**VIETNAM NATIONAL UNIVERSITY**

**UNIVERSITY OF INFORMATION TECHNOLOGY**

**FACULTY OF INFORMATION SYSTEM**

****

**DISTRIBUTED DATABASE MANAGEMENT SYSTEM**

**COUCHBASE SERVER**

Phan Nguyễn Lâm Hà 21522030

Trương Gia Huy 21522172

Nguyễn Duy Khánh 21522208

Nguyễn Hoàng Sơn 21520433

**Instructor:** Mr. Nguyễn Hồ Duy Tri

Ho Chi Minh City, 2023

[1. Introduction 4](#_Toc153856022)

[1.1. Overview 4](#_Toc153856023)

[1.2. Authors, management organization and history 5](#_Toc153856024)

[1.3. Data Model 6](#_Toc153856025)

[2.2.1. Keys 7](#_Toc153856026)

[2.2.2. Values 7](#_Toc153856027)

[2.2.3. Document Structure 7](#_Toc153856028)

[2.2.4. MetaData 8](#_Toc153856029)

[2.2.5. Document vs Table 10](#_Toc153856030)

[1.4. Bucket 12](#_Toc153856031)

[2.2.6. Bucket Types 12](#_Toc153856032)

[2.2.7. Vbuckets 13](#_Toc153856033)

[2.2.8. Scope and Collections 14](#_Toc153856034)

[1.5. SQL++ Language 15](#_Toc153856035)

[1.6. Local and Remote Replication 17](#_Toc153856036)

[1.6.1. Intra-Cluster Replication 17](#_Toc153856037)

[1.6.2. Cross Data Center Replication(XDCR) 19](#_Toc153856038)

[1.6.2.1. Different between XDCR and ICR 19](#_Toc153856039)

[1.6.2.2. XDCR Direction and Topology 20](#_Toc153856040)

[2. INSTALLATION AND SET UP 22](#_Toc153856041)

[2.1. Installation on one node 22](#_Toc153856042)

[2.1.1. Create a Cluster 22](#_Toc153856043)

[2.1.2. Add Sample Bucket 27](#_Toc153856044)

[2.2. Installation on cluster of nodes 28](#_Toc153856045)

[2.2.1. Add A Node and Rebalance 28](#_Toc153856046)

[3. Perform Distributed Stimulation 33](#_Toc153856047)

[3.1. Description of problem posed 33](#_Toc153856048)

[3.2. Description of data structure used 33](#_Toc153856049)

[3.3. Empirical steps 33](#_Toc153856050)

[References 50](#_Toc153856051)

# Introduction

## Overview

Distributed Database Management System: CouchBase Server

* CouchBase Server is an open source, NoSQL, JSON document-oriented, distributed data platform. Data is stored as items, each having a key and a value.
* Sub-millisecond data operations are provided by powerful services for querying and indexing, and by a feature-rich, document-oriented query-language, SQL++
* Multiple instance can be combined intro a single cluster. A Cluster Manager program coordinates all node-activites.
* Data can be retained either in memory only, or in both memory and storage. Data can also be replicated across the nodes of the cluster to ensure that node-loss does not entail data-loss. Data items can also be selectively replicated across data center; for the purpose of either of backup only, or of stimulat-neous, multi-geo application-accress.

## Authors, management organization and history

* Developers: CouchBase Inc
* Written in: C++, Erlang, C, Go, Java
* Intial release: August 2010
* Stable release: 7.2.2 / September 14,2023
* Membase was developed by several leaders of the memcached project, who had founded a company, NorthScale, to develop a key-value store with the simplicity, speed, and scalability of memcached, but also the storage, persistence and querying capabilities of a database. The original membase source code was contributed by NorthScale, and project co-sponsors Zynga and Naver Corporation (then known as NHN) to a new project on membase.org in June 2010.

