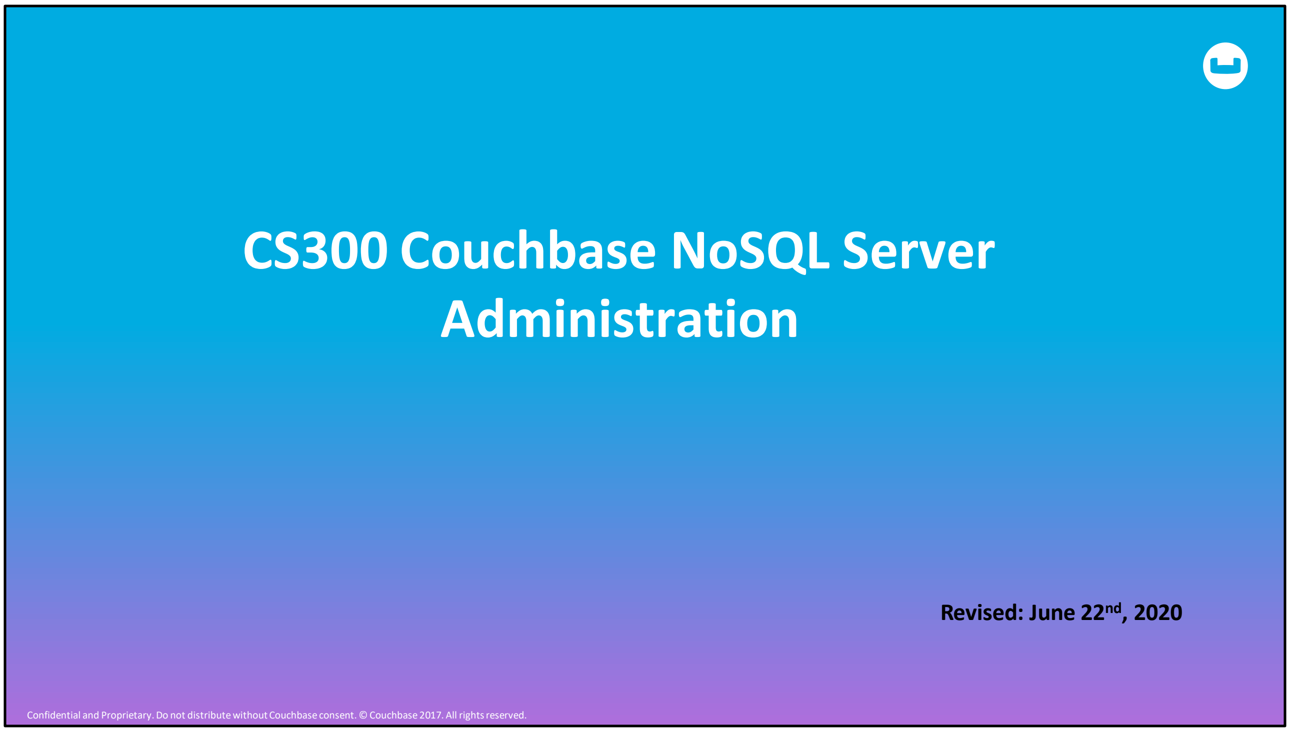
Couchbase

# CS300 SYSTEM ADMINISTRATION AND OPERATIONS 6.5

Couchbase Learning services

**Revised: June 22nd, 2020**

Confidential and Proprietary. Do not distribute without Couchbase consent. © Couchbase 2017. All rights reserved.



# CS300 Couchbase NoSQL Server Administration

Revised: June 22nd, 2020

Confidential and Proprietary. Do not distribute without Couchbase consent. © Couchbase 2017. All rights reserved.

## Agenda and Course Organization

Couchbase

## Technical Agenda – Overview

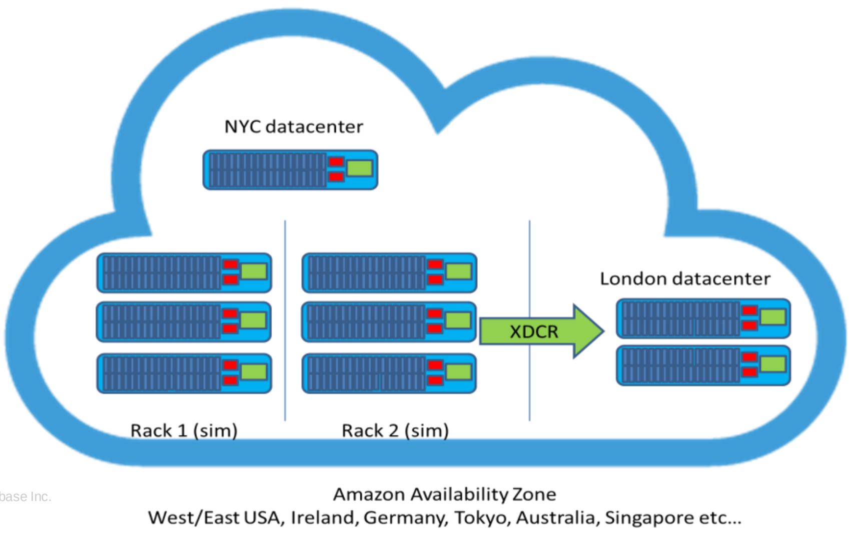
**Day 1: Origins of NoSQL & Couchbase, Architecture overview, Installation, Security & SDK**

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

**Day 3: Indexes, Search(FTS) Eventing & Analytics, Cross Datacenter Replication(XDCR)**

**Day 4: Backup/Restore, Performance, Sizing, Tools & Resources**

## Lab Environment in Amazon Cloud Availability Zones



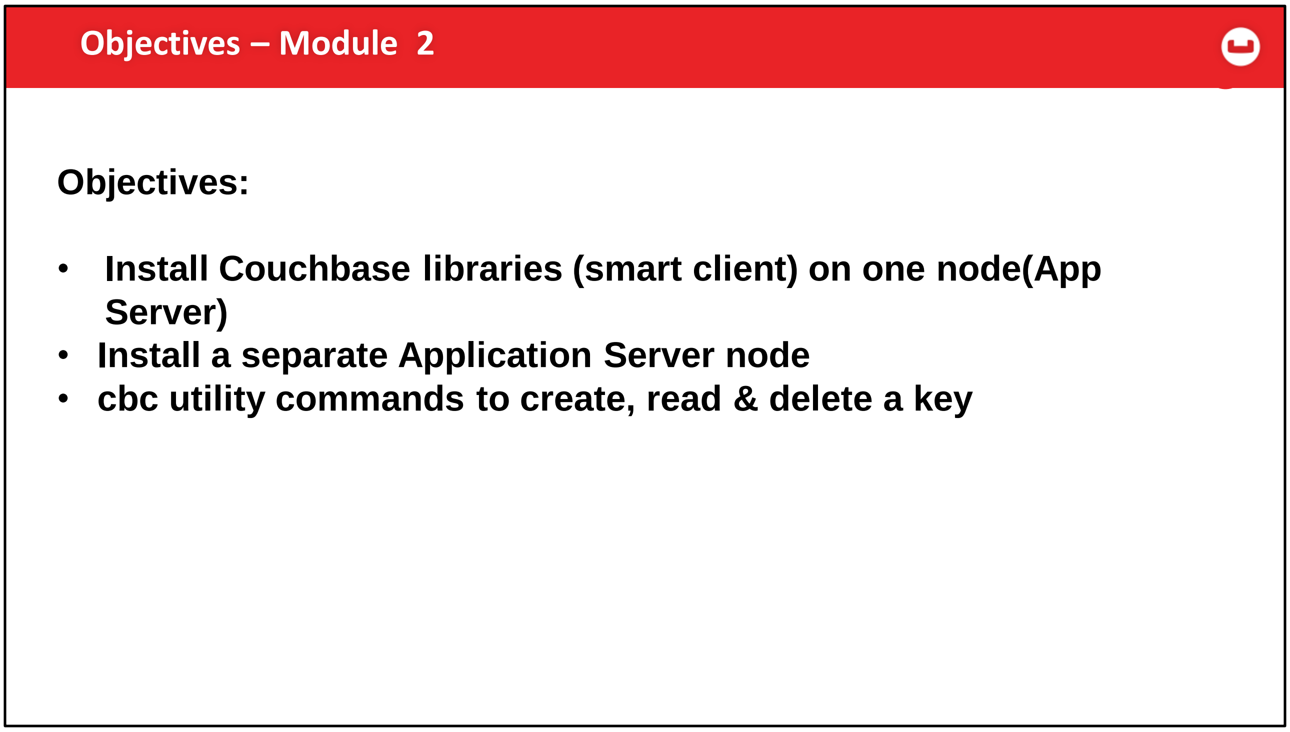
©2019Couchbase Inc.

## Objectives – Module 1

**Objectives:**

- **Install Couchbase 6.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**
- **cbc utility commands to create, read & delete a key**

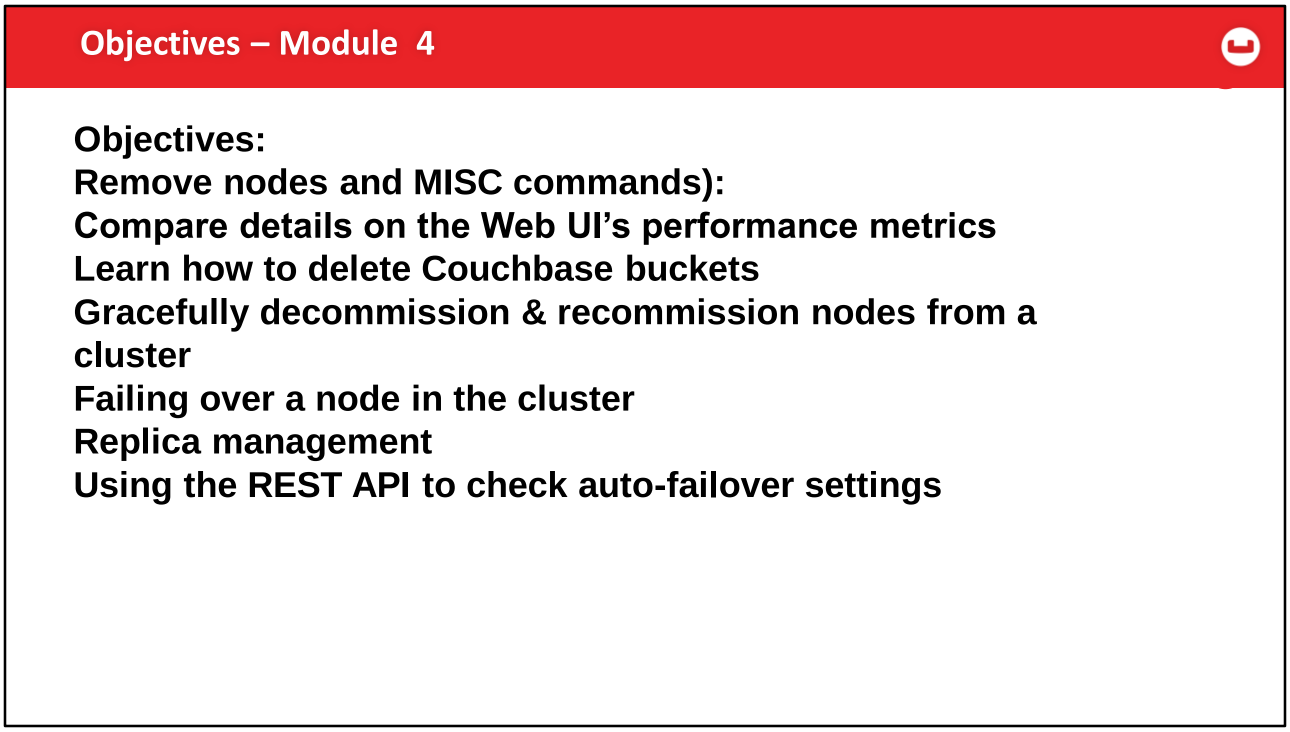
**Objectives:**

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

## Objectives – Module 3

**Objectives:**
**Grow cluster to 6 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**

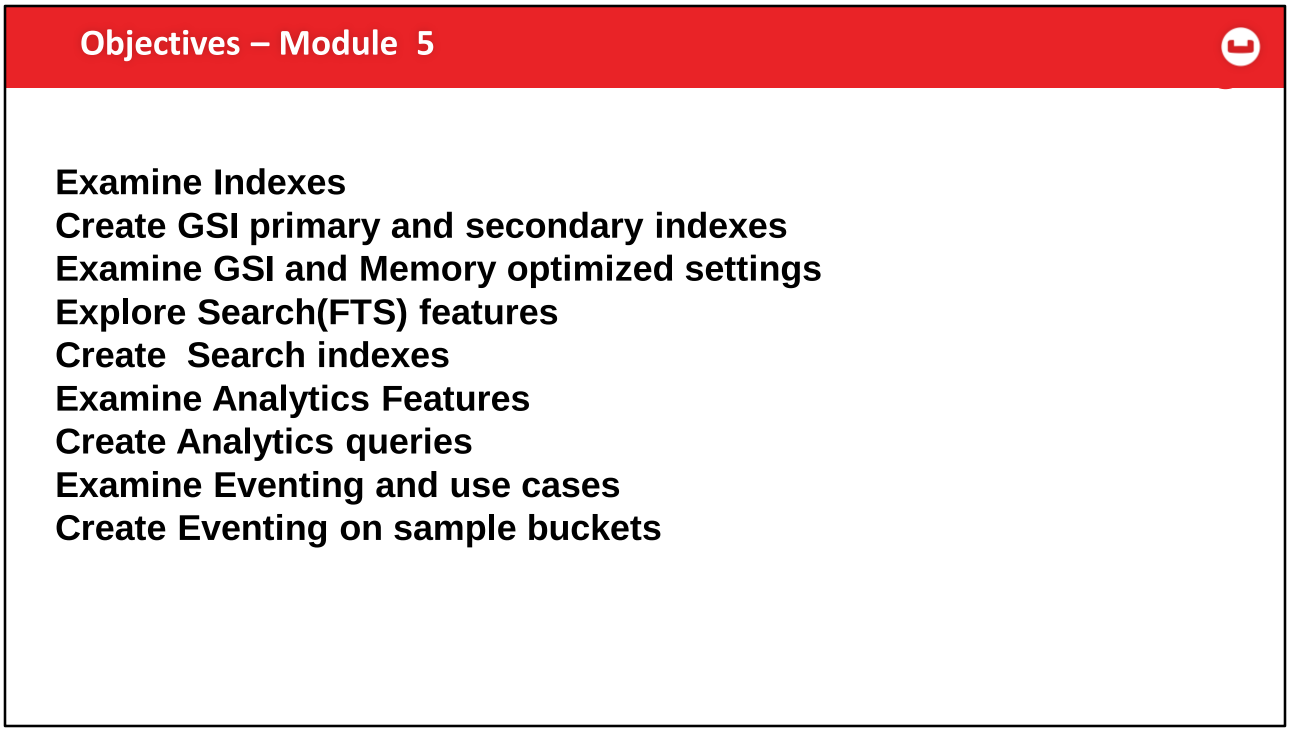
Objectives:
Grow cluster to 4 nodes
Exploring the cluster map
Differentiate Buckets and vBuckets
Rebalance the cluster
Introduction to performance metrics via Web UI
Adjust the number of replicas
Use the cbstats command to see active + replica vBuckets



## 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**

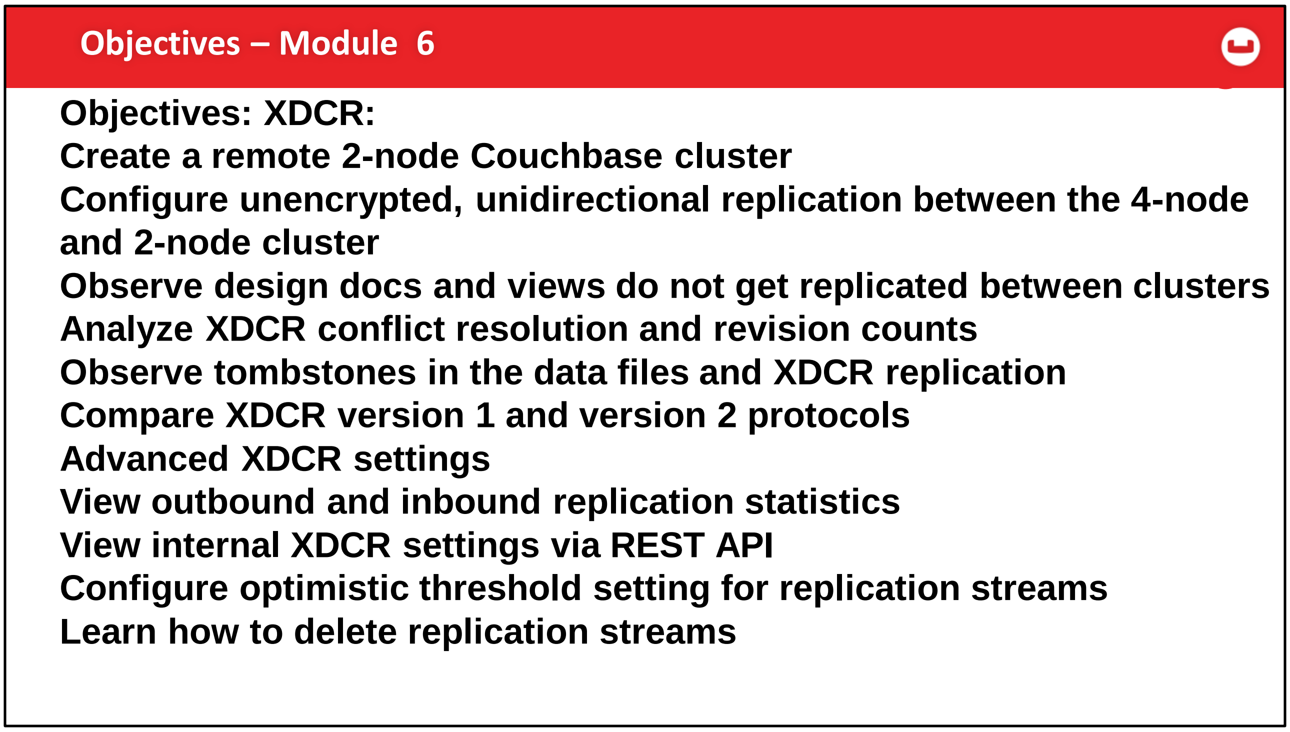
Objectives:
Remove nodes and MISC commands):
Compare details on the Web UI’s performance metrics
Learn how to delete Couchbase buckets
Gracefully decommission & recommission nodes from a cluster
Failing over a node in the cluster
Replica management
Using the REST API to check auto-failover settings



## Objectives – Module 5

Examine Indexes
Create GSI primary and secondary indexes
Examine GSI and Memory optimized settings
Explore Search(FTS) features
Create  Search indexes
Examine Analytics Features
Create Analytics queries
Examine Eventing and use cases
Create Eventing on sample buckets

