Couchbase
Learning Services

CS300 Couchbase NoSQL Server Administration

1

CS300 Couchbase NoSQL Server Administration

July 26th , 2016

Couchbase

Learning Services

Agenda and Course Organization

Technical Agenda – Overview

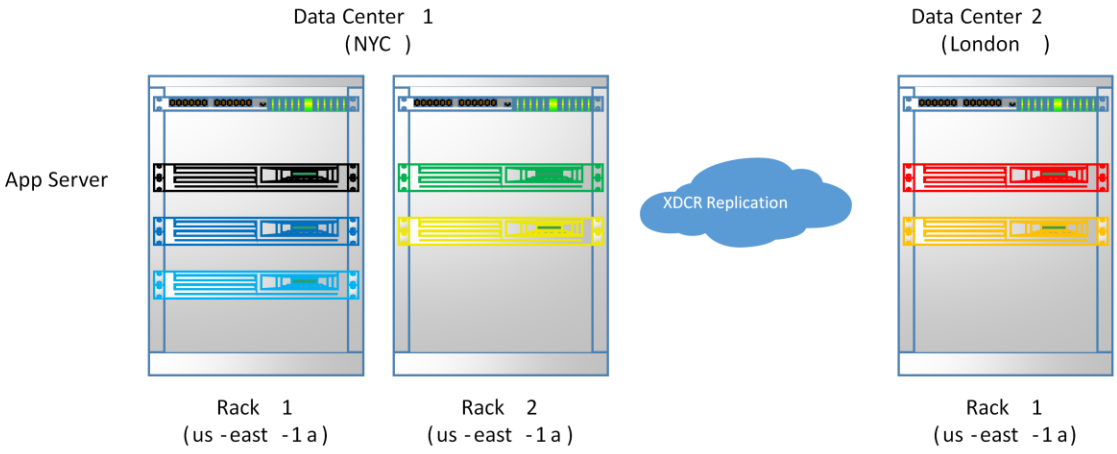
Day 1: Origins of NoSQL & Couchbase, Architecture overview, Installation

Day 2: Multiple node architecture, Deep dive into Admin tasks

Day 3: Views/Indexes, Cross Datacenter Replication(XDCR)

Day 4: Upgrades, Backup, Restore, Performance, Sizing

Lab Environment in Amazon



5

Simulated environment.

Objectives – Module 1

Objectives:

- Install Couchbase 4.x software
- Explore the Web UI
- Utilize command line options and Rest API
- Examine the Beer sample database
- Examine the Couchbase DB storage files in the Linux file system
- Examine log files for Couchbase
- Utilize Couchbase I/O utility commands (cbworkloadgen & pillowfight)

- Install Couchbase EE on 1-node VM in Amazon Web Services (AWS)
- Explore the UI: Cluster overview, cluster summary, viewing buckets, viewing server nodes, viewing data buckets, logs
- Utilize command line options
- Start and stop Couchbase server process
- Examine the Beer sample database
- Examine the Couchbase DB storage files in the Linux file system
- Examine log files for Couchbase

Objectives – Module 2

Objectives:

- Install Couchbase libraries (smart client) on one node(App Server)
- Install a separate Application Server node
- Test Memcached text protocol with telnet
- cbc utility commands to create, read & delete a key

Objectives:

- **Install Couchbase libraries (smart client) on one node(App Server)**
- **Install a separate Application Server node**
- **Test Memcached text protocol with telnet**
- **cbc utility commands to create, read & delete a key**

Objectives – Module 3

Objectives:
Grow cluster to 4 nodes
Exploring the cluster map
Differentiate Buckets and vBuckets
Rebalance the cluster
Introduction to performance metrics via Web UI
Adjust the number of replicas
Use the cbstats command to see active + replica vBuckets

Objectives:
Grow cluster to 4 nodes
Exploring the cluster map
Differentiate Buckets and vBuckets
Rebalance the cluster
Introduction to performance metrics via Web UI
Adjust the number of replicas
Use the cbstats command to see active + replica vBuckets

Objectives – Module 4

Objectives:
Remove nodes and MISC commands):
Compare details on the Web UI's performance metrics
Learn how to delete Couchbase buckets
Gracefully decommission & recommission nodes from a cluster
Failing over a node in the cluster
Replica management
Using the REST API to check auto-failover settings

Objectives:
Remove nodes and MISC commands):
Compare details on the Web UI's performance metrics
Learn how to delete Couchbase buckets
Gracefully decommission & recommission nodes from a cluster
Failing over a node in the cluster
Replica management
Using the REST API to check auto-failover settings

Objectives – Module 5

Objectives:
Server Warmup, Views and Indexes
View server warmup metrics with cbstats command
Locate the warmup access log file on disk
Use cbepctl command to change alog_sleep_time and flush_param settings
Introduction views in the beer-sample bucket
Filter results when running a view
Write a custom View with Map and Reduce code
Compare analytics and metrics for a view
Perform compaction on index files for a view
Locate view/index files on disk
Query a view and change settings on how often the indexer runs via REST API

Objectives:
Server Warmup, Views and Indexes
View server warmup metrics with cbstats command
Locate the warmup access log file on disk
Use cbepctl command to change alog_sleep_time and flush_param settings
Introduction views in the beer-sample bucket
Filter results when running a view
Write a custom View with Map and Reduce code
Compare analytics and metrics for a view
Perform compaction on index files for a view
Locate view/index files on disk
Query a view and change settings on how often the indexer runs via REST API

Objectives – Module 6

Objectives: XDCR:
Create a remote 2-node Couchbase cluster
Configure unencrypted, unidirectional replication between the 4-node and 2-node cluster
Observe design docs and views do not get replicated between clusters
Analyze XDCR conflict resolution and revision counts
Observe tombstones in the data files and XDCR replication
Compare XDCR version 1 and version 2 protocols
Advanced XDCR settings
View outbound and inbound replication statistics
View internal XDCR settings via REST API
Configure optimistic threshold setting for replication streams
Learn how to delete replication streams

Objectives: XDCR:
Create a remote 2-node Couchbase cluster
Configure unencrypted, unidirectional replication between the 4-node and 2-node cluster
Observe design docs and views do not get replicated between clusters
Analyze XDCR conflict resolution and revision counts
Observe tombstones in the data files and XDCR replication
Compare XDCR version 1 and version 2 protocols
Advanced XDCR settings
View outbound and inbound replication statistics
View internal XDCR settings via REST API
Configure optimistic threshold setting for replication streams
Learn how to delete replication streams

Objectives – Module 7

Objectives: Advanced XDCR & Backup/Restore:

Analyze optimistic replication in detail
Write 100,000 items under the optimistic threshold
Write 100,000 items over the optimistic threshold
Demonstrate that XDCR replication keeps occurring even after a node failure
Use vBucketkeygen tool to write 1 key to each of the 1024 vBuckets
Use cbbackup and cbrestore to control data availability

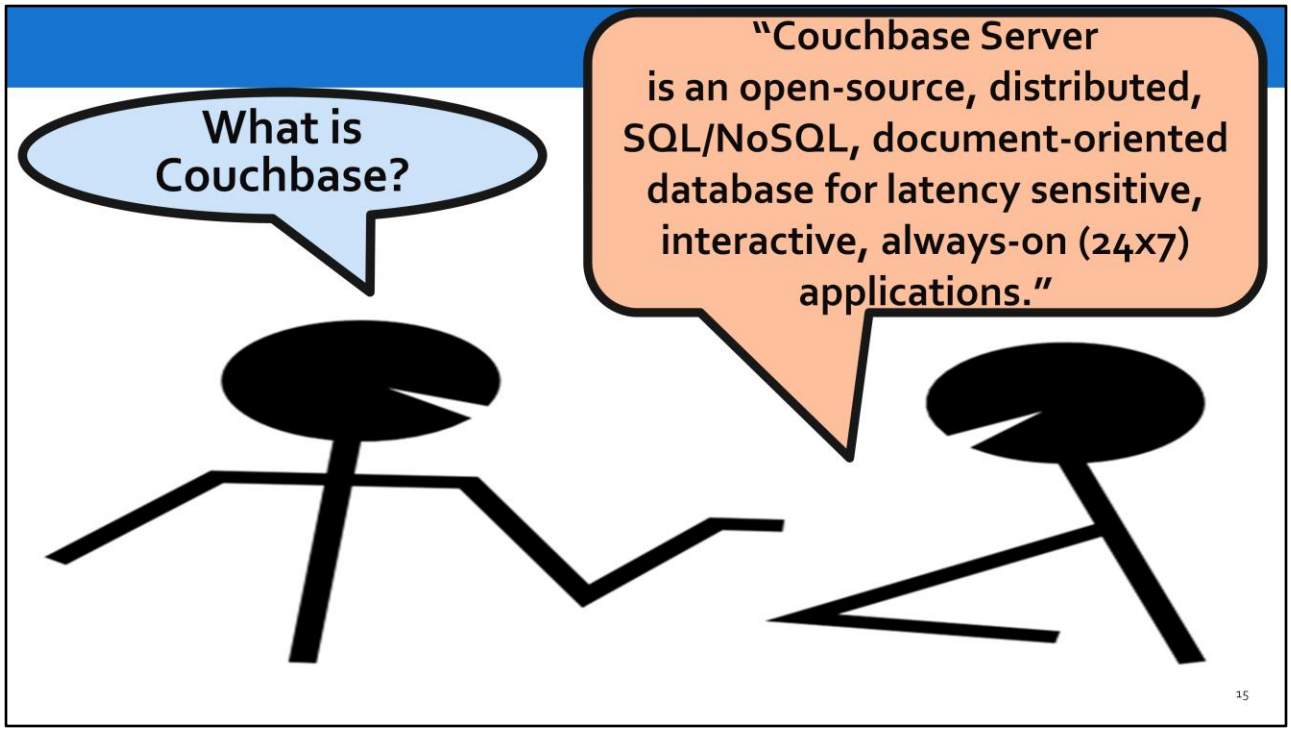
Objectives: Advanced XDCR & Backup/Restore:

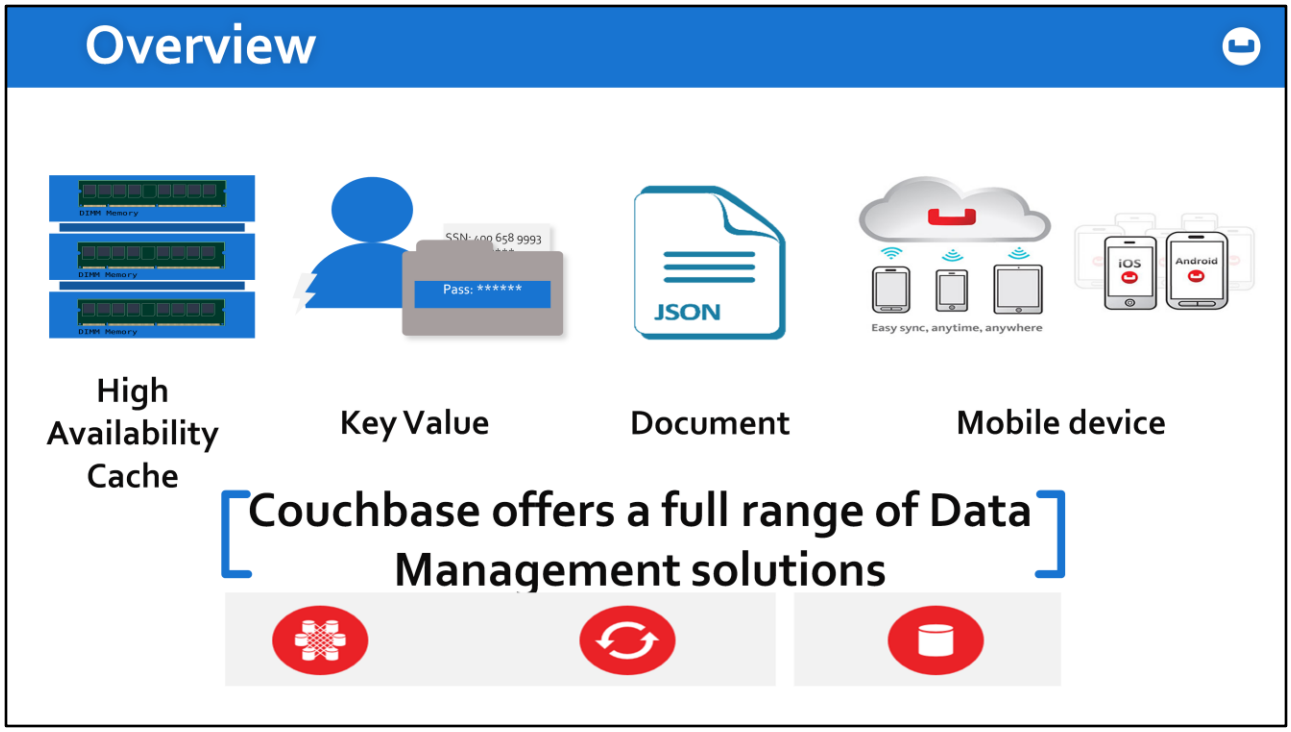
Analyze optimistic replication in detail
Write 100,000 items under the optimistic threshold
Write 100,000 items over the optimistic threshold
Demonstrate that XDCR replication keeps occurring
even after a node failure
Use vBucketkeygen tool to write 1 key to each of the 1024 vBuckets
Use cbbackup and cbrestore to control data availability

Objectives – Module 8

Objectives Performance & Compaction:
Compute high water mark and low water mark for a bucket
Observe how ejection works
Analyze the Not-Recently-Used metadata setting
Determine when the item pager runs
Determine when Disk reads vs RAM reads occur
Understand how different traffic patterns require
different Couchbase settings
Understand the 'resident %' in memory metric
Observe out of memory (OOM) errors
Display metrics via cbstats and studying item expiration
Display timing metrics with "cbstat timings"
Perform Compaction

Intro to Couchbase & NoSQL


What is Couchbase?
"Couchbase Server is an open-source, distributed, SQL/NoSQL, document-oriented database for latency sensitive, interactive, always-on (24x7) applications."
15

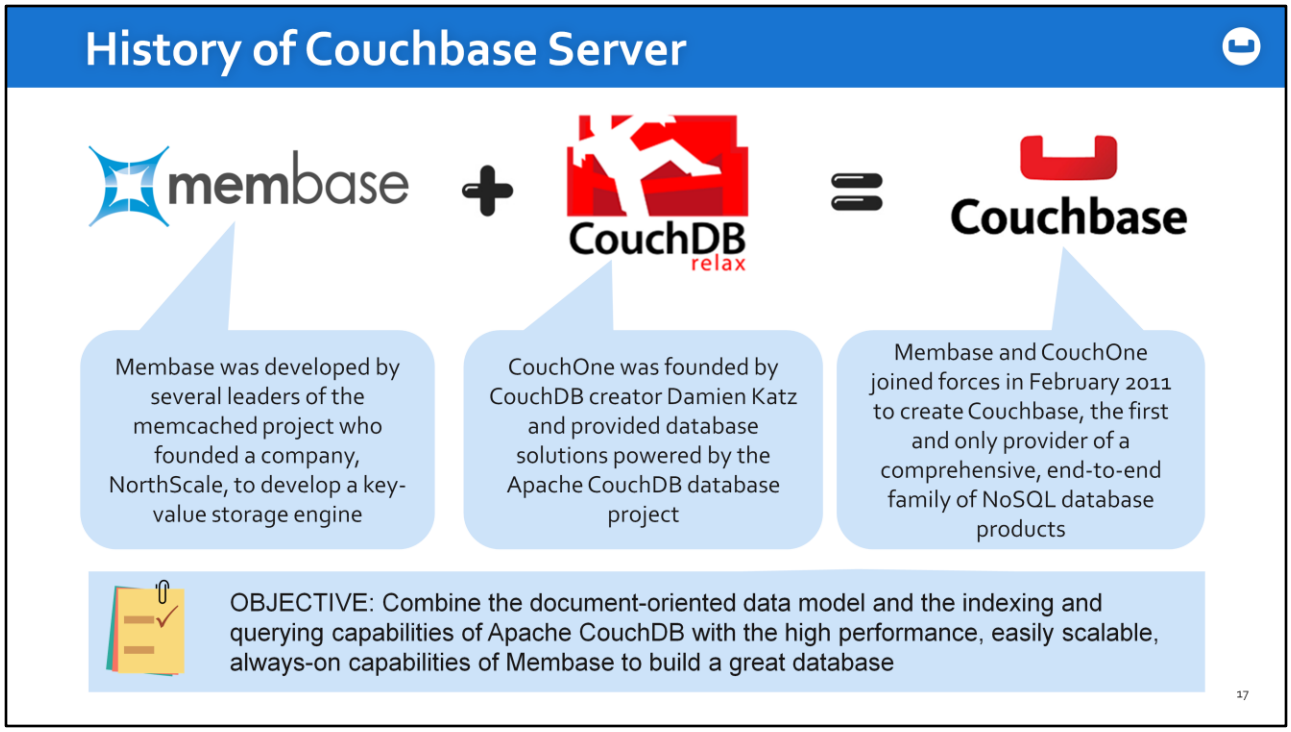


Couchbase Server
A high performance, scalable, always-on JSON database in the cloud. Submillisecond, high-throughput reads and writes give you consistent high performance. Couchbase Server is easy to scale out, and supports topology changes with no downtime.

Sync Gateway
Everything you need to securely sync on-device Couchbase Lite with Couchbase Server in the cloud.

Replication, Authentication , Access Control, Data Routing , Validation

Couchbase Lite
A fully functional, on-device, lightweight, native, embedded JSON database.
With Couchbase Lite, you have the full power of a Couchbase database locally on the device. You can create, update, delete, query, sync and much, much more.



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

Couchbase founders were key contributors to Memcached.

Apache CouchDB is the open source document database originally created by Damien Katz. Damien started a company (along with J Chris Anderson and Jan Lehnardt) called CouchOne Inc. to provide commercial support for the project. Around the same time another open source distributed key-value database called Membase was getting a lot of traction.

In February 2011, CouchOne Inc. and Membase Inc. merged to form Couchbase. The idea behind the merger was pretty simple: combine the document-oriented data model and the indexing and querying capabilities of Apache CouchDB with the high performance, easily scalable, always-on capabilities of Membase to build a great database.

Membase had a product called Membase which was a key/value, persistent, scalable solution that used the Memcached wire protocol.

CouchOne supported CouchDB. CouchDB is a document database which has a peer to peer replication approach, which is really good for mobile and geographically separated data centers. Couchbase created a new product combining parts of Membase and parts of CouchDB, and the new product is called Couchbase.

More info on membase: http://nosql.mypopescu.com/post/728983002/what-is-membase

There are important differences between Couchbase CE and EE

	Couchbase CE	Couchbase EE
Support & Updates		
24 x 7 Support	NO	YES
Patches & Updates	NO	YES
HA & DR		
XDCR Filtering	NO	YES
Rack-Zone Awareness	NO	YES
Advanced Backup & Restore	NO	YES
Security		
LDAP Integration	NO	YES
Encryption	NO	YES
Auditing	NO	YES
Scalability & Performance		
Multi-dimensional Scaling	NO	YES
High-performance concurrent queries	NO	YES

Couchbase

[BUILD SLIDE]

Finally, Couchbase EE will simply perform better than CE. You get much better query throughput and higher concurrent query executions due to EE's high performance query execution engine.

KEY QUESTIONS

- Which of your current applications do you anticipate will require the largest scalability boost in the medium term?
- Do any specific applications require very low latency and sub-millisecond response time?

Couchbase Server Core Principles

Easy Scalability

Grow cluster without application changes, without downtime with a single click

Consistent High Performance

Consistent sub-millisecond read and write response times with consistent high throughput

Always On 24x365

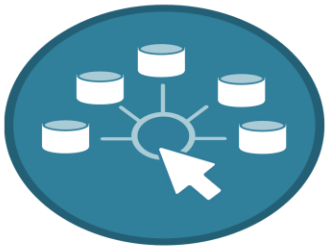
No downtime for software upgrades, hardware maintenance, etc.

Flexible Data Model

JSON Anywhere document model with no fixed schema.

Easy Scalability

FEATURES



Auto Sharding

No Manual Sharding

Database manages data movement to scale out - Not the user

XDCR

Database handles propagation of updates to scale across clusters and geos

Provides disaster recover / data locality

Single package Multi-service

Hugely simplifies management of clusters

Easy to scale clusters by adding any # of nodes

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 enables you to move data closer to your user at other data centers.

Consistent, High Performance

FEATURES

Massive Concurrent Connections	Fine Grained Locking	Built-in Cache	Hash Partitioning
Support a large number of users needed for interactive apps	Allows high concurrency and in turn high throughput via highly granular latches	No need of separate cache layer Database manages actively used data	Uniform data distribution Uniform load distribution – NO hotspots

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.

Always on 24x7 Capability

FEATURES

Online DB upgrades and maintenance	Online administrative operations	HA via Replication DR via XDCR
Online DB upgrades and HW maintenance Optimized swap operation to replace nodes	All admin operations online • Compaction • Indexing • Rebalance • Backup & Restore	• High availability using in-memory replication • Auto or manual failover • XDCR for disaster recovery

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.

Flexible Data Model

FEATURES

Schema-less for structured / un/semi-structured data

Data with mixed structure better managed via JSON in a document DB than an RDBMS

Maintains Native object representation

Represent data as objects instead of shredding into rows and columns

Create indexes on any attribute of a JSON document

Handles constantly changing data

Each document can have a different structure

Easy to change data without database changes and downtime

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.

Couchbase 4.5 - Key Features

EASIER, MORE EFFICIENT QUERYING

- **Integrated query workbench with auto-schema inference**
- Extended JOIN syntax
- Integrated full-text search queries [preview]

SIMPLER, MORE ADVANCED DATA ACCESS

- **Efficient Read/Write of data structures with sub-document API**
- Faster read-your-own-write (RYOW) consistency with N1QL queries

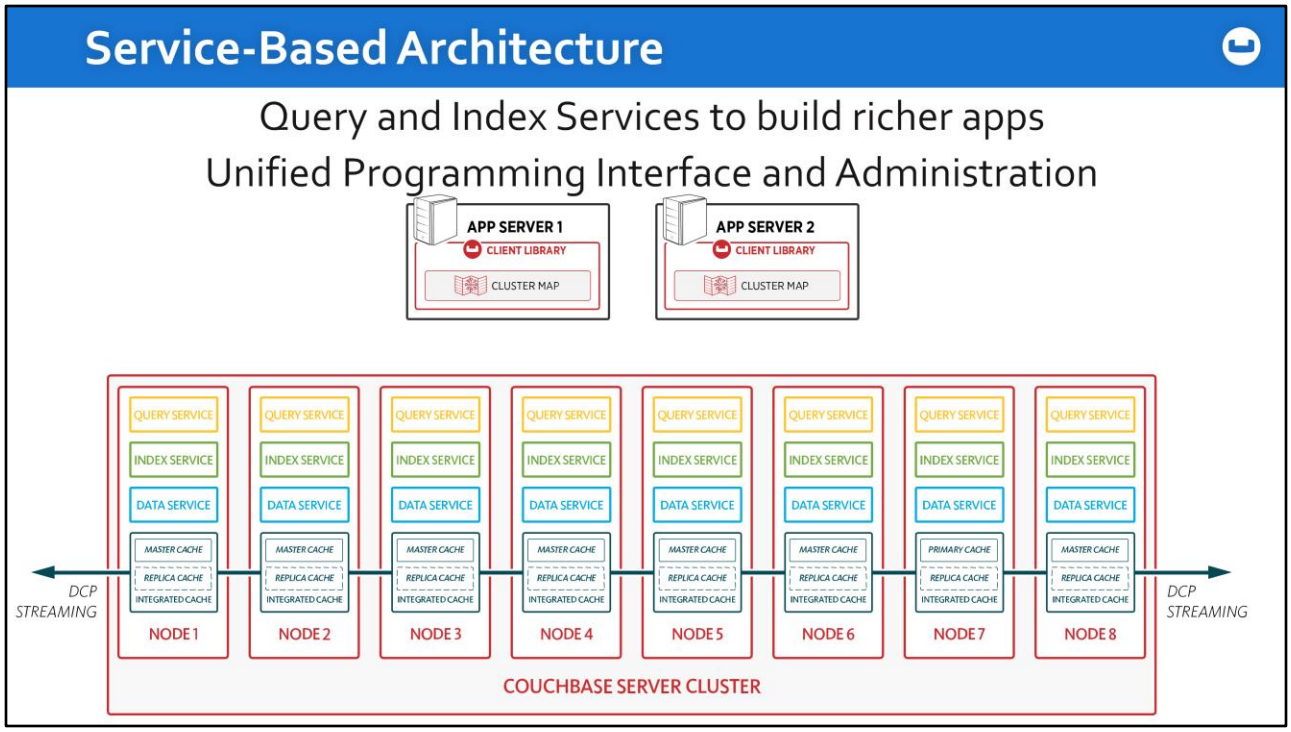
FASTER, MORE POWERFUL INDEXING

- **Faster Indexing with Memory-optimized indexes**
- Array indexing for faster N1QL queries on arrays in JSON
- Improved index write performance with ForestDB's circular writes

BETTER, MORE COMPREHENSIVE ADMINISTRATION

- Role-based-access control for administrators & X.509 certificates for encrypted communication
- Query profiling and monitoring
- Faster backup and restore

Couchbase Architecture 101



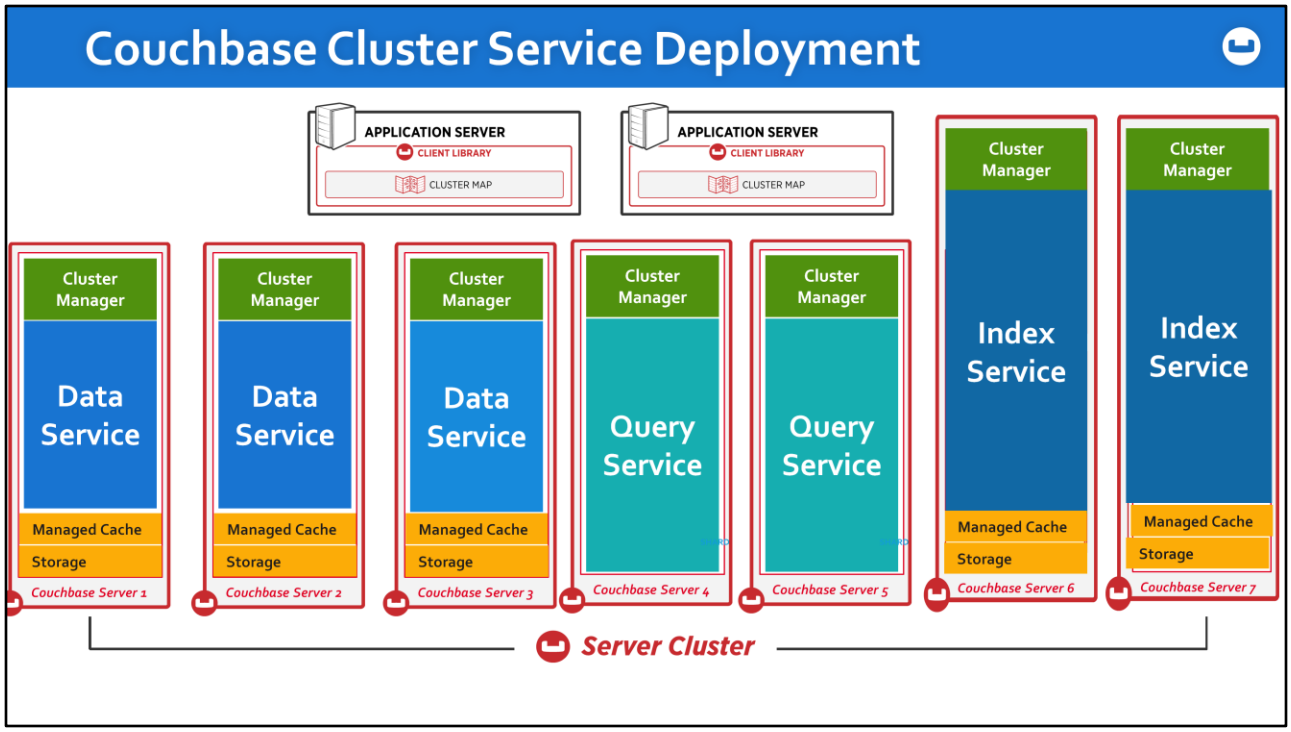
To get started, let's take a look at 4.5(since 4.0) couchbase cluster architecture.
In this architecture, you can have any service at any node
Data service is your repository for couchbase buckets for documents/data
Index service is for global secondary index(GSI)
Query service is for running N1QL query

These services all come in the same couchbase package(rpm)
They can all be run as services on the same node or they can be offered in any combination that makes sense for the user.
Separating services can allow the deployment to scale independently for each service.

On top of our scale-out and memory/network centric architecture, we layered a service-based architecture to allow us to add new services that are high performant, scalable, and highly available.

We started with KV data service, and we added the new query and global index services in 4.0. As Ravi mentioned, we are adding a full-text search in our next release, and analytic service after that.

