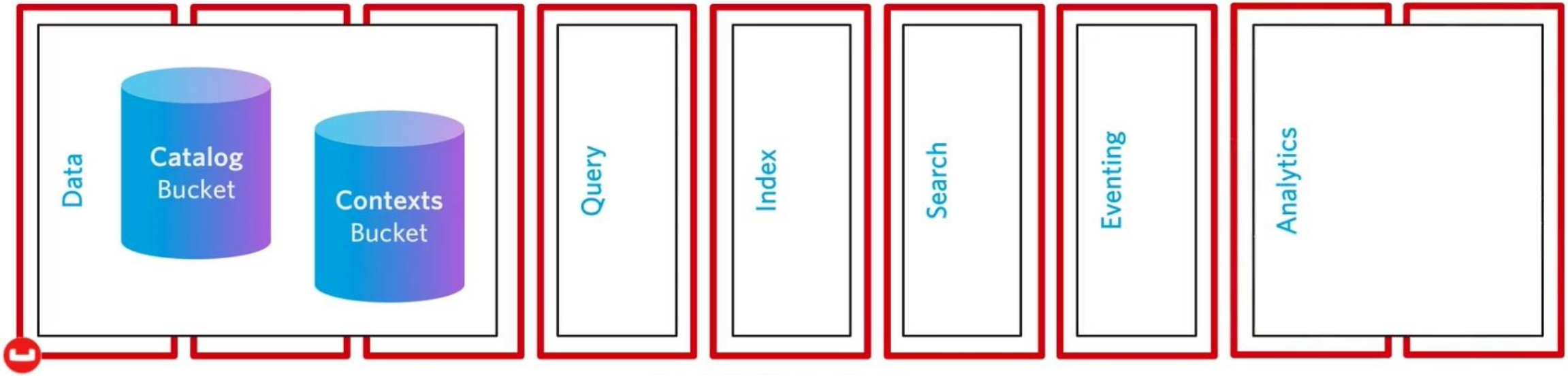
Buckets in Couchbase

ANJU MUNOTH

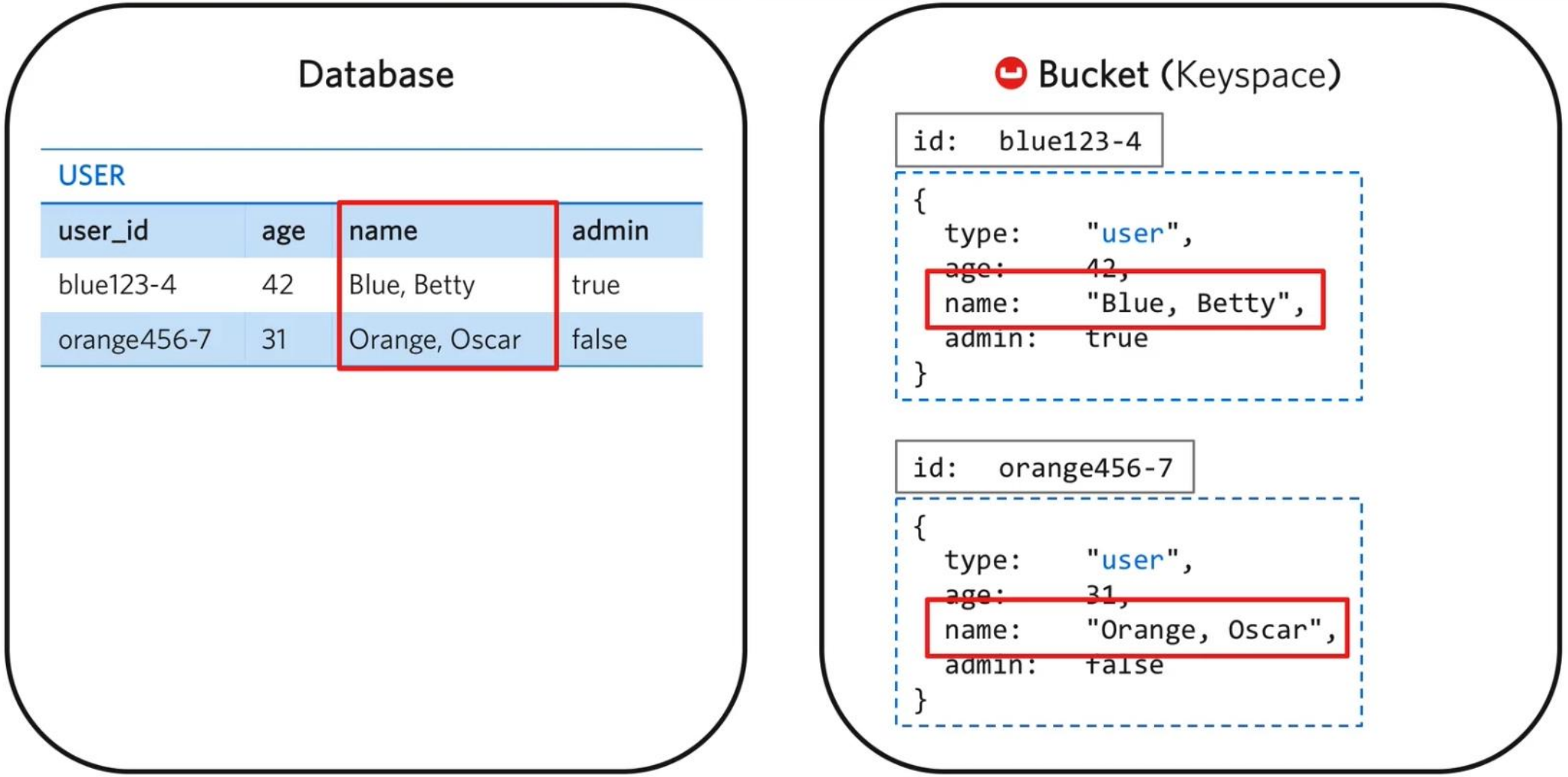
What is the role of a Data Bucket?

A bucket is a set of uniquely keyed docs ("keyspace")

- ✓ Buckets are equally distributed across Data nodes
- ✓ Source for Query, Index, and Eventing Services
- ✓ Source for Search indexes
- ✓ Source for Analytics shadow datasets



How do buckets and databases compare?