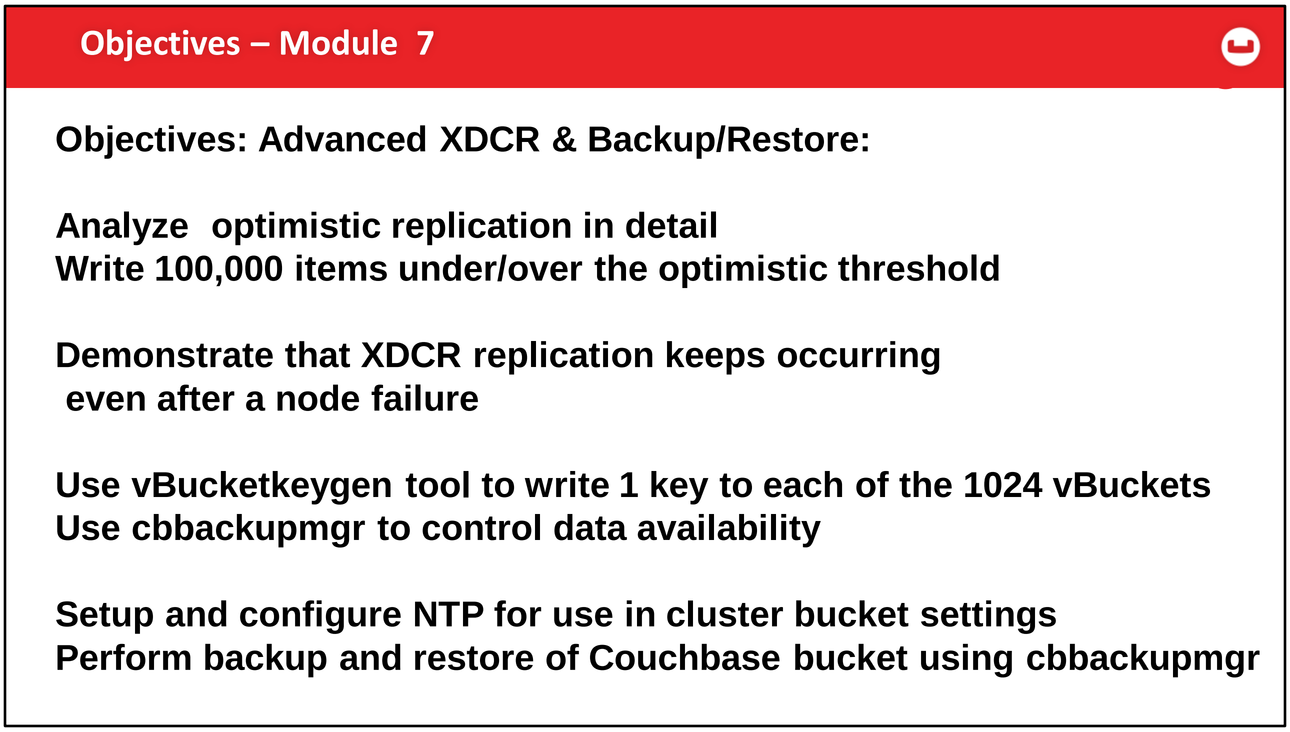
Objectives:



## 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/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 cbbackupmgr to control data availability**

**Setup and configure NTP for use in cluster bucket settings**
**Perform backup and restore of Couchbase bucket using cbbackupmgr**

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

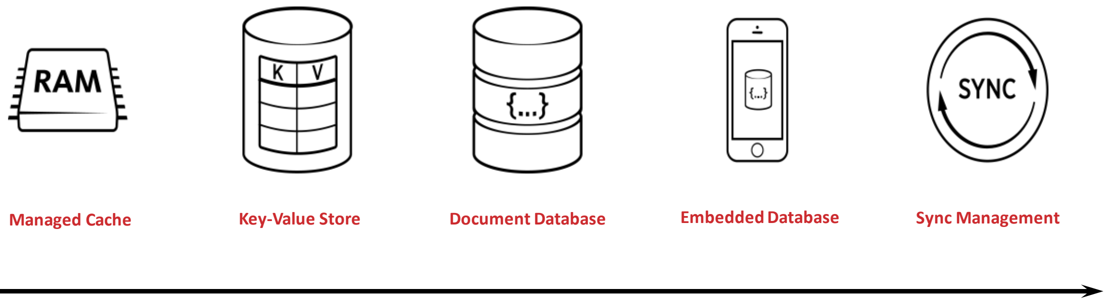
**Objectives Performance & Compaction:**
**Compute high water mark and low water mark for a bucket**
**Observe how ejection works**
**Analyze the Not-Recently-Used metadata setting**
**Determine when the item pager runs**
**Determine when Disk reads vs RAM reads occur**
**Understand how different traffic 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**

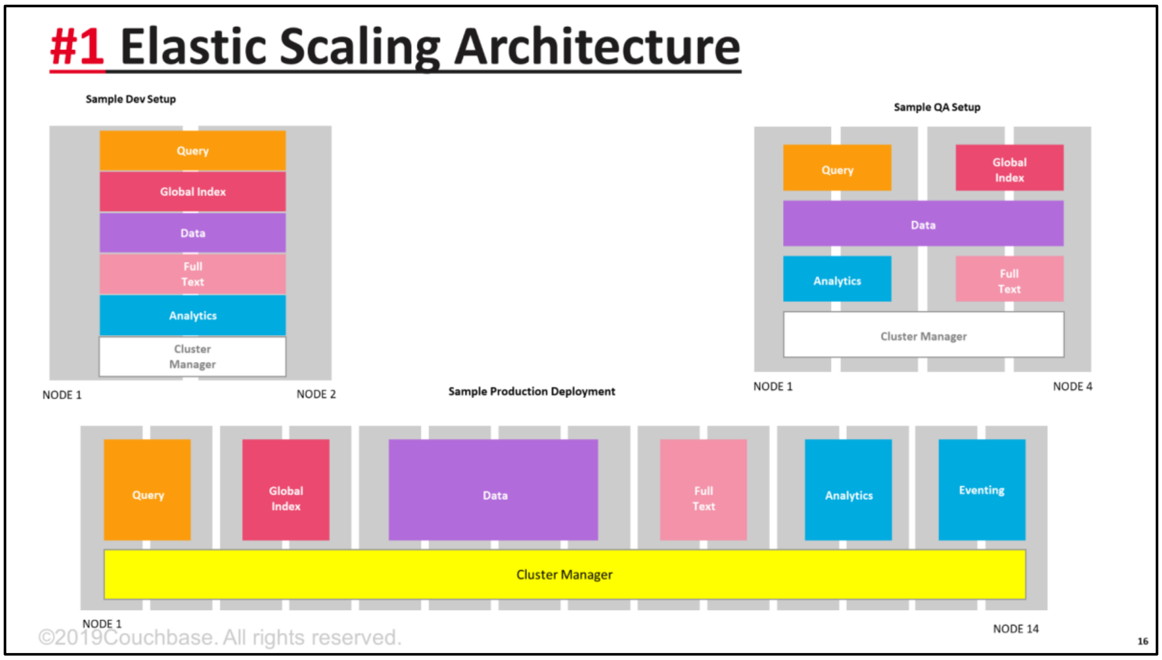
## Intro to Couchbase
## Core principles

Couchbase

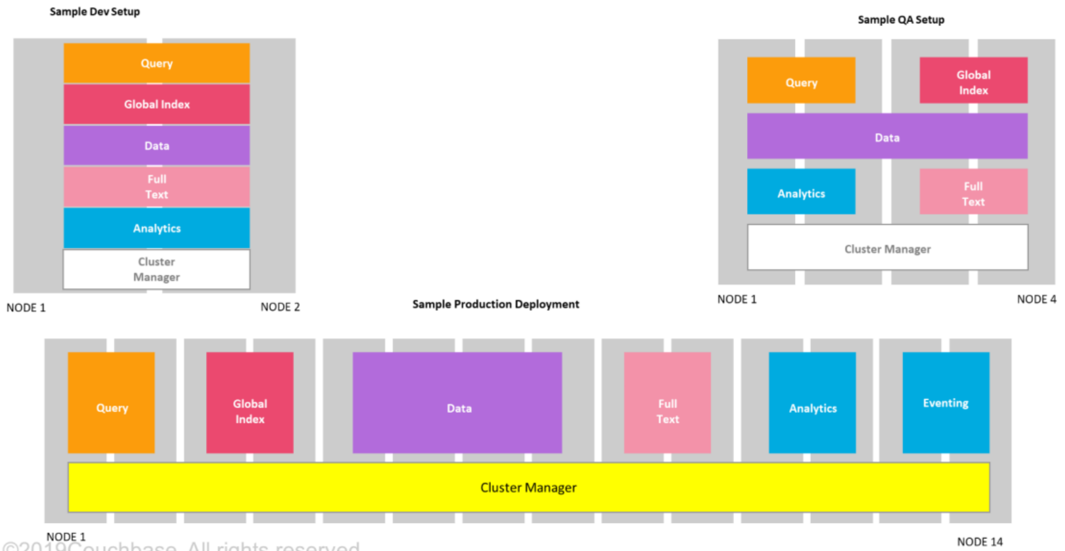
## Couchbase Server Defined

**Couchbase is the first NoSQL Database to enable you to develop with agility and operate at any scale.**

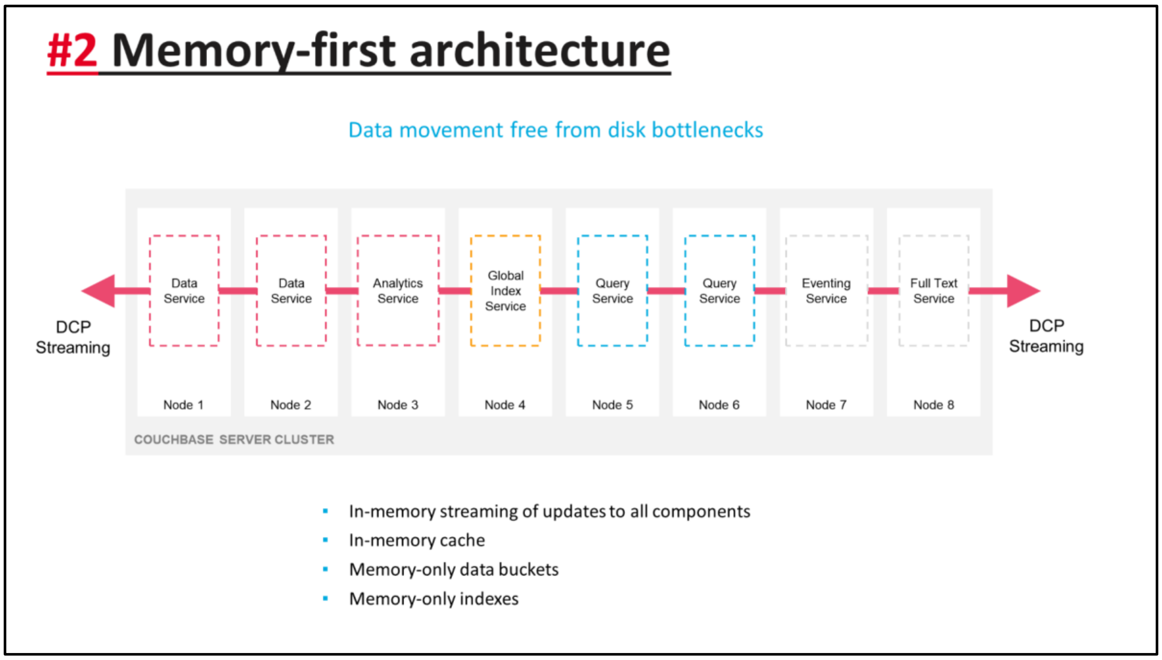




# #1 Elastic Scaling Architecture



©2019Couchbase. All rights reserved.



## #2 Memory-first architecture

Data movement free from disk bottlenecks

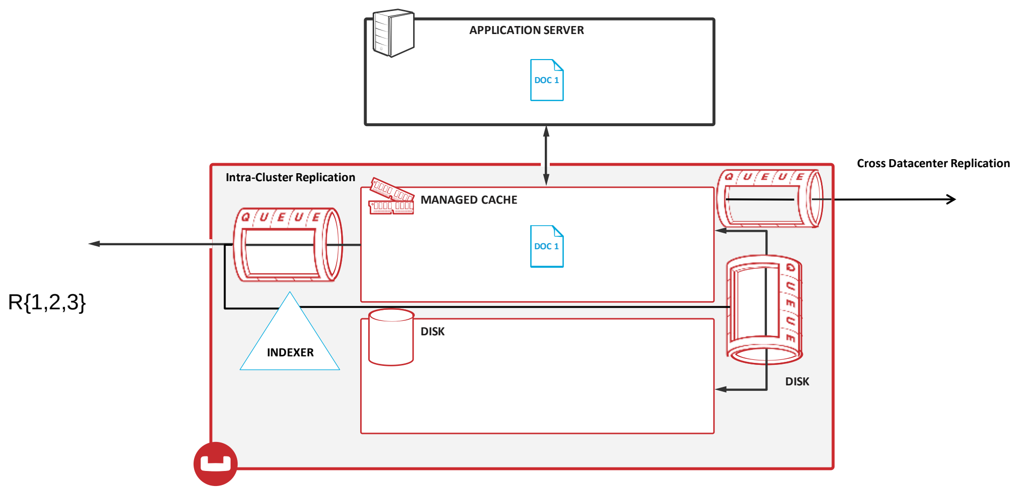
- In-memory streaming of updates to all components
- In-memory cache
- Memory-only data buckets
- Memory-only indexes

In-memory streaming of updates to all components
In-memory (cached) access to data and indexes
Memory-only indexes

## #3 Asynchronous approach to everything

- Persistence
- Intra-cluster Replication
- Inter-cluster Replication
- Global secondary Indexing updates
- Full-Text Search update
- Analytics service updates

## #3 Asynchronous approach to everything



## #3 Asynchronous approach to everything

### Configurable consistency per request / query

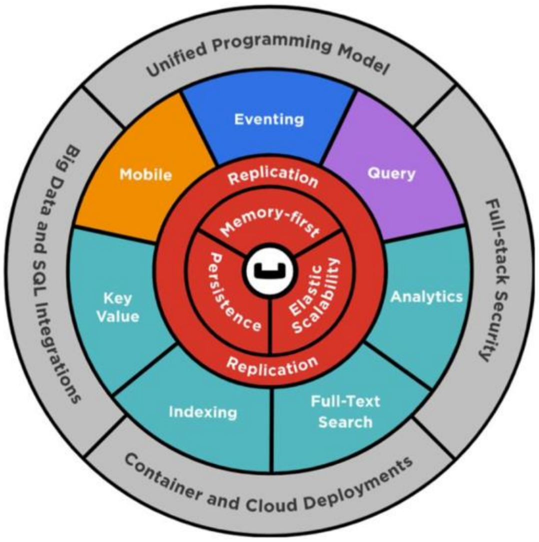
**Data Consistency**

Data access is strongly consistent within cluster
Eventually consistent across clusters

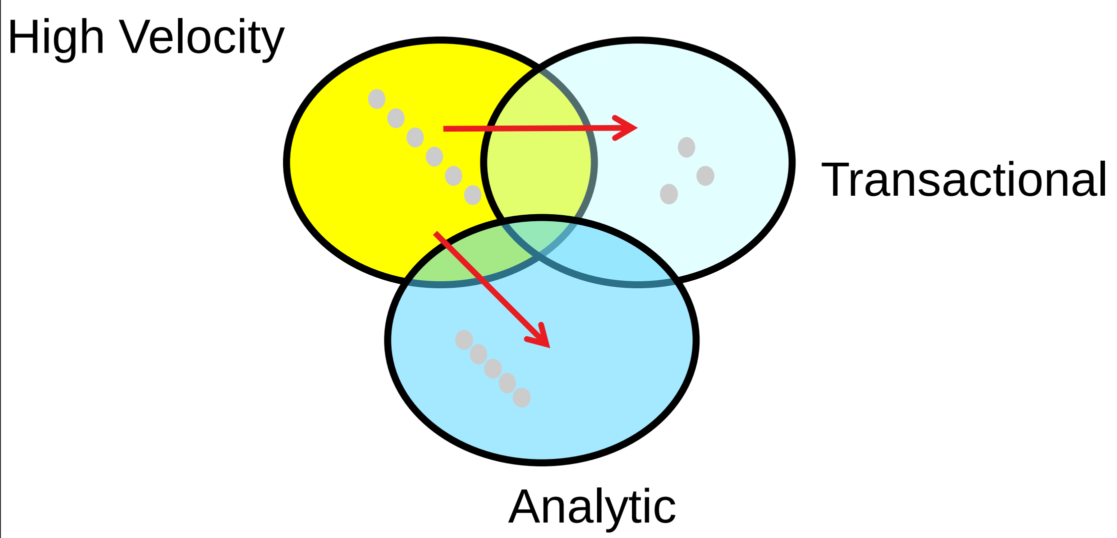
**Query Consistency**

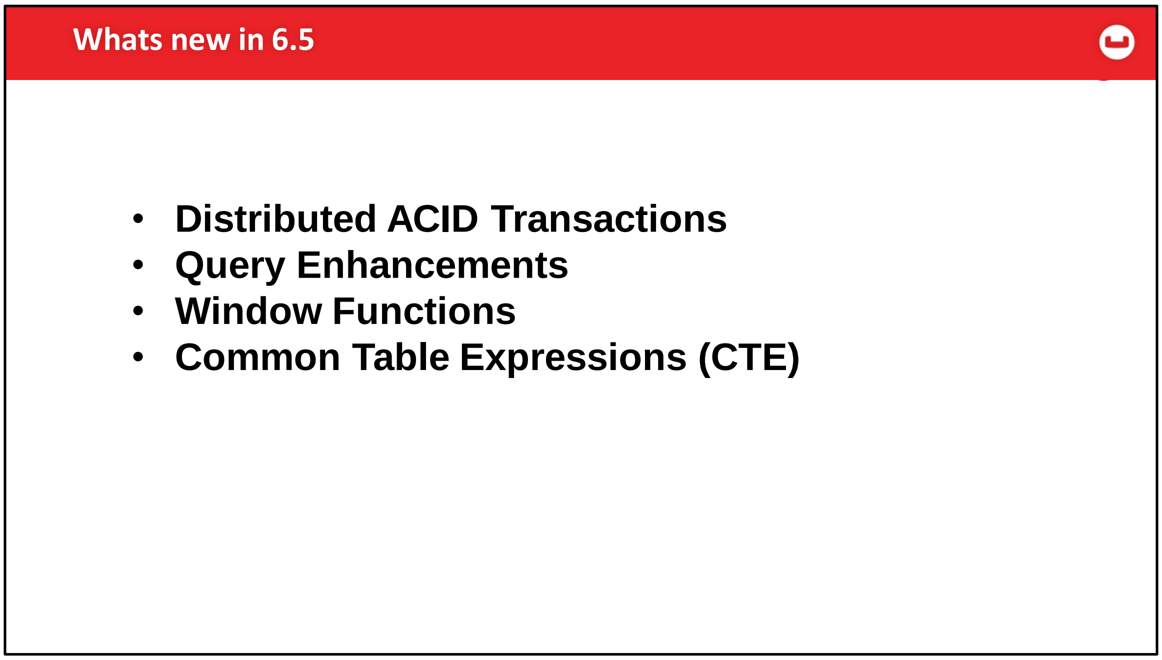
**Specify level of consistency for queries**

