Couchbase Server Manual 2.1.0

Couchbase

Couchbase Server Manual 2.1.0

Abstract

This manual documents the Couchbase Server 2.1.0 series, including installation, monitoring, and administration interface and associated tools.

For the corresponding Moxi product, please use the Moxi 1.8 series. See Moxi 1.8 Manual.

External Community Resources.

Download Couchbase Server 2.1
Couchbase Developer Guide 2.1
Client Libraries
Couchbase Server Forum

Last document update: 05 Sep 2013 23:46; *Document built*: 05 Sep 2013 23:46.

Documentation Availability and Formats.   This documentation is available *online: HTML Online* . For other documentation from Couchbase, see Couchbase Documentation Library

Contact: editors@couchbase.com or couchbase.com

Table of Contents

List of Figures

List of Tables

Preface

This manual documents the Couchbase Server database system, for the 2.1.0 release series. For differences between individual version within this release series, see the version-specific notes throughout the manual.

For more help on specific areas of using Couchbase Server 2.1.0, see:

- Chapter 2, *Installing and Upgrading*

- Chapter 4, *Best Practices*

- Section 5.7, "Backup and Restore"

- Section 5.6, "Failing Over Nodes"

- Section 4.6, "Using Couchbase in the Cloud"

- Chapter 6, *Using the Web Console*

- Section 4.2, "Sizing Guidelines"

- ???

1. Best Practice Guides

The following sections provide information on best practice for different aspects of Couchbase Server 2.1.0.

- Port Exhaustion on Windows *located in* Section 2.2.3, "Microsoft Windows Installation"

- Backup your data before performing an upgrade *located in* Section 2.5, "Upgrading to Couchbase Server 2.1"

- Ensure Capacity for Node Removal *located in* Section 5.8.2.2, "Removing a Node from a Cluster"

- Failed Over Nodes *located in* Section 5.8.3, "Swap Rebalance"

- Default Bucket Should Only for Testing *located in* Section 6.3.1.1, "Creating a New Bucket"

- Enable Parallel Compaction *located in* Section 6.8.4, "Enabling Auto-Compaction"

Chapter 1. Introduction to Couchbase Server

Couchbase Server is a NoSQL document database for interactive web applications. It has a flexible data model, is easily scalable, provides consistent high performance and is 'always-on,' meaning it is can serve application data 24 hours, 7 days a week. Couchbase Server provides the following benefits:

- **Flexible Data Model**

  With Couchbase Server, you use JSON documents to represent application objects and the relationships between objects. This document model is flexible enough so that you can change application objects without having to migrate the database schema, or plan for significant application downtime. Even the same type of object in your application can have a different data structures. For instance, you can initially represent a user name as a single document field. You can later structure a user document so that the first name and last name are separate fields in the JSON document without any downtime, and without having to update all user documents in the system.

  The other advantage in a flexible, document-based data model is that it is well suited to representing real-world items and how you want to represent them. JSON documents support nested structures, as well as field representing relationships between items which enable you to realistically represent objects in your application.

- **Easy Scalability**

  It is easy to scale your application with Couchbase Server, both within a cluster of servers and between clusters at different data centers. You can add additional instances of Couchbase Server to address additional users and growth in application data without any interruptions or changes in your application code. With one click of a button, you can rapidly grow your cluster of Couchbase Servers to handle additional workload and keep data evenly distributed.

  Couchbase Server provides automatic sharding of data and rebalancing at runtime; this lets you resize your server cluster on demand. Cross-data center replication providing in Couchbase Server 2.0 enables you to move data closer to your user at other data centers.

- **Consistent High Performance**

  Couchbase Server is designed for massively concurrent data use and consistent high throughput. It provides consistent sub-millisecond response times which help ensure an enjoyable experience for users of your application. By providing consistent, high data throughput, Couchbase Server enables you to support more users with less servers. The server also automatically spreads workload across all servers to maintain consistent performance and reduce bottlenecks at any given server in a cluster.

- **"Always Online"**

  Couchbase Server provides consistent sub-millisecond response times which help ensure an enjoyable experience for users of your application. By providing consistent, high data throughput, Couchbase Server enables you to support more users with less servers. The server also automatically spreads workload across all servers to maintain consistent performance and reduce bottlenecks at any given server in a cluster.

  Features such as cross-data center replication and auto-failover help ensure availability of data during server or datacenter failure.

All of these features of Couchbase Server enable development of web applications where low–latency and high throughput are required by end users. Web applications can quickly access the right information within a Couchbase cluster and developers can rapidly scale up their web applications by adding servers.

1.1. Couchbase Server and NoSQL

NoSQL databases are characterized by their ability to store data without first requiring one to define a database schema. In Couchbase Server, you can store data as key-value pairs or JSON documents. Data does not need to confirm to a rigid,

pre-defined schema from the perspective of the database management system. Due to this schema-less nature, Couchbase Server supports a *scale out* approach to growth, increasing data and I/O capacity by adding more servers to a cluster; and without any change to application software. In contrast, relational database management systems *scale up* by adding more capacity including CPU, memory and disk to accommodate growth.

Relational databases store information in relations which must be defined, or modified, before data can be stored. A relation is simply a table of rows, where each row in a given relation has a fixed set of columns. These columns are consistent across each row in a relation. Tables can be further connected through cross-table references. One table, could hold rows of all individual citizens residing in a town. Another table, could have rows consisting of parent, child and relationship fields. The first two fields could be references to rows in the citizens table while the third field describes the parental relationship between the persons in the first two fields such as father or mother.

1.2. Architecture and Concepts

In order to understand the structure and layout of Couchbase Server, you first need to understand the different components and systems that make up both an individual Couchbase Server instance, and the components and systems that work together to make up the Couchbase Cluster as a whole.

The following section provides key information and concepts that you need to understand the fast and elastic nature of the Couchbase Server database, and how some of the components work together to support a highly available and high performance database.

1.2.1. Nodes and Clusters

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

In this description:

- **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 ore 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 is stored across the entire cluster through the vBucket system.

  Clusters operate in a completely horizontal fashion. To increase the size of a cluster, you 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 the performance and scalability.

1.2.2. Cluster Manager

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 (see Section 1.2.16, "Administration Tools") on a dedicated network port , and through dedicated network ports for client access. Additional ports are configured for inter-node communication.

1.2.3. Data Storage

Couchbase Server provides data management services using *buckets*; these are isolated virtual containers for data. A bucket is a logical grouping of physical resources within a cluster of Couchbase Servers. They can be used by multiple client applications across a cluster. Buckets provide a secure mechanism for organizing, managing, and analyzing data storage resources.

There are two types of data bucket in Couchbase Server: 1) memcached buckets, and 2) couchbase buckets. The two different types of buckets enable you to store data in-memory only, or to store data in-memory as well as on disk for added reliability. When you set up Couchbase Server you can choose what type of bucket you need in your implementation:

Table 1.1. Couchbase Introduction — Bucket Types

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

Table 1.2. Couchbase Introduction — Bucket Capabilities

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.

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

Table 1.3. Couchbase Introduction - Bucket Type Capability Comparison

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

There are three bucket interface types that can be be configured:

- The 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 gives you the option to access isolated buckets using either the binary protocol with SASL authentication, or the ASCII protocol with no authentication

Couchbase Server enables you to use and mix different types of buckets, Couchbase and Memcached, as appropriate 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.

For more information about creating and managing buckets, see the following resources:

- Bucket RAM Quotas: see Section 1.2.4, "RAM Quotas".

- Creating and Managing Buckets with Couchbase Web Console: see Section 6.3, "Viewing Data Buckets".

- Creating and Managing Buckets with Couchbase REST-API: see Section 8.6, "Managing Buckets".

- Creating and Managing Buckets with Couchbase CLI (Command-Line Tool): see Section 7.4, "**couchbase-cli** Tool".

1.2.4. RAM Quotas

RAM is allocated to Couchbase Server in two different configurable quantities, the *Server Quota* and *Bucket Quota*. For more information about creating and changing these two settings, see ??? and Section 6.3.1, "Creating and Editing Data Buckets".

- **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, ther 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.

Figure 1.1. Couchbase Introduction — RAM Quotas
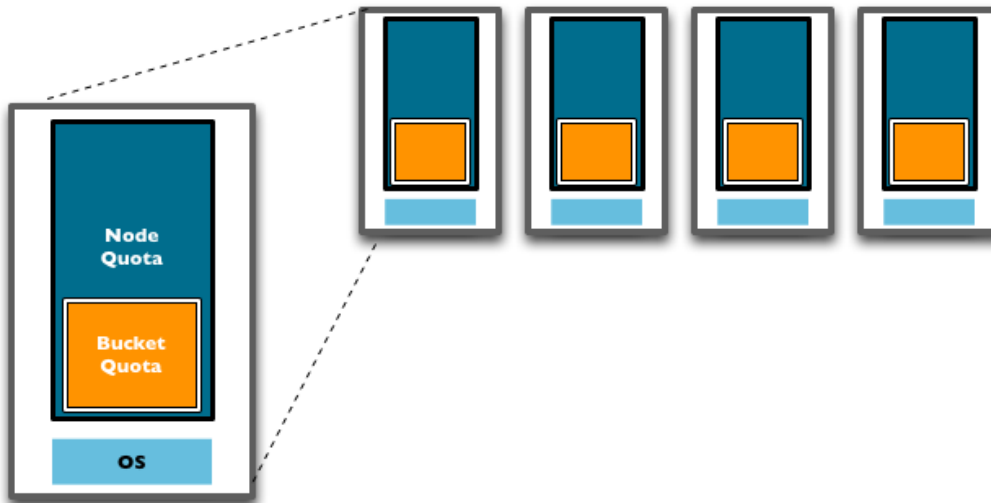
From this description and diagram, you can see that adding new nodes to the cluster expands the overall RAM quota, and the bucket quota, increasing the amount of information that can be kept in RAM.

The Bucket Quota is used by the system to determine when data should be ejected from memory. Bucket Quotas are dynamically configurable within the limit of your Server Quota, and enable you to individually control the caching of information in memory on a per bucket basis. You can therefore configure different buckets to cope with your required caching RAM allocation requirements.

The Server Quota is also dynamically configurable, but care must be taken to ensure that the nodes in your cluster have the available RAM to support your chosen RAM quota configuration.

For more information on changing Couchbase Quotas, see ???.

1.2.5. vBuckets

A vBucket is defined as the *owner* of a subset of the key space of a Couchbase cluster. These vBuckets are used to allow information to be distributed effectively across the cluster. The vBucket system is used both for distributing data, and for supporting replicas (copies of bucket data) on more than one node.

Clients access the information stored in a bucket by communicating directly with the node response 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 is abstracting the physical toplogy from the logical partitioning of data. This architecture is what gives Coucbase Server the elasticity.

This architecture differs from the method used by **memcached**, which uses client-side key hashes to determine the server from a defined list. This requires active management of the list of servers, and specific hashing algorithms such as Ketama to cope with changes to the topology. The structure is also more flexible and able to cope with changes than the typical sharding arrangement used in an RDBMS environment.

Note

vBuckets are not a user-accessible component, but they are a critical component of Couchbase Server and are vital to the availability support and the elastic nature.

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 diagram below shows how the Key to Server mapping (vBucket map) works. There are three servers in the cluster. A client wants to look up (`get`) the value of KEY. The client first hashes the key to calculate the vBucket which owns KEY. In this example, the hash resolves to vBucket 8 (`vB8`) By examining the vBucket map, the client determines Server C hosts vB8. The `get` operation is sent directly to Server C.

Figure 1.2. Couchbase Introduction — vBucket Mapping
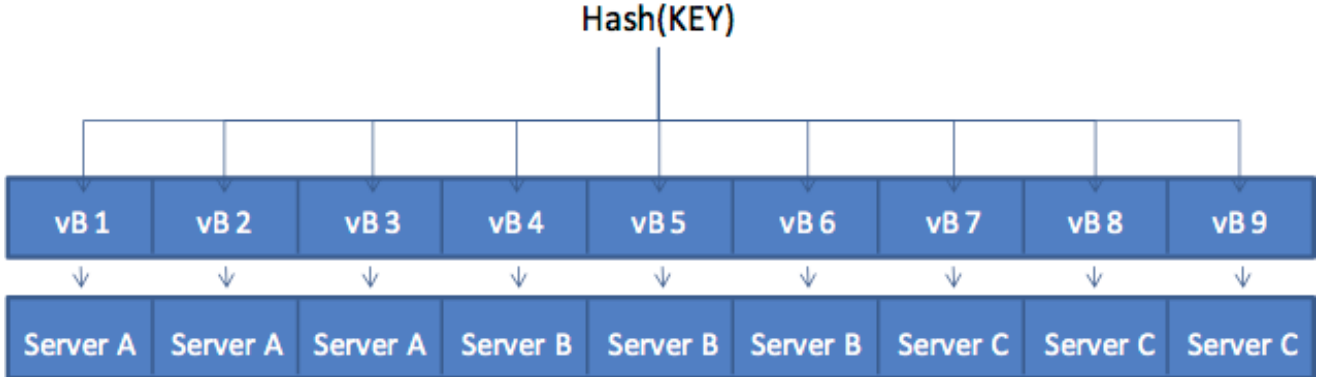
After some period of time, there is a need to add a server to the cluster. A new node, Server D is added to the cluster and the vBucket Map is updated.

Figure 1.3. Couchbase Introduction — vBucket Mapping after Rebalance
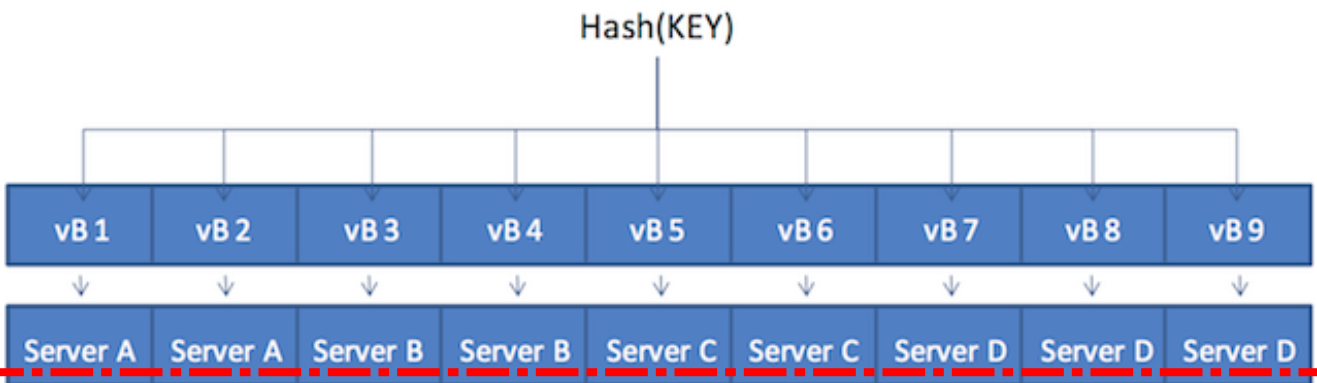
Note

The vBucket map is updated during the rebalance operation; the updated map is then sent the cluster 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 `get` the value of KEY, the hashing algorithm will still resolve to vBucket 8 (`vB8`). The new vBucket map however now maps that vBucket to Server D. The client now communicates directly with Server D to obtain the information.

1.2.6. Caching Layer

The architecture of Couchbase Server includes a built-in caching layer. This caching layer acts as a central part of the server and provides very rapid reads and writes of data. Other database solutions read and write data from disk, which results in much slower performance. One alternative approach is to install and manage a caching layer as a separate component which will work with a database. This approach also has drawbacks because the burden of managing transfer of data between caching layer and database and the burden managing the caching layer results in significant custom code and effort.

In contrast 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 can remove it from RAM to free space for other items. Similarly the server can retrieve infrequently-used items from disk and store them into the caching layer when the items are requested. So the entire process of managing data between the caching layer and data persistence layer is handled entirely by server. In order provide the most frequently-used data while maintaining high performance, Couchbase Server manages a *working set* of your entire information; this set consists of the all data you most frequently access and is kept in RAM for high performance.

Couchbase automatically moves data from RAM to disk asynchronously in the background in order 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 in the background while the server continues to service active requests. During sequences of high writes to the database, clients will be 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, it sends an individual document ID 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 can always return a 'document ID not found' response for an invalid document ID or it can immediately return the data from RAM, or return it after it fetches it from disk.

1.2.7. Disk Storage

For performance, Couchbase Server mainly stores and retrieves information for clients using RAM. At the same time, Couchbase Server will eventually store all data to disk to provide a higher level of reliability. If a node fails and you lose all data in the caching layer, you can still recover items from disk. We call this process of disk storage *eventual persistence* since the server does not block a client while it writes to disk, rather 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 enables you to grow your datasets larger than the built-in caching layer. For more information, see Section 1.2.8, "Ejection, Eviction and Working Set Management".

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 is made to wait until the data has been loaded back into memory before the information is returned.

Multiple Readers and Writers

As of Couchbase Server 2.1, we support multiple readers and writers 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. In order to utilize increased disk speeds and improve the read rate from disk, we now provide multi-threaded readers and writers so that multiple processes can simultaneously read and write data on disk:

Figure 1.4. Multiple Readers and Writers
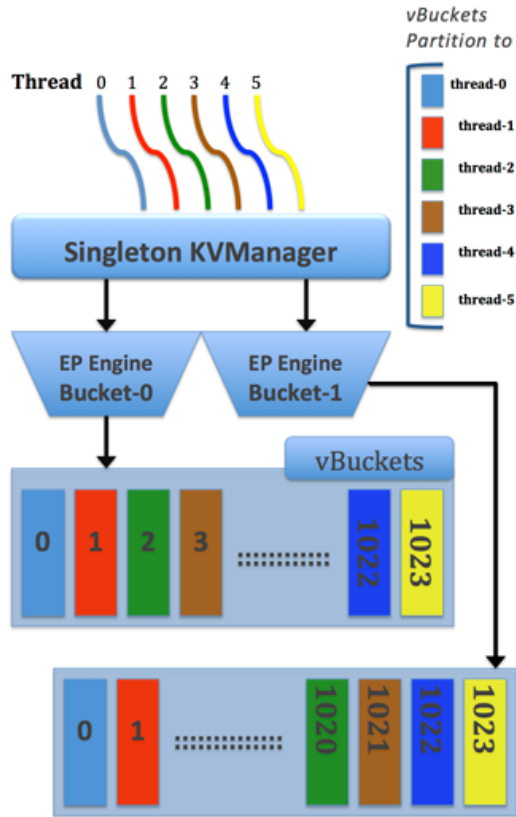
This multi-threaded engine includes additional synchronization among threads that are access the same data cache to avoid conflicts. To maintain performance while avoiding conflicts over data we use a form of locking between threads as well as thread allocation among vBuckets with static partitioning. 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 that access the same vBucket at any given time. We show this in the image above with six pre-allocated threads and two data Buckets. Each thread has the range of vBuckets that is statically partitioned for read and write access.

For information about configuring this option, see Section 5.1, "Using Multi- Readers and Writers".

Document Deletion from Disk

Couchbase Server will never delete 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. For more information about document expiration and deletion, see Couchbase Developer Guide, About Document Expiration .

1.2.8. Ejection, Eviction and Working Set Management

Ejection is a process automatically performed by Couchbase Server; it is the process of removing data from RAM to provide room for frequently-used items. 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. 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*.

In addition to memory quota for the caching layer, there are two watermarks the engine will use 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 will eventually reach `mem_high_wat`. At this point a background job is scheduled to ensure items are migrated to disk and the memory is then available for other Couchbase Server items. This job will run 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 will continue 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 your Couchbase Server Cluster.

Figure 1.5. Architecture — Bucket Configuration
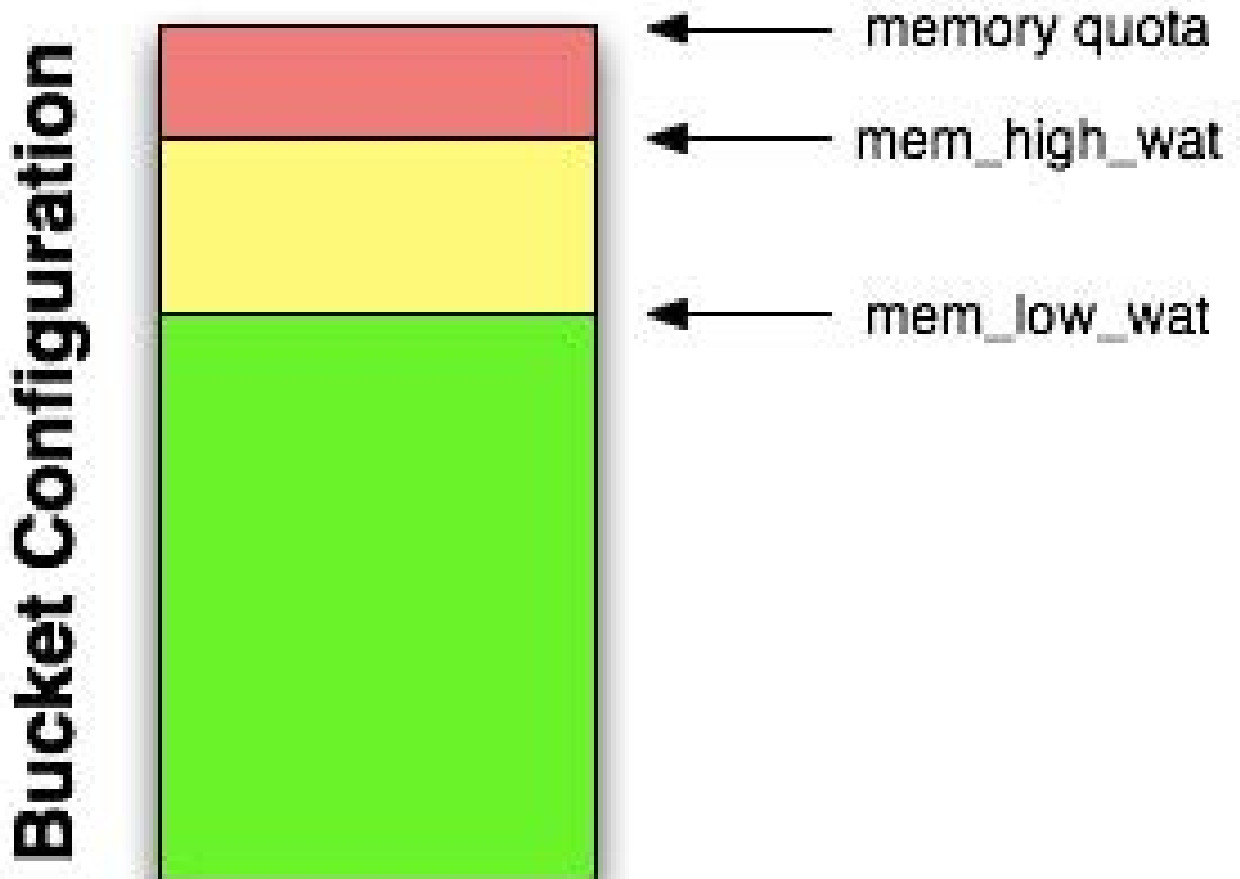
Some of you may be using only memcached buckets with Couchbase Server; in this case the server provides only a caching layer as storage and no data persistence on disk. If your server runs out of space in RAM, it will *evict* items from RAM on a least recently used basis (LRU). Eviction means the server will remove the key, metadata and all other data for the item from RAM. After eviction, the item is irretrievable.

For more detailed technical information about ejection and working set management, including any administrative tasks which impact this process, see Section 5.4, "Ejection and Working Set Management".

1.2.9. Expiration

Each document stored in the database has an optional expiration value (TTL, time to live). The default is for there to be no expiration, i.e. the information will be stored indefinitely. The expiration can be used for data that naturally has a limited life that you want to be automatically deleted from the entire database.

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).

Typical uses for an expiration value include web session data, where you want the actively stored information to be removed from the system if the user activity has stopped and not been explicitly deleted. The data will time out and be removed from the system, freeing up RAM and disk for more active data.

1.2.10. Server Warmup

Anytime you restart the Couchbase Server, or when you restore data to a server instance, the server must undergo a *warmup* process before it can handle requests for the data. During warmup the server loads data from disk into RAM; after the warmup process completes, the data is available for clients to read and write. Depending on the size and configuration of your system and the amount of data persisted in your system, server warmup may take some time to load all of the data into memory.

Couchbase Server 2.0 provides a more optimized warmup process; instead of loading data sequentially from disk into RAM, it divides the data to be loaded and handles it in multiple phases. Couchbase Server is also able to begin serving data before it has actually loaded all the keys and data from vBuckets. For more technical details about server warmup and how to manage server warmup, see Section 5.2, "Handling Server Warmup".

1.2.11. Rebalancing

The way data is stored within Couchbase Server is through the distribution offered by the vBucket structure. If you want to expand or shrink your Couchbase Server cluster then the information stored in the vBuckets needs to be redistributed between the available nodes, with the corresponding vBucket map 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, taking into account the data and replicas of the data required to support the system.

1.2.12. Replicas and Replication

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* will be 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 con-

taining 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. For more technical details about data replication within Couchbase clusters, or to learn about any configurations for replication, see Section 5.3, "Handling Replication within a Cluster".

As of Couchbase Server 2.0, you are also able to perform replication between two Couchbase clusters. This is known as cross datacenter replication (XDCR) and can provide a copy of your data at a cluster which is closer to your users, or to provide the data in case of disaster recovery. For more information about replication between clusters via XDCR see Section 5.9, "Cross Datacenter Replication (XDCR)".

1.2.13. Failover

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.

For more information, see Section 5.6, "Failing Over Nodes".

1.2.14. TAP

The TAP protocol is an internal part of the Couchbase Server system and is used in a number of different areas 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.

1.2.15. 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 an object with nested fields, arrays, and scalar datatypes.

- **Retrieve**

  To retrieve information from the database, there are two methods available:

- **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**

  If 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).

The methods of creating, updating and retrieving information are critical to the way you work with storing data in Couchbase Server.

1.2.16. Administration Tools

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. For a list of the most common administration tasks, see Chapter 5, *Administration Tasks*.

Couchbase Server includes three solutions for managing and monitoring your Couchbase Server and cluster:

- **Web Administration Console**

  Couchbase Server includes a built-in web-administration console that provides a complete interface for configuring, managing, and monitoring your Couchbase Server installation.

  For more information, see Chapter 6, *Using the Web Console*.

- **Administration 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.

  Full details are provided in Chapter 8, *Using the REST API*

- **Command Line Interface**

  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.

  For information on the command-line tools available, see Chapter 7, *Command-line Interface for Administration*.

1.2.17. Statistics and Monitoring

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.

1.3. Migration to Couchbase

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.

1.3.1. 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 Changes**

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 serialisation 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 formated as JSON.

1.3.2. 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.

Chapter 2. Installing and Upgrading

To start using Couchbase Server, you need to follow these steps:

1. Make sure your machine meets the system requirements. See Section 2.1, "Preparation".

2. Install Couchbase Server. See Section 2.2, "Installing Couchbase Server".

3. For more information on Upgrading Couchbase Server from a previous version, see Section 2.5, "Upgrading to Couchbase Server 2.1".

4. Test the installation by connecting and storing some data using the native Memcached protocol. See Section 2.8, "Testing Couchbase Server".

5. Setup the new Couchbase Server system by completing the web-based setup instructions. See Section 2.3, "Initial Server Setup".

2.1. Preparation

Warning

Mixed deployments, such as cluster with both Linux and Windows server nodes are not supported. This incomparability is due to differences in the number of shards between platforms. It is not possible either to mix operating systems within the same cluster, or configure XDCR between clusters on different platforms. You should use same operating system on all machines within a cluster and on the same operating systems on multiple clusters if you perform XDCR between the clusters.

Your system should meet the following system requirements.

2.1.1. Supported Platforms

Platform	Version	32 / 64 bit	Supported	Recomm
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
CentOS	6	32 and 64 bit	Developer and Production	CentOS6.
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 1
Windows 2008	R2 with SP1	64 bit	Developer and Production	Windows
Windows 2012		64 bit	Developer only	
Windows 7		64 bit	Developer only	
Windows 8		64 bit	Developer only	
MacOS	10.7	64 bit	Developer only	
MacOS	10.8	64 bit	Developer only	MacOS 1

Couchbase clusters with mixed platforms are not supported. Specifically, Couchbase Server on MacOSX uses 64 vBuckets as opposed to the 1024 vBuckets used by other platforms. Due to this difference, if you need to move data be-

tween a Mac OS X cluster and a cluster hosted on another platform use **cbbackup** and **cbrestore**. For more information, see Section 5.7.3, "Backup and Restore Between Mac OS X and Other Platforms".

For other platform-specific installation steps and dependencies, see the instructions for your platform under Section 2.2, "Installing Couchbase Server".

2.1.2. 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 you use XDCR and views.

- 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, you should not use a configuration lower than that specified in production. Performance on machines lower than the minimum specification will be 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 will be significantly reduced.

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

You must have the following amount of storage available:

- 1GB for application logging

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

For information and recommendations on server and cluster sizing, see Section 4.2, "Sizing Guidelines".

2.1.3. Supported Web Browsers

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.

2.1.4. Network Ports

Couchbase Server uses a number of different network ports for communication between the different components of the server, and for communicating with clients that accessing 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 will configure 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.

Table 2.1. Getting Started — Open Network Ports

Port	Description	Node to Node	Node to Client	Cluster Administration	XDCR
8091	Web Administration Port	Yes	Yes	Yes	Yes
8092	Couchbase API Port	Yes	Yes	No	Yes
11209	Internal Cluster Port	Yes	No	No	No
11210	Internal Cluster Port	Yes	Yes	No	No
11211	Client interface (proxy)	No	Yes	No	No
4369	Erlang Port Mapper (**epmd**)	Yes	No	No	No

Port	Description	Node to Node	Node to Client	Cluster Admin-istration	XDCR
21100 to 21199 (inclusive)	Node data exchange	Yes	No	No	No

2.2. Installing Couchbase Server

To install Couchbase Server on your machine you must download the appropriate package for your chosen platform from http://www.couchbase.com/downloads. For each platform, follow the corresponding platform-specific instructions.

Note

If you are installing Couchbase Server on to a machine that has previously had Couchbase Server installed and you do not want to perform an upgrade installation, you must remove Couchbase Server and any associated data from your machine before you start the installation. For more information on uninstalling Couchbase Server, see Appendix A, *Uninstalling Couchbase Server*.

To perform an upgrade installation while retaining your existing dataset, see Section 2.5, "Upgrading to Couchbase Server 2.1".

2.2.1. Red Hat Linux Installation

Before you install, make sure you check the supported platforms, see Section 2.1.1, "Supported Platforms". The RedHat installation uses the RPM package. Installation is supported on RedHat and RedHat-based operating systems such as CentOS.

1. For Red Hat Enterprise Linux version 6.0 and above, you need to install a specific OpenSSL dependency by running:

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

2. To install Couchbase Server, use the **rpm** command-line tool with the RPM package that you downloaded. You must be logged in as root (Superuser) to complete the installation:

```
root-shell> 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 RedHat RPM documentation for more information about installing packages using RPM.

After installation finishes, the installation process will display 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, you can use the RedHat **chkconfig** command to manage the Couchbase Server service, including checking the current status and creating the links to enable and disable automatic start-up. Refer to the RedHat documentation for instructions.

To do the initial setup for Couchbase, open a web browser and access the Couchbase Web Console. See Section 2.3, "Initial Server Setup".

2.2.2. Ubuntu Linux Installation

Before you install, make sure you check the supported platforms, see Section 2.1.1, "Supported Platforms".

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

```
root-shell> 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:

```
shell> 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 runlevels. Refer to the Ubuntu documentation for more information about installing packages using the Debian package manager.

After installation has completed, the installation process will display a message similar to that below:

```
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 (2.1.0) ...
 * 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 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, you can 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. See Section 2.3, "Initial Server Setup".

2.2.3. Microsoft Windows Installation

Before you install, make sure you check the supported platforms, see Section 2.1.1, "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.

Best Practice: Port Exhaustion on Windows
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 MSDN: Avoiding TCP/IP Port Exhaustion.

Warning

Couchbase Server uses the Microsoft C++ redistributable package, which will automatically download for you 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 Tech-Net.

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.

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

Figure 2.1. Couchbase Server Setup — Windows Ports
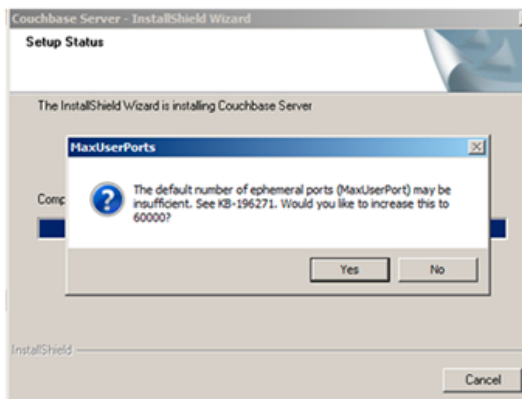
3. Click Yes.

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:

Figure 2.2. Couchbase Server Setup — Windows Ports
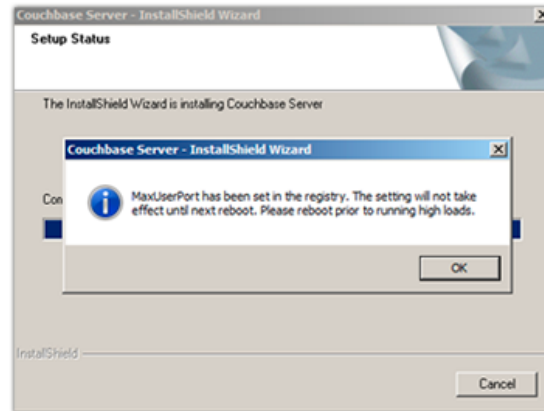
4. Restart the server for the port changes to be applied.

After installation you should follow the server setup instructions. See Section 2.3, "Initial Server Setup".

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 Powershell and start the installation executable with the `/r` command-line option:

```
shell> 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:

```
shell> 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.

2.2.4. Mac OS X Installation

Before you install, make sure you check the supported platforms, see Section 2.1.1, "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 hander 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, please use **cbbackup** and **cbrestore**. For more information, see Section 5.7.3, "Backup and Restore Between Mac OS X and Other Platforms".

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 will create 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.

Once the install 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. See Section 2.3, "Initial Server Setup" for more details.

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, as shown in Figure 3.1, "Couchbase Server on Mac OS X — Menubar Item".

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/`.

2.3. Initial Server Setup

Note

We recommend that you 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 this page .

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. For more information on best practices and disk storage, see Section 4.2.2, "Disk Throughput and Sizing".

The Configure Server Memory section sets the amount of physical RAM that will be allocated by Couchbase Server for storage. For more information and guidelines, see Section 4.2.1, "RAM Sizing".

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. For more details about using hostnames see Section 2.4, "Using Hostnames with Couchbase Server".

Figure 2.3. Couchbase Server Setup — Step 1 (New Cluster)

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.

Figure 2.4. Couchbase Server Setup — Step 2 — Loading Sample Data
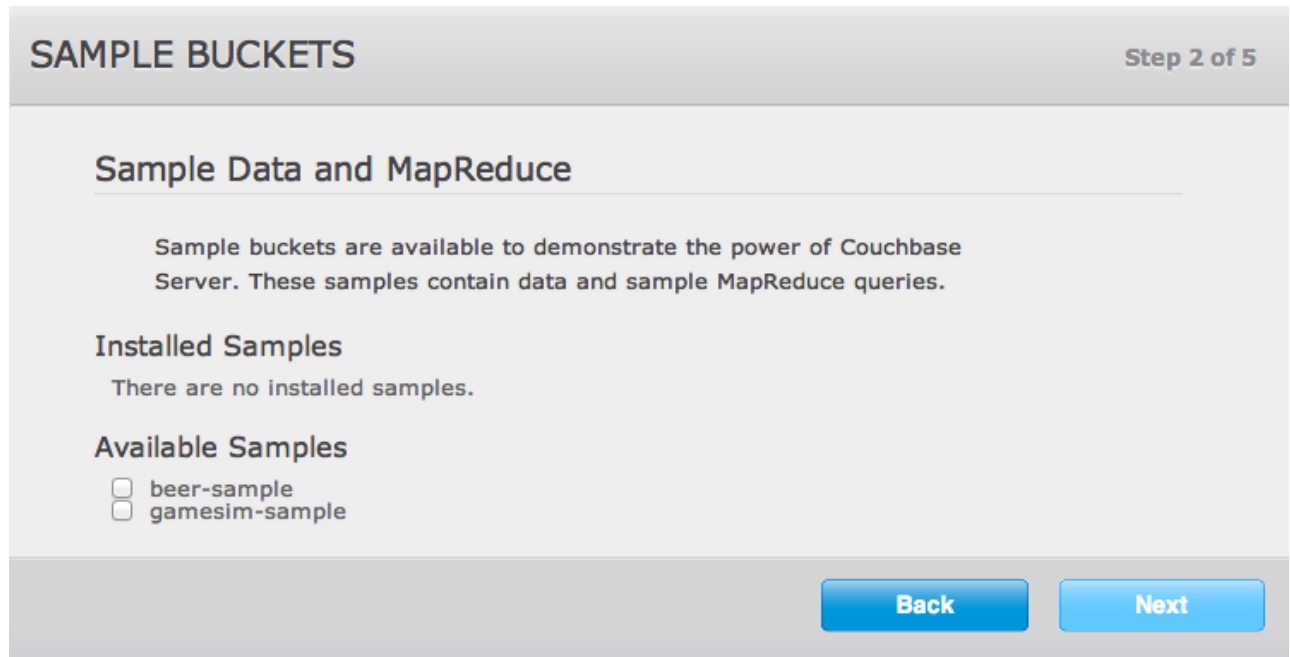
For more information on the contents of the sample buckets, see Appendix B, *Couchbase Sample Buckets*. After you create sample data buckets a Create Bucket panel appears where you create new data buckets

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. For more information about creating buckets, see Section 6.3.1, "Creating and Editing Data Buckets".

   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 newletter. 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 Chapter 8, *Using the REST API*.

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

Figure 2.5. Couchbase Server Setup — Completed
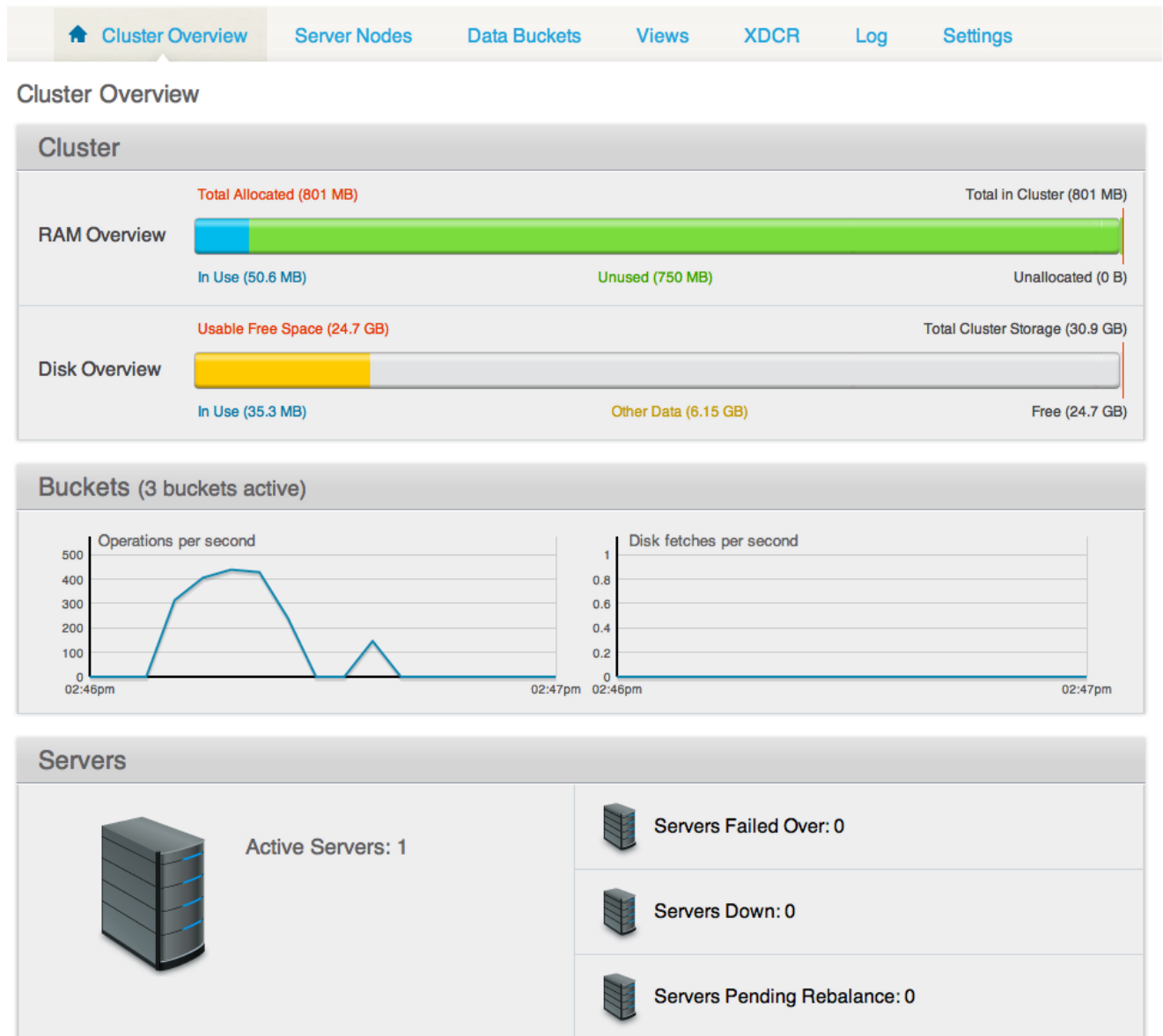
Your server is now running and ready to use. After you install your server and finish initial setup you can also optionally configure other settings, such as the port, RAM, using any of the following methods:

- **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 Section 7.4, "**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, Chapter 8, *Using the REST API*.

