `total_rows`

A count of the number of rows of information within the stored View. This shows the number of rows in the full View index, not the number of rows in the returned data set.

- `rows`

An array, with each element of the array containing the returned view data, consisting of the value, document ID that generated the row, and the key.

In the event of an error, the HTTP response will be an error type (not 200), and a JSON structure will be returned containing two fields, the basic `error` and a more detailed `reason` field. For example:

```
{
  "error":"bad_request",
  "reason":"invalid UTF-8 JSON: {{error,{1,\"lexical error: invalid char in
 json text.\\n\"}},\n                      \"Paris\"}"
}
```

If you supply incorrect parameters to the query, an error message is returned by the server. Within the Client Libraries the precise behavior may differ between individual language implementations, but in all cases, an invalid query should trigger an appropriate error or exception.

Related Links

Views REST API on page 475

Related topics

Views and indexes on page 183

Views requests

You can use the `/internalSettings` endpoint to limit the number of simultaneous requests each node can accept. In earlier releases, too many simultaneous views requests resulted in a node being overwhelmed.

When Couchbase Server rejects an incoming connection because one of these limits is exceeded, it responds with an HTTP status code of 503. The HTTP Retry-After header will be set appropriately. If the request is made to a REST port, the response body will provide the reason why the request was rejected. If the request is made on a CAPI port, such as a views request, the server will respond with a JSON object with a “error” and “reason” fields.

For example, to change this limit for the port used for views:

```
curl -X POST -u admin:password http://localhost:8091/internalSettings -d
 'capiRequestLimit=50'
```

Replace the *admin*, *password*, *localhost*, and *50* values in the above example with your actual values.

Will limit the number of simultaneous views requests and internal XDCR requests which can be made on a port. The following are all the port-related request parameters you can set:

- **restRequestLimit** : Maximum number of simultaneous connections each node should accept on a REST port. Diagnostic-related requests and `/internalSettings` requests are not counted in this limit.
- **capiRequestLimit** : Maximum number of simultaneous connections each node should accept on CAPI port. This port is used for XDCR and views connections.

- **dropRequestMemoryThresholdMiB** : In MB. The amount of memory used by Erlang VM that should not be exceeded. If the amount is exceeded the server will start dropping incoming connections.

By default these settings do not have any limit set. We recommend you leave this settings at the default setting unless you experience issues with too many requests impacting a node. If you set these thresholds too low, too many requests will be rejected by the server, including requests from Couchbase Web Console.

Related Links

Views REST API on page 475

Related topics

Buckets REST API on page 456
The bucket management and configuration REST API endpoints provide a fine level of control over the individual buckets in the cluster, their configuration, and specific operations.

XDCR REST API

Cross Datacenter Replication (XDCR) enables you to automatically replicate data between clusters and between data buckets.

When you use XDCR, you specify source and destination clusters. A source cluster is the cluster from which you want to copy data. A destination cluster is the cluster where you want the replica data to be stored. When you configure replication, you specify your selections for an individual cluster using Couchbase Admin Console. XDCR replicates data between specific buckets and specific clusters and you can configure replication be either uni-directional or bi-directional. Uni-directional replication means that XDCR replicates from a source to a destination; in contrast, bi-directional replication means that XDCR replicates from a source to a destination and also replicates from the destination to the source.

Related Links

Couchbase REST API on page 425

Creating a destination cluster reference on page 483

Creating XDCR replications on page 484

Managing XDCR data encryption on page 485

Deleting a destination cluster reference on page 486

Deleting XDCR replications on page 487

Changing internal XDCR settings on page 487
The XDCR advanced and internal settings change the replication behavior, performance, and timing.

Viewing internal XDCR settings on page 489

Getting a destination cluster reference on page 489

Getting XDCR stats via REST on page 490

Creating a destination cluster reference

When you use XDCR, you establish *source* and *destination* cluster. A source cluster is the cluster from which you want to copy data; a destination cluster is the cluster where you want the replica data to be stored. To create a reference to a destination cluster:

```
curl -v -u admin:password1 http://10.4.2.4:8091/pools/default/remoteClusters \
-d uuid=9eee38236f3bf28406920213d93981a3  \
-d name=remote1
-d hostname=10.4.2.6:8091
-d username=admin -d password=password2
```

Replace the *admin*, *password1*, *password2*, *10.4.2.4*, *10.4.2.6*, *9eee38236f3bf28406920213d93981a3*, and *remote1* values in the above example with your actual values.