## Couchbase Data Platform



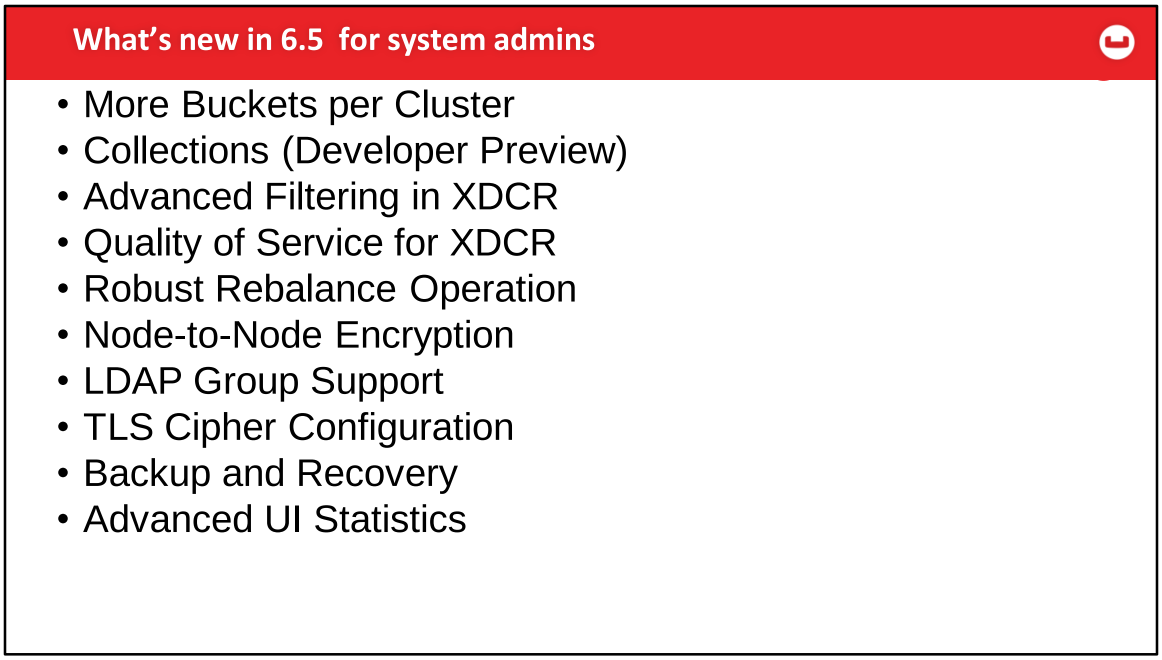
## Database use Case and model




## Whats new in 6.5

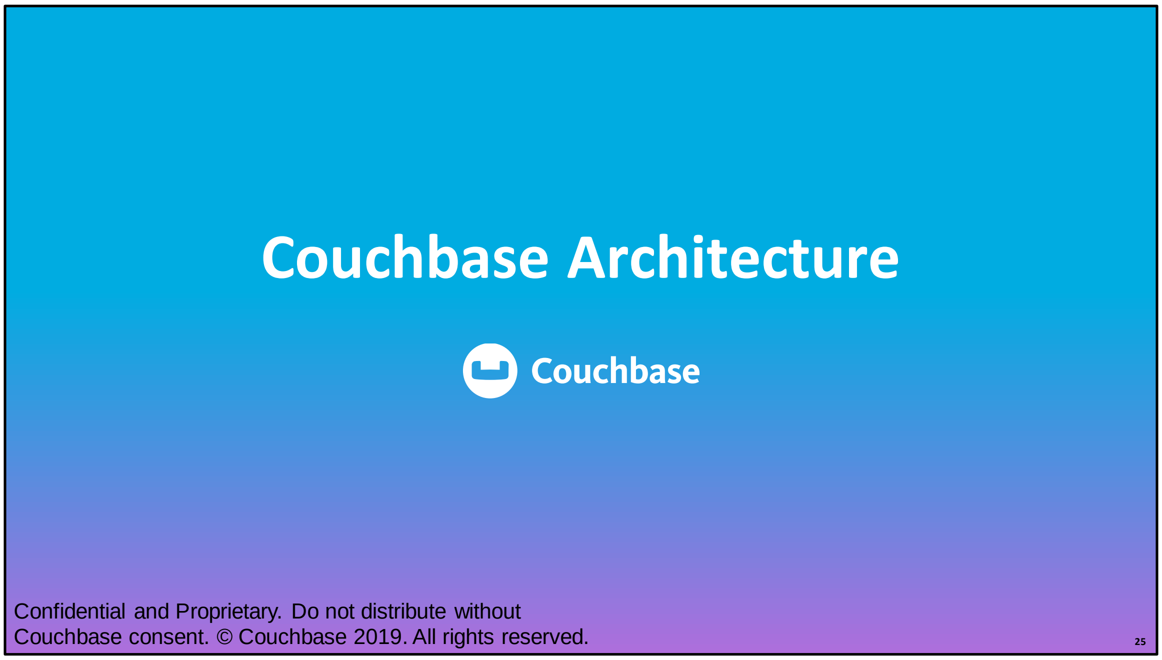
- **Distributed ACID Transactions**
- **Query Enhancements**
- **Window Functions**
- **Common Table Expressions (CTE)**

Distributed ACID Transactions
Query Enhancements
Window Functions
Common Table Expressions (CTE)
More Buckets per Cluster
Collections (Developer Preview)
Advanced Filtering in XDCR
Quality of Service for XDCR
Robust Rebalance Operation
Node-to-Node Encryption
LDAP Group Support
TLS Cipher Configuration
Backup and Recovery
Advanced UI Statistics



## What’s new in 6.5 for system admins

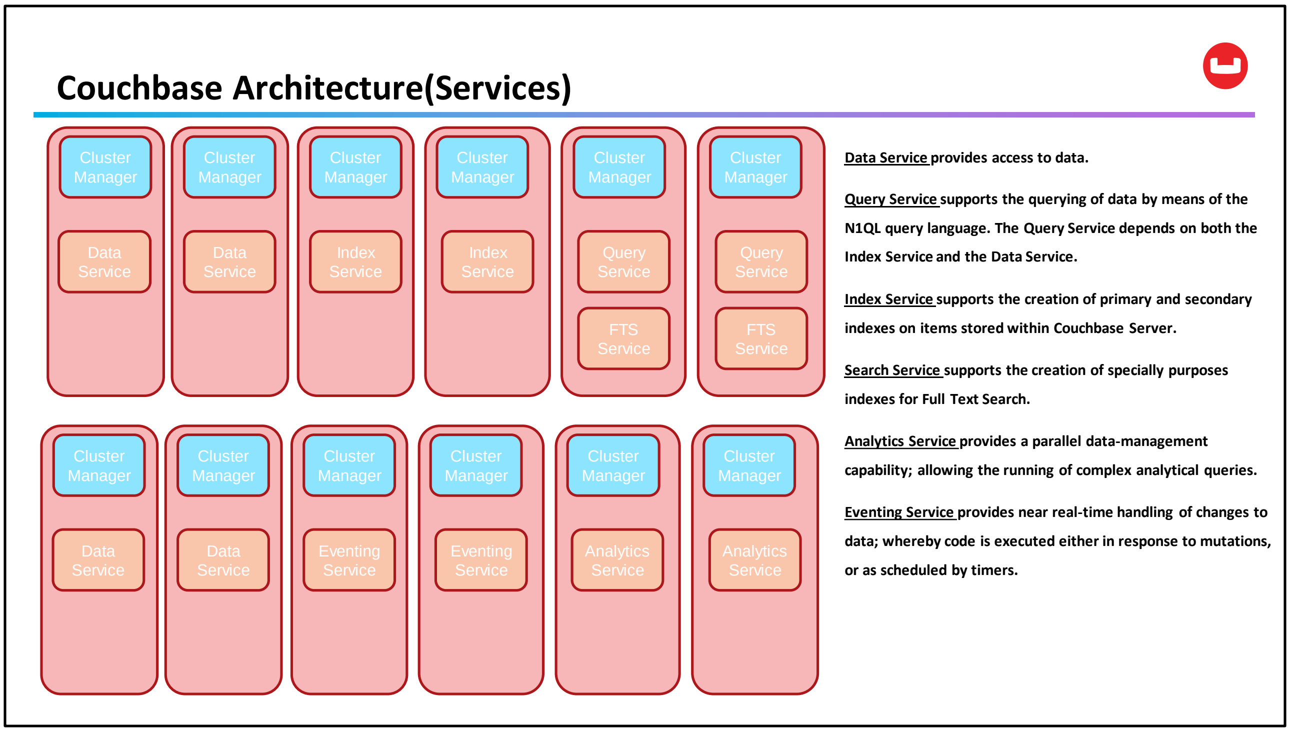
- More Buckets per Cluster
- Collections (Developer Preview)
- Advanced Filtering in XDCR
- Quality of Service for XDCR
- Robust Rebalance Operation
- Node-to-Node Encryption
- LDAP Group Support
- TLS Cipher Configuration
- Backup and Recovery
- Advanced UI Statistics



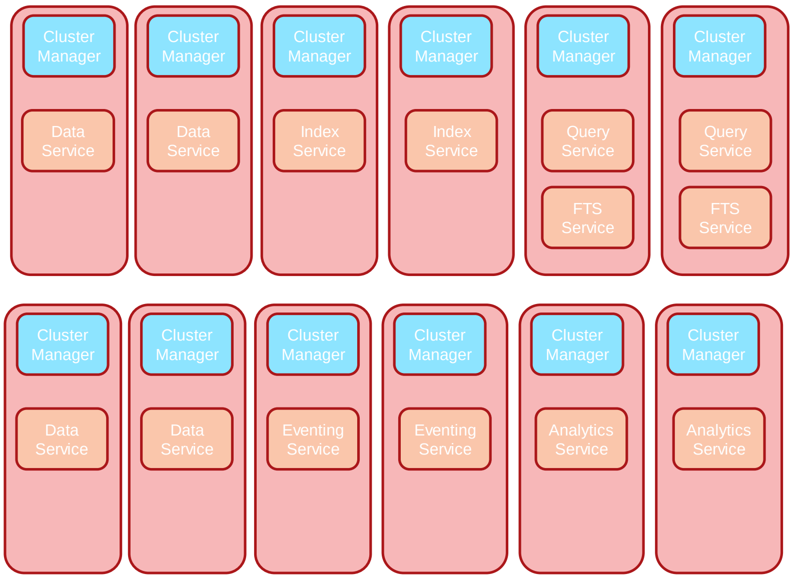
## Couchbase Architecture

Couchbase

Confidential and Proprietary. Do not distribute without
Couchbase consent. © Couchbase 2019. All rights reserved.



## Couchbase Architecture(Services)



Data Service provides access to data.

Query Service supports the querying of data by means of the N1QL query language. The Query Service depends on both the Index Service and the Data Service.

Index Service supports the creation of primary and secondary indexes on items stored within Couchbase Server.

Search Service supports the creation of specially purposes indexes for Full Text Search.

Analytics Service provides a parallel data-management capability; allowing the running of complex analytical queries.

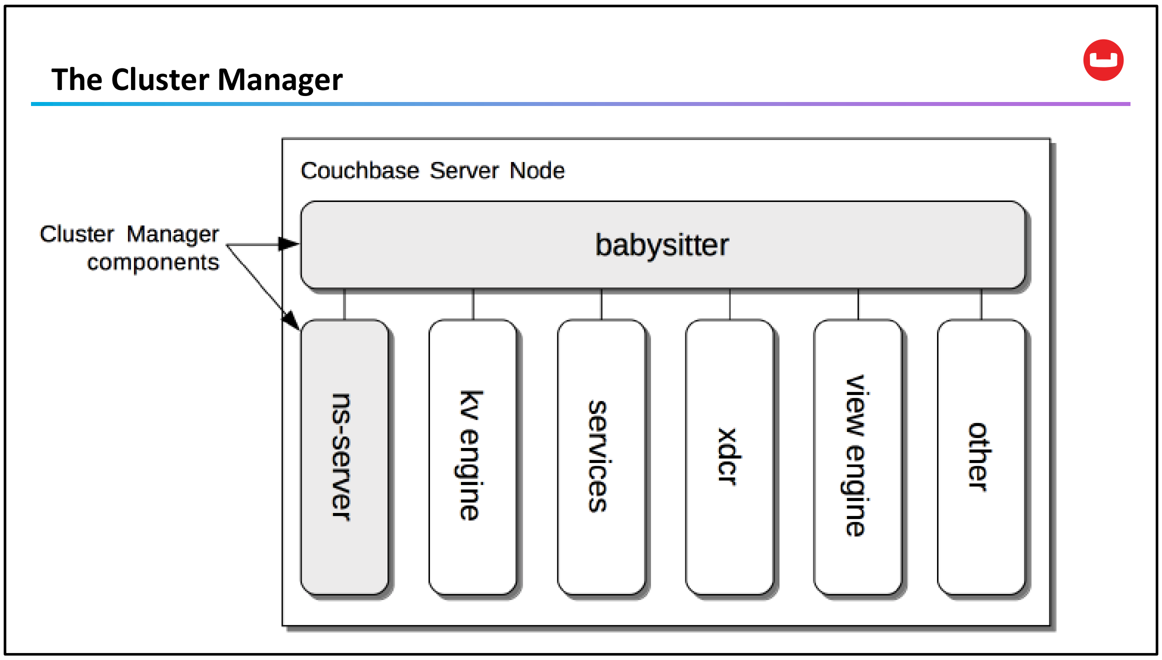
Eventing Service provides near real-time handling of changes to data; whereby code is executed either in response to mutations, or as scheduled by timers.

Data Service – Key Value Store and builds and maintains Distributed secondary indexes (MapReduce Views)
Indexing Engine – builds and maintains Global Secondary Indexes
Query Engine – plans, coordinates, and executes queries against either Global or Distributed indexes

Cluster Manager – configuration, heartbeat, statistics, RESTful Management interface

Data Service
The Data Service provides access to data.
Query Service
The Query Service supports the querying of data by means of the N1QL query language. The Query Service depends on both the Index Service and the Data Service.
Index Service
The Index Service supports the creation of primary and secondary indexes on items stored within Couchbase Server.
Search Service
The Search Service supports the creation of specially purposes indexes for Full Text

Search.
Analytics Service
The Analytics Service provides a parallel data-management capability; allowing the running of complex analytical queries.
Eventing Service
The Eventing Service provides near real-time handling of changes to data; whereby code is executed either in response to mutations, or as scheduled by timers.



## The Cluster Manager

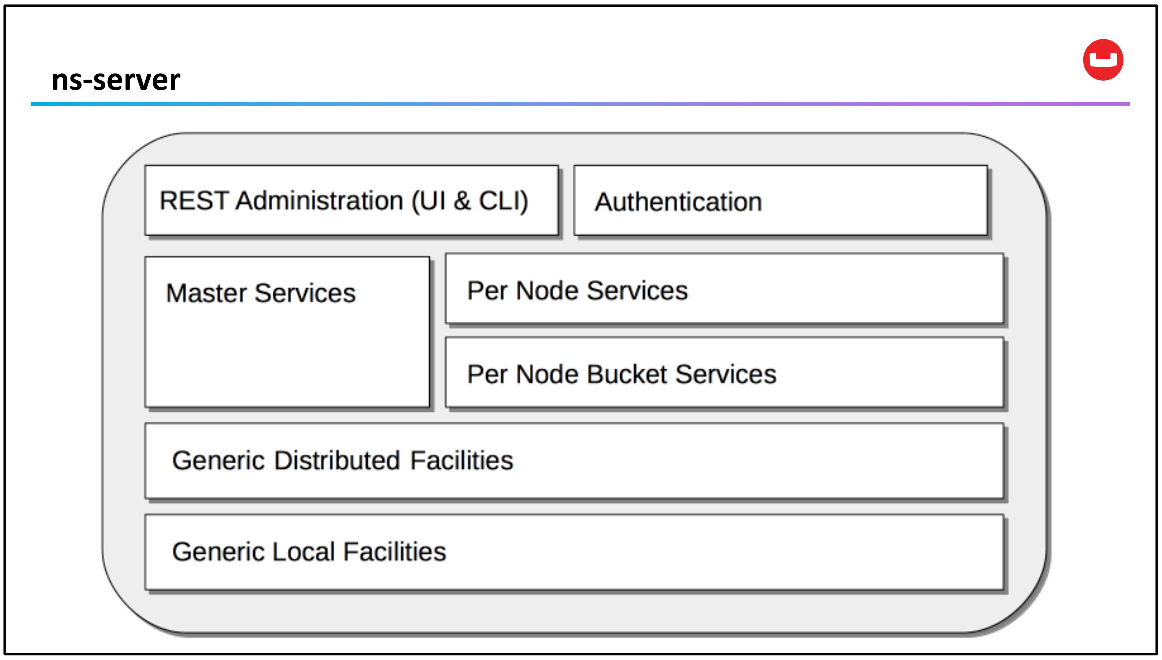
As shown, the Cluster Manager consists of two processes. The first, the *babysitter*, is responsible for maintaining a variety of Couchbase Server-processes, which indeed include the second Cluster Manager process, *ns-server*. The *babysitter* starts and monitors all of these processes, logging their output to the file babysitter.log (see Manage Logging, for information on logfile-locations). If any of the processes dies, the *babysitter* restarts it. The *babysitter* is not cluster-aware.

The processes for which the *babysitter* is responsible are:

- *ns-server*: Manages the node's participation in the cluster, as described in ns-server, below.
- *kv engine*: Runs as part of the Data Service, which must be installed on at least one cluster-node. Provides access to Data.
- *services*: One or more Couchbase Services that optionally run on the node.
- *xdcr*: The program for handling *Cross Data-Center Replication* (XDCR). This is installed with the Data Service, but runs as an independent OS-level process, separate from the Cluster Manager itself. See Availability, for information.
- *view engine*: The program for handling *Views*. This is installed with the Data Service, but runs as an independent OS-level process, separate from the Cluster

Manager itself. See Views, for more information.

- *other*: Various ancillary programs.



The modules are:
*REST Administration (UI and CLI)*: Supports administration of Couchbase Server, by means of a REST API; which itself underlies both the user interface provided by *Couchbase Web Console*, and the *Couchbase Command-Line Interface*.

*Authentication*: Protects node-resources with *Role-Based Access Control*. This is based on *credentials* (usernames and passwords) associated with system-defined *roles*, each of which is associated with a range of *privileges*.

*Master Services*: Manages cluster-wide operations; such as master and replica vBucket-placement, failover, node addition and subtraction, rebalance, cluster configuration-mapping, and aggregation of statistical data. Note that at any given time, only one of the instances of *Master Services* on a multi-node cluster is in charge: the instances having negotiated among themselves, to identify and elect the instance. Should the elected instance subsequently become unavailable, another takes over.
The *Master Services* are sometimes referred to as the *Orchestrator*.

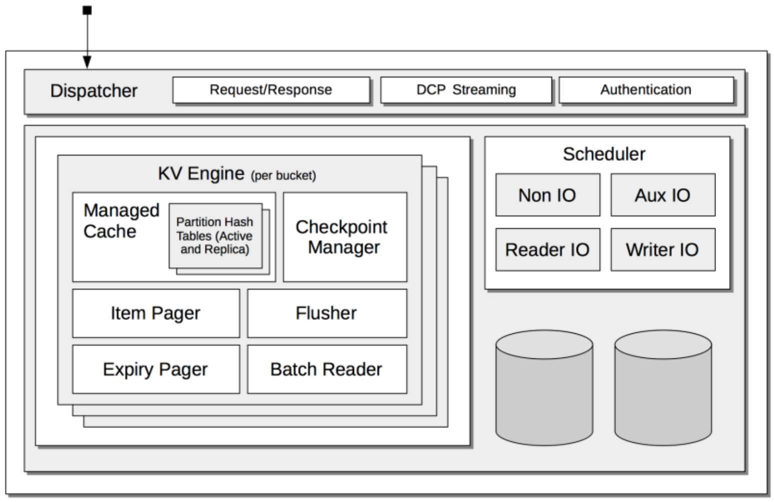
*Per Node Services*: Manages the health of the current node, and handles the monitoring and restart of its processes and services.

*Per Node Bucket Services*: Manages bucket-level operations for the current node; supporting replication, fail-over, restart, and statistics-collection.

*Generic Local Facilities*: Provides local configuration-management, libraries, workqueues, logging, clocks, ids, and events.

*Generic Distributed Facilities*: Supports node-discovery, configuration-messaging and alerts, replication, and heartbeat-transmission.