2.4. Using Hostnames with Couchbase Server

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. This section describes how you provide hostnames on Windows and Linux for the different versions of Couchbase Server. 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.

Couchbase 2.1 Linux and Windows

There are several ways you can provide hostnames for Couchbase 2.1+. You can provide a hostname when you install a Couchbase Server 2.1 on a machine, when you add the node to an existing cluster for online upgrade, or via a REST-API call. Couchbase Server stores this in a config file on disk. For earlier versions of Couchbase Server you must follow a manual process where you edit config files for each node which we describe below.

On Initial Setup

In the first screen that appears when you first log into Couchbase Server, you can provide either a hostname or IP address under **Configure Server Hostname**. Any hostname you provide will survive node restart:

Figure 2.6. Couchbase Server Setup — Establish Hostname

While Adding a Node

If you add a new 2.1+ node to an existing 2.0.1 or older Couchbase cluster you should first setup the hostname for the 2.1+ node in the setup wizard. If you add a new 2.1+ node to a 2.1 cluster you can provide either a hostname or IP address under **Add Server**. You provide it in the **Server IP Address** field:

Figure 2.7. Establish Hostname while Adding a Node

Providing Hostnames via REST-API

The third way you can provide a node a hostname is to do a REST request at the endpoint `http://127.0.0.1:9000/node/controller/rename`. If you use this method, you should provide the hostname before you add a node to a cluster. If you provide a hostname for a node that is already part of a Couchbase cluster; the server will reject the request and return `error 400 reason: unknown ["Renaming is disallowed for nodes that are already part of a cluster"].`:

```
curl -v -X POST -u Administrator:asdasd \
http://127.0.0.1:9000/node/controller/rename -d hostname=shz.localdomain
```

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.

Upgrading to 2.1 on Linux and Windows

If you perform an offline upgrade from Couchbase 1.8.1+ to 2.1 and you have a configured hostname using the instructions here Section 4.6.2, "Handling Changes in IP Addresses", a 2.1 server will use this configuration.

If you perform an online upgrade from 1.8.1+ to 2.1, you should add the hostname when you create the new 2.1 node. For more information about upgrading between versions, see Section 2.5, "Upgrading to Couchbase Server 2.1"

In the Cloud (such as EC2, Azure, etc). For more information about handling IP addresses and hostnames, see Section 4.6.2, "Handling Changes in IP Addresses".

2.4.1. Hostnames for Couchbase Server 2.0.1 and Earlier

For 2.0.1 please follow the same steps for 2.0 and earlier. The one difference between versions is the name and location of the file you change.

> **Warning**
>
> This operation on both Linux and Windows is data destructive. This process will reinitialize the node and remove all data on the node. You may want to perform a backup of node data before you perform this operation, see Section 7.8, "**cbbackup** Tool".

For Linux 2.0.1 and Earlier:

1. Install Couchbase Server.

2. Execute:

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

3. For 2.0, edit the `start()` function in the script located at `/opt/couchbase/bin/couchbase-server`. You do not need to edit this file for 2.0.1.

    Under the line that reads:

```
-run ns_bootstrap -- \
```

    Add a new line that reads:

```
-name ns_1@hostname \
```

    Where `hostname` is either a DNS name or an IP address that you want this server to identify the node (the 'ns_1@' prefix is mandatory). For example:

```
...
    -run ns_bootstrap -- \
    -name ns_1@couchbase1.company.com \
    -ns_server config_path "\"/opt/couchbase/etc/couchbase/static_config\"" \
...
```

4. Edit the IP address configuration file.

    For Linux 2.0 this is `/opt/couchbase/var/lib/couchbase/ip`. This file contains the identified IP address of the node once it is part of a cluster. Open the file, and add a single line containing the `hostname`, as configured in the previous step.

    For Linux 2.0.1. You update the ip_start file with the hostname. The file is at this location: /opt/couchbase/var/lib/couchbase/ip_start.

5. Delete the files under:

    * `/opt/couchbase/var/lib/couchbase/data/*`

    * `/opt/couchbase/var/lib/couchbase/mnesia/*`

    * `/opt/couchbase/var/lib/couchbase/config/config.dat`

6. Execute:

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

7. You can see the correctly identified node as the hostname under the Manage Servers page. You will again see the setup wizard since the configuration was cleared out; but after completing the wizard the node will be properly identified.

For Windows 2.0.1 and Earlier:

1. Install Couchbase Server.

2. Stop the service by running:

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

3. Unregister the service by running:

```
shell> C:\Program Files\Couchbase\Server\bin\service_unregister.bat
```

4. For 2.0, edit the script located at `C:\Program Files\Couchbase\Server\bin\service_register.bat`. You do not need this step for 2.0.1.

   - On the 7th line it says:`set NS_NAME=ns_1@%IP_ADDR%`

   - Replace`%IP_ADDR%`with the hostname/IP address that you want to use.

5. Edit the IP address configuration file.

   For Windows 2.0 edit `C:\Program Files\Couchbase\Server\var\lib\couchbase\ip`. This file contains the identified IP address of the node once it is part of a cluster. Open the file, and add a single line containing the `hostname`, as configured in the previous step.

   For Windows 2.0.1. Provide the hostname in `C:\Program Files\Couchbase\Server\var\lib\couchbase\ip_start`.

6. Register the service by running the modified script: `C:\Program Files\Couchbase\Server\bin\service_register.bat`

7. Delete the files located under:`C:\Program Files\Couchbase\Server\var\lib\couchbase\mnesia`.

8. Start the service by running:

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

9. See the node correctly identifying itself as the hostname in the GUI under the Manage Servers page. Note you will be taken back to the setup wizard since the configuration was cleared out, but after completing the wizard the node will be named properly.

2.5. Upgrading to Couchbase Server 2.1

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

- Couchbase 1.8.1 to Couchbase 2.0.x and above

- Couchbase 2.0 to Couchbase 2.0.x and above

- Couchbase 2.0.1 to Couchbase 2.1 and above

Important

> If you want to upgrade from 1.8.0 to 2.0 +, you must have enough disk space available for both your original Couchbase Server 1.8 data files and the new format for Couchbase Server 2.0 files. You will also need additional disk space for new functions such as indexing and compaction. You will need approximately three times the disk space.
>
> You cannot perform a direct upgrade from Couchbase Server 1.8.0 to 2.0+. You must first upgrade from Couchbase Server 1.8 or earlier to Couchbase Server 1.8.1 to provide data compatibility with Couchbase Server 2.0 +. After you perform this initial upgrade you can then upgrade to 2.0+.

You can perform a cluster upgrade in two ways:

- **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. For more information about a standard online upgrade, see Section 2.5.2, "Standard Online Upgrades".

  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 instructions on online upgrades, see Section 2.5.2, "Standard Online Upgrades" .

  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. For more information on swap rebalance, see Section 5.8.3, "Swap Rebalance".

- **Offline Upgrades**

  This type of 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. For full instructions on performing an offline upgrade, see Section 2.5.3, "Offline Upgrade Process" .

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

> **Best Practice: Backup your data before performing an upgrade**
> Before you perform an upgrade, whether it is online or offline, you should backup your data, see Section 5.7, "Backup and Restore".

2.5.1. Online Upgrade with Swap Rebalance

You can perform a swap rebalance to upgrade your nodes to Couchbase Server 2.0+, without reducing the performance of your cluster. This is the preferred method for performing and online upgrade of your cluster because cluster capacity is always maintained throughout the upgrade. If you are unable to perform an upgrade via swap rebalance, you may perform an standard online upgrade, see Section 2.5.2, "Standard Online Upgrades". For general information on swap rebalance, see Section 5.8.3, "Swap Rebalance".

You will need at least one extra node to perform a swap rebalance.

1. Install Couchbase Server 2.0 on one extra machine that is not yet in the cluster. For install instructions, see Section 2.2, "Installing Couchbase Server".

2. Create a backup of your cluster data using **cbbackup**. See Section 7.8, "**cbbackup** Tool".

3. Open Couchbase Web Console at an existing node in the cluster.

4. Go to Manage->Server Nodes. In the Server panel you can view and managing servers in the cluster:

   Figure 2.8. Swap Rebalance — Adding a New Node to the Cluster
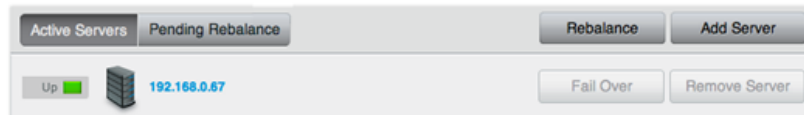
5. Click Add Server. A panel appears where you can provide credentials and either a host name or IP address for the new node: *At this point you can provide a hostname for the node you add. For more information, see Section 2.4, "Using Hostnames with Couchbase Server".*

   Figure 2.9. Swap Rebalance for Upgrade — Add Node to the Cluster

6. Remove one of your existing old nodes to from the cluster. Under Server Nodes | Server panel, Click Remove Server for the node you want to remove. This will flag this server for removal.

7. In the Server panel, click Rebalance.

   The rebalance will automatically take all data from the node flagged for removal and move it to your new node.

Repeat these steps for all the remaining old nodes in the cluster. You can add and remove multiple nodes from a cluster, however you should always add the same number of nodes from the cluster as you remove. For instance if you add one node, remove one node and if you add two nodes, you can remove two.

Until you upgrade all nodes in a cluster from 1.8.1 or earlier to Couchbase Server 2.0+, any features in 2.0+ will be disabled. This means views or XDCR will not yet function until you migrate all nodes in your cluster to 2.0+. After you do so, they will be enabled for your use.

For general information on swap rebalance, see Section 5.8.3, "Swap Rebalance".

2.5.2. Standard Online Upgrades

This is also known as a standard online upgrade process and it can take place without taking down the cluster or your application. This means that the cluster and applications can continue running while you upgrade the individual nodes in a cluster to the latest Couchbase version. You should only use this online upgrade method if you are not able to perform online upgrade via swap rebalance, see Section 2.5.1, "Online Upgrade with Swap Rebalance".

As a best practice, you should always add the same number of nodes a to a cluster as the number you remove and then perform rebalance. While it is technically possible you should avoid removing a node, rebalancing then adding back nodes into the cluster. This would reduce your cluster capacity while you add the new node back into the cluster, which could lead to data being ejected to disk.

> **Important**
>
> For information on upgrading from Couchbase Server 1.8 to Couchbase Server 2.1, see Section 2.7, "Upgrades Notes 1.8.1 to 2.1". You cannot directly upgrade from Couchbase Server 1.8 to 2.0+, instead you must first upgrade to Couchbase Server 1.8.1 for data compatibility and then upgrade to Couchbase Server 2.1+.

To perform an standard, online upgrade of your cluster:

1. Create a backup of your cluster data using **cbbackup**. See Section 7.8, "**cbbackup** Tool".

2. Choose a node to remove from the cluster and upgrade. You can upgrade one node at a time, or if you have enough cluster capacity, two nodes at a time. We do not recommend that you remove more than two nodes at a time for this upgrade.

3. In Couchbase Web Console under Manage->Server Nodes screen, click `Remove Server`. This marks the server for removal from the cluster, but does not actually remove it.

Figure 2.10. Online Upgrade — Marking a Node for Removal from a Cluster

4. The Pending Rebalance shows servers that require a rebalance. Click the `Rebalance` button next to the node you will remove.

Figure 2.11. Online Upgrade — Starting the Rebalance Process

This will move data from the node to remaining nodes in cluster. Once rebalancing has been completed, the Server Nodes display should display only the remaining, active nodes in your cluster.

Figure 2.12. Online Upgrade — Cluster with the Node Removed

5. Perform an individual node upgrade to the latest version of Couchbase Server. See Section 2.6, "Upgrading Individual Nodes".

   Couchbase Server starts automatically after the upgrade. You now need to add the node back to the cluster.

6. Open Web Console for an existing node in the cluster.

7. Go to Manage->Server Nodes.

8. Click the `Add Server` button. You will see a prompt to add a node to the cluster.

   At this point you can provide a hostname for the new node you add. For more information, see Section 2.4, "Using Hostnames with Couchbase Server".

Figure 2.13. Online Upgrade — Adding the Node back to the Cluster

After you add the new node, the Pending Rebalance count will indicate that servers need to be rebalanced into the cluster.

9. Click `Rebalance` to rebalance the cluster and bring the new node into an active state.

Repeat these steps for each node in the cluster in order to upgrade the entire cluster to a new version.

2.5.3. Offline Upgrade Process

The offline upgrade process requires you to shutdown all the applications and then the entire Couchbase Server cluster. You can then perform the upgrade the software on each machine, and bring your cluster and application back up again.

Note

If you are upgrade from Couchbase Server 1.8 to Couchbase 2.0 there are more steps for the upgrade because you must first upgrade to Couchbase 1.8.1 for data compatibility with 2.0. For more information, see Section 2.7, "Upgrades Notes 1.8.1 to 2.1".

Check that your disk write queue (Section 10.4, "Disk Write Queue") is completely drained to ensure all data has been persisted to disk and will be available after the upgrade. It is a best practice to turn off your application and allow the queue to drain before you upgrade it. It is also a best practice to perform a backup of all data before you upgrade

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. For instructions, see Section 6.8.2, "Enabling Auto-Failover Settings".

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.

Figure 2.14. Monitoring Disk Write Queue for Upgrade
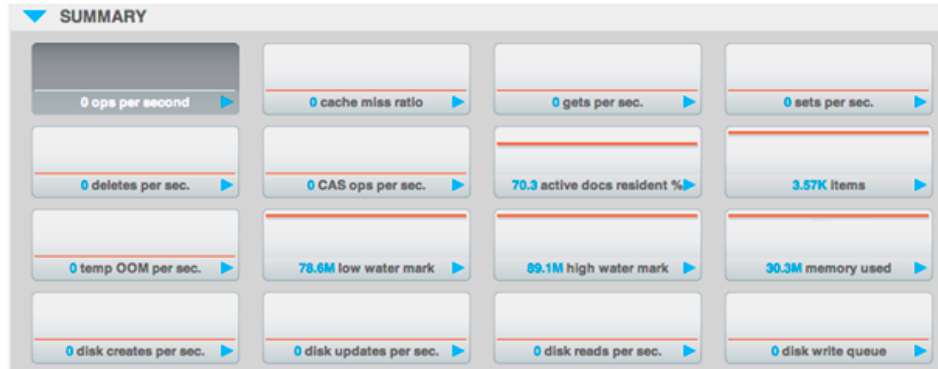
5. Create a backup of your cluster data using **cbbackup**. See Section 7.8, "**cbbackup** Tool".

6. Shutdown Couchbase Server on each machine in your cluster. For instructions, see Section 3.2, "Server Startup and Shutdown"

7. After you shutdown your nodes, perform a standard node upgrade to the new version of Couchbase Server. See Section 2.6, "Upgrading Individual Nodes" for instructions.

   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. See Section 10.3, "Monitoring startup (warmup)".

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

2.6. 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** . See Section 5.7.1, "Backing Up Using **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 you should make your own backup of these files:

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

4. Stop Couchbase Server. For instructions, see Section 3.2, "Server Startup and Shutdown".

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. See Section 2.4, "Using Hostnames with Couchbase Server"

6. Check for required components and if needed, install them. This ensures that Couchbase Server upgrades and migrates your existing data files. See Section 2.7, "Upgrades Notes 1.8.1 to 2.1".

7. Perform the installation upgrade for your platform:

   RHEL/Centos

   You can perform an upgrade install using the RPM package — this will keep the data and existing configuration.

   ```
   root-shell> rpm -U couchbase-server-architecture__ _meta_current_version__.rpm
   ```

   Ubuntu/Debian Linux

   You can perform a package upgrade by installing the updated `.pkg` package:

   ```
   shell> sudo dpkg -i couchbase-server-architecture_ _meta_current_release.deb
   ```

   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.

2.7. Upgrades Notes 1.8.1 to 2.1

You can upgrade from Couchbase Server 1.8.1 to Couchbase Server 2.1+ using either the online or offline upgrade method. See Upgrading to Couchbase Server 1.8 for more information.

Use Online Upgrades for Couchbase Server 1.8.1 to Couchbase Server 2.1+

We recommend online upgrade method for 1.8.1 to 2.1+. The process is quicker and can take place while your cluster and application are up and running. When you upgrade from Couchbase Server 1.8.1 to Couchbase Server 2.1+, 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 2. you should have a 10 second delay from adding a 2.1+ node to the cluster and rebalancing. If you request rebalance too soon after adding a 2.1+ node, the rebalance may fail.

Linux Upgrade Notes for 1.8.1 to 2.1+

When you upgrade from Couchbase Server 1.8 to Couchbase Server 2.1+ on Linux, you should be aware of the **OpenSSL** requirement. OpenSSL is a required component and you will get an error message during upgrade if it is not installed. To install it RedHat-based systems, use **yum**:

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

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

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

Windows Upgrade Notes for 1.8.1 to 2.1+

If you have configured your Couchbase Server nodes to use hostnames, rather than IP addresses, to identify themselves within the cluster, you must ensure that the IP and hostname configuration is correct both before the upgrade and after upgrading the software. See Section 2.4.1, "Hostnames for Couchbase Server 2.0.1 and Earlier".

Mac OSX Notes for 1.8.1 to 2.1+

There is currently no officially supported upgrade installer for Mac OSX. If you want to migrate to 1.8.1 to 2.1+ on OSX, you must make a backup of your data files with **cbbackup**, install the latest version, then restore your data with **cbrestore**. For more information, see Section 7.8, "**cbbackup** Tool" and Section 7.9, "**cbrestore** Tool".

2.7.1. Upgrade Notes 1.8 and Earlier to 2.1+

If you run Couchbase Server 1.8 or earlier, including Membase 1.7.2 and earlier, you must upgrade to Couchbase Server 1.8.1 first.You do this so that your data files can convert into 2.0 compatible formats. This conversion is only available from 1.8.1 to 2.0 + upgrades.

Offline Upgrade

- 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 will copy 1.8.1-compatible data and configuration files to a backup location.

        The **cbupgrade** program will automatically start. This will non-destructively convert data from the 1.8.1 database file format (SQLite) to 2.0 database file format (couchstore). The 1.8 database files are left "as-is", and new 2.0 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 you start Couchbase Server 2.0.

Once the upgrade process finishes, Couchbase Server 2.0 starts automatically. Repeat this process on all nodes within your cluster.

2.7.2. Upgrading from Community Edition to Enterprise Edition

You should use the same version number when you perform the migration process to prevent version differences which may result in a failed upgrade. To upgrade between Couchbase Server Community Edition and Couchbase Server Enterprise Edition, you can use two methods:

- Perform an online upgrade

    Here you remove one node from the cluster and rebalance. On the nodes you have taken out of the cluster, uninstall Couchbase Server Community Edition package, and install Couchbase Server Enterprise Edition. You can then add the new nodes back into the cluster and rebalance. Repeat this process until the entire cluster is using the Enterprise Edition.

    For more information on performing online upgrades, see Section 2.5.2, "Standard Online Upgrades".

- Perform an offline upgrade

    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.

    For more information on performing offline upgrades, see Section 2.5.3, "Offline Upgrade Process".

2.8. 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. For more information, see Section 2.8.1, "Testing Couchbase Server using **cbworkloadgen**".

Using **telnet** only checks the Memcached compatibility ports and the memcached text-only protocol. For more information, see Section 2.8.2, "Testing Couchbase Server using Telnet".

2.8.1. Testing Couchbase Server using cbworkloadgen

The **cbworkloadgen** is a basic tool that can be used to check the availability and connectivity of a Couchbase Server cluster. The tool executes a number of different operations to provide basic testing functionality for your server.

> **Note**
>
> **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
```

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
```

2.8.2. Testing Couchbase Server using Telnet

You can test your Couchbase Server installation by using Telnet to connect to the server and using the Memcached text protocol. This is the simplest method for determining if your Couchbase Server is running.

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

You will need to have **telnet** 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.

Connect to the server:

```
shell> 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.

2.9. Next Steps

- For basic instructions on using your Couchbase Server installation, see Chapter 3, *Administration Basics*.

- For information on deploying and building your Couchbase Server cluster, see Section 4.7, "Deployment Strategies".

- For instructions on how to use the Couchbase Web Console to manage your Couchbase Server installation, see Chapter 6, *Using the Web Console*.

- 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. For more information, see Couchbase SDKs.

Chapter 3. Administration Basics

This chapter covers everything on the Administration of a Couchbase Sever cluster. Administration is supported through three primary methods:

- **Couchbase Web Console**

  Couchbase includes a built-in web server and administration interface that provides access to the administration and statistic information for your cluster.

  For more information, read Chapter 6, *Using the Web Console*.

- **Command-line Toolkit**

  Provided within the Couchbase package are a number of command-line tools that allow you to communicate and control your Couchbase cluster.

  For more information, read Chapter 7, *Command-line Interface for Administration*.

- **Couchbase REST API**

  Couchbase Server includes a RESTful API that enables any tool capable of communicating over HTTP to administer and monitor a Couchbase cluster.

  For more information, read Chapter 8, *Using the REST API*.

3.1. Couchbase Data Files

By default, Couchbase Server will store the data files under the following paths:

Platform	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/lig/couchbase/data`

This path can be changed for each node at setup either via the Web UI setup wizard, using theREST API or using the Couchbase CLI:

> **Warning**
>
> Changing the data path for a node that is already part of a cluster will permanently delete the data stored.

Linux:

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

Windows:

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

> **Note**
>
> When using the command line tool, you cannot change the data file and index file path settings individually. If you need to configure the data file and index file paths individually, use the REST API. For more information, see Section 8.5.3, "Configuring Index Path for a Node"

For Couchbase Server 2.0, once a node or cluster has already been setup and is storing data, you cannot change the path while the node is part of a running cluster. You must take the node out of the cluster then follow the steps below:

1. Change the path on a running node either via the REST API or using the Couchbase CLI (commands above). This change will not actually take effect until the node is restarted. For more information about using a REST-API request for ejecting nodes from clusters, see Section 8.7.4, "Removing a Node from a Cluster".

2. Shut the node down.

3. Copy all the data files from their original location into the new location.

4. Start the service again and monitor the "warmup" of the data.

3.2. Server Startup and Shutdown

The packaged installations of Couchbase Server include support for automatically starting and stopping Couchbase Server using the native boot and shutdown mechanisms.

For information on starting and stopping Couchbase Server, see the different platform-specific links:

- Section 3.2.1, "Startup and Shutdown on Linux"

- Section 3.2.2, "Startup and Shutdown on Windows"

- Section 3.2.3, "Startup and Shutdown on Mac OS X"

3.2.1. 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
```

3.2.2. 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.

> **Note**
>
> 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 should start automatically when the machine boots. To manually start the service, open the Windows Task Manager and choose the Services tab, or select the `Start`, choose Run and then type `Services.msc` to open the Services management console.

Once open, find the CouchbaseServer service, right-click and then choose to Start or Stop the service as appropriate. You can also alter the configuration so that the service is not automatically started during boot.

Alternatively, you can start and stop the service from the command-line, either by using the system **net** command. For example, to start Couchbase Server:

```
shell> net start CouchbaseServer
```

To stop Couchbase Server:

```
> net stop CouchbaseServer
```

Start and Stop scripts are also provided in the standard Couchbase Server installation in the `bin` directory. To start the server using this script:

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

To stop the server using the supplied script:

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

3.2.3. 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 menubar item through which you can control the server.

Figure 3.1. Couchbase Server on Mac OS X — Menubar Item

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.

Note

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 menubar interface. To restart, you must open the Couchbase Server application from the installation folder.

Chapter 4. Best Practices

When building your Couchbase Server cluster, you need to keep multiple aspects in mind: the configuration and hardware of individual servers, the overall cluster sizing and distribution configuration, and more.

For more information on cluster designing basics, see: Section 4.1, "Cluster Design Considerations".

If you are hosting in the cloud, see Section 4.6, "Using Couchbase in the Cloud".

4.1. Cluster Design Considerations

- 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.

- Moxi: We always prefer 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. We don't recommend server-side moxi because of the drawback: if a server receives a client request and doesn't have the requested data, there's an additional hop. Read more about clients here. Read more about different Deployment Strategies here.

- 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.

4.2. Sizing Guidelines

Here are the primary considerations when sizing your Couchbase Server cluster:

- How many nodes do I need?

- How large (RAM, CPU, disk space) should those nodes be?

To answer the first question, consider following factors:

- 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, how you choose your primary factor will depend on the data set and information that you are storing.

- 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.

4.2.1. 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.

4.2.1.1. Working Set

Before we can decide how much memory we will need for the cluster, we should understand the concept of a 'working set.' The 'working set' is the data that your application actively uses at any point in time. Ideally you want all your working set to live in memory.

4.2.1.2. 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).

How much memory and disk space per node you will need depends on several different variables, which are defined below:

Note

Calculations are per bucket

The calculations below are per-bucket calculations. The calculations need to be summed up across all buckets. If all your buckets have the same configuration, you can treat your total data as a single bucket. There is no per-bucket overhead that needs to be considered.

Table 4.1. Deployment — Sizing — Input Variables

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:

Table 4.2. Deployment — Sizing — Constants

Constant	Description
Metadata per document (metadata_per_document)	This is the amount of memory that Couchbase needs to store metadata per document. Prior to Couchbase 2.1, metadata used 64 bytes. As of Couchbase 2.1, 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 [a]	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 70%.

[a] The cluster needs additonal 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.

This is a rough guideline to size your cluster:

Variable	Calculation
no_of_copies	`1 + number_of_replicas`
total_metadata [a]	`(documents_num) * (metadata_per_document + ID_size) * (no_of_copies)`
total_dataset	`(documents_num) * (value_size) * (no_of_copies)`
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`

[a] All the documents need to live in the memory.

Note

You will need at least the number of replicas + 1 nodes regardless of your data size.

Here is a sample sizing calculation:

Table 4.3. Deployment — Sizing — Input Variables

Input Variable	value
documents_num	1,000,000
ID_size	100
value_size	10,000
number_of_replicas	1
working_set_percentage	20%

Table 4.4. Deployment — Sizing — Constants

Constants	value
Type of Storage	SSD
overhead_percentage	25%
metadata_per_document	56 for 2.1, 64 for 2.0.X
high_water_mark	70%

Table 4.5. Deployment — Sizing — Variable Calculations

Variable	Calculation
no_of_copies	= 2 [a]
total_metadata	= 1,000,000 * (100 + 120) * (2) = 440,000,000
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

[a] 1 for original and 1 for replica

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
```

Note

RAM quota

You will not be able to allocate all your machine RAM to the per_node_ram_quota as there may be other programs running on your machine.

4.2.2. Disk Throughput and Sizing

Couchbase Server decouples RAM from the I/O layer. This is a huge advantage. It allows you to scale high at very low and consistent latencies. It also enables Couchbase Server to handle very high write loads without affecting your application's performance.

However, Couchbase Server still needs to be able to write data to disk. Your disks need to be capable of handling a steady stream of incoming data. It is important to analyze your application's write load and provide enough disk throughput to match.

While information is written to disk, the internal statistics system monitors the outstanding items in the disk write queue. From its display, you can see the disk write queue load. Its peak shows how many items stored in Couchbase Server would be lost in the event of a server failure. It is up to your own internal requirements to decide how much vulnerability you are comfortable with. Then you size the cluster accordingly so that the disk write queue level remains low across the entire cluster. Adding more nodes will provide more disk throughput.

Disk space is also required to persist data. How much disk space you should plan for is dependent on how your data grows. You will also want to store backup data on the system. A good guideline is to plan for at least 130% of the total data you expect. 100% of this is for data backup, and 30% for overhead during file maintenance.

4.2.3. 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 you can calculate your network bandwidth requirements using this formula:

```
Bandwidth = (operations per second * item size) +
    overhead for rebalancing
```

And you can calculate the operations per second with this formula:

```
Operations per second = Application reads +
    (Application writes * Replica copies)
```

4.2.4. 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.

4.2.4.1. Data distribution

Basically, more nodes are better than less. If you only have two nodes, your data will be split across the two nodes, half and half. This means that half of your dataset will be "impacted" if one goes away. On the other hand, with ten nodes, only 10% of the dataset will be "impacted" if one goes away. Even with automatic failover, there will still be some period of time when data is unavailable if nodes fail. This can be mitigated by having more nodes.

After a failover, the cluster will need to take 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.

4.2.4.2. 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. Here are examples:

- 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.

Note

After a node goes down and is failed over, try to replace that node as soon as possible and rebalance. The rebalance will recreate the replica copies (if you still have enough nodes to do so).

As a rule of thumb, we recommend that you 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.

4.2.4.3. 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, here is what we recommend:

- 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 we recommend that you have a separate devices for server install, data directories, and index directories.

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.

- Network: Most configurations will work with Gigabit Ethernet interfaces. Faster solutions such as 10GBit and Inifiniband will provide spare capacity.

4.2.4.4. 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).

Read about best practices with the cloud in Section 4.6, "Using Couchbase in the Cloud".

4.3. Deployment Considerations

- **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. For more information on this issue, including information on how to adjust the configuration and increase the available ports, see MSDN: Avoiding TCP/IP Port Exhaustion.

4.4. Ongoing Monitoring and Maintenance

To fully 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. Some of these 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. Read more about this concept here. Here are two important statistics related to water marks:

- High Water Mark (`ep_mem_high_wat`)

  The system will start ejecting values 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 a threshold known as low water mark is reached, this process starts ejecting inactive replica data from RAM on the node.

You can find values for these important stats with the following command:

```
shell> cbstats IP:11210 all | \
    egrep "todo|ep_queue_size|_eject|mem|max_data|hits|misses"
```

This will output the following statistics:

```
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:
```

Note

Make sure you monitor the disk space, CPU usage, and swapping on all your nodes, using the standard monitoring tools.

4.4.1. 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 will 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.

4.5. Couchbase Behind a Secondary Firewall

If you are deploying Couchbase behind a secondary firewall, you should open the ports that Couchbase Server uses for communication. In particular, the following ports should be kept open: 11211, 11210, 4369, 8091, 8092, and the port range from 21100 to 21199.

- Port 11210

  If you're using smart clients or client-side Moxi from outside the second level firewall, also open up port 11210 (in addition to the above port 8091), so that the smart client libraries or client-side Moxi can directly connect to the data nodes.

- Port 8091

  If you want to use the web admin console from outside the second level firewall, also open up port 8091 (for REST/HTTP traffic).

- Port 8092

  Access to views is provided on port 8092; if this port is not open, you won't be able to run access views, run queries, or update design documents, not even through the Web Admin Console.

- Port 11211

  The server-side Moxi port is 11211. Pre-existing Couchbase and memcached (non-smart) client libraries that are outside the second level firewall would just need port 11211 open to work.

  Note

  Nodes within the Couchbase Server cluster need all the above ports open to work: 11211, 11210, 4369, 8091, 8092, and the port range from 21100 to 21199

4.6. Using Couchbase in the Cloud

For the purposes of this discussion, we will refer to "the cloud" 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, we have done extensive testing within EC2 (and some of our 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.

Being simply a software package, 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.

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 prepackaged deployment. Check out the templates here Couchbase on RightScale

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

How to Handle 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.

4.6.1. 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.

4.6.2. 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 should configure Couchbase to use a hostname instead of an IP 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.

> **Warning**
>
> The following steps will completely destroy 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. For more information, see Section 2.5, "Upgrading to Couchbase Server 2.1".
>
> 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 2.1 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. For instructions, see Section 2.4, "Using Hostnames with Couchbase Server".

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. For instructions, see Section 2.4.1, "Hostnames for Couchbase Server 2.0.1 and Earlier".

4.6.3. 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.

4.6.4. Swap Space

Swap space in Linux 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. From a range of 0 to 100, swappiness indicates how frequently a system should use swap space based on RAM usage. We recommend the following for swap space:

- By default on most Linux platforms, swapiness is set to 60. However this will make a system go into swap too frequently for Couchbase Server.

- If you use Couchbase Server 2.0+ without views, we recommend setting swappiness of 10 to avoid going into swap too frequently.

- If you use Couchbase Server 2.0+ with views, we recommend setting swappiness to 0 or else your system swap usage will be far too high.

- Certain cloud systems by default do not have a swap partition configured. If you are using views, we recommend setting swappiness to 0. If you are not using views, you can set this to 10.

To view the currently set swappiness on your system, enter this:

```
sysctl vm.swappiness
```

Or you can use this command:

```
cat /proc/sys/vm/swappiness
```

To change the swappiness, edit `/etc/sysctl.conf` and add `vm.swappiness` at the end of the file. After you save this and reboot your system, this setting will be used.

4.7. 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.

4.7.1. 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.

Figure 4.1. Deployment Strategy — Using a vBucket Aware Client
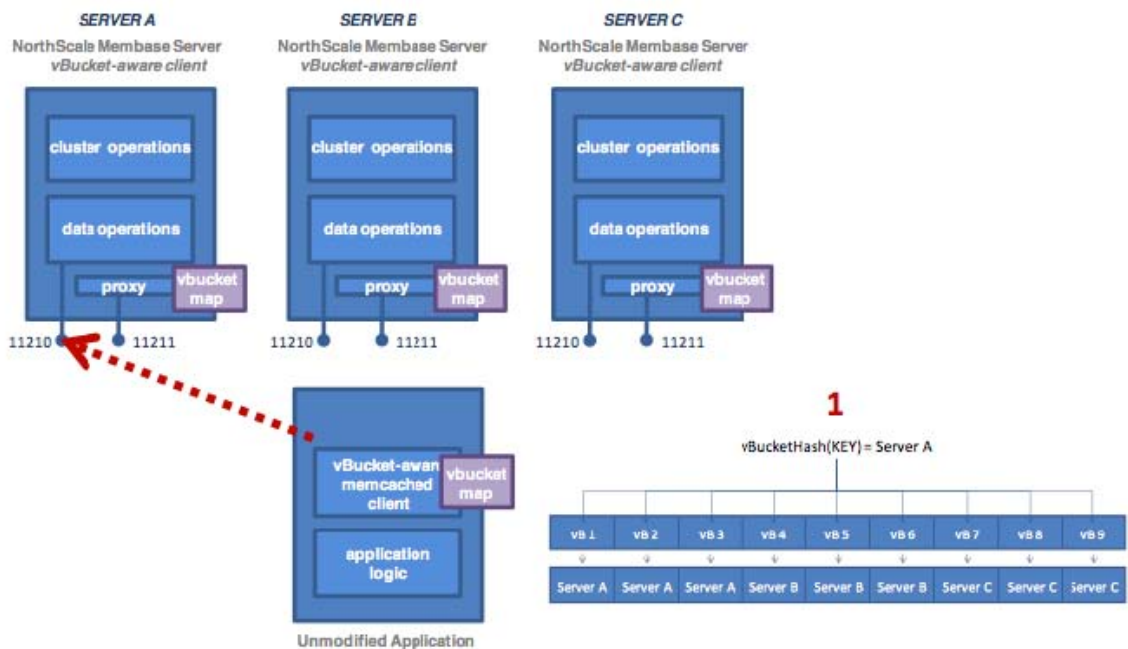
See also vBuckets for an in-depth description.

4.7.2. 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.

Figure 4.2. Deployment Strategy — Standalone Proxy
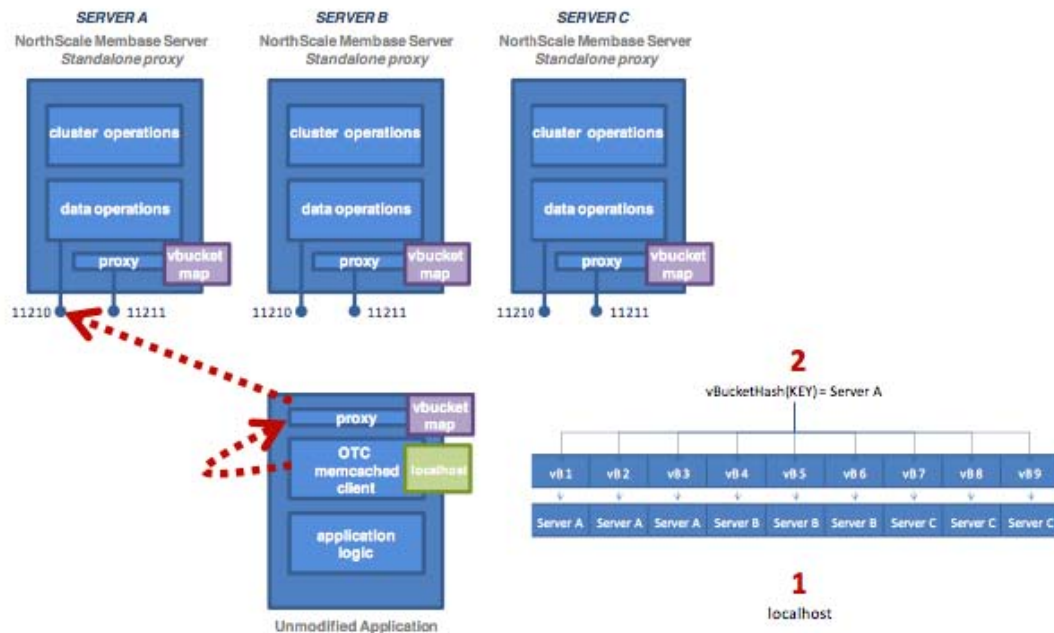
We configured the memcached client to have just one server in its server list (localhost), so all operations are forwarded to `localhost:11211` — a port serviced by the proxy. The proxy hashes the document ID to a vBucket, looks up the host server in the vBucket table, and then sends the operation to the appropriate Couchbase Server on port 11210.

> **Note**
>
> For the corresponding Moxi product, please use the Moxi 1.8 series. See Moxi 1.8 Manual.

4.7.3. Using Server-Side (Couchbase Embedded) Proxy

> **Warning**
>
> We do not recommend server-side proxy configuration for production use. You should 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 will happen 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.

Figure 4.3. Deployment Strategy — Using the Embedded Proxy
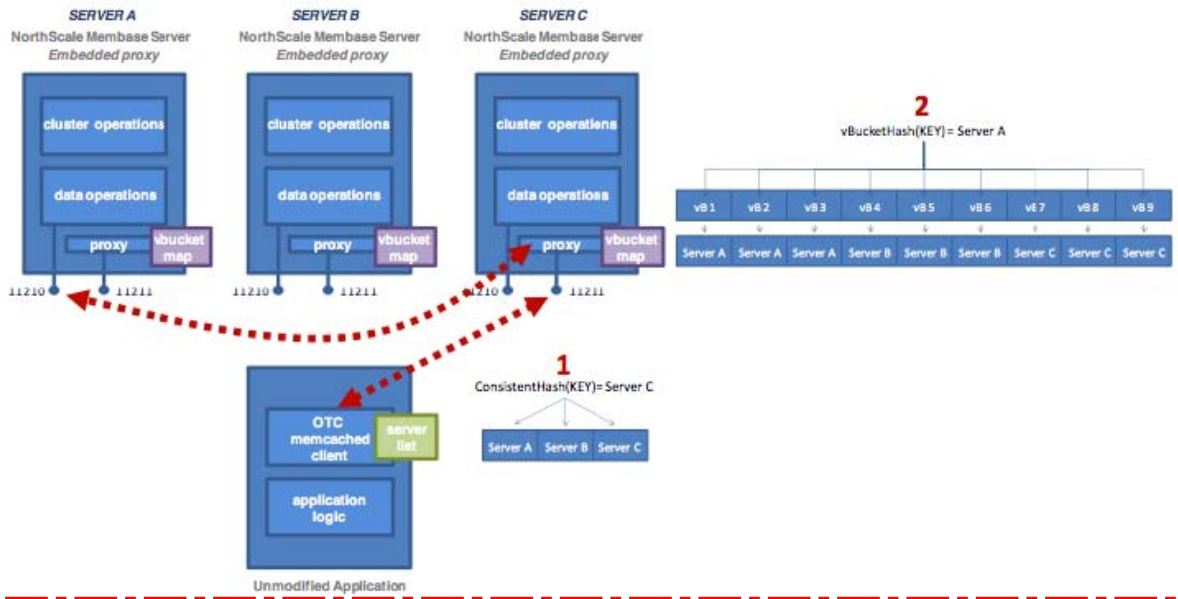
> **Note**
>
> For the corresponding Moxi product, please use the Moxi 1.8 series. See Moxi 1.8 Manual.

Chapter 5. Administration 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 also only a few different configuration variables, and the majority of these do not need to be modified or altered in most installations.

However, there are a number of different tasks that you will need to carry out over the lifetime of your cluster, such as backup, failover and altering the size of your cluster as your application demands change. You will also need to monitor and react to the various statistics reported by the server to ensure that your cluster is operating at the highest performance level, and to expand your cluster when you need to expand the RAM or disk I/O capabilities.

These administration tasks include:

- **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.

  More information on increasing and reducing your cluster size and performing a rebalance operation is available in Section 5.8, "Rebalancing".

- **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. To manage the warmup process for Couchbase Server instances, see Section 5.2, "Handling Server Warmup".

- **Handle 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.

  For more information, see Section 5.6, "Failing Over Nodes".

- **Manage 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.

  Information on how to enable and configure auto-compaction is available in Section 5.5, "Database and View Compaction".

- **Backup and Restore 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.

More information on the available backup and restore methods are available in Section 5.7, "Backup and Restore".

5.1. Using Multi- Readers and Writers

As of Couchbase Server 2.1, we support multiple readers and writers to persist data onto disk. For earlier versions of Couchbase Server, each bucket instance had only single disk reader and writer workers. 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 can help you increase your disk I/O throughput. If your disk utilization is below the optimal level, you can increase the setting to improve disk utilization. If your disk utilization is near the maximum and you see heavy I/O contention, you can decrease this setting. By default we allocate three total readers and writers.

How you change this setting depends on the hardware in your Couchbase cluster:

- If you deploy your cluster on the minimum hardware requirement which is dual-core CPUs running on 2GHz and 4GM of physical RAM, you should stay with the default setting of three.

- If you deploy your servers on recommended hardware requirements or above you can increase this setting to eight. The recommended hardware requirements are quad-core processes on 64-bit CPU and 3GHz, 16GB RAM physical storage. We also recommend solid state drives.

- If you have a hardware configuration which conforms to pre-2.1 hardware requirements, you should change this setting to the minimum, which is 2.

For more information about system requirements for Couchbase Server, see Section 2.1.2, "Resource Requirements".

Changing the Number of Readers and Writers

You should configure this setting in Couchbase Web Console when you initially create a data bucket. For general information on creating buckets, see Section 6.3.1.1, "Creating a New 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. Select a number of reader/writers under Disk Read-Write Concurrency.

   Figure 5.1. Number of Concurrent Readers and Writers

3. Provide other bucket-level settings of your choice.

4. Click Create.

   The new bucket will appear under the Data Buckets tabs in Web Console with a yellow indicator to show the bucket is in warmup phase:

Figure 5.2. Bucket Warmup: Concurrent Readers and Writers

After the bucket completes warmup, it will appear with a green indicator next to it:

Figure 5.3. Bucket Warmup: Concurrent Readers and Writers

This default bucket is now ready to receive and serve requests. If you create a named bucket, you will see a similar status indicator next to your named bucket.

Viewing Status of Multi- Readers and Writers

After you change this setting you can view the status of this setting with **cbstats**:

```
/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 indicates we have three reader threads and two writer threads on `bucket_name` in the cluster at `host-name:11210`. The vBucket map for the data bucket is grouped into multiple shards, where one read worker will access one of the shards. In this example we have one reader for each of the three shards. This report also tell us we are optimized for read data access because we have more reader threads than writer threads for the bucket. You can also view the number of threads if you view the data bucket properties via a REST call:

```
curl -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,
....
}
```

To view the changed behavior, go to the Data Buckets tab and select your named bucket. Under the summary section, you can view the `disk write queue` for change in drain rate. Under the Disk Queues section, you see a change in the active and replica `drain rate` fields after you change this setting. For more information about bucket information in Web Console, see Section 6.4.1, "Individual Bucket Monitoring".