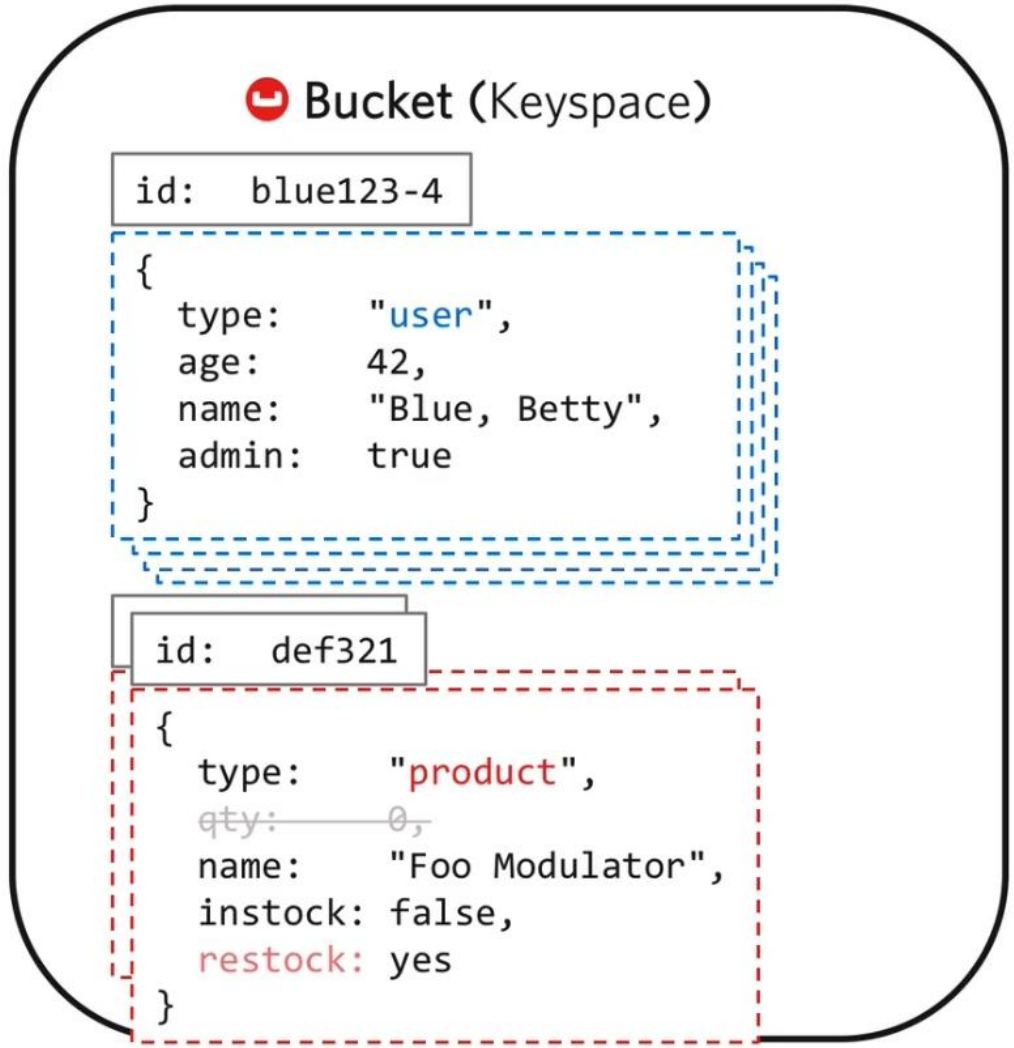
How do buckets and databases compare?

Schema not enforced by Couchbase

Up to 10 buckets per cluster

... so, why more than one?

- ✓ Varying **caching** needs
- ✓ Varying **replication** needs
- ✓ Varying **indexing** needs
- ✓ Varying **security** needs
- ✓ **Multi-tenancy**



What do Data Buckets control?

Caching and Persistence

Data is cached in RAM for immediate availability & asynchronously persisted to disk

Each bucket shares a thread pool to handle its queued persistence

If incoming docs exceed RAM, may fully eject old docs, or retain key/meta to speed future access

Uncached data is retrieved from disk, delivered, and cached

Replication

Each bucket shares a thread pool to handle its queued replication

To secure against node failure, up to 3 replicas can be made of each document

Replicas never reside on the same node as their active documents

Rack awareness enables group-level control of replica placement

Rebalancing

Fast, efficient background document redistribution when nodes are added, removed, or failed over

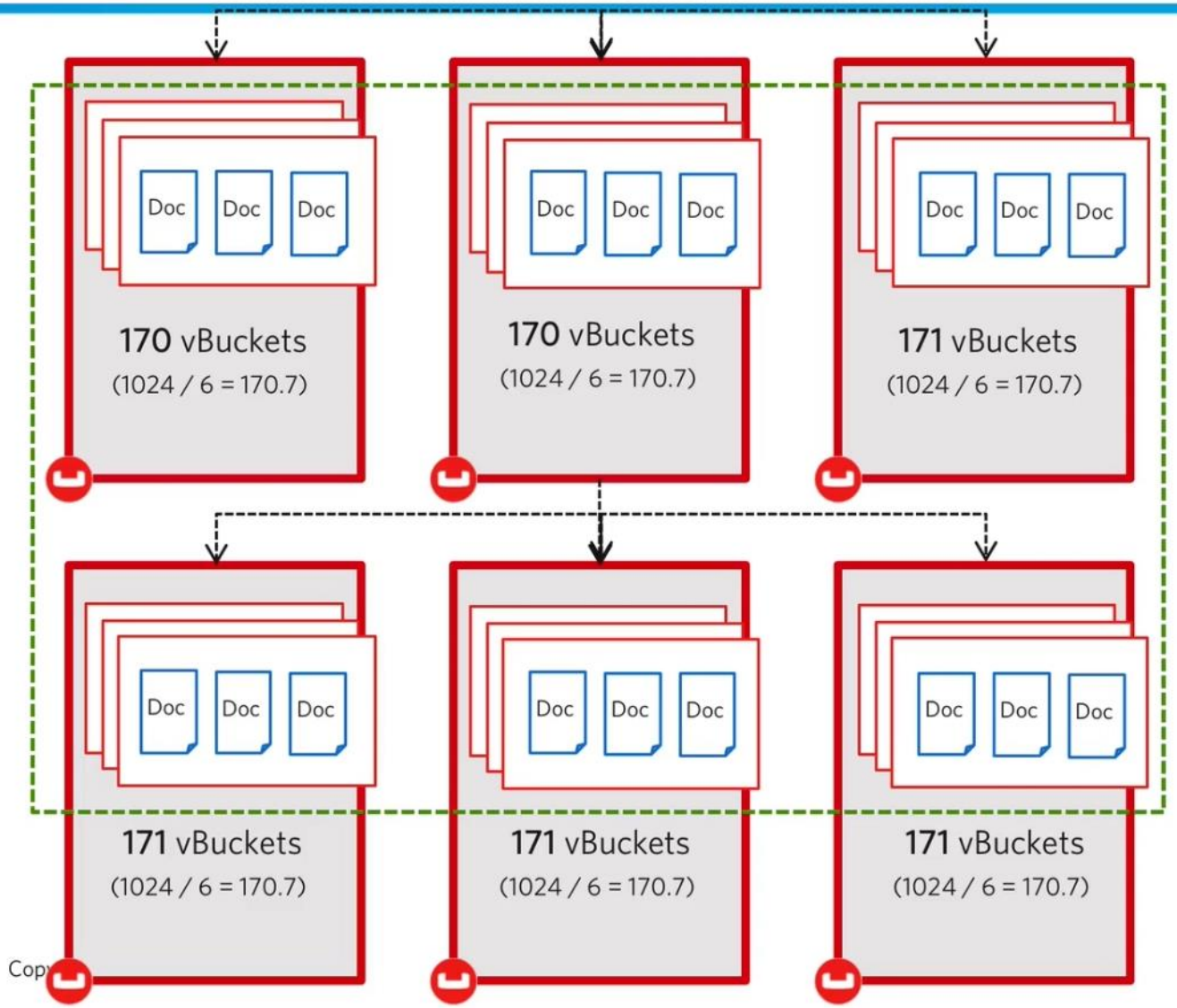
How are buckets virtualized with vBuckets?

vBucket: a data bucket segment physically residing on a node

Each bucket is divided into 1024 *vBuckets*, evenly distributed across all nodes in the cluster

As nodes join/leave cluster, *vBuckets* adjust automatically

Location of *vBuckets* is tracked by the Couchbase SDK *cluster map*, regularly updated to client



Buckets

- Couchbase Server uses buckets to group collections of keys and values logically
- A maximum of 30 buckets can be created in a cluster. Each bucket must be specified as one of the following three types.
  - Ephemeral buckets
  - Couchbase buckets
  - Memcached buckets
- For all bucket-types, items are selected for ejection by means of the Not Recently Used (NRU) algorithm.
- All bucket types are fully compatible with the Memcached open source distributed key-value cache.