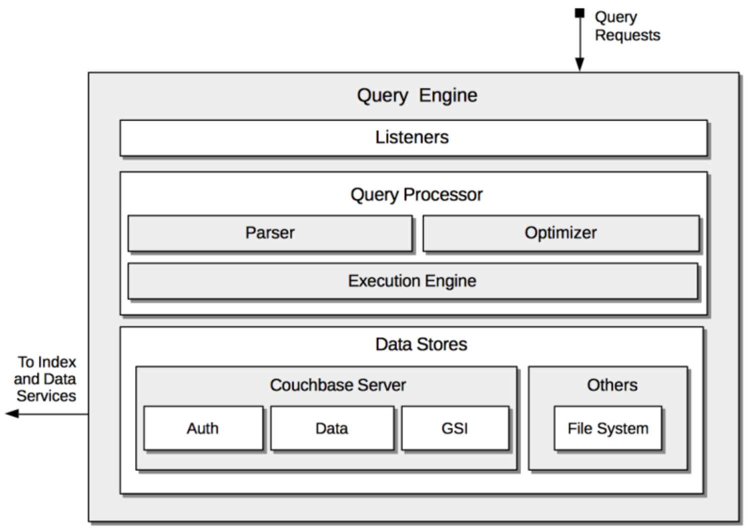
## The data Service



The principal components are:
**Dispatcher**: Manages networking, by handling each **Request** for data, and providing the **Response**. Also streams data to other nodes within the cluster, and to other clusters, by means of the **DCP** protocol; and handles **Authentication**.
For information on **DCP Streaming**, see Clusters and Availability. For information on **Authentication**, see Authentication. For information on ports used by the dispatcher, see Network Configuration.
**KV Engine**: A collection of facilities that is provided for each bucket on the cluster. These are:
**Managed Cache**: The memory allocated for the bucket, according to an established quota. This contains *Partition Hash Tables*, whereby the location of bucket-items, in memory and on disk, on different nodes across the cluster, is recorded. When written, items enter the cache, and subsequently are placed onto a replication queue, so as to be replicated to one or more other nodes; and (in the case of items for Couchbase buckets) onto a disk queue, so as to be written to disk.
For information on memory quotas, see Memory. For information on how items from a bucket are assigned to *vBuckets* across a cluster, see vBuckets. For illustrations of how items are written and updated for Couchbase buckets, see Memory and Storage.
**Checkpoint Manager**: Keeps track of item-changes, using data structures named

*checkpoints*. Changes already made to items in memory, but not yet placed on the replication and disk queues, are recorded.
**Item Pager**: Ejects from memory items that have not recently been used, in order to free up space, as required. For further information, see Memory.
**Flusher**: Deletes every item in the bucket. For information on how to activate, see Flush a Bucket.
**Expiry Pager**: Scans for items that have expired, and erases them from memory and disk; after which, a *tombstone* remains for a default period of 3 days. The expiry pager runs every 60 minutes by default: for information on changing the interval, see cbepctl set flush_param. For more information on item-deletion and tombstones, see Expiration.
**Batch Reader**: Enhances performance by combining changes made to multiple items into *batches*, which are placed on the disk queue, to be written to disk.
**Scheduler**: A pool of threads, mainly purposes for handling I/O. The threads are divided into four kinds, which run independently of and without effect on one another:
**Non IO**: Tasks private to the scheduler that do not require disk-access; including connection-notification, checkpoint removal, and hash-table resizing.
**Aux IO**: Fetch, scan, and backfill tasks.
**Reader IO**: Threads that read information from disk.
**Writer IO**: Threads that write information to disk.

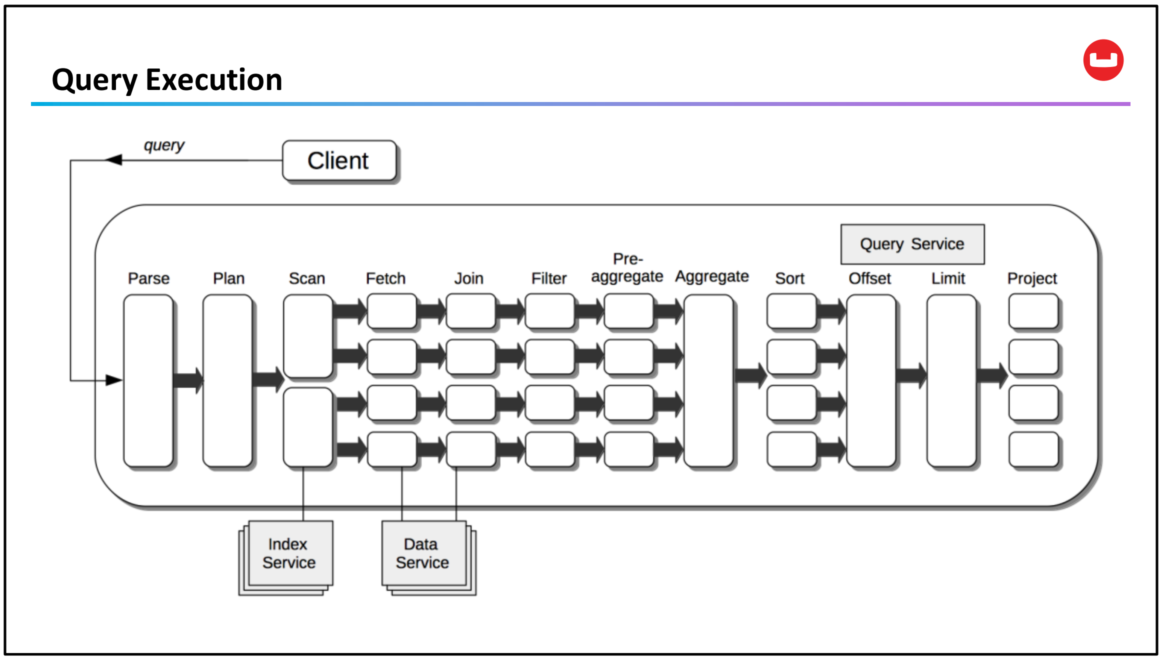
## The Query Service



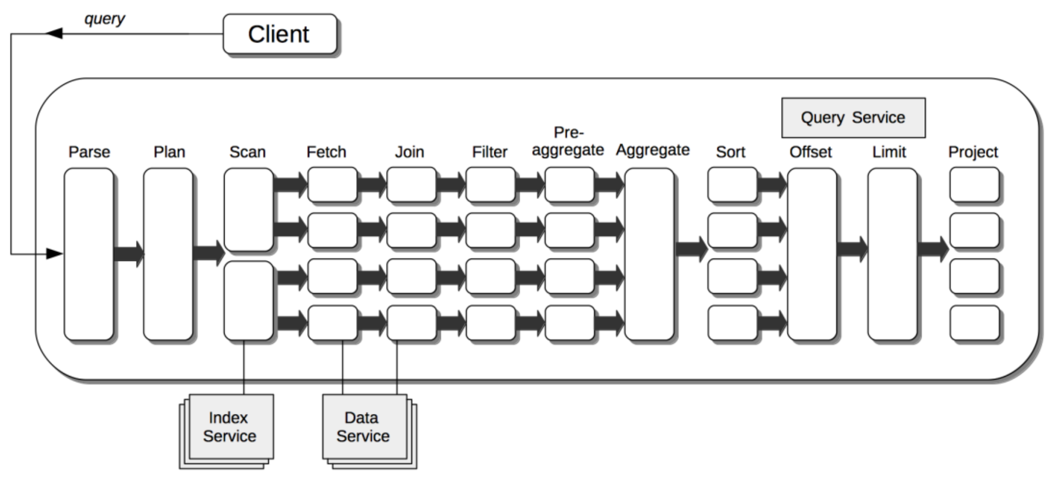
The principal components are:
**Listeners**: Concurrent query requests are received on ports 8093 and 18093

**Query Processor**: Responsible for applying the **Parser** to incoming queries, in order to determine whether each is a valid statement. Also employs the **Optimizer**, which evaluates available execution paths, so determining the path of lowest latency; generates a query-execution plan that uses the lowest-latency path; and assembles the plan into a series of operators.
The **Execution Engine** receives the operators, and executes them — in parallel where possible.

**Data Stores**: Provides access to various data-sources.
The **Couchbase Server** store is used to access the data and indexes on Couchbase Server, and to handle authentication.
Other data stores are also included, such as the store for the local filesystem



## Query Execution



The client’s N1QL query is shown entering the Query Service at the left-hand side. The Query Processor performs its

**Parse** routine, to validate the submitted statement, then creates the execution **Plan**.

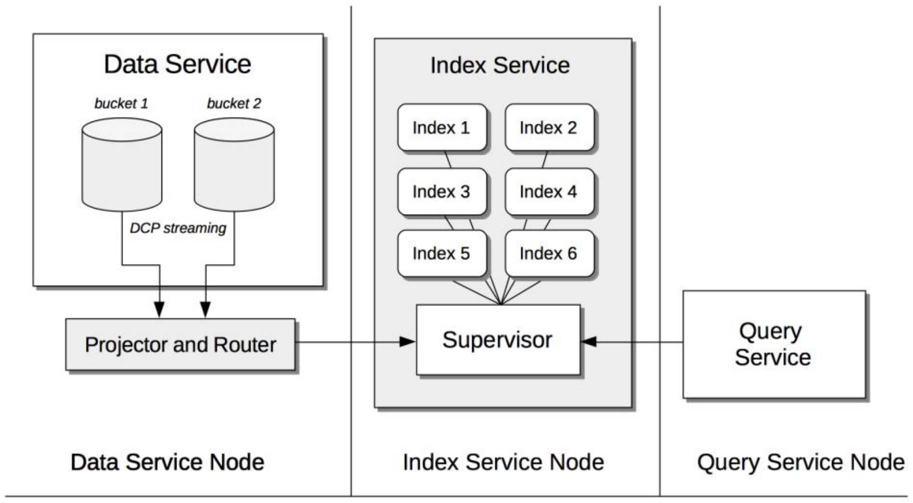
**Scan** operations are then performed on the relevant index, by accessing the **Index Service**.

Next, **Fetch** operations are performed by accessing the **Data Service**, and the data duly returned is used in **Join** operations.

The Query Service continues by performing additional processing, which includes **Filter**, **Aggregate**, and **Sort** operations.

Note the degree of parallelism with which operations are frequently performed, represented by the vertically aligned groups of right-pointing arrows

## The Index Service



The illustration depicts the following:
**Data Service**: Uses the DCP protocol to stream data-mutations to the **Projector and Router** process, which runs as part of the Index Service, on each Data Service node.

**Projector and Router**: Provides data to the Index Service, according to the index-definitions provided by the Index Service Supervisor.
When the **Projector and Router** starts running on the Data Service-node, the Data Service streams to the **Projector and Router** copies of all mutations that occur to bucket-items. Prior to the creation of any indexes, the **Projector and Router** takes no action.
When an index is first created, the Index Service **Supervisor** contacts the **Projector and Router**; and passes to it the corresponding index-definitions. The **Projector and Router** duly contacts the Data Service, and extracts data from the fields specified by the index-definitions. It then sends the data to the **Supervisor**, so that the index can be populated.
Subsequently, the **Projector and Router** continuously examines the stream of mutations provided by the Data Service. When this includes a mutation to an indexed field, the mutated data is passed by the **Projector and Router** to the **Supervisor**, and the index thereby updated.

**Supervisor**: The main program of the Index Service; which passes index-definitions to the **Projector and Router**, creates and stores indexes, and handles mutations sent from the **Projector and Router**.

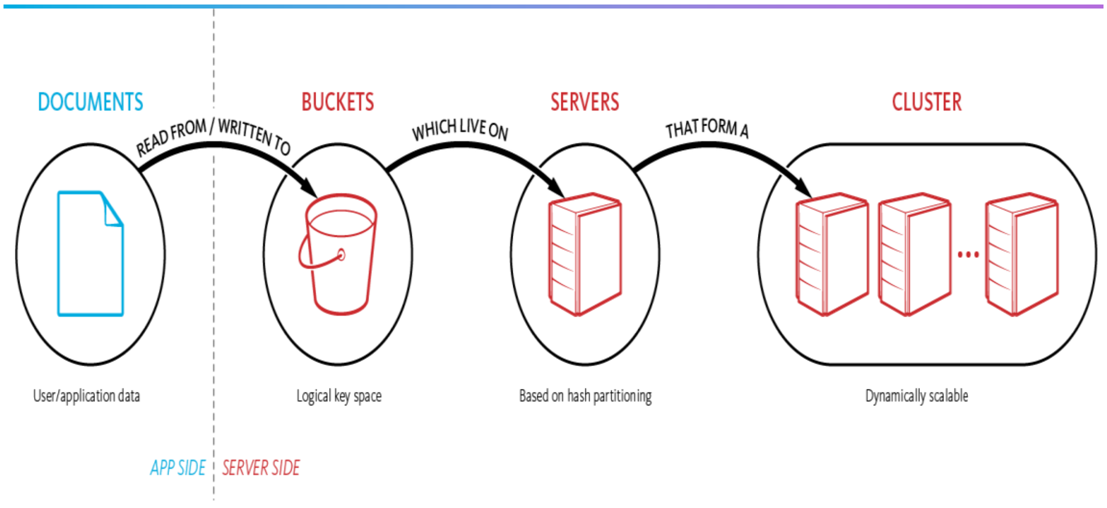
**Query Service**: Passes to the **Supervisor** clients' create-requests and queries, and handles the responses

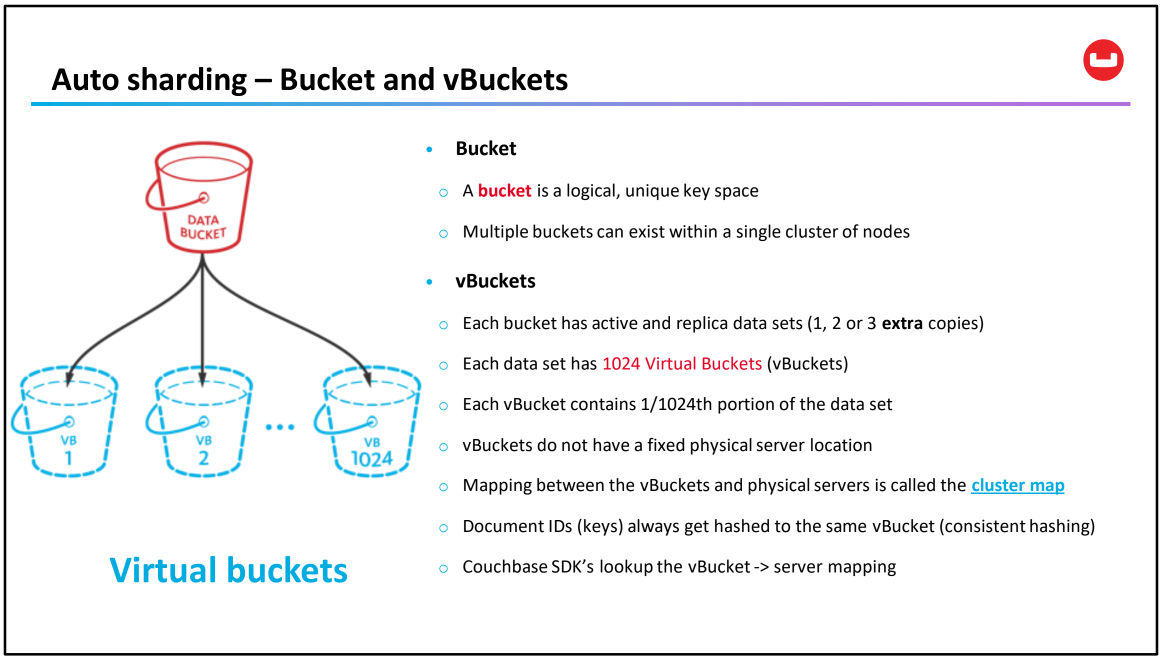
## Buckets

Couchbase

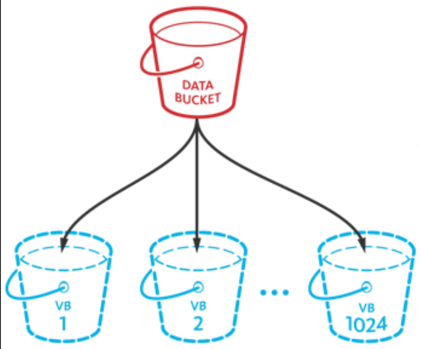
Confidential and Proprietary. Do not distribute without
Couchbase consent. © Couchbase 2019. All rights reserved.

## Storing And Retrieving Documents





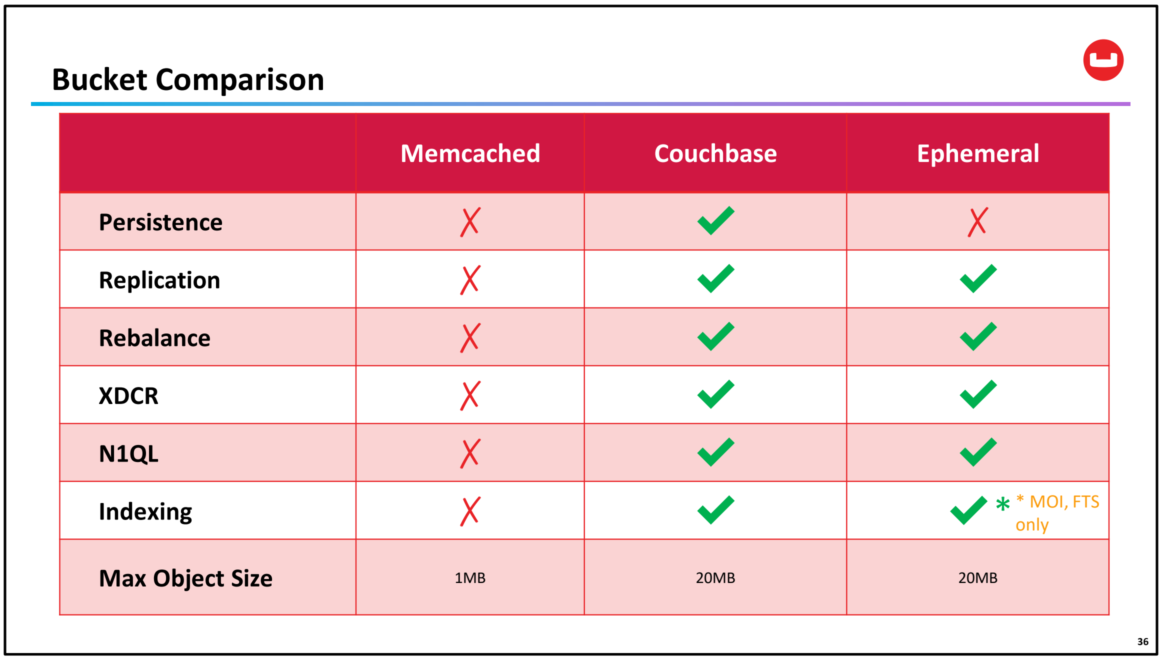
## Auto sharding – Bucket and vBuckets



Virtual buckets

- **Bucket**
  - A **bucket** is a logical, unique key space
  - Multiple buckets can exist within a single cluster of nodes
- **vBuckets**
  - Each bucket has active and replica data sets (1, 2 or 3 **extra** copies)
  - Each data set has 1024 Virtual Buckets (vBuckets)
  - Each vBucket contains 1/1024th portion of the data set
  - vBuckets do not have a fixed physical server location
  - Mapping between the vBuckets and physical servers is called the cluster map
  - Document IDs (keys) always get hashed to the same vBucket (consistent hashing)
  - Couchbase SDK's lookup the vBucket -> server mapping

Bucket
A bucket is a logical, unique key space
Multiple buckets can exist within a single cluster of nodes
vBuckets
Each bucket has active and replica data sets (1, 2 or 3 extra copies)
Each data set has 1024 Logical partitions/Virtual Buckets (vBuckets)
Each vBucket contains 1/1024th portion of the data set =Logical Partition
vBuckets do not have a fixed physical server location
Mapping between the vBuckets and physical servers is called the cluster map
Document IDs (keys) always get hashed to the same vBucket (consistent hashing)
Couchbase SDK's lookup the vBucket -> server mapping