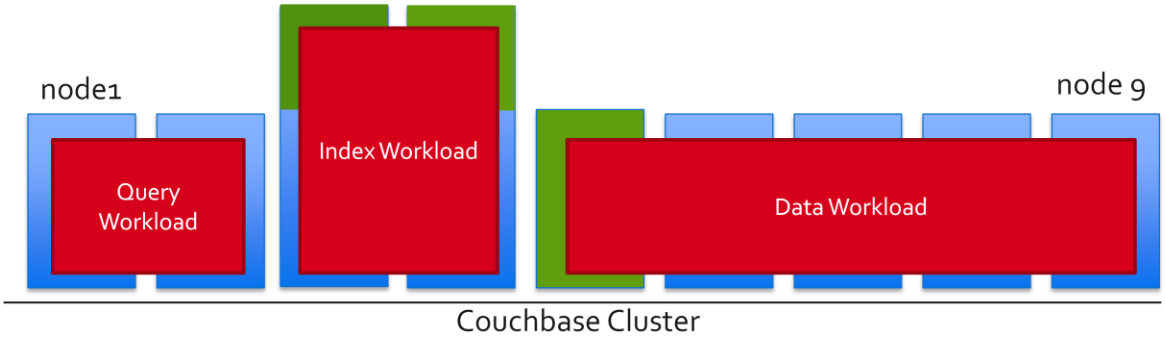
Because we have a consistent architecture, all these services can be accessed with a unified programming interface, making it easier for developers and they do not have to cobble together multiple systems with multiple data models and interfaces. For example, many of our customers are using our Elastic connector to support full text search with Couchbase Server and consistently they have been asking for a full text search services to be provided within the Couchbase Server platform so that it is easy to program to and easy to manage and operate.



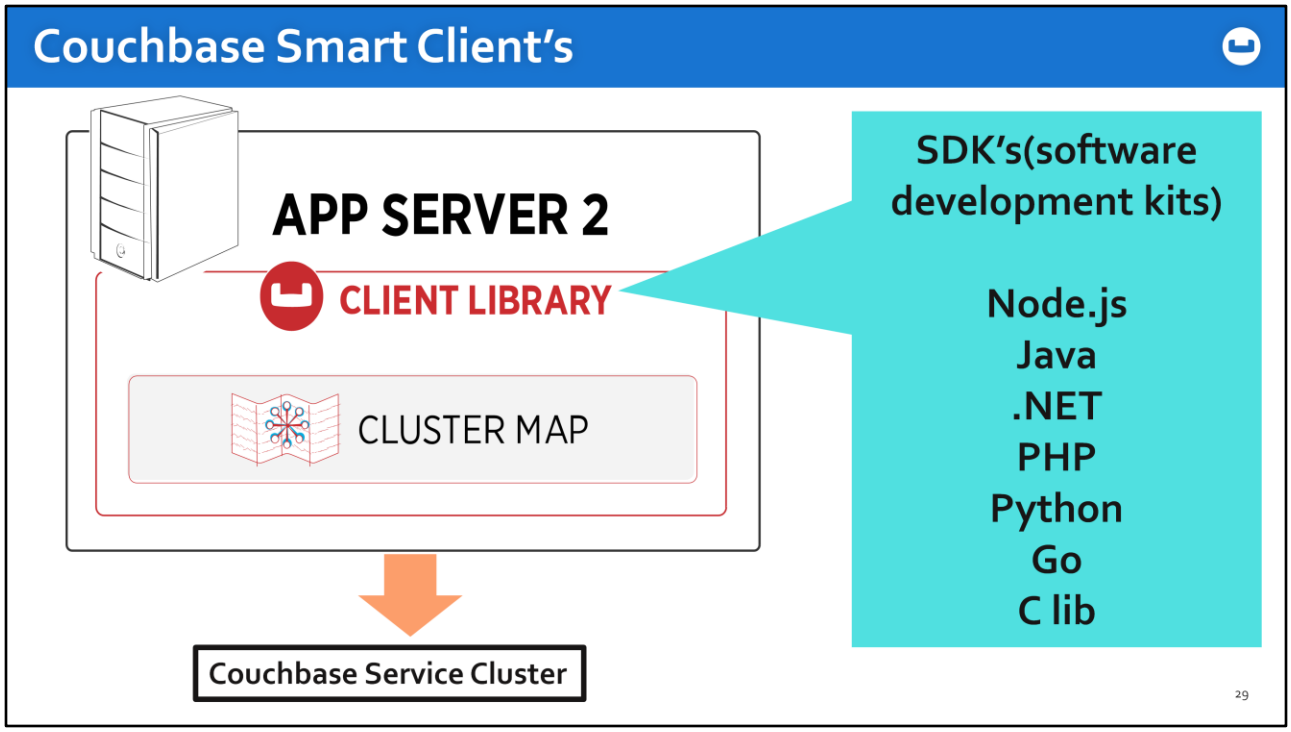
In this diagram, you can see how you can choose a service for any node.
Here you have 3 data services, 2 query service and 2 index services.
Or any combination thereof that is supported by the physical resources on the machine.

Independent Scaling

Add Indexing capacity as number of indexes increase
Scale-up and/or Scale-out Options



You can add index services as you add more indexes.
You have the option to size up your index node or add more index nodes as you go.



Once you've installed the server, you can start storing, retrieving, and querying documents with Couchbase. You can start with an SDK, the command-line cbc tool, or the web browser.

Every item in a database goes through the basic **CRUD** cycle, which is typical of an application's use of data. CRUD stands for create, read, update, and delete:

Create: when data is first inserted into the cluster

Read: when an application retrieves the data

Update: when data is modified to reflect a change in the state represented by the data

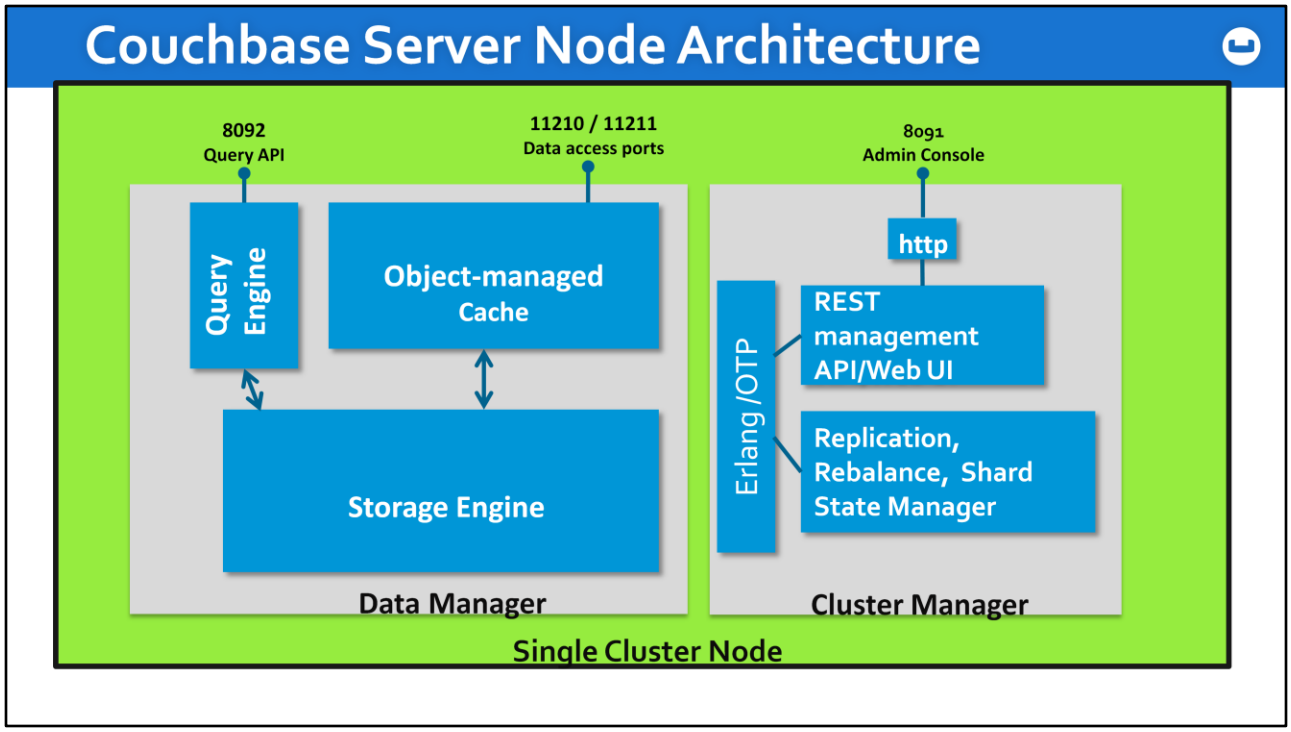
Delete: when the data is no longer needed

Couchbase clients

Clients access data by connecting to a Couchbase cluster over the network.
The most common type of client is a Couchbase SDK which is a full programmatic API that enables applications to take the best advantage of Couchbase.
The command line client provides a quick and streamlined interface for simple access

and is suitable if you just want to access an item without writing any code.

The Couchbase C SDK (libcouchbase) enables C and C++ programs to access a Couchbase cluster.
The C SDK is also commonly used as a core dependency of SDKs written in other language to provide a common implementation and high performance.



In Couchbase Server, the data manager stores and retrieves data in response to data operation requests from
applications. It exposes two ports to the network: one for non-smart clients (11211), that can be proxied via Moxi
(a Memcached proxy for couchbase) if needed, and the other for smart clients (11210). The majority of code in the
data manager is written in C and C++.

As I mentioned, each Couchbase node is exactly the same.

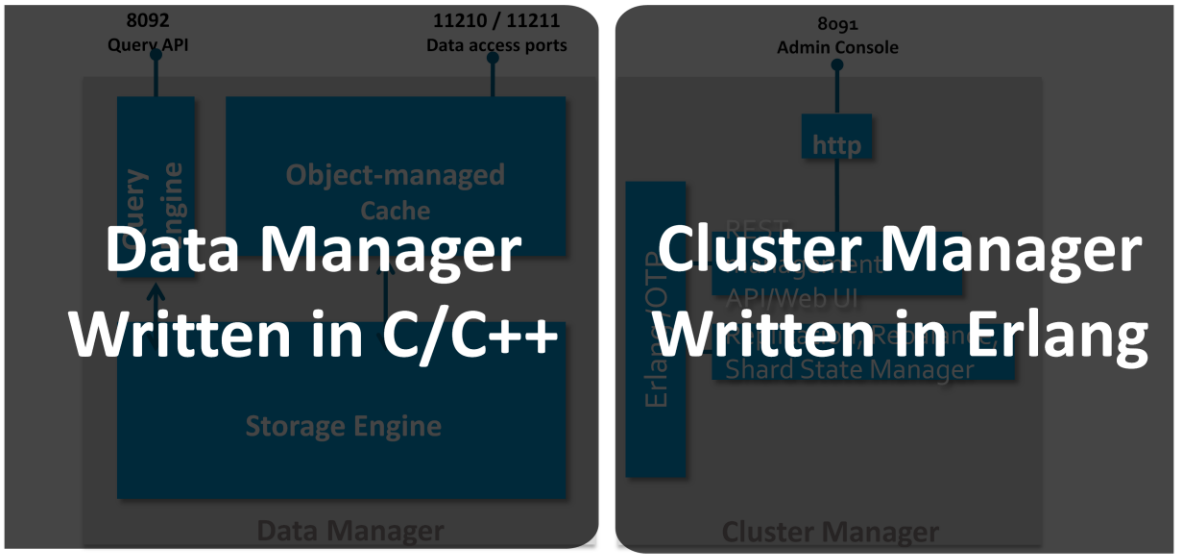
All nodes are broken down into two components: A data manager (on the left) and a cluster manager (on the right). It's important to realize that these are separate processes within the system specifically designed so that a node can continue serving its data even in the face of cluster problems like network disruption.

The data manager is written in C and C++ and is responsible both for the object caching layer, persistence layer and querying engine. It is based off of Memcached and so provides a number of benefits;
-The very low lock contention of Memcached allows for extremely high throughput and low latencies both to a small set of documents (or just one) as well as across millions of documents

-Being compatible with the Memcached protocol means we are not only a drop-in replacement, but inherit support for automatic item expiration (TTL), atomic incrementer.
-We've increased the maximum object size to 20mb, but still recommend keeping them much smaller
-Support for both binary objects as well as natively supporting JSON documents
-All of the metadata for the documents and their keys is kept in RAM at all times. While this does add a bit of overhead per item, it also allows for extremely fast "miss" speeds which are critical to the operation of some applications….we don't have to scan a disk to know when we **don't** have some data.

The cluster manager is based on Erlang/OTP which was developed by Ericsson to deal with managing hundreds or even thousands of distributed telco switches. This component is responsible for configuration, administration, process monitoring, statistics gathering and the UI and REST interface. Note that there is no data manipulation done through this interface.

Couchbase Server Architecture



The data manager is written in C and C++ and is responsible both for the object caching layer, persistence layer and querying engine. It is based off of Memcached and so provides a number of benefits;
-The very low lock contention of Memcached allows for extremely high throughput and low latencies both to a small set of documents (or just one) as well as across millions of documents
-Being compatible with the Memcached protocol means we are not only a drop-in replacement, but inherit support for automatic item expiration (TTL), atomic incrementer.
-We've increased the maximum object size to 20mb, but still recommend keeping them much smaller
-Support for both binary objects as well as natively supporting JSON documents
-All of the metadata for the documents and their keys is kept in RAM at all times. While this does add a bit of overhead per item, it also allows for extremely fast "miss" speeds which are critical to the operation of some applications….we don't have to scan a disk to know when we **don't** have some data.

The cluster manager is based on Erlang/OTP which was developed by Ericsson to deal with managing hundreds or even thousands of distributed telco switches. This component is responsible for configuration, administration, process monitoring, statistics gathering and the UI and REST interface. Note that there is no data manipulation done through this interface.

What is Erlang?

Erlang is a programming language used to build massively scalable soft real-time systems with requirements on high availability. Some of its uses are in telecoms, banking, e-commerce, computer telephony and instant messaging. Erlang's runtime system has built-in support for concurrency, distribution and fault tolerance.

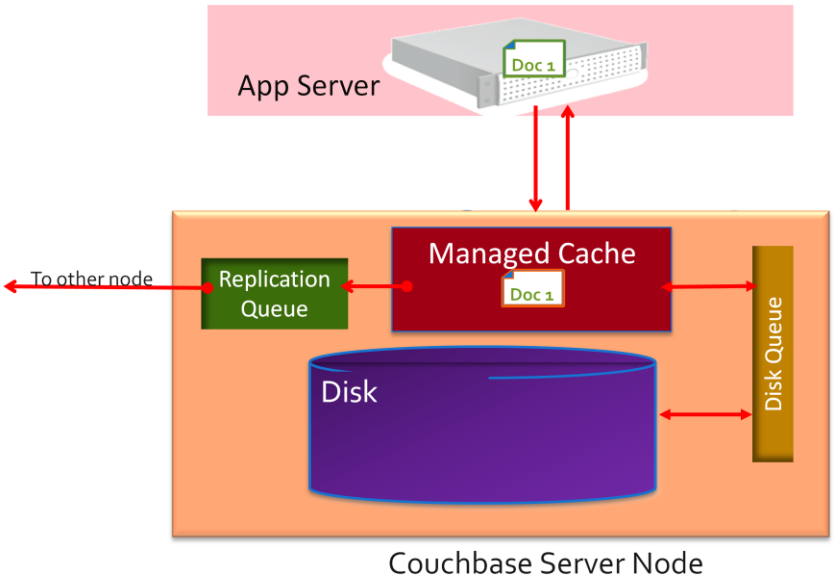
What is OTP?

OTP is set of Erlang libraries and design principles providing middle-ware to develop these systems. It includes its own distributed database, applications to interface towards other languages, debugging and release handling tools.
……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

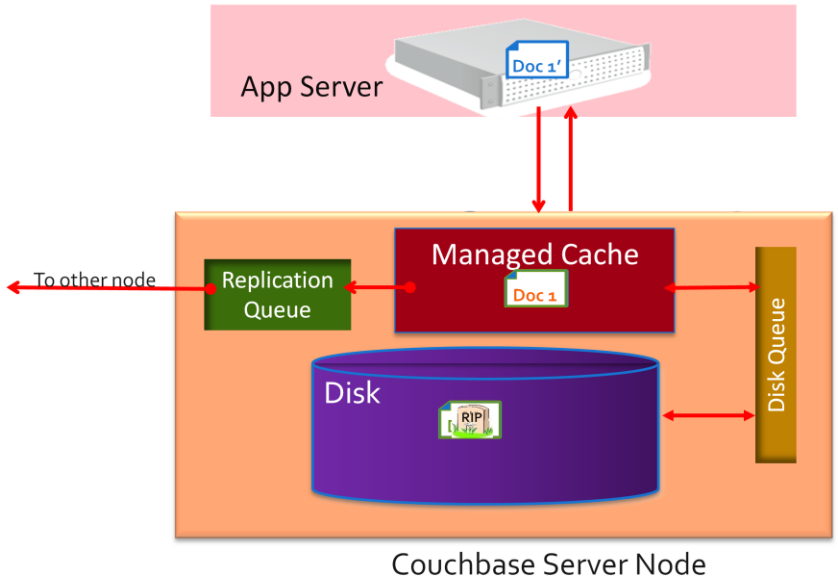
Access to the Cluster Manager is provided through the administration interface (see Administration Tools) on a dedicated network port, and through dedicated network ports for client access. Additional ports are configured for inter-node communication.

Single node - Couchbase Write Operation



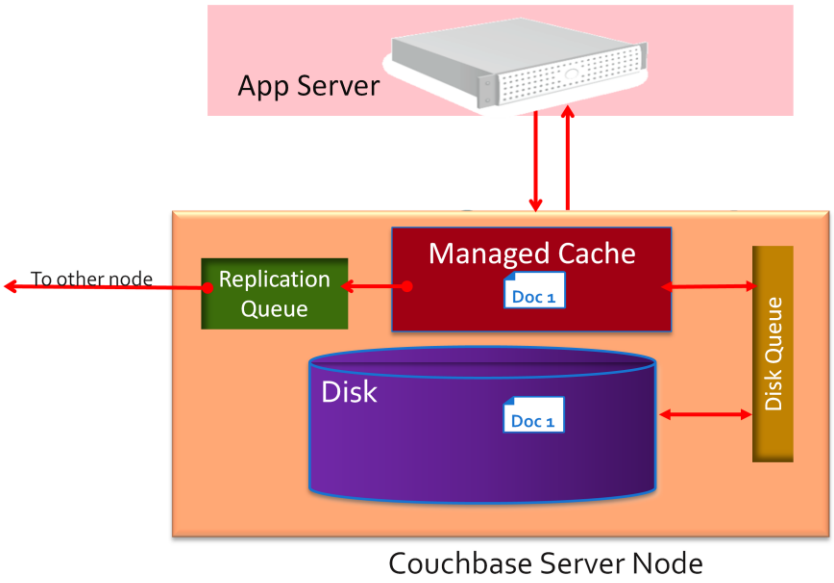
1. A set request comes in from the application .
2. Couchbase Server responses back that they key is written
3. Couchbase Server then quickly Replicates the data out to memory in the other nodes
4. At the same time it is put the data into a write que to be persisted to disk

Single node - Couchbase Update Operation



1. A set request comes in from the application .
2. Couchbase Server responses back that they key is updated
3. Couchbase Server then quickly Replicates the updated data out to memory in the other nodes
4. At the same time it is put the data into a write que to update the data on disk

Single node - Couchbase Read Operation



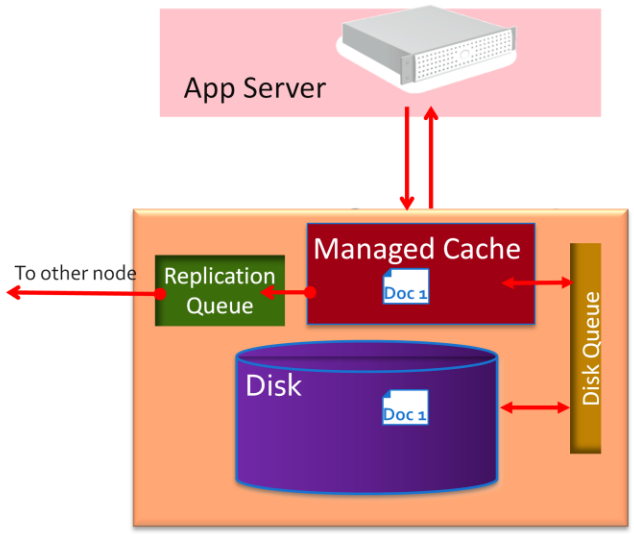
1. A get request comes in from the application .
2. Couchbase Server sends the data from the caching layer at very low latency to the reader (if the data is not already in the working set and cached, it gets read from disk into cache first)

Durability settings

- ReplicateTo option.
- PersistTo option.

Ensure the modification :

- stored (disk) on the active (master) node.
- exists (memory) on at least one replica.
- is stored (disk) on at least one replica.
- exists (memory) on all replicas.
- is stored (disk) on the active node, and exists (memory) on all replicas.
- is stored (disk) on the active node, and on all replicas.



35

Durability requirements

Modifications to items are done synchronously in memory and asynchronously on the disk and network. The synchronous modification in memory involves modifying the item and sending a success response to the client. This tells your application that the item has successfully been stored/modified, and allows your application to continue operating.

Applications may specify requirements for durability to the Couchbase client to wait until the item has also been persisted and replicated. This ensures greater durability of your data against failure in certain edge cases, such as a slow network or storage media. Operations performed using the durability requirement options will appear as failures to the application if the modification could not be successfully stored to disk and/or replicated to replica nodes. Upon receiving such a failure response, the application may retry the operation or mark it as a failure. In any event, the application will never be under the mistaken assumption that the modification was performed when it fact it was lost inside one of the aforementioned queues.

Configuring durability in SDKs

Durability requirements are typically present as an extra option to mutation operations (such as upsert). The level of durability is configurable depending on the

needs of the application.
The level of durability may be expressed in two values:

The **replication factor of the operation**: how many replicas must this operation be propagated to.
This is specified as the **ReplicateTo** option in most SDKs.
The **persistence of the modification**: how many persisted copies of the modified record must exist.
Often found as **a PersistTo** option.

Examples of these settings include:

Ensure the modification is stored (on disk) on the active (master) node.
Ensure the modification exists (in memory) on at least one replica.
Ensure the modification is stored (on disk) on at least one replica.
Ensure the modification exists (in memory) on all replicas.
Ensure the modification is stored (on disk) on the active node, and exists (in memory) on all replicas.
Ensure the modification is stored (on disk) on the active node, and on all replicas.

When an operation is propagated to a replica, it is first propagated to the replica's memory and then placed in a queue (as above) to be persisted to the replica's disk. You can control whether you desire the operation to only be propagated to a replica's memory (ReplicateTo), or whether to wait until it is fully persisted to the replica's disk (PersistTo).

PersistTo receives a value total to the number of nodes that an operation should be persisted. The maximum value is the number of replica nodes plus one. A value of 1 indicates master-only persistence (and no replica persistence).

ReplicateTo receives a value total to the number of replica nodes the operation should be replicated. Its maximum value is the number of replica nodes.

Optimistic and pessimistic locking

CAS :acronym for Compare And Swap, a form of *optimistic locking*. The CAS can be supplied by applications to mutation operations .When applications provide the CAS, server will check the application-provided version of CAS against its own version of the CAS:

- If two CAS values match (compare successfully), then mutation operation succeeds.
- If the two CAS values differ, then the mutation operation fails

Lock and Unlock: *pessimistic concurrency*

- Using Lock ensures that no other users can update the key or value if it is locked. Unlock removes the lock held on key so that it can be mutated by other users.
- First, we get the document and lock it, giving us a 5 second lock to complete the operation. Then we update the original document and replace it. Finally, we unlock the document using the original CAS value from the GetWithLockAsync call.

36

Concurrent document mutations

You can use the CAS value to control how concurrent document modifications are handled

The CAS is a value representing the current state of an item. Each time the item is modified, its CAS changes.

The CAS value itself is returned as part of a document's metadata whenever a document is accessed (for example, in the Python client, getting an item returns a ValueResult, the ValueResult contains the value (i.e. the JSON document) as well as the CAS.
CAS is an acronym for Compare And Swap, and is known as a form of optimistic locking. The CAS can be supplied by applications to mutation operations (insert, upsert, replace). When applications provide the CAS, server will check the application-provided version of CAS against its own version of the CAS:

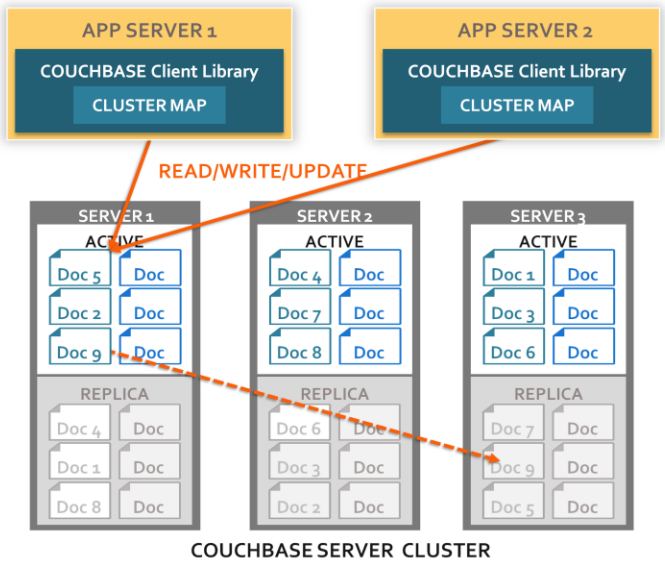
If the two CAS values match (they compare successfully), then the mutation operation succeeds.
If the two CAS values differ, then the mutation operation fails

Lock and Unlock: pessimistic concurrency

While CAS on an operation handles optimistic concurrency without requiring explicit locks, pessimistic concurrency can be achieved by using Lock and Unlock. Using Lock ensures that no other users can update the key or value if it is locked. Unlock removes the lock held on key so that it can be mutated by other users.

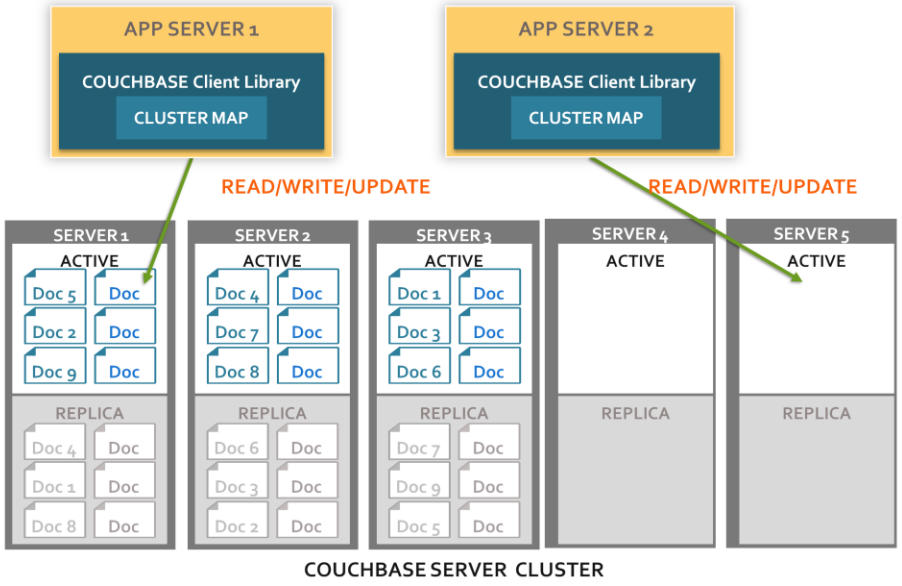
This is roughly equivalent to the optimistic concurrency example. First, we get the document and lock it, giving us a 5 second lock to complete the operation. Then we update the original document and replace it. Finally, we unlock the document using the original CAS value from the GetWithLockAsync call.