Changing Readers and Writers for Existing Buckets

You can change this setting after you create a data bucket in Web Console or REST-API. If you do so, the bucket will be re-started and will go through server warmup before it becomes available. For more information about warmup, see Section 5.2, "Handling Server Warmup".

To change this setting in Web Console:

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.

   Figure 5.4. Changing Readers and Writers for Existing Buckets
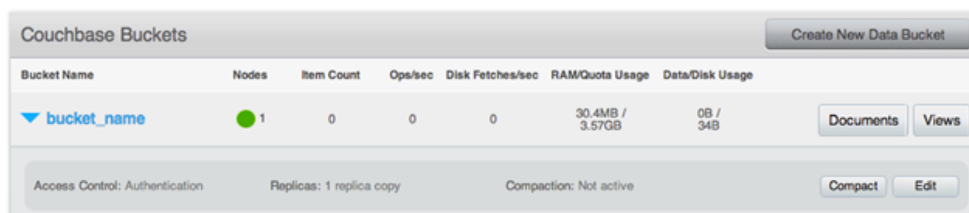
3. Click Edit.

   A Configure Bucket panel appears where you can edit the current settings for the bucket. The Disk Read-Write section is where you will change this setting.

4. Enter a number of readers and writers.

5. Click Save.

   A warning appears indicating that this change will recreate the data bucket.

   Figure 5.5. Changing Readers and Writers for Existing Buckets: Warning
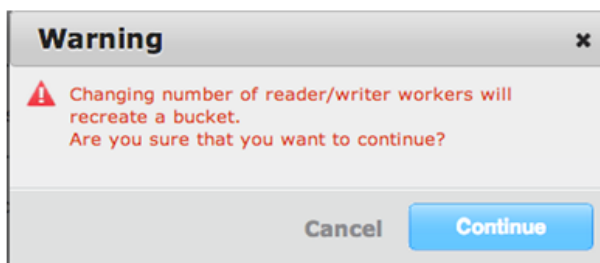
6. Click Continue

   The Data Buckets tab appears and you see the named bucket with a yellow indicator. This tells you the bucket is recreated and is warming up. The indicator turns green when the bucket has completed warmup. At this point it is ready to receive and serve requests.

To change this setting via REST, we provide 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
```

For details about changing bucket properties via REST, including limitations and behavior, see Section 8.6.5, "Creating and Editing Data Buckets".

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
 ....
```

Upon success you will see this response:

```
HTTP/1.1 200 OK
....
```

If you provide an invalid number of threads, you will a response similar to the following:

```
HTTP/1.1 400 Bad Request
....
{"errors":{"threadsNumber":"The number of threads can't be greater than 8"},"
```

If you upgrade a Couchbase cluster, a new node can use this setting without bucket restart and warmup. In this case you set up a new 2.1+ node, add that node to the cluster, and on that new node edit the existing bucket setting for readers and writers. After you rebalance the cluster, this new node will perform reads and writes with multiple readers and writers and the data bucket will not restart or go through warmup. All existing pre-2.1 nodes will remain with a single readers and writers for the data bucket. As you continue the upgrade and add additional 2.1+ nodes to the cluster, these new nodes will automatically pick up the setting and use multiple readers and writers for the bucket. For general information about Couchbase cluster upgrade, see Section 2.5, "Upgrading to Couchbase Server 2.1".

5.2. Handling Server Warmup

Couchbase Server 2.0+ provides improved performance for *server warmup*; this is the process a restarted server must undergo before it can serve data. During this process the server loads items persisted on disk into RAM. One approach to load data is to do sequential loading of items from disk into RAM; however it is not necessarily an effective process because the server does not take into account whether the items are frequently used. In Couchbase Server 2.0, we provide additional optimizations during the warmup process to make data more rapidly available, and to prioritize frequently-used items in an access log. The server pre-fetches a list of most-frequently accessed keys and fetches these documents before it fetches any other items from disk.

The server also runs a configurable scanner process which will determine which keys are most frequently-used. You can use Couchbase Server command-line tools to change the initial time and the interval for the process. You may want to do this for instance, 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. For more information, see Section 7.6.3, "Changing Access Log Settings".

The server can also switch into a ready mode before it has actually retrieved all documents for keys into RAM, and therefore can begin serving data before it has loaded all stored items. This is also a setting you can configure so that server warmup is faster.

The following describes the initial warmup phases for the Couchbase Server 2.0+. In these first phase, the server begins fetch all keys and metadata from disk. Then the server gets access log information it needs to retrieve the most-used keys:

- **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.

- **Key Dump**. In this next phase, the server begins pre-fetching all keys and metadata from disk based on items in the vBucket list.

- **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 can be configured. If this log exists, the server will first load items based on this log before it loads other items from disk.

Once Couchbase Server has information about keys and has read in any access log information, it is ready to load documents:

- **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 will sequentially load 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, or

- The server has finished loading documents for every key stored on disk for all vBuckets, or

- The total number of documents loaded into memory is greater than, or equal to, the setting for `ep_warmup_min_items_threshold`, or

- If total % of RAM filled by documents is greater than, or equal to, the setting for `ep_warmup_min_memory_threshold`, or

- 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, this is known as the *run level*; when Couchbase Server reaches this point, it immediately stops loading documents for the remaining keys. After this point, Couchbase Server will load this remaining documents from disk into RAM as a background data fetch.

In order to adjust warmup behavior, it is also important for you to understand the access log and scanning process in Couchbase Server 2.0. 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 will periodically scan 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 will run 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 will produce a `ENGINE_TMPFAIL (0x0d)` error code. This error indicates that data access is still not available because warmup has not yet finished. 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, or the client performs a backoff of requests, or it produces an error and does not retry the request. 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. For more information, refer to the Language Reference for your chosen Couchbase SDK.

5.2.1. Getting Warmup Information

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:

```
shell> cbstats localhost:11210 -b beer_sample -p bucket_password all | grep 'warmup'
shell> cbstats hostname:11210 -b my_bucket -p bucket_password raw warmup
```

Here the `localhost:11210` is the host name and default memcached port for a given node and `beer_sample` is a named bucket for the node. If you do not specify a bucket name, the command will apply to any existing default bucket for the node.

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 based, 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**. This indicates 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**. Server is ready to handle read and write requests.

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.

There are more detailed statistics available on the warmup process. For more information, see Section 7.5.2, "Getting Warmup Information".

5.2.2. Changing the Warmup Threshold

To modify warmup behavior by changing the setting for `ep_warmup_min_items_threshold` use the command-line tool provided with your Couchbase Server installation, **cbepctl**. This indicates the number 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.

5.2.3. Changing Access Scanner Settings

The server runs a periodic scanner process which will determine which keys are most frequently-used, and therefore, which documents should be loaded first during server warmup. You can use **cbepctl flush_param** to change the initial time and the interval for the process. You may want to do this, for instance, 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 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 will run 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 UTC by default. To change the time interval when the access scanner process runs to every 20 minutes:

```
shell> ./cbepctl localhost:11210 -b beer-sample set flush_param alog_sleep_time 20
```

This updates the parameter for the named bucket, beer-sample on the given node on `localhost`. To change the initial time that the access scanner process runs from the default of 2:00 AM UTC:

```
shell> ./cbepctl hostname:11210 -b beer-sample -p beer-password set flush_param alog_task_time 13
```

In this example we set the initial time to 1:00 PM UTC.

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. For more information, see Section 7.6, "**cbepctl** Tool".

5.3. Handling Replication within a Cluster

Within a Couchbase cluster, you have *replica data* which is a copy of an item at another node. After you write 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:

Figure 5.6. Replica vBuckets and Replication
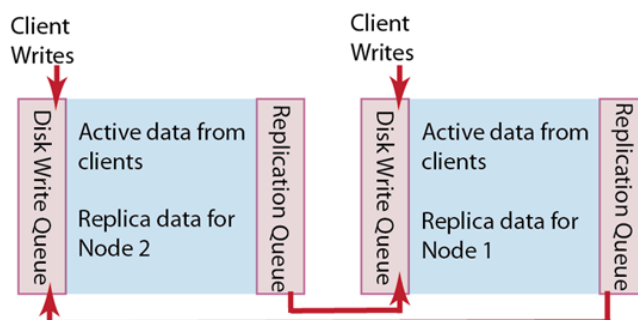
When a client application writes data to a node, that data will be placed in a replication queue and then a copy will be sent to another node. The replicated data will be available in RAM on the second node and will be placed in a disk write queue to be stored on disk at the second node.

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. For information about changing this setting, see Section 7.6.2, "Changing Disk Write Queue Quotas".

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. For more information about node failure and failover, see Section 5.6, "Failing Over Nodes".

5.3.1. 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.

For more information about creating and editing buckets, or specifying replicas for buckets, see Section 6.3.1, "Creating and Editing Data Buckets".

5.3.2. 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:

```
shell> ./cbepctl 10.5.2.31:11210 -b bucketname -p bucketpassword set tap_param tap_throttle_queue_cap 2000000
```

In this example we specify that a node sends replication backoff requests 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:

```
shell> ./cbepctl  10.5.2.31:11210 set -b bucketname 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 node reaches this point, it will send replication backoff requests to other nodes.

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.

For more information about changing this setting, see Section 7.6, "**cbepctl** Tool". You can also monitor the progress of this backoff operation in Couchbase Web Console under Tap Queue Statistics | back-off rate. For more information, see Section 6.4.1.4, "Monitoring TAP Queues".

5.4. Ejection and Working Set Management

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 will remove certain data to create free space and to maintain system performance. This process is called *working set management* and we refer to the set of data in RAM as a *working set*.

In general your working set consists of all the keys, metadata, and associated documents which are frequently used in your system and therefore require fast access. The process the server performs to remove data from RAM is known as *ejection*, and when the server performs this process, it removes the document, but not the keys or metadata for an 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; the server will then try 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 you quickly determine if an item is missing and then 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. For more information about the pager, and changing the setting for it, see Section 7.6.1, "Changing the Disk Cleanup Interval".

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 NRU, which is an abbreviation for *not-recently-used*. 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.

As of Couchbase Server 2.0.1+ we provide two NRU bits per item and also provide a replication protocol that can propagate items that are frequently read, but not mutated often. For earlier versions of Couchbase Server, we had 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 in 2.0.1+ the working set in replica vBuckets will be closer to the working set in the active vBucket.

NRUs will be decremented or incremented by server processes to indicate an item is more frequently used, or less frequently used. Items with lower bit values will have lower scores and will be 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: 1) if a client reads or writes an item, the server decrements NRU and lowers the item's score, 2) Couchbase Server also has a daily process which creates a list of frequently-used items in RAM. After this process runs, the server increment one of the NRU bits. Because two processes will change NRUs, they will also affect which items are candidates for ejection. For more information about the access scanner, see Section 5.2, "Handling Server Warmup".

You can adjust settings for Couchbase Server which change behavior during ejection. You can indicate the percentage of RAM you are willing to consume before items are ejected, or you can indicate whether ejection should occur more frequently on replicated data than on original data. Be aware that for Couchbase Server 2.0+, we recommend that you remain using the defaults provided.

Understanding the Item Pager

The process that periodically runs and removes documents from RAM is known as the *item pager*. When a threshold known as *low water mark* is reached, this process starts ejecting inactive replica data from RAM on the node. If the amount of RAM used by items reaches an upper threshold, known as the *high water mark*, both replica data and active data written from clients will be ejected. The item pager will continue to eject items from RAM until the amount of RAM consumed is below the *low water mark*. Both the high water mark and low water mark are expressed as an absolute amount of RAM, such as 5577375744 bytes.

When you change either of these settings, you can provide a percentage of total RAM for a node such as 80% or as an absolute number of bytes. For Couchbase Server 2.0 and above, we recommend you remain using the default settings provided. Defaults for these two settings are listed below.

Version	High Water Mark	Low Water Mark
2.0	75%	60%
2.0.1+	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:

Version	Probability for Active vBucket	Probability for Replica vBucket
2.0+	60%	40%

For instructions to change this setting, see Section 7.6.4, "Changing Thresholds for Ejection".

5.5. Database and View Compaction

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 can eventually lead 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.

5.5.1. Compaction Process

How it works

Couchbase compacts views and data files. For database compaction, a new file is created into which the active index 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. See Section 6.7, "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:

```
Service couchbase_compaction_daemon exited on node 'ns_1@127.0.0.1' in 0.00s
(repeated 1 times)
```

For information on accessing the log, see Section 6.7, "Log".

5.5.2. 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. For more information, see Section 8.6.10, "Compacting Bucket Data and Indexes" [222].

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. For more information, see Section 6.8, "Settings".

- *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. See Section 6.3.1, "Creating and Editing Data Buckets".

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. See Time Period [75].

- **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.

  Note

  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:

  - **Enabled**

    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.

  - **Disabled**

    If compaction is running when the time period ends, compaction will continue until 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.

  By enabling 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.

Configuration of auto-compaction is performed through the Couchbase Server Web Admin Console. For more information on the default settings, see Section 6.8, "Settings". Information on per-bucket settings is through the Couchbase Bucket create/edit screen. See Section 6.3.1, "Creating and Editing Data Buckets".

5.5.3. 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.

5.6. Failing Over Nodes

If a node in a cluster is unable to serve data you can *failover* that node. Failover means that Couchbase Server removes the node from a cluster and makes replicated data at other nodes available for client requests. 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. See Performing a Rebalance, Adding a Node to a Cluster.

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. For more information on choosing whether to use automated or manual failover see Section 5.6.1, "Choosing a Failover Solution".

  For information on how to enable and monitor automatic failover, see Section 5.6.2, "Using Automatic Failover".

- **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. Information on handling a failover is in Section 5.6.4, "Handling a Failover Situation".

- **Adding nodes after failover**. After you resolve the issue with the failed over node you can add the node back to your cluster. Information about this process is in Section 5.6.5, "Adding Back a Failed Over Node".

5.6.1. 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.

5.6.1.1. 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**

If you have a network partition across the nodes in a Couchbase cluster, automatic failover would lead to nodes on both sides of the partition to automatically failover. Each functioning section of the cluster would now have to assume responsibility for the entire document ID space. While there would be consistency for a document ID within each partial cluster, there would start to be inconsistency of data between the partial clusters. Reconciling those differences may be difficult, depending on the nature of your data and your access patterns.

Assuming one of the two partial clusters is large enough to cope with all traffic, the solution is to direct all traffic for the cluster to that single partial cluster. The separated nodes could then be re-added to the cluster to bring the cluster to its original size.

- **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.

5.6.1.2. 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 Management 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.

5.6.2. 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. For more information about potential issues, see Choosing a Failover Solution.

- **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. For more information about this setting, see Enabling and Disabling Auto-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. For more information, see Enabling and Disabling Email Notifications .

To configure automatic failover through the Administration Web Console, see Section 6.8.2, "Enabling Auto-Failover Settings". For information on using the REST API, see Section 8.7.8, "Retrieving Auto-Failover Settings".

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. For more information, see Section 5.6.4, "Handling a Failover Situation".

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.

> **Warning**
>
> You should only resetting the automatic failover after you resolve the node issue, rebalance and restore the cluster to a fully functioning state.

You can reset the counter using the REST API:

```
shell> curl -i -u cluster-username:cluster-password \
    http://localhost:8091/settings/autoFailover/resetCount
```

For more information on using this REST API see Resetting Auto-Failover.

5.6.3. 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.

> **Warning**
>
> Before you indicate the failover for a node you should read Section 5.6, "Failing Over Nodes". 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 will activate replicated data at other nodes which will reduce 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.

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:

```
shell> 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. For more information, see Section 5.6.4, "Handling a Failover Situation".

5.6.4. 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. For more information on adding new nodes, and performing the rebalance operation, see Section 5.8.2, "Performing a Rebalance".

5.6.5. 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.

For more information on adding a node to the cluster and rebalancing, see Section 5.8.2, "Performing a Rebalance".

5.7. Backup and Restore

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.

  For more information, see Section 5.7.1, "Backing Up Using **cbbackup**".

  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.

  For more information, see Section 5.7.1.2, "Backing Up Using File Copies".

  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.

For detailed information on the restore processes and options, see Section 5.7.2, "Restoring Using **cbrestore**".

Note

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.

5.7.1. 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. See Chapter 7, *Command-line Interface for Administration*.
>
> 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. To change the address format into a hostname format for the server, see Section 2.4, "Using Hostnames with Couchbase Server".

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. For a full list of the supported arguments, see Section 7.8, "**cbbackup** Tool".

  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:

    ```
    shell> 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:

```
shell> 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:

```
shell> 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:

```
shell> 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:

```
shell> ssh USER@HOST
remote-shell> sudo su - couchbase
remote-shell> 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:

```
shell> ssh USER@HOST
remote-shell> sudo su - couchbase
remote-shell> 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.

5.7.1.1. 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':

```
shell> 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**. For more information, see Section 5.7.2.2, "Restoring using **cbrestore** tool".

5.7.1.2. 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**:

```
shell> cbbackup \
    couchstore-files:///opt/couchbase/var/lib/couchbase/data/default \
    /mnt/backup-20120501
```

The same backup operation using **cp**:

```
shell> 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.

5.7.2. 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**. See Section 5.7.2.2, "Restoring using **cbrestore** tool".

If the information was backed up using a direct file copy, then you must restore the information back to an identical cluster. See Section 5.7.2.1, "Restoring Using File Copies".

5.7.2.1. 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.

Warning

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. For more information, see Section 3.2, "Server Startup and Shutdown".

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. For more information, see Section 3.2, "Server Startup and Shutdown".

5.7.2.2. Restoring using cbrestore tool

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.

  For information on all the options available when using **cbrestore**, see Section 7.9, "**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:

```
shell> 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:

```
shell> 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.

5.7.2.2.1. 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 (see Section 5.7.1.1, "Filtering Keys 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':

```
shell> 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.

5.7.3. Backup and Restore Between Mac OS X and Other Platforms

Couchbase Server 2.0 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:

  ```
  shell> 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:

  ```
  shell> 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:

  ```
  shell> 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:

  ```
  shell> 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:

  ```
  shell> cbtransfer http://linux:8091 memcached://mac:11211 -b default -B default
  shell> cbtransfer http://mac:8091 memcached://linux:11211 -b default -B default
  ```

  Note

  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:

  ```
  shell> cbbackup http://Administrator:password@linux:8091 /linuxbackup/today
  ```

  Remove or move the data files from the backup out of the way:

  ```
  shell> 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:

  ```
  shell> cbrestore /linuxbackup/today http://mac:8091 -b default -B default
  ```

5.8. 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.

For information and guidance on choosing how, and when, to rebalance your cluster, read Section 5.8.1, "Choosing When to Rebalance". This will provide background information on the typical triggers and indicators that your cluster requires changes to the node configuration, and when a good time to perform the rebalance is required.

Instructions on how to expand and shrink your cluster, and initiate the rebalance operation are provided in Section 5.8.2.3, "Performing a Rebalance".

Once the rebalance operation has been initiated, you should monitor the rebalance operation and progress. You can find information on the statistics and events to monitor using Section 5.8.4, "Monitoring a Rebalance".

Common questions about the rebalancing operation are located in Section 5.8.5, "Common Rebalancing Questions".

For a deeper background on the rebalancing and how it works, see Section 5.8.7, "Rebalance Behind-the-Scenes".

5.8.1. Choosing When to Rebalance

Choosing when each of situations applies is not always straightforward. Detailed below is the information you need to choose when, and why, to rebalance your cluster under different scenarios.

Choosing when to expand the size of 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.

  There are two 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.

> **Choosing 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.

Choosing when to Rebalance

Once you decide to add or remove nodes to your Couchbase Server cluster, there are a few things to take into consideration:

- 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.

5.8.2. Performing a Rebalance

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.

Caution

Until you complete a rebalance, you should avoid using the failover functionality since that may result in loss of data that has not yet been replicated.

- For information on adding nodes to your cluster, see Section 5.8.2.1, "Adding a Node to a Cluster".

- For information on removing nodes to your cluster, see Section 5.8.2.2, "Removing a Node from a Cluster".

- 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. For more information on how to handle a failover situation, see Section 5.6, "Failing Over Nodes"

The Couchbase Admin Web Console will indicate 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. You can see a sample of this in the figure below, here shown on the Manage Server Nodes page.

Figure 5.7. Rebalancing — Servers Pending Rebalance
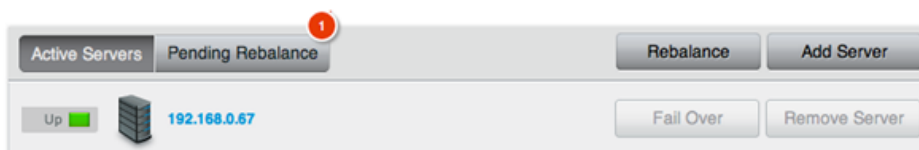
To rebalance the cluster, you must initiate the rebalance process, detailed in Section 5.8.2.3, "Performing a Rebalance".

5.8.2.1. Adding a Node to a Cluster

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. The methods are:

- **Web Console — During Installation**

When you are performing the Setup of a new Couchbase Server installation (see Section 2.3, "Initial Server Setup"), 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:

Figure 5.8. Rebalancing — Adding new node during setup

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:

Figure 5.9. Rebalancing — Node added during setup

- **Web Console — After Installation**

  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.

Figure 5.10. Rebalancing — Adding new node using the Web Console
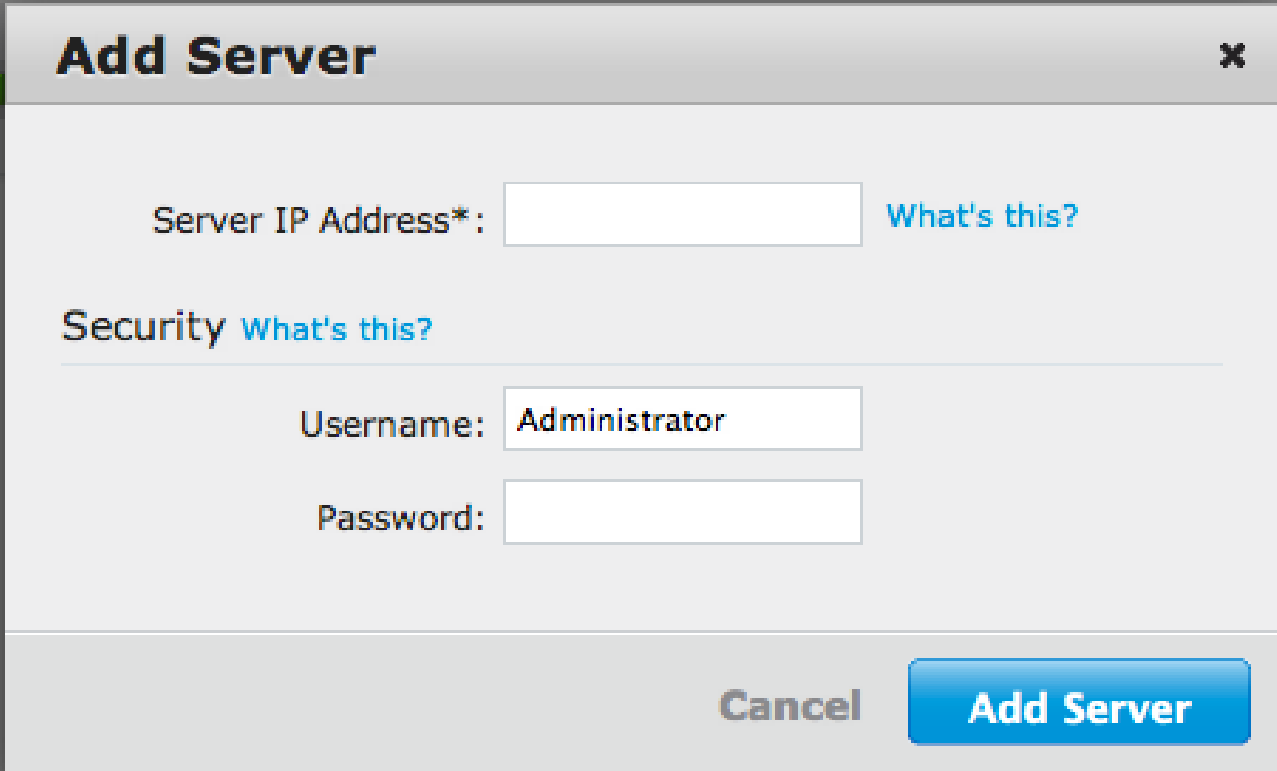
You will be presented with a dialog box, as shown below. Couchbase Server should be installed, and should have been configured as per the normal setup procedures. You can also add a server that has previously been part of this or another cluster using this method. The Couchbase Server must be running.

Figure 5.11. Rebalancing — Adding a new node dialog

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.

- **Using the REST API**

  Using 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:

```
shell> curl -u cluster-username:cluster-password\
    localhost:8091/controller/addNode \
    -d "hostname=192.168.0.68&user=node-username&password=node-password"
```

For more information, see Section 8.7.2, "Adding a Node to a Cluster".

- **Using the Command-line**

  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:

  ```
  shell> 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.

  > **Tip**
  >
  > You can add multiple nodes in one command by supplying multiple --server-add command-line options to the command.

  Once a server has been successfully added, the Couchbase Server cluster will indicate that a rebalance is required to complete the operation.

  > **Note**
  >
  > You can cancel the addition of a node to a cluster without having to perform a rebalance operation. Canceling the operation will remove 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.

5.8.2.2. Removing a Node from a Cluster

Removing a node marks the node for removal from the cluster, and will completely disable the node from serving any requests across the cluster. Once removed, a node is no longer part of the cluster in any way and can be switched off, or can be updated or upgraded.

> **Best Practice: Ensure Capacity for Node Removal**
> Before you remove a node from the cluster, you should ensure that you have the capacity within the remaining nodes of your cluster to handle your workload. For more information on the considera-

> tions, see Choosing when to shrink your cluster [91] For the best results, use swap rebalance to swap the node you want to remove out, and swap in a replacement node. For more information on swap rebalance, see Section 5.8.3, "Swap Rebalance".

Like adding nodes, there are a number of solutions for removing a node:

- **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:

```
shell> couchbase-cli rebalance --cluster=127.0.0.1:8091 \
        -u Administrator -p Password \
        --server-remove=192.168.0.73
```

  For more information on the rebalance operation, see Section 5.8.2.3, "Performing a Rebalance".

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.

5.8.2.3. 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. This 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:

- **Using the Web Console**

  Within the Manage Server Nodes area of the Couchbase Administration Web Console, a cluster pending a rebalance operation will have enabled the `Rebalance` button.

Figure 5.12. Rebalancing — Starting a Rebalance

Clicking this button will immediately initiate a rebalance operation. You can monitor the progress of the rebalance operation through the web console.

You can stop a rebalance operation at any time during the process by clicking the `Stop Rebalance` button. This only stops the rebalance operation, it does not cancel the operation. You should complete the rebalance operation.

- **Using the Command-line**

  You can initiate a rebalance using the **couchbase-cli** and the `rebalance` command:

  ```
  shell> 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:

  ```
  shell> 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.

  You can stop the rebalance operation by using the `stop-rebalance` command to **couchbase-cli**.

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 you should monitor the process to ensure that it completes successfully, see Section 5.8.4, "Monitoring a Rebalance".

5.8.3. 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 only occurs if the following are true:

• You are removing and adding the same number of nodes during rebalance. For example, if you have marked two nodes to be removed, and added another two nodes to the cluster.

Note

Swap rebalance occurs automatically if the number of nodes being added and removed are identical. There is no configuration or selection mechanism to force a swap rebalance. If a swap rebalance cannot take place, then a normal rebalance operation will be used instead.

When Couchbase Server identifies that a rebalance is taking place and that there are an even number of nodes being removed and added to the cluster, the swap rebalance method is used to perform the rebalance operation.

When a swap rebalance takes place, the rebalance operates as follows:

• Data will be 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 will be moved to node X.

• vBuckets from node B will be 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 behaviour 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.

  Best Practice: Failed Over Nodes
  With swap rebalance functionality, after a node has failed over, you should either clean up and re-add the failed over node, or add a new node and perform a rebalance as normal. The rebalance will be handled as a swap rebalance which will minimize the data movements without affecting the overall capacity of the cluster.

5.8.4. Monitoring a Rebalance

You should monitor the system during and immediately after a rebalance operation until you are confident that replication has completed successfully.

As of Couchbase Server 2.1 we provide a detailed rebalance report in Web Console. As the server moves vBuckets within the cluster, Web Console provides a detailed report. You can view the same statistics in this report via a REST API call, see Section 8.7.6, "Getting Rebalance Progress". If you click on the drop-down next to each node, you can view the detailed rebalance status:

Figure 5.13. Monitoring a Rebalance
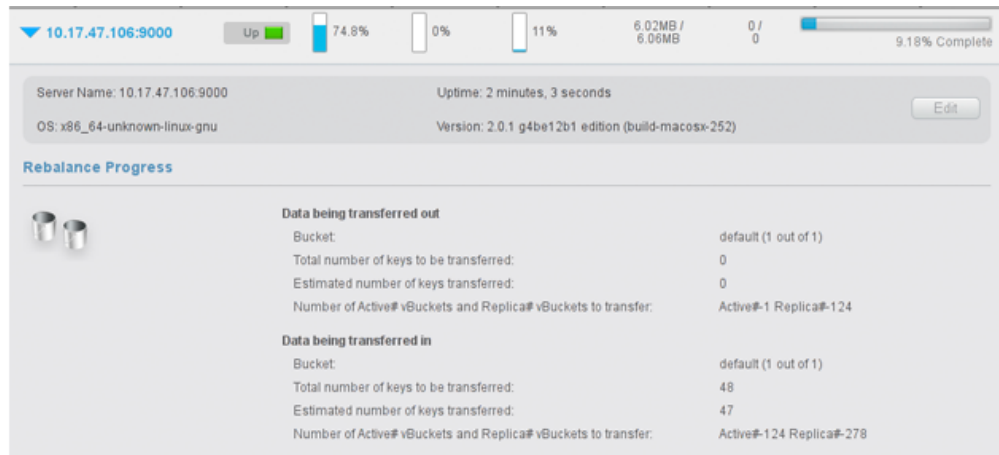
The section `Data being transferred out` means that a node sends data to other nodes during rebalance. The section `Data being transferred in` means that a node receives data from other nodes during rebalance. A node can be either a source, a destination, or both a source and destination for data. The progress report displays the following information:

- **Bucket**: Name of bucket undergoing rebalance. Number of buckets transferred during rebalance out of total buckets in cluster.

- **Total number of keys**: Total number of keys to be transferred during the rebalance.

- **Estimated number of keys**: Number of keys transferred during rebalance.

- **Number of Active# vBuckets and Replica# vBuckets**: Number of active vBuckets and replica vBuckets to be transferred as part of rebalance.

You can also use **cbstats** to see underlying rebalance statistics:

- **Backfilling**

The first stage of replication reads all data for a given active vBucket and sends it to the server that is responsible for the replica. This can put increased load on the disk as well as network bandwidth but it is not designed to impact any client activity. You can monitor the progress of this task by watching for ongoing TAP disk fetches. You can also watch **cbstats tap**, for example:

```
cbstats <node_IP>:11210 -b bucket_name -p bucket_password tap | grep backfill
```

This will return a list of TAP backfill processes and whether they are still running (true) or done (false). During the backfill process for a particular tap stream you will see output as follows:

```
eq_tapq:replication_building_485_'n_1@127.0.0.1':backfill_completed: false
 eq_tapq:replication_building_485_'n_1@127.0.0.1':backfill_start_timestamp: 1371675343
 eq_tapq:replication_building_485_'n_1@127.0.0.1':flags: 85 (ack,backfill,vblist,checkpoints)
 eq_tapq:replication_building_485_'n_1@127.0.0.1':pending_backfill: true
 eq_tapq:replication_building_485_'n_1@127.0.0.1':pending_disk_backfill: true
 eq_tapq:replication_building_485_'n_1@127.0.0.1':queue_backfillremaining: 202
```

When all have completed, you should see the Total Item count (`curr_items_tot`) be equal to the number of active items multiplied by replica count. The output you see for a TAP stream after backfill completes is as follows:

```
eq_tapq:replication_building_485_'n_1@127.0.0.1':backfill_completed: true
 eq_tapq:replication_building_485_'n_1@127.0.0.1':backfill_start_timestamp: 1371675343
 eq_tapq:replication_building_485_'n_1@127.0.0.1':flags: 85 (ack,backfill,vblist,checkpoints)
 eq_tapq:replication_building_485_'n_1@127.0.0.1':pending_backfill: false
 eq_tapq:replication_building_485_'n_1@127.0.0.1':pending_disk_backfill: false
 eq_tapq:replication_building_485_'n_1@127.0.0.1':queue_backfillremaining: 0
```

If you are continuously adding data to the system, these values may not correspond exactly at a given instant in time. However you should be able to determine whether there is a significant difference between the two figures.

- **Draining**

After the backfill process is complete, all nodes that had replicas materialized on them will then need to persist those items to disk. It is important to continue monitoring the disk write queue and memory usage until the rebalancing operation has been completed, to ensure that your cluster is able to keep up with the write load and required disk I/O.

5.8.5. Common Rebalancing Questions

Provided below are some common questions and answers for the rebalancing operation.

- **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.

5.8.6. Rebalance Effect on Bucket Types

The rebalance operation works across the cluster on both Couchbase and `memcached` buckets, but there are differences in the rebalance operation 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.

5.8.7. Rebalance Behind-the-Scenes

The rebalance process is managed through a specific process called the *orchestrator*. This 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.

5.9. Cross Datacenter Replication (XDCR)

Couchbase Server 2.0 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:

Figure 5.14. Replication within a Cluster
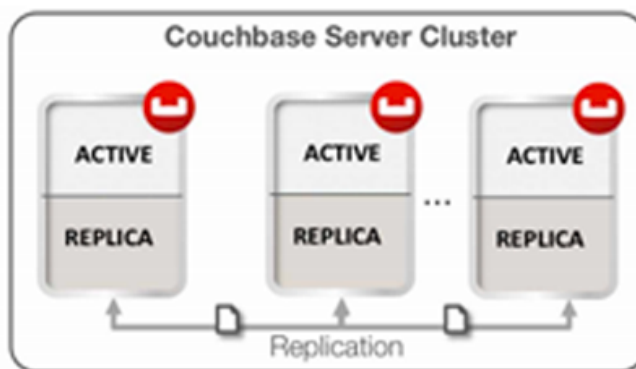
Cross datacenter replication in Couchbase Server involves replicating active data to multiple, geographically diverse datacenters either for disaster recovery or to bring data closer to its users for faster data access, as shown in below:

Figure 5.15. Cross Data Center Replication
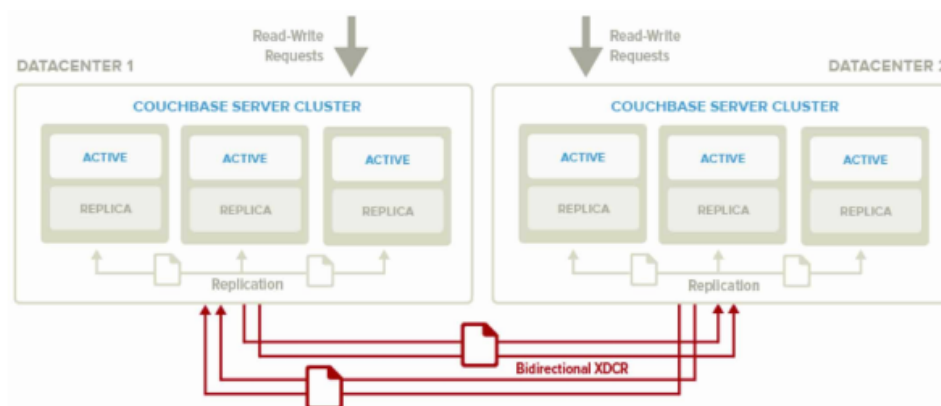
You can also see that XDCR and intra-cluster replication occurs simultaneously. Intra-cluster replication is taking place within the clusters at both Datacenter 1 and Datacenter 2, while at the same time XDCR is replicating documents across datacenters. Both datacenters are serving read and write requests from the application.

5.9.1. 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.

5.9.2. 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:

Figure 5.16. Replicating Selected Buckets via XDCR
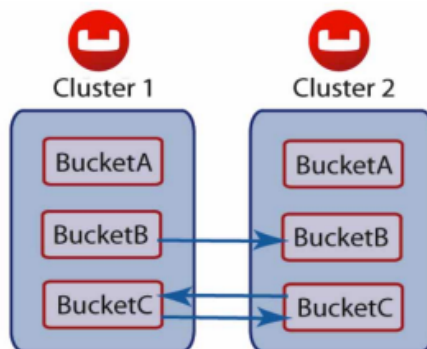
As shown above, after the document is stored in Couchbase Server and before XDCR replicates a document to other datacenters, a couple of things happen within each Couchbase Server node.

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:

Figure 5.17. XDCR Triggered after Disk Persistence
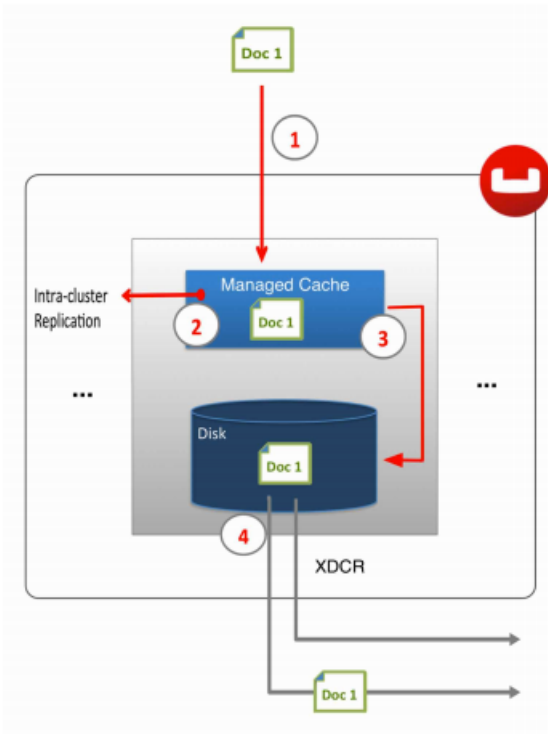
5.9.3. XDCR Architecture

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. For information on changing the number of data streams for replication, see Section 5.9.9, "Changing XDCR Settings"

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:

Figure 5.18. Conflict Resolution in XDCR
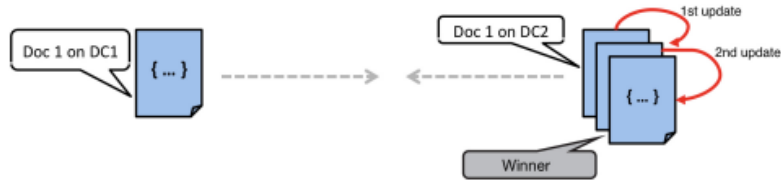
As shown in above, bidirectional replication is set up between Datacenter 1 and Datacenter 2 and both the clusters start off with the same JSON document (Doc 1). In addition, two additional updates to Doc 1 happen on Datacenter 2. In the case of a conflict, Doc 1 on Datacenter 2 is chosen as the winner because it has seen more updates.