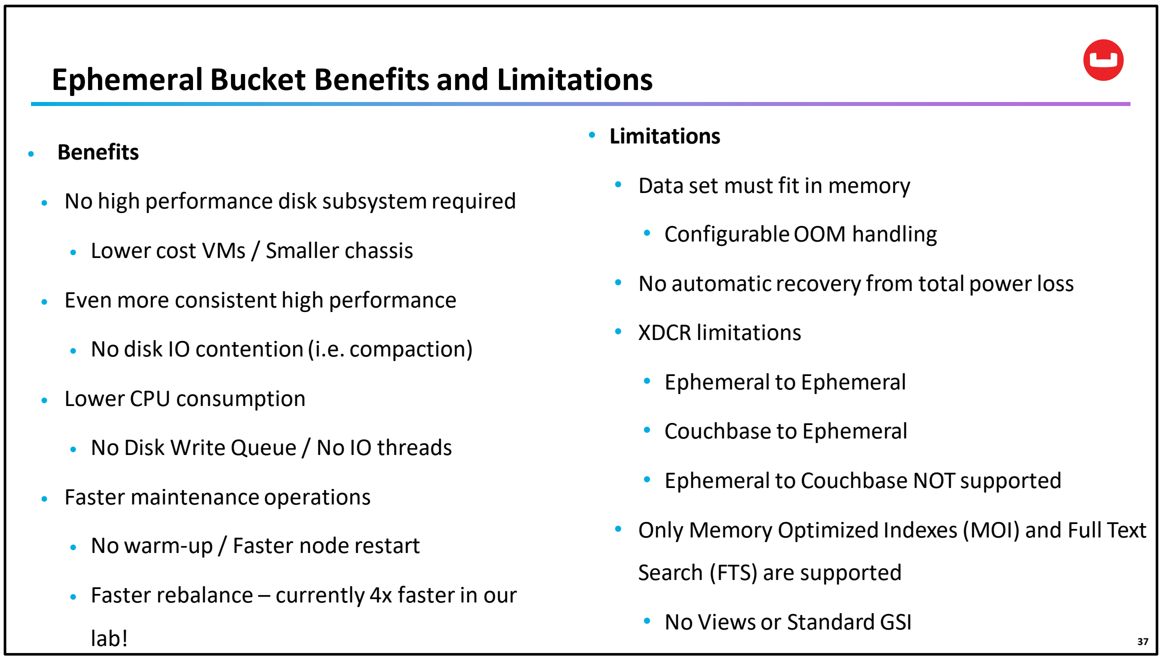
## Bucket Comparison

| | Memcached | Couchbase | Ephemeral |
|---|---|---|---|
| **Persistence** | ✗ | ✓ | ✗ |
| **Replication** | ✗ | ✓ | ✓ |
| **Rebalance** | ✗ | ✓ | ✓ |
| **XDCR** | ✗ | ✓ | ✓ |
| **N1QL** | ✗ | ✓ | ✓ |
| **Indexing** | ✗ | ✓ | ✓ * * MOI, FTS only |
| **Max Object Size** | 1MB | 20MB | 20MB |

36

Couchbase Server keeps items in Buckets. Before an item can be saved, a bucket must exist for it. Buckets can be created by means of the Couchbase Web Console, the REST API, or the CLI. Each bucket is assigned a name at its creation: this name is referenced by the application or user wishing to save or access items within it.

The types of bucket most frequently used are Couchbase and Ephemeral buckets. Couchbase buckets exist both in memory and on disk. Ephemeral buckets exist only in memory. Buckets can be configured to maintain data in compressed format, so as to maximize resource-effectiveness. Documents within a bucket can be assigned a Time To Live, to ensure that they expire after a specified time-period, and that their data is thus made permanently inaccessible.



## Ephemeral Bucket Benefits and Limitations

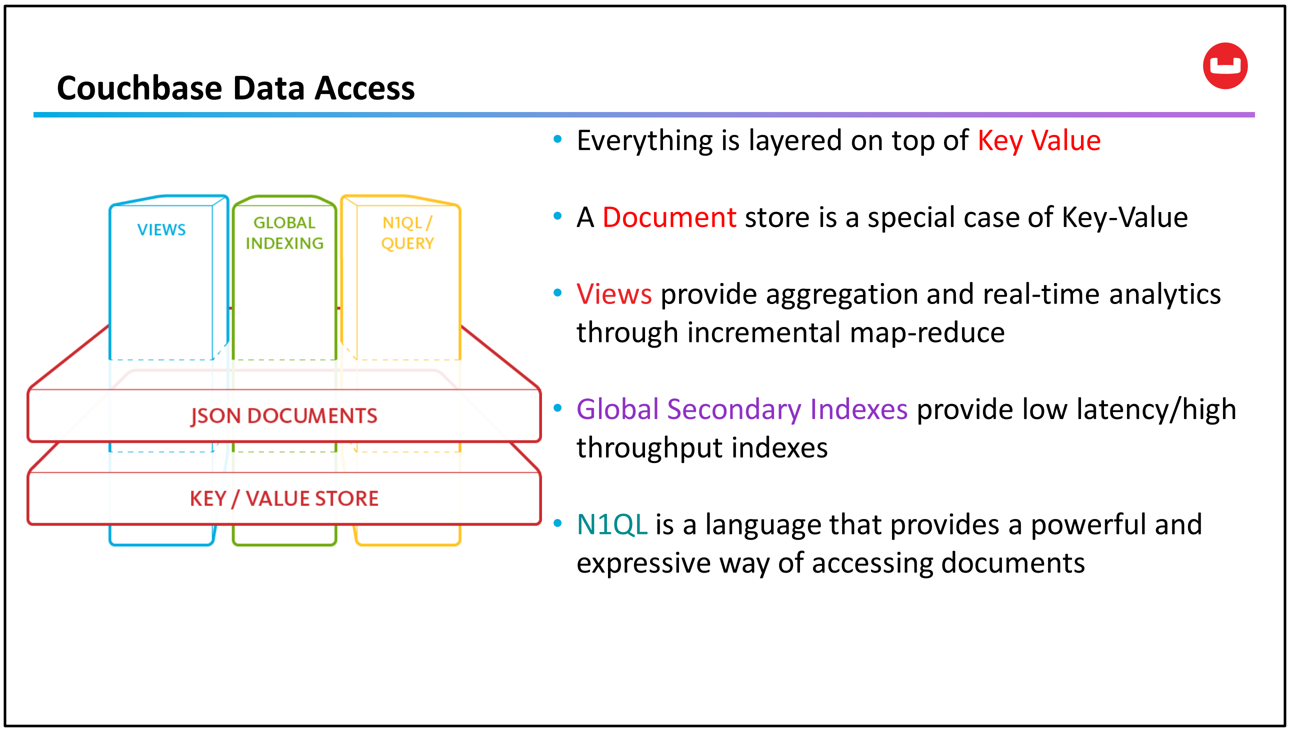
- **Benefits**
  - No high performance disk subsystem required
    - Lower cost VMs / Smaller chassis
  - Even more consistent high performance
    - No disk IO contention (i.e. compaction)
  - Lower CPU consumption
    - No Disk Write Queue / No IO threads
  - Faster maintenance operations
    - No warm-up / Faster node restart
    - Faster rebalance – currently 4x faster in our lab!

- **Limitations**
  - Data set must fit in memory
    - Configurable OOM handling
  - No automatic recovery from total power loss
  - XDCR limitations
    - Ephemeral to Ephemeral
    - Couchbase to Ephemeral
    - Ephemeral to Couchbase NOT supported
  - Only Memory Optimized Indexes (MOI) and Full Text Search (FTS) are supported
    - No Views or Standard GSI

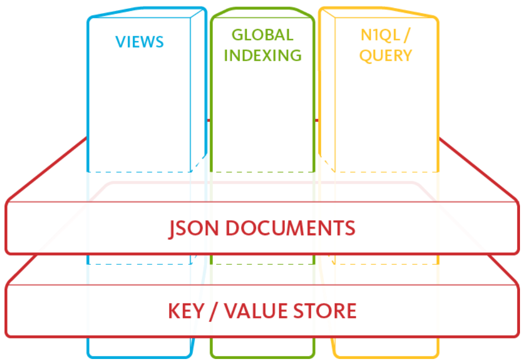
37

Benefits
No high performance disk subsystem required
Lower cost VMs
Smaller chassis
Even more consistent high performance
No disk IO contention (i.e. compaction)
Lower CPU consumption
No Disk Write Queue
No IO threads
Faster maintenance operations
No warm-up
Faster node restart
Faster rebalance – currently 4x faster in our lab!
Limitations
Data set must fit in memory
Configurable OOM handling
No automatic recovery from total power loss
Backups and XDCR still supported!
XDCR limitations

Ephemeral to Ephemeral
Couchbase to Ephemeral
Ephemeral to Couchbase NOT supported
Only Memory Optimized Indexes (MOI) and Full Text Search (FTS) are supported
No Views or Standard GSI



## Couchbase Data Access

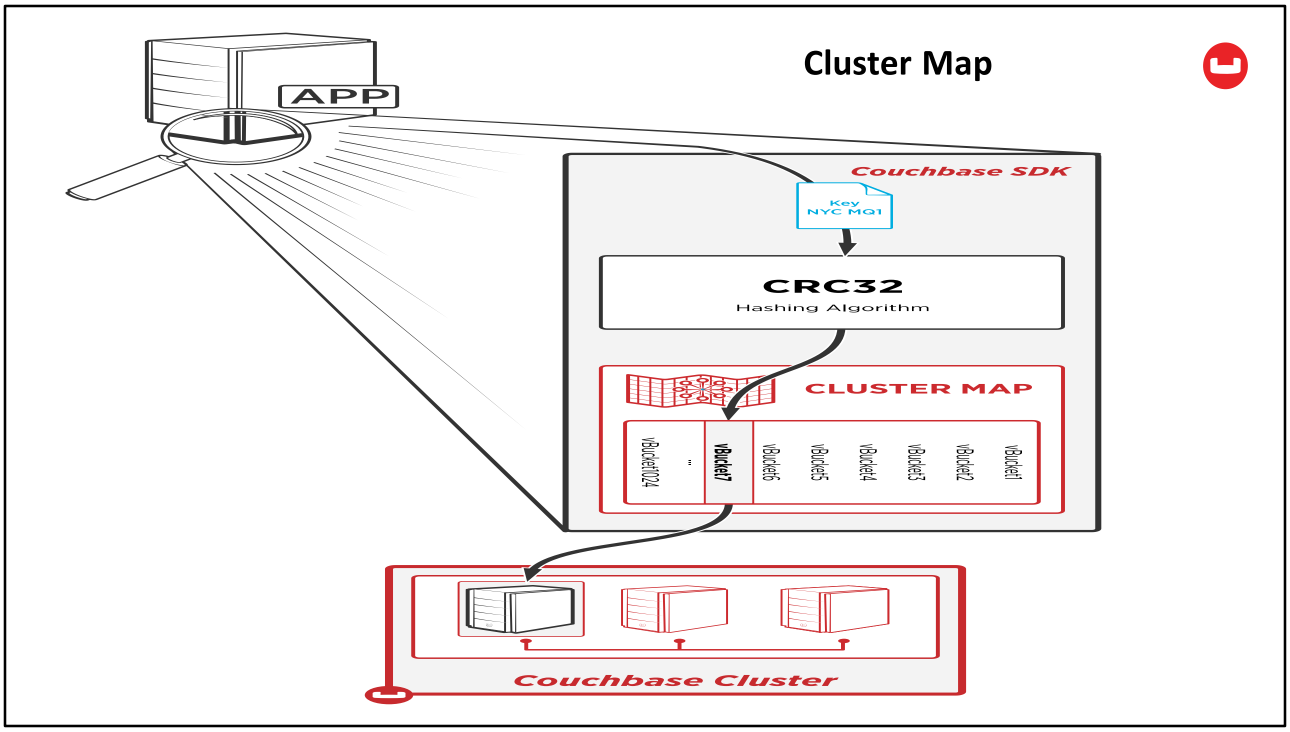


- Everything is layered on top of Key Value
- A Document store is a special case of Key-Value
- Views provide aggregation and real-time analytics through incremental map-reduce
- Global Secondary Indexes provide low latency/high throughput indexes
- N1QL is a language that provides a powerful and expressive way of accessing documents

**KEY POINT: COUCHBASE HAS MULTIPLE DIFFERENT AND LAYERED WAYS TO ACCESS DATA. EACH TO BE USED FOR WHAT THEY ARE BEST AT AND CAN BE COMBINED TO GET THE FUNCTIONALITY AND PERFORMANCE CHARACTERISTICS REQUIRED.**

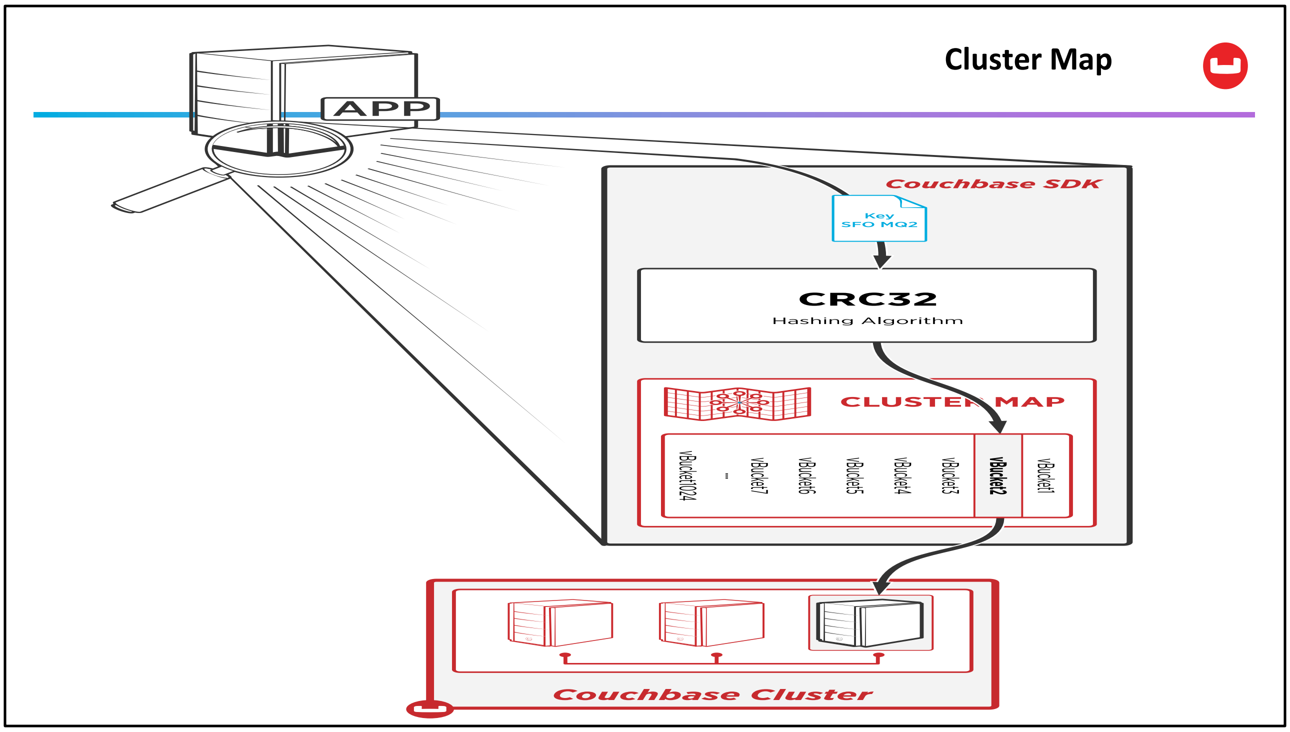
Given that the SDK is abstracting the mechanics of connecting and executing your queries, what's left for you as a developer is to understand Couchbase's document model and how query is layered on it.
<click>
At the very lowest level of that document model is key/value. The data stored as key/value can be anything, so long as the keys are less than 250 bytes and values are less than 20MB.
<click>
A special class of key/value are JSON documents. This is the object format that Couchbase and its clients are able to understand. If you format your objects as JSON, the values cease to be opaque to Couchbase and we can operate on them intelligently.
<click>
The first type of intelligence we can apply to JSON documents is Views. Views allow you to opportunistically index your documents according to business logic you insert

into Couchbase.
<click>

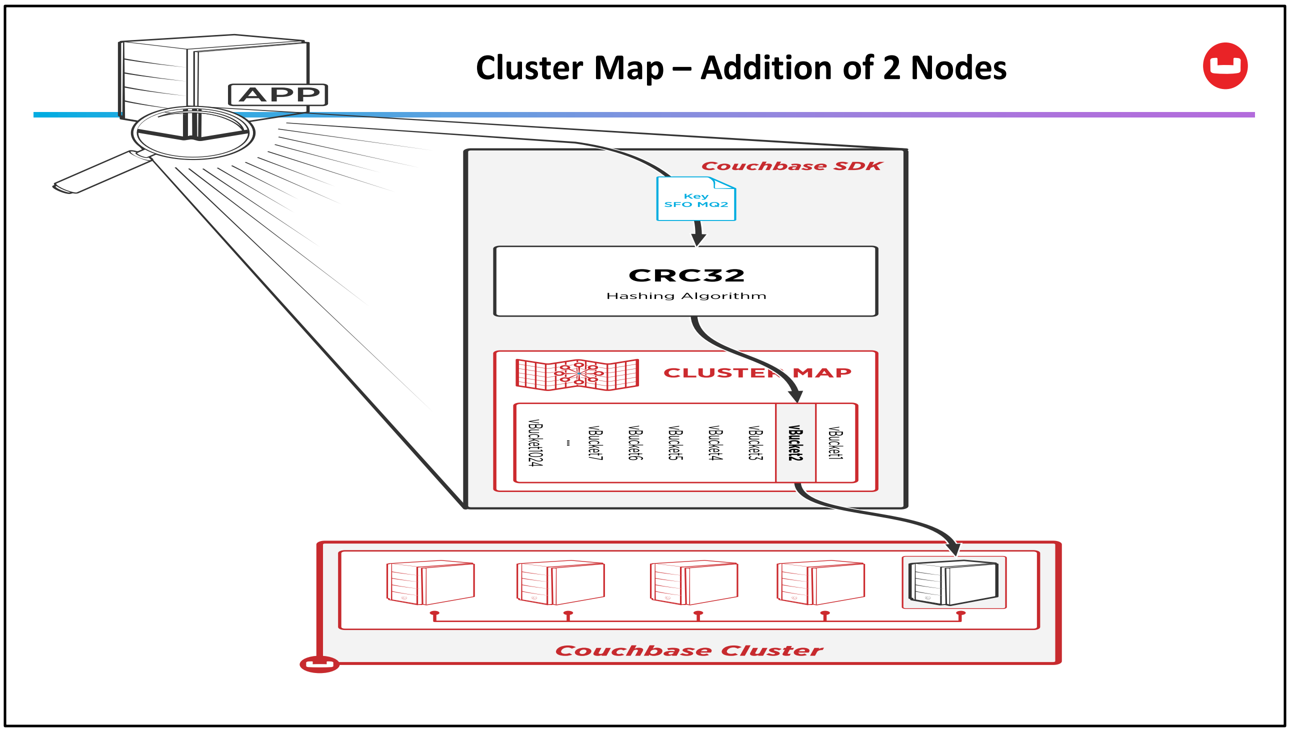


The application makes a call for a key called NYC MQ1
We run the key through the crc 32 function and the result of that hash function is that it points to vbucket3
Which in turn points to couchbase server number 1