Basic Operation



- Docs distributed evenly across servers via vbuckets+hash algorithm
- Each server stores both active and replica docs in memory and on disk
  - Only one server active at a time
- Client library provides app with simple interface to database
- Cluster map provides map to which server doc is on
  - App never needs to know
- App reads, writes, updates docs
- Multiple app servers can access same document at same time

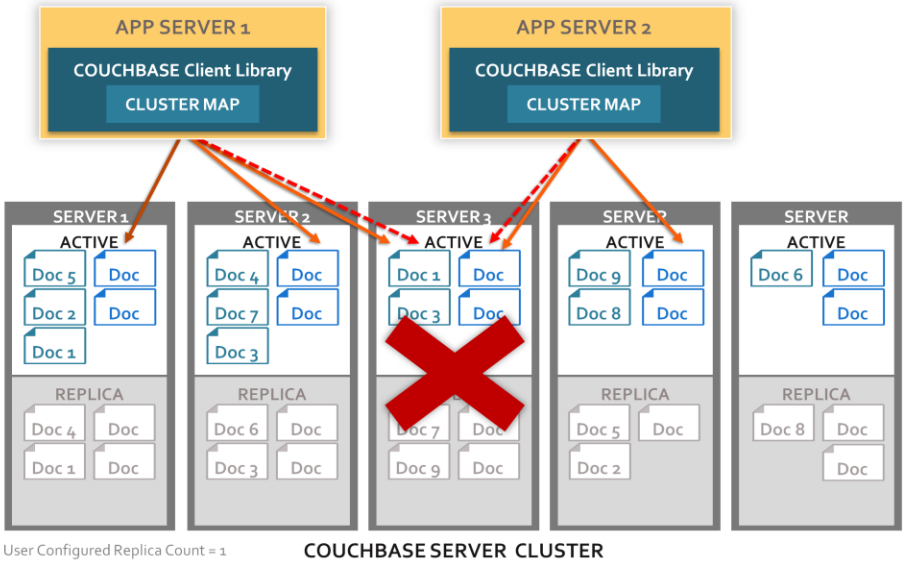
Add Nodes to Cluster



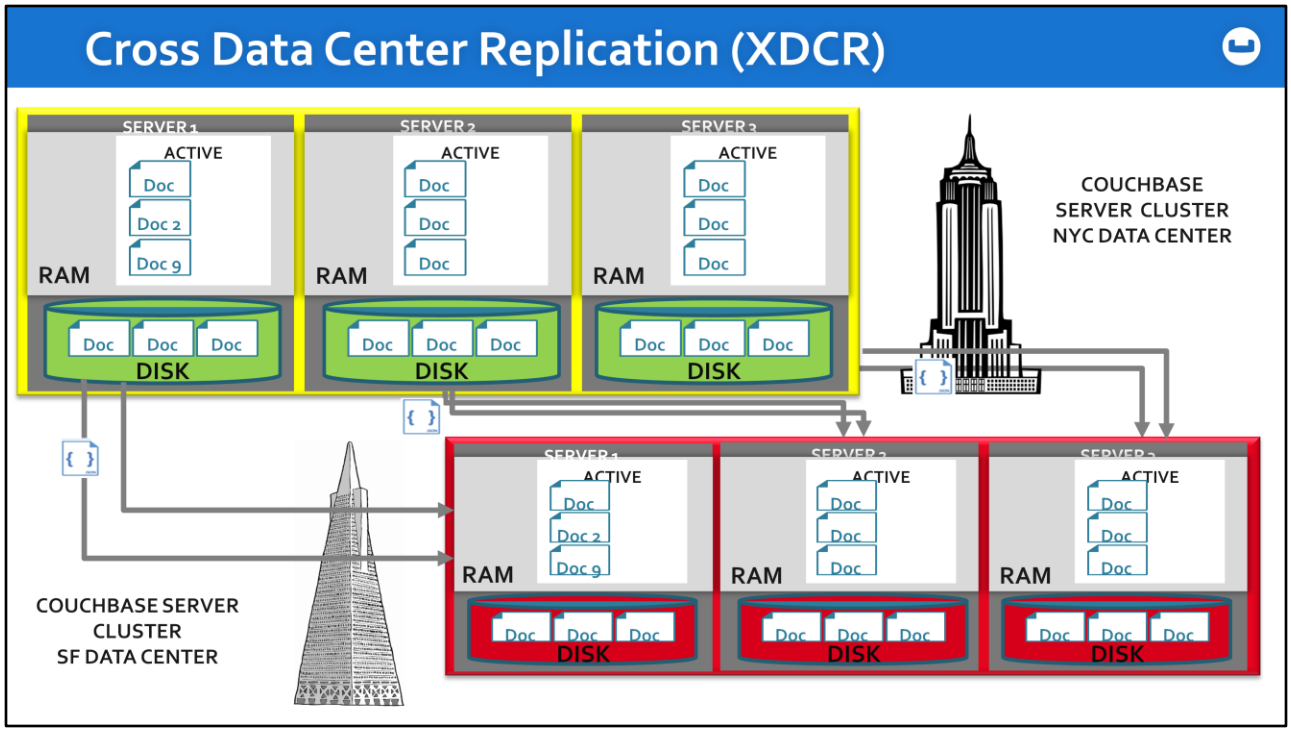
- Two servers added with one-click operation
- Docs automatically rebalance across cluster
  - Even distribution of docs
  - Minimum doc movement
- Cluster map updated
- App database calls now distributed over larger number of servers

UserConfigured ReplicaCount=1

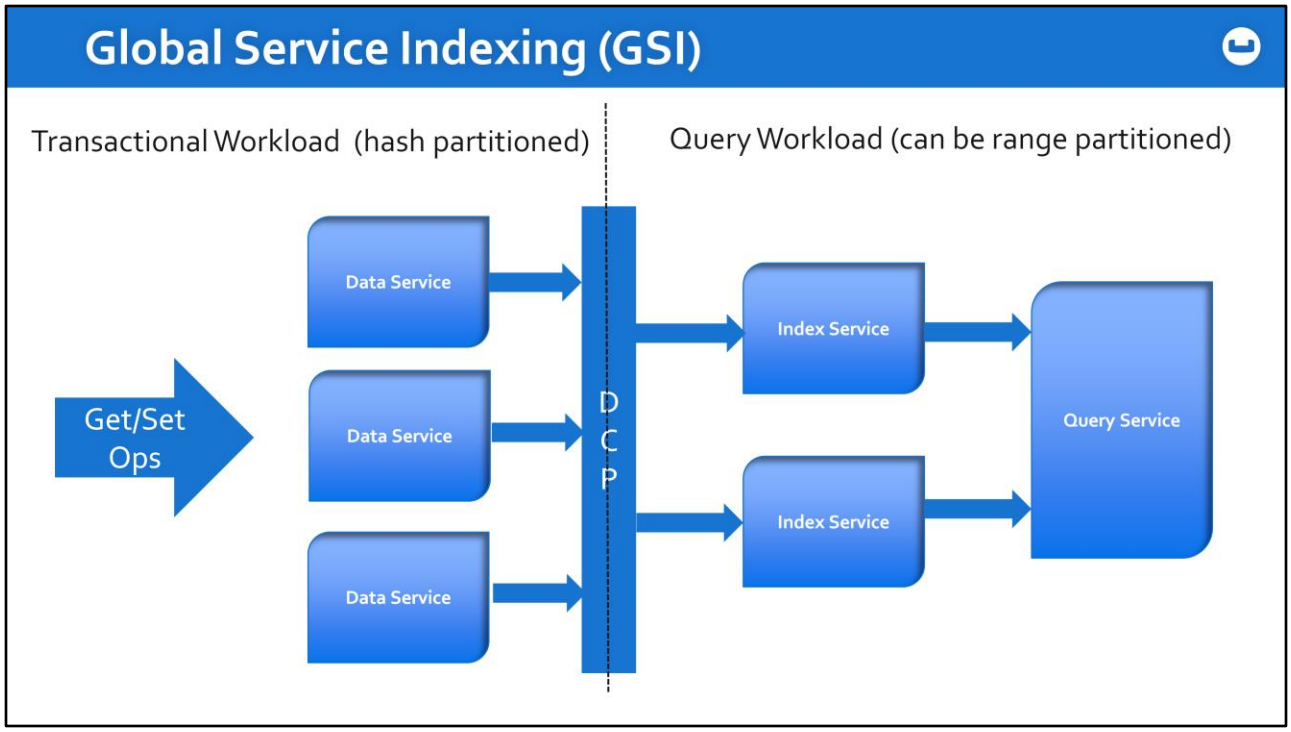
Fail Over Node



- App servers accessing docs
- Requests to Server 3 fail
- Cluster detects server failed
  - Promotes replicas of docs to active
  - Updates cluster map
- Requests for docs now go to appropriate server
- Typically rebalance would follow

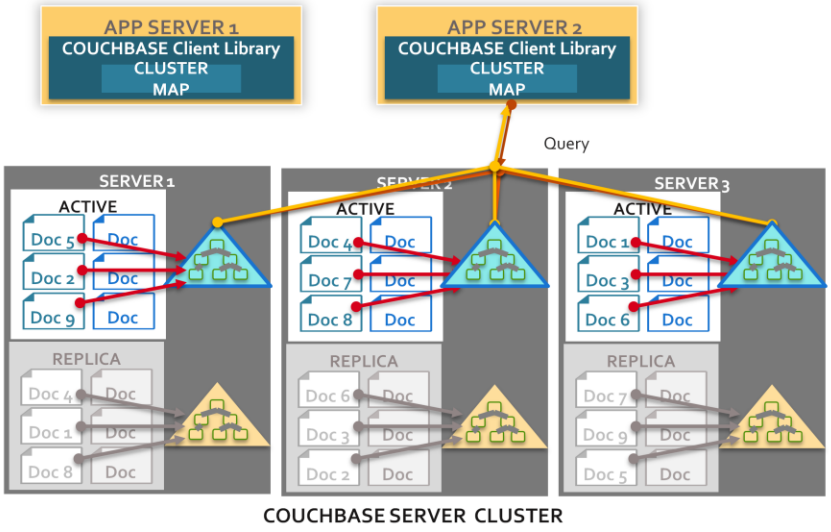


- Move data close to users
  - Read and write data in any datacenter
- Multiple locations for disaster recovery
- Independently managed clusters serving same data



With GSI, you can separate query workload from transaction workload.
Using DCP, mutations in transactional workload is replicated to the index services.
And with this, you can add indexes to the query workload with minimal impact to your KV performance.
In addition, you can have your documents to be sharded using hash partitioning, but index using range partitioning.

Views - Indexing and Querying



- Indexing work is distributed amongst nodes
- Large data set possible
- Parallelize the effort
- Each node has index for data stored on it
- Queries combine the results from required nodes

Indexing work is distributed amongst nodes

Each node creates a single index for all its active data and optionally for it replica data.

Large data set possible

Indexing is distributed across the nodes, so a cluster in total can cope with large amounts of data, and can keep each nodes index managable and fast

Parallelize the effort

Queries combine the results from all required indexes and are returned as one single result (scatter gather)

Any node can be queried, no bottleneck

Indexing and querying

distributed

create indexes on the fields in JSON documents

Called Views in Couchbase Server

Views are queried to find the objects you are interested in, e.g. range queries to find all players that have black sheep on their farm

(if asked: No ad-hoc query language. Index are described via simple Javascript.)

Incremental Map Reduce

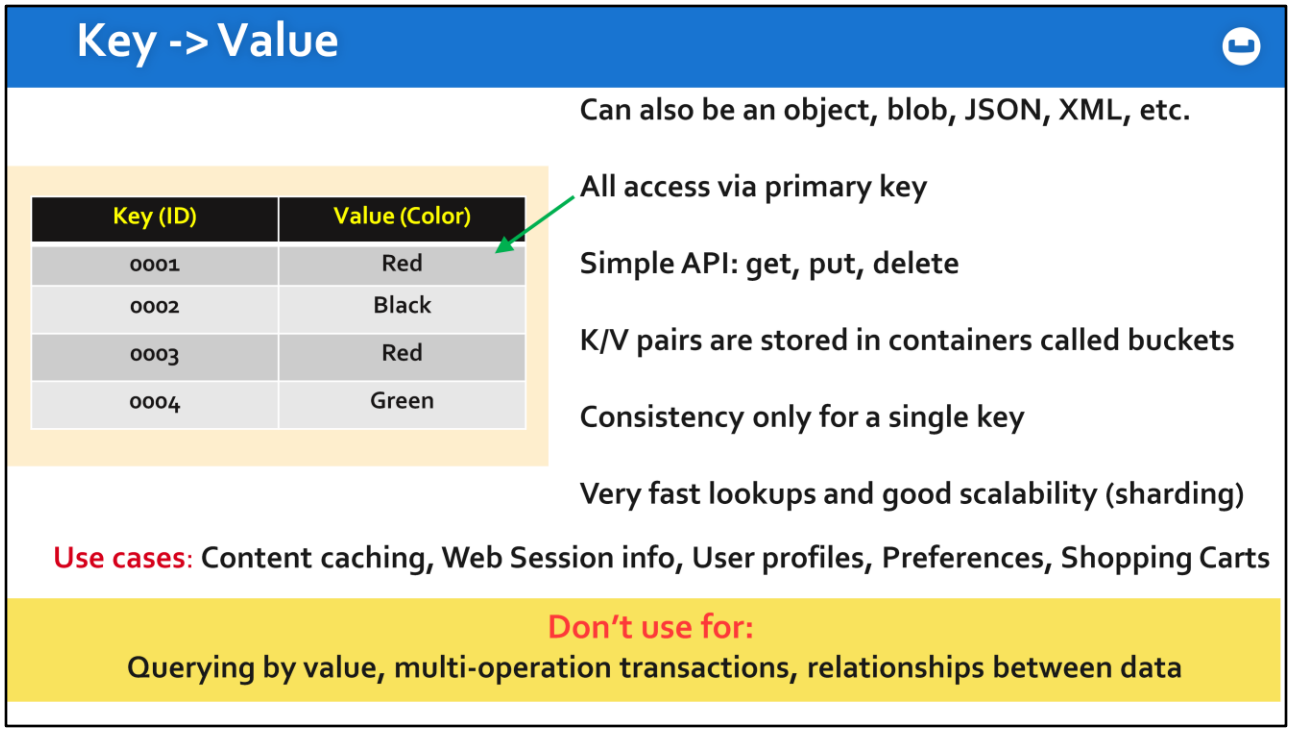
“Normal” map reduce is batch based: i.e. it has to run across all data everytime. So you don’t get updated results often, especially over large data sets. Incremental Map reduce is only considering data that has changed and then calculates the updated result.

This happens fast, in near real-time.

Distributed across all nodes, so able to cope with large data amounts

Does single map and reduce step, so great for simple analytics like leaderboards, counts sums, across data having specific attributes/charcteristics.

Couchbase Data Modeling Fundamentals

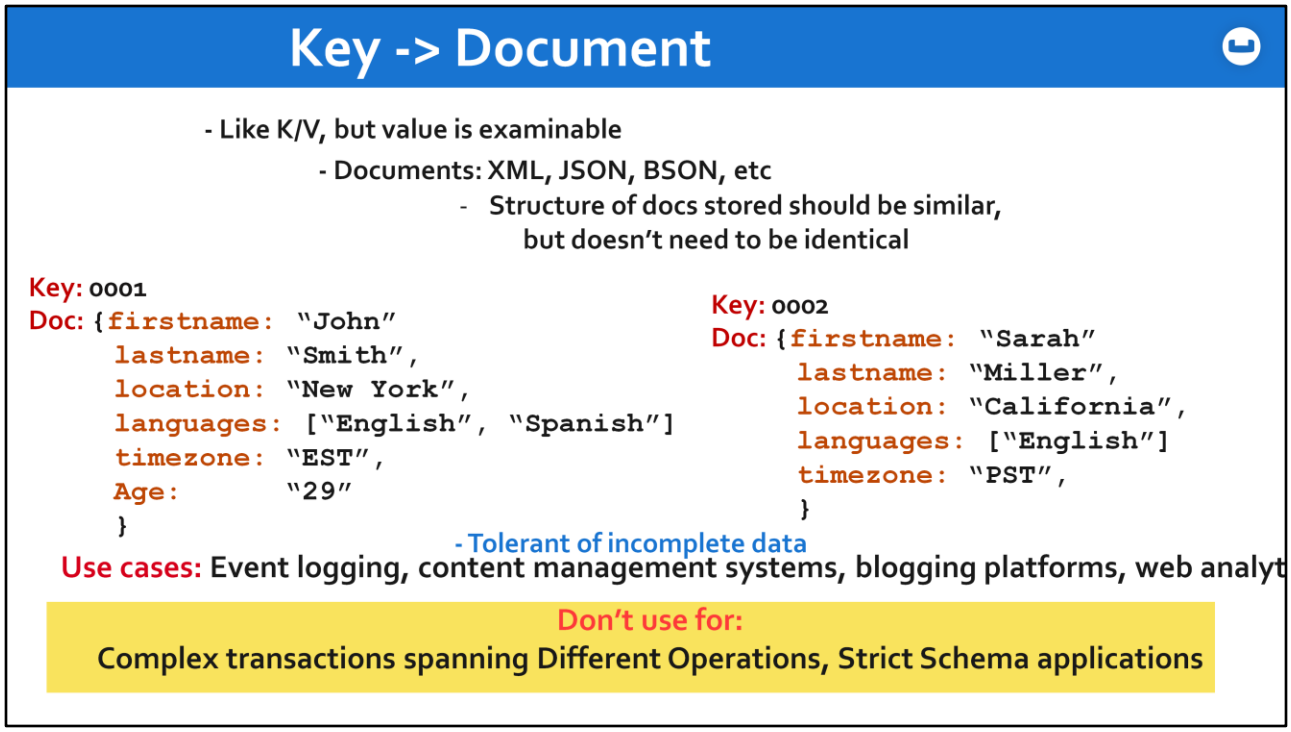


A key-value store is a simple hash table, primarily used when all access to the database is via primary key. It's kind of like a traditional RDBMS table, but with only two columns.

In redis, the value can be an aggregate like lists, sets or hashes.

This is the simplest data store from an API perspective: get, put, delete.

Example use case: You're developing a website and when the use logs in, you want to customize the start page for him, like the background color, language and time zone. Since you want the page to load as quickly as possible and only need to look up the preferences via the username (primary key), Key/Value databases are a good choice.



The documents here are not Word documents or PDF documents….

In MongoDB, the Key-Docs are stored in Collections.

In documents, there are no empty attributes (no NULLs); if a given attribute is not found, we assume that it was not set or not relevant to the document.

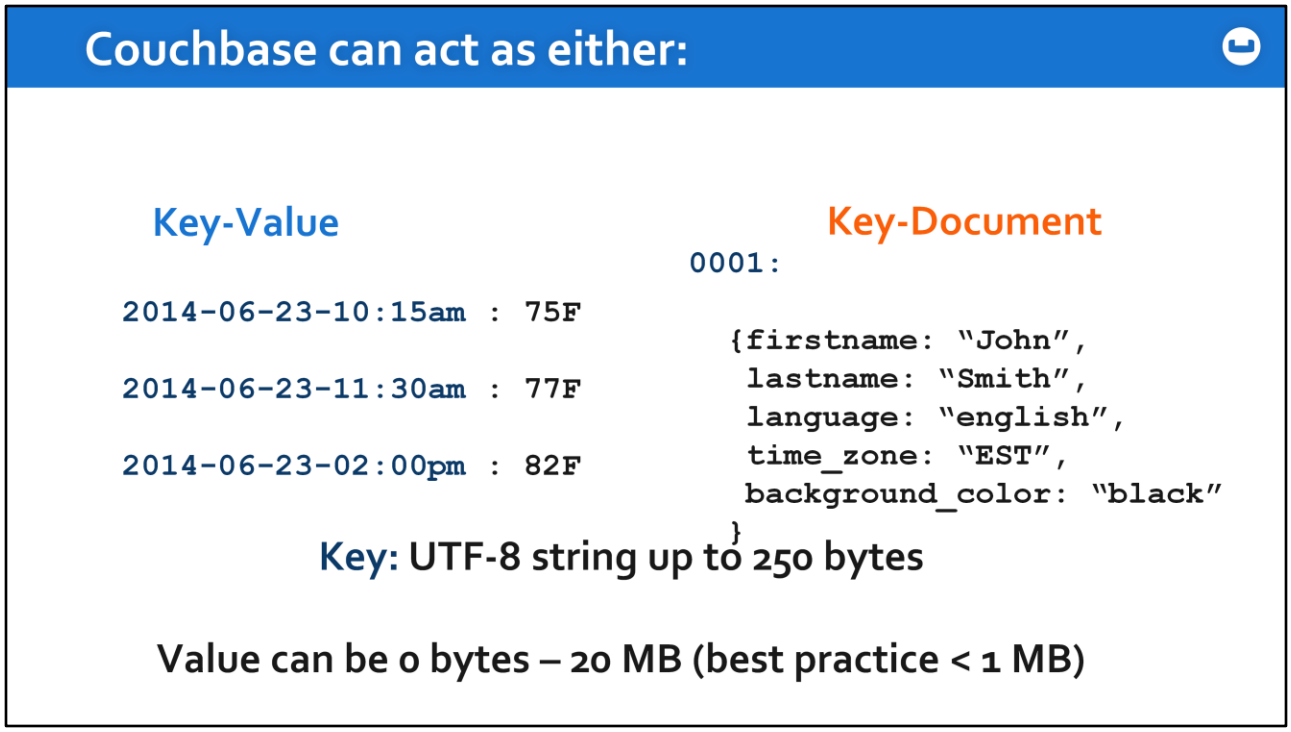
One of the good features of document databases, as compared to key-value stores, is that we can query the data inside the document via an index without having to retrieve the whole document by its key and then introspect the document. So, you can query into the document structure(via an index) and retrieve just portions of the document that have been indexed or update portions of the document(tombstone and append).

Transactions are atomic at the single-document level.

Replication is done in MongoDB via replica sets. So, you write to the master replica, which then asynchronously replicates to a slave replica set.

Event logging: Good use case b/c database can store events from different apps and shard by the name of the application where the event originated like "customer-login" or "purchase-order"

Example application: You are creating software that creates profiles of refugee children with the aim of reuniting them with their families. The details you need to record for each child vary tremendously with circumstances of the event and they are built up piecemeal, for example a young child may know their first name and you can take a picture of them but they may not know their parent's first names. Later a local may claim to recognize the child and provide you with additional information that you definitely want to record but until you can verify the information you have to treat it skeptically.



All information that you store in Couchbase Server are documents with keys. *Keys* are unique identifiers for a document, and *values* are either JSON documents or if you choose the data you want to store can be byte stream, data types, or other forms of serialized objects. Value can be JSON or binary objects, such as integers and strings.

Keys are also known as document IDs and serve the same function as a SQL primary key. A key in Couchbase Server can be any string, including strings with separators and identifiers, such as 'person_93679.' A key is unique.

When Couchbase Server is used as a store for JSON documents, the records can be indexed and queried. Couchbase Server provides a JavaScript-based query engine to find records based on field values.

Couchbase 101

Key: Document ID (like SQL primary key)
Can be any string
Must be unique

Value: Binary (integers, strings, Boolean) or JSON

Metadata: CAS Value (unique identifier for concurrency)
TTL
Flags (optional client library metadata)
Revision #

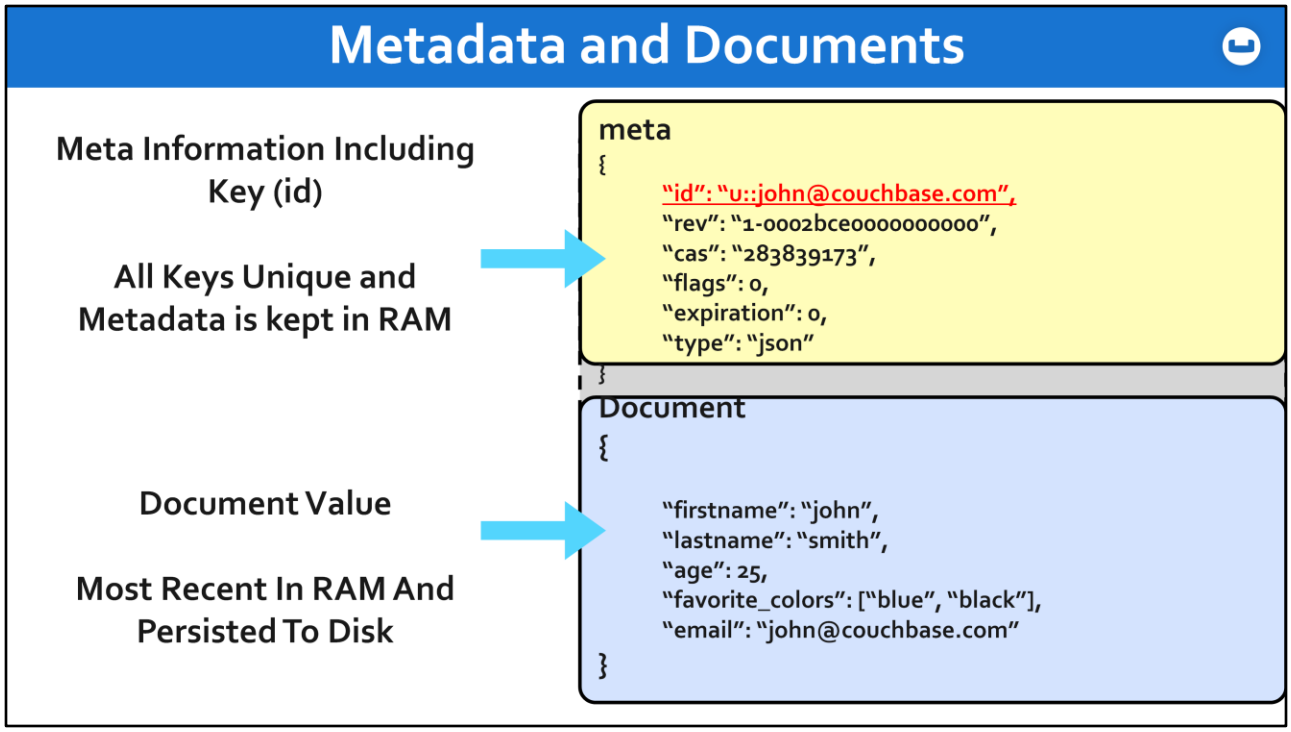
CAS: Also called *cas token* or *cas ID* ; this is a unique identifier associated with a document, and verified by the Couchbase Server before a document is deleted or changed. This provides a form of basic optimistic concurrency; when Couchbase Server checks a CAS value before changing data, it effectively prevents data loss without having to lock records. Couchbase Server will prevent a document from being altered by an operation if another process alters the document and its CAS value, in the meantime.

TTL: This is an expiration for a document typically specified in seconds. By default, any document created in Couchbase Server that does not have a given ttl will have an indefinite life span and will remain in Couchbase Server unless an explicit delete call from a client removes it. The Couchbase Server will delete values during regular maintenance if the ttl for an item has expired.

Flags: These are SDK- specific flags which are used to provides a variety of options during storage, retrieval, update, and removal of documents. Typically flags are optional metadata used by a Couchbase client library to perform additional processing of a document. An example of flags include the ability to specify that a document be formatted a specific way before it is stored.

Benefits of JSON

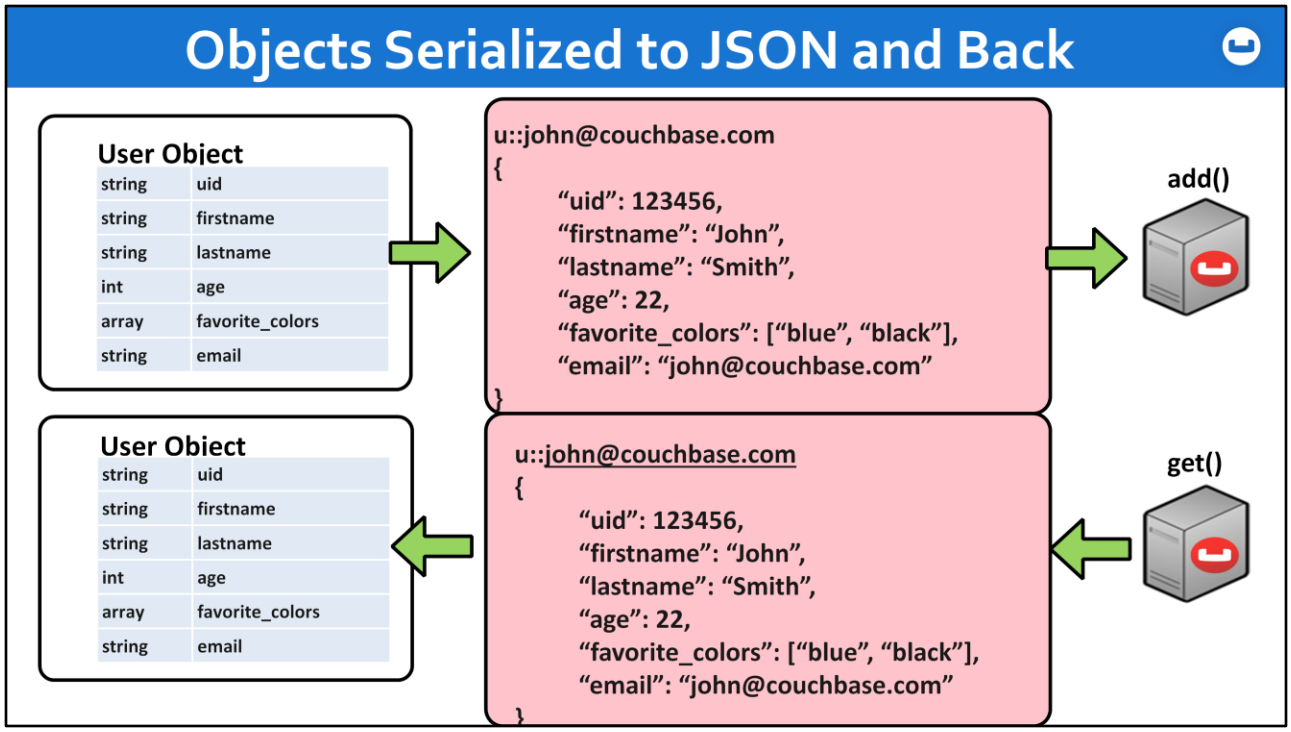
- Can Represent Complex Objects and Data Structures
- Very simple notation, lightweight, compact, readable
- The most common API return type for Integrations
  - Facebook, Twitter, you name it, return JSON
  - Native to Javascript (can be useful)
  - Can be inserted straight into Couchbase (faster development)
- Serialization and Deserialization are very fast



Question: Why would we want to keep the metadata in RAM at all times?

-All of the metadata for the documents and their keys is kept in RAM at all times. While this does add a bit of overhead per item, it also allows for extremely fast "miss" speeds which are critical to the operation of some applications….we don't have to scan a disk to know when we **don't** have some data.

Objects Serialized to JSON and Back



Couchbase Organization

Couchbase operates like a Key-Value Document Store

Simple Datatypes: strings, numbers, datetime (string), boolean, and binary data (string) can be stored

Complex Datatypes: dictionaries/hashes, arrays/lists, can be stored in JSON format (simple lists can be string based with delimiter)

JSON is a special class of string with a specific format for encoding simple and complex data structures

Schema is unenforced and implicit, schema changes are programmatic, done online, and can vary from Document to Document

Store & Retrieve Operations

- get (key)
  - **Retrieve a document**
- set (key, value)
  - **Store a document, overwrites if exists**
- add (key, value)
  - **Store a document, error/exception if exists**
- replace (key, value)
  - **Store a document, error/exception if *doesn't* exist**
- cas (key, value, cas)
  - **Compare and swap, mutate document only if it hasn't changed while executing this operation**

These happen on ports 11210/1

Atomic Counter Operations

These operations are always executed in order atomically.