5.9.4. Advanced Topologies

By combining unidirectional and bidirectional topologies, you have the flexibility to create several complex topologies such as the chain and propagation topology as shown below:

Figure 5.19. Replication Chain with Uni-Directional Replication
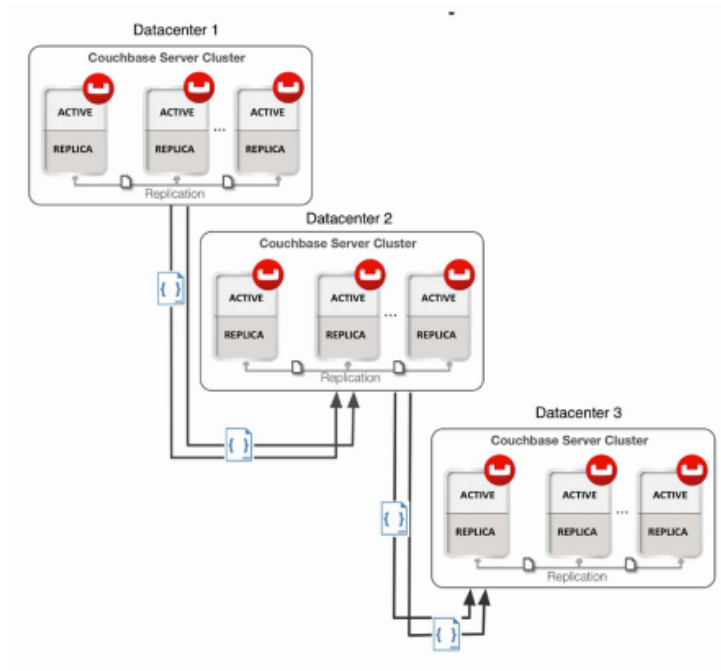
In the image below there is one bidirectional replication link between Datacenter 1 and Datacenter 2 and two unidirectional replication links between Datacenter 2 and Datacenters 3 and 4. Propagation replication can be useful in a scenario when you want to setup a replication scheme between two regional 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:

Figure 5.20. Bidirectional and Unidirectional Replication for Selective Replication

A description of the functionality, implementation and limitations of XDCR are provided in Section 5.9.8, "Behavior and Limitations".

To create and configure replication, see Section 5.9.5, "Configuring Replication".

5.9.5. Configuring Replication

You configure replications using the XDCR tab of the Administration Web Console. You configure replication on a bucket basis. If you want to replicate data from all buckets in a cluster, you should individually configure replication for each bucket.

Before You Configure XDCR

- All nodes within each cluster must be configured to communicate with all the nodes on the destination cluster. XDCR will use any node in a cluster to replicate between the two clusters.

- Couchbase Server versions and platforms, must 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.

To create a uni-directional replication (i.e. from cluster A to cluster B):

1. Check and ensure that a destination bucket exists on the cluster to which you will be replicating. To do so, perform this REST API request:

```
curl -u Admin:password http://ip.for.destination.cluster:8091/pools/default/buckets
```

2. To set up a destination cluster reference, click the `Create Cluster Reference` button. You will be prompted to enter a name used to identify this cluster, the IP address, and optionally the administration port number for the remote cluster.

Figure 5.21. Couchbase Web Console - Replication Cluster Reference

Enter the username and password for the administrator on the destination cluster.

3. Click Save to store new reference to the destination cluster. This cluster information will now be available when you configure replication for your source cluster.

4. Click `Create Replication` to configure a new XDCR replication. A panel appears where you can configure a new replication from source to destination cluster.

5. In the `Replicate changes from` section select a from the current cluster that is to be replicated. This is your source bucket.

6. In the `To` section, select a destination cluster and enter a bucket name from the destination cluster:

Figure 5.22. Couchbase Web Console - Replication Configuration

7. Click the `Replicate` button to start the replication process.

After you have configured and started replication, the web console will show the current status and list of replications in the Ongoing Replications section:

Figure 5.23. Couchbase Web Console - Replication Monitoring

Configuring Bi-Directional Replication

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

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.

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

You can also create a replication using the Administration REST API instead of Couchbase Web Console. For more information, see Section 8.9.1, "Getting a Destination Cluster Reference".

After you create a replication between clusters, you can configure the number of parallel replicators that run per node. The default number of parallel, active streams per node is 32, but you can adjust this. For information on changing the internal configuration settings, see Section 8.9.6, "Viewing Internal XDCR Settings".

5.9.6. Monitoring Replication Status

There are two different areas of Couchbase Web Console which contain information about replication via XDCR: 1) the XDCR tab, and 2) the outgoing XDCR section under the Data Buckets tab.

The Couchbase Web Console will display 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 you configure bi-directional you should use the web consoles that belong to source and destination clusters to monitor both clusters.

To see statistics on incoming and outgoing replications via XDCR see the following:

• Incoming Replications, see Section 6.4.1.7, "Monitoring Incoming XDCR".

• Outgoing Replications, see Section 6.4.1.6, "Monitoring Outgoing XDCR".

Any errors that occur during replication appear in the XDCR errors panel. In the example below, we show the errors that occur if replication streams from XDCR will fail due to the missing vBuckets:

Figure 5.24. Errors Panel for XDCR

You can tune your XDCR parameters by using the administration REST API. See Section 8.9.6, "Viewing Internal XDCR Settings".

5.9.7. Cancelling Replication

You can cancel replication at any time by clicking `Delete` next to the active replication that is to be canceled.

A prompt will confirm the deletion of the configured replication. Once the replication has been stopped, replication will cease on the originating cluster on a document boundary.

Canceled replications that were terminated while the replication was still active will be displayed within the Past Replications section of the Replications section of the web console.

5.9.8. Behavior and Limitations

- **Network and System Outages**

  - XDCR is resilient to intermittent network failures. In the event that the destination cluster is unavailable due to a network interruption, XDCR will pause replication and will then retry the connection to the cluster every 30 seconds. Once XDCR can successfully reconnect with a destination cluster, it will resume replication. In the event of a more prolonged network failure where the destination cluster is unavailable for more than 30 seconds, a source cluster will continue polling the destination cluster which may result in numerous errors over time. In this case, you should delete the replication in Couchbase Web Console, fix the system issue, then re-create the replication. The new XDCR replication will resume replicating items from where the old replication had been stopped.

  - Your configurations will be retained over host restarts and reboots. You do not need to re-configure your replication configuration in the event of a system failure.

- **Document Handling**

  - XDCR does not replicate views and view indexes; you must 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 and are not transferred to the remote cluster.

- **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.

5.9.8.1. 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.

5.9.8.2. 'Optimistic Replication' in XDCR

In Couchbase 2.1 you can also tune the performance of XDCR with a new parameter, `xdcrOptimisticReplicationThreshold`. 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.

As of Couchbase Server 2.1, XDCR will 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. For more information, see Section 8.9.7, "Changing Internal XDCR Settings".

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`:

Figure 5.25. Monitoring "Optimistic Replication"

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/couch-base/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%3ba19c9d4e733a97fa7cb38da
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. For more information about XDCR, see Section 5.9, "Cross Datacenter Replication (XDCR)".

5.9.9. Changing XDCR Settings

Besides Couchbase Web Console, you can use several Couchbase REST-API endpoints to modify XDCRsettings. Some of these settings are references used in XDCR and some of these settings will change XDCR behavior or performance:

- Viewing, setting and removing destination cluster references, can be found in Section 8.9.1, "Getting a Destination Cluster Reference", Section 8.9.2, "Creating a Destination Cluster Reference" and Section 8.9.3, "Deleting a Destination Cluster Reference".

- Creating and removing a replication via REST can be found in Section 8.9.2, "Creating a Destination Cluster Reference" and Section 8.9.3, "Deleting a Destination Cluster Reference".

- Concurrent replications, which is the number of concurrent replications per Couchbase Server instance. See Section 8.9.6, "Viewing Internal XDCR Settings".

- 'Optimistic Replication.' For more information about 'optimistic replication', see Section 5.9.8.2, "'Optimistic Replication' in XDCR".

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 that if you are using XDCR on multiple nodes in cluster and you want to change this setting throughout the cluster, you will need to 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: <http://Administrator/>asdasd@127.0.0.1:9000/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 will load this parameter. If you set both the environment variable and the server parameter, the value for the environment parameter will supersede.

5.9.10. Securing Data Communication with XDCR

When configuring XDCR across multiple clusters over public networks, the data is sent unencrypted across the public interface channel. To ensure security for the replicated information you will need to configure a suitable VPN gateway between the two datacenters that will encrypt the data between each route between datacenters.

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:

Figure 5.26. XDCR — Using Static VPN Routes
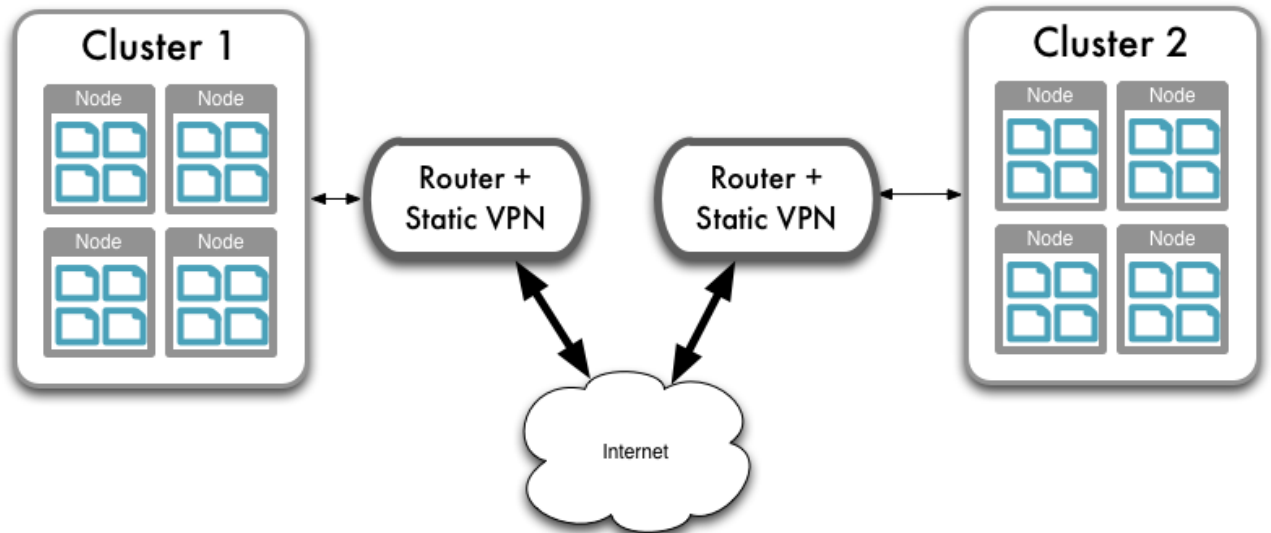
When using Amazon EC2 or other cloud deployment solutions, particularly when using different EC2 zones, there is no built-in VPN support between the different EC2 regional zones. However, there is VPN client support for your cluster within EC2 and Amazon VPC to allow communication to a dedicated VPN solution. For more information, see Amazon Virtual Private Cloud FAQs for a list of supported VPNs.

To support cluster to cluster VPN connectivity within EC2 you will need to configure a multi-point BGP VPN solution that can route multiple VPN connections. You can then route the VPN connection from one EC2 cluster and region to the third-party BGP VPN router, and the VPN connection from the other region, using the BGP gateway to route between the two VPN connections.

Figure 5.27. XDCR — Using Third-party BGP Routing
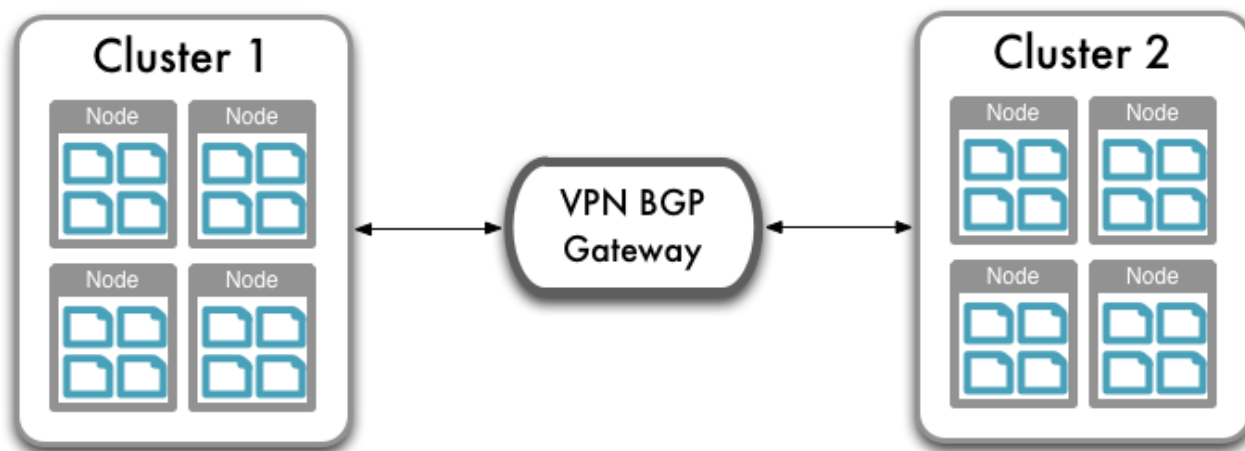
Configuration of these VPN routes and systems is dependent on your VPN solution.

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.

You must ensure when configuring your VPN connection 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.

5.9.11. Using 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.

You should configure the cluster with a fixed IP address and the public DNS name according to the information in Section 4.6.2, "Handling Changes in IP Addresses".

• 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.

Node Group	Ports	IP Addresses
Nodes within cluster	4369, 8091, 8092,9, 11210, 21100-21199	IP of cluster nodes
XDCR Nodes	8091, 8092	IP of remote clusters

For more information in general about using Couchbase Server in the cloud, see Section 4.6, "Using Couchbase in the Cloud".

5.10. Changing the Configured Disk Path

You cannot change the disk path where the data and index files are stored 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 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. For more information, see Section 5.8.3, "Swap Rebalance".

To change the disk path of the existing node, the recommended sequence is:

1. Remove the node where you want to change the disk path from the cluster. For more information, see Section 5.8.2.2, "Removing a Node from a Cluster". To ensure the performance of your cluster is not reduced, perform a swap rebalance with a new node (see Section 5.8.3, "Swap Rebalance").

2. Perform a rebalance operation, see Section 5.8.2.3, "Performing a Rebalance".

3. Configure the new disk path, either by using the REST API (see Section 8.5.3, "Configuring Index Path for a Node"), using the command-line (see cluster initialization for more information).

   Alternatively, connect to the Web UI of the new node, and follow the setup process to configure the disk path (see Section 2.3, "Initial Server Setup".

4. Add the node back to the cluster, see Section 5.8.2.1, "Adding a Node to a Cluster".

The above process will change the disk path only on the node you removed from the cluster. To change the disk path on multiple nodes, you will need to swap out each node and change the disk path individually.

Chapter 6. Using the Web Console

The Couchbase Web Console is the main tool for managing your Couchbase installation. The Web Console provides the following tabs:

- Cluster Overview: a quick guide to the status of your Couchbase cluster.

  For more information, read Section 6.1, "Viewing Cluster Summary".

- Data Buckets: view and update data bucket settings. You can create new buckets, edit existing settings, and see detailed statistics on the bucket.

  See Section 6.3, "Viewing Data Buckets".

- Server Nodes: shows your active nodes, their configuration and activity. Under this tab you can also fail over nodes and remove them from your cluster, view server-specific performance, and monitor cluster statistics.

  Read Section 6.2, "Viewing Server Nodes".

- Views: is were you can create and manage your views functions for indexing and querying data. Here you can also pre-view results from views.

  See Section 6.5, "Using the Views Editor" for the views editor in Web Console. For more information on views in general, see Chapter 9, *Views and Indexes*.

- Documents: you can be create and edit documents under this tab. This enables you to view and modify documents that have been stored in a data bucket and can be useful when you work with views.

  See Section 6.6, "Using the Document Editor".

- Log: displays errors and problems.

  See Section 6.7, "Log" for more information.

- Settings: under this tab you can configure the console and cluster settings.

  See Section 6.8, "Settings" for more information.

In addition to these sections of the Couchbase Web Console, there are additional systems within the web console, including:

- Update Notifications

  Update notifications indicates when there is an update available for the installed Couchbase Server. See Section 6.9, "Updating Notifications" for more information on this feature.

- Warnings and Alerts

  Warnings and alerts in Web Console will notify you when there is an issue that needs to be addressed within your cluster. The warnings and alerts can be configured through Section 6.8, "Settings".

  For more information on the warnings and alerts, see Section 6.10, "Warnings and Alerts".

6.1. Viewing Cluster Summary

Cluster Overview is the home page for the Couchbase Web Console. The page provides an overview of your cluster health, including RAM and disk usage and activity.

Figure 6.1. Web Console — Cluster Overview
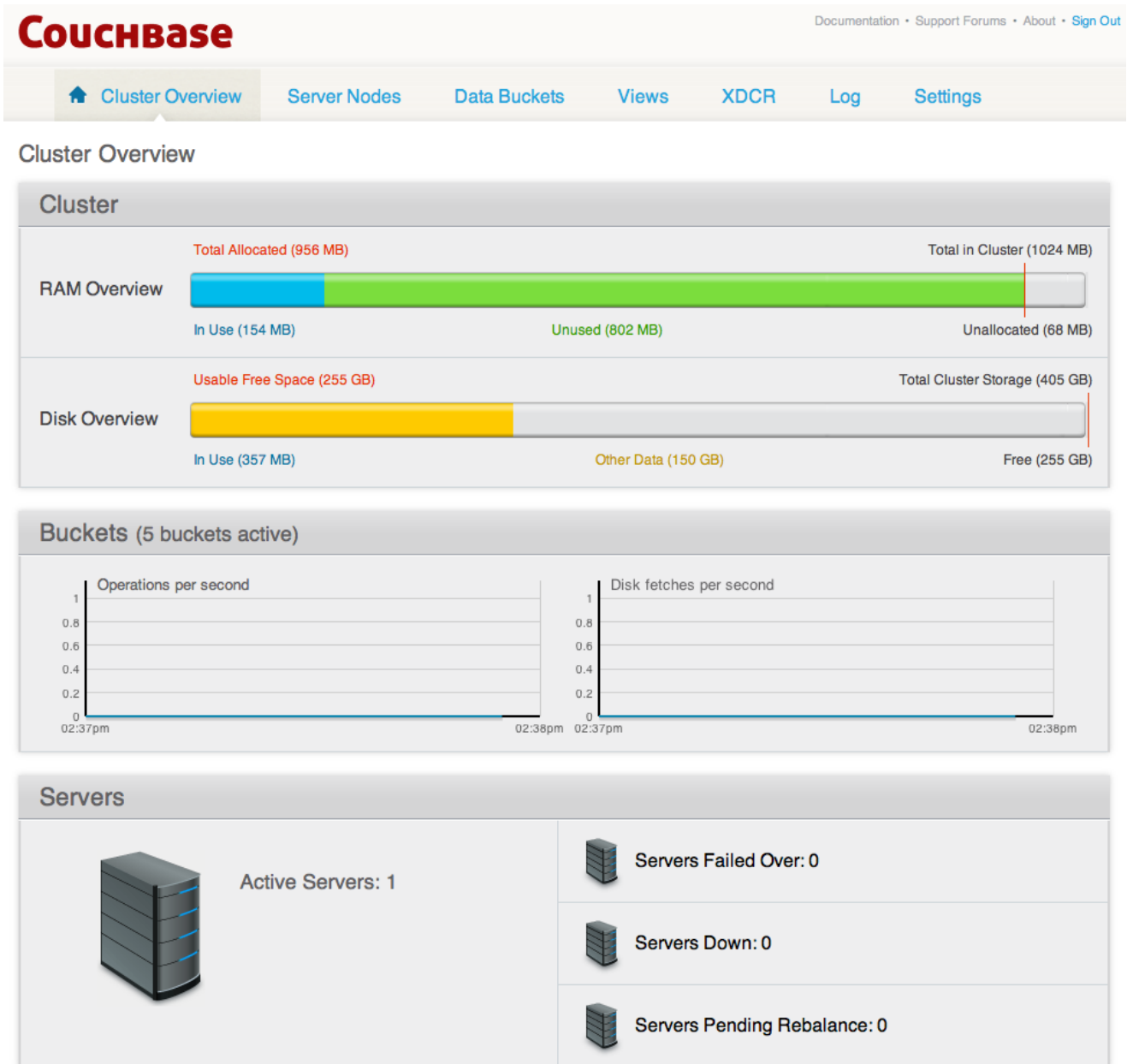
The page is divided into Cluster, Buckets, and Servers sections.

6.1.1. Viewing Cluster Overview

The Cluster section provides information on the RAM and disk usage information for your cluster.

Figure 6.2. Web Console — Cluster Overview — Cluster
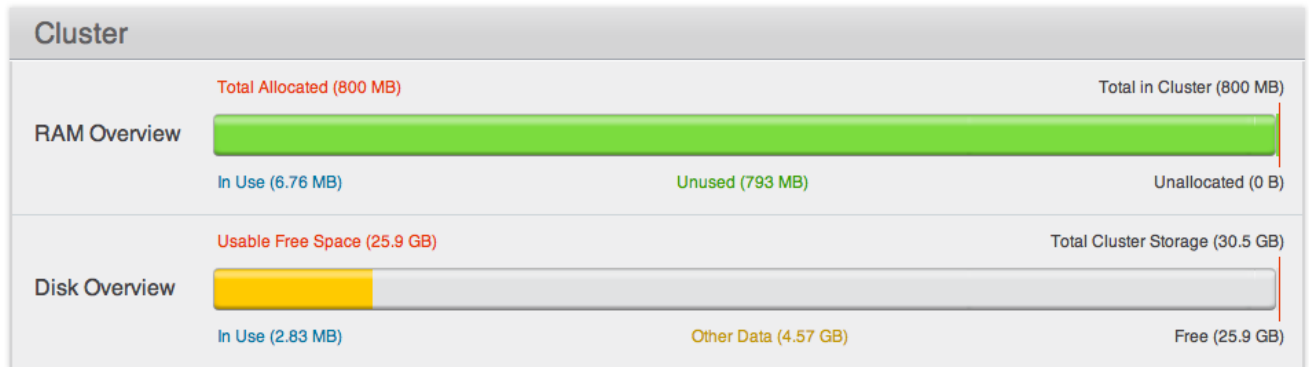
For the RAM information you are provided with a graphical representation of your RAM situation, including:

• 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.

6.1.2. Viewing Buckets

The Buckets section provides two graphs showing the Operations per second and Disk fetches per second.

Figure 6.3. Web Console — Cluster Overview — Buckets
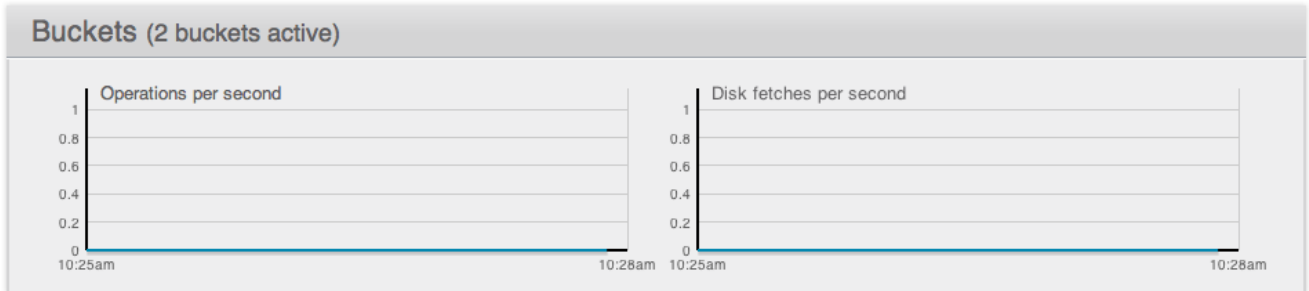
The Operations per second provides information on the level of activity on the cluster in terms of storing or retrieving objects from the data store.

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.

6.1.3. Viewing Servers

The Servers section indicates overall server information for the cluster:

Figure 6.4. Web Console — Cluster Overview — Servers
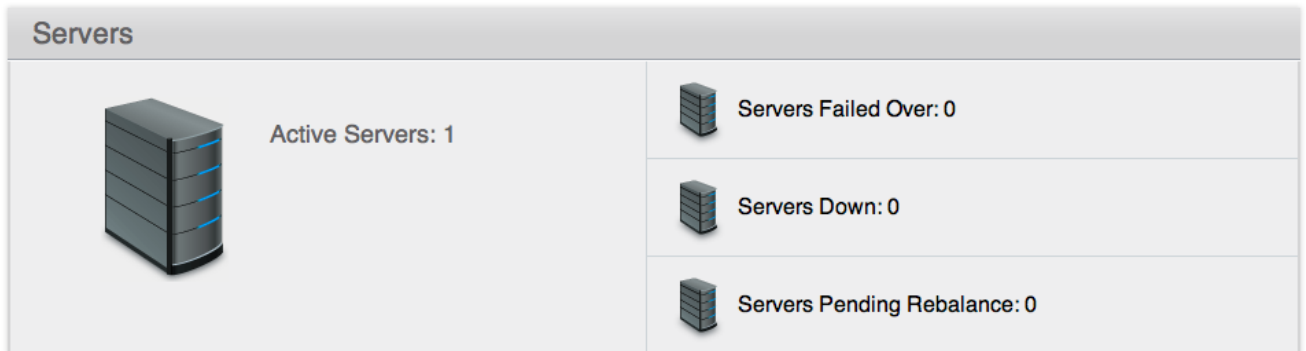
- Active Servers is the number of active servers within the current clsuter configuration.

- 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 joiining a cluster or being reactivated after failover.

6.2. Viewing Server Nodes

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.

Figure 6.5. Web Console — Server Nodes
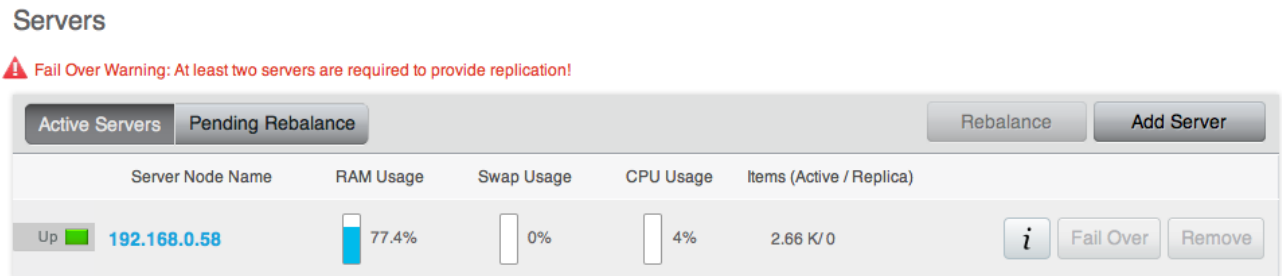
Clicking the triangle next to a server displays server node specific information, including the IP address, OS, Couchbase version and Memory and Disk allocation information.

Figure 6.6. Web Console — Server Node Detail
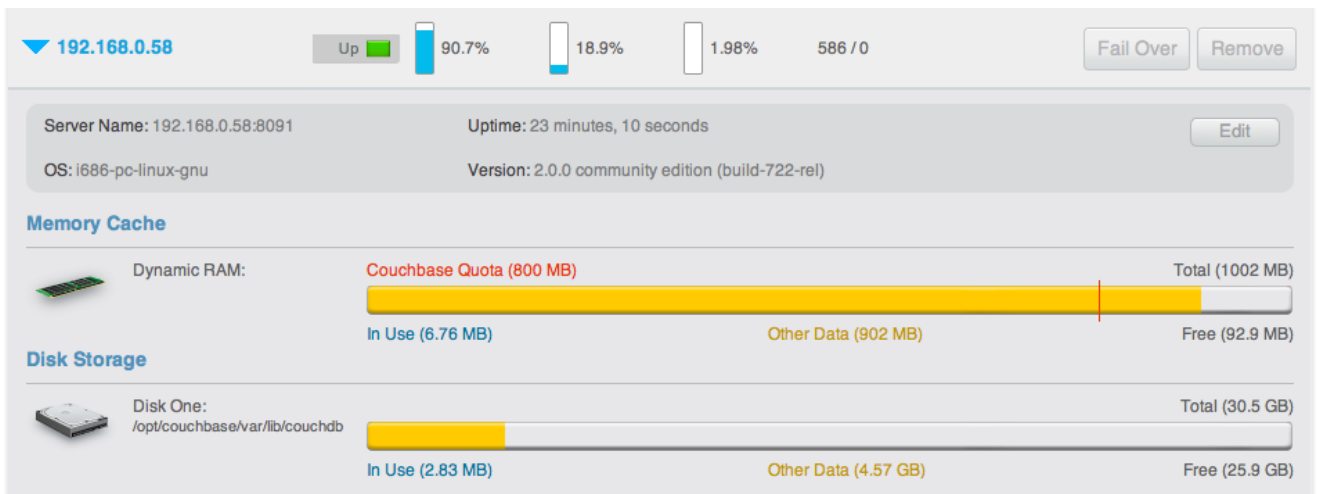
The detail display shows the following information:

- **Node Information**

  The node information provides detail node configuration data:

  - 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.

Figure 6.7. Web Console — Data Bucket/Server view

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.

Figure 6.8. Web Console — Server specific view
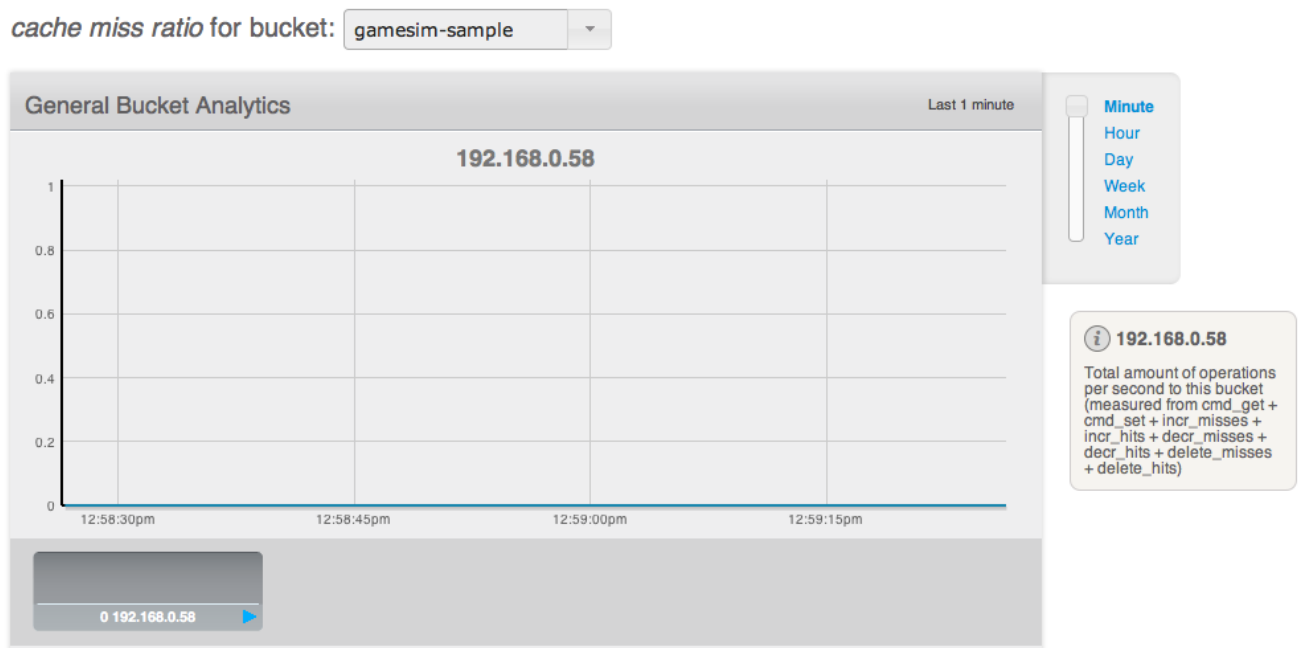
For more information on the data bucket statistics, see Section 6.3, "Viewing Data Buckets".

6.2.1. 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.

Figure 6.9. Web Console — Down Status

- **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.

Figure 6.10. Web Console — Pend Status

You can monitor the current server status using both the Manage: Server Nodes and Monitor: Server Nodes screens within the Web Console.

6.3. Viewing 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 cluser 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.

Figure 6.11. Web Console — Data Buckets Overview

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. For more information, see Section 6.4.1, "Individual Bucket Monitoring".

- 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. For more information, see Section 6.3.2, "Bucket Information".

- Clicking the `Documents` button will take you to a list of objects identified as parseable documents. See Section 6.6, "Using the Document Editor" for more information.

- The `Views` button allows you to create and manage views on your stored objects. For more information, see Section 6.5, "Using the Views Editor".

To create a new data bucket, click the `Create New Data Bucket`. See Section 6.3.1, "Creating and Editing Data Buckets" for details on creating new data buckets.

6.3.1. 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.

6.3.1.1. Creating a New Bucket

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:

Figure 6.12. Creating a Data Buckets

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.

Best Practice: Default Bucket Should Only for Testing
Any default bucket you initially set up with Couchbase Server should not be used for storing live application data; you should create a named bucket specifically for your application. The default bucket you create when you first install Couchbase Server should only be used for testing.

3. Select a Bucket Type, either `Memcached` or `Couchbase`. See Section 1.2.3, "Data Storage" for more information. The options that appear in this panel will differ based on your a bucket type you select.

   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. For more information, see Section 5.6, "Failing Over Nodes".

    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**

    As of Couchbase Server 2.1, we support multiple readers and writers 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. For general information about disk storage, see Section 1.2.7, "Disk Storage". For information on multi- readers and writers, see Section 5.1, "Using Multi- Readers and Writers".

- **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.

    For more information, see Section 4.2.1, "RAM Sizing".

> **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. For more information on database and view compaction, see Section 5.5, "Database and View Compaction".

You can opt to override the default auto compaction settings for this individual bucket. Default settings are configured through the Settings menu. For more information on setting the default autocompaction parameters, see Section 6.8.4, "Enabling Auto-Compaction". 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.

- **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, click the Enable checkbox.

4. Click Create.

  Creates the new bucket with bucket configuration.

6.3.1.2. 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.

6.3.1.3. 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.

6.3.2. 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.

Figure 6.13. Web Console — Bucket Information

You can edit the bucket information by clicking the `Edit` button within the bucket information display.

6.4. Viewing Bucket and Cluster Statistics

Within the Data Bucket monitor display, information is shown by default 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 agregated 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. You can select an individual node, which displays the Section 6.2, "Viewing Server Nodes" for that node. Selecting All Server Nodes shows the Section 6.3, "Viewing Data Buckets" page.

- 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.

  Note

  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 individually for each server within the cluster, instead of aggregating the information for the entire cluster.

6.4.1. 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. For more information, see Section 6.4.1.1, "Bucket Monitoring — Summary Statistics".

- **vBucket Resources**

  This section provides detailed information on the vBucket resources across the cluster, including the active, replica and pending operations. For more information, see Section 6.4.1.2, "Monitoring vBucket Resources".

- **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. For more information, see Section 6.4.1.3, "Monitoring Disk Queues".

- **TAP Queues**

  The TAP queues section provides information on the activity within the TAP queues across replication, rebalancing and client activity. For more information, see Section 6.4.1.4, "Monitoring TAP Queues".

- **XDCR Destination**

  The XDCR Destination section show you statistical information about the Cross Datacenter Replication (XDCR), if XDCR has been configured. For more information on XDCR, see Section 5.9, "Cross Datacenter Replication (XDCR)". For more information on the available statistics, see Section 6.4.1.6, "Monitoring Outgoing XDCR".

- **View Stats**

  The View Stats section allows you to monitor the statistics for each production view configured within the bucket or system. For more information on the available statistics, see Section 6.4.1.8, "Monitoring View Statistics".

- **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. See Section 6.4.1.5, "Bucket Memcached Buckets".

6.4.1.1. 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.

Figure 6.14. Web Console — Summary Statistics

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.

6.4.1.2. 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.

Figure 6.15. Web Console — vBucket Resources statistics

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.

6.4.1.3. 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.

Figure 6.16. Web Console — Disk Queue Statistics
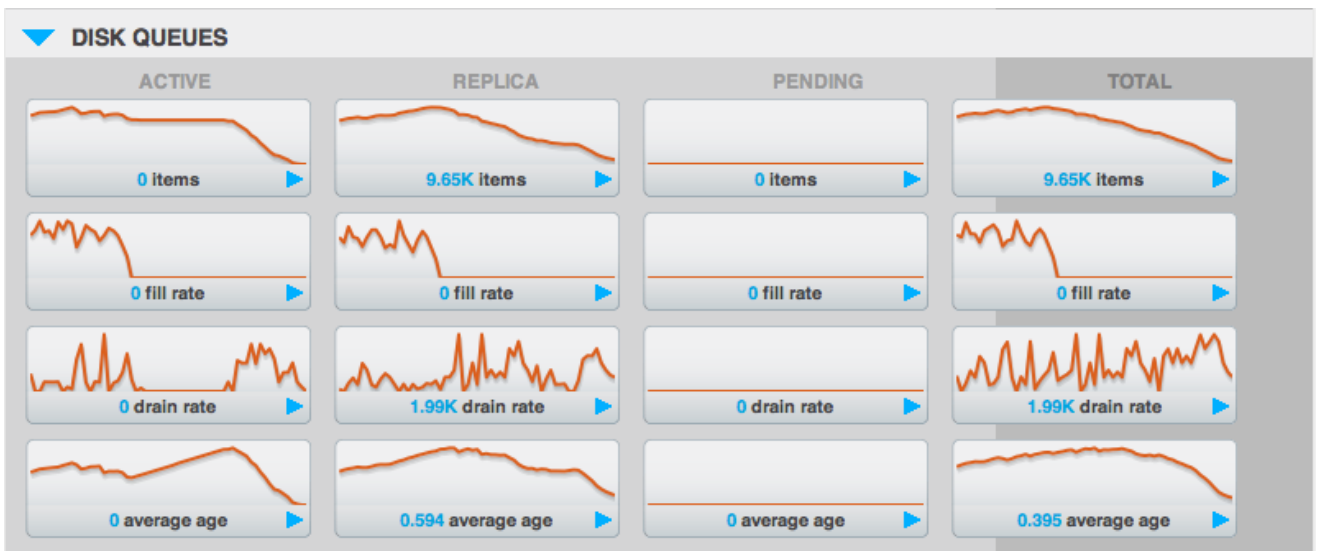
The Active column displays the information for the Disk Queues within the Active state. The Replica column displays the statistics for the Disk Queues within the Replica state (i.e. currently being replicated). The Pending columns shows statistics for the disk Queues in the Pending state, i.e. while 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.

6.4.1.4. 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.

Figure 6.17. Web Console — TAP Queue Statistics

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.

6.4.1.5. Bucket Memcached Buckets

For Memcached buckets, Web Console displays a separate group of statistics:

Figure 6.18. Web Console — Memcached Statistics
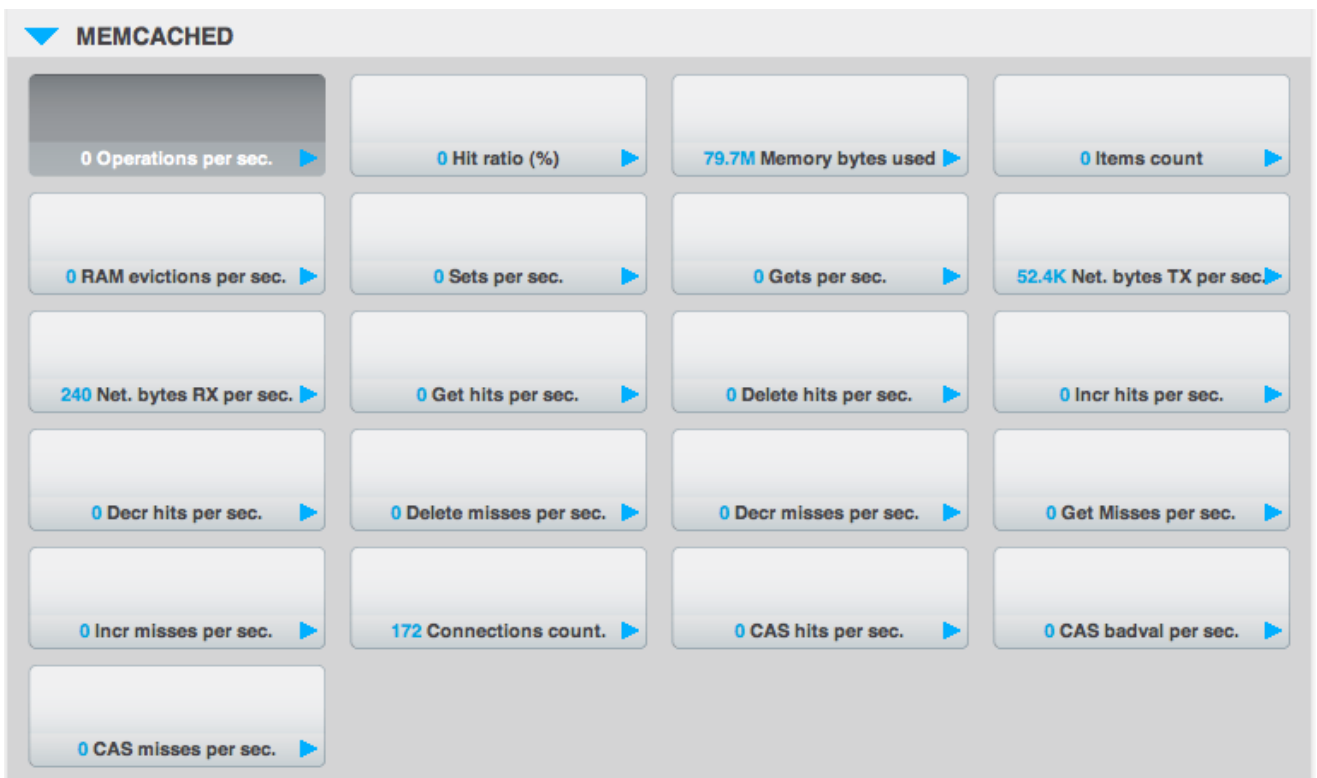
The Memcached statistics are:

- 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

6.4.1.6. Monitoring Outgoing XDCR

The Outgoing XDCR shows the XDCR operations that are supporting cross datacenter replication from the current cluster to a destination cluster. For more information on XDCR, see Section 5.9, "Cross Datacenter Replication (XDCR)".