You provide credentials for the source cluster and information, including credentials and UUID for destination cluster. This will generate a request similar to the following sample:

```
POST /pools/default/remoteClusters HTTP/1.1
Authorization: Basic QWRtaW5pc3RyYXRvcjpwYXNzd29yZA==
User-Agent: curl/7.21.4 (universal-apple-darwin11.0) libcurl/7.21.4
 OpenSSL/0.9.8r zlib/1.2.5
Host: 10.4.2.4:8091
Accept: */*
Content-Length: 114
Content-Type: application/x-www-form-urlencoded
```

If successful, Couchbase Server will respond with a JSON response similar to the following:

```
{"name":"remote1","uri":"/pools/default/remoteClusters/remote1",
"validateURI":"/pools/default/remoteClusters/remote1?just_validate=1",
"hostname":"10.4.2.6:8091",
"username":"Administrator",
"uuid":"9eee38236f3bf28406920213d93981a3",
```

```
"deleted":false}
```

The following describes the response elements:

- (String) name: Name of the destination cluster referenced for XDCR.
- (String) validateURI: URI to validate details of cluster reference.
- (String) hostname: Hostname/IP (and :port) of the remote cluster.
- (String) username: Username for the destination cluster administrator.
- (String) uuid: UUID of the remote cluster reference.
- (Boolean) deleted: Indicates whether the reference to the destination cluster has been deleted or not.

For more information about XDCR and creating references to destination clusters via the Couchbase Web Console.

Related Links

XDCR REST API on page 483
Cross Datacenter Replication (XDCR) enables you to automatically replicate data between clusters and between data buckets.

Related topics

Configuring XDCR replications on page 169
Configuration of XDCR replications is done on a per-bucket basis.

Creating XDCR replications

To replicate data to an established destination cluster from a source cluster, you can use the REST API or Couchbase Web Console. Once you create a replication it will automatically begin between the clusters. As a REST call:

```
curl -v -X POST -u admin:password1 http://10.4.2.4:8091/controller/
createReplication
-d uuid=9eee38236f3bf28406920213d93981a3
-d fromBucket=beer-sample
-d toCluster=remote1
-d toBucket=remote_beer
-d replicationType=continuous
```

Replace the *admin*, *password1*, *10.4.2.4*, *9eee38236f3bf28406920213d93981a3*, *beer-sample*, *remote1*, *remote_beer*, and *continuous* values in the above example with your actual values.

This will send a request similar to the following example:

```
POST / HTTP/1.1
Authorization: Basic QWRtaW5pc3RyYXRvcjpwYXNzd29yZDE=
User-Agent: curl/7.21.4 (universal-apple-darwin11.0) libcurl/7.21.4
 OpenSSL/0.9.8r zlib/1.2.5
Host: 10.4.2.4:8091
Accept: */*
Content-Length: 126
Content-Type: application/x-www-form-urlencoded
```

If Couchbase Server successfully creates the replication, it will immediately begin replicating data from the source to destination cluster. You will get a response similar to the following JSON:

```
{
  "id": "9eee38236f3bf28406920213d93981a3/beer-sample/remote_beer",
  "database": "http://10.4.2.4:8092/_replicator"
}
```

The unique document ID returned in the JSON is a reference you can use if you want to delete the replication.

Related Links

XDCR REST API on page 483
Cross Datacenter Replication (XDCR) enables you to automatically replicate data between clusters and between data buckets.

Related topics

Configuring XDCR replications on page 169
Configuration of XDCR replications is done on a per-bucket basis.

Managing XDCR data encryption

The process for configuring XDCR with data encryption (Enterprise Edition only) involves configuring the XDCR cluster reference with data encryption enabled, providing the SSL certificate, and configuring replication.

The following summarizes the HTTP methods used for defining XDCR data encryption:

HTTP method	URI path	Description
GET	`/pools/default/remoteClusters`	Gets the destination cluster reference
POST	`/pools/default/remoteClusters`	Creates a reference to the destination cluster
PUT	`/pools/default/remoteClusters/UUID`	Modifies the destination cluster reference
DELETE	`/pools/default/remoteClusters/UUID`	Deletes the reference to the destination cluster.

Retrieving certificates

To retrieve the SSL certificate from the destination cluster to the source cluster:

Destination endpoint

`/pools/default/certificate`

Example

```
curl http://remoteHost:port/pools/default/certificate > ./remoteCert.pem
```

Regenerating certificates

To regenerate a certificate on a destination cluster:

Destination endpoint

`/controller/regenerateCertificate`

Example

```
curl -X POST http://Administrator:asdasd@remoteHost:8091/controller/
regenerateCertificate
```

Configuring XDCR with data encryption

A POST to `/pools/default/remoteClusters` creates the XDCR cluster reference from the source cluster to the destination cluster. Setting the `demandEncryption` to one (1) and providing the certificate name and location enables data encryption.

Destination endpoint

`POST /pools/default/remoteClusters` creates the destination cluster reference.

`PUT /pools/default/remoteClusters` modifies the destination cluster reference.

Syntax

```
curl –X POST  -u Admin:myPassword
  http://localHost:port/pools/default/remoteClusters
```

```
  -d name=<clusterName>             // Remote cluster name
  -d hostname=<host>:<port>         // FQDN of the remote host.
  -d username=<adminName>           // Remote cluster Admin name
  -d password=<adminPassword>       // Remote cluster Admin password
  -d demandEncryption=[0|1] --data-urlencode "certificate=$(cat
 remoteCert.pem)"
```

Example

```
curl -X POST
-d name=remoteName
-d hostname=10.3.4.187:8091
-d username=remoteAdmin -d password=remotePassword
-d demandEncryption=1 --data-urlencode "certificate=$(cat remoteCert.pem)"
http://Administrator:asdasd@192.168.0.1:8091/pools/default/remoteClusters/
```

Disabling data encryption

To modify the XDCR configuration so that SSL data encryption is disabled, execute a PUT from the source cluster to the destination cluster with `demandEncryption=0`.