- incr (key)
  - **Increase an atomic counter value, default by 1**
    - **cb.incr("my_counter") # now it's 2**

- decr (key)
  - **Decrease an atomic counter value, default by 1**
    - **cb.decr("my_counter") # now it's 1**

These also happen on ports 11210/1

Non-JSON Operations

You can use these creatively!

- prepend (key, value)
  - **prepend existing string in couchbase with value**
    - **cb.prepend(“mykey”, "jumped over the lazy dog")**
    - **cb.prepend("mykey", "the brown fox ")**
- append (key, value)
  - **append existing string in couchbase with value**
    - **cb.append(“mykey2”, "oranges")**
    - **cb.append("mykey2", " apples bananas")**

Data Retrieval

- By Key
  - Usually served from RAM(data service node)
  - Strongly consistent
- By Index
  - Usually served from RAM(index service node)
  - Strongly consistent
- By View
  - Always served from disk!
  - Eventually consistent

55

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.

Basic Keying

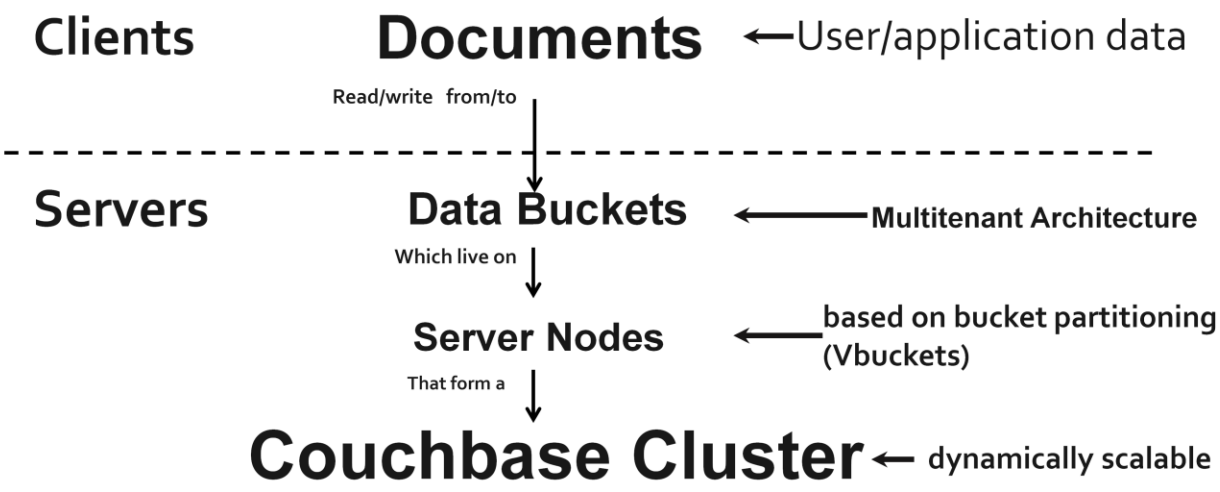
- Use Unique value for key (email, username, sku, isbn, etc.)
  - Users
    - e::john@couchbase.com
    - u::sarah12
  - Products
    - p::978-0321573513 [isbn]
- Predictable Keys can follow Key-Value patterns (Users typically can be done this way and are the most numerous items)
- Unpredictable Keys (GUID, UUID, etc.) require Indexes (GSI) or Views (Map-Reduce Indexes) to find their documents

- **Basic Keying**

- **Use Unique value for key (email, username, sku, isbn, etc.)**
- **Users**
- **e::john@couchbase.com**
- **u::sarah12**

- **Products**
- **p::978-0321573513 [isbn]**
- **Predictable Keys can follow Key-Value patterns (Users typically can be done this way and are the most numerous items)**
- **Unpredictable Keys (GUID, UUID, etc.) require Views (Map-Reduce Indexes) to find their documents**

Key Concepts



Supported SDK's

www.couchbase.com/communities


Java


.NET


PHP


Golang


Python


C


node.js

- Each supported SDK page has instructions for setup
- PHP, and Python clients are wrappers around libcouchbase C library, so libcouchbase must be installed *first*
- For other community clients, click on "All Clients" on left nav, scroll down the page and you can see clients for Go, Erlang, C, TCL, Perl and others.

Connectors & Plugins

http://developer.couchbase.com/documentation/server/4.5/connectors/intro.html

Elasticsearch **Hadoop**

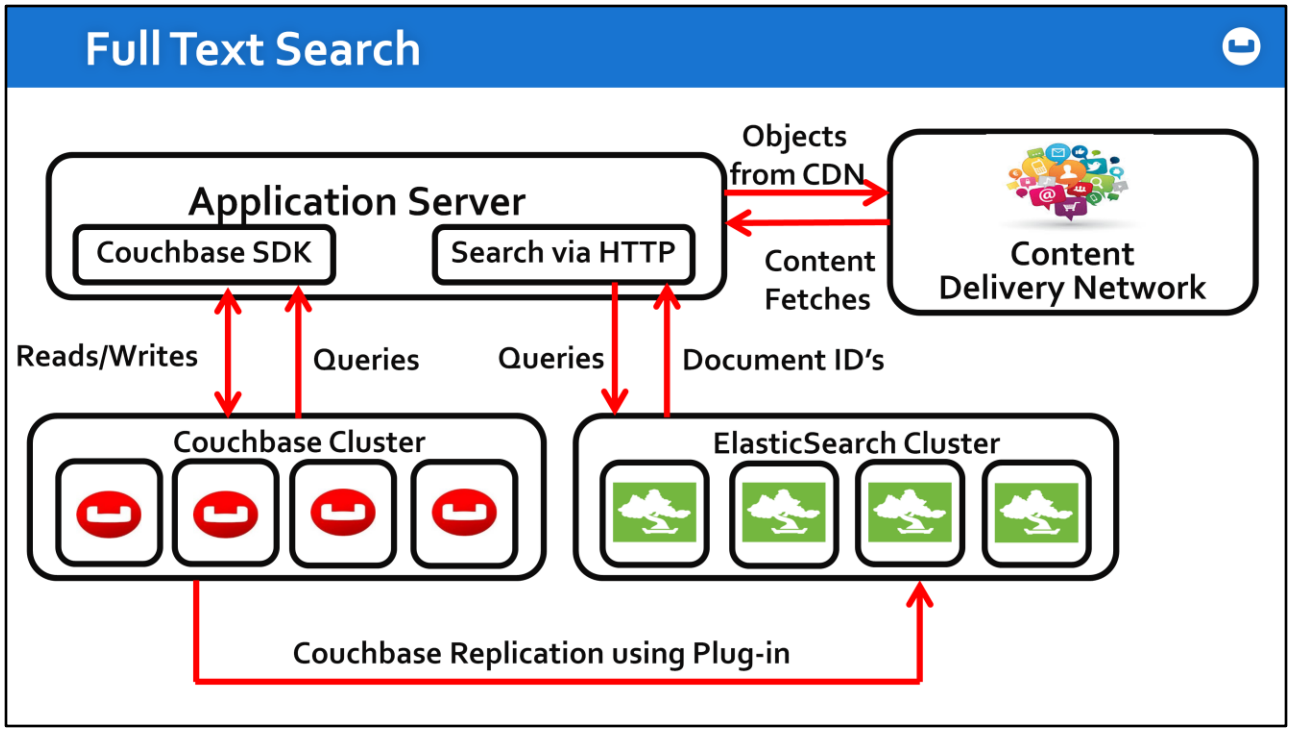
Kafka **Spark**

Talend

Couchbase ODBC

& JDBC Drivers

59



Full Text Search

Integration with separate Elastic search cluster, using XDCR technology

Robust, so will efficiently cope with node failures rebalances or interrupted connections to keep the full text index in sync

Elastic search is a very fast JSON document based full text indexing open source solution, based on Apache Lucene (the same as used by SOLR that more people will know)

Elastic search is also clustered and scales easily and provides very flexible and powerful full text search capabilities

Elasticsearch plug-in

The Couchbase plug-in for Elasticsearch enables you to provide full-text search in your application using the open source search engine, Elasticsearch. This means that your users can retrieve application documents from Couchbase Server based on text in your documents. For instance if you provide a product catalog, users can find items based on text descriptions of the products.

You use Couchbase Server with Elasticsearch to provide quality, rapid full-text search results. The data model for Elasticsearch is already very compatible with the schema-free, document-oriented model used by Couchbase Server. Since search is often a

more CPU-intensive process, you can scale your Elasticsearch cluster separately from your Couchbase cluster to best meet the demands of your users. In doing so, search functions will not slow the performance of Couchbase Server reads or writes.

Your website or web application interacts with Couchbase Server via a Couchbase SDK. These SDKs are provided in a variety of popular web programming languages and are responsible for establishing a connection with the server and for communicating reads/writes and other functions with the server. As mentioned earlier you can also index and query data from Couchbase Server using views and your Couchbase SDK.

To provide full-text search with Couchbase Server you need to have a cluster of Elasticsearch engines, the Couchbase Plug-in for Elasticsearch, and a running Couchbase cluster. After an application writes or updates data in Couchbase Server, the server replicates a copy of that data to the Elasticsearch cluster for indexing. The Elasticsearch cluster will perform indexing based on text content in your data; then via an Elasticsearch client, you send a search query to Elasticsearch via HTTP. Elasticsearch does not keep an entire copy of each item replicated from Couchbase cluster. After Elasticsearch indexes an item it keeps the ID for the item and other metadata, but discards the content to remain efficient. After your application queries Elasticsearch for an item via HTTP, it will send back document IDs which you can use to retrieve the entire document from Couchbase Server.

ElasticSearch cluster is fed the documents from the Couchbase Server cluster
Elasticsearch indexes the fields(configurable which ones) and by default will only store references back to the document id

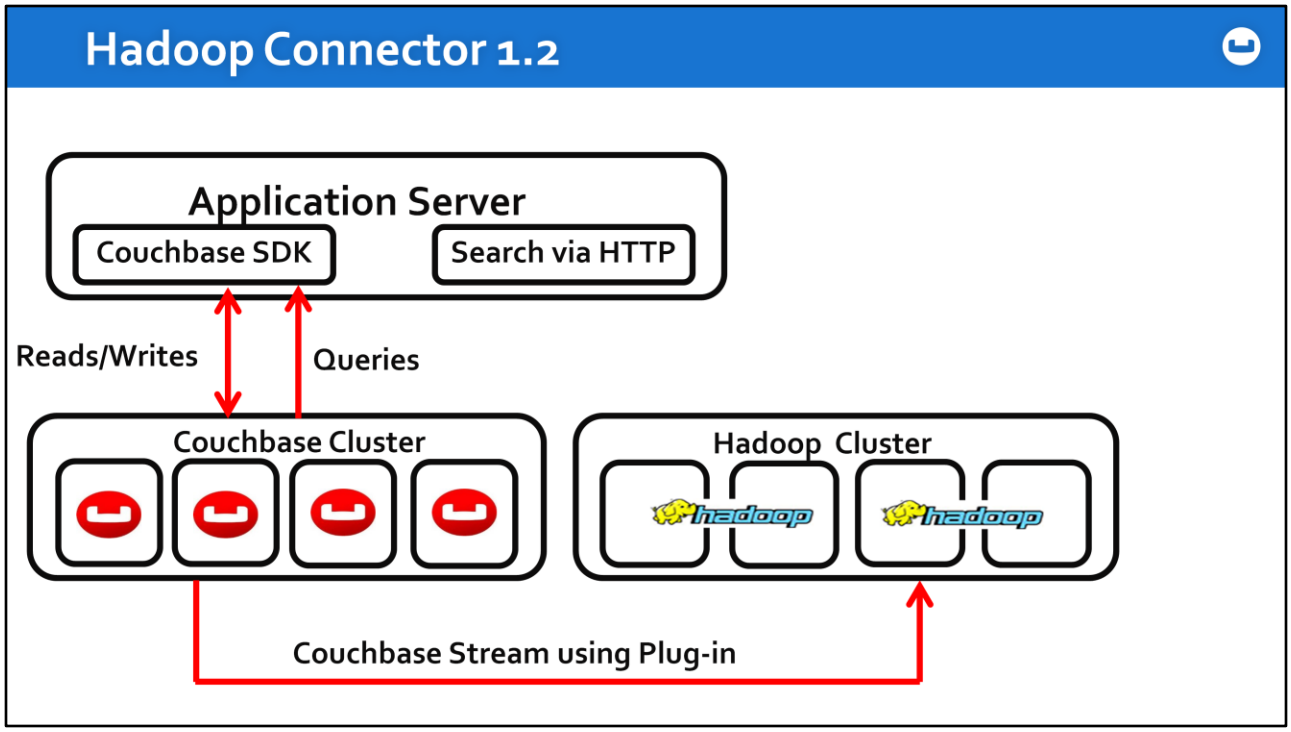
The application does document access via the Couchbase Server Cluster and uses The Views and incremental map reduce on the Couchbase cluster.

For full text queries it queries the Elasticsearch cluster directly (simple Http and JSON interface)

The full text queries typically returns the ids of the matching documents.

Documents are then retrieved from the Couchbase Server cluster.

This way the high throughput document access always comes from high performance Couchbase Cluster.

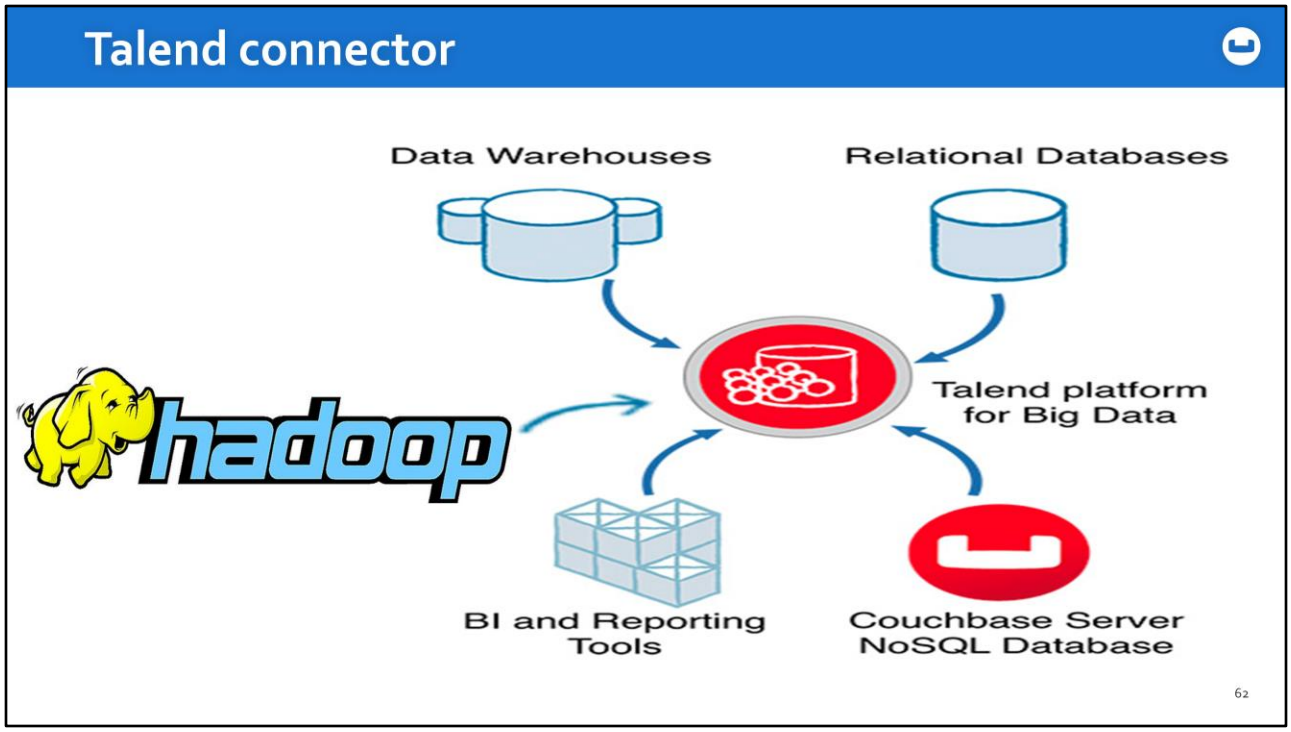


Hadoop connector 1.2
The Couchbase Hadoop Connector allows you to connect to Couchbase Server 2.5 or 3.0 to stream keys into Hadoop Distributed File System (HDFS) or Hive for processing with Hadoop. If you have used Apache Sqoop before with other databases, using this connector should be straightforward because it uses a similar command line argument structure. Some arguments might seem slightly different because Couchbase has a very different structure than a typical RDBMS.
Getting started
Download

Download the Couchbase Hadoop Connector version 1.2.0 from http://packages.couchbase.com/clients/connectors/couchbase-hadoop-plugin-1.2.0.zip.
Requirements

The Couchbase Hadoop Connector is supported on Cloudera 5. Cloudera has certified the Couchbase Hadoop Connector 1.2 release for Cloudera 5.

The Couchbase Hadoop Connector is supported on Hortonworks Data Platform (HDP) 2.2. Hortonworks has certified the Couchbase Hadoop Connector 1.2 release for HDP 2.2.



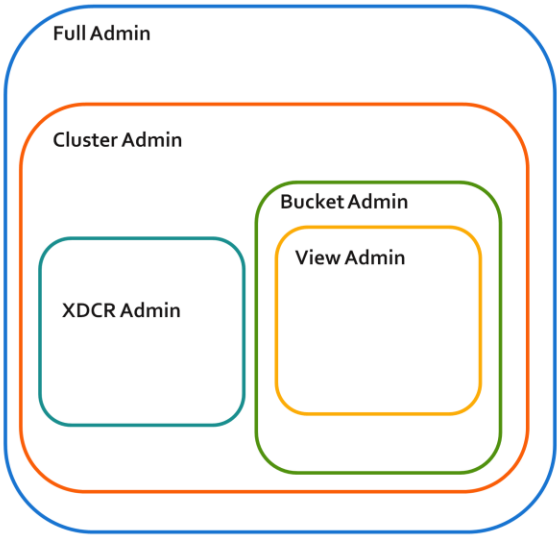
Talend connector
The Couchbase Talend connector enables you to manage and transform your data between Couchbase Server and other data stores in your enterprise.
High-level architecture

The Talend connector for Couchbase connects Couchbase to the Talend big data platform. You can use the Talend connector for Couchbase to move data between Couchbase and any other data store. The connector consists of two components – tCouchbaseInput, the input component, and tCouchbaseOutput, the output component. To bring data from other data sources into Couchbase, tCouchbaseInput takes incoming data streams and transforms them into JSON documents before they are stored in Couchbase. To import data into Couchbase, define which data fields need to be transformed into JSON attributes. Similarly, to export data from Couchbase to other data sources, the tCouchbaseOutput connector uses the schema mapping specified by the user to read JSON documents and transform them into target data formats.

Administrative Role Based Access Control

Authentication via existing LDAP
Authorization via six roles

- **Full** – control all settings, including security roles
  - **Read Only** – view all settings
- **Cluster** – control all cluster settings, except security roles
- **Bucket** – control all bucket settings, except XDCR creation
- **View** – define/run map-reduce and spatial views on a bucket
- **Replication** – configure XDCR topology and replications with no other admin access



Administrative RBAC enables segregation of admin duties
Administrative users can be mapped to out-of-the-box roles
Separation of security administration from full administration
Roles pre-defined with permissions for specific resources
Full Admin
Cluster Admin
Bucket Admin
View Admin
XDCR Admin

Security

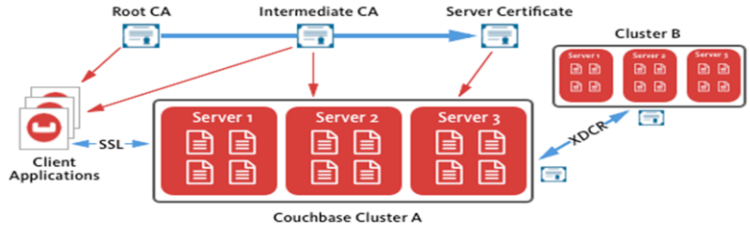
X509 Client-Server Certificates

Bring-Your-Own Certificate Authority

Support for commercially and internally issued X.509 certificates

Strongly CA trusted on-the-wire encryption for applications and between data-centers (XDCR)

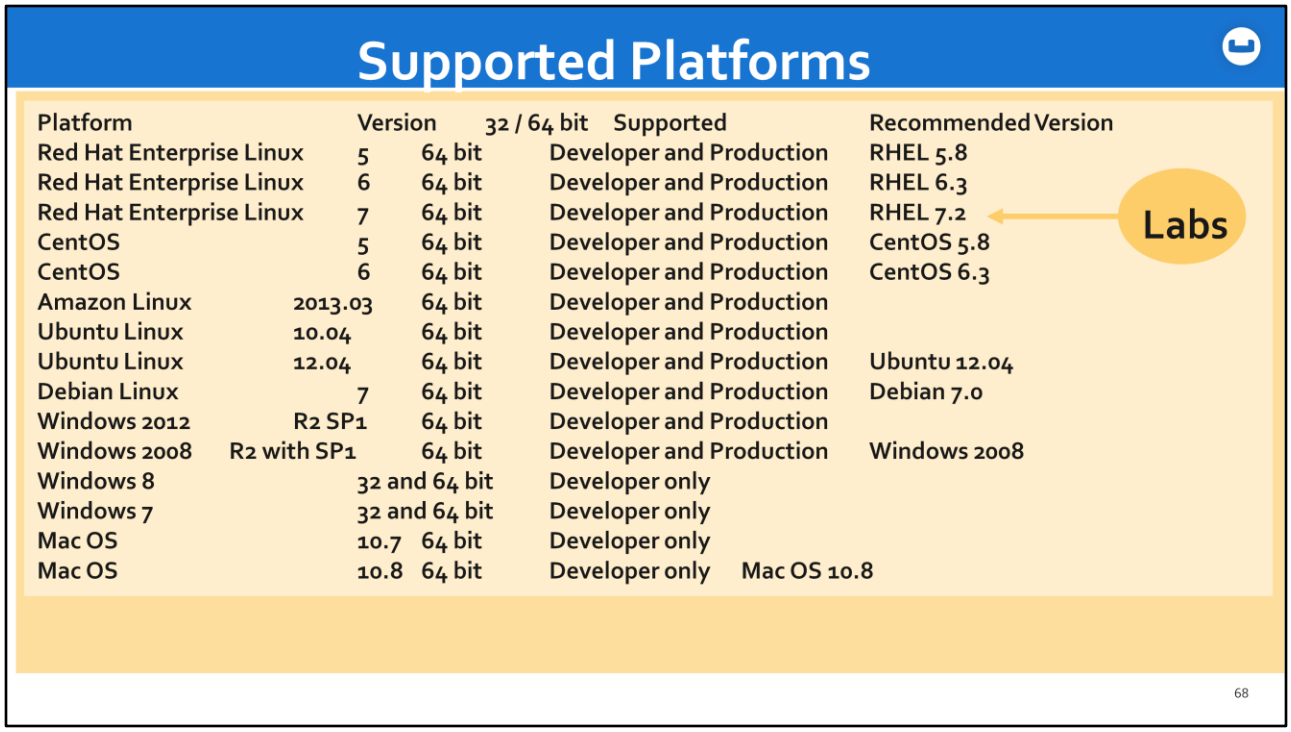
Simplified certificate management and rotation



Security with Couchbase

- **Administrative Security:**
  - Admin user (full access) vs. Read-only user (monitoring, developer access)
  - SSL encryption to REST API and Web UI
  - HTTP access log
- **Data Security:**
  - Applications connect via SASL with single user/pass
  - Data-at-Rest encryption via partnership with Vormetric
  - SSL encryption for over-the-wire
- **Couchbase 4.0:**
  - LDAP/Kerberos integration
  - More extensive administrative action auditing

Installation

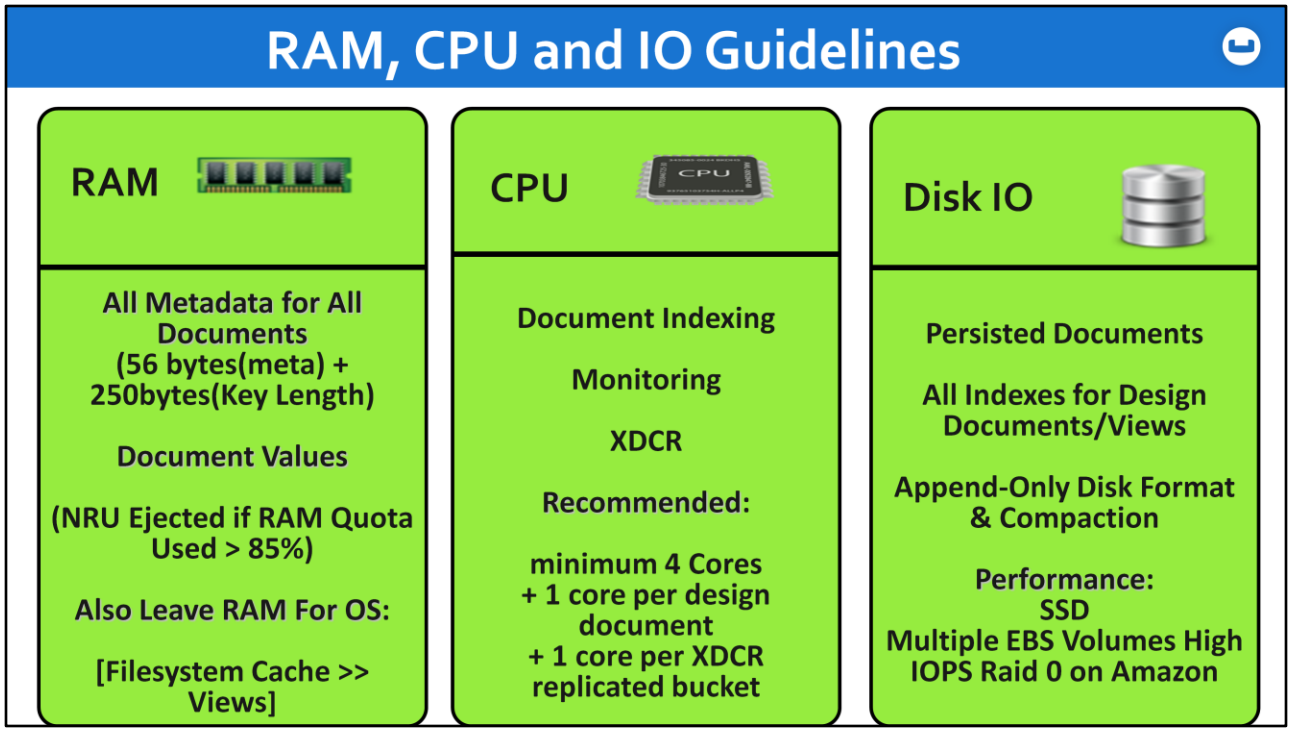


Supported platforms

System requirements for supported platforms.

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

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



Hardware Requirements

Recommended:
Quad-core 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.

Minimum:
Dual-core CPU running at 2GHz
4GB RAM (physical)

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.

Limits for sizing

Limit	Value
Max key length	250 bytes
Max value size	20 MB
Max data size	none
Max metadata	Approximately 56 bytes per document
Max Buckets per Cluster	10
Max View Key Size	4096 bytes

Network ports

Port	Description	Node to Node	Node to Client
8091	Web Administration Port	Yes	Yes
8092	Couchbase API Port	Yes	Yes
11207	Internal/Ext Bucket Port for SSL		Yes
11209	Internal Bucket Port	Yes	No
11210	Internal/External Bucket Port	Yes	Yes
11211	Client interface (proxy)	No	Yes
11214	Incoming SSL Proxy	No	No
11215	Internal Outgoing SSL Proxy	No	No
18091	Internal REST HTTPS for SSL	No	No
18092	Internal CAPI HTTPS for SSL	No	No
4369	Erlang Port Mapper (epmd)	Yes	No
21100 to 21299 (inclusive)	Node data exchange	Yes	No

Note: You can also create user-defined ports

71

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.

Full details: http://docs.couchbase.com/couchbase-manual-2.5/cb-install/#network-ports

User defined ports: http://docs.couchbase.com/couchbase-manual-2.5/cb-install/#using-user-defined-ports

Please note that you have to update your firewall configuration to allow connections to the following ports: 11211, 11210, 11209, 4369, 8091, 8092, 18091, 18092, 11214, 11215 and from 21100 to 21299.