You can monitor the current status for all active replications in the Ongoing Replications section under the XDCR tab:

Figure 6.19. Couchbase Web Console - Ongoing Replications

The Ongoing Replications section shows the following information:

Column	Description
Bucket	The source bucket on the current cluster that is being replicated.
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 originatin
  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 ongo-
ing replication for the particular bucket. Under the `Outgoing XDCR` panel if you have multiple replication streams you
will see statistics for each stream.

Figure 6.20. Web Console — Outgoing XDCR Statistics
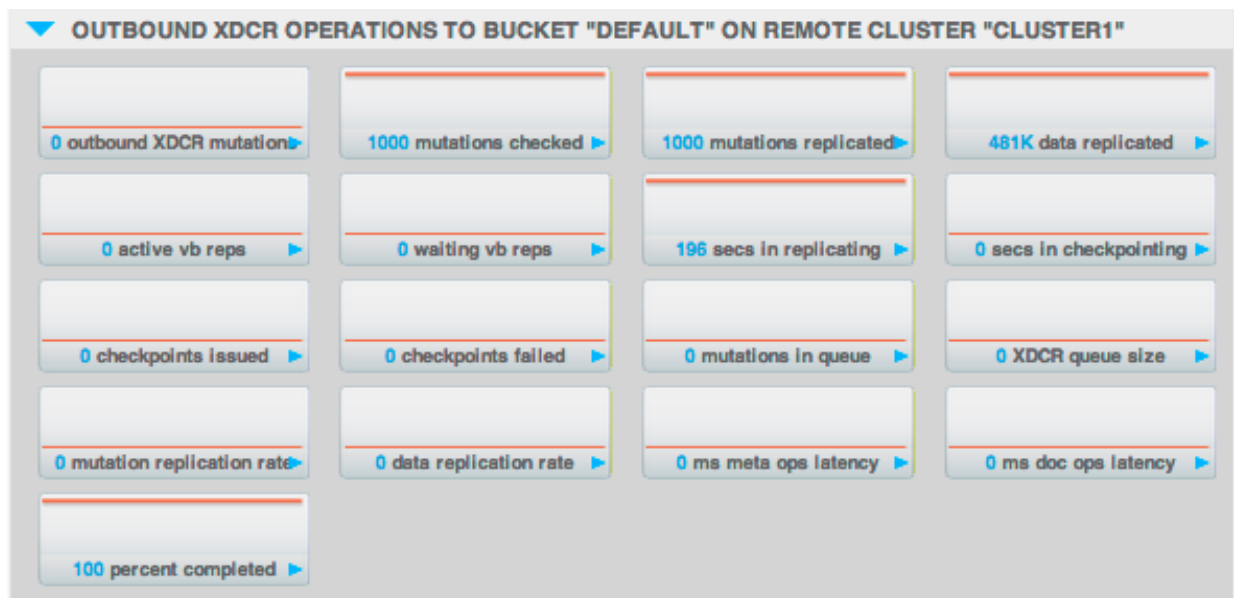
The statistics shown are:

- 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.

- 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. For more information including a full list of available statistics, see Section 8.9.8, "Getting XDCR Stats via REST".

6.4.1.7. Monitoring Incoming XDCR

The Incoming XDCR section shows the XDCR operations that are coming into to the current cluster from a remote cluster. For more information on XDCR, see Section 5.9, "Cross Datacenter Replication (XDCR)".

Figure 6.21. Web Console — Incoming XDCR Statistics
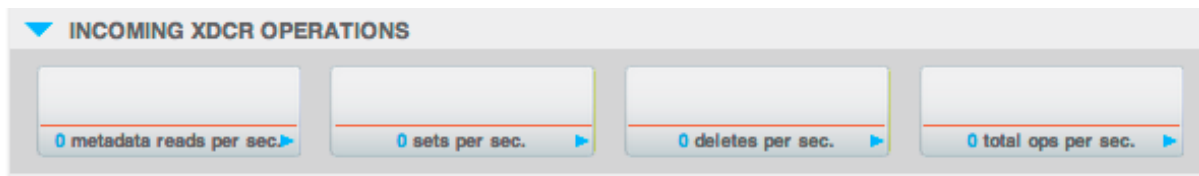
The statistics shown are:

* metadata reads per sec.

  Number of documents XDCR scans for metadata per second. XDCR uses this information for conflict resolution. See, Section 5.9.8, "Behavior and Limitations".

* sets per sec.

  Set operations per second for incoming XDRC 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.

6.4.1.8. 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. For more information on Views, see Chapter 9, *Views and Indexes*.

Figure 6.22. Web Console — Views Statistics
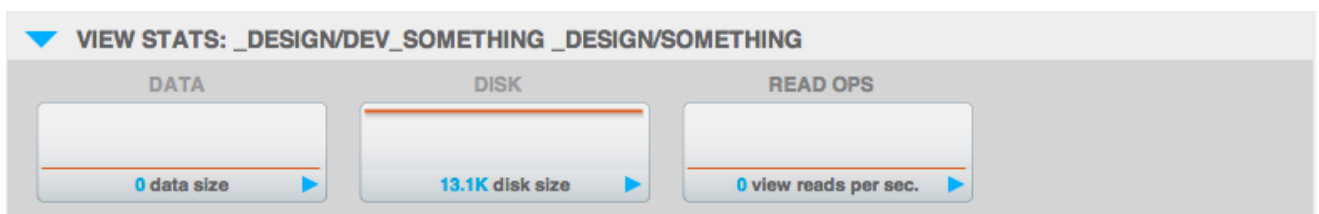
The statistics shown are:

- 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.

6.5. Using the Views Editor

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.

Figure 6.23. Web Console — View Manager
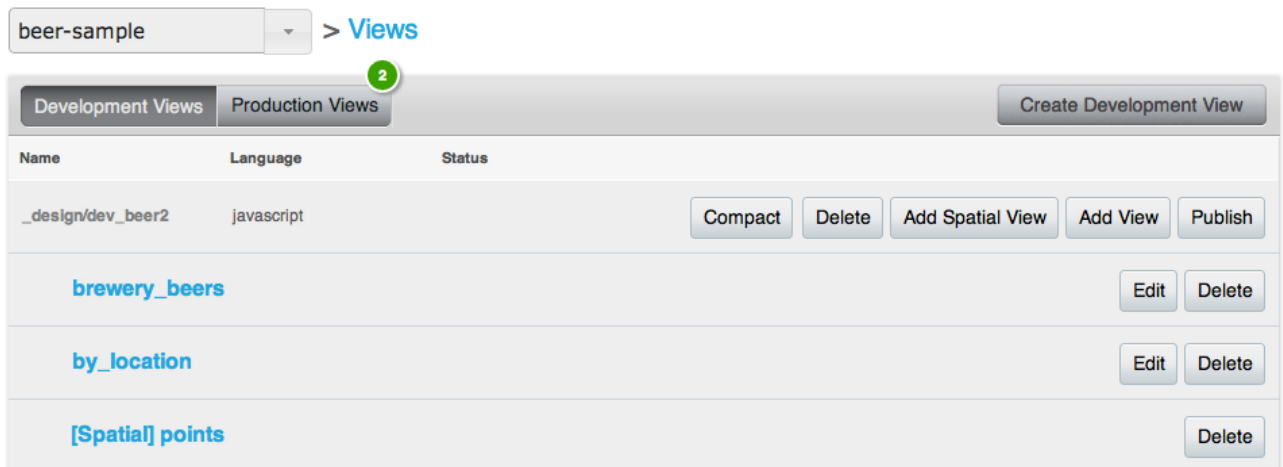
The individual elements of this interface are:

- 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. See Section 6.5.1, "Creating and Editing Views".

- You can switch between `Production Views` and `Development Views`. See Section 9.4, "Development and Production Views" for more information.

- 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 will delete all of the views defined within the design document.

- `Add Spatial View` creates a new spatial view within the corresponding design document. See Section 6.5.1, "Creating and Editing Views".

- `Add View` creates a new view within the corresponding design document. See Section 6.5.1, "Creating and Editing Views".

- `Publish` your design document (and all of the defined views) as a production design document. See Section 6.5.2, "Publishing Views".

- For each individual view listed:

  - `Edit`, or clicking the view name

    Opens the view editor for the current view name, see Section 6.5.1, "Creating and Editing Views".

  - `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 Section 6.5.3, "Getting View Results" for more information.

6.5.1. 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 or edit your documents using the REST API, see Section 9.7, "Design Document REST API".

To create a new view as part of an existing design document, click the `Add View` button against the corresponding design document.

> **Note**
>
> 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.

Figure 6.24. Web Console — View Editing

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` will take you to the Views editor, see Section 6.6, "Using the Document Editor"

> **Note**
>
> 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, see Section 9.5, "Writing Views".

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. For more information on querying and selecting information from a view, see Section 9.8, "Querying Views".

descending	☐
startkey	
endkey	
startkey_docid	
endkey_docid	
group	☐
group_level	
include_docs	☐
inclusive_end	☐
key	
keys	
reduce	true / false / none
update_seq	☐
stale	false / update_after / ok
connection_timeout	60000

Reset   Close

Clicking on the Filter Results query string will open a new window containing the raw, JSON formatted, version of the View results. To access the view results using the REST API, see Section 9.8.1, "Querying Using the REST API".

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.

Note

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.

For more information on the filter options, see Section 6.5.3, "Getting View Results".

6.5.2. 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.

6.5.3. 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.

Figure 6.26. Web Console — View Detail

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. For more information, see Section 9.8, "Querying Views".

Note

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. For more information on this character and usage, see Section 9.8.2.2, "Partial Selection and Key Ranges".

6.6. Using the Document Editor

The Document Viewer and Editor enables you to browser, 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. This will open a list of available documents. You are shown only a selection of the available documents, rather than all documents.

Figure 6.27. Web Console — Document Overview

You can select a different Bucket by using the bucket selection popup on the left. You can also 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 will open a prompt to specify the document Id of the created document.

Figure 6.28. Web Console — Document Create

Once the document Id has been set, you will be presented with the document editor. The document editor will also be opened when you click on the document ID within the document list. To edit the contents of the document, use the textbox to modify the JSON of the stored document.

Figure 6.29. Web Console — Document Edit

Within the document editor, you can click `Delete` to delete the current document, `Save As...` will copy the currently displayed information and create a new document with the document Id you specify. The `Save` will save the current document and return you to the list of documents.

6.7. Log

The Log section of the website allows you to view the built-in event log for Couchbase Server so that you can identify activity and errors within your Couchbase cluster.

Figure 6.30. Web Console — Log Viewer

6.8. Settings

The Settings interface sets the global settings for your Couchbase Server instance.

6.8.1. Update Notification Settings

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, you must click Save to record the change.

If update notifications are disabled then the Update Notifications screen will only notify you of your currently installed version, and no alert will be provided.

Figure 6.31. Web Console — Settings — Update Notifications
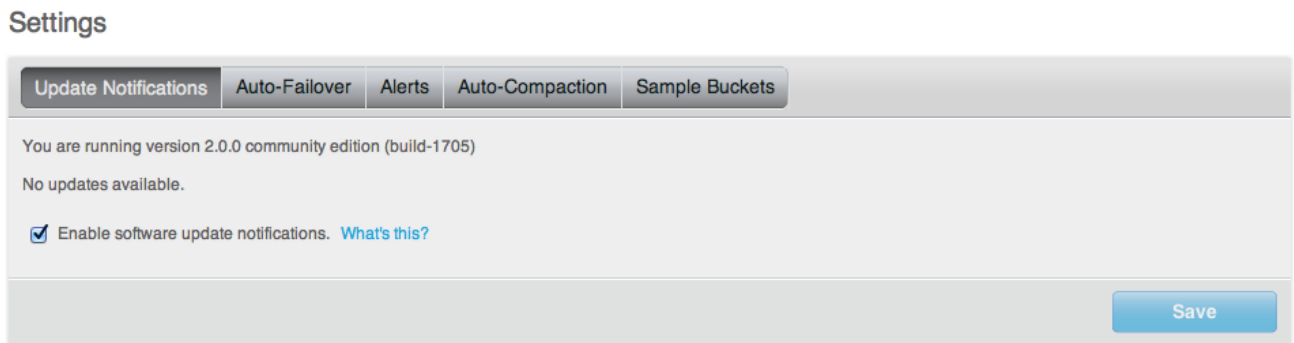
For more information on how Update Notifications work, see Section 6.9, "Updating Notifications".

6.8.2. Enabling Auto-Failover Settings

The Auto-Failover settings enable auto-failover, and 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 it the Timeout box. The default timeout is 30 seconds.

Figure 6.32. Web Console — Settings — Auto-Failover
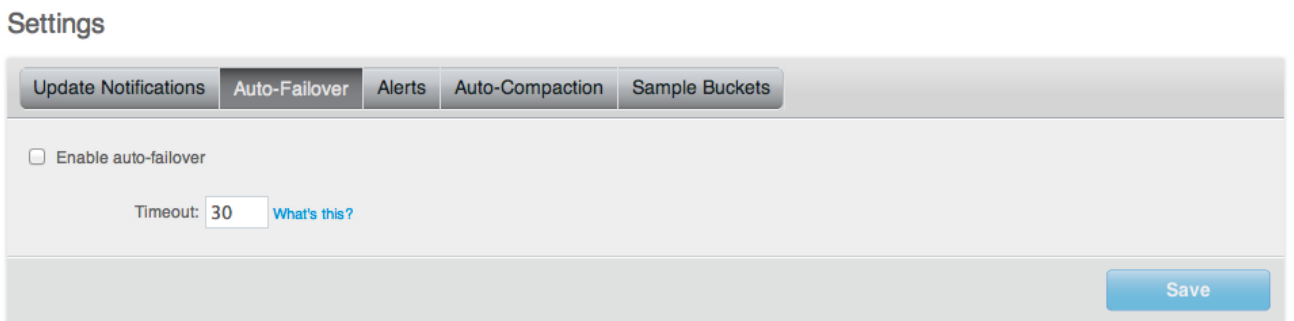
For more information on Auto-Failover, see Section 5.6.2, "Using Automatic Failover".

6.8.3. Enabling Alerts

You can enable email alerts to be raised when a signficant 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.

The available settings are:

• Enable email alerts

  If checked, email alerts will be raised on the specific error enabled within the Available Alerts section of the configuration.

• 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.

- 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 TLS

Enable Transport Layer Security (TLS) when sending the email through the designated server.

- 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. You can specify more than one recipient by separating each address by a space, comma or semicolon.

Clicking the `Test Mail` button will 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:

- Node was auto-failovered

The sending node has been auto-failovered.

- Maximum number of auto-failovered nodes was reached

The auto-failover system will stop 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.

- Writing data to disk for a specific bucket has failed

The disk or device used for persisting data has failed to store persitent data for a bucket.

Figure 6.33. Web Console — Settings — Alerts

For more information on Auto-Failover, see Section 5.6.2, "Using Automatic Failover".

6.8.4. Enabling Auto-Compaction

The Auto-Compaction tab configures the default auto-compaction settings for all the databases. These can be overridden using per-bucket settings available within Section 6.3.1, "Creating and Editing Data Buckets".

Figure 6.34. Web Console — Settings — Auto-Compaction

The settings tab sets the following default parameters:

- 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 compaction 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.

- 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.

Best Practice: Enable Parallel Compaction

> It is recommended to run data and view compaction in parallel based on the throughput of your disk.

For more information on compaction, see Section 5.5, "Database and View Compaction". For information on how auto-compaction operates, see Section 5.5.2, "Auto-Compaction Configuration".

6.8.5. Installing Sample Buckets

The Sample Buckets tab enables you to install the sample bucket data if the data has not already been loaded in the system. For more information on the sample data available, see Appendix B, *Couchbase Sample Buckets*.

Figure 6.35. Web Console — Settings — Sample Buckets
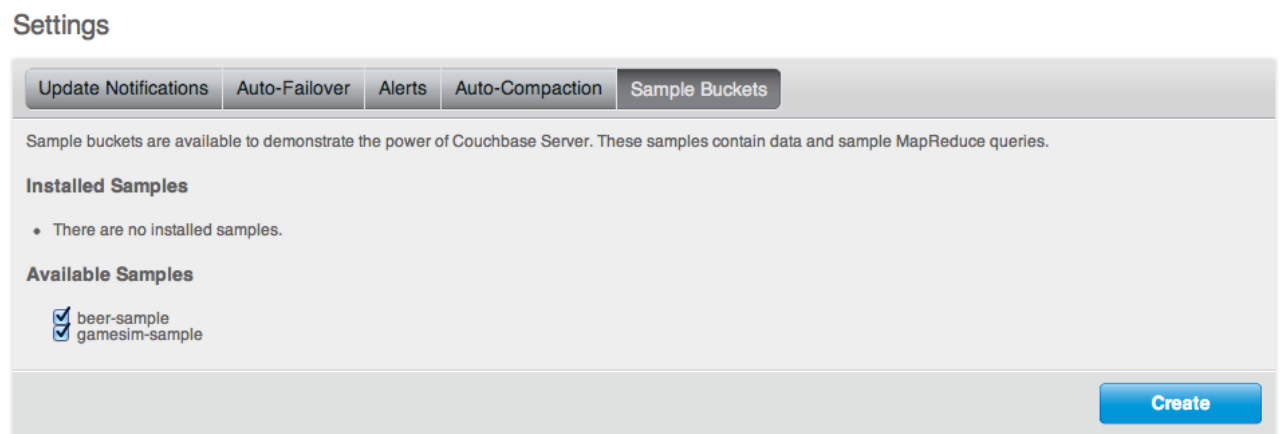
If the sample bucket data was not loaded during setup, select the sample buckets that you want to load using the checkboxes, and click the `Create` button.

If the sample bucket data has already been loaded, it will be listed under the Installed Samples section of the page.

6.9. Updating Notifications

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 you select the Update Notifications option, the Web Console will communicate 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 will be provided with notification of the new version and information on where you can download the new version.

- Basic information about the size and configuration of your Couchbase cluster. This information will be used to help us prioritize our development efforts.

You can enable/disable software updaate notifications

> **Note**
>
> The process occurs within the browser accessing the web console, not within the server itself, and no further configuration or internet access is required on the server to enable this functionality. Pro-

viding 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 us improve the future development process for Couchbase Server and related products.

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 will not work.

Notifications

If an update notification is available, the counter within the button display within the Couchbase Console will be displayed with the number of available updates.

Viewing Available Updates

To view the available updates, click on the Settings link. This displays your current version and update availability. From here you can be taken to the download location to obtain the updated release package.

6.10. Warnings and Alerts

A new alerting systems has been built into the Couchbase Web Console. This is sued 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. A sample of the IP address popup is shown below:

Figure 6.36. Web Console — Warning Notification
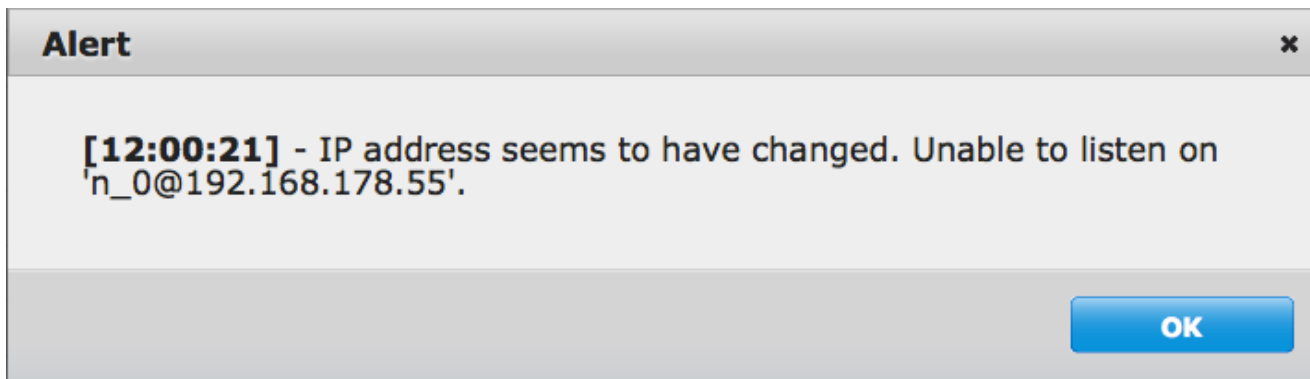
The following errors and alerts are supported:

• **IP Address Changes**

If the IP address of a Couchbase Server in your cluster changes, you will be warned that the address is no longer available. You should check the IP address on the server, and update your clients or server configuration.

• **OOM (Hard)**

Indicates if the bucket memory on a node is entirely used for metadata.

• **Commit Failure**

Indicates that writing data to disk for a specific bucket has failed.

- **Metadata Overhead**

  Indicates that a bucket is now using more than 50% of the allocated RAM for storing metadata and keys, reducing the amount of RAM available for data values.

- **Disk Usage**

  Indicates that the available disk space used for persistent storage has reached at least 90% of capacity.

Chapter 7. Command-line Interface for Administration

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 Chapter 8, *Using the 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.

7.1. 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:

Table 7.1. Administration — Command-line Tools and Availability

Tool	Server Versions	Description/Status
couchbase-cli	2.0+	The main tool for communicating and managing your Couchbase Server.
cbstats	2.0+	A tool for checking a Couchbase Server node for its statistics.
cbepctl	2.0+	A tool for controlling the vBucket states on a Couchbase Server node. Also responsible for controlling the configuration, memory and disk persistence behavior. Formerly provided as the separate tools, **cbvbucketctl** and **cbflushctl** in Couchbase 1.8.
cbcollect_info	2.0+	A support tool for gathering statistics from a Couchbase Server node and used by Couchbase Technical Support.
cbbackup	2.0+	Creates a copy of all the data for a cluster, bucket, node, or combination and stores it in a file on disk.
cbrestore	2.0+	Reads in data from a backup file and reload data for a cluster, bucket, node, or combination into RAM.
cbtransfer	2.0+	This tool is the underlying, generic data transfer tool that **cbbackup** and **cbrestore** are built upon.
cbhealthchecker	2.1+	Generate a health report for your Couchbase cluster with `cbhealthchecker`.
cbdocloader	2.0+	Used for loading sample data sets. Can be used to load a directory of JSON files into Couchbase Server.
cbworkloadgen	2.0+	Generates test workload for cluster.
cbanalyze-core	2.0+	Helper script to parse and analyze core dump from a Couchbase node.
vbuckettool	2.0+	Returns vBucket and node where a key should be for a Couchbase bucket. These two values based on Couchbase Server internal hashing algorithm. Moved as of 1.8 to /bin/tools directory.
cbflushctl	1.8	Replaced by **cbepctl** in 2.0.
cbvbucketctl	1.8	Replaced by **cbepctl** in 2.0.

By default, the command-line tools are installed into the following locations on each platform:

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\couch- base\server\bin\tools`.
Mac OS X	`/Applications/Couchbase Server.app/Contents/Re- sources/couchbase-core/bin`

7.2. 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**

7.3. 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:

Table 7.2. Administration — Deprecated/Removed Command-line Tools

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
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-in- cremental	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**

Tool	Server Versions	Description/Status
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**

7.4. couchbase-cli Tool

You can find this tool in the following locations, depending upon your 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 you use this tool to create a data bucket, it will create a bucket that all nodes in the cluster have access to.

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. You can use either:

```
--cluster=HOST[:PORT]
```

Or the shortened form:

```
-c HOST[:PORT]
```

- `OPTIONS` are zero or more options as follows:

`-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

Table 7.3. Administration — couchbase-cli Commands

Command	Description
`server-list`	List all servers in a cluster

Command	Description
server-info	Show details on one server
server-add	Add one or more servers to the cluster
server-readd	Readd a server that was failed over
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

The following are options which can be used with their respective commands. Administration — **couchbase-cli** Tool commands options:

Table 7.4. Couchbase CLI Command Options

Command	Option	Description
server-add	--server-add=HOST[:PORT]	Server to add to cluster
server-add	--server-add-username=USERNAME	Admin username for the server to be added
server-add	--server-add-password=PASSWORD	Admin password for the server to be added
server-readd	--server-add=HOST[:PORT]	Server to re-add to cluster
server-readd	--server-add-username=USERNAME	Admin username for the server to be added

167

Command	Option	Description
`server-readd`	`--server-add-password=PASSWORD`	Admin password for the server to be added
`rebalance`	`--server-add*`	See server-add OPTIONS
`rebalance`	`--server-remove=HOST[:PORT]`	The server to remove from cluster
`failover`	`--server-failover=HOST[:PORT]`	Server to failover
`cluster-*`	`--cluster-username=USER`	New admin username
`cluster-*`	`--cluster-password=PASSWORD`	New admin password
`cluster-*`	`--cluster-port=PORT`	New cluster REST/http port
`cluster-*`	`--cluster-ramsize=RAMSIZEMB`	Per node RAM quota in MB
`node-init`	`--node-init-data-path=PATH`	Per node path to store data
`node-init`	`--node-init-index-path=PATH`	Per node path to store index
`bucket-*`	`--bucket=BUCKETNAME`	Named bucket to act on
`bucket-*`	`--bucket-type=TYPE`	Bucket type, either memcached or couchbase
`bucket-*`	`--bucket-port=PORT`	Supports ASCII protocol and does not require authentication
`bucket-*`	`--bucket-password=PASSWORD`	Standard port, exclusive with bucket-port
`bucket-*`	`--bucket-ramsize=RAMSIZEMB`	Bucket RAM quota in MB
`bucket-*`	`--bucket-replica=COUNT`	Replication count
`bucket-*`	`--enable-flush=[0|1]`	Enable/disable flush
`bucket-*`	`--enable-index-replica=[0|1]`	Enable/disable index replicas
`bucket-*`	`--wait`	Wait for bucket create to be complete before returning
`bucket-*`	`--force`	Force command execution without asking for confirmation
`bucket-*`	`--data-only`	Compact database data only
`bucket-*`	`--view-only`	Compact view data only

Command	Option	Description
`setting-compacttion`	`--compaction-db-percentage=PERCENTAGE`	Percentage of disk fragmentation when database compaction is triggered
`setting-compacttion`	`--compaction-db-size=SIZE[MB]`	Size of disk fragmentation when database compaction is triggered
`setting-compacttion`	`--compaction-view-percentage=PERCENTAGE`	Percentage of disk fragmentation when views compaction is triggered
`setting-compacttion`	`--compaction-view-size=SIZE[MB]`	Size of disk fragmentation when views compaction is triggered
`setting-compacttion`	`--compaction-period-from=HH:MM`	Enable compaction from this time onwards
`setting-compacttion`	`--compaction-period-to=HH:MM`	Stop enabling compaction at this time
`setting-compacttion`	`--enable-compaction-abort=[0|1]`	Allow compaction to abort when time expires
`setting-compacttion`	`--enable-compaction-parallel=[0|1]`	Allow parallel compaction processes for database and view
`setting-notification`	`--enable-notification=[0|1]`	Allow notifications
`setting-alert`	`--enable-email-alert=[0|1]`	Allow email alert
`setting-alert`	`--email-recipients=RECIPIENT`	Email recipents, separate addresses with , or ;
`setting-alert`	`--email-sender=SENDER`	Sender email address
`setting-alert`	`--email-user=USER`	Email server username
`setting-alert`	`--email-password=PWD`	Email server password
`setting-alert`	`--email-host=HOST`	Email server hostname
`setting-alert`	`--email-port=PORT`	Email server port
`setting-alert`	`--enable-email-encrypt=[0|1]`	Email encryption with 0 the default for no encryption
`setting-alert`	`--alert-auto-failover-node`	Node was failed over via autofailover
`setting-alert`	`--alert-auto-failover-max-reached`	Maximum number of auto failover nodes reached
`setting-alert`	`--alert-auto-failover-node-down`	Node not auto failed-over as other nodes are down at the same time
`setting-alert`	`--alert-auto-failover-cluster-small`	Node not auto failed-over as cluster was too small
`setting-alert`	`--alert-ip-changed`	Node ip address changed unexpectedly
`setting-alert`	`--alert-disk-space`	Disk space used for persistent storage has reached at least 90% capacity

Command	Option	Description
`setting-alert`	`--alert-meta-overhead`	Metadata overhead is more than 50% of RAM for node
`setting-alert`	`--alert-meta-oom`	Bucket memory on a node is entirely used for metadata
`setting-alert`	`--alert-write-failed`	Writing data to disk for a specific bucket has failed
`setting-autofailover`	`--enable-au-to-failover=[0|1]`	Allow auto failover
`setting-autofailover`	`--auto-failover-timeout=TIMEOUT (>=30)`	Specify amount of node timeout that triggers auto failover
`setting-xdcr`	`--max-concur-rent-reps=[32]`	Maximum concurrent replicators per bucket, 8 to 256.
`setting-xdcr`	`--checkpoint-inter-val=[1800]`	Intervals between checkpoints, 60 to 14400 seconds.
`setting-xdcr`	`--worker-batch-size=[500]`	Doc batch size, 500 to 10000.
`setting-xdcr`	`--doc-batch-size=[2048]KB`	Document batching size, 10 to 100000 KB
`setting-xdcr`	`--failure-restart-in-terval=[30]`	Interval for restarting failed xdcr, 1 to 300 seconds
`setting-xdcr`	`--optimistic-replica-tion-threshold=[256]`	Document body size threshold (bytes) to trigger optimistic replication
`xdcr-setup`	`--create`	Create a new xdcr configuration
`xdcr-setup`	`--edit`	Modify existed xdcr configuration
`xdcr-setup`	`--delete`	Delete existing xdcr configuration
`xdcr-setup`	`--xdcr-clus-ter-name=CLUSTERNAME`	Remote cluster name
`xdcr-setup`	`--xd-cr-hostname=HOSTNAME`	Remote host name to connect to
`xdcr-setup`	`--xd-cr-username=USERNAME`	Remote cluster admin username
`xdcr-setup`	`--xd-cr-password=PASSWORD`	Remote cluster admin password
`xdcr-replicate`	`--create`	Create and start a new replication
`xdcr-replicate`	`--delete`	Stop and cancel a replication
`xdcr-replicate`	`--xdcr-from-bucket=BUCKET`	Source bucket name to replicate from

Command	Option	Description
xdcr-replicate	--xdcr-cluc-ter-name=CLUSTERNAME	Remote cluster to replicate to
xdcr-replicate	--xdcr-to-bucket=BUCKETNAME	Remote bucket to replicate to

You can also perform many of these same settings using the REST-API, see Chapter 8, *Using the REST API*.

Some examples of commonly-used **couchbase-cli** commands:

```
Set data path for an unprovisioned cluster:
  couchbse-cli node-init -c 192.168.0.1:8091 \
      --node-init-data-path=/tmp/data \
      --node-init-index-path=/tmp/index

 List servers in a cluster:
   couchbase-cli server-list -c 192.168.0.1:8091

 Server information:
   couchbase-cli server-info -c 192.168.0.1:8091

 Add a node to a cluster, but do 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

 Add a node to a cluster and rebalance:
   couchbase-cli rebalance -c 192.168.0.1:8091 \
      --server-add=192.168.0.2:8091 \
      --server-add-username=Administrator \
      --server-add-password=password

 Remove a node from a cluster and rebalance:
   couchbase-cli rebalance -c 192.168.0.1:8091 \
      --server-remove=192.168.0.2:8091

 Remove and add nodes from/to a cluster and rebalance:
   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

 Stop the current rebalancing:
   couchbase-cli rebalance-stop -c 192.168.0.1:8091

 Set 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

 change 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

 Change the data path:
    couchbase-cli node-init -c 192.168.0.1:8091 \
      --node-init-data-path=/tmp

 List buckets in a cluster:
   couchbase-cli bucket-list -c 192.168.0.1:8091

 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

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

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

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 \
    --enable-index-replica=1

Delete a bucket:
  couchbase-cli bucket-delete -c 192.168.0.1:8091 \
    --bucket=test_bucket

Flush a bucket:
  couchbase-cli bucket-flush -c 192.168.0.1:8091 \
    --force

Compact a bucket for both data and view:
  couchbase-cli bucket-compact -c 192.168.0.1:8091 \
      --bucket=test_bucket

Compact a bucket for data only:
  couchbase-cli bucket-compact -c 192.168.0.1:8091 \
      --bucket=test_bucket \
      --data-only

Compact a bucket for view only:
  couchbase-cli bucket-compact -c 192.168.0.1:8091 \
      --bucket=test_bucket \
      --view-only

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

Delete a XDCR remote cluster:
  couchbase-cli xdcr-delete -c 192.168.0.1:8091 \
      --xdcr-cluster-name=test

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

Delete a replication stream:
  couchbase-cli xdcr-replicate -c 192.168.0.1:8091 \
      --delete \
      --xdcr-replicator=f4eb540d74c43fd3ac6d4b7910c8c92f/default/default
```

7.4.1. Flushing Buckets with couchbase-cli

Enabling Flush of Buckets:

When you want to flush a data bucket you must first enable this option then actually issue the command to flush the data bucket. *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.* The other option available to enable data bucket flush is to use the Couchbase Web Console, see Section 6.3.1, "Creating and Editing Data Buckets". You can enable this option when you actually create the data bucket, or when you edit the bucket properties:

```
> couchbase-cli bucket-create [bucket_name] [cluster_admin:pass] --enable-flush=[0|1]  // 0 the default and 1 to enabl
 > couchbase-cli bucket-edit [bucket_name] [cluster_admin:pass] --enable-flush=[0|1]  // 0 the default and 1 to enable
```

After you enable this option, you can then flush the data bucket.

Flushing a Bucket:

After you explicitly enable data bucket flush, you can then flush 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. See also Section 6.3.1, "Creating and Editing Data Buckets".

```
> couchbase-cli bucket-flush [cluster_admin:pass] [bucket_name OPTIONS]
```

By default this command will confirm 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.

7.5. 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.

You 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 bucket1 -p password all | egrep 'item|mem|flusher|ep_queue|bg|eje|resi|warm'"
```

The following table provides a list of results that will be returned when you perform a `cbstats all` command:

Stat	Description
ep_version	Version number of ep_engine.
ep_storage_age	Seconds since most recently stored object was initially queued.
ep_storage_age_highwat	ep_storage_age high water mark
ep_min_data_age	Minimum data age setting.
ep_startup_time	System-generated engine startup time
ep_queue_age_cap	Queue age cap setting.
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	Deprecated in 2.1.0. Number of times an object was not stored due to being too young.
ep_too_old	Deprecated in 2.1.0. 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 remaining to be 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
ep_commit_time_total	Cumulative milliseconds spent committing.

Stat	Description
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
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 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 queues, replication queues, checkpoints, etc.
ep_max_data_size	Max amount of data allowed in memory.
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 background fetch jobs.
ep_tap_bg_fetched	Number of tap disk fetches
ep_tap_bg_fetch_requeued	Number of times a tap background fetch task is re-queued.
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.

Stat	Description
ep_num_value_ejects	Number of times item values got 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 keep-alive time.
ep_dbname	DB path.
ep_dbinit	Number of seconds to initialize DB.
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 is 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_store_max_concurrency	Maximum allowed concurrency at the storage layer.
ep_store_max_readers	Maximum number of concurrent read-only storage threads.
ep_store_max_readwrite	Maximum number of concurrent read/write storage threads.
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_mlog_compactor_runs	Number of times mutation log compactor is executed
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

Stat	Description
ep_vb_snapshot_total	Total VB state snapshots persisted in disk
vb_active_num	Number of active vBuckets
vb_active_curr_items	Number of in memory items
vb_active_num_non_resident	Number of non-resident items
vb_active_perc_mem_resident	% memory resident
vb_active_eject	Number of times item values got ejected
vb_active_expired	Number of times an item was expired
vb_active_ht_memory	Memory overhead of the hashtable
vb_active_itm_memory	Total item memory
vb_active_meta_data_memory	Total metadata memory
vb_active_ops_create	Number of create operations
vb_active_ops_update	Number of update operations
vb_active_ops_delete	Number of delete operations
vb_active_ops_reject	Number of rejected operations
vb_active_queue_size	Active items in disk queue
vb_active_queue_memory	Memory used for disk queue
vb_active_queue_age	Sum of disk queue item age in milliseconds
vb_active_queue_pending	Total bytes of pending writes
vb_active_queue_fill	Total enqueued items
vb_active_queue_drain	Total drained items
vb_active_num_ref_items	Number of referenced items
vb_active_num_ref_ejects	Number of times referenced item values got ejected
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	% of memory used for resident items
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 in 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
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

Stat	Description
vb_pending_queue_pending	Total bytes of pending writes
vb_pending_queue_fill	Total enqueued items
vb_pending_queue_drain	Total drained items
vb_pending_num_ref_items	Number of referenced items
vb_pending_num_ref_ejects	Number of times referenced item values got ejected

7.5.1. 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

7.5.2. 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 bucket1 -p bucket_password | grep 'warmup'

cbstats hostname:port -b bucket1 -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**. This indicates 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**. 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_item_threshold	Enable data traffic after loading this number 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

Name	Description	Value Type
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 -p bucketname -b 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
ep_warmup_min_item_threshold	Enable data traffic after loading this number 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)

7.5.3. 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_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

7.6. 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.

> **Caution**
>
> Changes to the cluster configuration using **cbepctl** are not persisted over a cluster restart.

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 bucket1 -p bucket_password start
cbepctl host:11210 -b bucket1 -p bucket_password stop
cbepctl host:11210 -b bucket1 -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.

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.
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.

Parameter	Description
max_checkpoints	Max number of checkpoints allowed per vbucket.

7.6.1. Changing the Disk Cleanup Interval

One of the most important use cases for the `cbepctl flush_param` is the set the time interval for disk cleanup in Couchbase Server 2.0. 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 bucket1 -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.

7.6.2. 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 bucket1 -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 bucket1 -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, see Section 1.2.12, "Replicas and Replication". The other command options for `tap_param` are:

Parameter	Description
tap_keepalive	Seconds to hold a named tap connection.

Parameter	Description
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.

7.6.3. Changing Access Log Settings

In Couchbase Server 2.0, we provide a more optimized disk warmup. In past versions of Couchbase Server, the server would load all keys and data sequentially from vBuckets in RAM. Now the server pre-fetches a list of most-frequently accessed keys and fetches these documents first. The server runs a periodic scanner process which will determine which keys are most frequently-used. You can use **cbepctl flush_param** to change the initial time and the interval for the process. You may want to do this, for instance, 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.

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.

By default the scanner process will run 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 bucket1 -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 bucket1 -p bucket_password set flush_param alog_task_time 23
```

In this example we set the initial time to 11:00 PM UTC.

7.6.4. Changing Thresholds for Ejection

Couchbase Server has a process to *eject* items from RAM when too much space is being taken up in RAM; ejection means that documents will be removed from RAM, however the key and metadata for the item will remain in RAM. When a certain amount of RAM is consumed by items, the server will eject items starting with replica data. This threshold is known as the *low water mark*. If a second, higher threshold is breached, Couchbase Server will not only eject replica data, it will also eject less-frequently used items. This second RAM threshold is known as the *high water mark*. The server determines that items are not frequently used based on a boolean for each item known as NRU (Not-Recently-used). 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 2.0+ unless you are required to do so.

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.

For background information about the ejection process, the role of NRU and server processes related to ejection, see Section 5.4, "Ejection and Working Set Management".

Setting the Low Water Mark

This represents the amount of RAM you ideally want to consume on a node. If this threshold is met, the server will begin ejecting replica items as they are written to disk. To change this percentage for instance:

```
> ./cbepctl 10.5.2.31:11210 -b bucket_name -p bucket_password set flush_param mem_low_wat 70
```

You can also provides an absolute number of bytes when you change this setting.

Setting the High Water Mark

This represents the amount of RAM consumed by items that must be breached before infrequently used items will be ejected. To change this amount, you use the Couchbase command-line tool, **cbepctl**:

```
> ./cbepctl 10.5.2.31:11210 -b bucket_name -b bucket_password set flush_param mem_high_wat 80
```

Here we set the high water mark to be 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. You can also specify an absolute number of bytes when you set this threshold.

Setting Percentage of Ejected Items

After Couchbase Server removes all infrequently-used items and the high water mark is still breached, the server will then eject replicated data and active data from a node whether or not the data is frequently or infrequently used. You change also the default percentage for ejection of active items versus replica items using the Couchbase command-line tool, **cbepctl**:

```
> ./cbepctl 10.5.2.31:11210 -b bucket_name -p bucket_password set flush_param pager_active_vb_pcnt 50
```

This increases the percentage of active items that can be ejected from a node to 50%. Be aware of potential performance implications when you make this change. In very simple terms, 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 will not be immediately available. In that case, Couchbase Server has to retrieve the items from disk back into RAM and then it can respond to the requests. For Couchbase Server 2.0 we generally recommend that you do not change these defaults.