Destination endpoint

```
/pools/default/remoteClusters
```

Example

```
curl -X PUT  -u Admin:myPassword
  http://192.168.0.1:8091/pools/default/remoteClusters/
-d name=remoteName
-d hostname=10.3.4.187:8091
-d username=remoteAdmin -d password=remotePassword
-d demandEncryption=0
```

Related Links

XDCR REST API on page 483
Cross Datacenter Replication (XDCR) enables you to automatically replicate data between clusters and between data buckets.

Deleting a destination cluster reference

You can remove a reference to destination cluster using the REST API. A destination cluster is a cluster to which you replicate data. After you remove it, it will no longer be available for replication via XDCR:

```
curl -v -X DELETE -u admin:password1 10.4.2.4:8091/pools/default/
remoteClusters/remote1
```

Replace the *admin*, *password1*, *10.4.2.4*, and *remote1* values in the above example with your actual values.

This will send a request similar to the following example:

```
DELETE /pools/default/remoteClusters/remote1 HTTP/1.1
Authorization: Basic QWRtaW5pc3RyYXRvcjpwYXNzd29yZDE=
User-Agent: curl/7.21.4 (universal-apple-darwin11.0) libcurl/7.21.4
 OpenSSL/0.9.8r zlib/1.2.5
Host: 10.4.2.4:8091
Accept: */*
```

If successful, Couchbase Server will respond with a 200 response as well as the string, ‘OK’:

```
HTTP/1.1 200 OK
Server: Couchbase Server 2.0.0-1941-rel-community
Pragma: no-cache

....

"ok"
```

Related Links

XDCR REST API on page 483
Cross Datacenter Replication (XDCR) enables you to automatically replicate data between clusters and between data buckets.

Related topics

Configuring XDCR replications on page 169
Configuration of XDCR replications is done on a per-bucket basis.

Deleting XDCR replications

When you delete a replication, it stops replication from the source to the destination. If you re-create the replication between the same source and destination clusters and buckets, it XDCR will resume replication. To delete replication via REST API:

```
curl -u admin:password1  \
http://10.4.2.4:8091/controller/
cancelXDCR/9eee38236f3bf28406920213d93981a3%2Fbeer-sample%2Fremote_beer  \
-X DELETE
```

Replace the *admin*, *password1*, *10.4.2.4*, *9eee38236f3bf28406920213d93981a3*, *beer-sample*, and *remote_beer* values in the above example with your actual values.

You use a URL-encoded endpoint which contains the unique document ID that references the replication. You can also delete a replication using the Couchbase Web Console.

Related Links

XDCR REST API on page 483
Cross Datacenter Replication (XDCR) enables you to automatically replicate data between clusters and between data buckets.

Related topics

Configuring XDCR replications on page 169
Configuration of XDCR replications is done on a per-bucket basis.

Changing internal XDCR settings

The XDCR advanced and internal settings change the replication behavior, performance, and timing.

The URI endpoints are available to change global settings for replications for a cluster and to change settings for a specific replication ID.

Table 19: XDCR URI paths for settings

URI path	Description
/settings/replications/	Global setting supplied to all replications for a cluster.
/settings/replications/replication_id	Settings for specific replication for a bucket.
/internalSettings	Settings applied to all replications for a cluster.

In the REST API you can change these settings globally for all replications for a cluster or for a specific replication ID.

Table 20: XDCR advanced settings

Parameter	Value	Description
`xdcrMaxConcurrentReps`	Integer	Same as Web Console setting `XDCR Max Replications per Bucket`.

Parameter	Value	Description
`xdcrCheckpointInterval`	Integer	Same as Web Console setting`XDCR Checkpoint Interval`.
`xdcrWorkerBatchSize`	Integer	Same as Web Console setting `XDCR Batch Count`.
`xdcrDocBatchSizeKb`	Integer	Same as Web Console setting `XDCR Batch Size (KB)`.
`xdcrFailureRestartInterval`	Integer	Same as Web Console setting`XDCR Failure Retry Interval`.
`xdcrOptimisticReplicationThreshold`	Integer	Same as Web Console setting `XDCR Optimistic Replication Threshold`.

The following settings are additional internal settings for XDCR that are only available via the REST API.

Table 21: XDCR internal settings

Parameter	Value	Description
`workerProcesses`	Integer from 1 to 32. Default 32.	The number of worker processes for each vbucket replicator in XDCR. Setting is available for replications using either memcached or REST for replication.
`httpConnections`	Integer from 1 to 100. Default 2.	Number of maximum simultaneous HTTP connections used for REST protocol.

Modifying the number of parallel replication streams

The following example updates an XDCR setting for parallel replication streams per node:

```
curl -X POST -u admin:password1  \
http://10.4.2.4:8091/settings/replications/  \
-d xdcrMaxConcurrentReps=64
```

Note: Replace the *admin*, *password1*, *10.4.2.4*, and *64* values in the above example with your actual values.