## Cluster Map



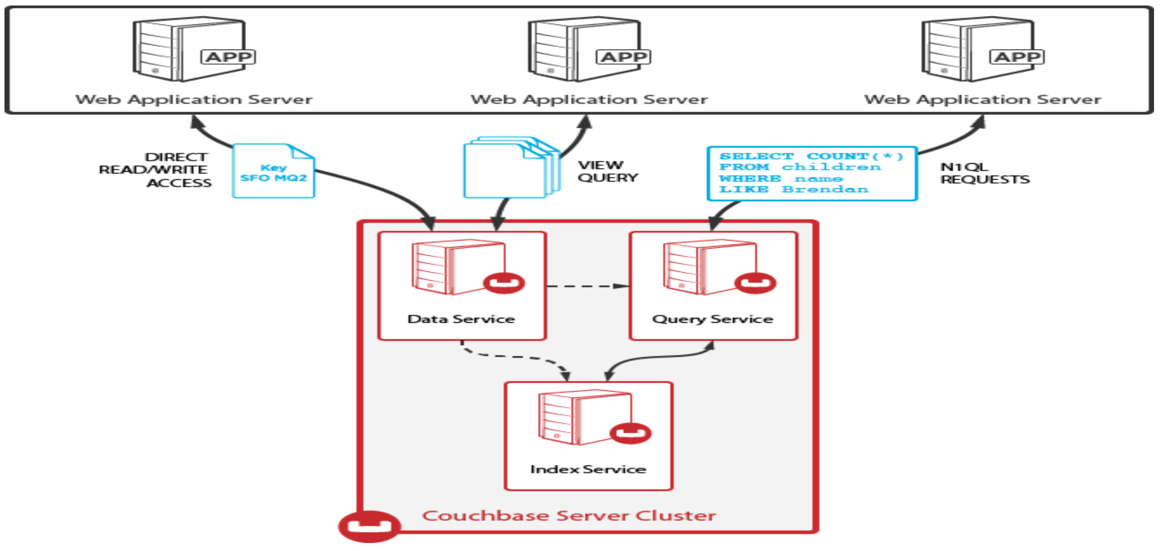
We now run a different key through through the has and we now come up with different vbucket, vbucket 4 and that points to server 3



## Cluster Map – Addition of 2 Nodes

We now run a different key through through the has and we now come up with different vbucket, vbucket 4 and that points to server 3

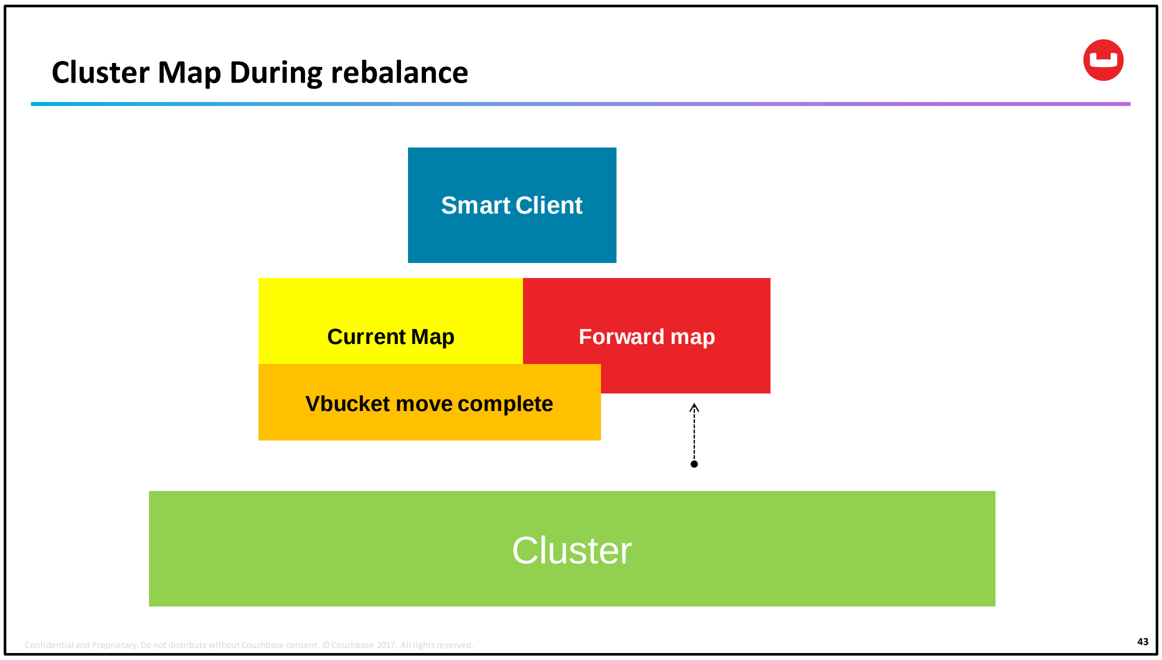
## Application to Database Interaction



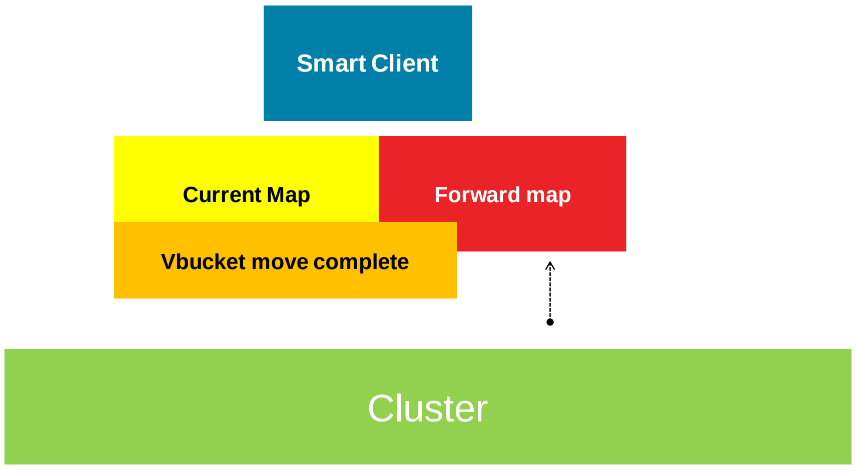
```
SELECT COUNT(*)
FROM children
WHERE name
LIKE Brendan
```

KEY POINT: Applications communicate directly to the services they need to fulfill the application request and the application does not need to be topology aware as the SDK has that already.

- Single node type, services defined dynamically
- One node acts the same as 100, just the services are spread out in the cluster
- Query service accesses Index and Data to formulate response
- All query and document access is topology aware and dynamically scalable
- Develop with one node, deploy against multiple production nodes
- The Couchbase SDK handles knowing about where in the cluster it needs to go to satisfy whatever the application is requesting, be I CRUD or cluster management.



## Cluster Map During rebalance



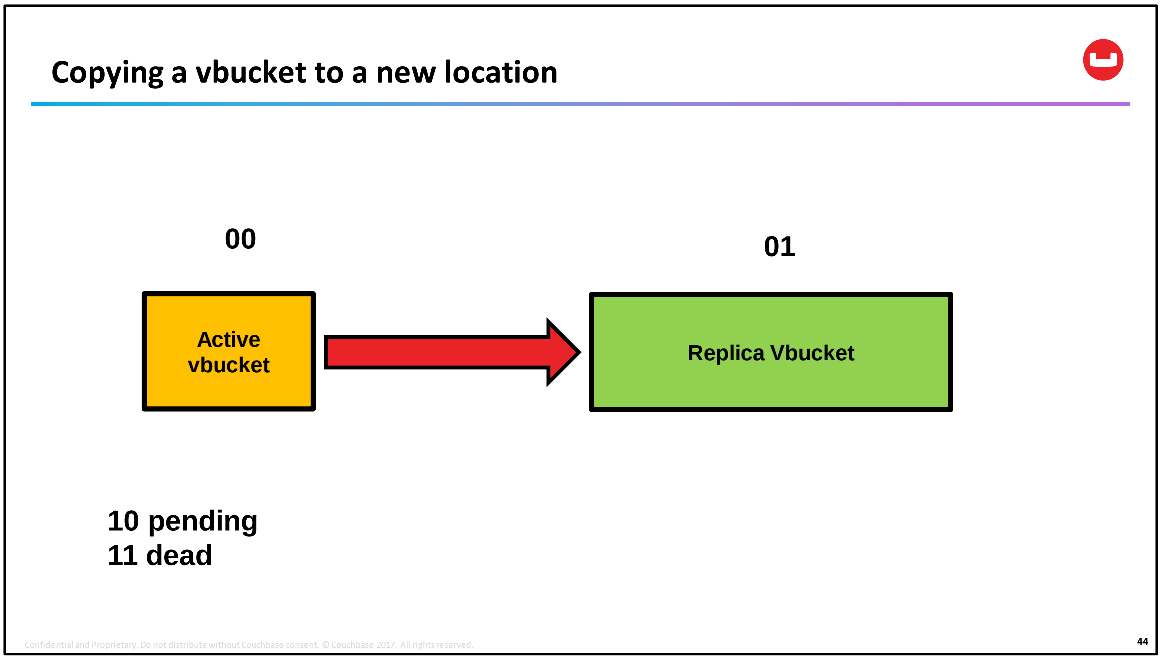
Confidential and Proprietary. Do not distribute without Couchbase consent. © Couchbase 2017. All rights reserved.

Rebalance behind-the-scenes

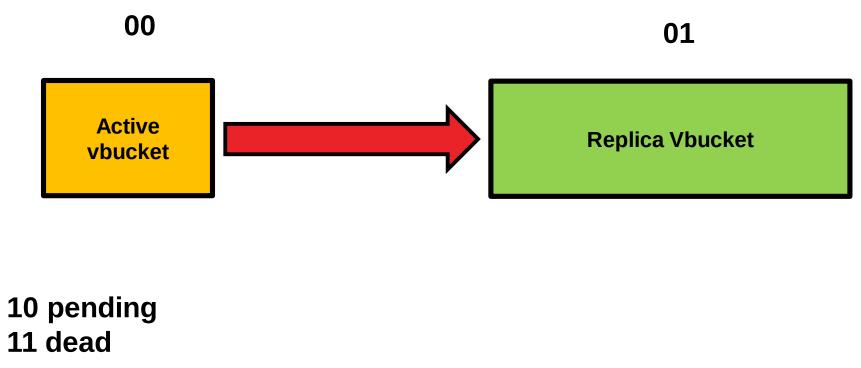
The rebalance process is managed through a specific process called orchestrator .

The orchestrator examines the current vBucket map and then combines that information with the node additions and removals in order to create a new vBucket map.

The orchestrator starts the process of moving the individual vBuckets from the current vBucket map to the new vBucket structure. It only starts the process while the nodes themselves are responsible for actually performing the movement of data between the nodes. The aim is to make the newly calculated vBucket map match the current situation.



## Copying a vbucket to a new location



Confidential and Proprietary. Do not distribute without Couchbase consent. © Couchbase 2017. All rights reserved.

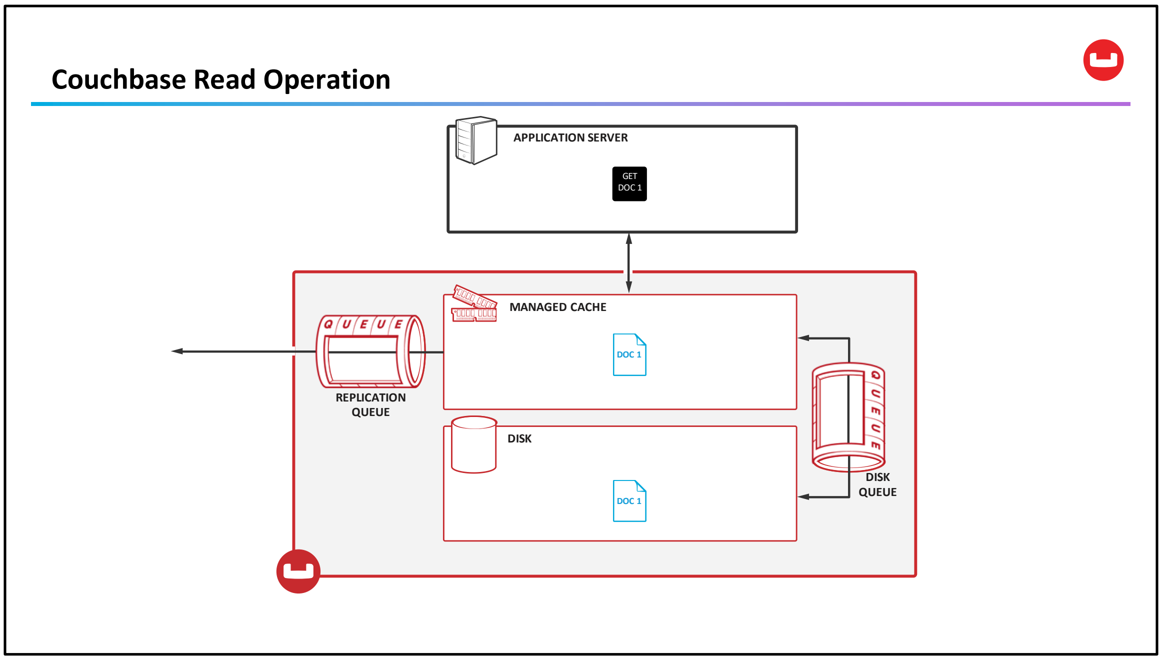
Each vBucket is moved independently, and a number of vBuckets can be migrated simultaneously in parallel between the different nodes in the cluster. On each destination node, a process called ebucketmigrator is started. It uses the DCP protocol to request that all the data be transferred for a single vBucket, and that the new vBucket data becomes the active vBucket once the migration has been completed.

While the vBucket migration process is taking place, clients are still sending data to the existing vBucket. This information is migrated along with the original data that existed before the migration was requested. Once the migration of all the data has completed, the original vBucket is marked as disabled, and the new vBucket is enabled. This updates the vBucket map and is communicated back to the connected clients that will now use the new location.

After a vBucket has been copied over to the new location the system can discard the old vBucket or keep it depending on replica count requirements.

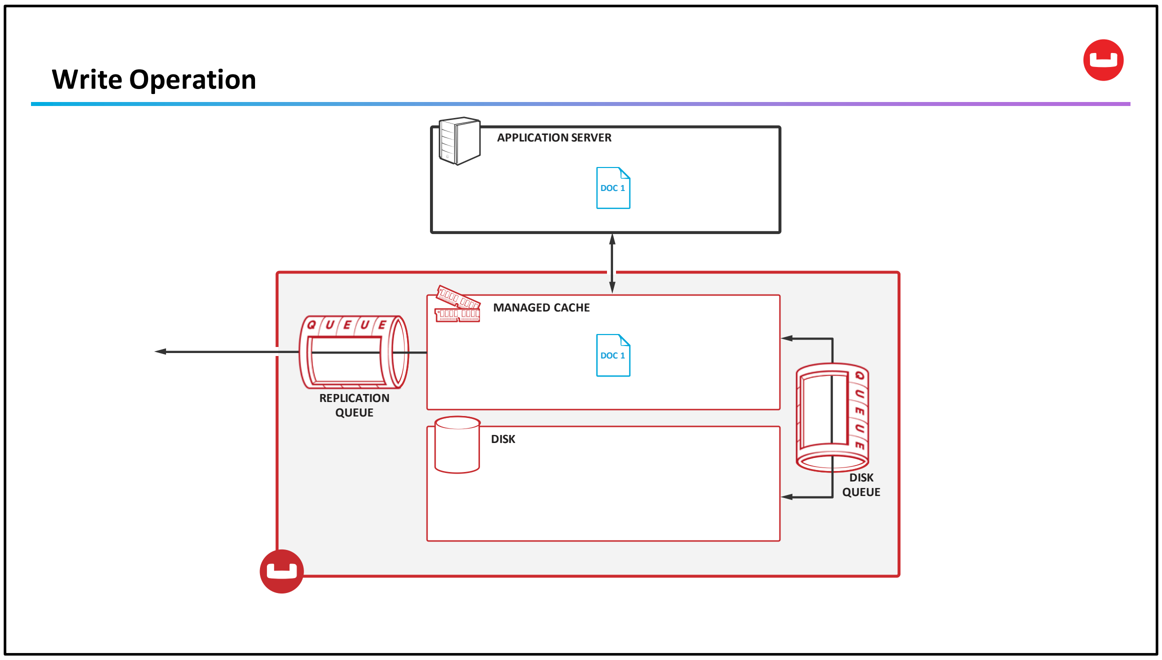
## Couchbase Operations - Fundamentals

Confidential and Proprietary. Do not distribute without Couchbase consent. © Couchbase 2017. All rights reserved.



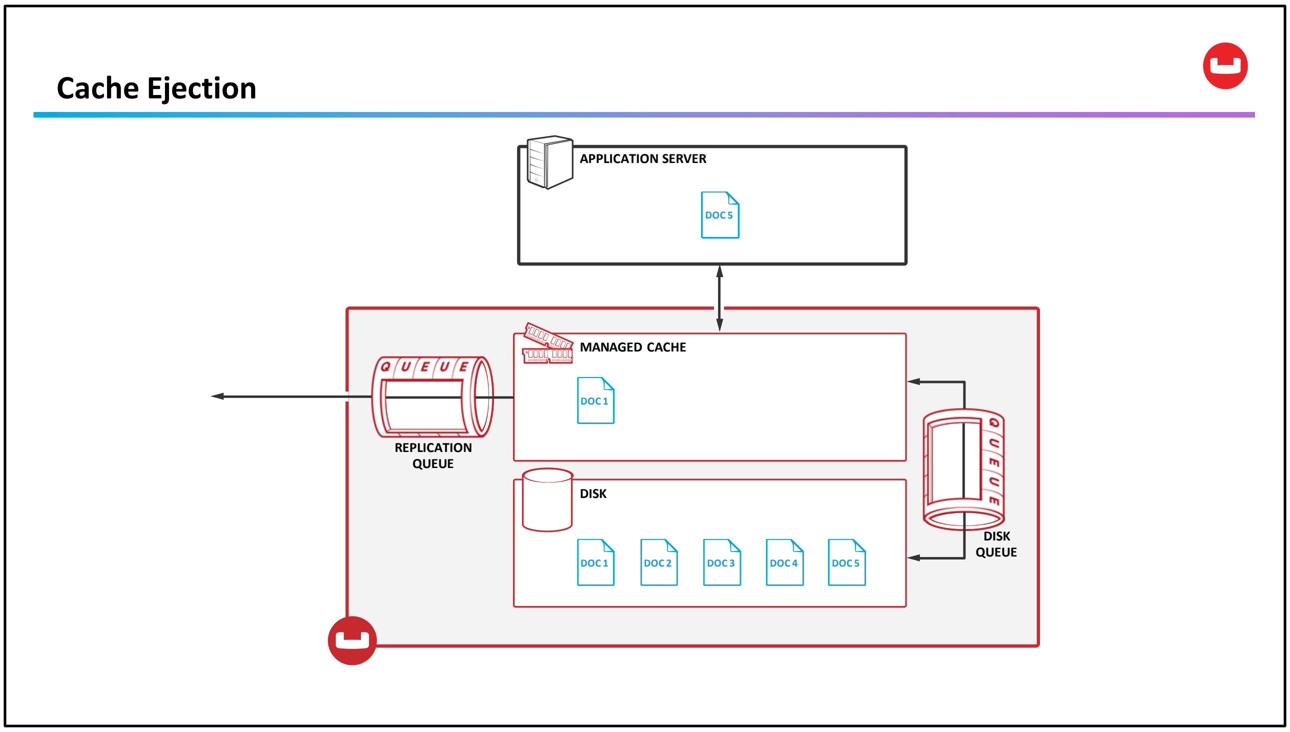
## Single-node type means easier administration and scaling

- Reads out of cache are extremely fast
- No other process/system to communicate with
- Data connection is a TCP-binary protocol