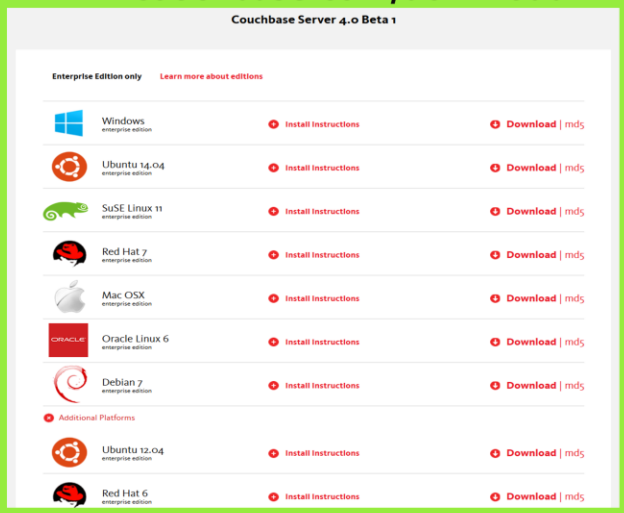
Installation/Initial Setup

- Unattended/programmatic installation
  - **Command-line or REST API**
- GUI Installer:
  - **Quickstart Wizard once installed: http://<hostname>:8091**

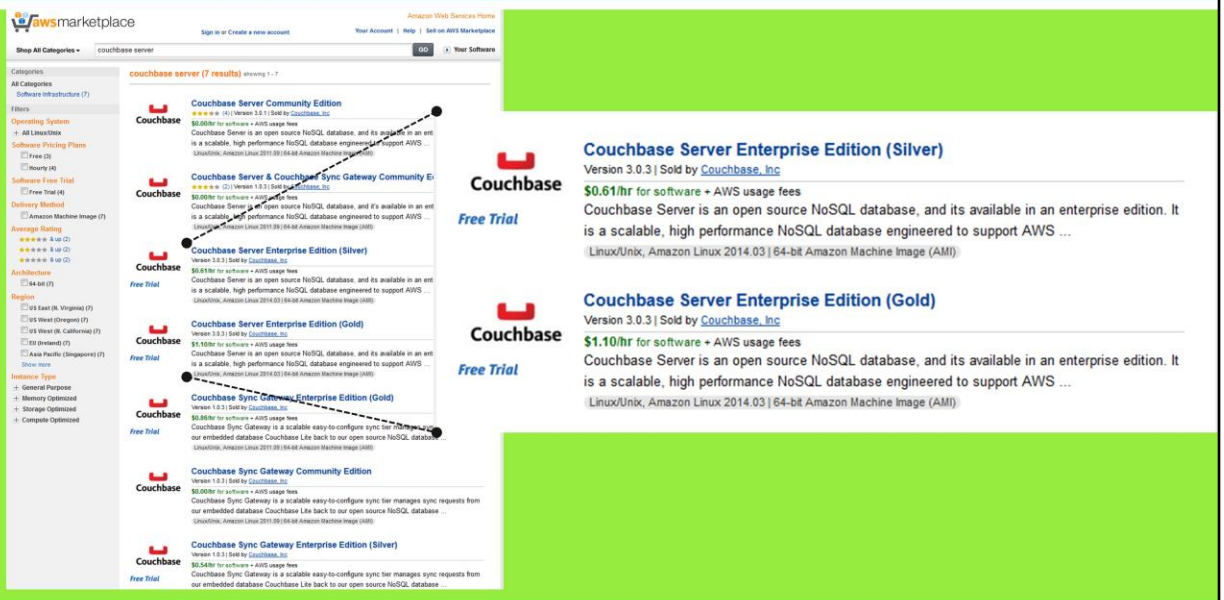
/opt/couchbase/etc/couchbase/static_config

Download Couchbase

www.couchbase.com/download



Install on Amazon via AMI



Download Couchbase 4.5.0 EE:

```
[ec2-user@ip-172-31-19-30 ~]$ wget
http://packages.couchbase.com/releases/4.5.0/couchbase-server-enterprise-4.5.0-
centos7.x86_64.rpm
--2016-07-19 16:25:27--  http://packages.couchbase.com/releases/4.5.0/couchbase-
server-enterprise-4.5.0-centos7.x86_64.rpm
Resolving packages.couchbase.com (packages.couchbase.com)... 54.231.49.244
Connecting to packages.couchbase.com (packages.couchbase.com)|54.231.49.244|:80...
connected.
HTTP request sent, awaiting response... 200 OK
Length: 111171784 (106M) [application/x-redhat-package-manager]
Saving to: 'couchbase-server-enterprise-4.5.0-centos7.x86_64.rpm'

100%[=============================================================================
====================================>] 111,171,784 44.7MB/s   in 2.4s

2016-07-19 16:25:29 (44.7 MB/s) - 'couchbase-server-enterprise-4.5.0-
centos7.x86_64.rpm' saved [111171784/111171784]
```

Install Couchbase:

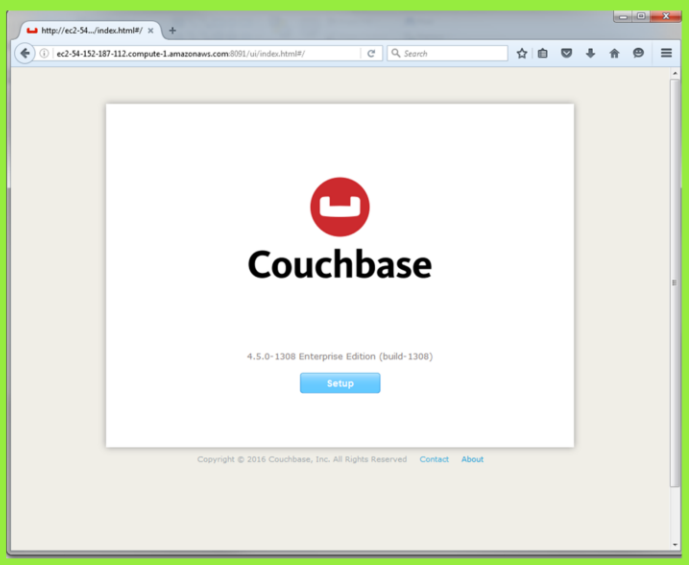
```
[ec2-user@ip-172-31-19-30 ~]$ sudo rpm --install couchbase-server-enterprise-4.5.0-
centos7.x86_64.rpm
Minimum RAM required  : 4 GB
System RAM configured : 3.45 GB

Minimum number of processors required : 4 cores
Number of processors on the system    : 1 cores
Starting couchbase-server[  OK  ]

You have successfully installed Couchbase Server.
Please browse to http://ip-172-31-36-52:8091/ to configure your server.
Please refer to http://couchbase.com for additional resources.

Please note that you have to update your firewall configuration to
allow connections to the following ports: 11211, 11210, 11209, 4369,
8091, 8092, 18091, 18092, 11214, 11215 and from 21100 to 21299.

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

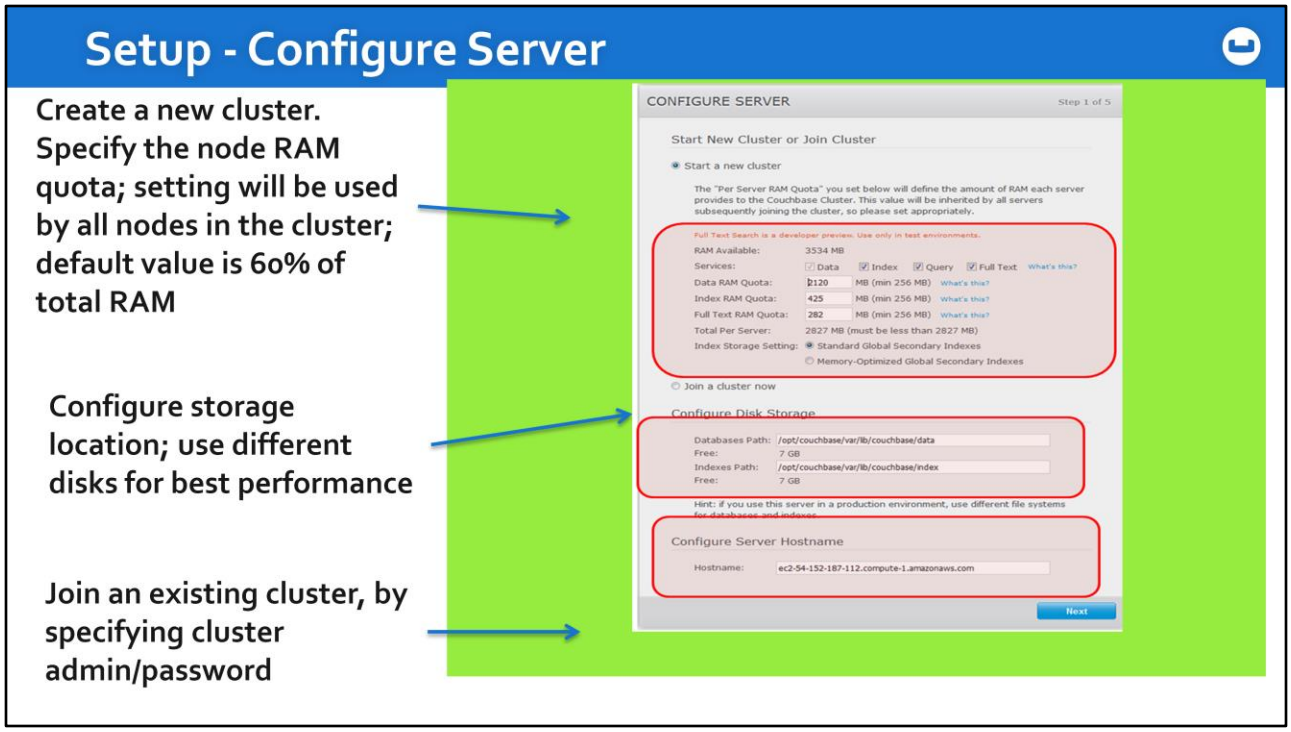
Be aware of Amazon use of inside addresses which are not normally resolvable from outside

76

In your lab environment the displayed address for “Please browse to….” will use the inside Amazon address.
This will not resolve in the outside world.
Please use your assigned EC2-w-x.y.z.amazonaws.com addresses

Initial login screen on first node and subsequent





In Windows, the data files by default are at:
C:\Program Files\couchbase\server\var\lib\couchbase\data

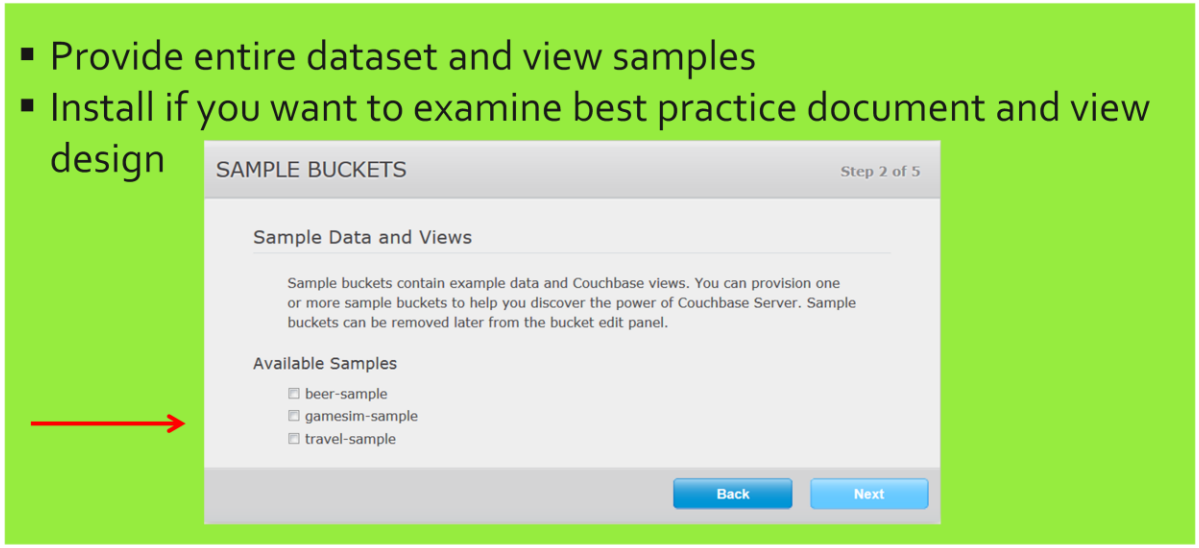
The Configure Server Memory section sets the amount of physical RAM that will be allocated by Couchbase Server for storage.

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

The default value is 60% of your total RAM. This figure is designed to allow RAM capacity for use by the operating system caching layer when accessing and using views.

Services
Data: enter data service ram quota, holds data, views, and copies of data
Index: enter Index service ram quota, holds indexes,
Query: run query service, ram automatically calculated and set.

Setup – Sample Buckets



- Provide entire dataset and view samples
- Install if you want to examine best practice document and view design

SAMPLE BUCKETS

Step 2 of 5

Sample Data and Views

Sample buckets contain example data and Couchbase views. You can provision one or more sample buckets to help you discover the power of Couchbase Server. Sample buckets can be removed later from the bucket edit panel.

Available Samples

- beer-sample
- gamesim-sample
- travel-sample

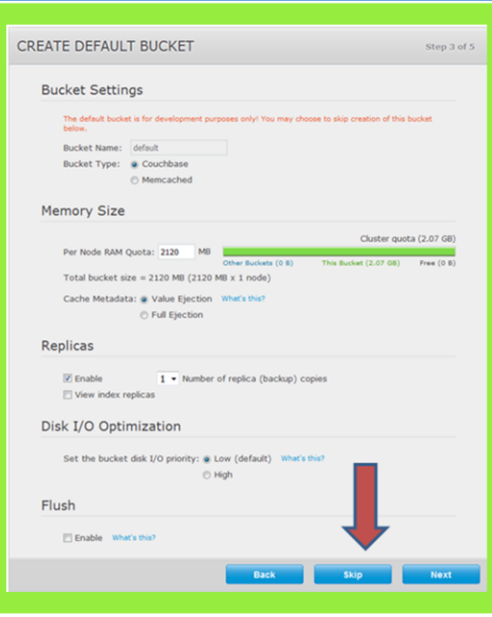
Back Next

Setup – Create a Default Bucket

Create a default bucket for sample data storage.

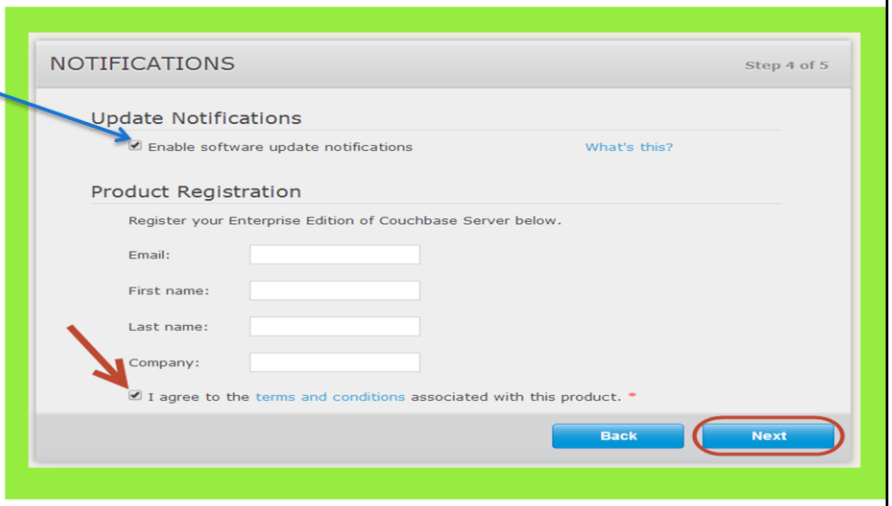
Bucket quota is per node. i.e. four nodes of 1024MB provides 4096MB of RAM quota. Quota is used for caching data

Data and index replicas create replicas of data that are used during failover. Replicas are not created until there are enough nodes to support. Zero disables replicas



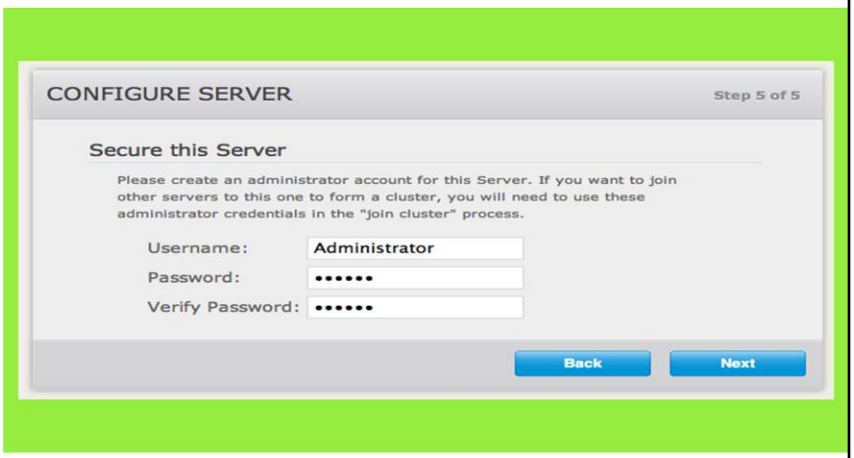
Setup - Notifications

Notifications warn of software updates. Community updates connects you to the Couchbase community newsletter



Setup – Configure Server

Setup the username and password. This will be used for accessing the UI, and required when using the REST API and command-line for administration



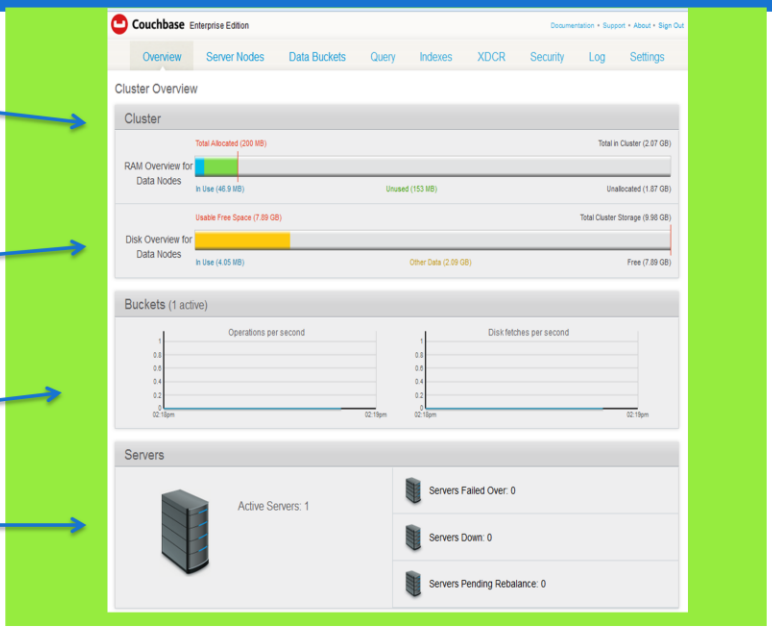
Web UI - Overview

Navigation

RAM and Disk Usage overview. Shows actual usage, remaining quota, and system RAM; shows disk usage, other data and disk free

Bucket activity overview

Node status overview, total servers, failed over count, and pending count



/opt/couchbase/etc/couchbase_init.d start

Starting couchbase-server

/opt/couchbase/etc/couchbase_init.d stop

Stopping couchbase-server

/opt/couchbase/etc/couchbase_init.d status

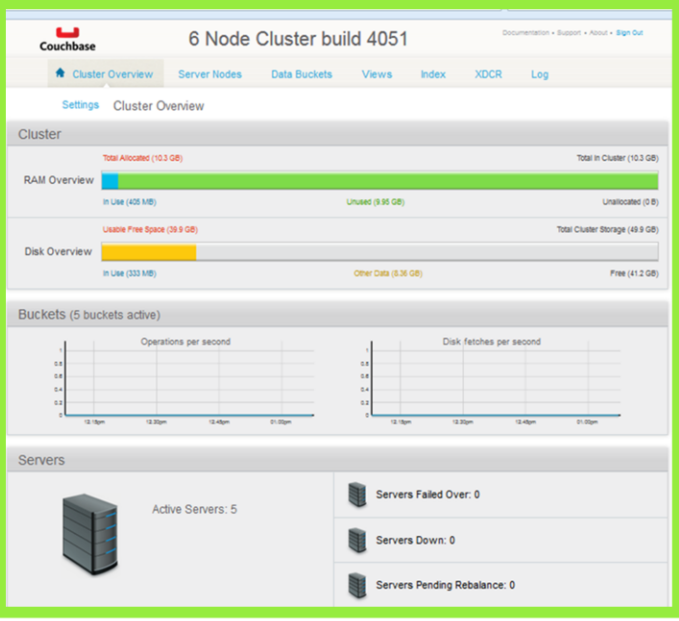
Lab #1: Installation on one node

Time: 1 hour

- Installation of Couchbase 4.5.0
- Introduction to the Web UI
- Couchbase CLI and Rest API
- Data files vs Index files
- cbworkloadgen & pillowfight

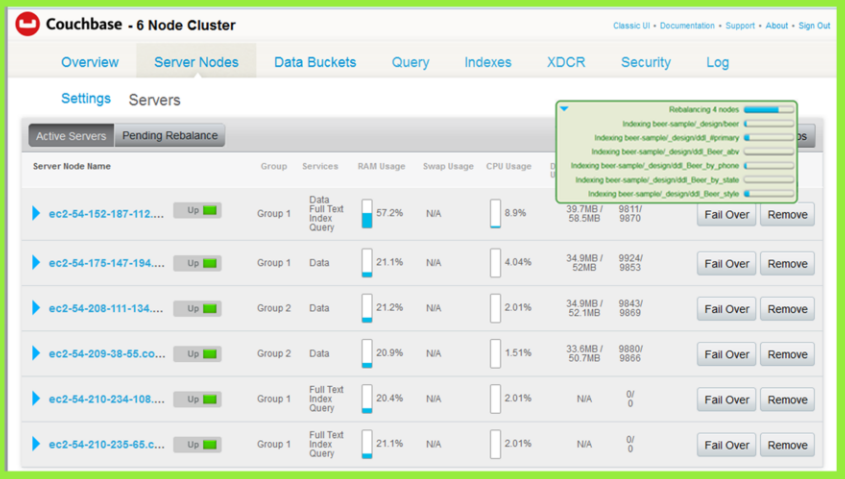
Web UI tour

Initial login screen-Cluster Overview tab

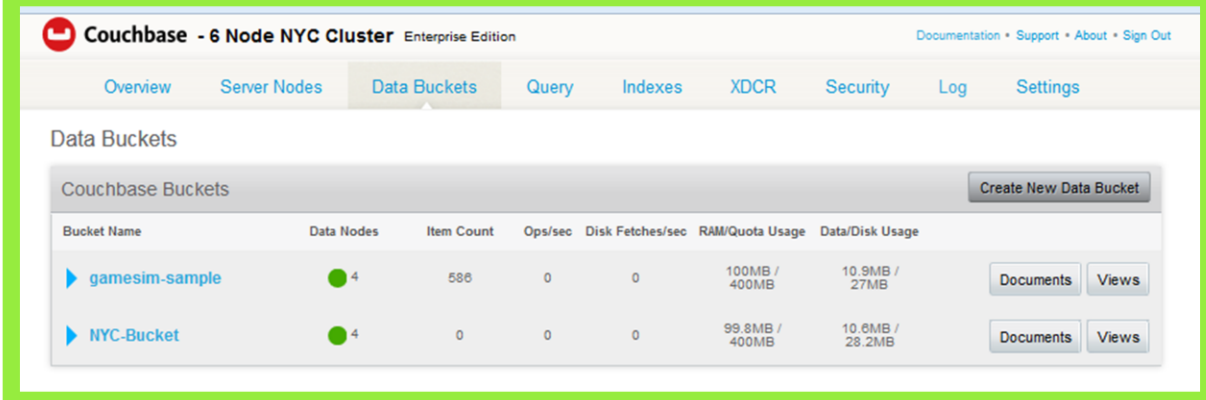


Web UI – Server Nodes Tab

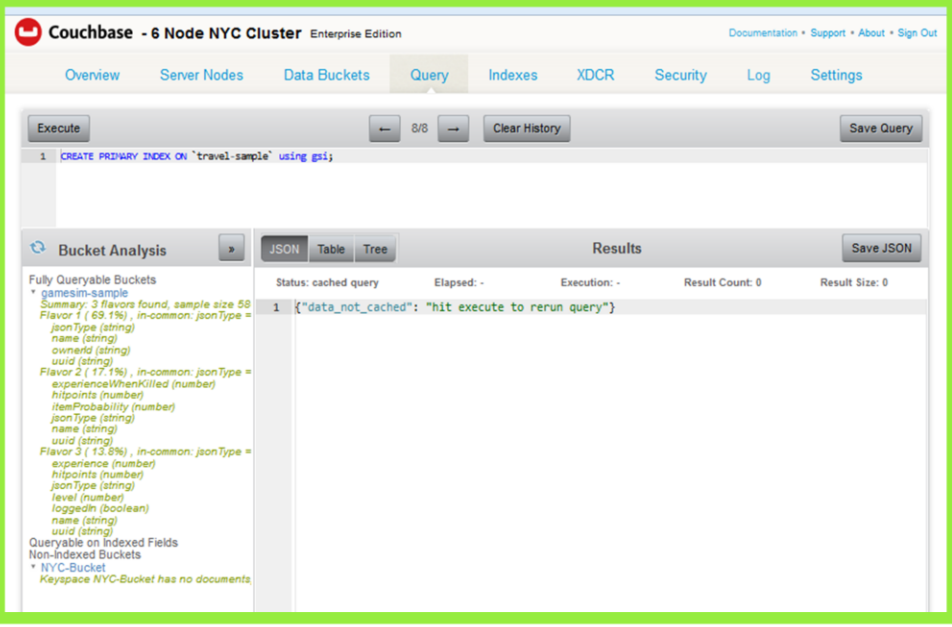
- Main admin interface for adding, removing, failing over and rebalancing your cluster
- Shows overview statistics for each node
- Clicking node triangle shows detailed RAM/disk info
- Client node IP address opens the per-node statics page



Buckets-Databuckets tab



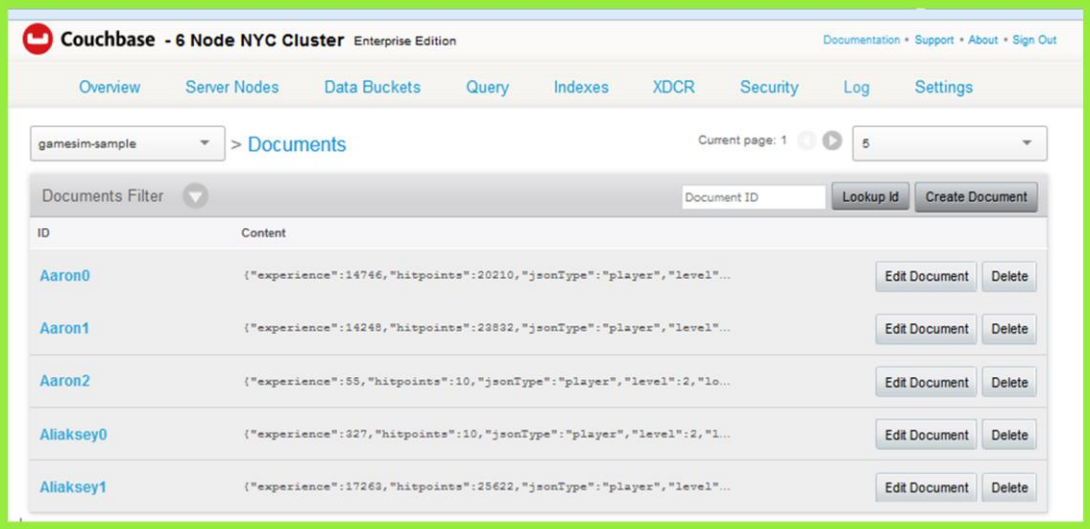
Query workbench tab



Indexes tab

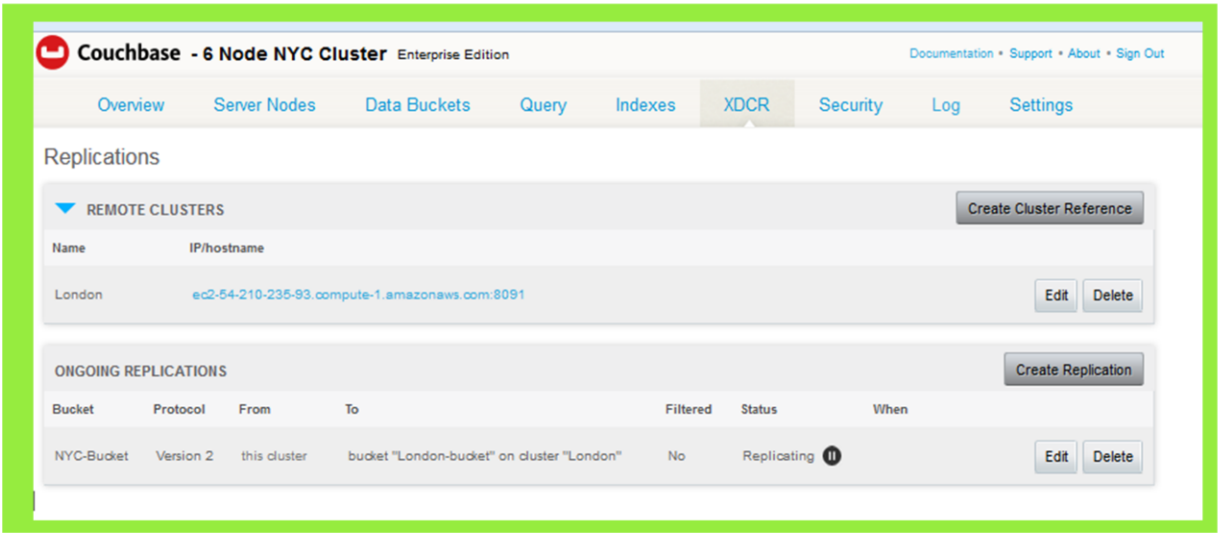

Couchbase - 6 Node NYC Cluster Enterprise Edition
Documentation • Support • About • Sign Out
Overview
Server Nodes
Data Buckets
Query
Indexes
XDCR
Security
Log
Settings
Global Indexes
Views
Full Text
Bucket
Node
Index Name
Storage Type
Status
Initial Build Progress
gamesim-sample
ec2-54-152-187-112.com...
gamesim_primary
Standard Global Secondary In...
Ready
100 %
Definition: CREATE PRIMARY INDEX `gamesim_primary` ON `gamesim-sample`

Specific documents- edit documents button

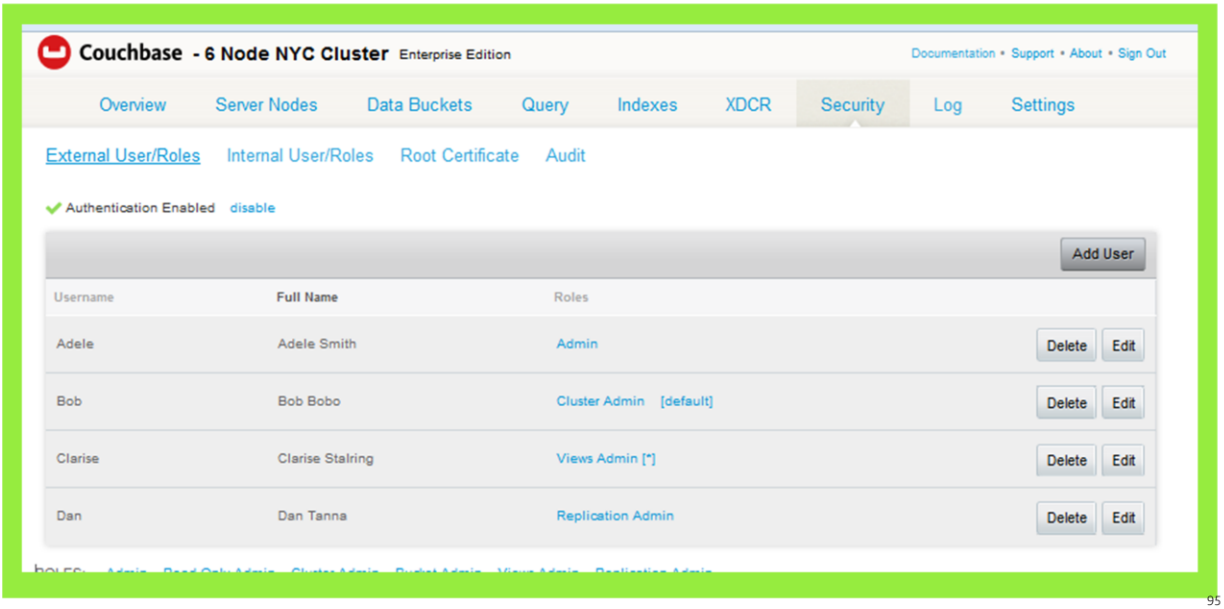


Bucket metrics-bucket name link

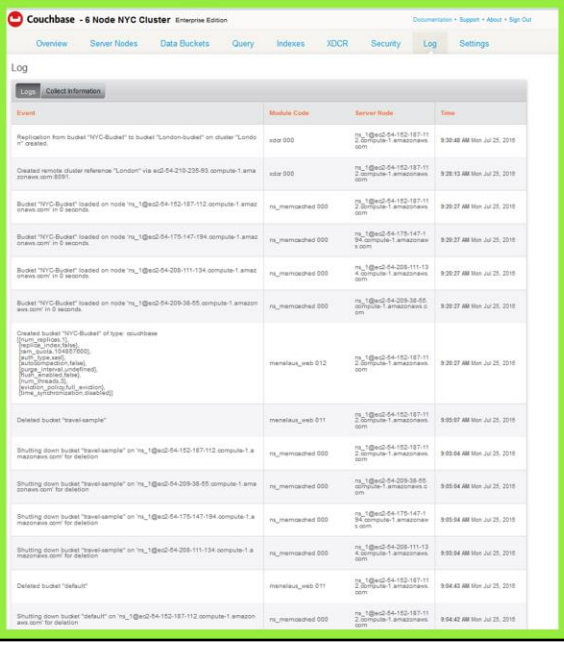
XDCR –XDCR tab



Security tab

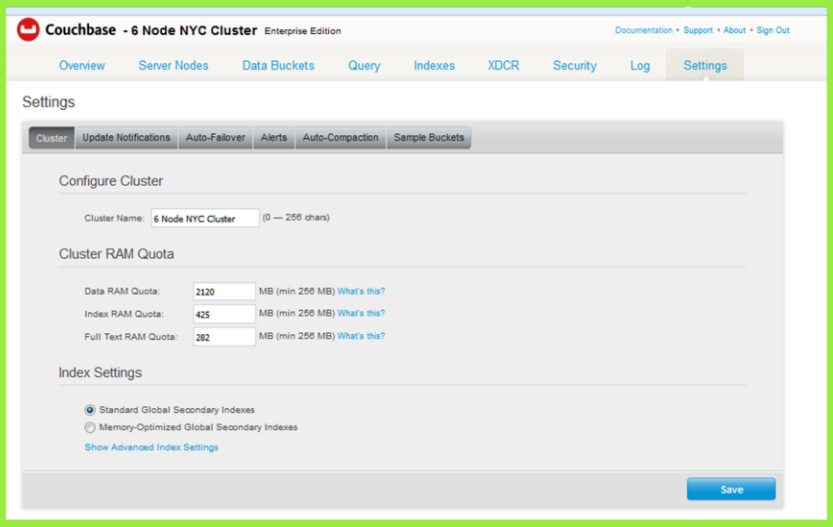


Log information-Log tab



96

Setting/config - Settings tab



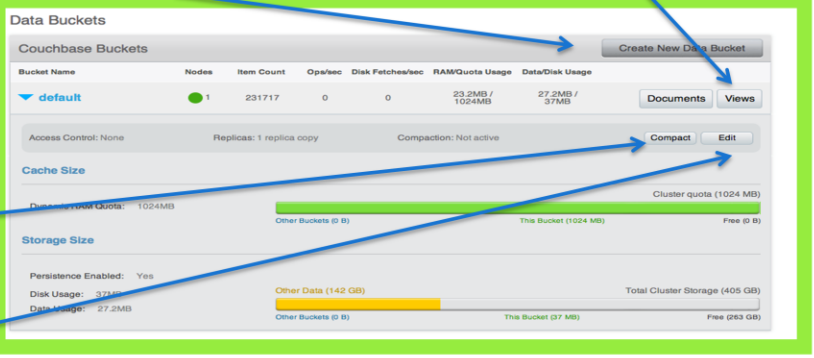
Web UI – Data Buckets

Buckets display shows main overview statistics and usage for each bucket. Monitor RAM/Quota usage, data/disk usage, and detailed stats for each bucket across entire cluster.

Compaction reclaims fragmented disk space
Some bucket parameters can be edited live

Click to display/create/edit documents

Click to display view information



Web UI – Detailed Statistics

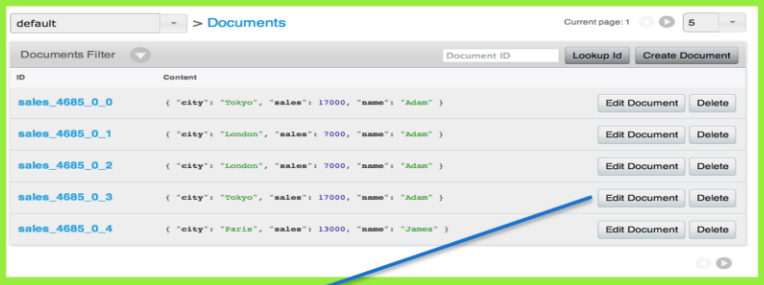
Web UI - Detailed Statistics

Many more statistics available. Smaller, summary versions updated alongside main graph. Clicking a smaller graph makes it the main graph display.

Statistics available for all the components of the system, including disk, vBucket, Views and XDCR

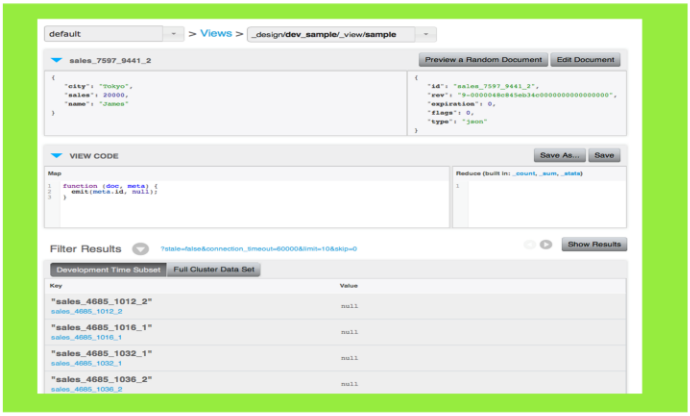
Web UI – Document View

- Documents can be created and edited through the UI



Web UI – View Editor

- Create/edit views
- Examine sample documents to get the correct fields
- Examine document metadata
- Test view output



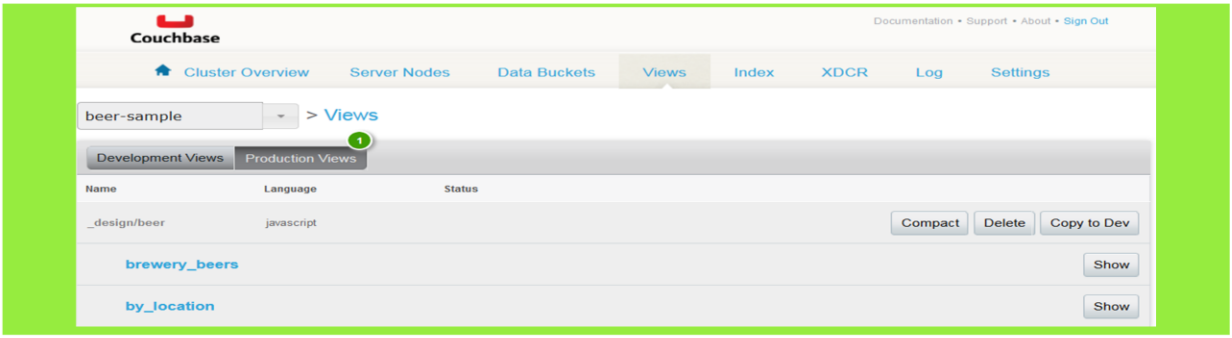
Web UI – XDCR Management

Configure remote clusters. Remote clusters must be identified before they are used.

Create/delete replication configurations to named clusters. Replication monitoring is through statistics interface.

Web UI - Views

- Views manager allows you create design documents/views
- Promote views to production
- Compact on-disk view index information



Web UI – Logs and Settings

- Log shows important cluster-wide errors
  - Use 'Generate Diagnostic Report' to create a file containing data for analysis
- Settings
  - Configure cluster wide configuration
  - Update Notifications (for upgrades)
  - Auto-Failover settings
  - Alerts (including email)
  - Auto-compaction
  - Sample bucket installation

Buckets

Buckets

Virtual Data Partitions
Based on data type or application
Semi-synonymous with “database” (not table)

Separate, unique key spaces
Separate vbucket map
Separate threading and disk queues
Defined and isolated RAM quotas
Separate paths on disk

Buckets can be created/sized/deleted independently

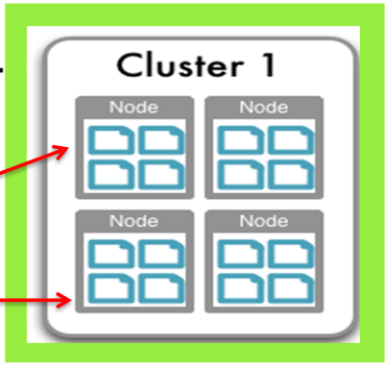
Each bucket sharded across entire cluster

Where does Data Live?

Key value pairs:

- Data lives in a Bucket as key/value pairs.
- Create a named bucket on the cluster.
  - Data managed for you as vBuckets.
- Data is spread around the cluster automatically.
- Each key is mapped to a vBucket on a node.
- Each node has 1 - 1024 vBuckets (1024 / nodes).

Bucket name:
`beer-sample`



Two Kinds of Buckets

Buckets are like a database, or keyspace (namespace):

- Partition data with multiple named buckets
- Security at the named bucket level (bucket name/password)

1. Couchbase Bucket: works with both in memory and disk
   - Asynchronous persistence to disk
   - Data replication (HA) and rebalance at bucket level
   - 20 MB per item value limit, BLOBs allowed[1]
   - 100% protocol compatible with Memcached

2. Memcached Bucket: in memory only
   - No High Availability
   - 1 MB per item value limit

Notes:

1. Consider limitations and ramifications: Good for: shopping carts, user profile, user sessions, time lines, game states, pages, conversations and product catalogs, etc. Not so good for: audio/video binary streaming. Perhaps storage of a JPEG image, or other needed BLOB. Couchbase Client SDKs vary on how they handle compression/decompression, SerDe, encoding and decoding. For example, what if an MSWindows client wrote the data, and a Mac OS Client reads the data? Any Java POJO can be stored if it is Serializable.

Memcached Buckets
Used for pure in-memory-only caching
Off-the-shelf Memcached
Not persistent (no disk storage)
Not replicated
Not dynamically rebalanced
No support for views

Couchbase Buckets
Built-in caching
Persistent
Replicated
Views/Indexing/Querying

Cross Data Center Replication
Dynamically reconfigurable
Memcached API compatible
Use DCP protocol

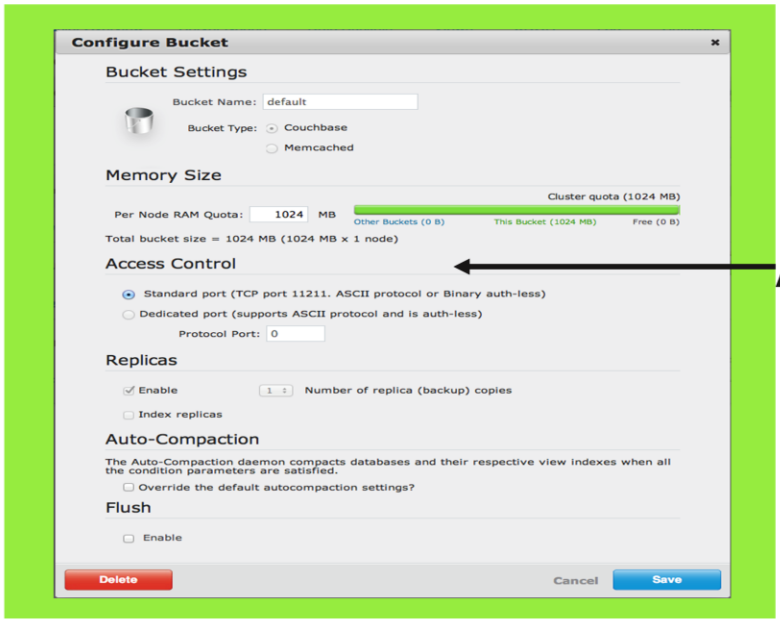
Compare memcache & Couchbase Buckets

memcached

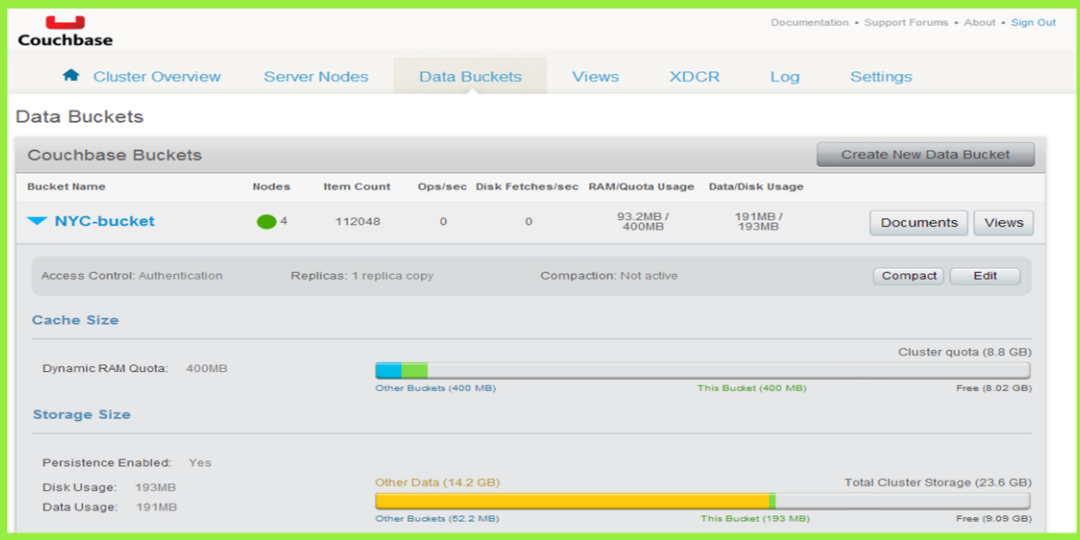
Couchbase

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

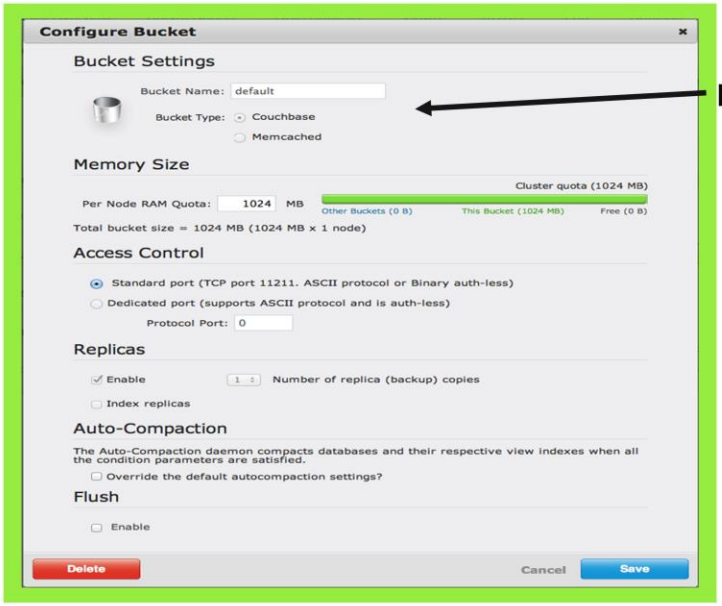
Access Control - Ports



Data Buckets – Bucket config - Blue triangle link



Bucket Name and Type



Specifying an alternate directory path location for a bucket:

(default) /opt/couchbase/var/lib/couchbase/data
gamesim-sample
beer-sample
bucket_1 etc……

(default) /opt/couchbase/var/lib/couchbase/index/@indexes
gamesim-sample
beer-sample
bucket_1 etc……

Linux:
Use system logical volume manager to construct volumes(raid level depends)

For buckets, mount volumes at opt/couchbase/var/lib/couchbase/data/bucket_1
bucket_2 bucket_3 etc…….

With permissions set for owner and group to couchbase (chown)
Edit /etc/fstab to have them mounted at boot

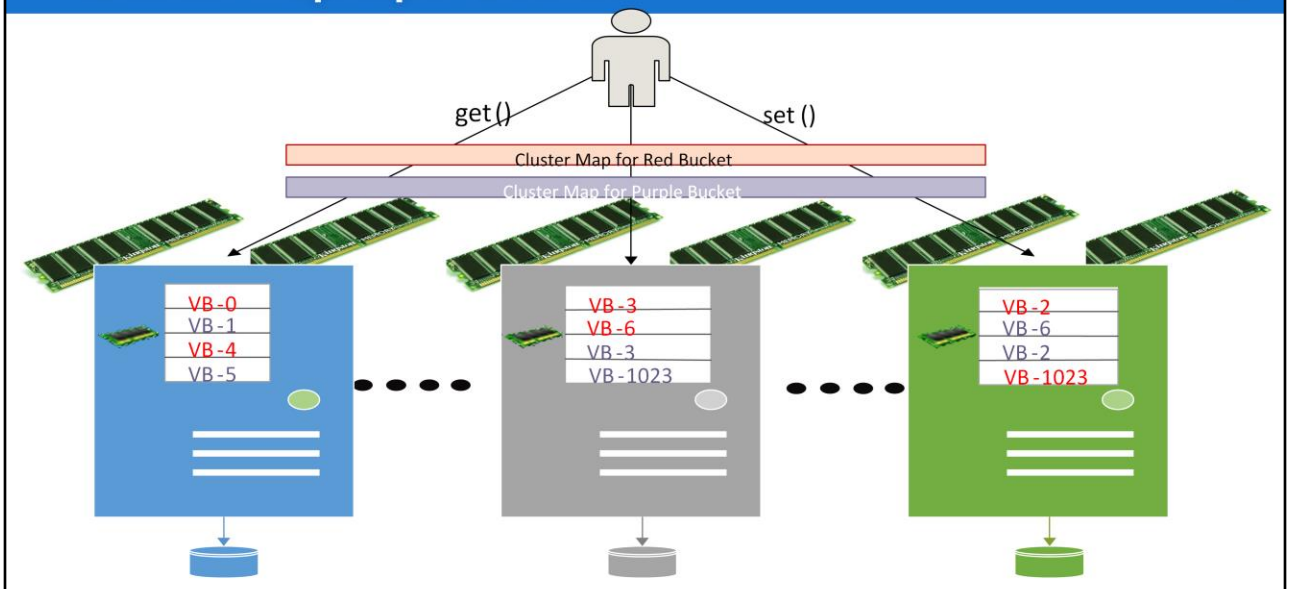
Test