Returns:

If Couchbase Server successfully updates this setting, a response (similar to the following) is sent:

```
HTTP/1.1 200 OK
Server: Couchbase Server 2.0.0-1941-rel-community
Pragma: no-cache
Date: Wed, 28 Nov 2012 18:20:22 GMT
Content-Type: application/json
Content-Length: 188
Cache-Control: no-cache
```

Related Links

XDCR REST API on page 483
Cross Datacenter Replication (XDCR) enables you to automatically replicate data between clusters and between data buckets.

Viewing internal XDCR settings

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

HTTP method and URI path

```
GET /internalSettings
```

Curl syntax

curl --username=ADMIN --password=PASSWORD HOST:PORT/internalSettings

```
curl -u admin:password1  \
    http://10.4.2.4:8091/internalSettings
```

You will receive a response similar to the following. For the sake of brevity, we are showing only the XDCR-related items:

```
{
     ....
    "xdcrMaxConcurrentReps":32,
    "xdcrCheckpointInterval":1800,
    "xdcrWorkerBatchSize":500,
    "xdcrDocBatchSizeKb":2048,
    "xdcrFailureRestartInterval":30,
    "xdcrOptimisticReplicationThreshold":256,
    "xdcrAnticipatoryDelay":0
     ....
}
```

Related Links

XDCR REST API on page 483
Cross Datacenter Replication (XDCR) enables you to automatically replicate data between clusters and between data buckets.

Related topics

XDCR advanced settings on page 288
XDCR advanced settings are internal settings that are available for configuration.

Getting a destination cluster reference

When you use XDCR, you establish *source* and *destination* cluster. A source cluster is the cluster from which you want to copy data; a destination cluster is the cluster where you want the replica data to be stored. To get information about a destination cluster:

```
curl -u admin:password
    http://localhost:8091/pools/default/remoteClusters
```

Replace the *admin*, *password*, and *localhost* values in the above example with your actual values.

You provide credentials for the cluster and also the hostname and port for the remote cluster. This will generate a request similar to the following sample:

```
GET /pools/default/remoteClusters HTTP/1.1
Authorization: Basic QWRtaW5pc3RyYXRvcjpwYXNzd29yZA==
User-Agent: curl/7.21.4 (universal-apple-darwin11.0) libcurl/7.21.4
 OpenSSL/0.9.8r zlib/1.2.5
```

```
Host: 10.4.2.4:8091
Accept: */*
```

If successful, Couchbase Server responds with a JSON response similar to the following:

```
[{
"name":"remote1",
"uri":"/pools/default/remoteClusters/remote1",
"validateURI":"/pools/default/remoteClusters/remote1?just_validate=1",
"hostname":"10.4.2.6:8091",
"username":"Administrator",
"uuid":"9eee38236f3bf28406920213d93981a3",
"deleted":false
}]
```

The following describes the response elements:

- (String) name: Name of the destination cluster referenced for XDCR.
- (String) uri: URI for destination cluster information.
- (String) validateURI: URI to validate details of cluster reference.
- (String) hostname: Hostname/IP (and :port) of the remote cluster.
- (String) uuid: UUID of the remote cluster reference.
- (String) username: Username for the destination cluster administrator.
- (Boolean) deleted: Indicates whether the reference to the destination cluster has been deleted or not.

Related Links

XDCR REST API on page 483
Cross Datacenter Replication (XDCR) enables you to automatically replicate data between clusters and between data buckets.

Related topics

Configuring XDCR replications on page 169
Configuration of XDCR replications is done on a per-bucket basis.

Getting XDCR stats via REST

You can get XDCR statistics from either Couchbase Web Console, or the REST API. You perform all of these requests on a source cluster to get information about a destination cluster. All of these requests use the UUID, a unique identifier for destination cluster. You can get this ID by using the REST API if you do not already have it. The endpoints are as follows:

```
http://hostname:port/pools/default/buckets/[bucket_name]/stats/
[destination_endpoint]

# where a possible [destination endpoint] includes:

# number of documents written to destination cluster via XDCR
replications/[UUID]/[source_bucket]/[destination_bucket]/docs_written

# size of data replicated in bytes
replications/[UUID]/[source_bucket]/[destination_bucket]/data_replicated

# number of updates still pending replication
replications/[UUID]/[source_bucket]/[destination_bucket]/changes_left

# number of documents checked for changes
replications/[UUID]/[source_bucket]/[destination_bucket]/docs_checked

# number of checkpoints issued in replication queue
replications/[UUID]/[source_bucket]/[destination_bucket]/num_checkpoints

# number of checkpoints failed during replication
```

```
replications/[UUID]/[source_bucket]/[destination_bucket]/num_failedckpts

# size of replication queue in bytes
replications/[UUID]/[source_bucket]/[destination_bucket]/size_rep_queue

# active vBucket replicators
replications/[UUID]/[source_bucket]/[destination_bucket]/active_vbreps

# waiting vBucket replicators
replications/[UUID]/[source_bucket]/[destination_bucket]/waiting_vbreps

# seconds elapsed during replication
replications/[UUID]/[source_bucket]/[destination_bucket]/time_committing

# time working in seconds including wait time
replications/[UUID]/[source_bucket]/[destination_bucket]/time_working

# bandwidth used during replication
replications/[UUID]/[source_bucket]/[destination_bucket]/bandwidth_usage

# aggregate time waiting to send changes to destination cluster in
 milliseconds
# weighted average latency for sending replicated changes to destination
 cluster
replications/[UUID]/[source_bucket]/[destination_bucket]/docs_latency_aggr
replications/[UUID]/[source_bucket]/[destination_bucket]/docs_latency_wt

# Number of documents in replication queue
replications/[UUID]/[source_bucket]/[destination_bucket]/docs_rep_queue

# aggregate time to request and receive metadata about documents
# weighted average time for requesting document metadata
# XDCR uses this for conflict resolution prior to sending document into
 replication queue
replications/[UUID]/[source_bucket]/[destination_bucket]/meta_latency_aggr
replications/[UUID]/[source_bucket]/[destination_bucket]/meta_latency_wt

# bytes replicated per second
replications/[UUID]/[source_bucket]/[destination_bucket]/rate_replication
# number of docs sent optimistically
replications/[UUID]/[source_bucket]/[destination_bucket]/docs_opt_repd
```