7.6.5. 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.

7.6.6. Enabling Flush of Data Buckets - Will be Deprecated

By default, this setting appears in Couchbase Web Console and is disabled; when it is enabled Couchbase Server is able to flush all the data in a bucket. **Be also aware that this operation will be deprecated as a way to enable data buck-**

et flushes. This is because **cbepctl** is designed for individual node configuration not operating on data buckets shared by multiple nodes.

The preferred way to enable data bucket flush is either 1) Couchbase Web Console or via 2) couchbase-cli. For more information about these two options, see Section 6.3.1, "Creating and Editing Data Buckets" and Section 7.4.1, "Flushing Buckets with **couchbase-cli**".

> **Warning**
>
> Flushing a bucket is data destructive. If you use **cbepctl**, it 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**.

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.

To enable flushing a data bucket:

```
> ./cbepctl hostname:port -b bucket_name -p bucket_password set flush_param flushall_enabled true
```

To disable flushing a data bucket:

```
> ./cbepctl hostname:port -b bucket_name -p bucket_password set flush_param flushall_enabled false
```

You can initiate the flush via the REST-API. For information about changing this setting in the Web Console, see Section 6.3, "Viewing Data Buckets". For information about flushing data buckets via REST, see Section 8.6.12, "Flushing a Bucket".

7.6.7. Other cbepctl 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.
flushall_enabled	Enable flush operation.
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_active_vb_pcnt	Percentage of active vbuckets items among all ejected items by item pager.
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_size	Max memory used by the server.

Parameter	Description
max_txn_size	Maximum number of items in a flusher transaction.
min_data_age	Minimum data age before flushing data.
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.

7.7. 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:

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.

As of Couchbase Server 2.1+ 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. For more information, see Section 7.5, "**cbstats** Tool"

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. For more information, see Working with Couchbase Customer Support.

7.8. 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. For general information about hostnames for the server, see Section 2.4, "Using Hostnames with Couchbase Server".

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**, see Section 7.10, "**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 Section 7.10, "**cbtransfer** Tool".

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, see Section 5.7.1, "Backing Up Using **cbbackup**".

Backing Up Design Documents Only

As of Couchbase Server 2.1 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**, see Section 7.9, "**cbrestore** Tool":

```
> ./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/setUpdateMinChange
        },
        "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. You can get a copy of the tool from the Couchbase command-line tools GitHub repository. 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.

Be aware that you can also use the **cbrestore 2.0** tool to restore backup data onto a 1.8.x cluster. See Section 7.9, "**cbrestore** Tool".

7.9. 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. For general information about hostnames for the server, see Section 2.4, "Using Hostnames with Couchbase Server".

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**, see Section 7.10, "**cbtransfer** Tool".

- `[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 bucker 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**. For a list of standard and special-use options, see Section 7.10, "**cbtransfer** Tool".

Using cbrestore for Design Documents Only

As of Couchbase Server 2.1 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**, see Section 7.8, "**cbbackup** Tool":

```
> ./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. For more information about the Views Editor, see Section 6.5, "Using the Views Editor".

Using cbrestore from Couchbase Server 2.0 with 1.8.x

You can use **cbrestore** 2.0 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. You can get a copy of the tool from the Couchbase command-line tools GitHub repository. 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 accomodate the difference in vBuckets:

```
cbrestore /backups/backup-42 memcached://HOST:11211 \
    --bucket-source=sessions --bucket-destination=sessions2
```

If you want more information about using **cbbackup** 2.0 tool to backup data onto a 1.8.x cluster. See Section 7.8, "**cb-backup** Tool".

For general information on using **cbbackup**, see Section 5.7.2.2, "Restoring using **cbrestore** tool".

7.10. 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.

> **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 per-form 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 infor-mation about hostnames for the server, see Section 2.4, "Using Hostnames with Couchbase Server".

Locations for this tool are as follows:

Linux	`/opt/couchbase/bin/`
Windows	`C:\Program Files\Couchbase\Server\bin\`
Mac OS X	`/Applications/Couchbase Server.app/Contents/Re-sources/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
-b BUCKET_SOURCE	Single named bucket from source cluster to transfer
-B BUCKET_DESTINATION, --buck-et-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

-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 will store all data from a bucket, node or cluster, but not the associated design documents. To to so, you should explicitly use **cbbackup** to store the information and **cbrestore** to read it back into memory.

Exporting and Importing CSV Files

As of Couchbase Server 2.1 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

As of Couchbase Server 2.1 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.

7.11. 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, see Section 1.2.17, "Statistics and Monitoring".

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 2.1 and later. You can find the tool in the following locations, depending upon your platform:

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
```

Where:

- CLUSTER

  The cluster for which you want a report:

-c HOST[:PORT] --cluster=HOST[:PORT]	Hostname and port of a node in the cluster. The default port is 8091.

- USERNAME

  Username of the cluster administrator account:

-u USERNAME --user=USERNAME	Admin username of the cluster.

- PASSWORD

  Password of the cluster administrator account:

-p PASSWORD --password=PASSWORD	Admin password of the cluster.

- OPTIONS

  Command 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.
-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:

7.12. 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:

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_PASSW

-b BUCKETNAME, --bucket=BUCKETNAME Specific bucket name. Default is default bucket. Bucket will be created if it does
```

```
-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. For more information about changing RAM quotas for Couchbase Server nodes, see ???.

7.13. cbworkloadgen Tool

Tool that generates random data and perform read/writes for Couchbase Server. This is useful for testing your Couchbase node.

Linux	/opt/couchbase/bin/tools/
Windows	C:\Program Files\Couchbase\Server\bin\tools\
Mac OS X	/Applications/Couchbase Server.app/Contents/Re-sources/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 10.17.30.161:9000 -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=="
```

7.14. 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:

Linux	/opt/couchbase/bin/tools/

Windows	Not Available on this platform.
Mac OS X	`/Applications/Couchbase Server.app/Contents/Re-sources/couchbase-core/bin/tools/`

7.15. vbuckettool Tool

Returns vBucket and node where a key should be for a Couchbase bucket. These two values based on Couchbase Server internal hashing algorithm. Moved as of 1.8 to /bin/tools directory.

Linux	`/opt/couchbase/bin/tools/`
Windows	`C:\Program Files\Couchbase\Server\bin\tools\`
Mac OS X	`/Applications/Couchbase Server.app/Contents/Re-sources/couchbase-core/bin/tools/`

Chapter 8. Using the REST API

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. You 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 you define. For more information about views, see Chapter 9, *Views and Indexes*.

> **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. See Couchbase SDKs.

The REST API accesses several different systems within the Couchbase Server product.

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, you should 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 node, 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 (http://www.ietf.org/rfc/rfc4627.txt) and www.json.org.

- **HTTP Basic Access Authentication**

  The Couchbase Management REST API uses HTTP basic authentication. The browser-based Chapter 6, *Using the Web Console* and Chapter 7, *Command-line Interface for Administration* 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.

8.1. Types of Resources

There are a number of different resources within the Couchbase Server and these resources will 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. For more information, see Chapter 9, *Views and Indexes*.

- **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. For more information, see Section 5.9, "Cross Datacenter Replication (XDCR)".

8.2. HTTP Request Headers

You will use the following HTTP request headers when you create your request:

Table 8.1. REST API — Supported Request Headers

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

8.3. HTTP Status Codes

The Couchbase Server will return one of the following HTTP status codes in response to your REST API request:

Table 8.2. REST API — HTTP Status Codes

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 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.

8.4. Using the Couchbase Administrative Console

The Couchbase Administrative Console uses many of the same REST API endpoints you would use 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.

For a list of supported browsers, see System Requirements. For the Couchbase Web Console, a separate UI hierarchy is served from each node of the system (though asking for the root "/" would likely return a redirect to the user agent). To launch the Couchbase Web Console, point your browser to the appropriate host and port, for instance on your development machine: `http://localhost:8091`

The operation and interface for the console is described in Chapter 6, *Using the Web Console*. For most of the administrative operations described in this chapter for the REST-API, you can perform the functional equivalent in Couchbase Web Console.

8.5. Managing Couchbase 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.

To view information about nodes that exist in a Couchbase Cluster, you use this request:

```
shell> curl -u admin:password 10.4.2.4:8091/pools/nodes
```

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}
```

8.5.1. Retrieving Statistics from Nodes

To retrieve statistics about a node, you can first retrieve a list of nodes in a cluster with this request:

```
shell> curl -u Admin:password http://10.4.2.4:8091/pools/default/buckets/default/nodes
```

You can send this request using the IP address and port for any node in the cluster. This 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: 10.4.2.4:8091
Accept: */*
```

If Couchbase Server successfully handles the request, you will get a response similar to the following example:

```
{"servers":[
  {"hostname":"10.4.2.6:8091",
  "uri":"/pools/default/buckets/default/nodes/10.4.2.6%3A8091",
  "stats":
      {"uri":"/pools/default/buckets/default/nodes/10.4.2.6%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:

```
shell> curl -u Administrator:password http://10.4.2.4:8091/pools/default/buckets/default/nodes/10.4.2.4%3A8091/stats
```

This 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: */*
```

If Couchbase Server successfully handles the reuqest, you will get a response similar to the following example:

```
{"hostname":"10.4.2.4:8091","hot_keys":[{"name":"[2012-11-05::3:47:01]"
....
"samplesCount":60,"isPersistent":true,"lastTStamp":1352922180718,"interval":1000}}
```

The statistics returned will be for the individual bucket associated with that node.

8.5.2. Provisioning a Node

Creating a new cluster or adding a node to a cluster is called *provisioning*. You need to:

- 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 you want to add a node to an existing cluster. If you create a new cluster you will need to secure it by providing an administrative username and password. If you add a node to an existing cluster you will need the URI and credentials to use the REST API with that cluster.

8.5.3. Configuring Index Path for a Node

The path for the index files can be configured through the use of the `index_path` parameter:

Example as follows:

```
shell> curl -X POST -u admin:password \
    -d index_path=/var/tmp/text-index \
    http://localhost:8091/nodes/self/controller/settings
```

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
```

The HTTP response will contain the response code and optional error message:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 0
```

As of Couchbase Server 2.0.1 if you try to set the data path at this endpoint, you will receive this 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'}
```

8.5.4. Setting Username and Password for a Node

While this can be done at any time for a cluster, it is typically the last step you complete when you add node into being a new cluster. The response will indicate the new base URI if the parameters are valid. Clients will want to send a new request for cluster information based on this response.

For example, using **curl**:

```
shell> curl -u admin:password -d username=Administrator \
    -d password=letmein \
    -d port=8091 \
    http://localhost:8091/settings/web
```

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
```

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**
>
> Note that even if it is not to be changed, the port number must be specified when you update user-name/password.

8.5.5. Configuring Node Memory Quota

The node memory quota configures how much RAM to be allocated to Couchbase for every node within the cluster.

Method	POST /pools/default
Request Data	Payload with memory quota setting
Response Data	Empty
Authentication Required	yes
Return Codes	
200	OK
400	Bad Request JSON: The RAM Quota value is too small.
401	Unauthorized

For example, to set the memory quota for a cluster at 400MB:

```
shell> curl -X POST -u admin:password -d memoryQuota=400 http://localhost:8091/pools/default
```

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
```

The HTTP response will contain the response code and optional error message:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 0
```

8.5.6. Providing Hostnames for Nodes

There are several ways you can provide hostnames for Couchbase 2.1+. You can provide a hostname when you install a Couchbase Server 2.1 node, when you add it to an existing cluster for online upgrade, or via a REST-API call. If a node restarts, any hostname you establish will be used. You cannot provide a hostname for a node that is already part of a Couchbase cluster; the server will reject the request and return error 400 reason: unknown ["Renaming is disallowed for nodes that are already part of a cluster"].

To see the specific REST request, see Section 2.4, "Using Hostnames with Couchbase Server".

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 more information, see Section 2.4, "Using Hostnames with Couchbase Server".

8.5.7. Manually Failing Over a Node

You can use this request to failover a node in the cluster. When you failover a node, it indicates the node is no longer available in a cluster and replicated data at another node should be available to clients. You can also choose to perform node failover using the Web Console, for more information, see Couchbase Server Manual, Initiating Node Failover.

Using the REST-API endpoint host:port/controller/failOver, provide your administrative credentials and the parameter optNode which is an internal name for the node:

```
> curl -v -X POST -u admin:password http://10.3.3.61:8091/controller/failOver -d otpNode=ns_2@10.3.3.63
```

The HTTP request will be similar to the following:

```
POST /controller/failOver HTTP/1.1
Authorization: Basic
```

Upon success, Couchbase Server will send a response as follows:

```
HTTP/1.1 200 OK
HTTP/1.1 200 OK
```

If you try to failover a node that does not exist in the cluster, you will get a HTTP 404 error. To learn more about how to retrieve `optNode` information for the nodes in a cluster, see Viewing Cluster Details.

8.6. Managing Buckets

The bucket management and configuration REST API endpoints are provided to fine level control over the individual buckets in the cluster, their configuration, and specific operations such as `FLUSH`.

8.6.1. Viewing Buckets and Bucket Operations

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:

```
shell> curl -u Administrator:password http://10.4.2.5:8091/pools/default/buckets
```

```
GET /pools/default/buckets
Host: localhost:8091
Authorization: Basic xxxxxxxxxxxxxxxxxxxx
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
        }
    }
]
```

8.6.2. Getting Individual Bucket Information

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:

```
shell> curl -u Administrator:password \
    http://10.4.2.5:8091/pools/default/buckets/default
```

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
```

8.6.3. Getting Bucket Statistics

You can use the REST API to get statistics with the 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** - provide statistics 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** - request statistics from this timestamp until now. You provide the timestamp as UNIX epoch time. You can get a timestamp for a timeframe by making a REST request to the endpoint with a zoom level.

The following is a sample request to the endpoint with no parameters:

```
curl -u user:password http://hostname:8091/pools/default/buckets/bucket_name/stats
```

The actual request appears as follows:

```
GET /pools/default/buckets/<bucket name>/stats
Host: localhost:8091
Authorization: Basic xxxxxxxxxxxxxxxxxx
Accept: application/json X-memcachekv-Store-Client-Specification-Version: 0.1
```

Upon success, you will see output similar to the following:

```
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, 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 follow are sample requests at this endpoint with optional parameters:

- ```
  curl -u user:password  -d zoom=minute http://hostname:8091/pools/default/buckets/bucket_name/stats
  ```

  This will sample statistics from a bucket for the last minute.

- ```
  curl -u user:password  -d zoom=day http://hostname:8091/pools/default/buckets/bucket_name/stats
  ```

  This will sample statistics from a bucket for the past day.

- Using zoom level of a month:

  ```
  curl -u user:password  -d zoom=month http://hostname:8091/pools/default/buckets/bucket_name/stats
  ```

  This will sample statistics from a bucket for the last month.

- Using zoom level of an hour from a specific timestamp:

  ```
  curl -u user:password   -d zoom=hour&haveTStamp=1376963720000 http://hostname:8091/pools/default/buckets/bucket_name
  ```

  This will sample statistics from a bucket from the timestamp until the server receives the REST request.

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. We 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, ... ],
         "ef_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, ... ]
      }
   }
}
```

8.6.4. Using the Bucket Streaming 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 described previously. 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 xxxxxxxxxxxxxxxxxxxx
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
  }
}
```

8.6.5. Creating and Editing Data Buckets

You can create a new bucket with a POST command sent to the URI for buckets in a cluster. This can be used to create either a Couchbase or a Memcached type bucket. The bucket name cannot have a leading underscore.

To create a new Couchbase bucket, or edit the existing parameters for an existing bucket, you can send a POST to the REST API endpoint. You can also use this same endpoint to get a list of buckets that exist for a cluster.

Be aware that when you edit bucket properties, if you do not specify an existing bucket property Couchbase Server may reset this the property to be the default. So even if you do not intend to change a certain property when you edit a bucket, you should specify the existing value to avoid this behavior.

This REST API will return a successful response when preliminary files for a data bucket are created on one node. Because you may be using a multi-node cluster, bucket creation may not yet be complete 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 ensure a bucket is available the recommended approach is try to read a 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. See Couchbase Developer Guide 2.0, Performing Connect, Set and Get.

Method	POST /pools/default/buckets
Request Data	List of payload parameters for the new bucket
Response Data	JSON of the bucket confirmation or error condition
Authentication Required	yes
Payload Arguments	
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.
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. For detailed information about this feature, see Section 5.1, "Using Multi- Readers and Writers".
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.

For example:

```
shell> 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.

8.6.6. Getting Bucket Configuration

To obtain the information about an existing bucket, use the main REST API bucket endpoint with the bucket name. For example:

```
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
            }
        }
    ]
}
```

Clients MUST use the nodes list from the bucket, not the pool to indicate which are the appropriate nodes to connect to.

8.6.7. Modifying Bucket Parameters

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:

```
shell> curl -v -X POST -u Administrator: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
```

Available parameters are identical to those available when creating a bucket. See bucket parameters.

If the request is successful, HTTP response 200 will be returned with an empty data content.

Warning

You cannot change the name of a bucket via the REST API.

8.6.8. Increasing the Memory Quota for a Bucket

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.

Warning

As of 1.6.0, there are some known issues with changing the ramQuotaMB for memcached bucket types.

Example of a request:

```
shell> curl -X POST -u admin:password -d ramQuotaMB=25 -d authType=none \
    -d proxyPort=11215 http://localhost:8091/pools/default/buckets/newbucket
```

The response will be 202, indicating the quota will be changed asynchronously throughout the servers in the cluster. An example:

```
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
```

8.6.9. Changing Bucket Authentication

Changing a bucket from port based authentication to SASL authentication can be achieved by changing the active bucket configuration. You must specify the existing configuration parameters and the changed authentication parameters in the request:

```
shell> curl -X POST -u admin:password -d ramQuotaMB=130 -d authType=sasl \
    -d saslPassword=letmein \
    http://localhost:8091/pools/default/buckets/acache
```

8.6.10. Compacting Bucket Data and Indexes

Couchbase Server will write 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 for-

mat 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. For more general information on this administrative task, see Section 5.5, "Database and View Compaction".

Compacting Data Buckets and Indexes

To compact data files for a given bucket as well as any indexes associated with that bucket, you perform a request as follows:

```
shell> curl -i -v -X POST -u Administrator:password http://[ip]:[port]/pools/default/buckets/[bucket-name]/controller
```

Where you provide the ip and port for a node that accesses the bucket as well as the bucket name. You will also need to provide administrative credentials for that node in the cluster. To stop bucket compaction, you issue this request:

```
shell> curl -i -v -X POST -u Administrator:password http://[ip]:[port]/pools/default/buckets/[bucket-name]/controller
```

Compacting Spatial Views

If you have spatial views configured within your dataset, these are not automatically compacted for you. Instead, you must manually compact each spatial view through the REST API.

To do this, you must call the spatial compaction routine at the URL format:

```
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**:

```
shell> curl -X POST \
    'http://127.0.0.1:9500/default/_design/dev_test_spatial_compaction/_spatial/_compact'
    -H 'Content-type: application/json'
```

8.6.11. Deleting a Bucket

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.

To delete a bucket, you supply the URL of the Couchbase bucket using the `DELETE` operation. For example:

```
DELETE /pools/default/buckets/default
Host: localhost:8091
Authorization: Basic xxxxxxxxxxxxxxxxxx
```

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.

8.6.12. Flushing a Bucket

> **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**.
>
> For information about changing this setting in the Web Console, see Section 6.3, "Viewing Data Buckets". For information about flushing data buckets via REST, see Section 8.6.12, "Flushing a Bucket".

The `doFlush` operation empties the contents of the specified bucket, deleting all stored data. The operation will only succeed if flush is enabled on configured bucket. The format of the request is the URL of the REST endpoint using the `POST` HTTP operation:

```
http://localhost:8091/pools/default/buckets/default/controller/doFlush
```

For example, using **curl**:

```
shell> curl -X POST 'http://Administrator:Password@localhost:8091/pools/default/buckets/default/controller/doFlush'
```

The equivalent HTTP protocol request:

```
POST /pools/default/buckets/default/controller/doFlush
Host: localhost:8091
Authorization: Basic xxxxxxxxxxxxxxxxxx
```

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
```

Warning

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

Also note that the flush request is not transmitted over XDCR replication configurations; the remote bucket will not be flushed.

Couchbase Server will return 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
```

You can configure whether flush is enabled for a bucket by configuring the individual bucket properties, either the REST API (see Section 8.6.7, "Modifying Bucket Parameters"), or through the Admin Console (see Section 6.3.1, "Creating and Editing Data Buckets").

8.7. Managing Clusters

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:

```
shell> curl -u admin:password http://localhost:8091/pools
```

As a raw HTTP request:

```
GET /pools
Host: localhost:8091
Authorization: Basic xxxxxxxxxxxxxxxxxxxx
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.

8.7.1. 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**
>
> Note that since buckets could be renamed and there is no way to determine the name for the default bucket for a cluster, the system will attempt to connect non-SASL, non-proxied to a bucket clients to a bucket named "default". If it does not exist, Couchbase Server will drop the connection.

You should not rely on the node list returned by this request to connect to a Couchbase Server. You should 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 xxxxxxxxxxxxxxxxxxxx
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:

Table 8.3. REST API — Controller 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.

8.7.2. Adding a Node to a Cluster

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`. You will need to provide an administrative username and password as parameters:

```
shell> curl -u Administrator:password \
    10.2.2.60:8091/controller/addNode \
    -d "hostname=10.2.2.64&user=Administrator&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. If successful, Couchbase Server will respond:

```
HTTP/1.1 200 OK
{"otpNode":"ns_1@10.4.2.6"}
```

8.7.3. Joining a Node into a Cluster

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:

```
shell> curl -u admin:password -d clusterMemberHostIp=192.168.0.1 \
    -d clusterMemberPort=8091 \
    -d user=admin -d password=admin123
    http://localhost:8091/node/controller/doJoinCluster
```

The following arguments are required:

Table 8.4. REST API — Cluster Joining Arguments

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, you will need to provide the following parameters in your request:

Table 8.5. REST API — Cluster Joining Additional Arguments

Argument	Description
user	Administration user
password	Password associated with the Administration user

```
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
```

```
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
```

8.7.4. Removing a Node from a Cluster

When a node is temporarily or permanently down, you may want to remove it from a cluster:

```
shell> curl -u admin:password -d otpNode=ns_1@192.168.0.107 \
    http://192.168.0.106:8091/controller/ejectNode
```

```
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
```

8.7.5. Initiating a Rebalance

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. You can obtain this information by getting the current node configuration from Section 8.5, "Managing Couchbase Nodes". 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. For example:

```
> curl -v -u Administrator:password -X POST
'http://172.23.121.11:8091/controller/rebalance' -d
'ejectedNodes=&knownNodes=ns_1@172.23.121.11,ns_1@172.23.121.12'
```

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}
```

Progress of the rebalance operation can be obtained by using Section 8.7.6, "Getting Rebalance Progress".

8.7.6. 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, see Section 5.8.4, "Monitoring a Rebalance" for details and definitions.

This first request returns a JSON structure containing the current progress information:

```
> curl -u admin:password 'http://Administrator:Password@192.168.0.77:8091/pools/default/rebalanceProgress'
```

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 this request

```
gt; curl -u admin:password 'http://ip_address:port/pools/default/tasks'
```

```
GET /pools/default/rebalanceProgress HTTP/1.1
Authorization: Basic QWRtaW5pc3RyYXRvcjpUYWlzaW4=
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 will show 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, see Section 5.8.4, "Monitoring a Rebalance". 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."
  }
]
```

8.7.7. Adjusting Rebalance during Compaction

If you perform a rebalance while a node is undergoing index compaction, you may experience delays in rebalance. There is REST-API parameter as of Couchbase Server 2.0.1 you can use 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.

To make this request:

```
wget --post-data='rebalanceMovesBeforeCompaction=256'
--user=Administrator --password=pass http://lh:9000/internalSettings
```

This needs to be made as POST request to the `/internalSettings` endpoint. By default this setting is 16, 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.

8.7.8. 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:

```
shell> curl -u Administrator:letmein http://localhost:8091/settings/autoFailover
```

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 }
```

8.7.9. 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:

```
shell> curl "http://localhost:8091/settings/autoFailover" \
    -i -u Administrator:letmein -d 'enabled=true&timeout=600'
```

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.
```

8.7.10. 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:

```
shell> curl -X POST -i -u Administrator:letmein \
    http://localhost:8091/settings/autoFailover/resetCount
```

```
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
```

8.7.11. 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:

```
shell> curl -X POST -u admin:password -d maxBucketCount=6 http://ip_address:8091/internalSettings
```

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."]
```

8.7.12. 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.

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.

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

8.7.13. 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:

```
shell> curl -u Administrator:letmein http://localhost:8091/settings/alerts
```

```
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.
```

8.7.14. 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 will trigger notification:

```
shell> curl -i -u Administrator:letmein \
    -d 'enabled=true&sender=couchbase@localhost&recipients=admin@localhost,membi@localhost&emailHost=localhost&emailP
```

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=localh
```

```
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.

8.7.15. 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.

```
shell> curl -i -u Administrator:letmein http://localhost:8091/settings/alerts/sendTestEmail \
  -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_t
```

```
POST /settings/alerts/sendTestEmail HTTP/1.1
Host: localhost:8091
Content-Type: application/x-www-form-urlencoded
Authorization: Basic YWRtaW46YWRtaW4=
```

200 OK

Possible errrors include:

```
400 Bad Request: Unknown macro: {"error"} 401 Unauthorized
This endpoint isn't available yet.
```

8.7.16. 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.

Method	GET /internalSettings
Request Data	None
Response Data	JSON of current internal settings
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.

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:

```
shell> curl -v -X POST http://Administrator:Password@localhost:8091/internalSettings \
    -d maxBucketCount=20
```

8.7.17. 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.

> 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:

```
shell> curl -v -u Administrator:password -X POST http://10.4.2.4:8091/internalSettings \
    -d indexAwareRebalanceDisabled=true
```

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, see Section 9.2, "View Operation".

8.8. Managing Views with REST

In Couchbase 2.0 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. Please refer to the following resources:

- Storing a Design Document.

- Retrieving a Design Document.

- Deleting a Design Document.

- Querying Views via the REST-API. Section 9.8.1, "Querying Using the REST API".

8.8.1. Limiting Simultaneous Node Requests

As of Couchbase 2.1+ 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. For general information about this endpoint, see Section 8.7.16, "Managing Internal Cluster Settings".

 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:

```
wget --post-data='capiRequestLimit=50'
--user=Administrator --password=pass http://a_hostname:9000/internalSettings
```

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.

8.9. Managing Cross Data Center Replication (XDCR)

Cross Datacenter Replication (XDCR) enables you to automatically replicate data between clusters and between data buckets. There are several endpoints for the Couchbase REST API that you can use specifically for XDCR. For more information about using and configuring XDCR, see Section 5.9, "Cross Datacenter Replication (XDCR)".

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 will replicate 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. For more information about using Couchbase Web Console to configure XDCR, see Section 5.9, "Cross Datacenter Replication (XDCR)".

8.9.1. 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:

```
shell> curl -u Administrator:password http://10.4.2.5:8091/pools/default/remoteClusters
```

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 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) 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.

For more information about XDCR and using XDCR via the Couchbase Web Console, see Section 5.9, "Cross Datacenter Replication (XDCR)".

8.9.2. 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:

```
shell> curl -v -u Administrator:password1 10.4.2.4:8091/pools/default/remoteClusters \
-d uuid=9eee38236f3bf28406920213d93981a3  \
-d name=remote1
-d hostname=10.4.2.6:8091
-d username=Administrator -d password=password2
```

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, see Section 5.9.5, "Configuring Replication".

8.9.3. 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:

```
shell> curl -v -X DELETE -u Administrator:password1 10.4.2.4:8091/pools/default/remoteClusters/remote1
```

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"
```

For more information about XDCR and references to destination clusters via the Couchbase Web Console, see Section 5.9.5, "Configuring Replication".

8.9.4. 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:

```
shell> curl -v -X POST -u Administrator: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
```

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 create 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.

For more information about XDCR and creating a new replication see Section 5.9.5, "Configuring Replication".

8.9.5. 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:

```
shell> curl -u Administrator:password1  \
http://10.4.2.4:8091/controller/cancelXDCR/9eee38236f3bf28406920213d93981a3%2Fbeer-sample%2Fremote_beer  \
-X DELETE
```

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. For more information, see Section 5.9.5, "Configuring Replication".

8.9.6. Viewing Internal XDCR Settings

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

```
shell> curl -u Administrator:password1  \
http://10.4.2.4:8091/internalSettings
```

You will recieve a response similar to the following. For the sake of brevity, we are showing only the XDCR-related items:

```
{
....

"xdcrMaxConcurrentReps":33,
"xdcrCheckpointInterval":222,
"xdcrWorkerBatchSize":555,
"xdcrDocBatchSizeKb":999,
"xdcrFailureRestartInterval":44
}
```

The the XDCR-related values are defined as follows:

- (Number) xdcrMaxConcurrentReps: Maximum concurrent replications per bucket, 8 to 256. Default is 32. 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.

- (Number) xdcrCheckpointInterval: Interval between checkpoints, 60 to 14400 (seconds). Default 1800.

- (Number) xdcrWorkerBatchSize: Document batching count, 500 to 10000. Default 500.

- (Number) xdcrDocBatchSizeKb: Document batching size, 10 to 100000 (kB). Default 2048.

- (Number) xdcrFailureRestartInterval: Interval for restarting failed XDCR, 1 to 300 (seconds). Default 30.

For more information about XDCR, see Section 5.9, "Cross Datacenter Replication (XDCR)".

8.9.7. Changing Internal XDCR Settings

There are internal settings for XDCR which are only exposed via the REST API. These settings will change the replication behavior, performance, and timing. The following updates an XDCR setting for parallel replication streams per node:

```
shell> curl -X POST -u Administrator:password1  \
http://10.4.2.4:8091/internalSettings  \
-d xdcrMaxConcurrentReps=64
```

If Couchbase Server successfully updates this setting, it will send a response as follows:

```
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
```

How you adjust these variables differs based on what whether you want to perform uni-directional or bi-directional 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 will impact performance of your clusters as well as XDCR replication performance. The the XDCR-related settings which you can adjust are defined as follows:

- xdcrMaxConcurrentReps (Integer)

  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.

- xdcrCheckpointInterval (Integer)

  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 bi-directional 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.

- `xdcrWorkerBatchSize` (Integer)

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 bi-directional replication between two clusters and the destination already handles a significant volume of reads/writes.

- `xdcrDocBatchSizeKb` (Integer)

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 bi-directional replication between two clusters and the destination already handles a significant volume of reads/writes.

- `xdcrFailureRestartInterval` (Integer)

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.

- `xdcrOptimisticReplicationThreshold` (Integer)

Document size in bytes. 0 to 2097152 Bytes (20MB). Default is 256 Bytes. XDCR will get metadata for documents larger than this size on a single time before replicating the document to a destination cluster.

8.9.8. 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. For instructions, see Section 8.9.1, "Getting a Destination Cluster Reference". 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
```

You need to provide properly URL-encoded `/[UUID]/[source_bucket]/[destination_bucket]/` `[stat_name]`. To get the number of documents written:

```
curl-X GET http://hostname:port/pools/default/buckets/default/stats/replications%2F8ba6870d88cd72b3f1db113fc8aee675%2
```

Will produce this output:

```
{"samplesCount":60,"isPersistent":true,"lastTStamp":1371685106753,"interval":1000,
"timestamp":[1371685048753,1371685049754,1371685050753,1371685051753,1371685052753,1371685053753,1371685054753,
1371685055753,1371685056753,1371685057753,1371685058752,1371685059753,1371685060753,1371685061753,1371685062753,
1371685063753,1371685064753,1371685065753,1371685066753,1371685067753,1371685068753,1371685069753,1371685070753,
1371685071753,1371685072753,1371685073753,1371685074753,1371685075753,1371685076753,1371685077753,1371685078753,
1371685079753,1371685080753,1371685081753,1371685082753,1371685083753,1371685084753,1371685085753,1371685086753,
1371685087753,1371685088753,1371685089753,1371685090753,1371685091754,1371685092753,1371685093753,1371685094753,
1371685095753,1371685096753,1371685097753,1371685098753,1371685099753,1371685100753,1371685101753,1371685102753,
1371685103753,1371685104753,1371685105753,1371685106753],
"nodeStats":{"127.0.0.1:9000":[1000000,1000000,1000000,1000000,1000000,1000000,1000000,1000000,1000000,1000000,
1000000,1000000,1000000,1000000,1000000,1000000,1000000,1000000,1000000,1000000,1000000,1000000,1000000,1000000,
1000000,1000000,1000000,1000000,1000000,1000000,1000000,1000000,1000000,1000000,1000000,1000000,1000000,1000000,
1000000,1000000,1000000,1000000,1000000,1000000,1000000,1000000,1000000,1000000,1000000,1000000,1000000,1000000,
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 -X GET http://hostanme:port/pools/default/buckets/default/stats/replications%2F8ba6870d88cd72b3f1db113fc8aee675%2
```

This will produce this output:

```
{"samplesCount":60,"isPersistent":true,"lastTStamp":1371685006753,"interval":1000,
"timestamp":[1371684948753,1371684949753,1371684950753,1371684951753,1371684952753,1371684953753,1371684954753,
```

```
1371684955754,1371684956753,1371684957753,1371684958753,1371684959753,1371684960753,1371684961753,1371684962753,
1371684963753,1371684964753,1371684965753,1371684966753,1371684967753,1371684968752,1371684969753,1371684970753,
1371684971753,1371684972753,1371684973753,1371684974753,1371684975753,1371684976753,1371684977753,1371684978753,
1371684979753,1371684980753,1371684981753,1371684982753,1371684983753,1371684984753,1371684985754,1371684986753,
1371684987754,1371684988753,1371684989753,1371684990753,1371684991753,1371684992753,1371684993753,1371684994753,
1371684995753,1371684996753,1371684997753,1371684998776,1371684999753,1371685000753,1371685001753,1371685002753,
1371685003753,1371685004753,1371685005753,1371685006753],
"nodeStats":{"127.0.0.1:9000":[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,
```

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 will return 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, \
................
}
```

Finally be aware that we expose many of these statistics in Couchbase Web Console. For more information, see Section 6.4.1.6, "Monitoring Outgoing XDCR".

8.10. Using System Logs

Couchbase Server logs various messages, which are available via the REST API. These log messages are optionally categorized by the module. You can retrieve a generic list of recent log entries or recent log entries for a particular category. If you perform a GET request on the systems logs URI, Couchbase Server will return all categories of messages.

Messages may be labeled, "info" "crit" or "warn". Accessing logs requires administrator credentials if the system is secured.

```
GET /pools/default/logs?cat=crit
Host: localhost:8091
Authorization: Basic xxxxxxxxxxxxxxxxxx
Accept: application/json
X-memcachekv-Store-Client-Specification-Version: 0.1
```

```
201: bucket was created and valid URIs returned
HTTP/1.1
200 OK
Content-Type: application/json
Content-Length: nnn
[
    {
        "cat":"info",
        "date": "",
        "code": "302",
        "message": "Some information for you."
    },
    {
        "cat":"warn",
        "date": "",
        "code": "502",
        "message": "Something needs attention."
    }
]
```

8.11. Client Logging Interface

If you create your own Couchbase SDK you may might want to add entries to the central log. These entries would typically be responses to exceptions such as difficulty handling a server response. For instance, the Web UI uses this functionality to log client error conditions. To add entries you provide a REST request:

```
POST /logClientError
Host: localhost:8091
Authorization: Basic xxxxxxxxxxxxxxxxxx
Accept: application/json
X-memcachekv-Store-Client-Specification-Version: 0.1
```

```
200 - OK
```

```
POST /logClientError
Host: localhost:8091
Authorization: Basic xxxxxxxxxxxxxxxxxx
Accept: application/json
X-memcachekv-Store-Client-Specification-Version: 0.1
```

Chapter 9. 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.

> **Note**
>
> 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. For more information, read Section 9.2, "View Operation".

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.

- For background information on the creation of views and how they relate to the contents of your Couchbase Server database, see Section 9.1, "View Basics".

- For more information on how views work with stored information, see Section 9.3, "Views and Stored Data".

- For information on the rules and implementation of views, see Section 9.2, "View Operation".

- Two types of views, development and production, are used to help optimize performance and view development. See Section 9.4, "Development and Production Views".

- Writing views, including the language and options available are covered in Section 9.4, "Development and Production Views".

- For a detailed background and technical information on troubleshooting views, see Appendix C, *Troubleshooting Views (Technical Background)*.

- The Couchbase Server Web Console includes an editor for writing and developing new views. See Section 6.5, "Using the Views Editor". You can also use a REST API to create, update and delete design documents. See Section 9.7, "Design Document REST API".

9.1. View Basics

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 generat-

ed 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.

Figure 9.1. Views — Basic Overview
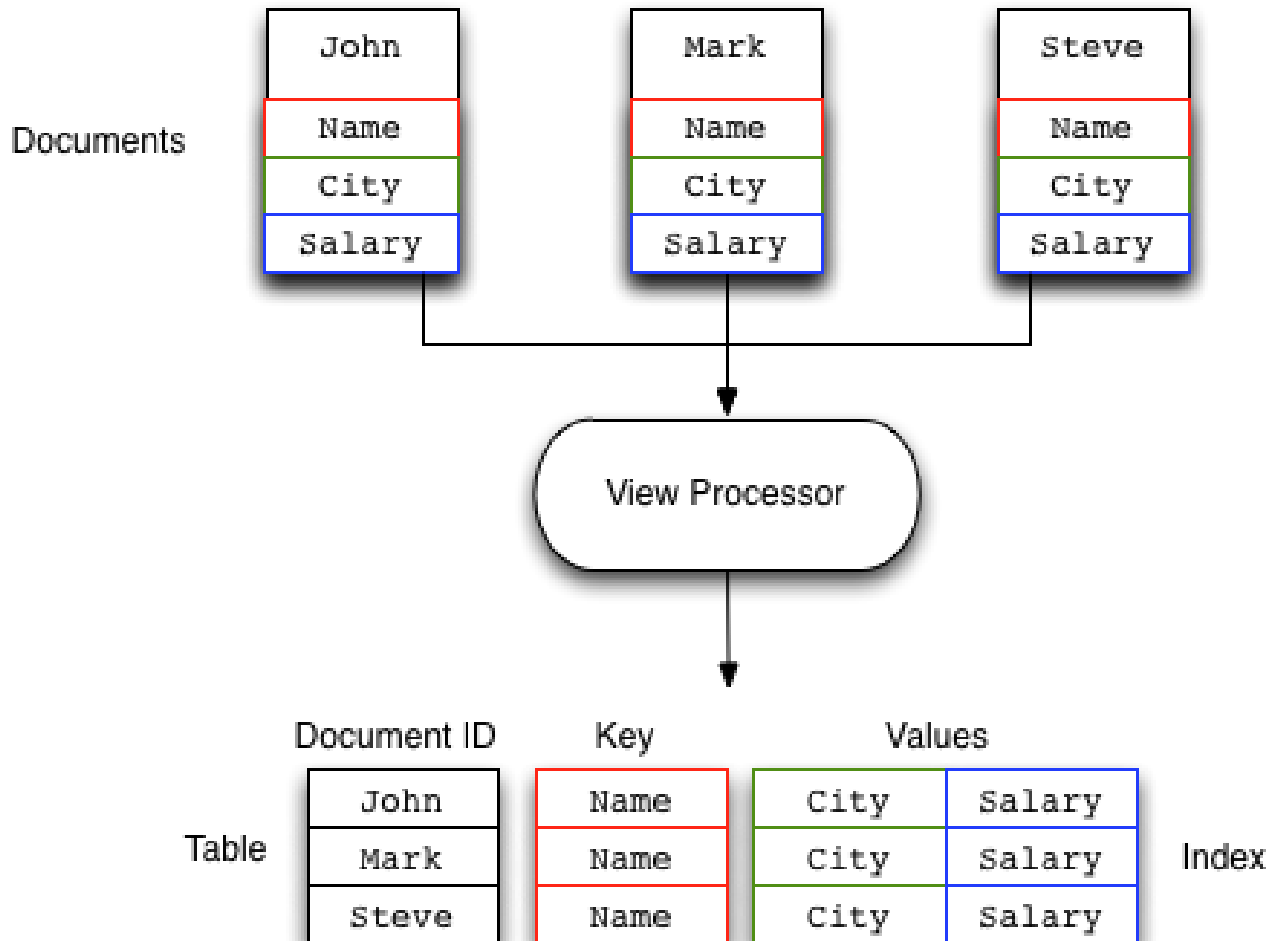
In the above example, the view takes the Name, City and Salary fields from the stored documents and then creates a array of this information for each document in the view. A view is created by iterating over every single document within the Couchbase bucket and outputting the specified information. The 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.

9.2. View 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 (see Section 9.2.4, "Index Updates and 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. See Section 9.2.5, "Automated Index Updates".

- 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. See Section 5.5, "Database and View Compaction".

- 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.

> **Note**
>
> 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. For more information, see Section 9.2.4, "Index Updates and the `stale` Parameter".

- 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.

9.2.1. 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_doc` 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","expir
{"id":"test2","key":"test2","value":null,"doc":{"meta":{"id":"test2","rev":"3-0000004134bd596f50bce37d00000000","ex
]
}
```

For expired documents if you set `include_doc=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","expir
{"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 2.0. 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 Manual command line tool, Couchbase Server Manual 2.0, Specifying Disk Cleanup Interval and refer to the examples on `exp_pager_stime`. For more information about views and view query parameters, see Finding Data with Views .