What are the three bucket types?

- ✓ **Couchbase** – queryable, replicated, persistent k-v/document storage
- ✓ **Ephemeral** – queryable, replicated, k-v/document storage, not persisted
- ✓ **Memcached** – binary cache

	Couchbase	Ephemeral	Memcached
Persistence	✓	✗	✗
Replication	✓	✓	✗
Rebalance	✓	✓	✗
XDCR	✓	✓	✗
N1QL	✓	✓	✗
Indexing	✓	✓ (MOI, FTS)	✗
Max Object Size	20mb	20mb	1mb

Couchbase buckets

- Store data persistently, as well as in memory.
- Allow data to be automatically replicated for high availability, using the Database Change Protocol (DCP); and dynamically scaled across multiple clusters, by means of Cross Datacenter Replication (XDCR).
- If a Couchbase bucket's RAM-quota is exceeded, items are ejected.
- This means that data, which is resident both in memory and on disk, is removed from memory, but not from disk.
- Therefore, if removed data is subsequently needed, it is reloaded into memory from disk.
- For a Couchbase bucket, ejection can be either of the following, based on configuration performed at the time of bucket-creation:
  - Value-only: Only key-values are removed. Generally, this favors performance at the expense of memory.
  - Full: All data — including keys, key-values, and metadata — is removed. Generally, this favors memory at the expense of performance.

Ephemeral buckets

- Used whenever persistence is not required
- For example, when repeated disk-access involves too much overhead.
- Allows highly consistent in-memory performance, without disk-based fluctuations.
- Also allows faster node rebalances and restarts.
- If an Ephemeral bucket's RAM-quota is exceeded, one of the following occurs, based on configuration performed at the time of bucket-creation:
  - Resident data-items remain in RAM. No additional data can be added; and attempts to add data therefore fail.
  - Resident data-items are ejected from RAM, to make way for new data.
    - For an Ephemeral bucket, this means that data, which is resident in memory (but, due to this type of bucket, can never be on disk), is removed from memory. Therefore, if removed data is subsequently needed, it cannot be re-acquired from Couchbase Server.
- For an Ephemeral bucket, ejection removes all of an item's data: however, a tombstone (a record of the ejected item, which includes keys and metadata) is retained until the next scheduled purge of metadata for the current node

Memcached buckets

- Designed to be used alongside other database platforms, such as ones employing relational database technology.
- By caching frequently-used data, Memcached buckets reduce the number of queries a database-server must perform.
- Each Memcached bucket provides a directly addressable, distributed, in-memory key-value cache.
- Memcached buckets are not persistent on disk: they only exist in RAM.
- If a Memcached bucket's RAM-quota is exceeded, items are ejected.
- For a Memcached bucket, this means that data, which is resident in memory (but, due to this type of bucket, can never be on disk), is removed from memory.
- Therefore, if removed data is subsequently needed, it cannot be re-acquired from Couchbase Server.
- Ejection removes all of an item's data.

Bucket Capabilities

Capability	Memcached buckets	Couchbase buckets	Ephemeral buckets
Item size limit	1 MB	20 MB	20 MB
Persistence	No	Yes	No
Replication (DCP)	No	Yes	Yes
Cross Datacenter Replication (XDCR)	No	Yes	Yes
Rebalance	No	Yes	Yes
Encrypted data access	Yes	Yes	Yes
Durability	No	Yes	Yes

Bucket Capabilities

Capability	Memcached buckets	Couchbase buckets	Ephemeral buckets
TTL	Yes	Yes	Yes
Bucket TTL	No	Yes	Yes
Data Compression	No	Yes	Yes
Statistics	All except disk-related	All	All except disk-related
Client support	Ketama consistent hashing	Full smart client support	Full smart client support
Backup	No	Yes	Yes
Standard Index Storage	No	Yes	No

Bucket Capabilities

Capability	Memcached buckets	Couchbase buckets	Ephemeral buckets
Memory Optimized Index Storage	No	Yes	Yes
Query	No	Yes	Yes
Search	No	Yes	Yes
Analytics	No	Yes	Yes
Eventing	No	Yes	Yes
External connectors	No	Yes	Yes
Map-reduce views	No	Yes	No

Couchbase vs Ephemeral Buckets

Capability	Couchbase Buckets	Ephemeral Buckets
Persistence	Couchbase buckets are persisted asynchronously, from memory to disk. This provides protection from server-restarts. You can set persistence-properties at the bucket level.	Ephemeral buckets are not persisted to disk: they are retained in RAM only.
Replication (DCP and XDCR)	Couchbase buckets can be replicated across a configurable number of servers. If the host machine fails, a replica server is promoted to be the host server, providing high availability cluster operations via failover. You can configure replication at the bucket level. Additionally, *Cross Datacenter Replication* (XDCR) allows replication of Couchbase bucket-data between clusters.	Ephemeral buckets can be replicated across a configurable number of servers, exactly as can Couchbase buckets; but without being persisted to disk. Likewise, XDCR allows replication of Ephemeral bucket-data between clusters and without persistence: note however, that if an Ephemeral bucket configured to eject data when its RAM-quota is exceeded is used as a source for XDCR, not all data written to the bucket is guaranteed to be replicated by XDCR.