You need to provide properly URL-encoded `/[UUID]/[source_bucket]/[destination_bucket]/[stat_name]`. To get the number of documents written:

```
curl -u admin:password http://localhost:8091/pools/default/buckets/default/
stats/replications%2F8ba6870d88cd72b3f1db113fc8aee675%2Fsource_bucket
%2Fdestination_bucket%2Fdocs_written
```

Replace the *admin*, *password*, *localhost*, *8ba6870d88cd72b3f1db113fc8aee675*, *source_bucket*, and *destination_bucket* values in the above example with your actual values.

The above command produces the following output:


```
{"samplesCount":60,"isPersistent":true,"lastTStamp":1371685106753,"interval":1000,
"timestamp":
[1371685048753,1371685049754,1371685050753,1371685051753,1371685052753,1371685053753,1371
1371685055753,1371685056753,1371685057753,1371685058752,1371685059753,1371685060753,13716
1371685063753,1371685064753,1371685065753,1371685066753,1371685067753,1371685068753,13716
1371685071753,1371685072753,1371685073753,1371685074753,1371685075753,1371685076753,13716
1371685079753,1371685080753,1371685081753,1371685082753,1371685083753,1371685084753,13716
1371685087753,1371685088753,1371685089753,1371685090753,1371685091754,1371685092753,13716
1371685095753,1371685096753,1371685097753,1371685098753,1371685099753,1371685100753,13716
1371685103753,1371685104753,1371685105753,1371685106753],
```

```
"nodeStats":{"127.0.0.1:8091":
[1000000,1000000,1000000,1000000,1000000,1000000,1000000,1000000,1000000,1000000,
1000000,1000000,1000000,1000000,1000000,1000000,1000000,1000000,1000000,1000000,1000000,1
1000000,1000000,1000000,1000000,1000000,1000000,1000000,1000000,1000000,1000000,1000000,1
1000000,1000000,1000000,1000000,1000000,1000000,1000000,1000000,1000000,1000000,1000000,1
1000000,1000000,1000000,1000000,1000000,1000000,1000000]}}
```

This shows that XDCR transferred 1 million documents at each of the timestamps provided. To get the rate of replication, make this REST request:

```
curl -u admin:password http://localhost:8091/pools/default/buckets/default/
stats/replications%2F8ba6870d88cd72b3f1db113fc8aee675%2Fsource_bucket
%2Fdestination_bucket%2Frate_replication
```

Replace the *admin*, *password*, *localhost*, *8ba6870d88cd72b3f1db113fc8aee675*, *source_bucket*, and *destination_bucket* values in the above example with your actual values.

This will produce the following output:

```
{"samplesCount":60,"isPersistent":true,"lastTStamp":1371685006753,"interval":1000,
"timestamp":
[1371684948753,1371684949753,1371684950753,1371684951753,1371684952753,1371684953753,1371
1371684955754,1371684956753,1371684957753,1371684958753,1371684959753,1371684960753,1371
1371684963753,1371684964753,1371684965753,1371684966753,1371684967753,1371684968752,1371
1371684971753,1371684972753,1371684973753,1371684974753,1371684975753,1371684976753,1371
1371684979753,1371684980753,1371684981753,1371684982753,1371684983753,1371684984753,1371
1371684987754,1371684988753,1371684989753,1371684990753,1371684991753,1371684992753,1371
1371684995753,1371684996753,1371684997753,1371684998776,1371684999753,1371685000753,1371
1371685003753,1371685004753,1371685005753,1371685006753],
"nodeStats":{"127.0.0.1:8091":
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
```

To get `docs_opt_repd` you first get the replication id for a source and destination bucket. First get a list active tasks for a cluster:

```
    curl -s -u admin:password \
    http://localhost:8091/pools/default/tasks
```

Replace the *admin*, *password*, and *localhost* values in the above example with your actual values.