9.2.2. 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:

Figure 9.2. View Indexes — in a Cluster
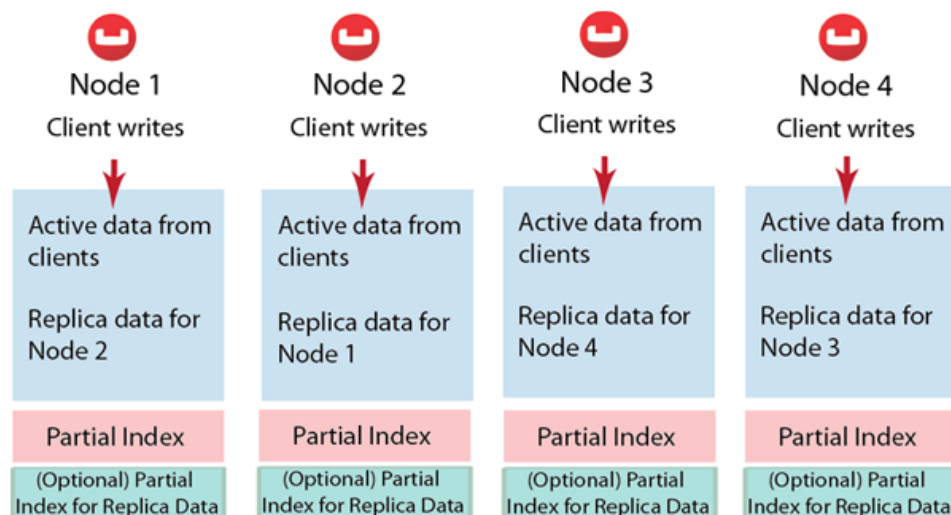
Couchbase Server can optionally create replica indexes on nodes that are contain replicated data; this is to prepare your cluster for a failover scenario. The server does not replicate index information from another node, instead each node creates an index for the replicated data it stores. The server 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. For more information, see Creating and Editing Data Buckets

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. For more information about controlling index updates using query parameters, see Index Updates and the stale Parameter.

To handle errors when you perform a query, you can configure how the cluster behaves when errors occur. See Section 9.8.7, "Error Control".

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. For more information, see Disabling Consistent Query Results on Rebalance. 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.

9.2.3. 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.

9.2.4. 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.

> **Note**
>
> 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.

> **Note**
>
> Views can also be updated automatically according to a document change, or interval count. See Section 9.2.5, "Automated Index Updates".

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.

Figure 9.3. Views — Index Updates — Stale OK

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.

Figure 9.4. Views — Index Updates — Update Before
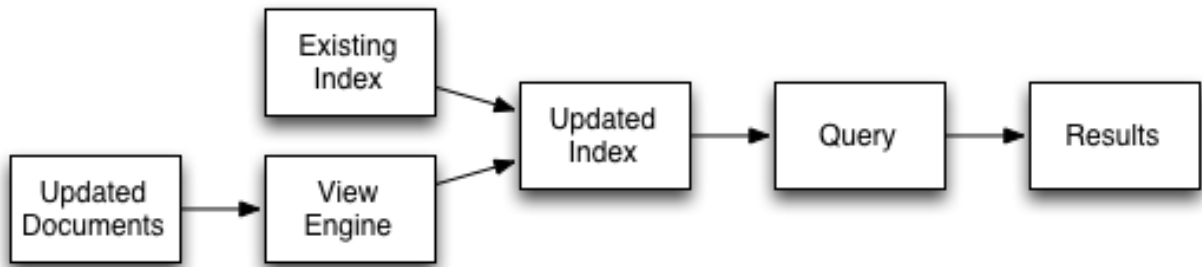
- **stale=update_after**

  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.

Figure 9.5. Views — Index Updates — Update After
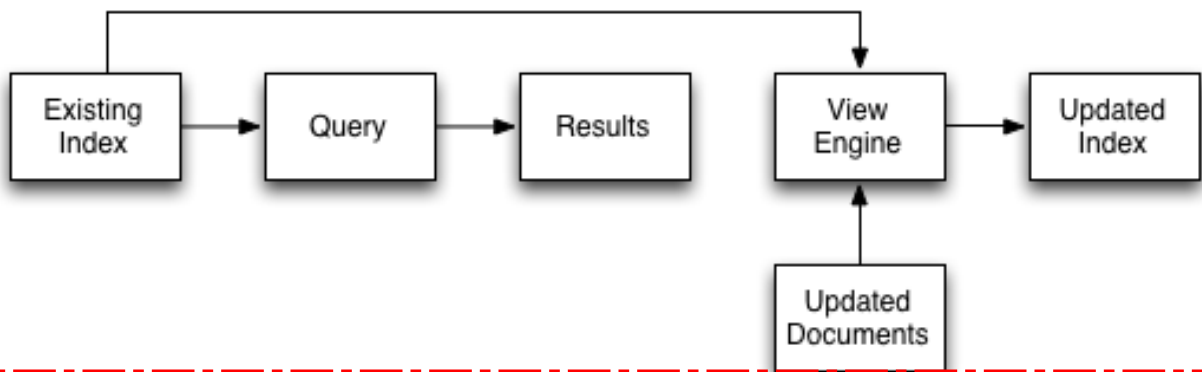
> **Warning**
>
> The indexing engine is an asynchronous process; this means querying an index may produce results you may not expect. For example, if you update a document, and then immediately run a query on that document you may not get the new information in the emitted view data. This is because 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. For more information, see Couchbase Developer Guide, Using Observe.

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.

Note

> 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.

9.2.5. 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.

> **Note**
>
> 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 update values explicitly on individual design documents, you must 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. For more information, see Section 9.7, "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'
```

Note

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.

9.3. 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.

Figure 9.6. Views — Data Storage
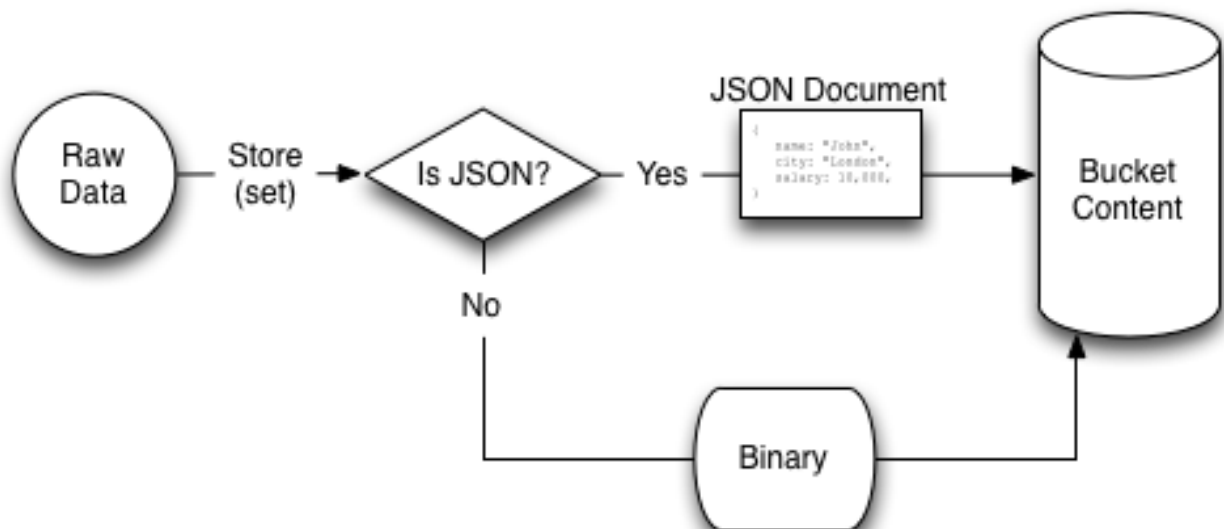
When retrieving the stored data, the format of the information depends on whether the data was tagged as valid JSON or not:

- **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 whitespace, 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 parse-able 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.

9.3.1. 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).

Note

> 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-literals 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"
}
```

Warning

If the submitted data cannot be parsed as a JSON, the information will be stored as a binary object, not a JSON document.

9.3.2. 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 sotred as an absolute Unix epoch time value.

Note

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.

9.3.3. 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.

> **Note**
>
> 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.
>
> For more information and examples, see Section 9.5.3, "Views on non-JSON Data".

9.3.4. 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.

> **Note**
>
> 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. For more information, see Section 9.2.4, "Index Updates and the `stale` Parameter". Views can also be automatically updated on a schedule so that their data is not too out of sync with stored documents. For more information, see Section 9.2.5, "Automated Index Updates".

9.4. Development and 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: `Development` views and `Production` views. The two view types work identically but have different purposes and restrictions placed upon their operation.

- **Development Views**

  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.

    Warning

    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.

- **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:

Figure 9.7. Views — View Type Workflow
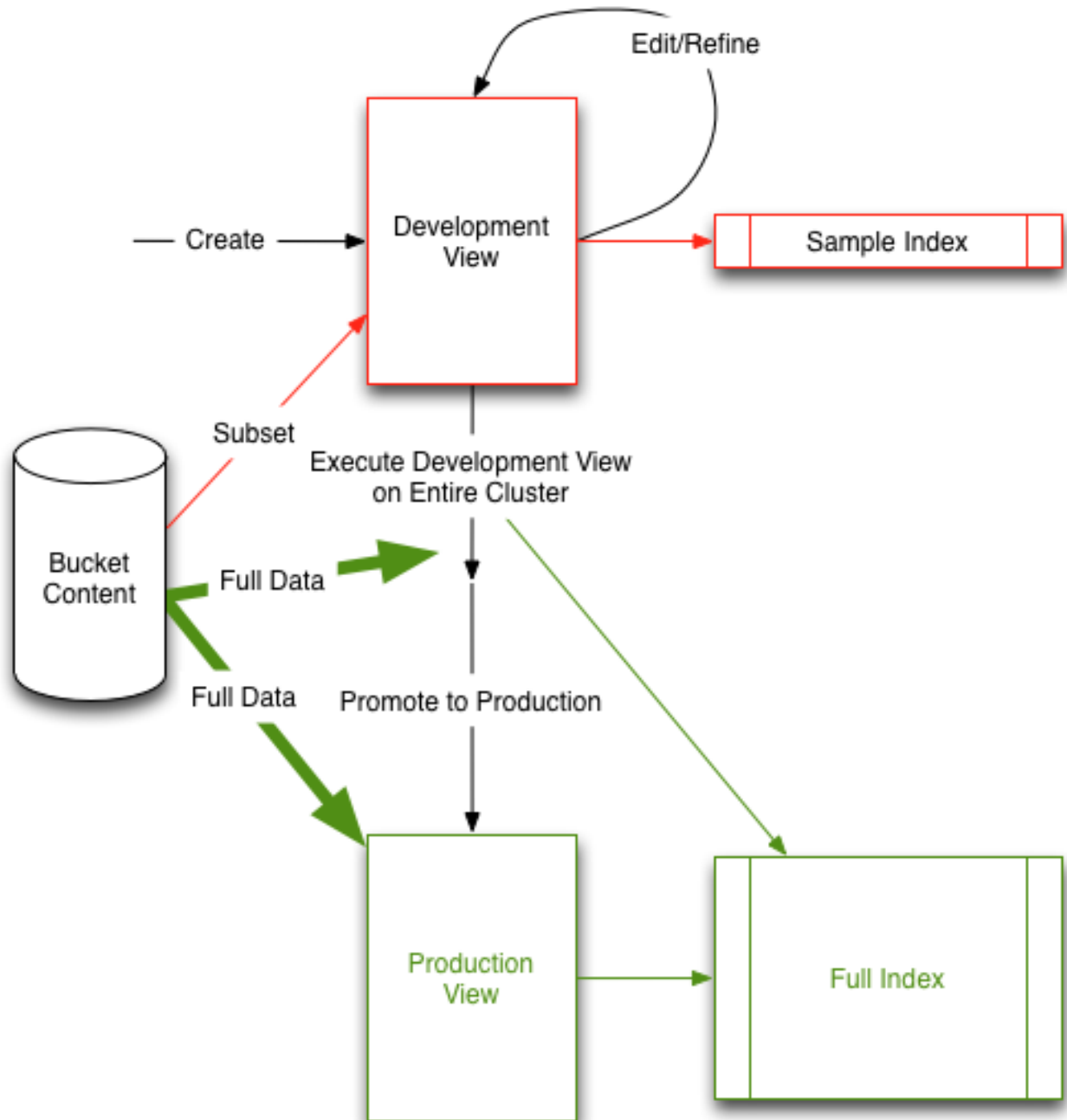
The above diagram features the following steps:

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, see Section 6.5.2, "Publishing Views".

9.5. 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 sum-

marising 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.

Figure 9.8. Views — View Building
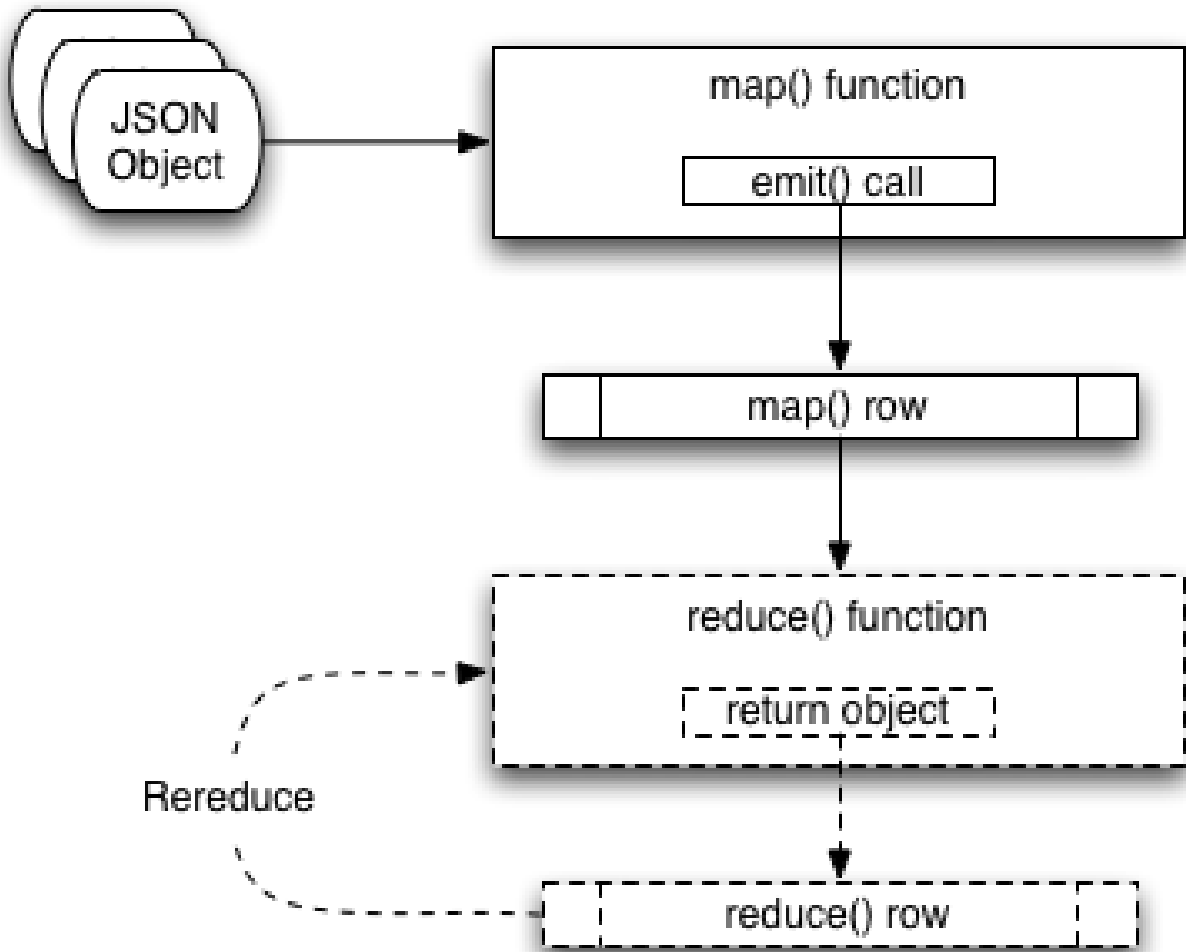
Because of the separation of the two elements, you can consider the two functions individually.

For information on how to write map functions, and how the output of the map function affects and supports searching, see Section 9.5.1, "Map Functions". For details on writing the reduce function, see Section 9.5.2, "Reduce Functions".

Note

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 (see Section 6.5, "Using the Views Editor"), use the REST API for design documents (see Section 9.7, "Design Document REST API"), or use one of the client libraries that support view management.

For more information and examples on how to query and obtain information from a map, see Section 9.8, "Querying Views".

9.5.1. 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.

Figure 9.9. Views — Writing Map Functions
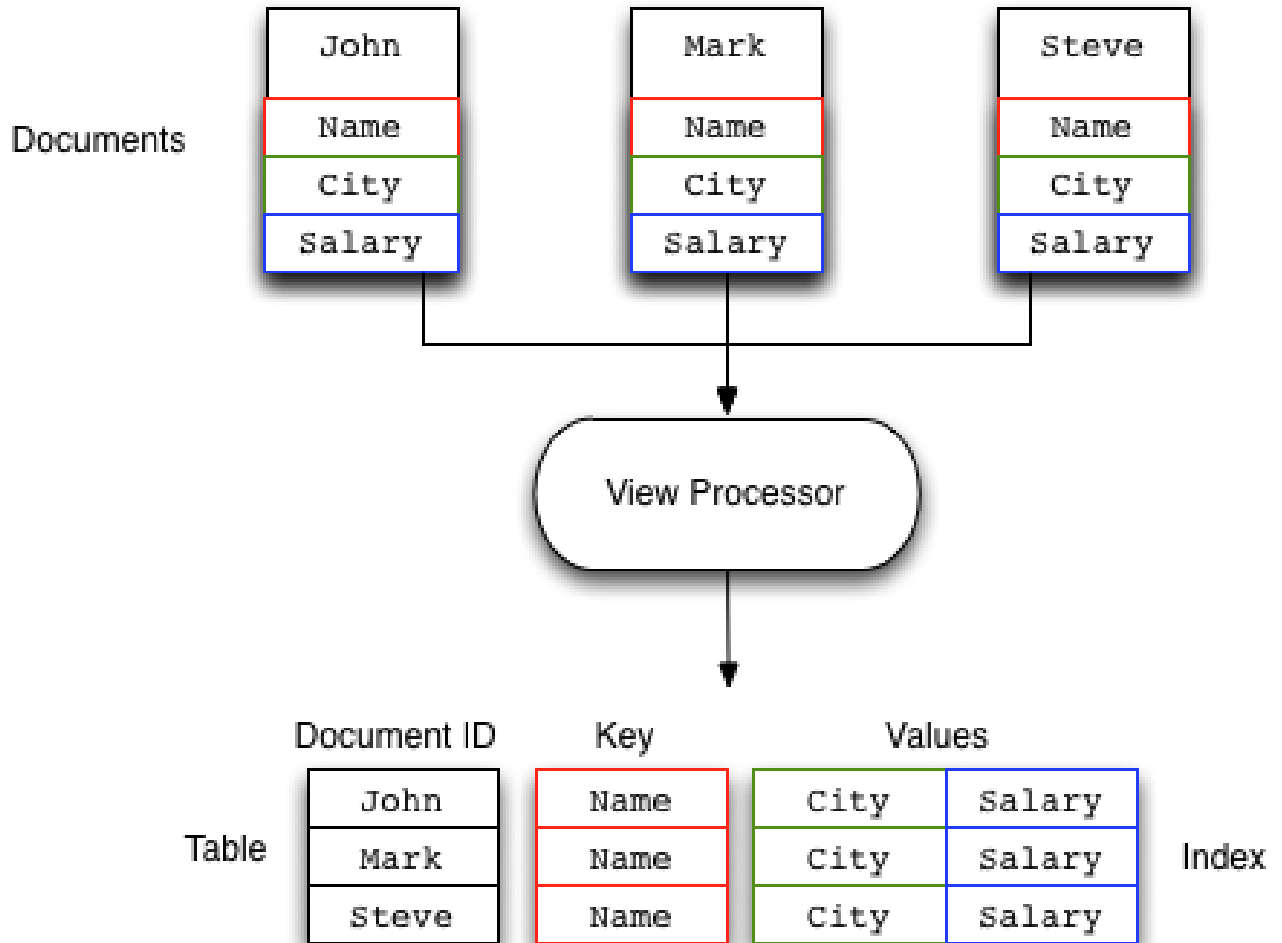
In this example, a map function is taking the Name, City, and Salary fields from the JSON documents stored in the Couchbase bucket and mapping them to a table of these fields. The map function which produces this output might look like this:

```
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. For more information, see Section 9.3.2, "Document Metadata".

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. For more information on the sorting algorithm and sequence, see Section 9.8.5, "Ordering".

  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.

  For more information on querying and extracting information using the key value, see Section 9.8, "Querying Views".

- *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 `SALES-MAN` 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, see Section 9.9.6, "Emitting Multiple Rows".

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.

Figure 9.10. Views — Writing Map Functions with Missing Fields
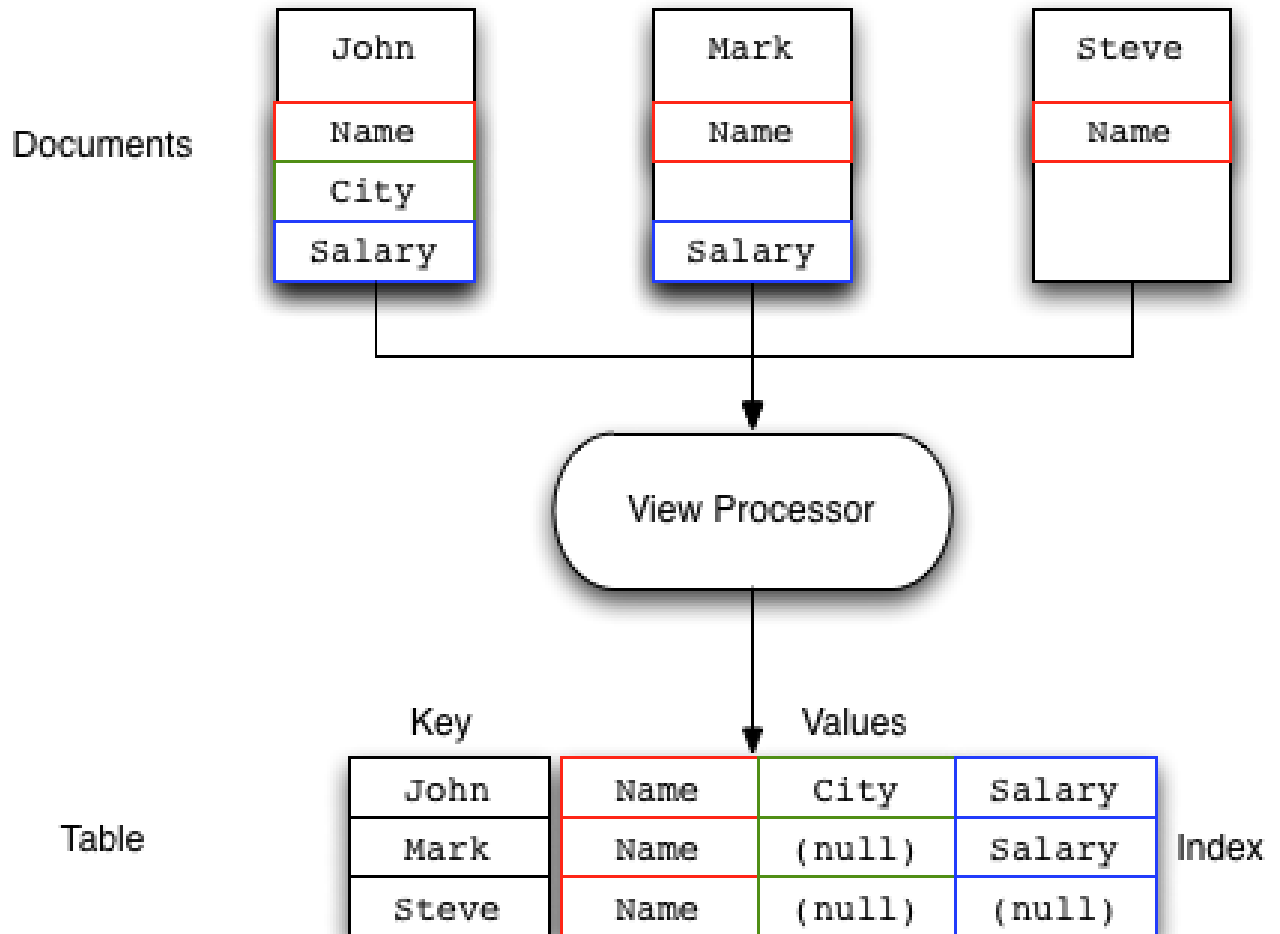
> **Note**
>
> You should check that the field or data source exists during the map processing before emitting the data.

To better understand how the map function works to output different types of information and retrieve it, see Section 9.9, "View and Query Pattern Samples".

9.5.2. 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.

> **Note**
>
> 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. For more information, see Section 9.8.4, "Grouping in Queries".

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 three built-in reduce functions, `_count`, `_sum`, and `_stats`. 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.

> **Note**
>
> 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.

9.5.2.1. 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.

9.5.2.2. 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.

> **Note**
>
> 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"  ] }
    ]
}
```

9.5.2.3. 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.

9.5.2.4. 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.

Figure 9.11. Views — Writing Custom Reduce Functions
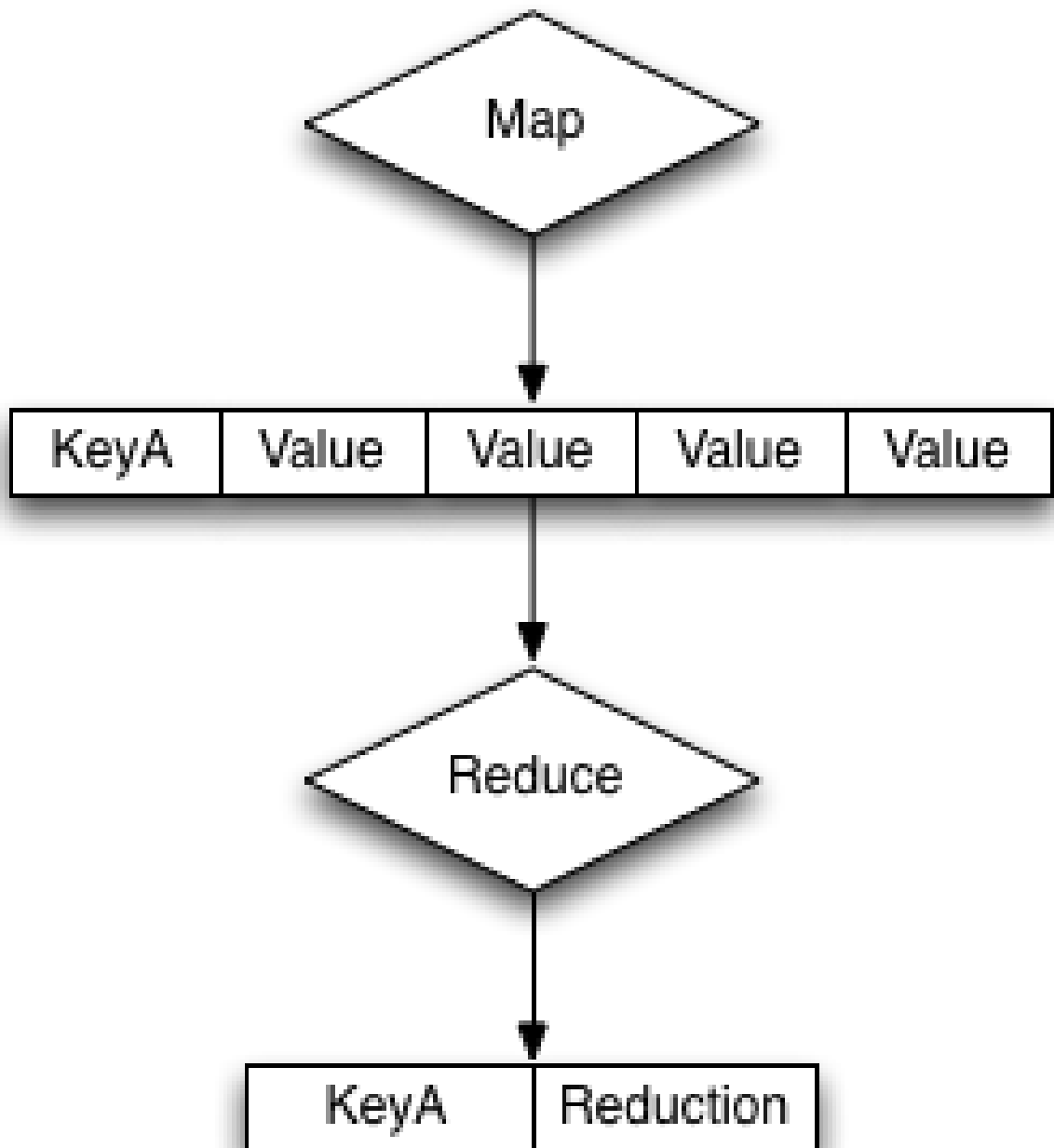
The base format of the `reduce()` function is as follows:

```
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.

9.5.2.5. 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.

9.5.2.6. Handling Rereduce

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.

Figure 9.12. Views — Handling Rereduce
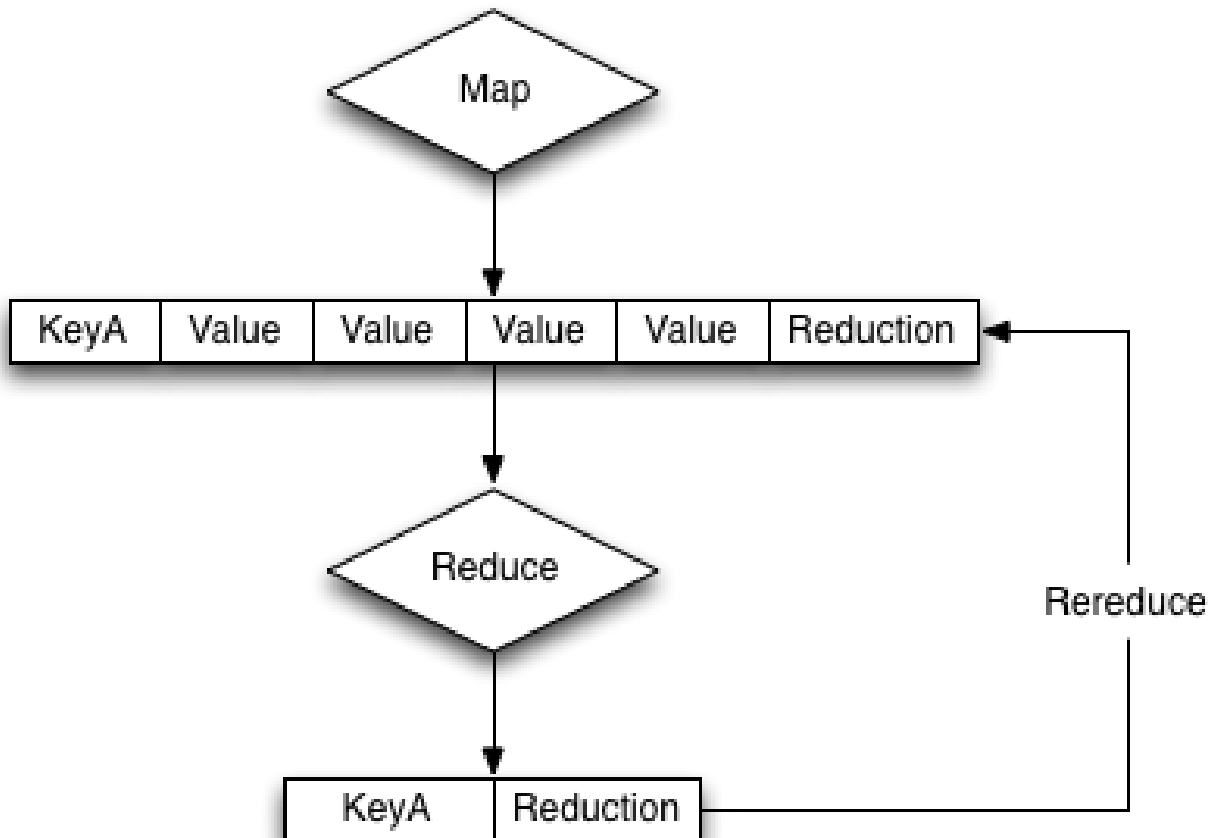
That is, the input of a reduce function can be not only the raw data from the map phase, but also the output of a previous reduce phase. This is called `rereduce`, and can be identified by the third argument to the `reduce()`. When the `rere-`

duce 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.

Note

> 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.

9.5.3. 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:

```
OBJECTTYPE: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']
```

9.5.4. 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. For an example, see Section 9.9.7, "Date and Time Selection".

  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])
```

9.5.5. 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 (see Section 9.2.4, "Index Updates and the `stale` Parameter"). You can also configure different automated view update times for individual design documents, for more information see Section 9.2.5, "Automated Index Updates".

- **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. The 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. See Couchbase SDKs.

  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);
}
```

Warning

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`, `_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.

9.6. 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.

9.7. Design Document REST API

Design documents are used to store one ore more view definitions. Views can be defined within a design document and uploaded to the server through the REST API.

9.7.1. Storing a Design Document

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:

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.

> **Note**
>
> When creating a design document through the REST API it is recommended that you create a development (`dev`) view. It is recommended 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.

> **Note**
>
> 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 designes 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  }
```

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 {\
      }
   }
}
```

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.

9.7.2. Retrieving a Design Document

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.

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.

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:

```
shell&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 QWRtaW5pc3RyYXRvcjpUYW1zaW44=
> 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.

9.7.3. Deleting a Design Document

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:

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.

9.8. 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 sequence and precedence of the different parameters during queries is shown in Figure 9.13, "Views — Querying — Flow and Parameter Interaction".
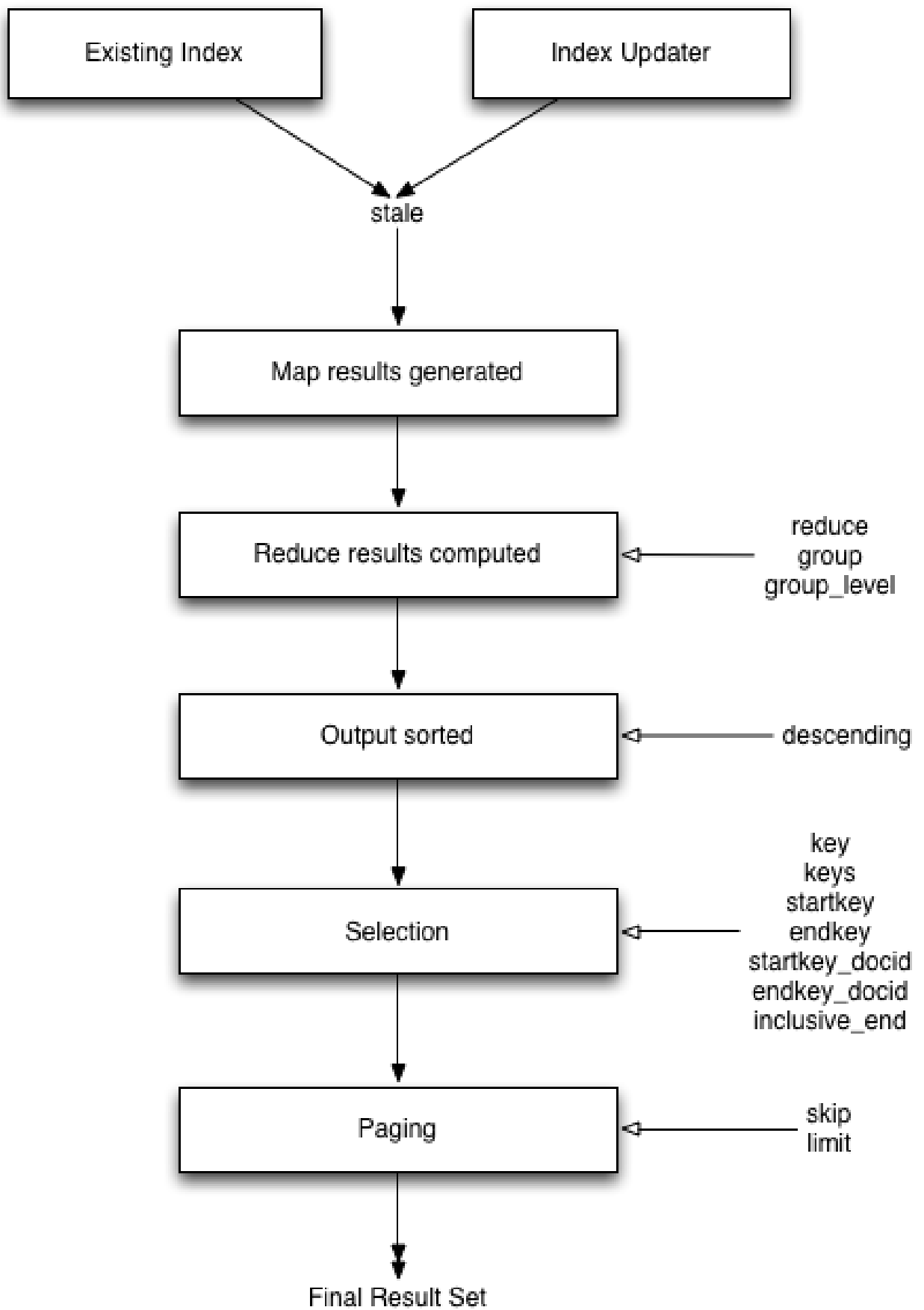
The core arguments and selection systems are the same through both the REST API interface, and the client libraries. The setting of these values differes between different client libraries, but the argument names and expected and supported values are the same across all environments.

9.8.1. 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.

Using the REST API you can 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 (see Section 9.4, "Development and Production Views"), 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 literald or escaped according to the HTTP standard rules.

The additional supported arguments are detailed in the table below.

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
`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
}
```

> **Note**
>
> 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.

Detail on each of the parameters, and specific areas of interaction are described within the additional following sections, which also apply to all client library interfaces.

9.8.2. 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.

> **Important**
>
> 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 literald ("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.

  > **Note**
  >
  > 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. See Section 9.8.2.2, "Partial Selection and Key Ranges"

9.8.2.1. 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. For more information, see Section 9.8.2.3, "Partial Selection with Compound Keys".

9.8.2.2. 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 greather 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.

9.8.2.3. 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.

For more information and examples on formatting and querying this data, see Section 9.9.7, "Date and Time Selection".

9.8.3. 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.

9.8.4. 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. You can see described in Figure 9.14, "View Grouping".

Figure 9.14. View Grouping
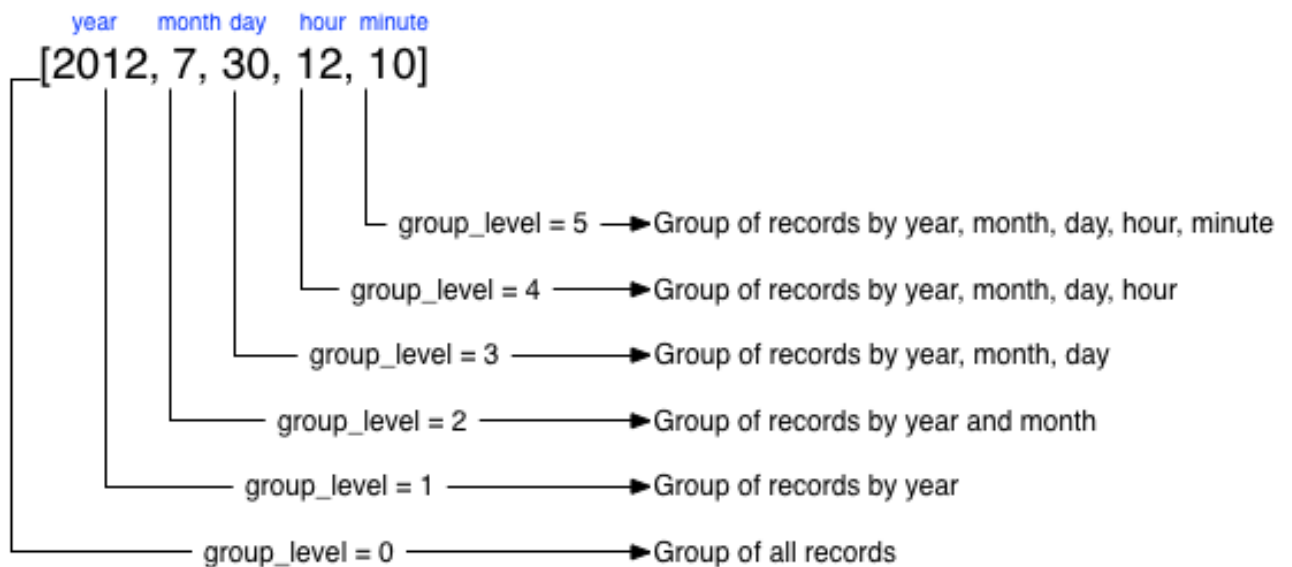
The `group_level` parameter specifies the array index (starting at 1) at which you want the grouping occur, and generate a unique value based on this value that is used to identify all the items in the view output that include this unique value:

- 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.

9.8.4.1. 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.

9.8.5. 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).

Note

> 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.

Tip

> 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.

9.8.6. 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. For more information about ordering results, see Section 9.8.2.2, "Partial Selection and Key Ranges".

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 2.0. For more information, see Section B.2, "Beer Sample Bucket".

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. If you want to learn more about Unicode collation, refer to these resources: Unicode Technical Standard #10 and ICU User Guide, Customization, Default Options.

9.8.7. 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 behaviour 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
        }
    ]
}
```

9.9. 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.