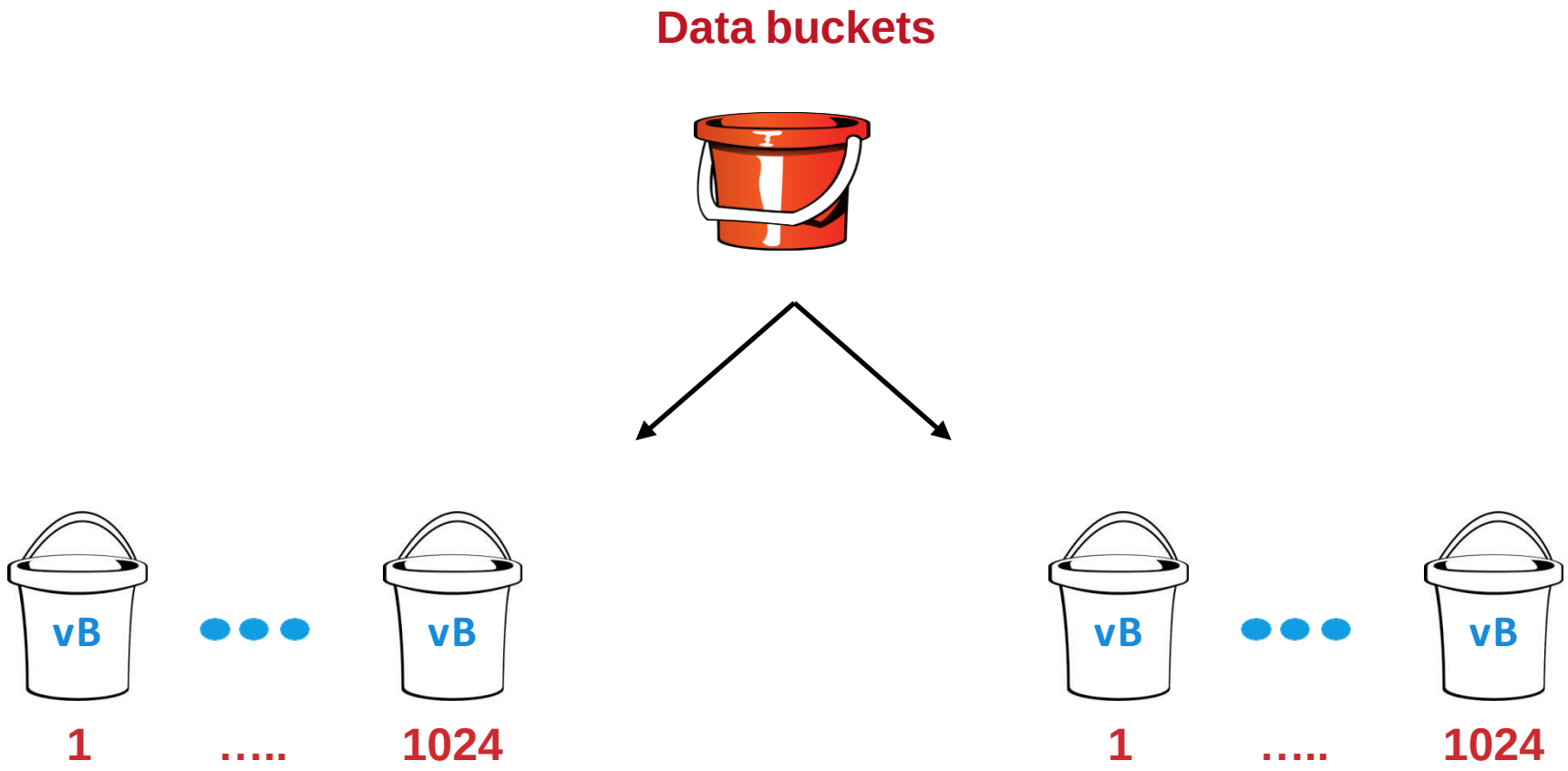
Couchbase vs Ephemeral Buckets

Capability	Couchbase Buckets	Ephemeral Buckets
Rebalance	By means of rebalancing, the load constituted by Couchbase buckets is distributed evenly across nodes within the cluster. Buckets and nodes can be dynamically added and removed.	Rebalancing redistributes Ephemeral buckets, exactly as it does Couchbase buckets; but without the data being persisted to disk.
Auto-failover and auto-reprovision	By default, Auto-failover starts when a node has been inaccessible for 120 seconds. Auto-failover can happen only up to a specified maximum number of times, prior to manual reset. When a failed node becomes accessible again, delta-node recovery is used: re-using data on disk, and resynchronizing it.	Auto-reprovision starts as soon as a node is inaccessible. Auto-reprovision can happen multiple times, for multiple nodes. When a failed node becomes accessible again, no delta-node recovery is required, since no data resides on disk.