This will result in output as follows:

```
    ....
    "id": "def03dbf5e968a47309194ebe052ed21\/bucket_source\/
bucket_destination",
    "source": "bucket_source",
    "target":"\/remoteClusters\/def03dbf5e968a47309194ebe052ed21\/buckets\/
bucket_name",
    "continuous": true,
    "type": "xdcr",
    ....
```

With this replication id you can get a sampling of stats for `docs_opt_repd`:

```
http://10.3.121.119:8091/pools/default/buckets/default/stats/ \
replications%2fdef03dbf5e968a47309194ebe052ed21%2fbucket_source
%2fbucket_destination%2fdocs_opt_repd
```

This results in output similar to the following:

```
    {
       "samplesCount":60,
       "isPersistent":true,
       "lastTStamp":1378398438975,
       "interval":1000,
       "timestamp":[
          1378398380976,
          1378398381976,
```

```
          ....
```

You can also see the incoming write operations that occur on a destination cluster due to replication via XDCR. For this REST request, you need to make the request on your destination cluster at the following endpoint:

```
http://[Destination_IP]:8091/pools/default/buckets/[bucket_name]/stats
```

This returns results for all stats as follows. Within the JSON you find an array `xdc_ops` and the value for this attribute will be the last sampling of write operations on the destination due to XDCR:

```
{
.................
"xdc_ops":[0.0,0.0,0.0,0.0,633.3666333666333,1687.6876876876877, \
2610.3896103896104,3254.254254254254,3861.138861138861,4420.420420420421, \
................
}
```

Many of these statistics are exposed in the Couchbase Web Console.

Related Links

XDCR REST API on page 483
Cross Datacenter Replication (XDCR) enables you to automatically replicate data between clusters and between data buckets.

Related topics

Data Buckets on page 259

XDCR on page 283
The XDCR panel is used to create a remote cluster reference and specify replication.

Compaction REST API

Couchbase Server writes all data that you append, update and delete as files on disk. This process can eventually lead to gaps in the data file, particularly when you delete data. Be aware the server also writes index files in a sequential format based on appending new results in the index. You can reclaim the empty gaps in all data files by performing a process called compaction. In both the case of data files and index files, you will want to perform frequent compaction of the files on disk to help reclaim disk space and reduce disk fragmentation.

Compacting bucket data and indexes

To compact data files for a given bucket as well as any indexes associated with that bucket, you make this request:

```
curl -i -v -X POST -u admin:password http://localhost:8091/pools/default/
buckets/bucket_name/controller/compactBucket
```

Replace the *admin*, *password*, *localhost*, and *bucket_name* values in the above example with your actual values.

Where you provide the ip and port for a node that accesses the bucket as well as the bucket name. You need to provide administrative credentials for that node in the cluster. To stop bucket compaction, issue this request:

```
curl -i -v -X POST -u admin:password http://localhost:8091/pools/default/
buckets/bucket_name/controller/cancelBucketCompaction
```

Replace the *admin*, *password*, *localhost*, and *bucket_name* values in the above example with your actual values.

Compacting spatial views

If you have spatial views in your dataset, these are not automatically compacted with data and indexes. Instead, you must manually compact each spatial view through the REST API.

To do this, you must call the spatial compaction endpoint:

```
http://127.0.0.1:9500/BUCKETNAME/_design/DDOCNAME/_spatial/_compact
```

This URL contains the following special information:

- `127.0.0.1:9500`

The port number, 9500, is unique to the spatial indexing system.

- `BUCKETNAME`

The `BUCKETNAME` is the name of the bucket in which the design document is configured.

- `DDOCNAME`

The name of the design document that contains the spatial index or indexes that you want to compact.

For example, you can send a request using `curl` :

```
curl -u admin:password -X POST \
'http://localhost:8091/default/_design/dev_test_spatial_compaction/_spatial/
_compact'
-H 'Content-type: application/json'
```

Replace the *admin*, *password*, and *localhost* values in the above example with your actual values.

Setting auto-compaction

In Couchbase Server you can also provide auto-compaction settings which will trigger data and view compaction based on certain settings. These settings can be made for an entire cluster or for a bucket in a cluster.

Auto-compaction API

REST API	Description
POST /controller/setAutoCompaction	Set cluster-wide auto-compaction intervals and thresholds
GET /settings/autoCompaction	Read cluster-wide settings for auto-compaction
GET /pools/default/buckets/*bucket_name*	Read auto-compaction settings for named bucket
POST/pools/default/buckets/*bucket_name*	Set auto-compaction interval or thresholds for named bucket

Auto-compaction parameters

You can use the following parameters for global auto-compaction settings which apply to all buckets in a cluster at `/controller/setAutoCompaction`. You also use these at `/pools/default/buckets/<bucket_name>` for bucket-level auto-compaction. You will need to provide administrative credentials to change these settings.

You can provide a purge interval to remove the key and metadata for items that have been deleted or are expired. This is known as tombstone purging.