9.9.1. 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.

9.9.2. 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.

9.9.3. 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"
```

Note

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.

9.9.4. 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.

9.9.5. 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.

9.9.6. 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}
]
}
```

9.9.7. 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 convertes 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()` [275].

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}}
]
}
```

> **Note**
>
> 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.

9.9.8. 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.

9.9.9. 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.

9.9.10. 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.

9.9.11. 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.

9.9.12. Simulating Multi-phase Transactions

The technique in Section 9.9.11, "Simulating Transactions" will work 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.

Warning

> 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.

9.10. 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. See Section 9.10.1, "Translating SQL Field Selection (SELECT) to Map/Reduce"

- `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.

  For examples of this in action, see Section 9.9.8, "Selective Record Output".

- `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. For examples of this in action, see Section 9.10.2, "Translating SQL WHERE to Map/Reduce".

- `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.

  For further discussion, see Section 9.10.3, "Translating SQL ORDER BY to Map/Reduce".

- `LIMIT limitcount OFFSET offsetcount`

  There are a number of different paging strategies available within the map/reduce and views mechanism. Discussion on the direct parameters can be seen in Section 9.10.5, "Translating SQL LIMIT and OFFSET". For alternative paging solutions, see Section 9.8.3, "Pagination".

- `GROUP BY groupfield`

Grouping within SQL is handled within views through the use of the `reduce()` function. For comparison examples, see Section 9.10.4, "Translating SQL `GROUP BY` to Map/Reduce".

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.

9.10.1. Translating SQL Field Selection (`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, see Section 9.5.5, "View Writing Best Practice".

> **Note**
>
> 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.

9.10.2. 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
```

9.10.3. 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. For more information, see Section 9.8.5, "Ordering".

9.10.4. 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 in Section 9.5.2.2, "Built-in _sum".

More complex grouping operations may require a custom reduce function. For more information, see Section 9.5.2.4, "Writing Custom Reduce Functions".

9.10.5. 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. See Section 9.8.3, "Pagination" for some further examples of paging strategies.

9.11. Writing Geospatial Views

Not experimental by 2.2 / 2.5?

Warning

Geospatial support was introduced as an *experimental* feature in Couchbase Server 2.0. 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.

9.11.1. 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"
}
```

9.11.2. 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]]}
  }
}
```

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://local-host: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.

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 that 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.

Chapter 10. Monitoring Couchbase

There are a number of different ways in which you can monitor Couchbase. You should be aware however of some of the basic issues that you will need to know before starting your monitoring procedure.

10.1. Underlying Server Processes

There are several different server processes that constantly run in Couchbase Server 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. Prior to 2.1 we had a single process for memcached, Moxi and to monitor all server processes. This resulted in server disruption and crashes due to lack of memory.

  As of Couchbase Server 2.1+ 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**. See Section 7.7, "**cbcollect_info** Tool".

- **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.

10.2. 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.

> **Warning**
>
> 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 MSDN: Avoiding TCP/IP Port Exhaustion.

10.3. Monitoring startup (warmup)

If a Couchbase Server node is starting up for the first time, it will create 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. For more information about server warmup, see Section 5.2, "Handling Server Warmup".

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 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 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
ep_warmup_thread:	complete
ep_warmup_time	1400

Table 10.1. Monitoring — Stats

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

Stat	Description
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

10.4. Disk Write Queue

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 acknowledgement 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.

We have implemented disk writing as a 2-queue system and they 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.

10.4.1. 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.

From the Web UI, click on Monitor Data Buckets and select the particular bucket that you want to monitor. Click "Configure View" in the top right corner and select the "Disk Write Queue" statistic. Closing this window will show 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, you can monitor each node individually using the 'stats' output (see Section 6.2, "Viewing Server Nodes" for more information about gathering node-level stats). There are two statistics to watch here:

ep_queue_size (where new mutations are placed) flusher_todo (the queue of items currently being written to disk)

See The Dispatcher for more information about monitoring what the disk subsystem is doing at any given time.

10.5. Couchbase Server Statistics

Couchbase Server provides statistics at multiple levels throughout the cluster. These are 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.

10.5.1. REST Interface Statistics

The easiest to use interface into the statistics provided by REST is to use the Chapter 6, *Using the 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. See the REST documentation for more information.

10.5.2. 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 shipped with Couchbase Server.

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

10.5.2.1. `stats` Command

This displays a large list of statistics related to the Couchbase process including the underlying engine (ep_* stats).

10.5.2.2. `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.

10.6. Couchbase Server Moxi Statistics

Moxi, as part of it's support of memcached protocol, has support for the memcached stats command. Regular memcached clients can request statistics through the memcached stats command. The stats command accepts optional argu-

ments, and in the case of Moxi, there is a stats proxy sub-command. A detailed description of statistics available through Moxi can be found here.

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
```

Chapter 11. Troubleshooting

When troubleshooting your Couchbase Server deployment there are a number of different approaches available to you. For specific answers to individual problems, see Section 11.4, "Common Errors".

11.1. General Tips

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.

    For more information, see Section 7.7, "**cbcollect_info** Tool".

11.2. Responding to Specific Errors

The following table outlines some specific areas to check when experiencing different problems:

Table 11.1. Troubleshooting — Responses to Specific Errors

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.

Severity	Issue	Suggested Action(s)
		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.

11.3. 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. For a list of the different file locations for each platform, see Table 11.2, "Log File Locations".

Table 11.2. Log File Locations

Platform	Location
Linux	`/opt/couchbase/var/lib/couchbase/logs`
Windows	`C:\Program Files\Couchbase\Server\log` [a]
Mac OS X	`~/Library/Logs`

[a] Assumes default installation location

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.

Table 11.3, "Log File Locations" contains a list of the different log files are create in the logging directory and their contents.

Table 11.3. Log File Locations

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`	XSCR 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.

File	Log Contents
stats	Contains periodic reports of the core statistics.
memcached.log	Contains information relating to the core memcache component, including vBucket and replica and re-balance data streams requests.

> **Note**
>
> Each logfile group will also include a `.idx` and `.siz` file which holds meta information about the logfile group. These files are automatically updated by the logging system.

11.4. Common Errors

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 additional 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
  ```

Appendix A. Uninstalling Couchbase Server

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. For more information on the methods of shutting down your server for your platform, see Section 3.2, "Server Startup and Shutdown".

- If your machine is part of an active cluster, you should rebalance your cluster to take the node out of your configuration. See Section 5.8, "Rebalancing".

- Update your clients to point to an available node within your Couchbase Server cluster.

A.1. Uninstalling on a RedHat Linux System

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

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

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

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.

A.2. Uninstalling on an Debian/Ubuntu Linux System

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

```
shell> 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.

A.3. Uninstalling on a Windows System

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.

A.4. Uninstalling on a Mac OS X System

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.

Appendix B. Couchbase Sample Buckets

Couchbase Server comes with sample buckets that contain both data and MapReduce queries to demonstrate the power and capabilities.

This appendix provides information on the structure, format and contents of the sample databases. The available sample buckets include:

- Game Simulation data
- Beer and Brewery data

B.1. 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.

B.1.1. `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&l
```

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
      }
   ]
}
```

B.1.2. `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"
        }
    ]
}
```

B.2. 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 categorisation 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 reconrd includes basic contact and address information for the brewery, and contains a spatial record consisting of the latitute and longitude of the brewery location.

To demonstrate the view functionality in Couchbase Server, three views are defined.

B.2.1. `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, as detailed in Section 9.9.10, "Solutions for Simulating Joins". 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"
            ]
```

```json
      },
      {
         "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.

B.2.2. `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.

```javascript
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:

```json
{
   "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}
]
}
```

Appendix C. Troubleshooting Views (Technical Background)

A number of errors and problems with views are generally associated with the eventual consistency model of the view system. In this section, some further detail on this information and strategies for identifying and tracking view errors are provided.

It also gives some guidelines about how to report potential view engine issues, what information to include in JIRA.

C.1. Timeout errors in query responses

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
```

C.2. Blocked indexers, 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:9000/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 `>max-ParallelIndexers` is 4:

```
shell> 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>",
```

```machine_data
        "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`.

C.3. 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:

```
shell> 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 k
 numbers"},
{"from":"http://192.168.1.80:9501/_view_merge/?stale=false","reason":"Builtin _stats function requires map values to k
 numbers"},
{"from":"http://192.168.1.80:9503/_view_merge/?stale=false","reason":"Builtin _stats function requires map values to k
 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 make 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/de
 error executing reduce
function for view `view1'
    reason:                 Builtin _stats function requires map values to be
numbers
```

C.4. Wrong documents or rows 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 behaviour? Well, `include_docs=true` works by 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 (http://en.wikipedia.org/wiki/Multiversion_concurrency_control), 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 behaviour as in the given example.

C.5. Expired documents still have their associated Key-Value pairs returned in queries with `stale=false`

See http://www.couchbase.com/issues/browse/MB-6219

C.6. 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:9000/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,
```

```
> curl -s http://localhost:9000/pools/default/buckets |
```

```
                  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 is at http://www.couchbase.com/issues/browse/MB-5534. In this case Tommie was able to identify that the problem was in ep-engine.

C.7. Index filesystem structure and meaning

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 (see Section C.6, "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
```

C.8. 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 (see Section C.7, "Index filesystem structure and meaning"). This means that both point to the same in-

dex 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:

```
shell> 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"
   }
]
```

C.9. Getting query results from 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. Here follows an example where we query 2 different nodes from a 4 nodes 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"}
]}
shell> 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`.

C.10. Verifying replica index and querying it (debug/testing)

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.

To find out which replica vbuckets exist in each node, see section Section C.6, "Data missing in query response or it's wrong (potentially due to server issues)".

C.11. Expected cases for `total_rows` with a too high value

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.

C.12. Getting view btree stats for performance and longevity analysis

As of 2.0 build 1667, there is a special (non-public) URI to get statistics for all the btrees of an index (design document). These statistics are developer oriented and are useful for analyzing performance and longevity issues. Example:

```
> curl -s 'http://localhost:9500/_set_view/default/_design/test3/_btree_stats' | python -mjson.tool
{
    "id_btree": {
        "avg_elements_per_kp_node": 19.93181818181818,
        "avg_elements_per_kv_node": 75.00750075007501,
        "avg_kp_node_size": 3170.159090909091,
        "avg_kp_node_size_compressed": 454.0511363636364,
        "avg_kv_node_size": 2101.2100210021,
        "avg_kv_node_size_compressed": 884.929492949295,
        "btree_size": 3058201,
        "chunk_threshold": 5120,
        "file_size": 11866307,
        "fragmentation": 74.22786213098988,
        "kp_nodes": 176,
        "kv_count": 250000,
        "kv_nodes": 3333,
        "max_depth": 4,
        "max_elements_per_kp_node": 27,
        "max_elements_per_kv_node": 100,
        "max_kp_node_size": 4294,
        "max_kp_node_size_compressed": 619,
        "max_kv_node_size":,
        "max_kv_node_size_compressed": 1161,
        "max_reduction_size": 133,
        "min_depth": 4,
        "min_elements_per_kp_node": 8,
        "min_elements_per_kv_node": 75,
        "min_kp_node_size":,
        "min_kp_node_size_compressed": 206,
        "min_kv_node_size": 2101,
        "min_kv_node_size_compressed": 849,
```

```
            "min_reduction_size": 133
        },
        "view1": {
            "avg_elements_per_kp_node": 17.96416938110749,
            "avg_elements_per_kv_node": 23.99923202457521,
            "avg_kp_node_size": 3127.825732899023,
            "avg_kp_node_size_compressed": 498.3436482084691,
            "avg_kv_node_size": 3024.903235096477,
            "avg_kv_node_size_compressed": 805.7447441681866,
            "btree_size": 8789820,
            "chunk_threshold": 5120,
            "file_size": 11866307,
            "fragmentation": 25.92623804524862,
            "kp_nodes": 614,
            "kv_count": 250000,
            "kv_nodes": 10417,
            "max_depth": 5,
            "max_elements_per_kp_node": 21,
            "max_elements_per_kv_node": 24,
            "max_kp_node_size": 3676,
            "max_kp_node_size_compressed": 606,
            "max_kv_node_size": 3025,
            "max_kv_node_size_compressed": 852,
            "max_reduction_size": 141,
            "min_depth": 5,
            "min_elements_per_kp_node": 2,
            "min_elements_per_kv_node": 16,
            "min_kp_node_size": 357,
            "min_kp_node_size_compressed": 108,
            "min_kv_node_size": 2017,
            "min_kv_node_size_compressed": 577,
            "min_reduction_size": 137
        }
}
```

Note that these statistics are per node, therefore for performance and longevity analysis, you should query this URI for all nodes in the cluster. Getting these statistics can take from several seconds to several minutes, depending on the size of the dataset (it needs to traverse the entire btrees in order to compute the statistics).

C.13. 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. An example issue where this happened is MB-7161.

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 (see Section C.6, "Data missing in query response or it's wrong (potentially due to server issues)" 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.

C.14. What to include in good issue reports (JIRA)

When reporting issues to Couchbase (using couchbase.com/issues), 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_tasksAPI 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_tasksAPI contains information per node, so you'll have to query_active_tasksor every node in the cluster to verify if progress is stuck, etc.

Appendix D. Release Notes

The following sections provide release notes for individual release versions of Couchbase Server. To browse or submit new issues, see Couchbase Server Issues Tracker.

D.1. Release Notes for Couchbase Server 2.1.1 GA (July 2013)

Couchbase Server 2.1.1 is first maintenance release for Couchbase Server 2.1. This release includes some major bug fixes and enhancements:

New Edition in 2.1.1

The Enterprise Edition of Couchbase Server is now available on Mac OSX. See Couchbase, Downloads.

Fixes in 2.1.1

- **Database Operations**

  - There was an underlying Windows Management Instrumentation issue in `wmi_port.cpp` which caused memory leaks. This has been fixed.

    Issues: MB-8674

  - The 2.1 version of the server exposes fewer server stats than it did in earlier versions. The five stats that have been removed are `key_data_age`, `key_last_modification_time`, `paged_out _time`, `ep_too_young` and `ep_too_old`.

    Issues: MB-8539

- **Cluster Operations**

  - The rebalance speed for small datasets has been significantly improved. This includes time to rebalance empty buckets and buckets containing tens of thousands of items.

    Issues: MB-8521

  - In Couchbase 2.1.0 if you tried to assign a hostname to a node when you join the node to a cluster, it will be reset. The hostname will not be saved for the node and will not be used by the cluster to identify the node. This has been fixed. For more information about managing hostnames, see Section 2.4, "Using Hostnames with Couchbase Server".

    Issues: MB-8545

Known Issues in 2.1.1

- **Installation and Upgrade**

  - The 2.1.1 READMEs for the Server contain the wrong version number and are labeled 2.1.0 instead of 2.1.1. Ignore this oversight.

D.2. Release Notes for Couchbase Server 2.1.0 GA (June 2013)

Couchbase Server 2.1.0 is the first minor release for Couchbase Server 2.0 and includes several optimizations, new features and important bug fixes.

The **major enhancements** available in Couchbase Server 2.1.0 include:

- Improved disk read and write performance with new multi-threaded persistence engine. With the newly designed persistence engine, users can configure the multiple read-write workers on a per Bucket basis. See Section 1.2.7, "Disk Storage".

- Optimistic Replication mode for XDCR to optimistically replicate mutations to the target cluster. More information can be found here: Section 5.9.8.2, "'Optimistic Replication' in XDCR"

- More XDCR Statistics to monitor performance and behavior of XDCR. See Section 6.4.1.6, "Monitoring Outgoing XDCR".

- Support for hostnames when setting up Couchbase Server. See Section 2.4, "Using Hostnames with Couchbase Server".

- Rebalance progress indicator to provide more visibility into rebalance operation. See Section 5.8.4, "Monitoring a Rebalance".

- Ability to generate a health report with `cbhealthchecker`, now included as part of the Couchbase Server install. See Section 7.11, "**cbhealthchecker** Tool".

- Importing and exporting of CSV files with Couchbase using **cbtransfer**. See Section 7.10, "**cbtransfer** Tool".

- Server-side replica-read API. Couchbase Developer Guide, Replica Read.

Additional behavior changes in 2.1.0 include:

- Backup, Restore and Transfer tool to optionally transfer data or design documents only. The default is to transfer both data and design documents. See Section 7.8, "**cbbackup** Tool", Section 7.9, "**cbrestore** Tool", Section 7.10, "**cbtransfer** Tool"

- Improved cluster manager stability via separate process for cluster manager. See Section 10.1, "Underlying Server Processes".

- Command Line tools updated so you can manage nodes, buckets, clusters and XDCR. See Section 7.4, "**couchbase-cli** Tool"

- More XDCR Statistics to monitor performance and behavior of XDCR. See Section 6.4.1.6, "Monitoring Outgoing XDCR".

- Command Line tools updated so you can manage nodes, buckets, clusters and XDCR. See Section 7.4, "**couchbase-cli** Tool".

- Several new and updated statistics for XDCR on the admin Console and via the REST-API. For more information, see Section 6.4.1.7, "Monitoring Incoming XDCR", Section 6.4.1.6, "Monitoring Outgoing XDCR", and Section 8.9.8, "Getting XDCR Stats via REST".

Fixes in 2.1.0

- **Installation and Upgrade**

  - In the past Couchbase Server 2.0.0 upgrade installers on Linux did not replace the `file2.beam` with the latest version. This will cause indexing and querying to fail. This has been fixed.

    Issues: MB-7770

  - The Windows installer for Windows 32-bit and 64-bit now prompts you to set the MaxUserPort registry setting. This will increase the number of ephemeral ports available to applications on Windows, as documented in Microsoft Knowledge Base Article 196271. The installer also warns you that a reboot is necessary for this change to take effect. If this registry key is not set, it may lead to port exhaustion leading to various problems, see as MB-8321. For installer instructions, see Section 2.2.3, "Microsoft Windows Installation".

    Issues: MB-8321

- **Cluster Operations**

  - Previously, there was only one process that was responsible for monitoring and managing all the other underlying server processes. This includes Moxi and memcached, and also statistics gathering. Now there are two processes. One is responsible for just Moxi/Memcached and the other is responsible for monitoring all other processes. This should help prevent the max_restart_intensity seen when timeouts start and temporarily disrupted the server. The most noticeable change you see with this fix is that there are now two beam.smp processes running on Linux and two erl.exe running on Windows. For more details, see Section 10.1, "Underlying Server Processes".

    Issues: MB-8376

- **Command-line Tools**

  - For earlier versions of Couchbase Server, some internal server directories were accessible all users, which was a security issue. This is now fixed. The fix now means that you should have root privileges when you run **cbcollect_info** because this tool needs this access level to collect all the information it needs to collect about the server. For more information about **cbcollect_info**, see Section 7.7, "**cbcollect_info** Tool".

  - One XDCR REST API endpoint had a typo which is now fixed. The old endpoint was `/con-troller/cancelXCDR/:xid`. The new, correct endpoint is `/controller/cancelXDCR/:xid`. See Section 8.9.5, "Deleting XDCR Replications".

    Issues: MB-8347

  - In the past when you used **cbworkloadgen** you see this error `ImportError: No module named _sqlite3`. This has been fixed.

    Issues: MB-8153

- **Indexing and Querying**

  - In the past too many simultaneous views requests could overwhelm a node. You can now limit the number of simultaneous requests a node can receive. For more information, see REST-API, see Section 8.8.1, "Limiting Simultaneous Node Requests".

    Issues: MB-8199

- **Cross Datacenter Replication (XDCR)**

  - When you create a replication between two clusters, you may see two error messages: "Failed to grab remote bucket info, vbucket" and "Error replicating vbucket X". Nonetheless, replication will still start and then function as expected, but the error messages may appear for some time in the Web Console. This has been fixed.

    Issues: MB-7786, MB-7457

Known Issues in 2.1.0

- **Installation and Upgrade**

  - On Ubuntu if you upgrade from Couchbase Server 2.0 to 2.1.0 and use a non-default port, the upgrade can fail and return the message 'Failed to stop couchbase-server.' We recommend you use the default ports on both 2.0 and 2.1.0 when you perform an upgrade.

    Issues: MB-8051

  - You may have a installation of Couchbase Server with a custom data path. If you perform a server uninstall and then upgrade to 2.1.0 with the same custom data path, some older XDCR replication files may be left intact. This will re-

sult in server crashes and incorrect information in Web Console. The workaround for this case is to make sure you manually delete the `_replicator.couch.1` file from the server data directory before you install the new version of the server. Alternately you can delete the entire data directory before you install the new version of the server. If you choose this workaround, you may want to backup your data before you delete the entire directory. By default, on Linux this directory is at `/opt/couchbase/var/lib/couchbase/data/` and on Windows at `C:/Program Files/Couchbase/Server/var/lib/couchbase/data/`.

Issues: MB-8460

- **Database Operations**

  - On Windows platforms, statistics from the server will incorrectly show a very high swap usage rate. This will be resolved in future releases of Couchbase Server.

    Issues: MB-8461

  - Any non-UTF-8 keys are not filtered or logged by Couchbase Server. Future releases will address this issue.

    Issues: MB-8427

  - If you edit a data bucket using the REST-API and you do not provide existing values for bucket properties, the server may reset existing bucket properties to the default value. To avoid this situation you should specify all existing bucket properties as well as the updated properties as parameters when you make this REST request. For more information, see Couchbase Manual, Creating and Editing Data Buckets.

    Issues: MB-7897

- **Cluster Operations**

  - If you query a view during cluster rebalance it will fail and return the messages "error Reason: A view spec can not consist of merges exclusively" and then "no_active_vbuckets Reason: Cannot execute view query since the node has no active vbuckets." The workaround for this situation is to handle this error and retry later in your code. Alternatively the latest version of the Java SDK will automatically retry upon these errors.

    Issues: MB-7661

  - If you perform asynchronous operations from a client to Couchbase on Windows, you may receive unclear status messages due to character encoding issues on Windows. This impacts status messages in the response only; actual response data is unaffected by this problem. This will be resolved in future releases of Couchbase Server.

    Issues: MB-8149

  - By default most Linux systems have swappiness set to 60. This will lead to overuse of disk swap with Couchbase Server. Please follow our current recommendations on swappiness and swap space, see Section 4.6.4, "Swap Space".

    Issues: MB-7737, MB-7774

  - A cluster rebalance may exit and produce the error {not_all_nodes_are_ready_yet} if you perform the rebalance right after failing over a node in the cluster. You may need to wait 60 seconds after the node failover before you attempt the cluster rebalance.

    This is because the failover REST API is a synchronous operation with a timeout. If it fails to complete the failover process by the timeout, the operation internally switches into a asynchronous operation. It will immediately return and re-attempt failover in the background which will cause rebalance to fail since the failover operation is still running.

    Issues: MB-7168

- Stats call via Moxi are currently enabled, however if you use the command-line tool **cbstats** through Moxi on port 11211 to Couchbase Server, you will receive incorrect server statistics. To avoid this issue you should use port 11210 when you make a **cbstats** request.

  Issues: MB-7678

- If you perform a scripted rolling upgrade from 1.8.1 to 2.1.0 nodes, you should script a delay of 10 seconds after you add a node and before you request rebalance. Without this delay, you may request rebalance too soon and rebalance may fail and produce an error.

  Issues: MB-8094

- **Command-line Tools**

  - **cbbackup** will backup even deleted items from a cluster which are not needed. In future releases it will ignore these items and not back them up.

    Issues: MB-8377

  - **cbstats** will not work on port 11210 if you installed Couchbase Server without root-level permissions.

    Issues: MB-7878

- **Indexing and Querying**

  - If you rebalance a cluster, index compaction starts but bucket compaction will not start. This is because vBucket cleanup from an index is not accounted for in index fragmentation. After this cleanup process completes it will cause more index fragmentation and therefore we run compaction again. Index compaction will therefore always run after a certain number of changes to vBuckets on nodes. You can change this setting using the REST-API, see Section 8.7.7, "Adjusting Rebalance during Compaction".

    Issues: MB-8319

- **Performance**

  - RHEL6 and other newer Linux distributions running on physical hardware are known to have transparent hugepages feature enabled. In general this can provide a measurable performance boost. However under some conditions that Couchbase Server is known to trigger, this may cause severe delays in page allocations. Therefore we strongly recommend you disable this feature with Couchbase Server.

    Issues: MB-8456

Appendix E. Limits

Couchbase Server has a number of limits and limitations that may affect your use of Couchbase Server.

Table E.1. Couchbase Server Limits

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

Appendix F. Licenses

This documentation and associated software is subject to the following licenses.

F.1. Documentation License

This documentation in any form, software or printed matter, contains proprietary information that is the exclusive property of Couchbase. Your access to and use of this material is subject to the terms and conditions of your Couchbase Software License and Service Agreement, which has been executed and with which you agree to comply. This document and information contained herein may not be disclosed, copied, reproduced, or distributed to anyone outside Couchbase without prior written consent of Couchbase or as specifically provided below. This document is not part of your license agreement nor can it be incorporated into any contractual agreement with Couchbase or its subsidiaries or affiliates.

Use of this documentation is subject to the following terms:

You may create a printed copy of this documentation solely for your own personal use. Conversion to other formats is allowed as long as the actual content is not altered or edited in any way. You shall not publish or distribute this documentation in any form or on any media, except if you distribute the documentation in a manner similar to how Couchbase disseminates it (that is, electronically for download on a Web site with the software) or on a CD-ROM or similar medium, provided however that the documentation is disseminated together with the software on the same medium. Any other use, such as any dissemination of printed copies or use of this documentation, in whole or in part, in another publication, requires the prior written consent from an authorized representative of Couchbase. Couchbase and/or its affiliates reserve any and all rights to this documentation not expressly granted above.

This documentation may provide access to or information on content, products, and services from third parties. Couchbase Inc. and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Couchbase Inc. and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

F.2. Couchbase, Inc. Community Edition License Agreement

IMPORTANT-READ CAREFULLY: BY CLICKING THE "I ACCEPT" BOX OR INSTALLING, DOWNLOADING OR OTHERWISE USING THIS SOFTWARE AND ANY ASSOCIATED DOCUMENTATION, YOU, ON BEHALF OF YOURSELF OR AS AN AUTHORIZED REPRESENTATIVE ON BEHALF OF AN ENTITY ("LICENSEE") AGREE TO ALL THE TERMS OF THIS COMMUNITY EDITION LICENSE AGREEMENT (THE "AGREEMENT") REGARDING YOUR USE OF THE SOFTWARE. YOU REPRESENT AND WARRANT THAT YOU HAVE FULL LEGAL AUTHORITY TO BIND THE LICENSEE TO THIS AGREEMENT. IF YOU DO NOT AGREE WITH ALL OF THESE TERMS, DO NOT SELECT THE "I ACCEPT" BOX AND DO NOT INSTALL, DOWNLOAD OR OTHERWISE USE THE SOFTWARE. THE EFFECTIVE DATE OF THIS AGREEMENT IS THE DATE ON WHICH YOU CLICK "I ACCEPT" OR OTHERWISE INSTALL, DOWNLOAD OR USE THE SOFTWARE.

1. License Grant. Couchbase Inc. hereby grants Licensee, free of charge, the non-exclusive right to use, copy, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to Licensee including the following copyright notice in all copies or substantial portions of the Software:

```
Couchbase ©
http://www.couchbase.com
Copyright 2011 Couchbase, Inc.
```

As used in this Agreement, "Software" means the object code version of the applicable elastic data management server software provided by Couchbase, Inc.

2. Support. Couchbase, Inc. will provide Licensee with access to, and use of, the Couchbase, Inc. support forum available at the following URL: http://forums.membase.org. Couchbase, Inc. may, at its discretion, modify, suspend or terminate support at any time upon notice to Licensee.

3. Warranty Disclaimer and Limitation of Liability. THE SOFTWARE IS PROVIDED "AS IS," WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL COUCHBASE INC. OR THE AUTHORS OR COPYRIGHT HOLDERS IN THE SOFTWARE BE LIABLE FOR ANY CLAIM, DAMAGES (INCLUDING, WITHOUT LIMITATION, DIRECT, INDIRECT OR CONSEQUENTIAL DAMAGES) OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

F.3. Couchbase, Inc. Enterprise License Agreement: Free Edition

IMPORTANT-READ CAREFULLY: BY CLICKING THE "I ACCEPT" BOX OR INSTALLING, DOWNLOADING OR OTHERWISE USING THIS SOFTWARE AND ANY ASSOCIATED DOCUMENTATION, YOU, ON BEHALF OF YOURSELF OR AS AN AUTHORIZED REPRESENTATIVE ON BEHALF OF AN ENTITY ("LICENSEE") AGREE TO ALL THE TERMS OF THIS ENTERPRISE LICENSE AGREEMENT – FREE EDITION (THE "AGREEMENT") REGARDING YOUR USE OF THE SOFTWARE. YOU REPRESENT AND WARRANT THAT YOU HAVE FULL LEGAL AUTHORITY TO BIND THE LICENSEE TO THIS AGREEMENT. IF YOU DO NOT AGREE WITH ALL OF THESE TERMS, DO NOT SELECT THE "I ACCEPT" BOX AND DO NOT INSTALL, DOWNLOAD OR OTHERWISE USE THE SOFTWARE. THE EFFECTIVE DATE OF THIS AGREEMENT IS THE DATE ON WHICH YOU CLICK "I ACCEPT" OR OTHERWISE INSTALL, DOWNLOAD OR USE THE SOFTWARE.

1. **License Grant**. Subject to Licensee's compliance with the terms and conditions of this Agreement, Couchbase Inc. hereby grants to Licensee a perpetual, non-exclusive, non-transferable, non-sublicensable, royalty-free, limited license to install and use the Software only for Licensee's own internal production use on up to two (2) Licensed Servers or for Licensee's own internal non-production use for the purpose of evaluation and/or development on an unlimited number of Licensed Servers.

2. **Restrictions**. Licensee will not: (a) copy or use the Software in any manner except as expressly permitted in this Agreement; (b) use or deploy the Software on any server in excess of the Licensed Servers for which Licensee has paid the applicable Subscription Fee unless it is covered by a valid license; (c) transfer, sell, rent, lease, lend, distribute, or sublicense the Software to any third party; (d) use the Software for providing time-sharing services, service bureau services or as part of an application services provider or as a service offering primarily designed to offer the functionality of the Software; (e) reverse engineer, disassemble, or decompile the Software (except to the extent such restrictions are prohibited by law); (f) alter, modify, enhance or prepare any derivative work from or of the Software; (g) alter or remove any proprietary notices in the Software; (h) make available to any third party the functionality of the Software or any license keys used in connection with the Software; (i) publically display or communicate the results of internal performance testing or other benchmarking or performance evaluation of the Software; or (j) export the Software in violation of U.S. Department of Commerce export administration rules or any other export laws or regulations.

3. **Proprietary Rights**. The Software, and any modifications or derivatives thereto, is and shall remain the sole property of Couchbase Inc. and its licensors, and, except for the license rights granted herein, Couchbase Inc. and its licensors retain all right, title and interest in and to the Software, including all intellectual property rights therein and thereto. The Software may include third party open source software components. If Licensee is the United States Government or any contractor thereof, all licenses granted hereunder are subject to the following: (a) for acquisition by or on behalf of civil agencies, as necessary to obtain protection as "commercial computer software" and related documentation in accordance with the terms of this Agreement and as specified in Subpart 12.1212 of the Federal Acquisition Regulation (FAR), 48 C.F.R.12.1212, and its successors; and (b) for acquisition by or on behalf of the Department of Defense (DOD) and any agencies or units thereof, as necessary to obtain protection as "commercial computer software" and related documentation in accordance with the terms of this Agreement and as specified in Subparts 227.7202-1 and 227.7202-3 of the DOD FAR Supplement, 48 C.F.R.227.7202-1 and 227.7202-3, and its successors. Manufacturer is Couchbase, Inc.

4. **Support**. Couchbase Inc. will provide Licensee with: (a) periodic Software updates to correct known bugs and errors to the extent Couchbase Inc. incorporates such corrections into the free edition version of the Software; and (b) access to, and use of, the Couchbase Inc. support forum available at the following URL: http://forums.membase.org. Licensee must have Licensed Servers at the same level of Support Services for all instances in a production deployment running the Software. Licensee must also have Licensed Servers at the same level of Support Services for all instances in a development and test environment running the Software, although these Support Services may be at a different level than the production Licensed Servers. Couchbase Inc. may, at its discretion, modify, suspend or terminate support at any time upon notice to Licensee.

5. **Records Retention and Audit**. Licensee shall maintain complete and accurate records to permit Couchbase Inc. to verify the number of Licensed Servers used by Licensee hereunder. Upon Couchbase Inc.'s written request, Licensee shall: (a) provide Couchbase Inc. with such records within ten (10) days; and (b) will furnish Couchbase Inc. with a certification signed by an officer of Licensee verifying that the Software is being used pursuant to the terms of this Agreement. Upon at least thirty (30) days prior written notice, Couchbase Inc. may audit Licensee's use of the Software to ensure that Licensee is in compliance with the terms of this Agreement. Any such audit will be conducted during regular business hours at Licensee's facilities and will not unreasonably interfere with Licensee's business activities. Licensee will provide Couchbase Inc. with access to the relevant Licensee records and facilities. If an audit reveals that Licensee has used the Software in excess of the authorized Licensed Servers, then (i) Couchbase Inc. will invoice Licensee, and Licensee will promptly pay Couchbase Inc., the applicable licensing fees for such excessive use of the Software, which fees will be based on Couchbase Inc.'s price list in effect at the time the audit is completed; and (ii) Licensee will pay Couchbase Inc.'s reasonable costs of conducting the audit.

6. **Confidentiality**. Licensee and Couchbase Inc. will maintain the confidentiality of Confidential Information. The receiving party of any Confidential Information of the other party agrees not to use such Confidential Information for any purpose except as necessary to fulfill its obligations and exercise its rights under this Agreement. The receiving party shall protect the secrecy of and prevent disclosure and unauthorized use of the disclosing party's Confidential Information using the same degree of care that it takes to protect its own confidential information and in no event shall use less than reasonable care. The terms of this Confidentiality section shall survive termination of this Agreement. Upon termination or expiration of this Agreement, the receiving party will, at the disclosing party's option, promptly return or destroy (and provide written certification of such destruction) the disclosing party's Confidential Information.

7. **Disclaimer of Warranty**. THE SOFTWARE AND ANY SERVICES PROVIDED HEREUNDER ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND. COUCHBASE INC. DOES NOT WARRANT THAT THE SOFTWARE OR THE SERVICES PROVIDED HEREUNDER WILL MEET LICENSEE'S REQUIREMENTS, THAT THE SOFTWARE WILL OPERATE IN THE COMBINATIONS LICENSEE MAY SELECT FOR USE, THAT THE OPERATION OF THE SOFTWARE WILL BE ERROR-FREE OR UNINTERRUPTED OR THAT ALL SOFTWARE ERRORS WILL BE CORRECTED. COUCHBASE INC. HEREBY DISCLAIMS ALL WARRANTIES, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, TITLE, AND ANY WARRANTIES ARISING OUT OF COURSE OF DEALING, USAGE OR TRADE.

8. **Agreement Term and Termination**. The term of this Agreement shall begin on the Effective Date and will continue until terminated by the parties. Licensee may terminate this Agreement for any reason, or for no reason, by providing at least ten (10) days prior written notice to Couchbase Inc. Couchbase Inc. may terminate this Agreement if Licensee materially breaches its obligations hereunder and, where such breach is curable, such breach remains uncured for ten (10) days following written notice of the breach. Upon termination of this Agreement, Licensee will, at Couchbase Inc.'s option, promptly return or destroy (and provide written certification of such destruction) the applicable Software and all copies and portions thereof, in all forms and types of media. The following sections will survive termination or expiration of this Agreement: Sections 2, 3, 6, 7, 8, 9, 10 and 11.

9. **Limitation of Liability**. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, IN NO EVENT WILL COUCHBASE INC. OR ITS LICENSORS BE LIABLE TO LICENSEE OR TO ANY THIRD PARTY FOR ANY INDIRECT, SPECIAL, INCIDENTAL, CONSEQUENTIAL OR EXEMPLARY DAMAGES OR FOR THE COST OF PROCURING SUBSTITUTE PRODUCTS OR SERVICES ARISING OUT OF OR IN ANY WAY RELATING TO OR IN CONNECTION WITH THIS AGREEMENT OR THE USE OF OR INABILITY TO USE

THE SOFTWARE OR DOCUMENTATION OR THE SERVICES PROVIDED BY COUCHBASE INC. HERE-UNDER INCLUDING, WITHOUT LIMITATION, DAMAGES OR OTHER LOSSES FOR LOSS OF USE, LOSS OF BUSINESS, LOSS OF GOODWILL, WORK STOPPAGE, LOST PROFITS, LOSS OF DATA, COMPUTER FAILURE OR ANY AND ALL OTHER COMMERCIAL DAMAGES OR LOSSES EVEN IF ADVISED OF THE POSSIBILITY THEREOF AND REGARDLESS OF THE LEGAL OR EQUITABLE THEORY (CONTRACT, TORT OR OTHERWISE) UPON WHICH THE CLAIM IS BASED. IN NO EVENT WILL COUCHBASE INC.'S OR ITS LICENSORS' AGGREGATE LIABILITY TO LICENSEE, FROM ALL CAUSES OF ACTION AND UNDER ALL THEORIES OF LIABILITY, EXCEED ONE THOUSAND DOLLARS (US $1,000). The parties expressly acknowl-edge and agree that Couchbase Inc. has set its prices and entered into this Agreement in reliance upon the limitations of liability specified herein, which allocate the risk between Couchbase Inc. and Licensee and form a basis of the bargain between the parties.

10. **General**. Couchbase Inc. shall not be liable for any delay or failure in performance due to causes beyond its reasonable control. Neither party will, without the other party's prior written consent, make any news release, public announce-ment, denial or confirmation of this Agreement, its value, or its terms and conditions, or in any manner advertise or publish the fact of this Agreement. Notwithstanding the above, Couchbase Inc. may use Licensee's name and logo, con-sistent with Licensee's trademark policies, on customer lists so long as such use in no way promotes either endorsement or approval of Couchbase Inc. or any Couchbase Inc. products or services. Licensee may not assign this Agreement, in whole or in part, by operation of law or otherwise, without Couchbase Inc.'s prior written consent. Any attempt to as-sign this Agreement, without such consent, will be null and of no effect. Subject to the foregoing, this Agreement will bind and inure to the benefit of each party's successors and permitted assigns. If for any reason a court of competent ju-risdiction finds any provision of this Agreement invalid or unenforceable, that provision of the Agreement will be en-forced to the maximum extent permissible and the other provisions of this Agreement will remain in full force and ef-fect. The failure by either party to enforce any provision of this Agreement will not constitute a waiver of future en-forcement of that or any other provision. All waivers must be in writing and signed by both parties. All notices per-mitted or required under this Agreement shall be in writing and shall be delivered in person, by confirmed facsimile, overnight courier service or mailed by first class, registered or certified mail, postage prepaid, to the address of the par-ty specified above or such other address as either party may specify in writing. Such notice shall be deemed to have been given upon receipt. This Agreement shall be governed by the laws of the State of California, U.S.A., excluding its conflicts of law rules. The parties expressly agree that the UN Convention for the International Sale of Goods (CISG) will not apply. Any legal action or proceeding arising under this Agreement will be brought exclusively in the federal or state courts located in the Northern District of California and the parties hereby irrevocably consent to the person-al jurisdiction and venue therein. Any amendment or modification to the Agreement must be in writing signed by both parties. This Agreement constitutes the entire agreement and supersedes all prior or contemporaneous oral or written agreements regarding the subject matter hereof. To the extent there is a conflict between this Agreement and the terms of any "shrinkwrap" or "clickwrap" license included in any package, media, or electronic version of Couchbase Inc.-furnished software, the terms and conditions of this Agreement will control. Each of the parties has caused this Agree-ment to be executed by its duly authorized representatives as of the Effective Date. Except as expressly set forth in this Agreement, the exercise by either party of any of its remedies under this Agreement will be without prejudice to its other remedies under this Agreement or otherwise. The parties to this Agreement are independent contractors and this Agreement will not establish any relationship of partnership, joint venture, employment, franchise, or agency between the parties. Neither party will have the power to bind the other or incur obligations on the other's behalf without the other's prior written consent.

11. **Definitions**. Capitalized terms used herein shall have the following definitions: "Confidential Information" means any proprietary information received by the other party during, or prior to entering into, this Agreement that a party should know is confidential or proprietary based on the circumstances surrounding the disclosure including, without limitation, the Software and any non-public technical and business information. Confidential Information does not include infor-mation that (a) is or becomes generally known to the public through no fault of or breach of this Agreement by the re-ceiving party; (b) is rightfully known by the receiving party at the time of disclosure without an obligation of confiden-tiality; (c) is independently developed by the receiving party without use of the disclosing party's Confidential Infor-mation; or (d) the receiving party rightfully obtains from a third party without restriction on use or disclosure. "Docu-mentation" means any technical user guides or manuals provided by Couchbase Inc. related to the Software. "Licensed Server" means an instance of the Software running on one (1) operating system. Each operating system instance may be

running directly on physical hardware, in a virtual machine, or on a cloud server. "Couchbase" means Couchbase, Inc. "Couchbase Website" means www.couchbase.com. "Software" means the object code version of the applicable elastic data management server software provided by Couchbase Inc. and ordered by Licensee during the ordering process on the Couchbase Website.

If you have any questions regarding this Agreement, please contact sales@couchbase.com.