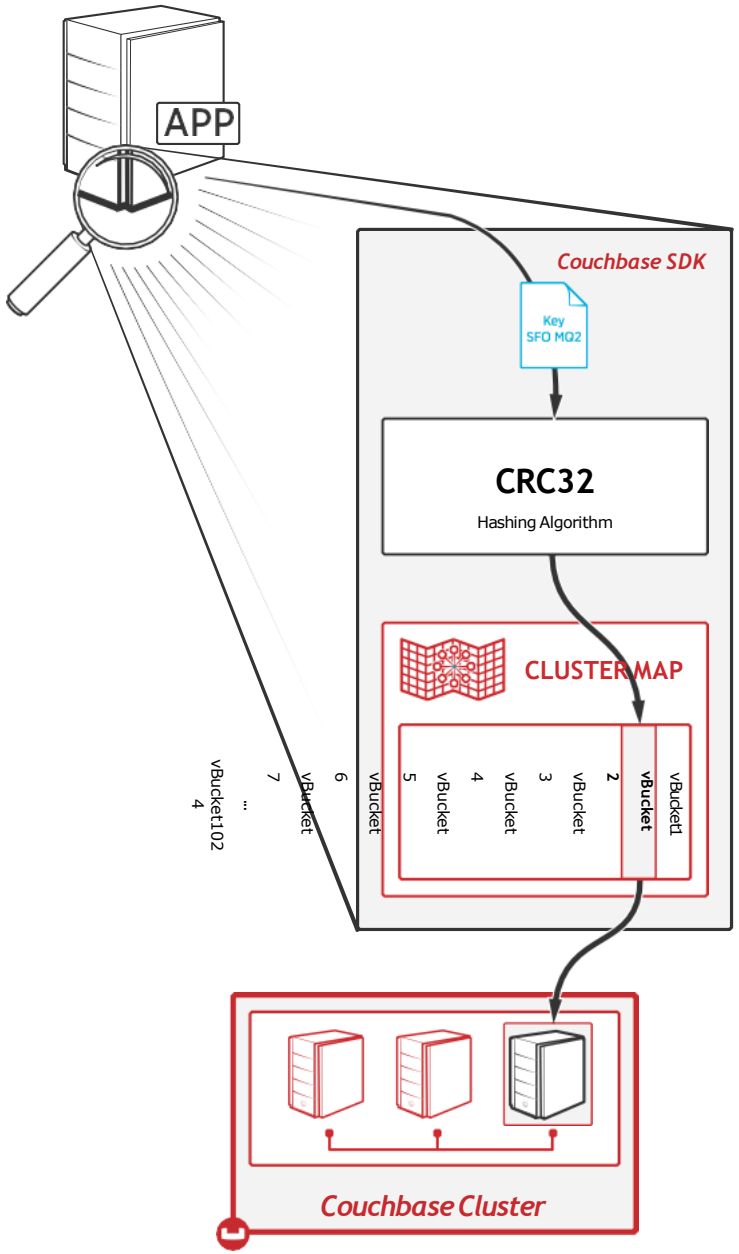
Understanding vBuckets

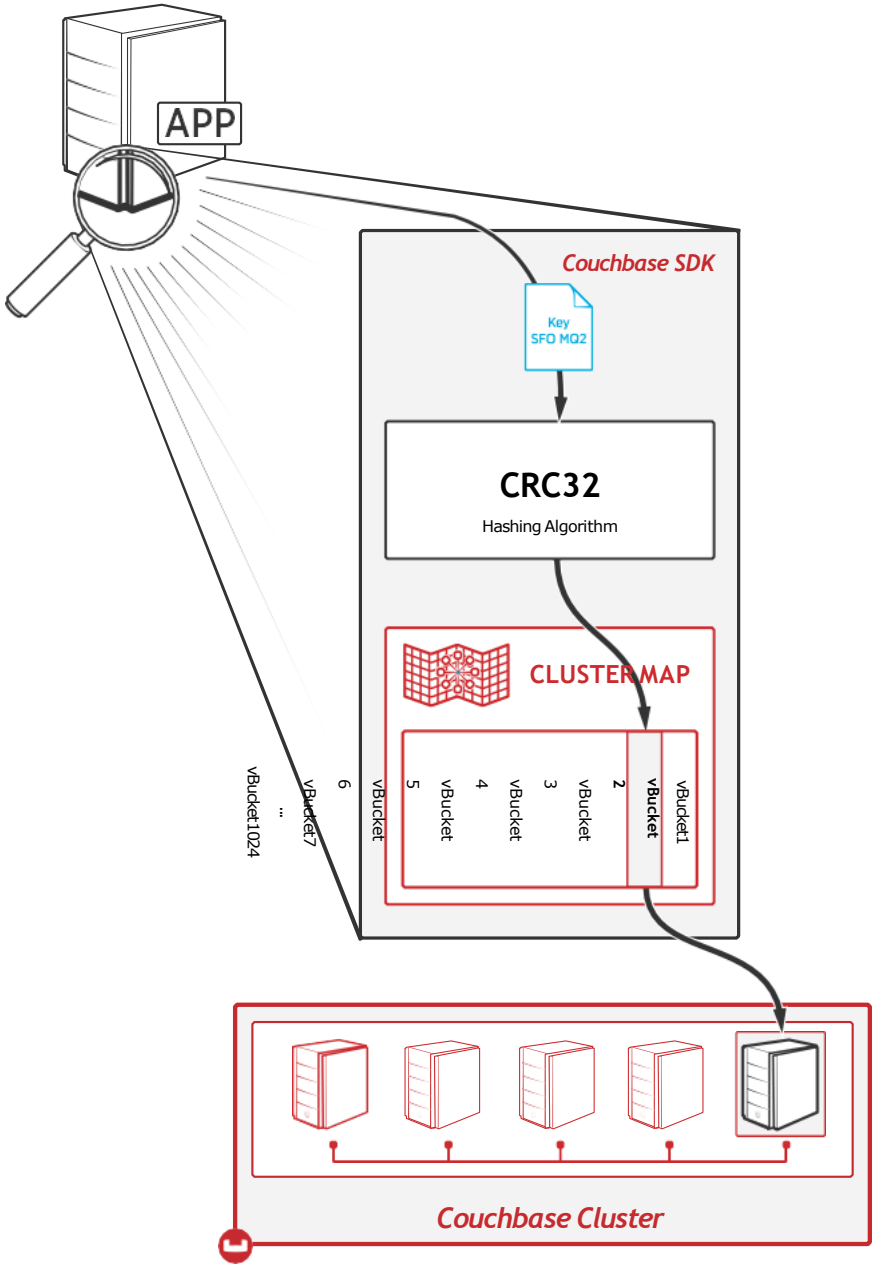
- Couchbase Server allows users and applications to save data, in binary or JSON format, in named buckets.
- Each bucket therefore contains keys and associated values.
- Within the memory and storage management system of Couchbase Server, both Couchbase and Ephermal buckets are implemented as vBuckets, 1024 of which are created for every bucket (except on MacOS, where the number is 64).
- vBuckets are distributed evenly across the memory and storage facilities of the cluster; and the bucket's items are distributed evenly across its vBuckets.
- This evenness of distribution ensures that all instances of the Data Service take an approximately equal share of the workload, in terms of numbers of documents to maintain, and operations to handle.

Auto sharding – Bucketand vBuckets



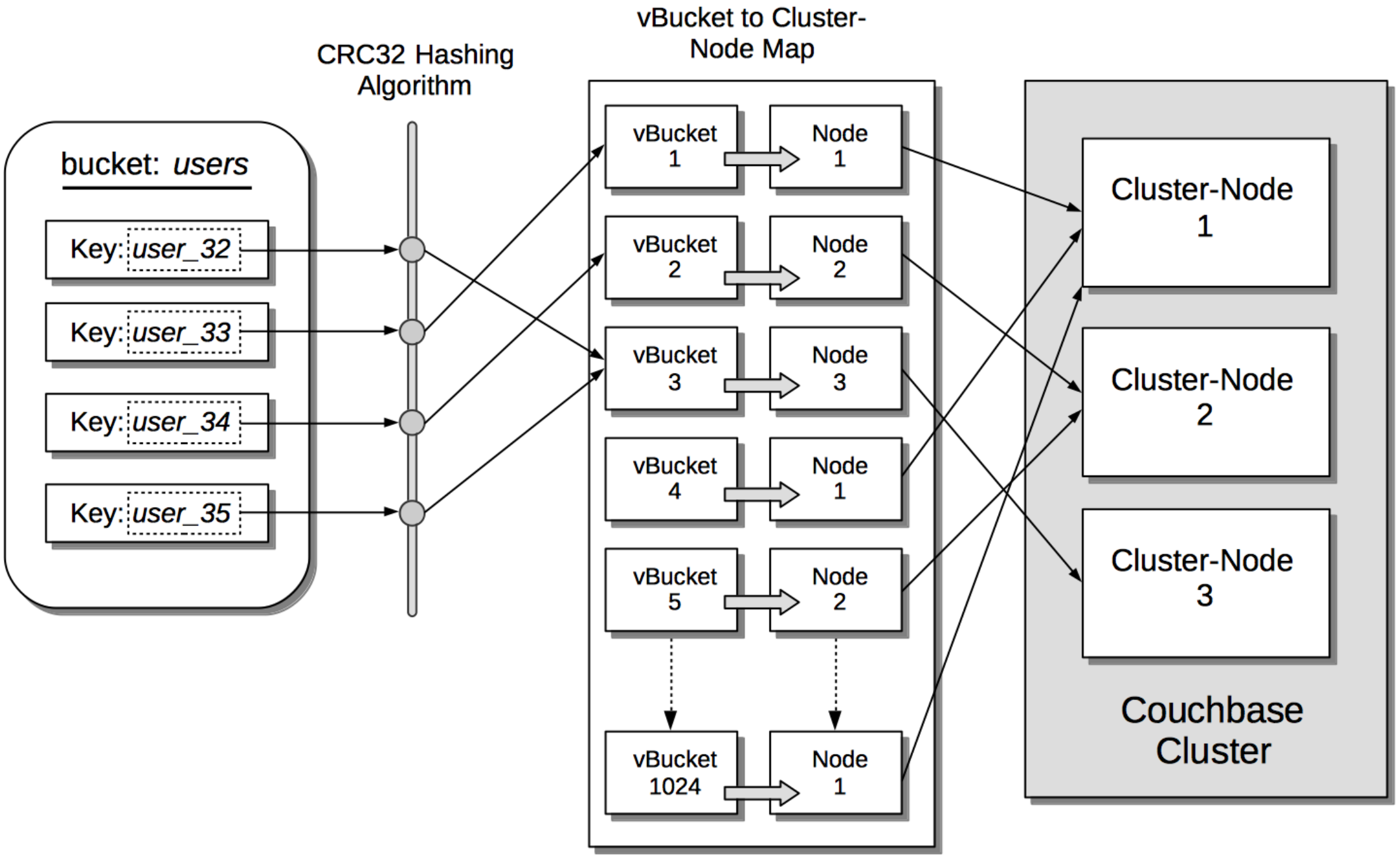
Cluster Map


APP
Couchbase SDK
Key
SFO MQ2
CRC32
Hashing Algorithm
CLUSTER MAP
vBucket
vBucket1
vBucket2
vBucket3
vBucket4
vBucket5
vBucket6
vBucket7
...
vBucket1024
Couchbase Cluster