Parameter	Value	Notes
databaseFragmentationThreshold[percentage]	Integer between 2 and 100	Percentage disk fragmentation for data
databaseFragmentationThreshold[size]	Integer greater than 1	Bytes of disk fragmentation for data
viewFragmentationThreshold[percentage]	Integer between 2 and 100	Percentage disk fragmentation for index
viewFragmentationThreshold[size]	Integer greater than 1	Bytes of disk fragmentation for index
parallelDBAndViewCompaction	True or false	Run index and data compaction in parallel
allowedTimePeriod[fromHour]	Integer between 0 and 23	Compaction can occur from this hour onward
allowedTimePeriod[toHour]	Integer between 0 and 23	Compaction can occur up to this hour
allowedTimePeriod[fromMinute]	Integer between 0 and 59	Compaction can occur from this minute onward
allowedTimePeriod[toMinute]	Integer between 0 and 59	Compaction can occur up to this minute
allowedTimePeriod[abortOutside]	True or false	Terminate compaction if process takes longer than allowed time
purgeInterval	Integer between 1 and 60	Number of days a item is deleted or expired. The key and metadata for that item will be purged by auto-compaction

To read current auto-compaction settings for a cluster:

```
curl -u admin:password http://localhost:8091/settings/autoCompaction
```

Replace the *admin*, *password*, and *localhost* values in the above example with your actual values.

This will result in JSON response as follows:

```
{
   "purgeInterval": 3,
   "autoCompactionSettings": {
```

```
            "viewFragmentationThreshold": {
                "size": "undefined",
                "percentage": "undefined"
             },
             "databaseFragmentationThreshold": {
                 "size": "undefined",
                 "percentage": "undefined"
             },
          "parallelDBAndViewCompaction": false
        }
    }
```

This tells us we have a `purgeInterval` of three days and no current thresholds set for data or index compaction. The field `parallelDBAndViewCompaction` set to 'false' indicates the cluster will not perform data and index compaction in parallel. To see auto-compaction settings for a single bucket, use this request:

```
    curl -u admin:password http://localhost:8091/pools/default/buckets/
bucket_name
```

Replace the *admin*, *password*, *localhost*, and *bucket_name* values in the above example with your actual values.

Couchbase Server sends a JSON response with auto-compaction settings for the `bucket_name`:

```
{
    "purgeInterval": 2,
    "autoCompactionSettings": {
        "viewFragmentationThreshold": {
            "size": "undefined",
            "percentage": 30
        },
        "databaseFragmentationThreshold": {
            "size": "undefined",
            "percentage": 30
        },
        "parallelDBAndViewCompaction": true
    }
}
```

This indicates a tombstone `purgeInterval` of two days with a threshold of 30% disk fragmentation for data and views. This means items can be expired for two days or deleted two ago and their tombstones will be purged during the next auto-compaction run.cluster-wide

Related Links

Couchbase REST API on page 425

Related topics

Tombstone purging on page 104
Tombstones are records of expired or deleted items that include item keys and metadata.

Compaction on page 138
Database and view compaction helps to reclaim disk space and reduce fragmentation.

Logs REST API

This section provides the REST API endpoints for retrieving log and diagnostic information as well as how an SDK can add entries into a log.

HTTP method	URI path	Description
GET	`/diag`	Retrieves log and additional server diagnostic information.
GET	`/sasl_logs`	Retrieves log information.

Retrieving log information

Couchbase Server logs various messages, which are available via the REST API. These log messages are optionally categorized by the module. A generic list of log entries or log entries for a particular category can be retrieved.

Note

If the system is secured, administrator credentials are required to access logs.

To retrieve log and server diagnostic information, perform a GET with the `/diag` endpoint.

```
curl -v -X GET -u Administrator:password
   http://127.0.0.1:8091/diag
```

To retrieve a generic list of logs, perform a GET with the `/sasl_logs` endpoint.

```
curl -v -X GET -u Administrator:password
  http://127.0.0.1:8091/sasl_logs
```

To retrieve a specific log file, perform a GET on the `sasl_logs` endpoint and provide a specific log category.

```
curl -v -X GET -u Administrator:password
 http://127.0.0.1:8091/sasl_logs/<logName>
```

Where the *logName* is one of the following log types:

- babysitter
- couchdb
- debug
- error
- info
- mapreduce_errors
- ssl_proxy
- stats
- view
- xdcr
- xdcr_errors

Example

```
curl -v -X GET -u Administrator:password
 http://127.0.0.1:8091/sasl_logs/ssl_proxy
```