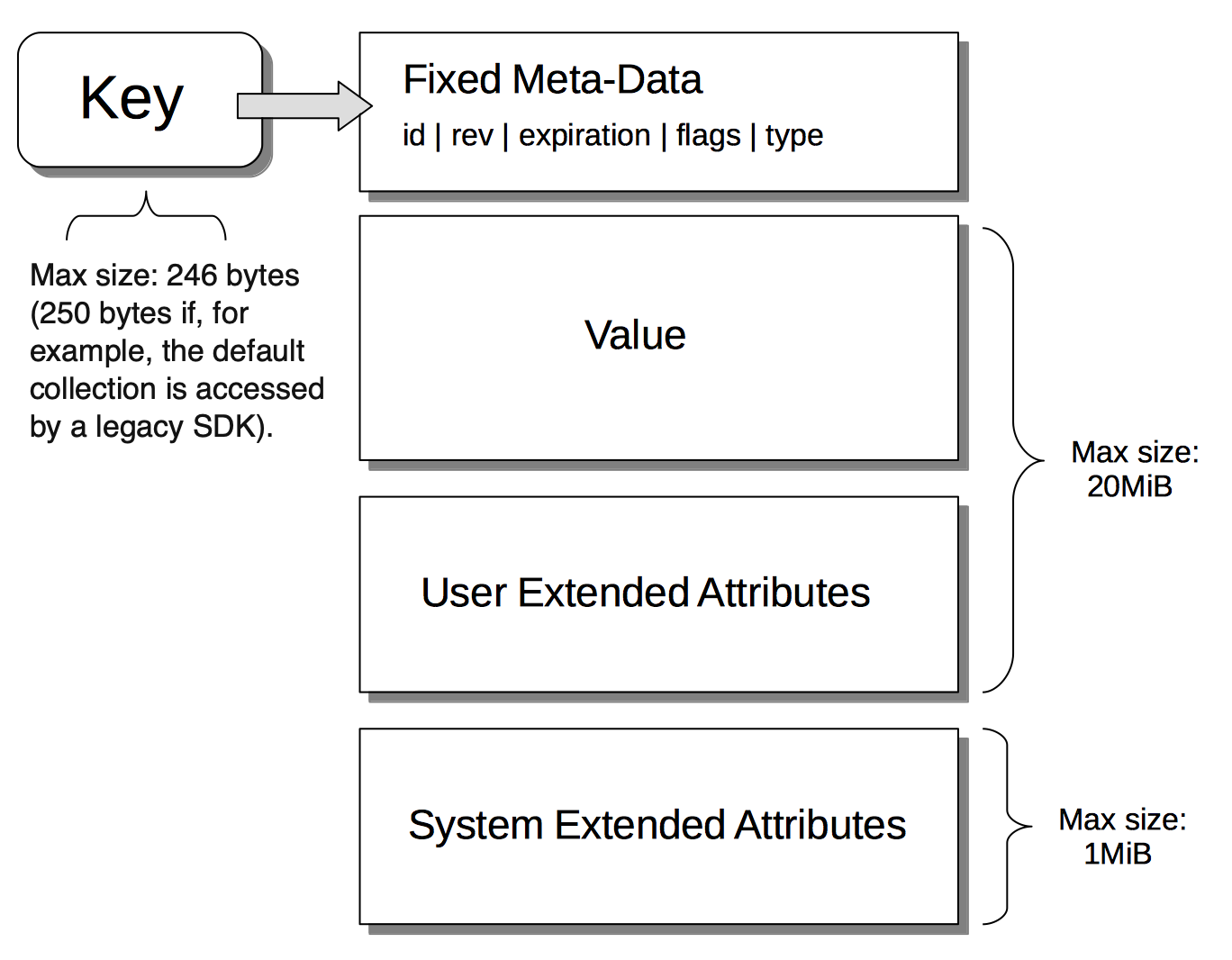
Couchbase product history at a glance

**Table 1-1**

|  |  |
| --- | --- |
| 2011 | Merger of Membase and CouchOne to form Couchbase |
| 2012 | Couchbase Server 2.0 transforms Couchbase from a pure key-value database to a JSON document-oriented database |
| 2014 | Couchbase Mobile 1.0 – First mobile NoSQL database |
| 2015 | Couchbase Server 4.0 adds SQL-like queries with N1QL and Multi-Dimensional Scaling |
| 2017 | Couchbase Server 5.0 introduces Full-Text Search and ephemeral buckets |
| 2018 | Couchbase Server 6.0 eliminates ETL for operational workloads with the introduction of analytics based on SQL++; Couchbase Autonomous Operator for Kubernetes debuts in its 1.0 release |
| 2019 | Couchbase Server 6.5 adds distributed ACID transactions |
| 2020 | Couchbase debuts its Database-as-a-Service product, Couchbase Cloud™ |
| 2021 | Couchbase Server 7.0 fuses the flexibility and scalability of NoSQL with the trusted strengths of relational database |

## Data Model

CouchBase Server saves data as items, each of which has a key and a value. Each key is unique within its bucket, and values can be either binary or JSON.



Picture ‑

### Keys

Each value is identified by a unique key(id), defined by the user or application when the items is saved. The key is immutable: once the items is saves, the key cannot be changed.

### Values

The maximum size of a value is 20 MiB. A value can be either:

* **Binary:** Any form of binary. Note that a binary value cannot be parsed, indexed, or queries: it can only be retrieved by key