APP
Couchbase SDK
Key
SFO MQ2
CRC32
Hashing Algorithm
CLUSTER MAP
vBucket
vBucket1
vBucket2
vBucket3
vBucket4
vBucket5
vBucket6
vBucket7
...
vBucket1024
Couchbase Cluster

Understanding vBuckets

- 1024 vBuckets that implement a defined bucket are referred to as active vBuckets.
- If a bucket is replicated, each replica is implemented as a further 1024 (or 64) vBuckets, referred to as replica vBuckets.
- Write operations are performed only on active vBuckets.
- Most read operations are performed on active vBuckets, though items can also be read from replica vBuckets when necessary.
- vBuckets are sometimes referred to as shards.
- Items are written to and retrieved from vBuckets by means of a CRC32 hashing algorithm, which is applied to the item's key, and so produces the number of the vBucket in which the item resides.
- vBuckets are mapped to individual nodes by the Cluster Manager: the mapping is constantly updated and made generally available to SDK and other clients.


bucket: users
Key: user_32
Key: user_33
Key: user_34
Key: user_35
CRC32 Hashing Algorithm
vBucket to Cluster-Node Map
vBucket 1
Node 1
vBucket 2
Node 2
vBucket 3
Node 3
vBucket 4
Node 1
vBucket 5
Node 2
vBucket 1024
Node 1
Cluster-Node 1
Cluster-Node 2
Cluster-Node 3
Couchbase Cluster

Understanding vBuckets

- An authorized client attempting to access data performs a hash operation on the appropriate key, and thereby calculates the number of the vBucket that owns the key.
- Client then examines the vBucket map to determine the server-node on which the vBucket resides; and finally performs its operation directly on that server-node.
- When a cluster-configuration changes — due to rebalance, failover, or node-addition — replica buckets are promoted to primaries if appropriate, and both primaries and replicas are redistributed across the available nodes of the modified cluster.
- The vBucket map is duly updated by the Cluster Manager.
- The updated map is then sent to all cluster-participants.
- Use of client-side hashing for access to Couchbase and Ephemeral bucket-data contrasts with the Memcached data-access method; which requires active management of the server-list, and a specific hashing algorithm, such as Ketama, to handle topology-changes.

Expiration

- Bucket Time To Live (TTL) imposes a maximum lifespan on items within a bucket, and thus ensures the expiration of such items, once the specified period is complete.
- TTL is a non-negative integer-value, specified per bucket, that determines the maximum expiration times of individual items
- If Bucket TTL is enabled, each newly created item lives for the number of seconds specified by Bucket TTL, following the item's creation.
- After its expiration time is reached, the item will be deleted by Couchbase Server.
- Deleted items and their contents are no longer available: access-attempts are treated as if the items never existed.
- Maximum value for Bucket TTL is MAX32INT seconds (2147483648, or 68.096 years).
- Default is 0, which indicates that Bucket TTL is disabled.
- If Bucket TTL is changed from the default, it is thereby enabled.

Expiration

Whenever Bucket TTL is modified, items that existed prior to the modification:

- Remain subject to the previous Bucket TTL, if Bucket TTL was previously enabled.
- Remain subject to no Bucket TTL, if Bucket TTL was not previously enabled.
- Items created or modified following the modification of Bucket TTL are subject to the modified Bucket TTL.
- Bucket TTL can be established on Couchbase and Ephemeral buckets.
- Cannot be established on Memcached buckets.
- Note that Bucket TTL is only available in the Enterprise Edition of Couchbase Server.

Expiration: Bucket versus Item

- An expiration time can be specified per document, by means of Couchbase SDK APIs: this is referred to as the item's TTL.

If a new item's TTL is specified to be:

- Above that of Bucket TTL, the item's TTL is reduced to the value of Bucket TTL.
- Below that of Bucket TTL, the item's TTL is left unchanged.
- 0, the item's TTL is reset to the value of Bucket TTL.

Post-Expiration Purging

When its expiration time is reached, an item is deleted as soon as one of the following occurs:

- An attempt is made to access the item.
- The expiry pager is run.
- Compaction is run.

- For each item that is deleted, a tombstone, which includes key and metadata, continues to be maintained by Couchbase Server for some period of time, as a record
- Applies to all instances of item-deletion, including those achieved by specifying Bucket TTL.
- To ensure that no trace of deleted items remains, tombstones are removed through a Metadata Purge.
- This is an automatic, non-disruptive background-process, which can be scheduled if necessary by means of Couchbase Web Console
- Console allows the intervals between purges to be established as appropriately low, so that tombstones are removed promptly.

Bucket-Expiration and XDCR

- When Cross Data-Center Replication occurs, the Bucket TTL setting does not get propagated from the source bucket to the target.
- However, items within the source bucket that have individual expiration times, including ones derived from the source Bucket TTL setting, are replicated to the target along with their individual expiration times.
- Buckets in the target cluster can have their own Bucket TTL settings, which may differ from those of the buckets in the source.
- If, on the target cluster, the TTL of a replicated item is:
  - Above that of Bucket TTL, the item's TTL is reduced to the value of Bucket TTL.
  - Below that of Bucket TTL, the item's TTL is left unchanged.
  - 0, the item's TTL is reset to the value of Bucket TTL.

Bucket-Expiration and XDCR

- In cases where the TTL of the replicated item is above that of Bucket TTL or is 0, if bi-directional XDCR has been set up, the TTL of the item at the source is eventually also either reduced or reset to the value of the target's Bucket TTL.
- When an item is replicated by XDCR, its expiration time is communicated to the target as absolute.
- This requires that the system clocks of the respective clusters be fully synchronized; otherwise, inconsistent behavior may result (as, for example, in the case where an item arrives at the target-cluster with an absolute expiration time that is earlier than the current time acknowledged by the target-cluster).
- Note that optionally, TTL can be omitted from a document replicated by means of XDCR

Setting Bucket-Expiration

An expiration time can be set for a bucket by means of any of the following:

- The UI provided by Couchbase Web Console.
- The Couchbase CLI.
- The Couchbase REST API.