Create bucket using GUI or REST api specifying bucket name (case match)

Create index pathing using the same procedure

Warning: if permissions are set to root:root for bucket directories you may lose contact with the cluster(GUI).

Cluster Map is per Bucket



114

There are 3 interfaces to buckets

Default Bucket: - always resides on port 11211
- non-SASL authenticating bucket
- automatically set up during installation
- can be removed/re-added
- reached via: vBucket aware smart client, an ASCII client or binary client that doesn't use SASL authentication

Non-SASL buckets: - may be placed on any available port except 11211
- only one bucket can be on any port
- reached via: vBucket aware smart client, an ASCII client or binary client that doesn't use SASL

SASL buckets: - may only be placed on port 11211
- many buckets can be on this port
- each bucket is differentiated by its name and password
- reached via: vBucket aware smart client or binary client that supports SASL authentication

115

Simple Authentication and Security Layer (SASL) is a framework for authentication and data security in Internet protocols.
It decouples authentication mechanisms from application protocols, in theory allowing any authentication mechanism supported by SASL to be used in any application protocol that uses SASL.

Update port #s.

Reasons for multiple buckets

To isolate individual applications

To provide multi-tenancy

Enhanced visibility

Isolate datatypes

Improved indexing rebuilding(ongoing , pertinent queries) and need for index rebuilding)

116

You can mix Couchbase and Memcached buckets in the same cluster

Buckets can be used to isolate individual applications to provide multi-tenancy
Or Buckets can isolate data types in the cache to enhance performance and visibility
Quotas for RAM and disk usage are configurable per bucket
Quotas can be modified on a running cluster as usage patterns or priorities change
You can have 10 buckets max (best practice is to keep it under 5)
Quotas can be modified on a running cluster so that administrators can reallocate resources as usage patterns or priorities change over time.

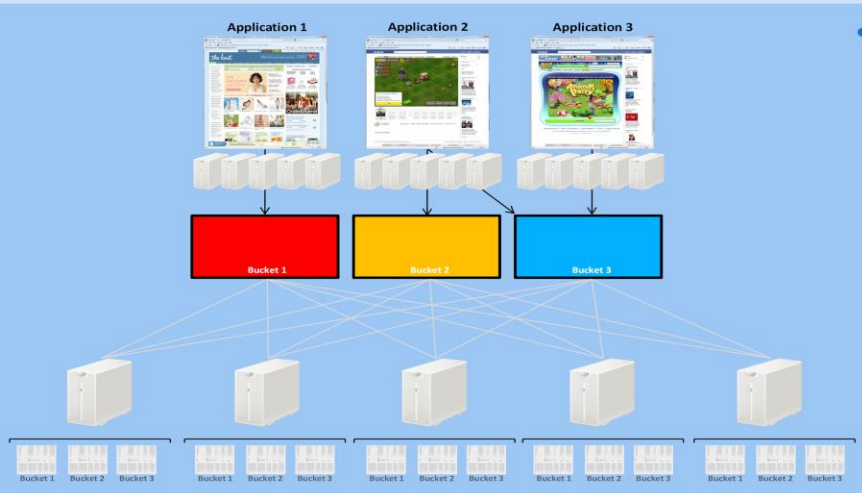
Bucket configurations:

- **RAM quota**
- **Replica count**
- **Disk location (different paths/partitions/devices depending on IO requirements)**
- **Authentication (username/password)**
- **Design Documents/Views (for indexing)**
- **Index replicas**
- **Cross Data Center Replication**
- **Compaction rules for optimizing disk usage**

Traffic is monitored per-bucket

All operations are per bucket
Buckets operate independently per-node
Statistics and behavior are monitored by bucket, per node
Each bucket is treated individually by the system

Multi-tenancy via Buckets

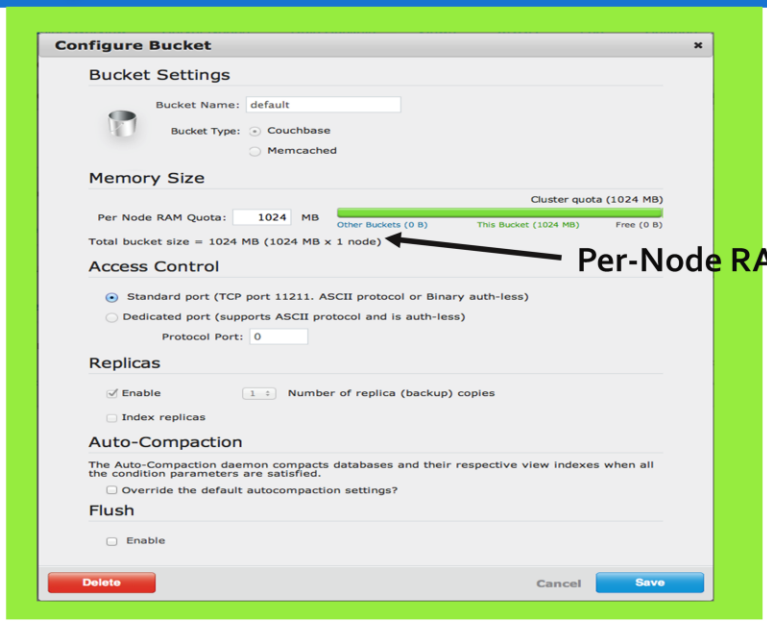


- Multiple buckets:
  - Store multiple object types in the same database for the same application
  - Use multiple buckets when needing separate RAM, replication or view configurations
  - Avoid too many buckets, aim for less than 10

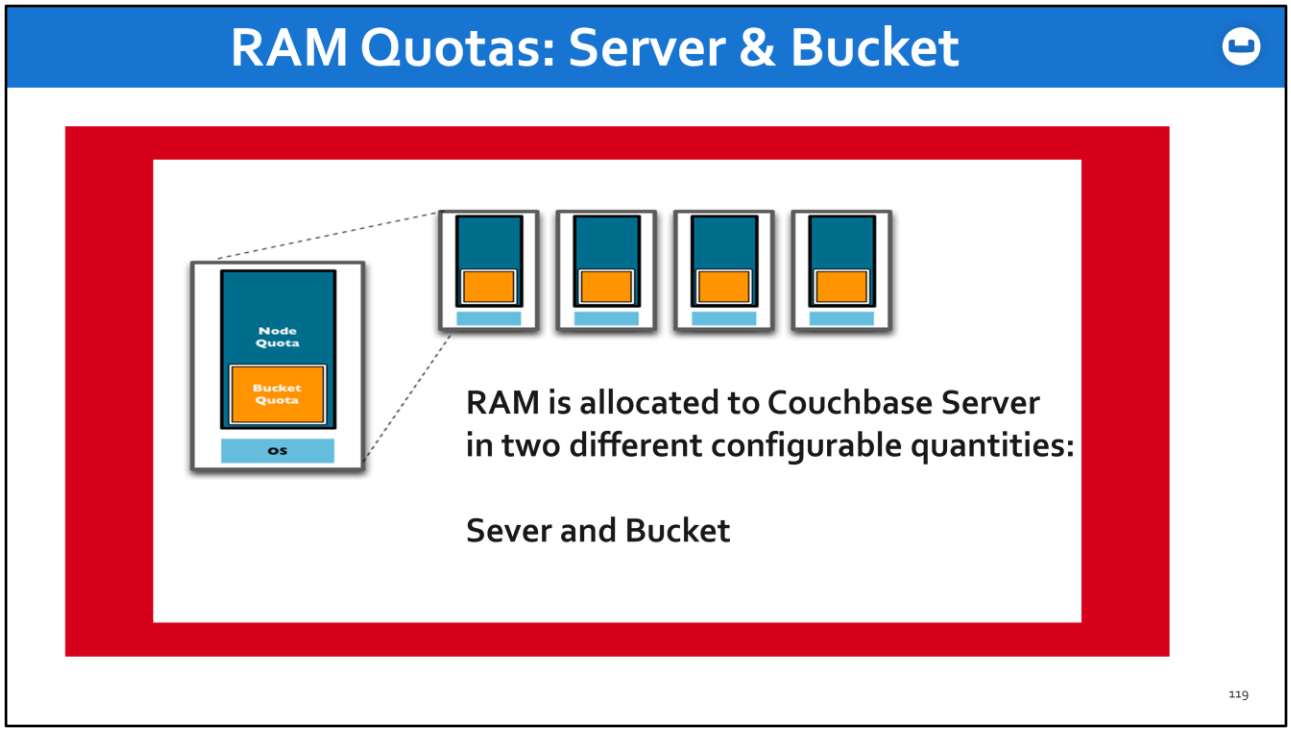
Only 1 replica in clusters < 20 nodes

Buckets have the following configurations:
-RAM – they are independently and dynamically sizeable
-Replica count – although it can be changed once the bucket is created, replicas are automatically created as servers are added and you can have 1, 2 or 3 extra copies of the data. For most clusters, we recommend only 1 replica as the overhead in capacity and load very often is not justified by the increased resiliency. It's very rare to have more than one node failing at a time.
-Disk location: You can put different buckets on different paths/partitions/devices depending on their IO requirements.

Memory Size



Per-Node RAM Quota

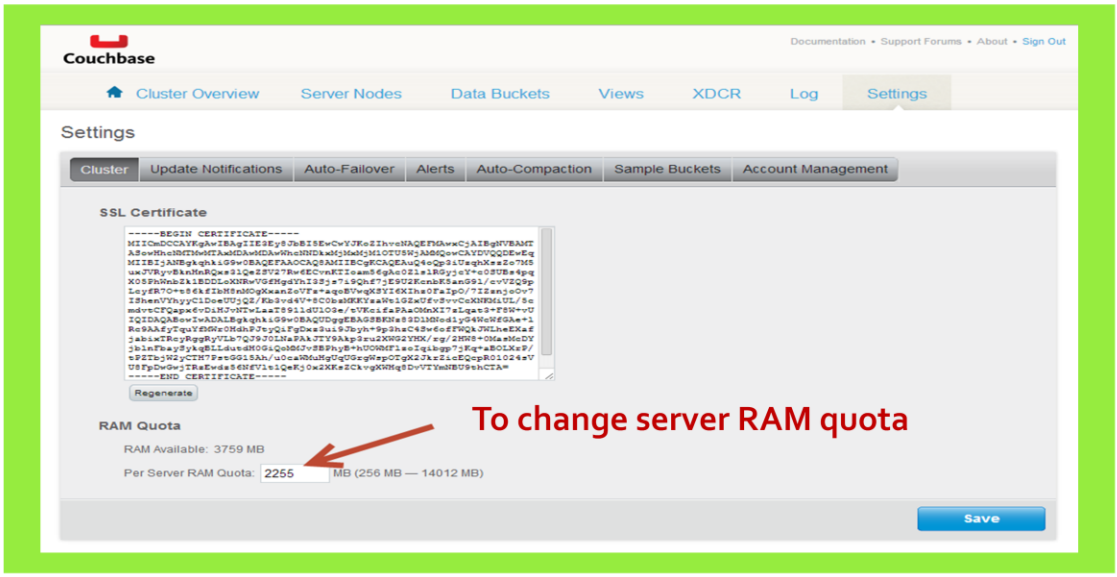


Server and Bucket Quota

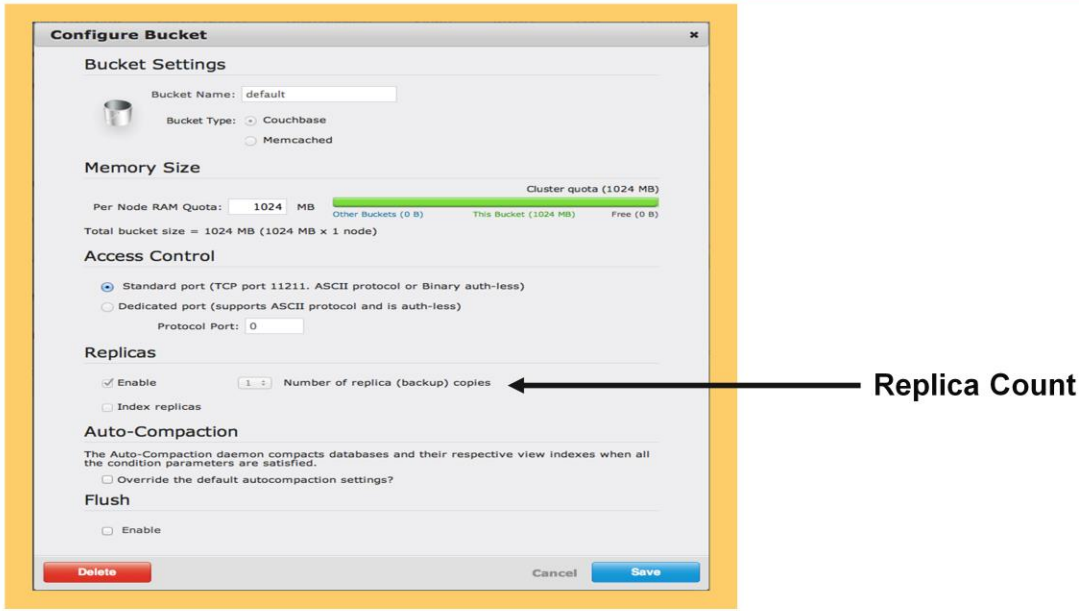
Server quota:
RAM that is allocated to the server when Couchbase Server is first installed
other nodes inherit from the 1st node
configured on a per-node basis (but must be the same across nodes)
can be changed
sets the limit of RAM allocated by Couchbase for caching data for all buckets
if you have 10 nodes and a 16GB Server Quota,
    there is 160GB RAM available across the cluster

Bucket quota
amount of RAM allocated to an individual bucket for caching data
is allocated out of the RAM defined by the Server Quota
dynamically configurable
is used by the system to determine when data should be ejected from memory
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

Server RAM Quota



Replica Count



Full copies of entire bucket

- **Shard by shard (vbucket by vbucket)**
- **Using TAP interface, RAM-RAM for near-immediate HA**

Evenly spread throughout cluster

- **Peer-to-peer replication, not master slave**
- **No replica shard on same server as active shard**
- **One server replicates to many others in cluster**

Not accessible by default

- **Strong consistency**
- **Not needed for scaling reads**

Replicas increase resource requirements:

- **RAM**
- **Disk IO**
- **Network**
- **CPU**

Replica count:

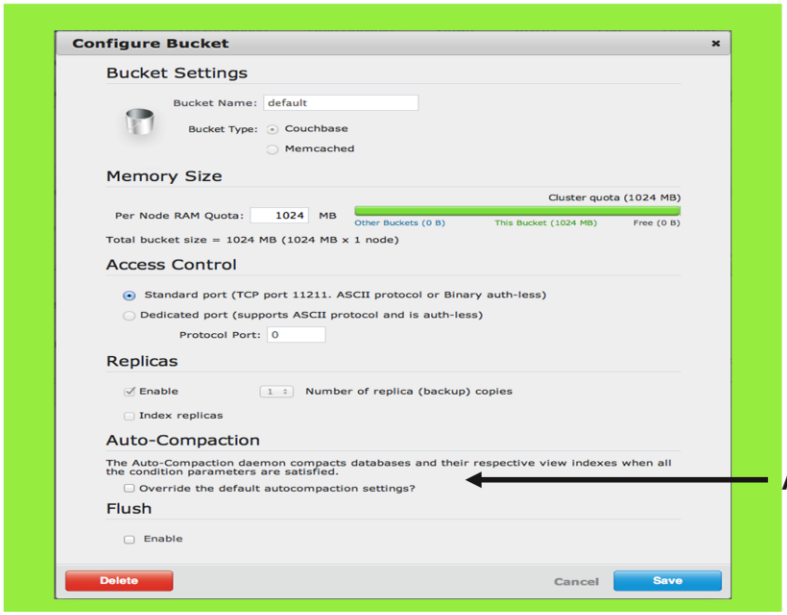
- **Configured at bucket creation**

1, 2 or 3 replicas (2, 3, or 4 total copies of the dataset)
Only takes effect when number of servers is larger than replica count
Best practice is 1 replica on clusters smaller than 20 nodes

Index Replicas allow for indexes to be created on replica data

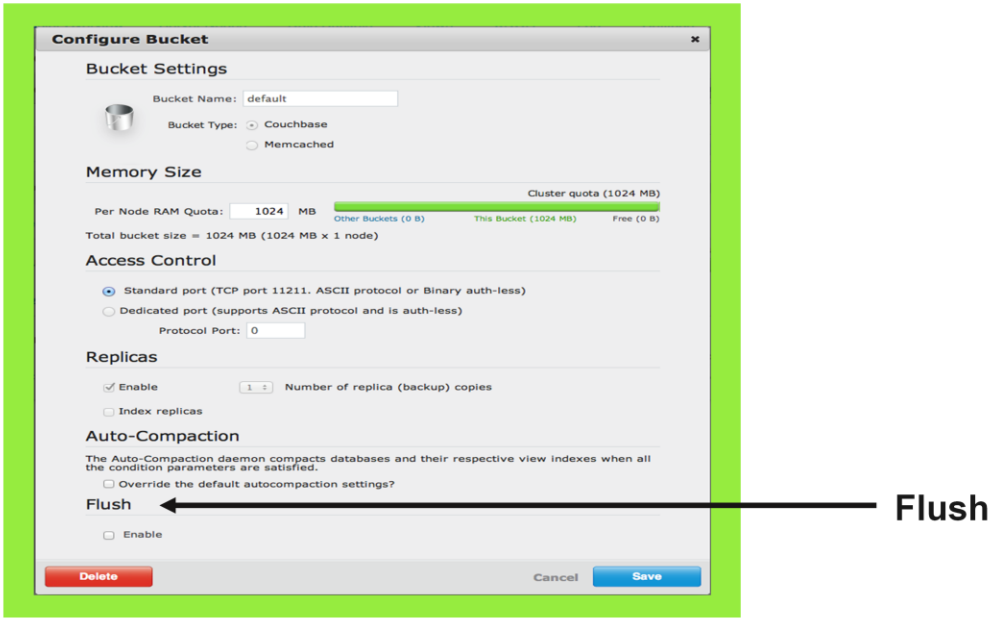
Improved failover behavior by having replica indexes pre-built
Increased disk IO and CPU requirements

Auto-Compaction



Auto-Compaction

Flush



Client-side operation to delete all data in a bucket

Also supported within the Admin UI

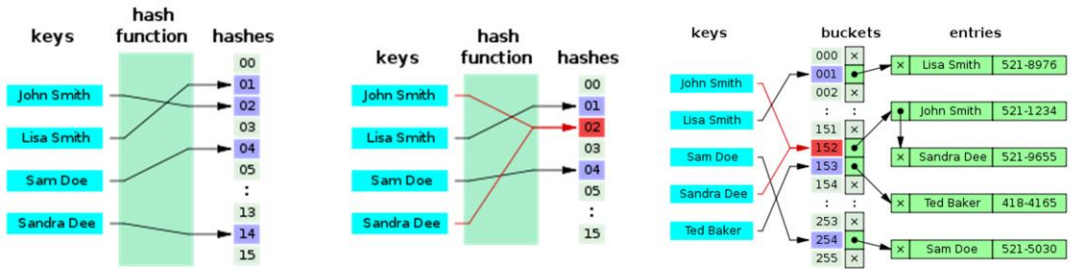
Useful for testing

Should be disabled on production buckets

Default setting is disabled

vBuckets

Generic Hash table



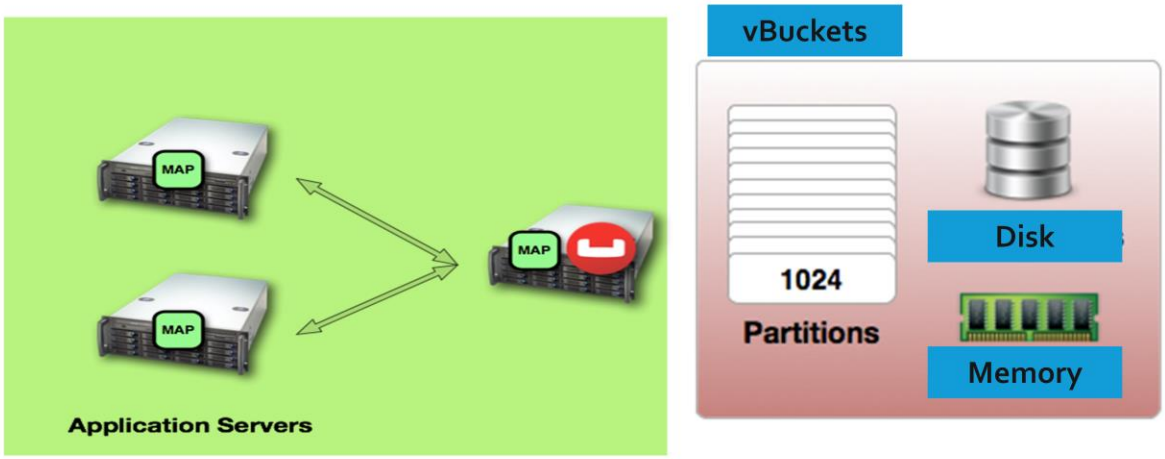
125

Hash functions are primarily used in hash tables,[1] to quickly locate a data record (e.g., a dictionary definition) given its search key (the headword). Specifically, the hash function is used to map the search key to an index; the index gives the place in the hash table where the corresponding record should be stored. Hash tables, in turn, are used to implement associative arrays and dynamic sets.

Typically, the domain of a hash function (the set of possible keys) is larger than its range (the number of different table indices), and so it will map several different keys to the same index. Therefore, each slot of a hash table is associated with (implicitly or explicitly) a set of records, rather than a single record. For this reason, each slot of a hash table is often called a bucket, and hash values are also called bucket indices.

Thus, the hash function only hints at the record's location — it tells where one should start looking for it. Still, in a half-full table, a good hash function will typically narrow the search down to only one or two entries.(wiki)

Key Hash-Partitioning

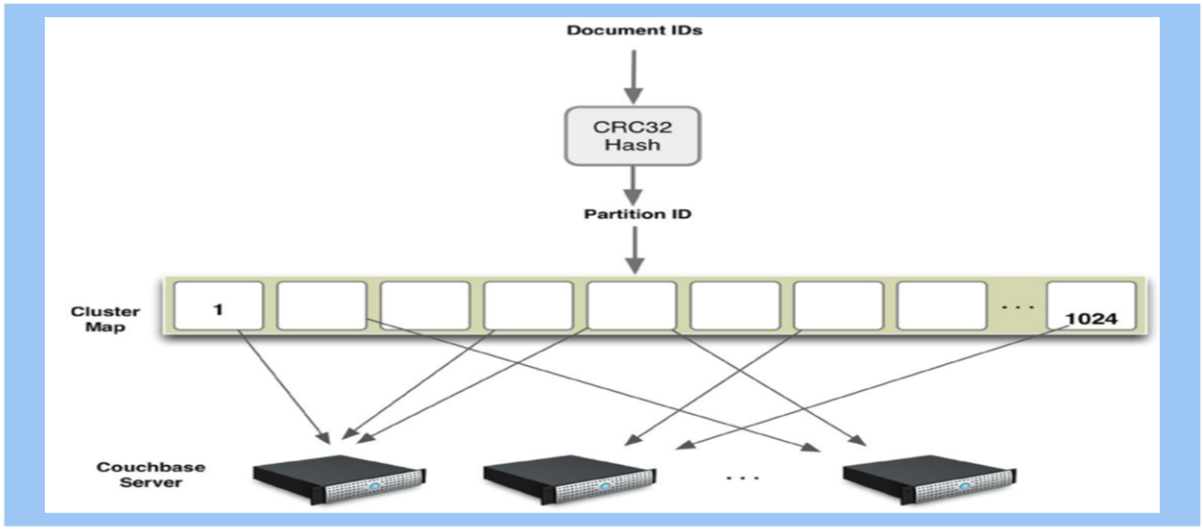


ClientHashFunction("john@couchbase.com") => vBucket(Partition)[0..1023]
{25}
ClusterMap[P(25)] => [x.x.x.x] => IP of Server Responsible for Partition 25

- **Used for both for distributing data and supporting replicas**
- **Clients access data in a bucket by directly communicating with the node responsible for the corresponding vBucket**
- **Every document ID belongs to a vBucket**
- **Hashing function is used to calculate the vBucket in which a given document belongs**

Mapping document IDs to partitions

.....and then to servers



127

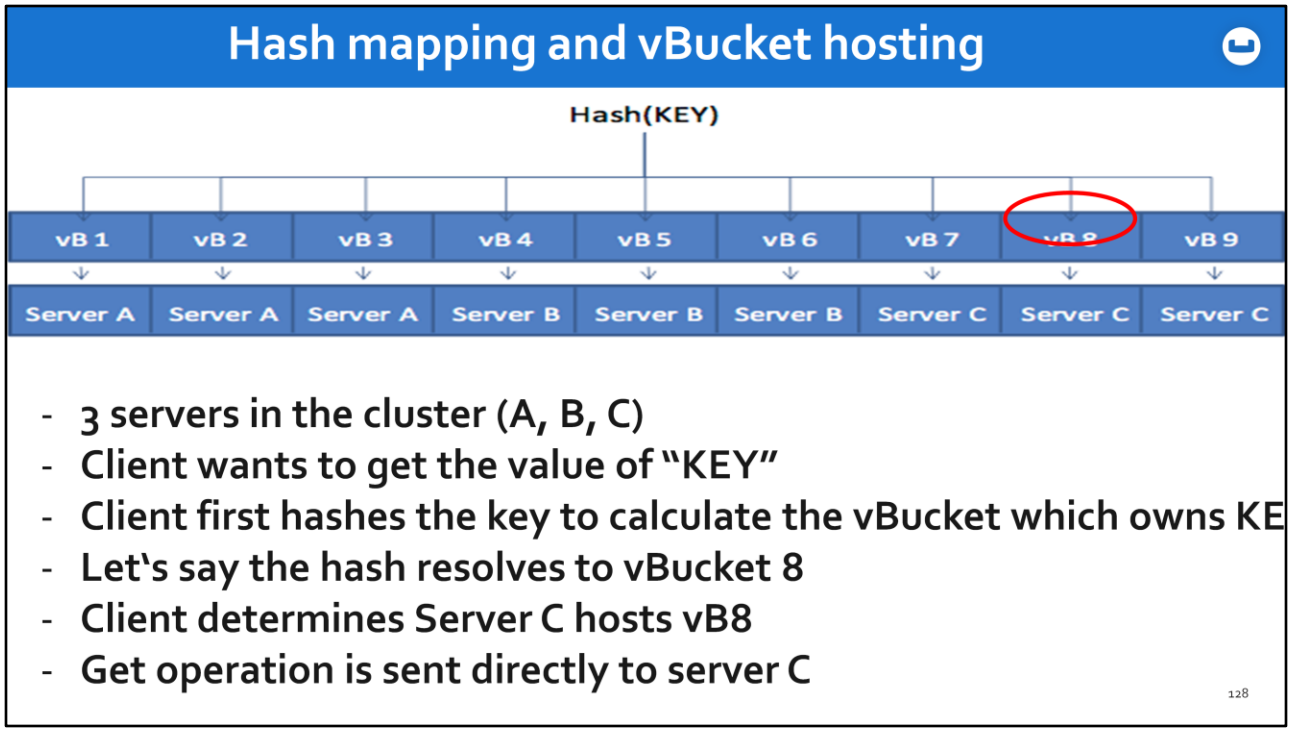
By design, each bucket is split into 1024 logical partitions called vBuckets. Partitions are mapped to servers
across the cluster, and this mapping is stored in a lookup structure called the cluster map . Applications can use
Couchbase's smart clients to interact with the server.

Smart clients hash the document ID , which is specified by
the application for every document added to Couchbase. As shown in Figure 2, the client applies a hash function
(CRC32) to every document that needs to be stored in Couchbase, and the document is sent to the server where it
should reside.

You can replicate a bucket up to three times within a cluster. Because Couchbase Server splits buckets into
vBuckets(partitions), a server manages only a subset of active and replica partitions.

This means that at any point in time, for
a given partition, only one copy is active with zero or more replica partitions on other servers. If a server hosting

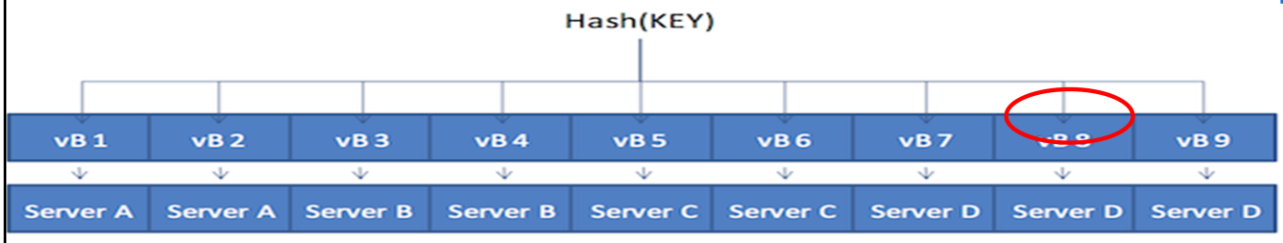
an active partition fails, the cluster promotes one of the replica partitions to active, ensuring the application can
continue to access data without any downtime.



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

The following diagram 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.

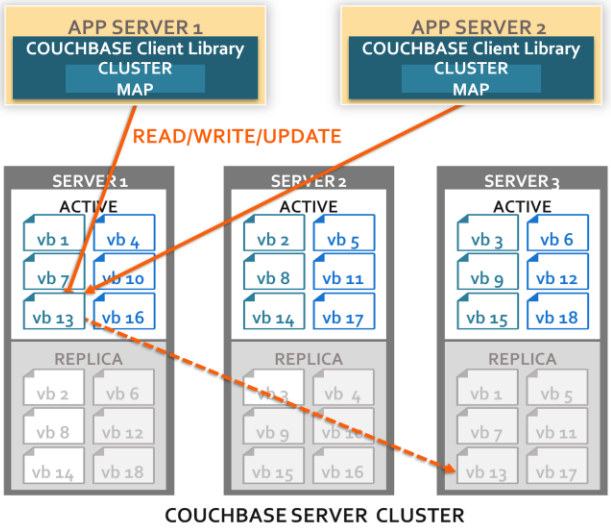
Hash mapping and vBucket hosting



- A new node, Server D is added to the cluster and the vBucket Map is updated
- After adding Server D, administrator has to run the 'rebalance' operation
- After rebalance, the new cluster map is sent to all cluster nodes and 'smart' clients

129

vbucket Distribution



- Vbuckets distributed evenly across servers
- Each server stores both active and replica vbuckets
  Only one server active at a time
- Client library provides app with simple interface to database
- Cluster map provides map to which server vbucket is on
  App never needs to know
- App reads, writes, updates docs
- Multiple app servers can access same document at same time

User Configured Replica Count = 1

Understanding those same operations, let's look at how this functions across a cluster.

With a Couchbase Server cluster of three nodes, you can see that the documents are evenly distributed throughout the cluster. (click) Additionally, the replica document are also evenly distributed so that no replica document is on the same node as it's active. This is showing one replica copy, but the same logic applies when there are two or three.