## Single-node type means easier administration and scaling

- Writes are async by default
- Application gets acknowledgement when successfully in RAM and can trade-off waiting for replication or persistence per-write
- Replication to 1, 2 or 3 other nodes
- Replication is RAM-based so extremely fast
- Off-node replication is primary level of HA
- Disk written to as fast as possible – no waiting



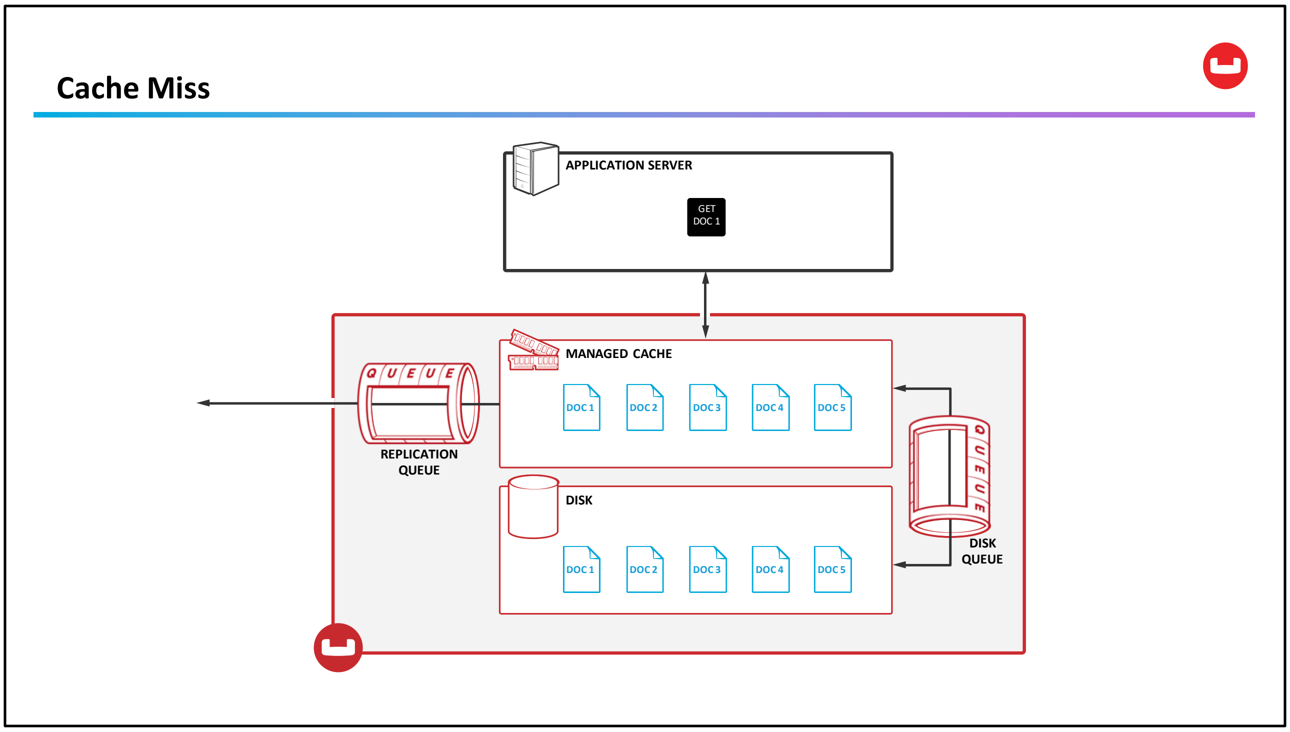
## Cache Ejection

Now, as you fill up memory (click), some data that has **already been written to disk** will be ejected from RAM to make room for new data. (click)

Couchbase supports holding much more data than you have RAM available. It's important to size the RAM capacity appropriately for your working set: the portion of data your application is working with at any given point in time and needs very low latency, high throughput access to. In some applications this is the entire data set, in others it is much smaller. As RAM fills up, we use a "not recently used" algorithm to determine the best data to be ejected from cache.

### Single-node type means easier administration and scaling

- Layer consolidation means read through and write through cache
- Couchbase automatically removes data that has already been persisted from RAM



Cache Miss

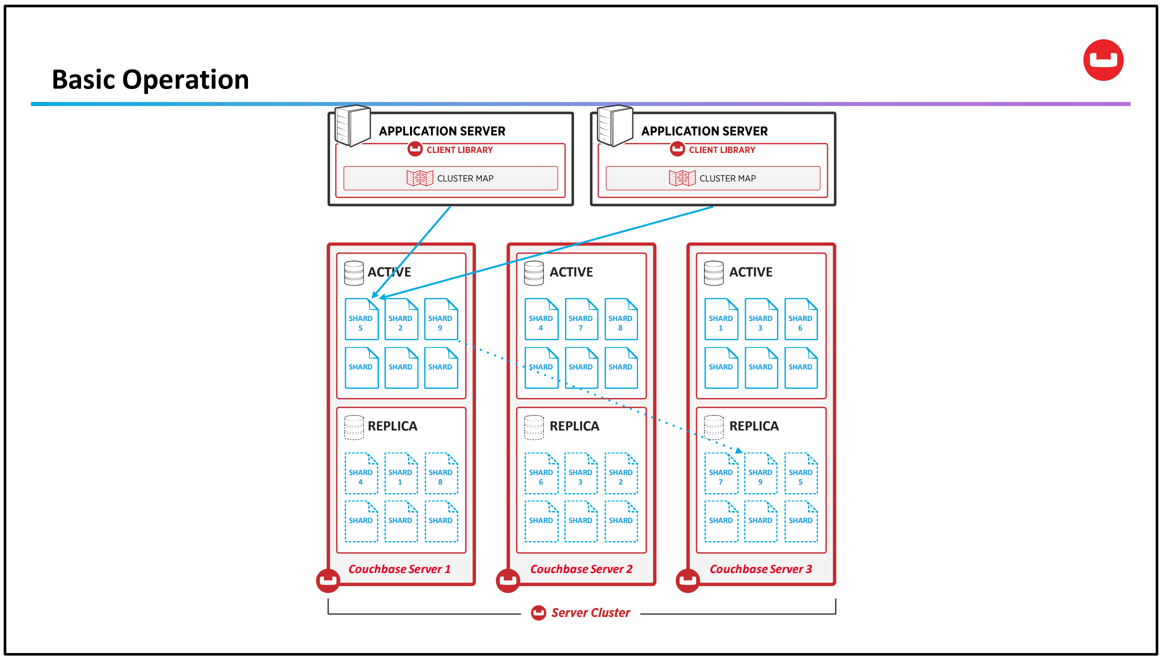
Should a read now come in for one of those documents that has been ejected (click), it is copied back from disk into RAM and sent back to the application. The document then remains in RAM as long as there is space and it is being accessed.

## Single-node type means easier administration and scaling

- Layer consolidation means 1 single interface for App to talk to and get its data back as fast as possible
- Separation of cache and disk allows for fastest access out of RAM while pulling data from disk in parallel

## Couchbase Cluster Operations & High Availability

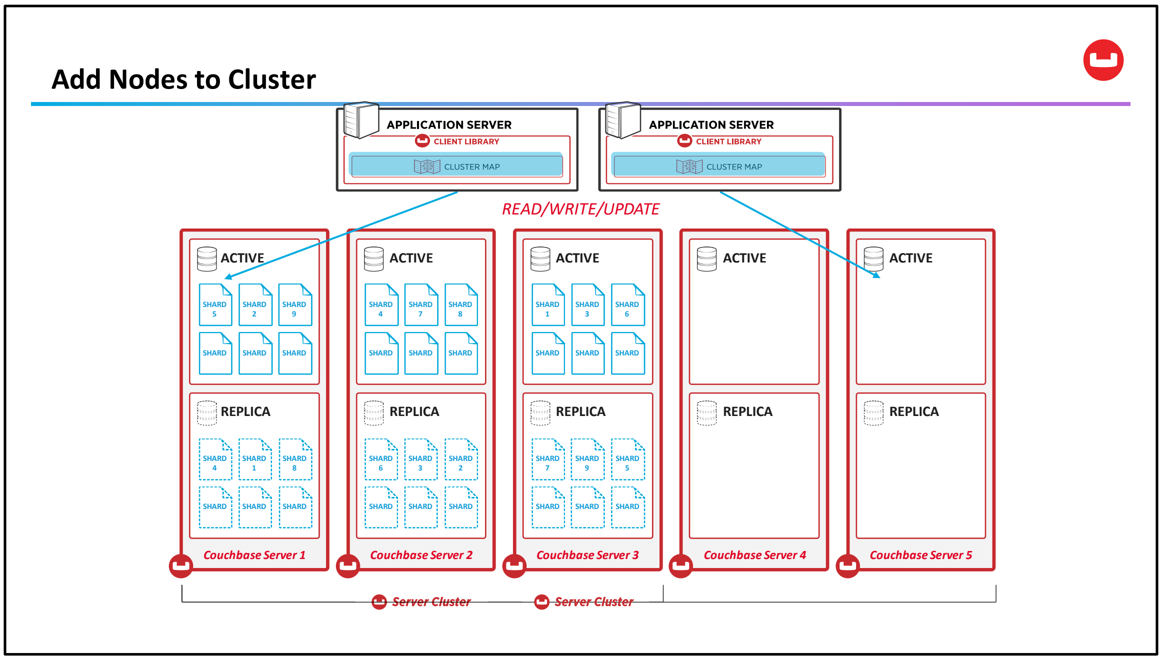
Couchbase



## Basic Operation

## Application has single logical connection to cluster (client object)

- Data is automatically sharded resulting in even document data distribution across cluster
- Each vBucket replicated 1, 2 or 3 times (“peer-to-peer” replication)
- Docs are automatically hashed by the client to a shard’
- Cluster map provides location of which server a shard(K,V) is on
- Every read/write/update/delete goes to same node for a given key
- Strongly consistent data access (“read your own writes”)
- A single Couchbase node can achieve 100k’s ops/sec so no need to scale reads

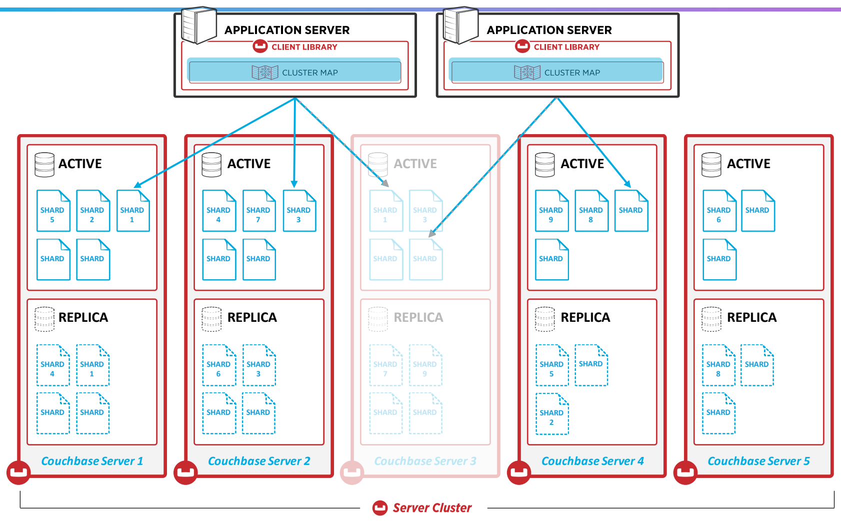


## Add Nodes to Cluster

Application has single logical connection to cluster (client object)

- Multiple nodes added or removed at once
- One-click operation
- Incremental movement of active and replica vbuckets and data
- Client library updated via cluster map
- Fully online operation, no downtime or loss of performance

## Fail Over Node



### Application has single logical connection to cluster (client object)

- When node goes down, some requests will fail
- Failover is either automatic or manual`
- Client library is automatically updated via cluster map
- Replicas not recreated to preserve stability
- Best practice to replace node and rebalance

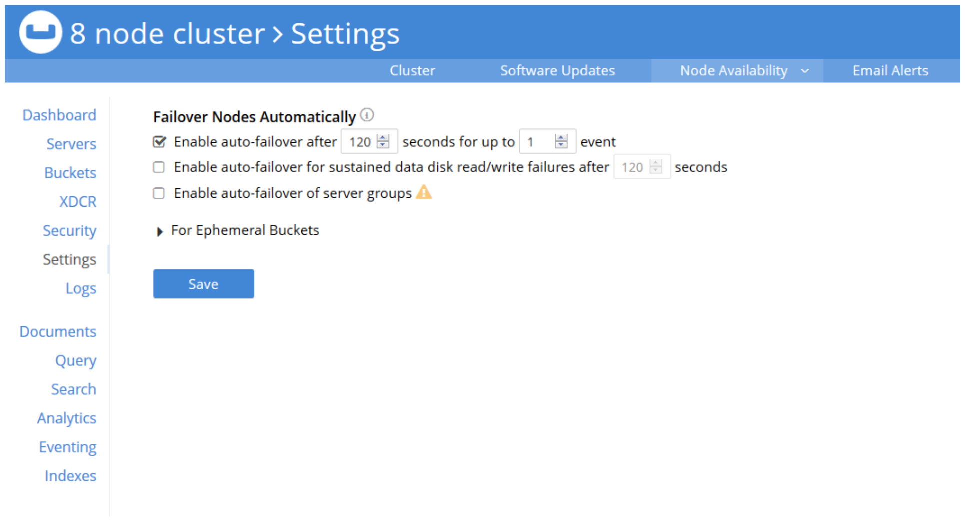
Automatic Failover — or auto-failover — can be configured to fail over a node or group automatically: no immediate administrator intervention is required. Specifically, the Cluster Manager autonomously detects and verifies that the node or group is unavailable, and then initiates the hard failover process. Auto-failover does not fix or identify problems that may have occurred. Once appropriate fixes have been applied to the cluster by the administrator, a rebalance is required. Automatic failover is always hard failover.

## Failover events

Auto-failover occurs in response to failover events. Events are of three kinds:

- Node failure.
- A server-node within the cluster is unavailable (due to a network failure, out-of-memory problem, or other node-specific issue).
- Disk read/write failure.
- Attempts to read from or write to disk on a particular node have resulted in a significant rate of failure, for longer than a specified time-period. The node is removed by auto-failover, even though the node continues to be contactable.
- Group failure.
- An administrator-defined group of server-nodes within the cluster is unavailable (perhaps due to a network or power failure that has affected an individual, physical rack of machines, or a specific subnet).

## Failover(cont)



Failover Nodes Automatically

[✓] Enable auto-failover after 120 seconds for up to 1 event

[ ] Enable auto-failover for sustained data disk read/write failures after 120 seconds

[ ] Enable auto-failover of server groups

For Ephemeral Buckets

Enable auto-failover of server groups

DO NOT enable this unless you have set up homogenous server groups for failure scenarios (like racks/zones), have 3 or more server groups, and have the capacity to absorb the load of the failed-over group.

Auto-failover is configured by means of parameters that include the following.

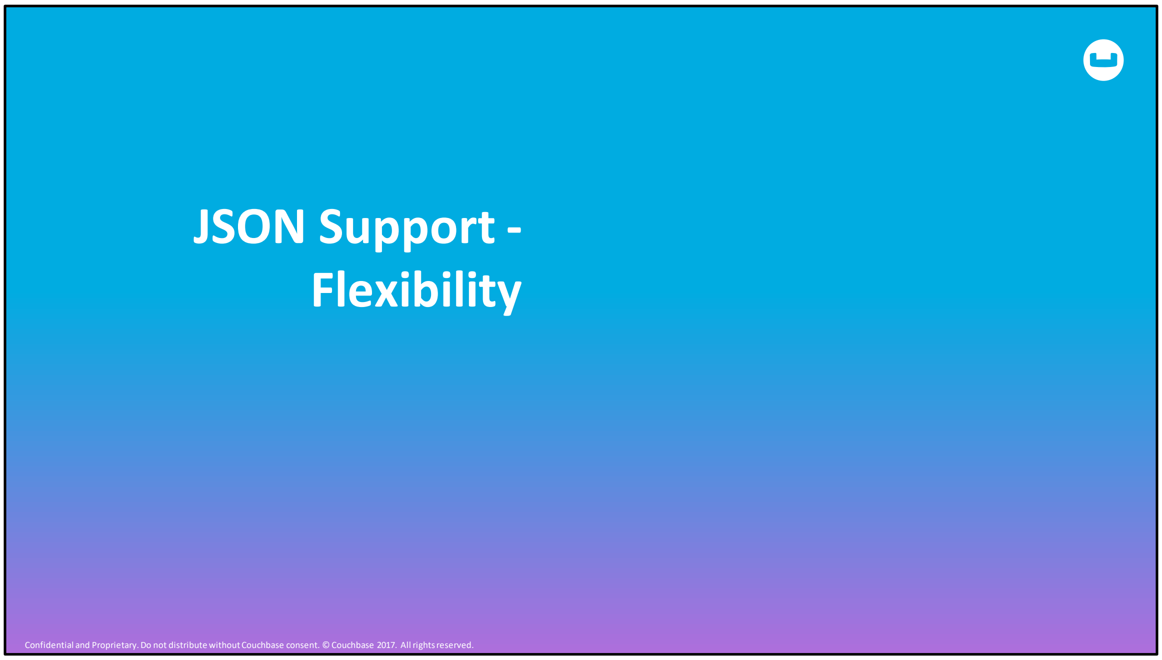
Timeout. The number of seconds that must elapse, after a node or group has become unavailable, before auto-failover is triggered. The default is 120.

Maximum count. The maximum number of failover events that can occur sequentially and be handled by auto-failover. The maximum-allowed value is 3, the default is 1. This parameter is available in Enterprise Edition only: in Community Edition, the maximum number of failover events that can occur sequentially and be handled by auto-failover is always 1.

Count. The number of failover events that have occurred. The default value is 0. The value is incremented by 1 for every automatic-failover event that occurs, up to the defined maximum count: beyond this point, no further automatic failover can be triggered until the count is reset to 0 through administrator-intervention.

Enablement of disk-related automatic failover; with corresponding time-period. Whether automatic failover is enabled to handle continuous read-write failures. If it is enabled, a number of seconds can also be specified: this is the length of a constantly recurring time-period against which failure-continuity on a particular node is evaluated. If at least 60% of the most recently elapsed instance of the time-period has consisted of continuous failure, failover is automatically triggered. The default value for the enablement of disk-related automatic failover is false. The default value for the corresponding time-period is 120. This parameter is available in Enterprise Edition only.

Group failover enablement. Whether or not groups should be failed over. A group failover is considered to be a single event, even if many nodes are included in the group. The default value is false. This parameter is available in Enterprise Edition only.

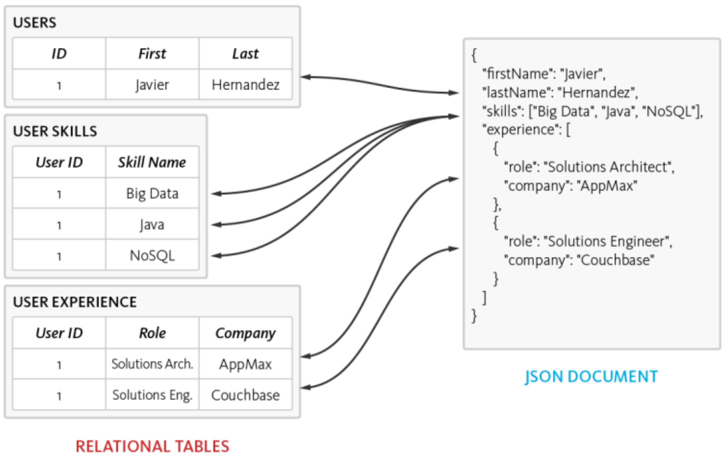


## JSON Support - Flexibility

Confidential and Proprietary. Do not distribute without Couchbase consent. © Couchbase 2017. All rights reserved.