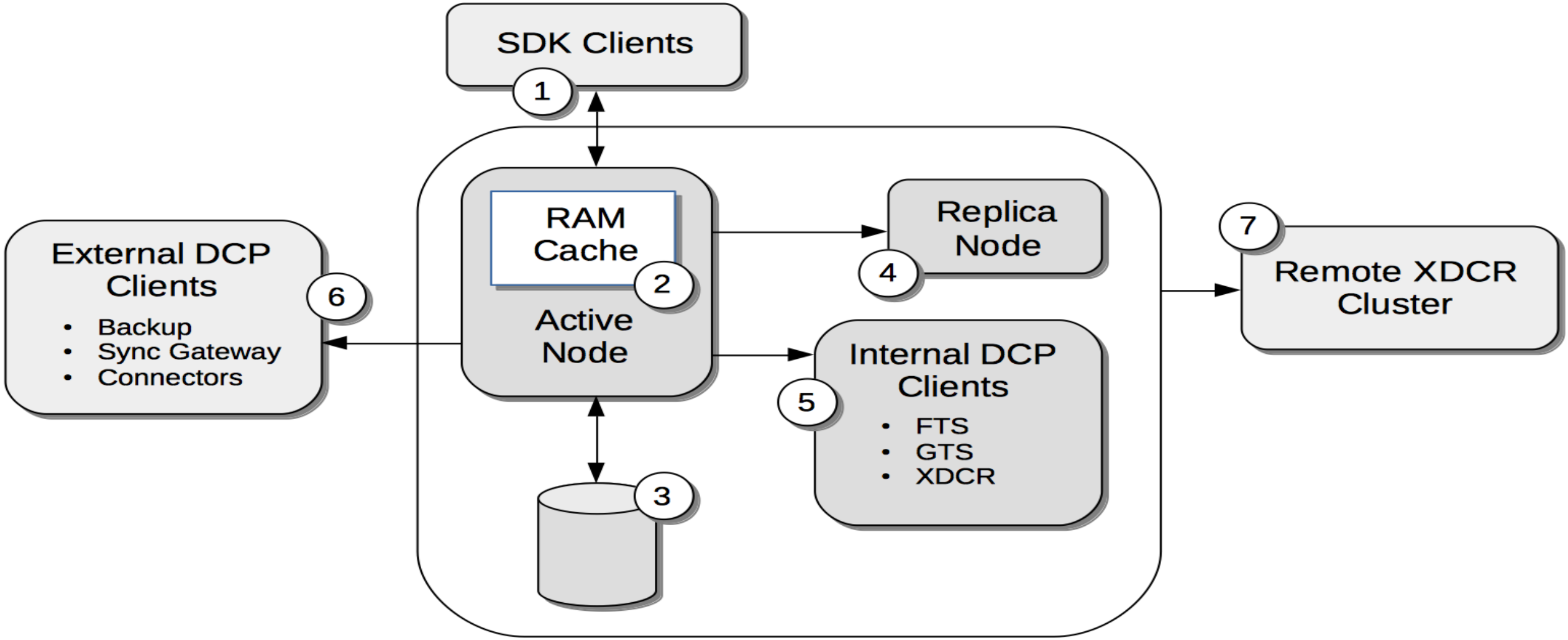
Create a Bucket Using Couchbase Web Console

- A maximum of 30 buckets can be created per cluster.
- To create a bucket, access Couchbase Web Console, and left-click on the Buckets tab, in the vertical navigation-bar at the left-hand side

Compression

- Supports data compression in its communications with internal and external clients, and in its internal handling of items.
- Allows RAM and disk-space to be used with increased efficiency.
- Reduces consumption of network bandwidth.
- Higher consumption of CPU resources may result.
- Compression, provided by the open-source library Snappy, is applied to items based on the client's capabilities, on the compression mode established by the user for the given bucket, and on a required minimum size for the document before compression, in relation to its size after compression (referred to as the compression ratio).
- Available only in Couchbase Enterprise Edition, and can be applied only to Couchbase and Ephemeral buckets.
- Applies to both binary and JSON items.

Where Compression is Used



Where Compression is Used

- Clients based on the Couchbase SDK (1),
- Nodes within a cluster that participate in intra-cluster replication (4)
- Internal Couchbase Services (5),
- External DCP clients (6), and remote clusters that participate in Cross Data-Center Replication (7) communicate their ability to send and receive compressed items by using the HELO command, with a flag that confirms support of the Snappy Compression data type.
- Couchbase Server may (depending on the mode of the bucket) store items in compressed form in memory (2).
- The server always compresses items when storing them on disk (3

Compression Modes

- Each bucket is configured to support one of three modes.
- After a client has communicated its ability to send and receive compressed data, the server's running of compression and decompression routines depends on the mode supported by the specified bucket.
- The modes are as follows:
- Off
- Active
- Passive

Compression Mode - Off

- Provides the behavior of Couchbase Server pre-5.5.
- On receipt of a compressed item, Couchbase Server decompresses the item when storing in memory; and recompresses it when storing on disk.
- Couchbase Server sends the item in uncompressed form.
- Mode is assigned by default to buckets upgraded from a previous version of Couchbase Server.
- Recommended for use where clients cannot benefit from compression, and where neither memory-resources nor network-bandwidth will be negatively impacted by the size and quantity of items to be handled.
- Note also that this is the only mode under which Memcached buckets operate.

Compression Modes - Passive

- On receipt of a compressed item, Couchbase Server stores it in compressed form both in memory and on disk.
- Sends the item back to the client in compressed form if this is requested by the client; otherwise, it sends the item back in uncompressed form.
- On receipt of an uncompressed item, Couchbase Server stores it in memory in uncompressed form, and stores it on disk in compressed form.
- Returns the item to the client in uncompressed form.
- Mode is assigned by default to new buckets, in Couchbase Server 5.5 and beyond.
- Supports clients that themselves handle compression, and additionally allows Couchbase Server to limit its use of memory-resources and network-bandwidth.
- Does not force Couchbase Server to use CPU resources for the compression and decompression of items that clients do not themselves require in compressed form.

Compression Mode - Active

- Actively compresses items for storage in memory and on disk, even if the items are received in uncompressed form.
- Items are decompressed before being sent back to those clients that do not support the receiving of compressed data.
- Items are sent in compressed form to clients that do support the receiving of compressed data, even if those clients originally sent the items to the server in uncompressed form.
- Mode allows the server to practice the maximum conservation of memory-resources and network-bandwidth.
- More items can be held in memory simultaneously, and thereby accessed with improved, overall efficiency (since the processing-time required for compression and decompression is significantly less than that required for fetching data from disk).
- Clients that do not themselves require the compression and decompression of items may be negatively affected by the compression and decompression performed by the server on items resident in memory.

Switching Between Compression Modes

- Buckets can be switched between modes.
- When a bucket formerly in Passive mode has been switched to Off mode, any compressed item that Couchbase Server receives for that bucket is stored in memory in uncompressed form.
  - If the server needs to send from the bucket an item that is currently compressed, the server decompresses the item before sending: however, to preserve memory-efficiency, the item remains compressed in memory.
- When a bucket has been switched from Passive to Active mode, it periodically runs a task that compresses uncompressed items.
- When a bucket formerly in Active mode is switched to Off mode, this disables the task that was periodically run to compress uncompressed items.
  - Compressed items can continue to be received for the bucket, and are decompressed for storage in memory.
  - Compressed items within the bucket are decompressed before sending: however, to ensure continued memory-efficiency, the item remains compressed in memory.

Compression Ratio

- Cluster-wide parameter that specifies the minimum required size of a document before compression, in relation to its size after compression.
- If the document does not meet this requirement, it is retained in uncompressed form.
- Cluster-wide value for the parameter is 1.2.