After the application server comes online and receives the vbucket map, all requests (read/write/update/delete) to a given document are sent to the node that is active for it. In this way, Couchbase ensures immediate and strong consistency. An application will always read its own writes. At no point is the replica data read which would introduce inconsistency. We will see later what happens when a node fails and the replica data needs to be activated.

The data is distributed (or "sharded") based upon a CRC32 hash of the key name which creates very even and random distribution of the data across all the nodes. Other systems shard based upon some user-generated value, which can lead to hot spots and imbalances within a cluster which we don't have. By distributing the data evenly across the cluster and letting the clients load balance themselves, the load is also evenly distributed across all the nodes in the cluster, making them "active-

active". Other systems using "master-slave" configurations basically end up wasting processing power and hardware in the background.

Although the diagram only shows a few "shards" of data, we actually use 1024 slices/shards/vbuckets, technically this limits us to 1024 active nodes in a cluster, but also has lots of benefits for smaller clusters. The data is sharded very granularly and can be moved and compacted as such. This allows the cluster to scale very evenly and linearly for more RAM/disk/network and CPU.

vBucket States

- vBucket has 1 of 4 states on each node:
  - **Active: Handling all request for this vBucket**
  - **Dead: Not responsible for this vBucket**
  - **Replica: Not accepting client traffic, but accepting replica traffic**
  - **Pending: Blocking on requests (waiting to become active)**
- In a healthy cluster:
  - **1024 'active' buckets**
  - **1024 'replica' buckets (or some multiple of)**

(During rebalancing or failover there may be more or less replica vbuckets)

"not_my_vbucket"

- If a request is made for a key in a vBucket to a server that it's not active on, the server responds with "not_my_vbucket"

- It is up to the client library to handle this response and "find" the right server

vBuckets

- vBuckets are the concept that powers Couchbase:
  - key -> vBucket -> server
- Replication/Failover
- Rebalancing
- Indexing operates per vBucket to allow distributed indexing
- Compaction operates per vBucket

Buckets and vBuckets

- A bucket is a grouping of data for the purpose of management

- Each bucket has a separate vBucket map
- Each bucket is spread evenly across the whole cluster
- Each node has a portion of each bucket

- Client libraries connect to a bucket and receive its vbucket map, then send requests directly to nodes responsible for a particular vbucket

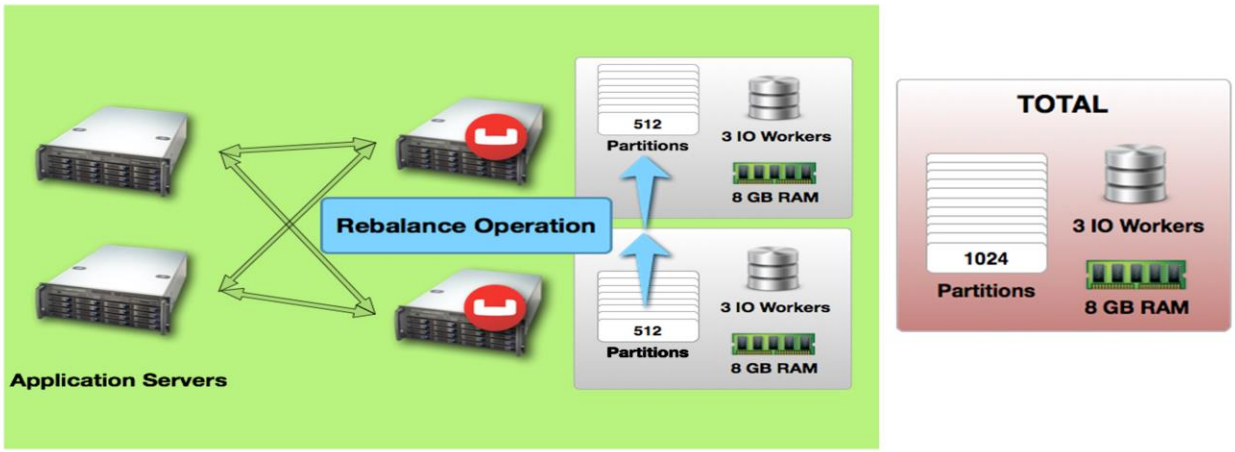
Rebalancing

- Used when expanding or shrinking a Couchbase Cluster
- Redistributes vBuckets evenly within the cluster (so it physically moves the data between the nodes to match the new cluster map)
- Must be initiated manually (never done automatically)
- Can be run on a live cluster that is servicing requests
- During rebalance, clients will read vBuckets from the existing structure. In the background vBuckets get moved to other nodes. After all the moves are finished, smart clients are notified of the new cluster map

135

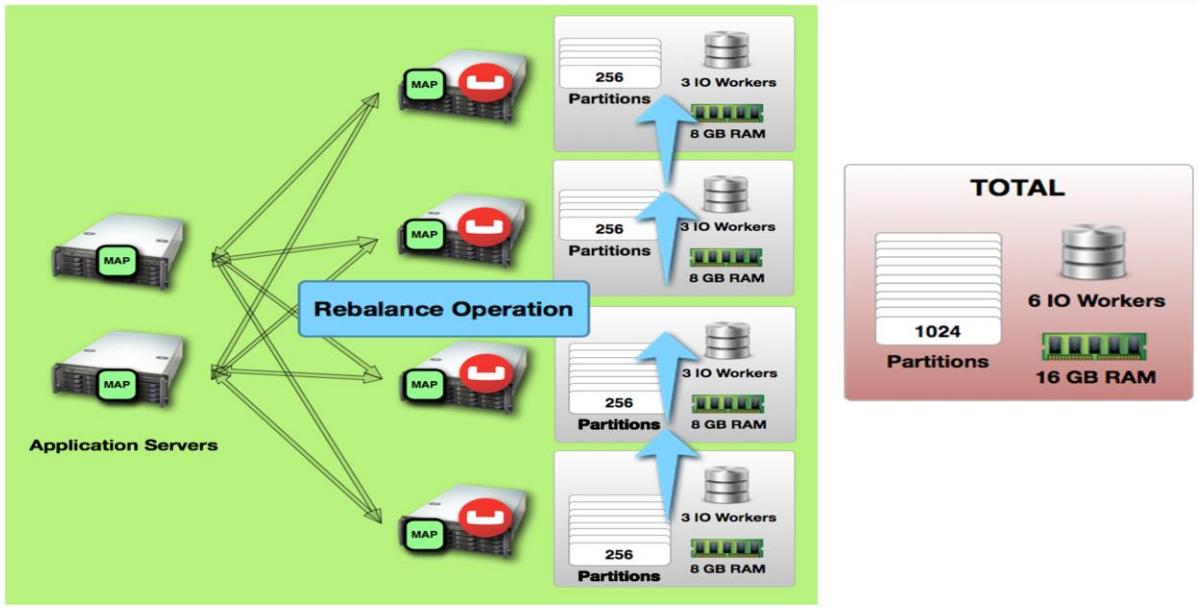
Rebalancing is never done automatically b/c it can put a lot of load on the remaining nodes.

Horizontal Scale-Rebalance

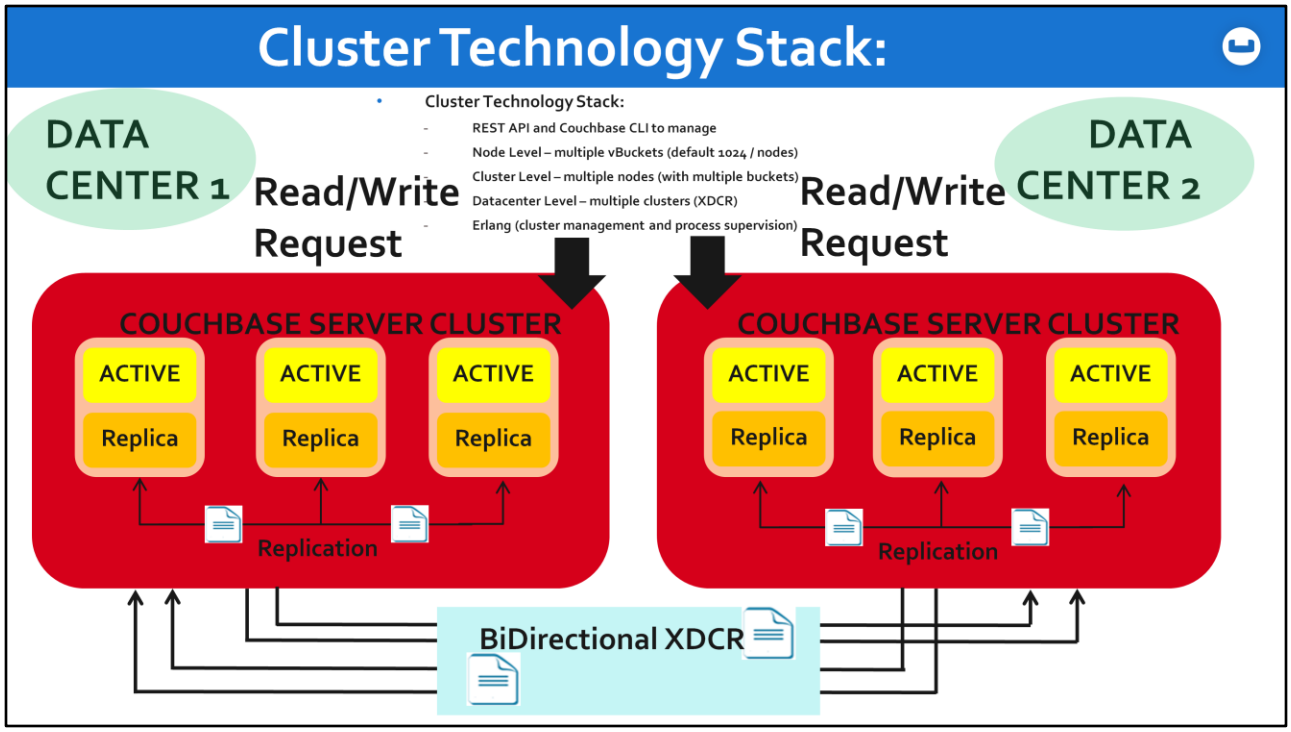


Note: Fix the #s in total

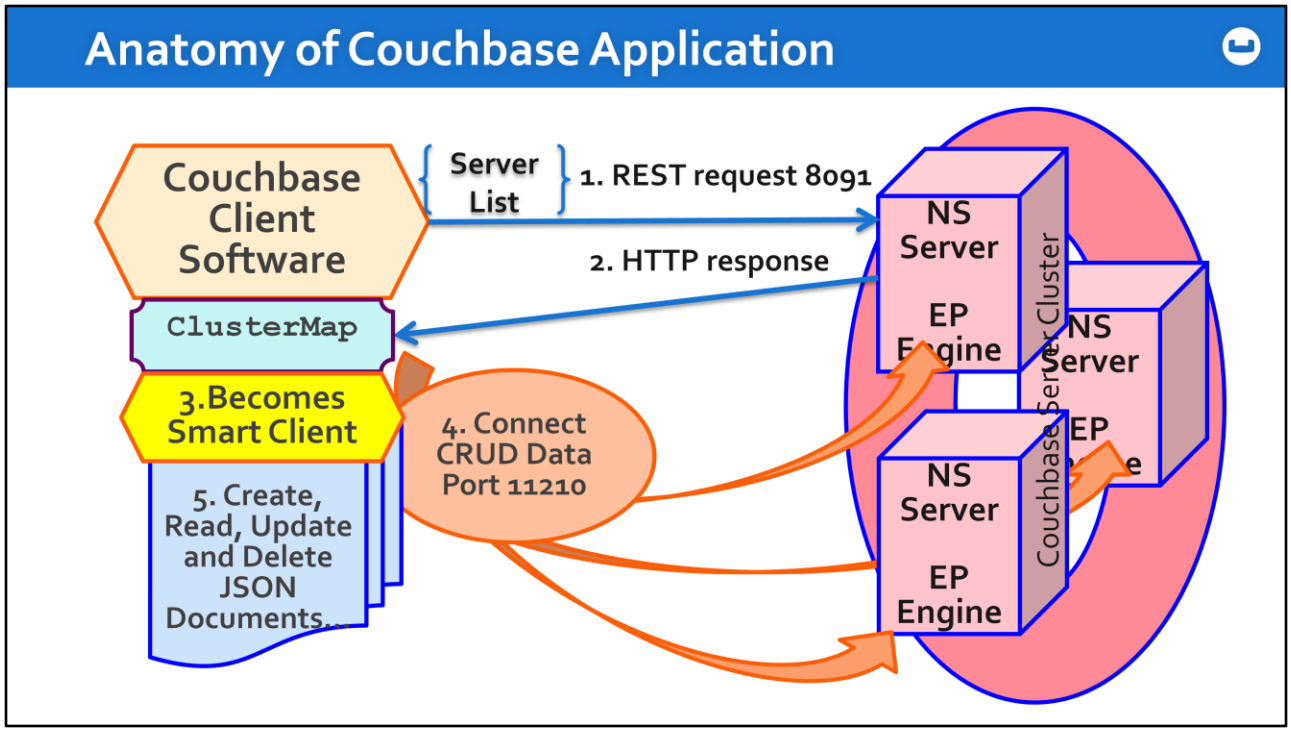
Horizontal Scale-Rebalance



Couchbase Architecture



1. Most modern operating systems want a few gigabytes (Windows usually a bit more than Linux), and there may be other processes running on these nodes such as monitoring agents. There are also needs for IO caching both for views and for the general functioning of the system. We typically recommend about 60-80% of an system's RAM to be allocated to Couchbase's quota, leaving the rest for headroom and memory needs outside of Couchbase itself.
2. Cross Datacenter Replication (XDCR) is covered later in this course.
3. See https://blog.couchbase.com/tag/erlang



The Memcache Client also uses a server list, but as contrasted to the Couchbase Client, there are no REST calls, it is only working over port 1Upon creation of client object, connection is made to 8091 on provided IP, supplying bucket name and administrator password
Connection remains open (“comet stream”) for topology changes

Client Library Connection

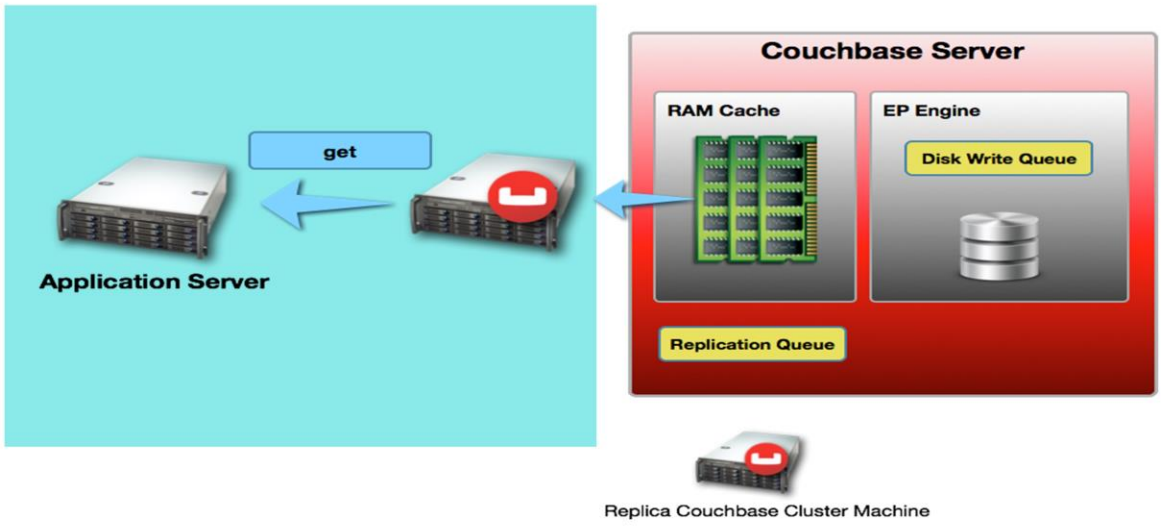
Node responds with vbucket map for supplied bucket
Map contains list of other nodes and vbucket assignments

Client makes connection to every node in the cluster (when varies a bit by language) on port 11210, supplying bucket name and optional password

Individual document access is made over connections to 11210 based on vbucket map

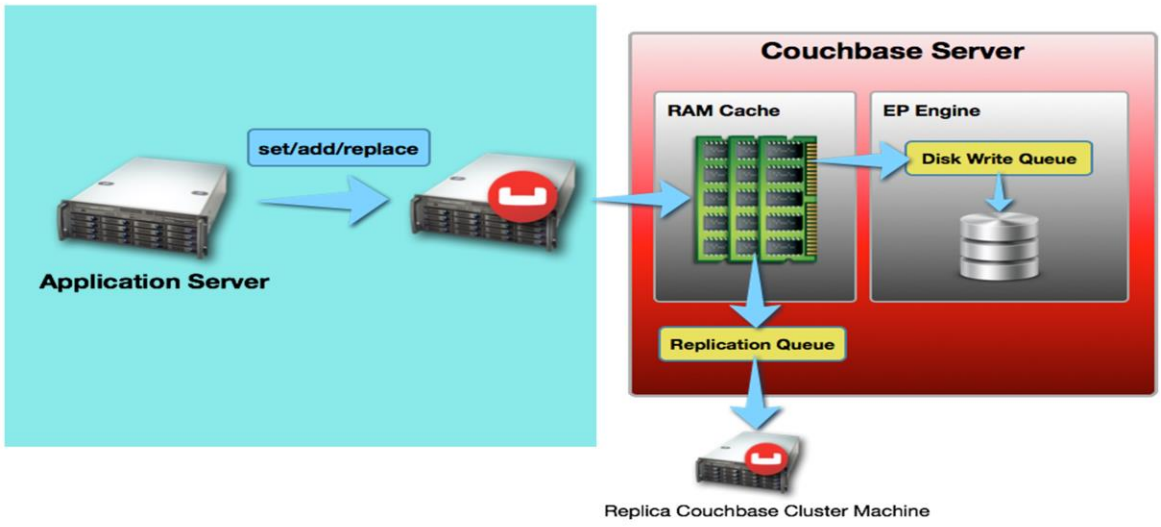
View queries are made to port 8092, round-robin’ed through node list
1210, and is very fast. This is using a proprietary Memchached protocol.

Retrieval Operations

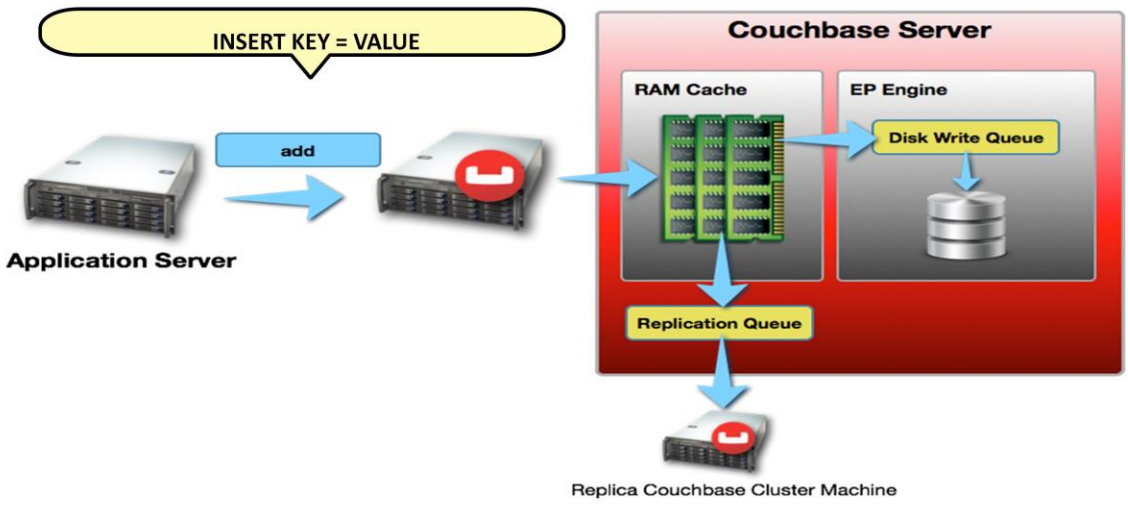


EP Engine: Eventual Persistance layer

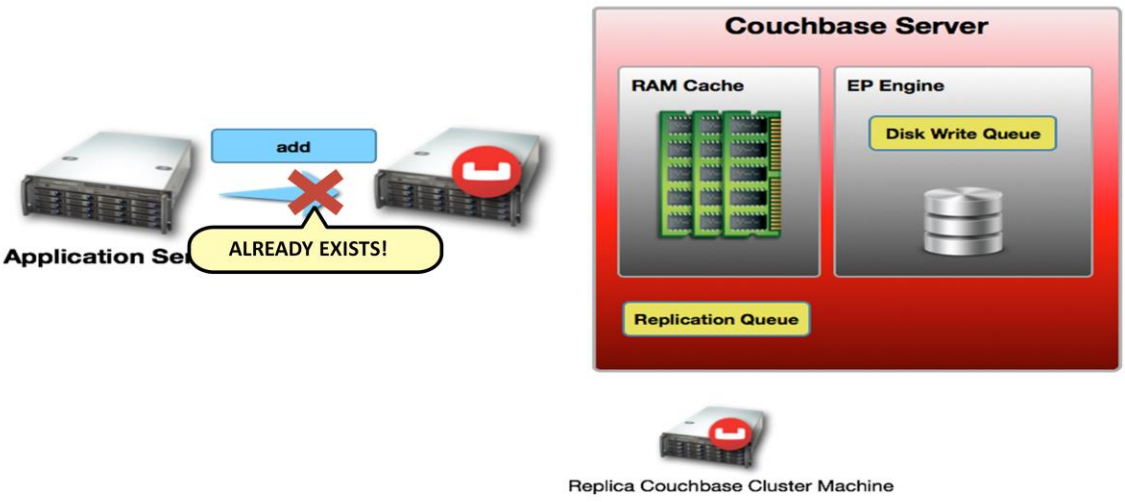
Storage Operations



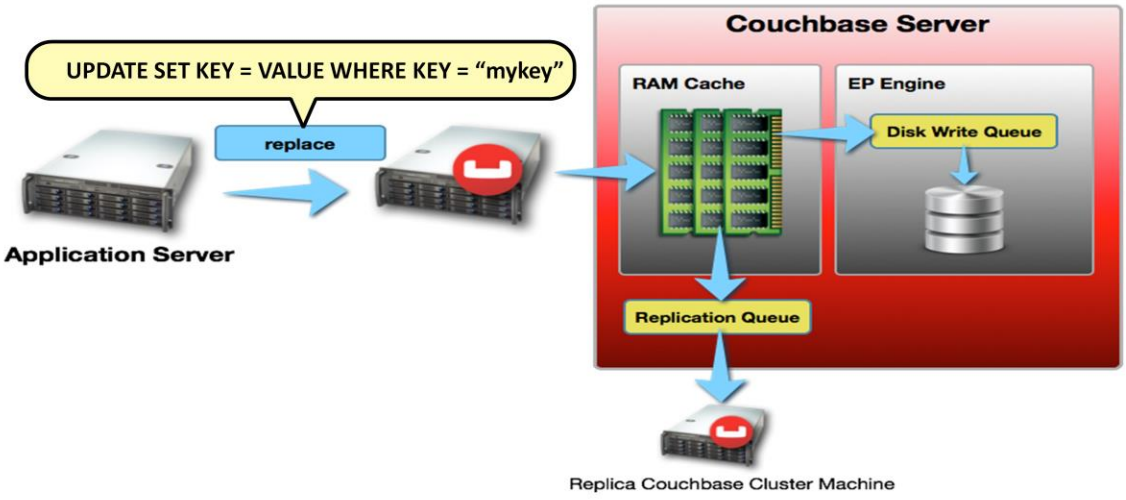
Storage Operations - add



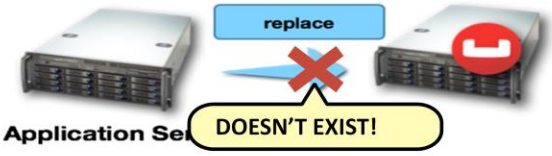
Storage Operations - add

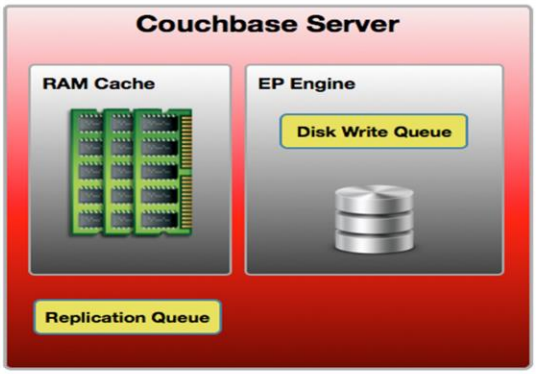


Storage Operations - replace



Storage Operations - replace





Replica Couchbase Cluster Machine

Deletes and Tombstones

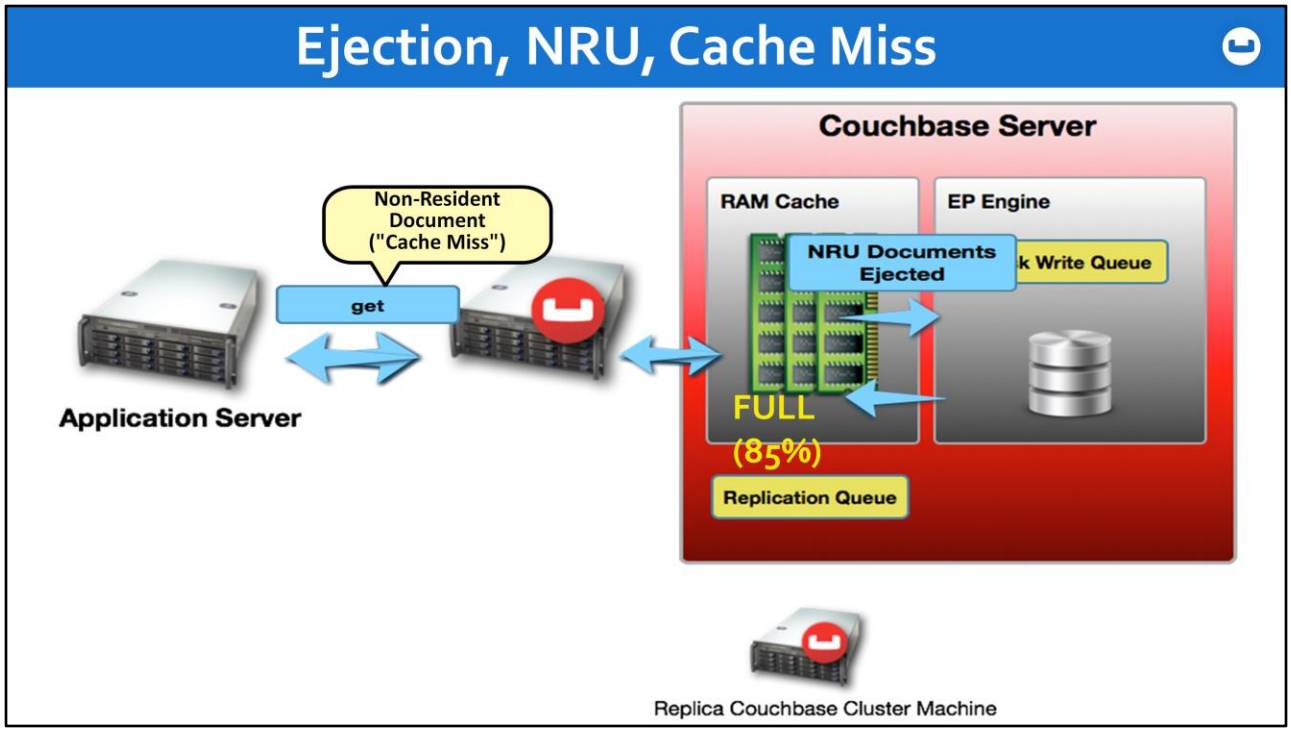
- Items on disk can be deleted b/c of a client request or an expired TTL
- Default TTL = 0 (document will be kept indefinitely)
- A delete is just a write with a value of tombstone and an updated revision # for the item
- Tombstones are records of expired or deleted items and they include the key for the item as well as metadata
- Couchbase Server stores the key plus several bytes of metadata per deleted item
- You have to run compaction to truly get rid of deleted items
- Metadata Purge Interval: how long to keep the tombstones before removing them permanently, defaults to 3 days (runs as part of auto-compaction) This is to help keep in sync: elastic search, views and XDCR

147

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).

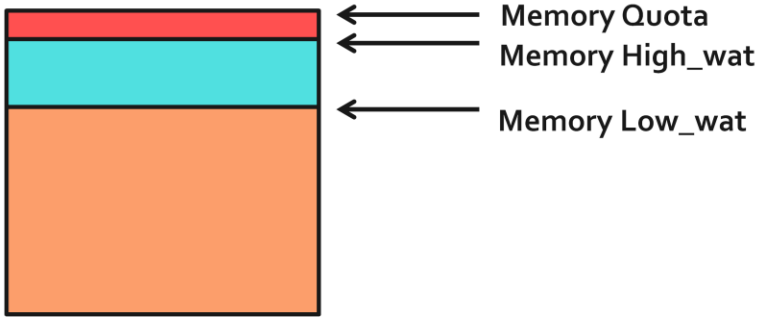
Typical uses for an expiration value include web session data, where you want the actively stored information to be removed from the system if the user activity has stopped and not been explicitly deleted. The data will time out and be removed from the system, freeing up RAM and disk for more active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



All items in the server contain metadata indicating whether the item has been recently accessed or not. This metadata is known as not-recently-used (NRU). If an item has not been recently used, then the item is a candidate for ejection if the high water mark has been exceeded. When the high water mark has been exceeded, the server evicts items from RAM.

Ejection, eviction and working set management

Bucket Configuration



Ejection: Applies to Couchbase buckets only. Performed automatically. removes items from RAM to make room for frequently-used items (but the items are persisted to disk before ejection!)

Eviction: Applies to Memcached buckets only. Least recently used items can get evicted and then they are completely irretrievable

149

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*.

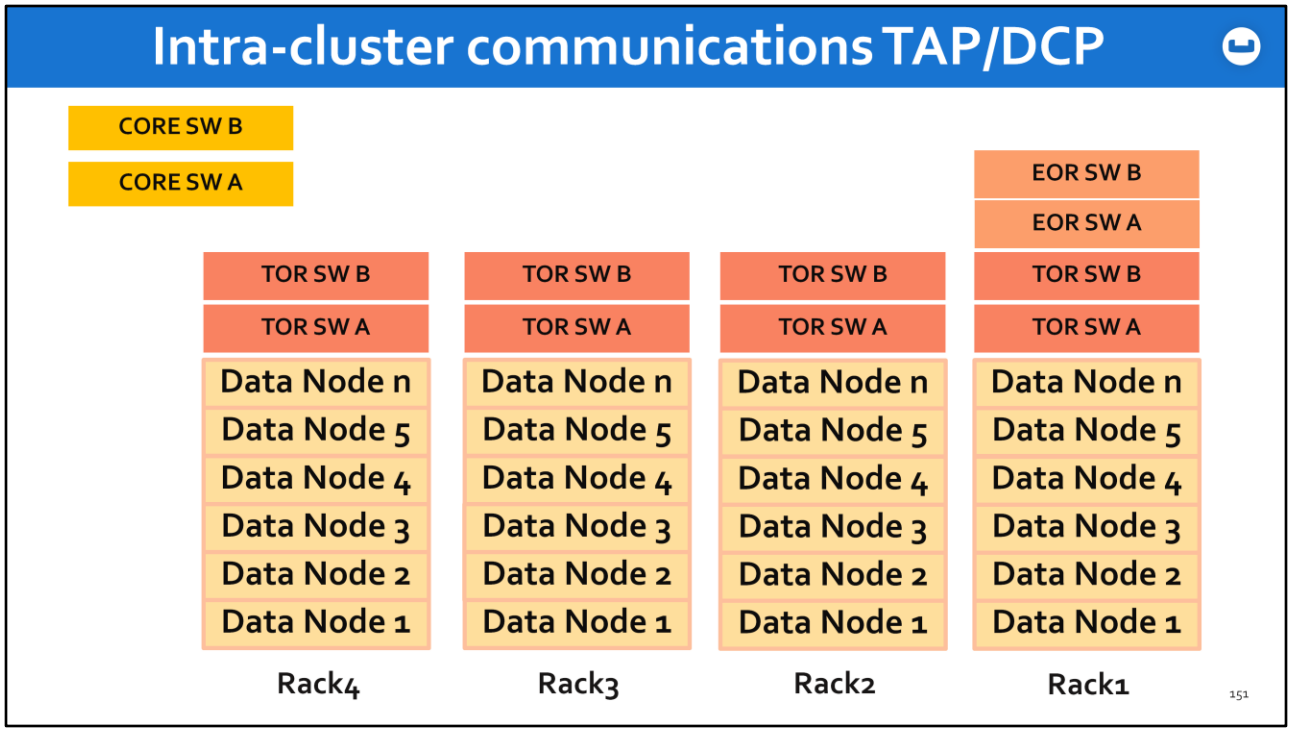
In addition to memory quota for the caching layer, there are two watermarks the engine uses to determine when it is necessary to start persisting more data to disk. These aremem_low_wat and mem_high_wat.

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 runs until measured memory reaches mem_low_wat. If the rate of incoming items is faster than the migration of items to disk, the system may return errors indicating there is not enough space. This 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.

Failover

- If a server crashes, it will be marked as down in the UI and then an Admin can trigger a failover on it
- After failover, the replica vBucket partners of the active vBuckets on the remaining node are enabled
- The failover process also updates the cluster map
- Can be performed manually or automatically

150



- **TAP protocol is an internal part of the Couchbase Server system**
- **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.**

Database Change Protocol (DCP) Couchbase 3.X is the protocol used to stream data changes to buckets.
The Database Change Protocol (DCP) is a streaming protocol that significantly reduces latency for view updates. With DCP, changes made to documents in memory are immediately streamed to be indexed without being written to disk. This provides faster view consistency which provides fresher data. DCP reduces latency for cross data center replication (XDCR). Data is replicated memory-to-memory from the source cluster to the destination cluster before being written to disk on the source

cluster.

Lab #2: App Client Installation

Time: 1 hour

- Install a separate Application Server
- Test Memcached text protocol with telnet
- cbc commands to create, read & delete a key

152

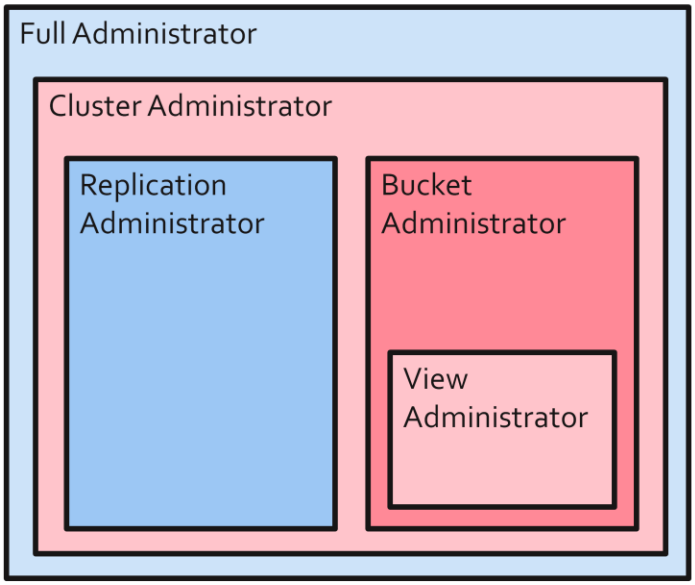

Couchbase
Learning
Services

How is access controlled by user role?

Authentication via existing LDAP

Authorization via six roles

- **Full** – control all settings, including security roles
  - **Read Only** – view all settings
- **Cluster** – control all cluster settings, except security roles
- **Bucket** – control all bucket settings, except XDCR creation
- **View** – define/run map-reduce and spatial views on a bucket
- **Replication** – configure XDCR topology and replications with no other admin access



Authentication – admin (ldap or pw) vs app (pw)
Authorization – admin (see slide) app (pw per bucket = full control = full crud)
Encryption -
Auditing