## The Power of the Flexible JSON Schema

- **Ability to store data in multiple ways**
  - Denormalized single document, as opposed to normalizing data across multiple table
  - Dynamic Schema to add new values when needed



USERS

| ID | First | Last |
|---|---|---|
| 1 | Javier | Hernandez |

USER SKILLS

| User ID | Skill Name |
|---|---|
| 1 | Big Data |
| 1 | Java |
| 1 | NoSQL |

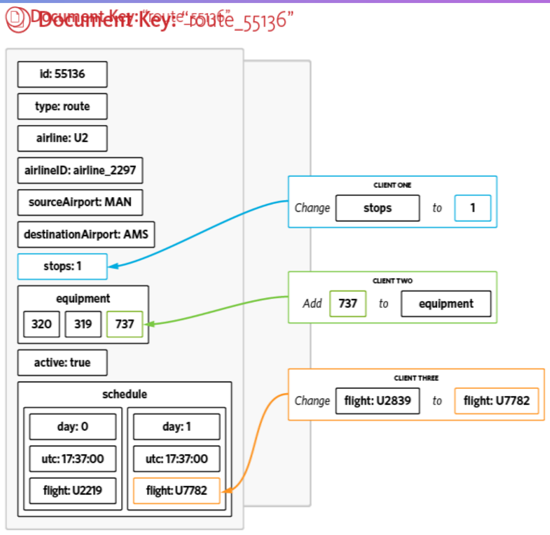
USER EXPERIENCE

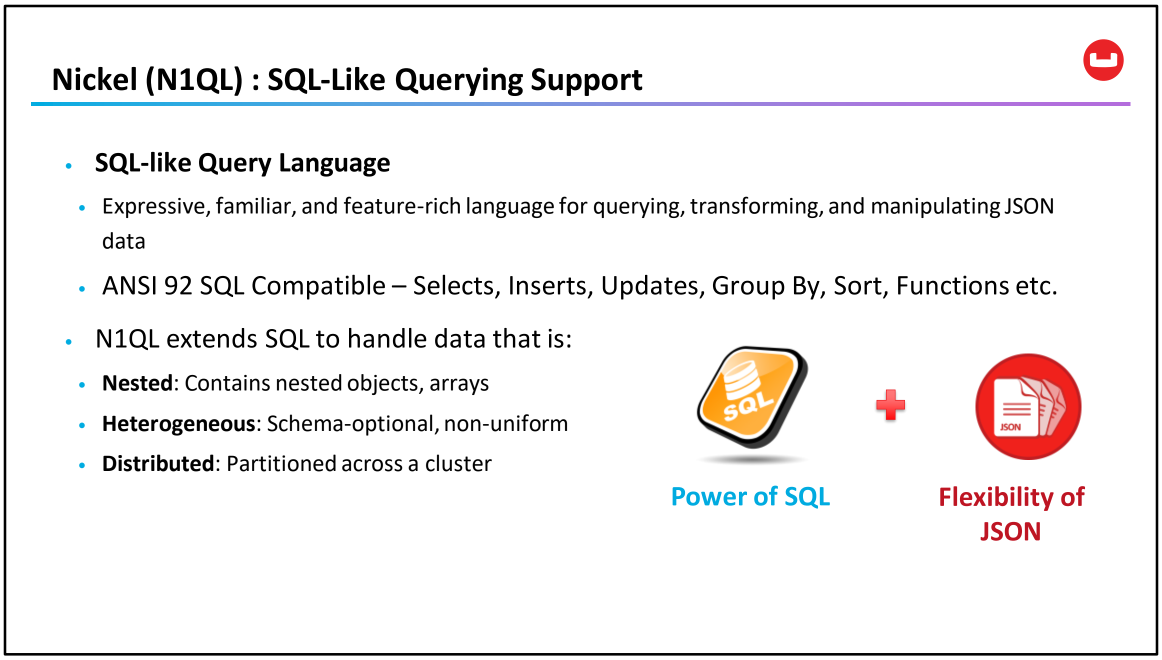
| User ID | Role | Company |
|---|---|---|
| 1 | Solutions Arch. | AppMax |
| 1 | Solutions Eng. | Couchbase |

```
{
  "firstName": "Javier",
  "lastName": "Hernandez",
  "skills": ["Big Data", "Java", "NoSQL"],
  "experience": [
    {
      "role": "Solutions Architect",
      "company": "AppMax"
    },
    {
      "role": "Solutions Engineer",
      "company": "Couchbase"
    }
  ]
}
```

## Efficient Sub-Document Operations

- **Document Mutations:**
- Atomic Operate on individual fields
- Identical syntax behavior to regular bucket methods (upsert, insert, get, replace)
- Support for JSON fragments.
- Support for Arrays with uniqueness guarantees and ordinal placement (front/back)



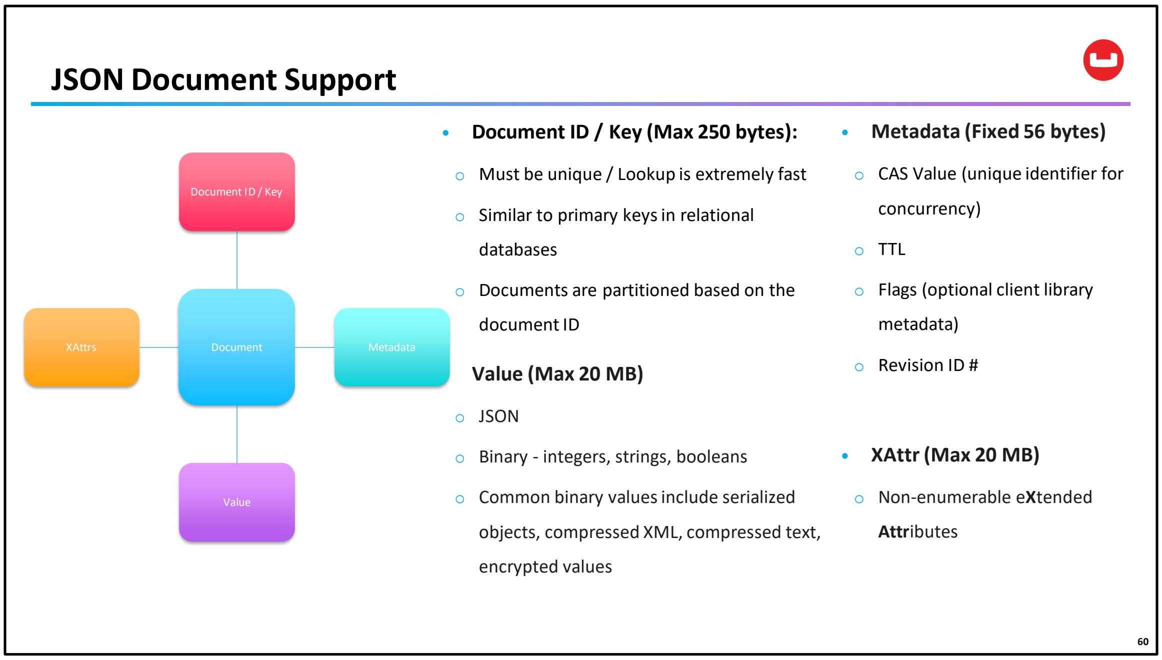


## Nickel (N1QL) : SQL-Like Querying Support

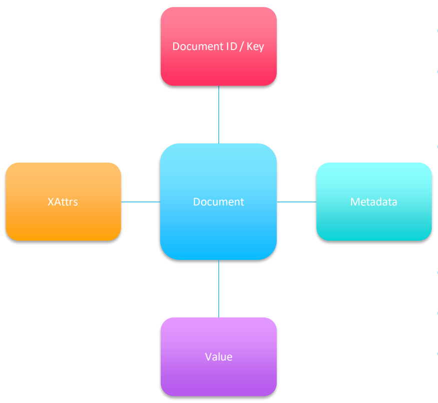
- **SQL-like Query Language**
  - Expressive, familiar, and feature-rich language for querying, transforming, and manipulating JSON data
  - ANSI 92 SQL Compatible – Selects, Inserts, Updates, Group By, Sort, Functions etc.
- N1QL extends SQL to handle data that is:
  - **Nested**: Contains nested objects, arrays
  - **Heterogeneous**: Schema-optional, non-uniform
  - **Distributed**: Partitioned across a cluster

Power of SQL

Flexibility of JSON



## JSON Document Support



- **Document ID / Key (Max 250 bytes):**
  - Must be unique / Lookup is extremely fast
  - Similar to primary keys in relational databases
  - Documents are partitioned based on the document ID

**Value (Max 20 MB)**
  - JSON
  - Binary - integers, strings, booleans
  - Common binary values include serialized objects, compressed XML, compressed text, encrypted values

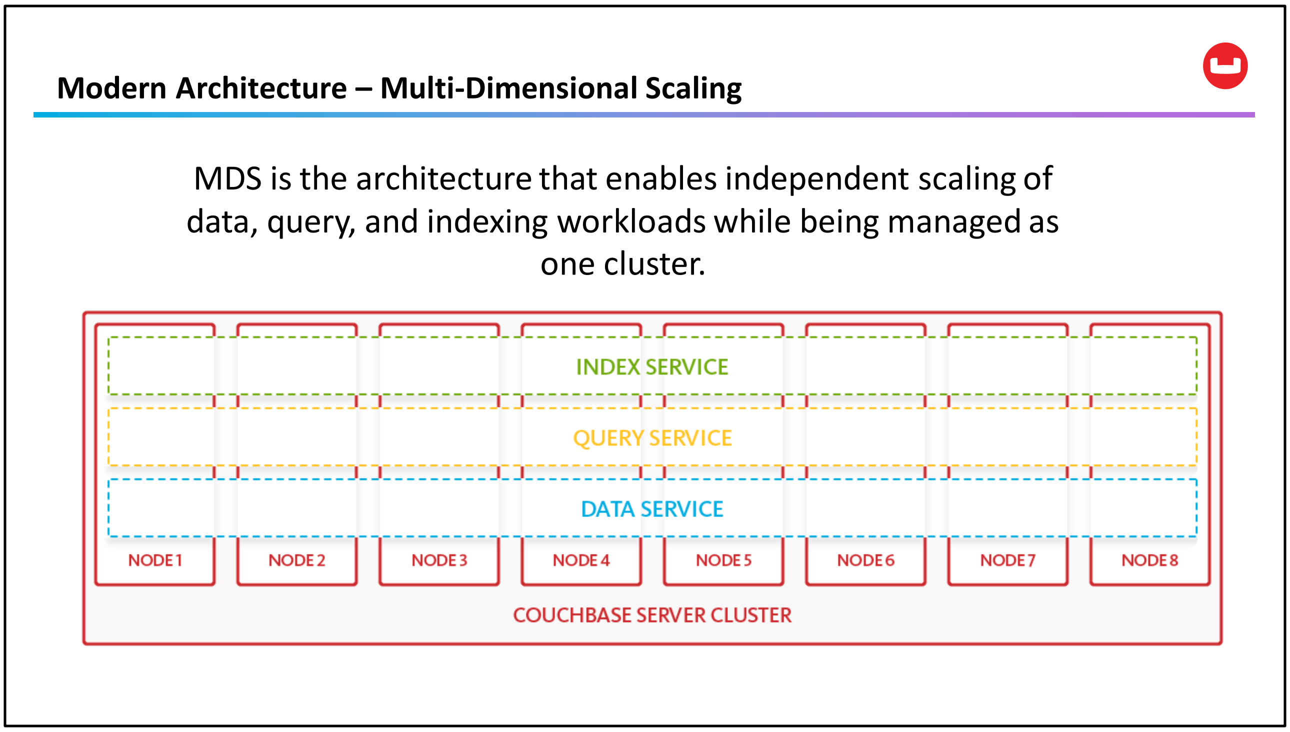
- **Metadata (Fixed 56 bytes)**
  - CAS Value (unique identifier for concurrency)
  - TTL
  - Flags (optional client library metadata)
  - Revision ID #

- **XAttr (Max 20 MB)**
  - Non-enumerable e**X**tended **Attr**ibutes

60

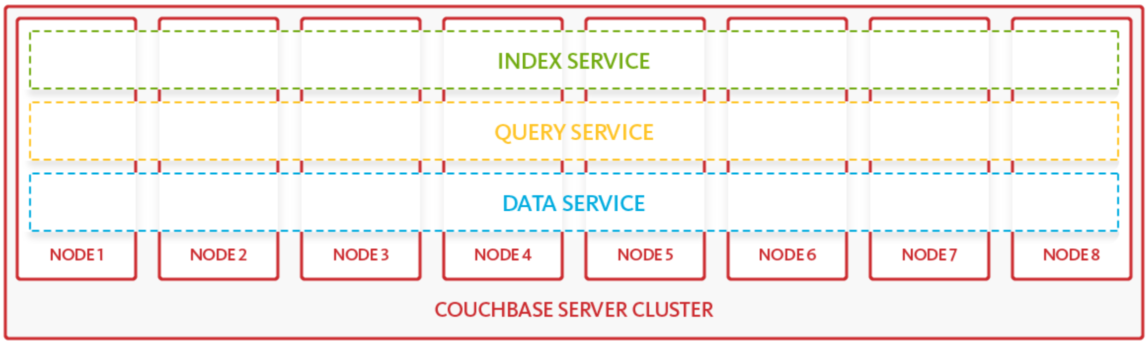
## Couchbase Multi-Dimensional Scaling (MDS)

Confidential and Proprietary. Do not distribute without Couchbase consent. © Couchbase 2017. All rights reserved.

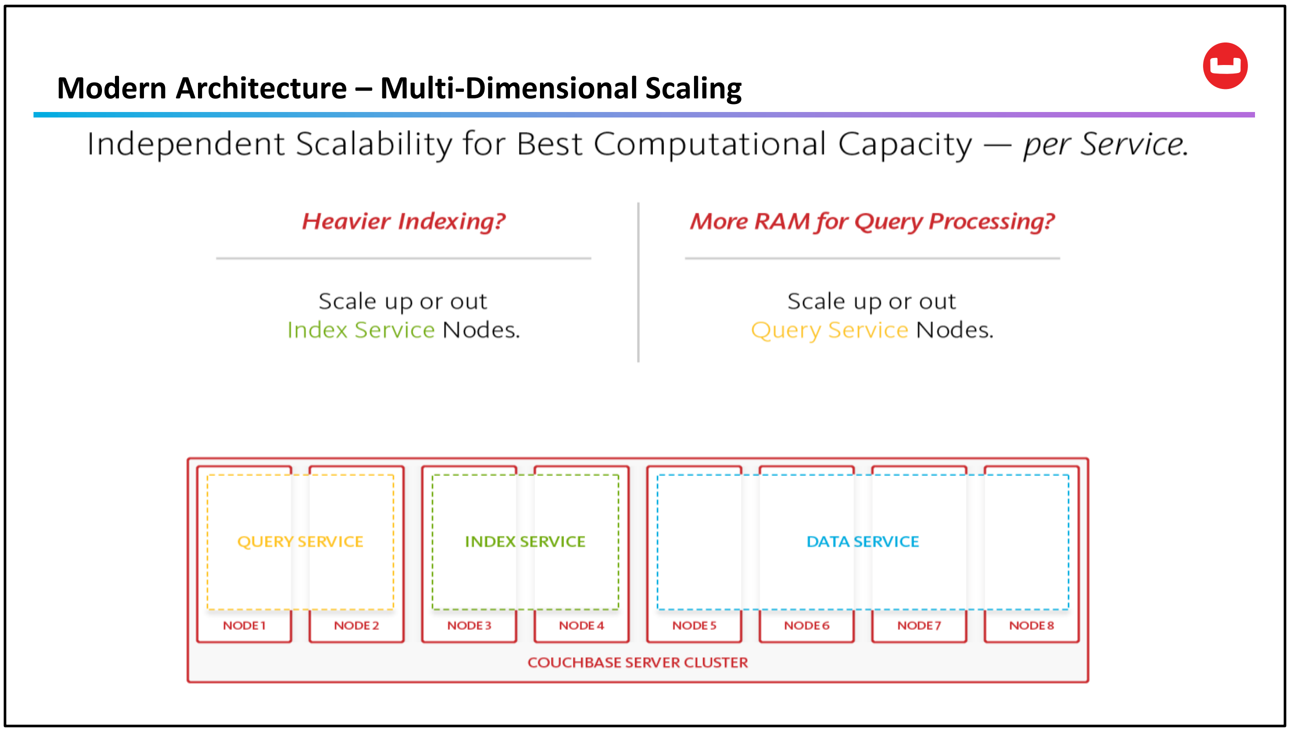


## Modern Architecture – Multi-Dimensional Scaling

MDS is the architecture that enables independent scaling of data, query, and indexing workloads while being managed as one cluster.



Homogenous approach to scaling requires all nodes to have the same physical characteristics.



## Modern Architecture – Multi-Dimensional Scaling

Independent Scalability for Best Computational Capacity — *per Service.*

***Heavier Indexing?***

Scale up or out
Index Service Nodes.

***More RAM for Query Processing?***

Scale up or out
Query Service Nodes.

QUERY SERVICE
INDEX SERVICE
DATA SERVICE
NODE 1
NODE 2
NODE 3
NODE 4
NODE 5
NODE 6
NODE 7
NODE 8
COUCHBASE SERVER CLUSTER

Key Points: MDS provides the ability for cluster administrator to separate the various workloads and tailor the cluster architecture for the application's scalability and performance needs. This while providing the functionality needed by by the application.

- Pictured is a possible architecture where all the cluster has been split with the Query and Index Service on different nodes of the cluster.
- This allows you to not only isolate the workload, but you could have the nodes with the other services with different server configs.
    - For example, the query service needs CPU and RAM and has no storage, but the Index service needs Ram and fast disks. So you could configure these nodes different from the Data Service nodes to tailor performance even further.
- This is what allows Couchbase to scale with querying.
- The indexing and query services are eventually consistent by default, but in code can be made to fully

## Modern Architecture – Multi-Dimensional Scaling

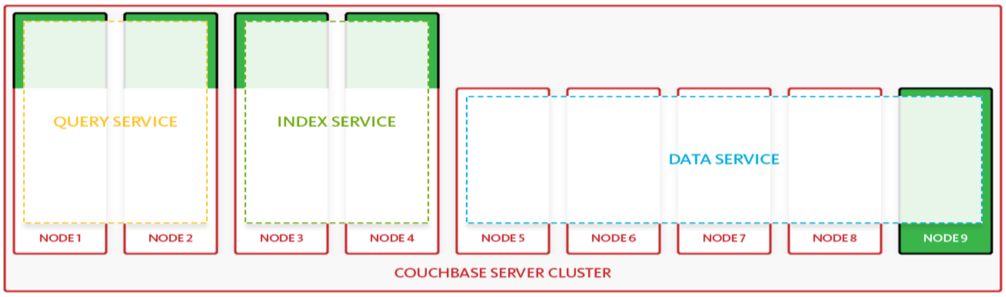
Independent Scalability for Best Computational Capacity — *per Service.*

***Heavier Indexing?***

Scale up or out
Index Service Nodes.

***More RAM for Query Processing?***

Scale up or out
Query Service Nodes.



The same applies for other services

Key Points: MDS provides the ability for cluster administrator to separate the various workloads and tailor the cluster architecture for the application's scalability and performance needs. This while providing the functionality needed by by the application.

- Pictured is a possible architecture where all the cluster has been split with the Query and Index Service on different nodes of the cluster.
- This allows you to not only isolate the workload, but you could have the nodes with the other services with different server configs.
    - For example, the query service needs CPU and RAM and has no storage, but the Index service needs Ram and fast disks. So you could configure these nodes different from the Data Service nodes to tailor performance even further.
- This is what allows Couchbase to scale with querying.
- The indexing and query services are eventually consistent by default, but in code can be made to fully

## Couchbase

Confidential and Proprietary. Do not distribute without Couchbase consent. © Couchbase 2017. All rights reserved.