Results

```
* About to connect() to 10.5.2.118 port 8091 (#0)
*   Trying 10.5.2.118... connected
* Connected to 10.5.2.118 (10.5.2.118) port 8091 (#0)
* Server auth using Basic with user 'Administrator'
> GET /sasl_logs/ssl_proxy HTTP/1.1
> Authorization: Basic QWRtaW5pc3RyYXRvcjpwYXNzd29yZA==
```

```
> User-Agent: curl/7.21.4 (x86_64-unknown-linux-gnu) libcurl/7.21.4
 OpenSSL/0.9.8b zlib/1.2.3
> Host: 10.5.2.118:8091
> Accept: */*
>
< HTTP/1.1 200 OK
< Transfer-Encoding: chunked
< Server: Couchbase Server
< Pragma: no-cache
< Date: Thu, 06 Feb 2014 22:50:12 GMT
< Content-Type: text/plain; charset=utf-8
< Cache-Control: no-cache
<
logs_node (ssl_proxy):
-------------------------------
[ns_server:info,2014-01-24T11:25:18.066,nonode@nohost:<0.30.0>:ns_ssl_proxy:init_logging
 up ns_ssl_proxy logging
[error_logger:info,2014-01-24T11:25:18.082,nonode@nohost:error_logger<0.5.0>:ale_error_lo
=========================PROGRESS REPORT=========================
          supervisor: {local,ns_ssl_proxy_sup}
             started: [{pid,<0.64.0>},
                       {name,ns_ssl_proxy_server_sup},
                       {mfargs,{ns_ssl_proxy_server_sup,start_link,[]}},
                       {restart_type,permanent},
                       {shutdown,infinity},
                       {child_type,supervisor}]
```

Creating a client logging interface

Entries can be added to the central log from a custom Couchbase SDK. These entries are typically responses to exceptions such as difficulty handling a server response. For instance, the Web Console uses this functionality to log client error conditions.

To add entries, provide a REST request similar to the following:

```
POST /logClientError
Host: localhost:8091
Authorization: Basic xxxxxxxxxxxxxxxxxxx
Accept: application/json
X-memcachekv-Store-Client-Specification-Version: 0.1
```

```
200 - OK
```

Related Links

Couchbase REST API on page 425

Users REST API

As of Couchbase Server 2.2+ you can create one read-only user with the REST API. For more information about this type of user.

To create a read-only user, you need administrative access:

```
curl -X POST -u admin:password http://localhost:8091/settings/readOnlyUser -d
 username=a_name -d password=a_password
```

Replace the *admin*, *password*, *localhost*, *a_name*, and *a_password* values in the above example with your actual values.

Upon success, you will get this response:

```
success: 200 | []
```

The endpoint has one additional, optional parameter `just_validate=1`. If you provide this in your request the server will validate the username and password for a read-only user but will not actually create the user.

The following are the endpoints, parameters, expected return values and possible errors:

Request	Description	Parameters	Returns	Errors	
POST /settings/readOnlyUser	Create read-only user	username, password, just_validate	success: 200 []	error: 400	{“errors”: {field_name:error_message}}
PUT /settings/readOnlyUser	Change read-only user password	password	success: 200 []	error: 400	{“errors”: {field_name:error_message}}
DELETE /settings/readOnlyUser	Delete user	none	success: 200 []	error: 400	{“errors”: {field_name:error_message}}
GET /settings/readOnlyAdminName	Get the read-only username	none	success: 200 “username”	not found: 404	

A `username` is a UTF–8 string that does not contain spaces, control characters or any of these characters: ()<>@,;:\"/[]?={} characters. Any `password` must be UTF–8 with no control characters and must be at least six characters long.

To change the password for a read-only user:

```
curl -X POST -u admin:password http://localhost:8091/settings/readOnlyUser -d
 username=a_name -d password=new_password
```

To delete this user:

```
curl -X DELETE -u admin:password http://localhost:8091/settings/readOnlyUser
```

To get the read-only username, you can have administrative or read-only permissions:

```
curl -u username:password  http://localhost:8091/settings/readOnlyAdminName
```

Replace the *admin*, *password*, *localhost*, *username*, *a_name*, and *new_password* values in the above examples with your actual values.

This will return a response with the read-only username as payload, `success: 200 | "username"`. If there is no read-only user you will get this error `not found: 404`.

Related Links

Couchbase REST API on page 425

Related topics

Creating a read-only user on page 302

One non-administrative user can be created with read-only access for the Web Console and REST API.

Release notes (beta)

This release note is for the 3.0.0 BETA release. It contains a list of known issues.

Known issues

MB-11643	Incoming workload suffers when XDCR enabled
MB-11642	Intra-replication falling behind under moderate-heavy workload Under a moderate replication workload (20-30K sets per second), the backlog becomes too large resulting in high memory usage.
MB-11606	Error displayed when node did not mark to collect
MB-11597	KV+XDCR System test: Mutation replication rate for uni-xdcr is almost zero(900K items remaining) while another bi-xdcr to same cluster is ~10k ops/sec
MB-11568	IP does not show the same in each step of cluster wide collect info
MB-11434	600-800% CPU consumption by memcached on the source cluster. In XDCR scenarios, the CPU usage for memcached process is more than two times the usage in the previous release. This is due to increased scheduling overhead from the shared thread pool. Workaround: Reduce the number of threads on systems that have more than 30 cores.
MB-11405	~2400% CPU consumption by memcached during ongoing workload with five (5) buckets. In XDCR scenarios, the CPU usage for memcached process is more than two times the usage in the previous release. This is due to increased scheduling overhead from the shared thread pool. Workaround: Reduce the number of threads on systems that have more than 30 cores.
MB-11383	warmup_min_items_threshold setting is not honored correctly in 3.0 warmup
MB-11299	UPR replica streams cannot send items from partial snapshots

Fixed issues

Not applicable