- Document is retained in compressed form only if original_size >= (1.2 * compressed_size).
- In cases where original_size < (1.2 * compressed_size), the document is retained in uncompressed form.

Service Memory Quotas

- Memory-quota allocation on Couchbase Server occurs per service (except in the case of the Query Service, which does not require a specific allocation).
- Allows the availability of memory-resources to be tuned according to the assignment of services, node by node.
- Data Service must run on at least one node; and that on each of those nodes, quotas for buckets, specified at the time of bucket-creation, are subtracted from the quota allocated to the Data Service.
- Applies to every instance of that service across the cluster.
- For example, if 2048 MB is specified for the Analytics Service, and the Analytics Service is run on three of a cluster's five nodes, each of the three instances of the service is duly allocated 2048 MB.
- Not possible to have different memory allocations across multiple instances of the same service within a single cluster.
- By default, Couchbase Server allows 90% of a node's total available memory to be allocated to the server and its services.
- Consequently, if a node's total available memory is 100 GB, any attempt to allocate memory beyond 90 GB produces an error.

Service Memory Quotas

- When a node is added to a cluster, the Default Configuration, as established by the set-up of the first node in the cluster, is available: this covers all configurable elements, including memory quotas.
- If insufficient memory for the default configuration is available on the new node, the default configuration is prohibited: in such cases, settings for the new node can be custom-configured, allowing an appropriate subset of services to be specified.
- If, when the initial node of a cluster is set up, only a subset of services is assigned, additional nodes can subsequently be added, in order to host additional services: in which case, each new service can be given its initial memory allocation as its node is added.

Minimum-required memory-quotas

Service	Minimum Memory Quota (in MB)
Data	256
Index	256
Search	256
Analytics	1024
Eventing	256

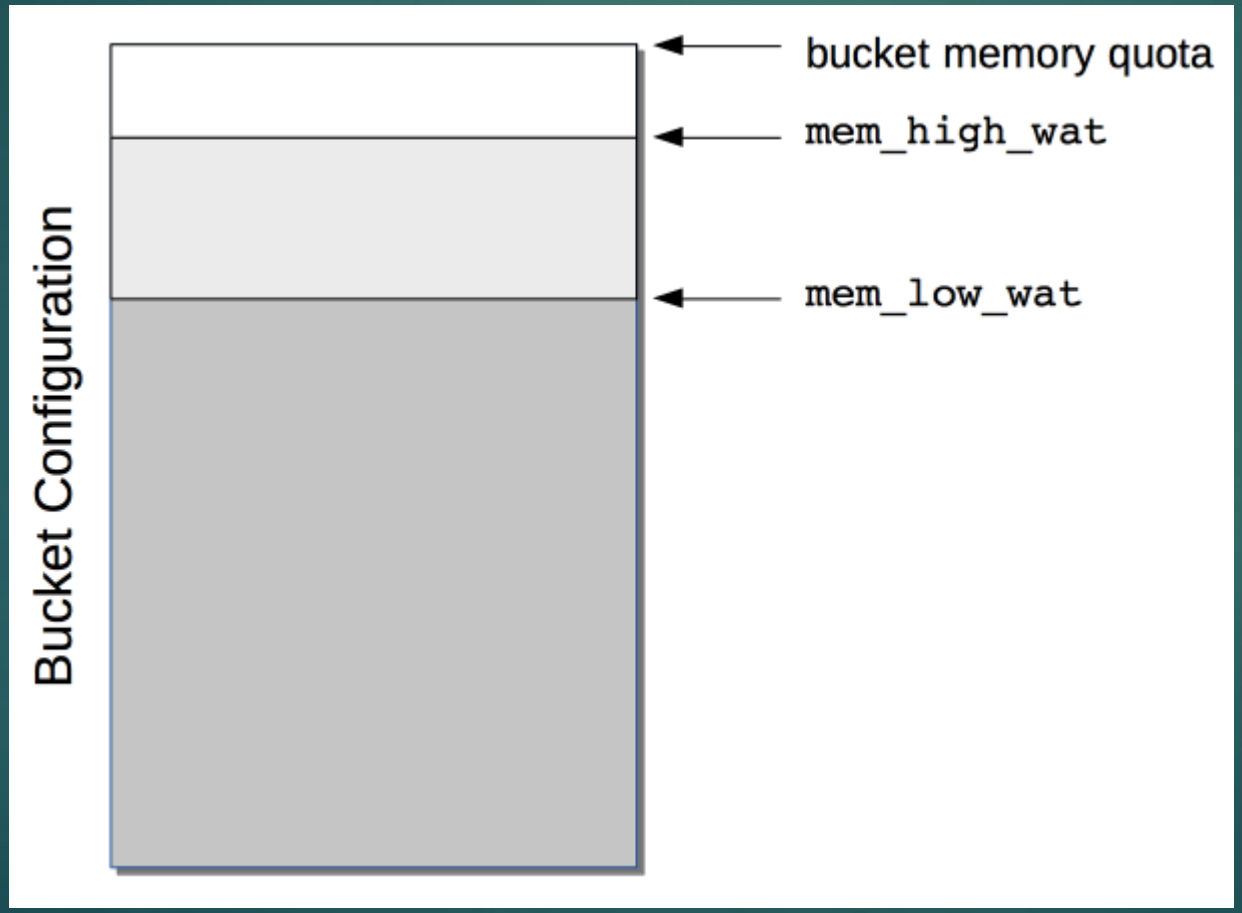
Ejection

- If a bucket's memory quota is exceeded, items may be ejected from the bucket by the Data Service.
- Different ejection methods are available, and are configured per bucket. Note that in some cases, ejection is configured not to occur.
- Items are selected for ejection based on metadata that each contains, indicating whether the item can be classified as Not Recently Used (NRU). If an item has not been recently used, it is a candidate for ejection.

Ejection

- For each bucket, available memory is managed according to two watermarks, which are mem_low_wat and mem_high_wat.
- If data is continuously loaded into the bucket, its quantity eventually increases to the value indicated by the mem_low_wat watermark.
- At this point, no action is taken.
- Then, as still more data is loaded, the data's quantity increases to the value indicated by the mem_high_wat watermark.
- If, based on the bucket's configuration, items can be ejected from the bucket, the Data Service ejects items from the bucket until the quantity of data has decreased to the mem_low_wat watermark.
- In cases where ejection cannot free enough space to support continued data-ingestion, the Data Service stops ingesting data, error messages are sent to clients, and the system displays an insufficient memory notification.
- When sufficient memory is again available, data-ingestion resumes.

Ejection


Bucket Configuration
bucket memory quota
mem_high_wat
mem_low_wat

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.
- Expiry pager runs every 60 minutes by default

Active Memory Defragmenter

- Over time, Couchbase Server-memory can become fragmented.
- Each page in memory is typically responsible for holding documents of a specific size-range.
- Over time, if memory pages assigned to a specific size-range become sparsely populated (due to documents of that size being ejected, or to items changing in size), the unused space in those pages cannot be used for documents of other sizes, until a complete page is free, and that page is re-assigned to a new size.
- Such effects, which are highly workload-dependent, may result in memory that cannot be used efficiently.
- Periodically scans the cache, to identify pages that are sparsely used.
- Repacks the items on those pages, to free up space.