* **JSON:** A JSON value, reffered to as a document, can be parsed, indexed and queries. Each document consist of one or more atrribute, each of which has its own value. An attribute’s value can be a basic type, such as a number, string, or Boolean; or a complex, such as an embedded document or an array.

### Document Structure

A sample JSON document is provided immediately below. Its contents are as follows:

* a1 and a2 are attributes that respectively have a single number and a single string as their values.
* a3 is an attribute whose value is an embedded document consisting of a single attribute, b1, whose own value is an array of three numbers.
* a4 is an attribute whose value is an array of two documents, each document consisting an attribute (c1 or c2) whose own values are respectively a string and a number.

A computer screen shot of a number

Description automatically generated

Picture ‑

**Sub-Document:** an inner component of an JSON document. The sub-document is refferd to by a path, which specified attributes and array-position

Ex: a3.b1[0], value is the first number in an array

### MetaData

Metadata is automatically generated and stored for each item saved in Couchbase Server. For example, the following document, which contains airport-information, has been saved with the key **airport\_1306:**

**A screenshot of a computer code

Description automatically generated**

Picture ‑

The meta data might be as follows:

A screenshot of a computer code

Description automatically generated

Picture ‑

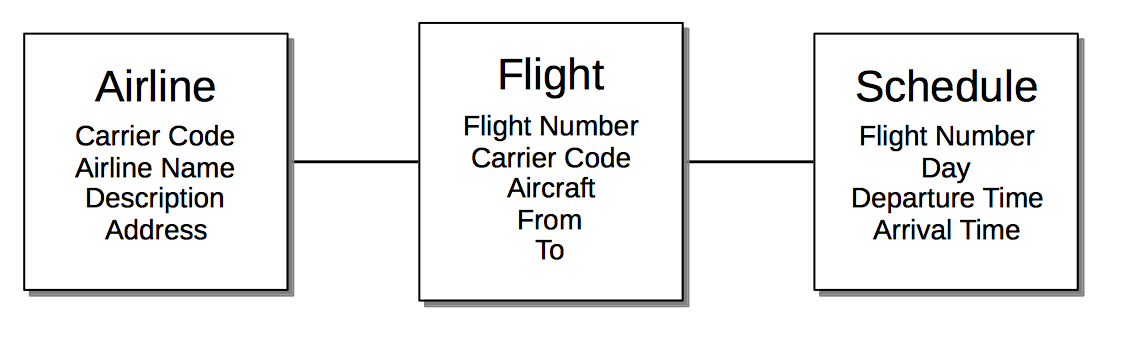
The attributes within the metadata are:

* meta: The attribute whose value is the standard metadata for the saved document, airport\_1306.
* **id: The key of the saved document, which is airport\_1306.**
* rev: The revision or sequence number. This value is for internal server-use only: it is used in the resolution of conflicts that occur when replicated documents are updated concurrently on different servers, representing the number of times the document has been mutated.
* expiration: The expiration-time (or Time-To-Live) of the document. If non-zero, this determines the point at which the document is removed from the system.
* flags: Couchbase SDK-specific values that may be used to identify the type of data saved, or to specify formatting.
* type: The type of the saved value, which in this case is json.
* xattrs: Extended Attributes, which constitute a special kind of metadata, some of which is system-internal, some of which can optionally be written and read by user-applications.

### Document vs Relational Model

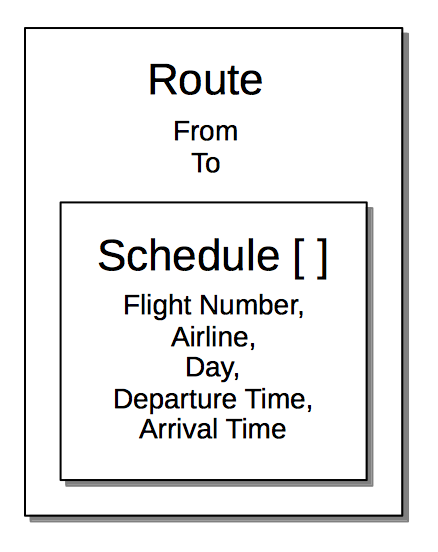
* The CouchBase data model is based on JSON, which provides a simple, lightweight, human-readable notation.
* An individual document often represents a single instance of an object in application code. A document might be considered equivalent to a row in a relational table; with each of the document’s attributes being equivalent to a column. **Couchbase, however, provides greater flexibility than relational databases, in that it can store JSON documents with varied schemas.**
* Documents can contain nested structures. This allows developers to express many-to-many relationships without requiring a reference or junction table; and is naturally expressive of hierarchical data.

Example: The nature and value of the document data model is clarified by comparison with the relational. To support an online flight-booking application, allowing users to search for flights by date, the relational model requires multiple tables — for flights, airlines, and schedules. The result may be as follows:



Picture ‑

By contrast, the document model likely requires only a single document, which embeds an array of schedules for all flights between each of two airports:



Picture ‑

Thus, in the document model, each document can be highly self-contained. This supports the rapid fulfillment of application-requests, and has important implications for both scalability and latency: one document can be replicated, or atomically changed, without other documents needing to be accessed; eradicating the need for complex inter-node coordination, and minimizing contention

## Services

* Couchbase Services support access to and maintenance of data. Services can be deployed with flexibility across available hardware-resources, providing Multi-Dimensional Scaling, whereby a cluster can be tuned for optimal handling of emergent workloads.
* Services are configured and deployed by the Full Administrator who initializes Couchbase Server on one or more nodes. The standard configuration-sequence allows a subset of services to be selected per node, with an individual memory-allocation for each. Each service supports a particular form of data-access. Services not required need not be deployed. Services intended to support a heavy workload can be deployed across multiple cluster-nodes, to ensure optimal performance and resource-availability.



Picture ‑

In this revised configuration, the Data Service is the only service to run on two of the nodes; the Index Service the only service on two futher nodes; and the Query and Search Services share the fifth and final node.

Couchbase Server provides the following services:

* **Data**: Supports the storing, setting, and retrieving of data-items, specified by key.
* **Query**: Parses queries specified in the *N1QL* query-language, executes the queries, and returns results. The Query Service interacts with both the Data and Index services.
* **Index**: Creates indexes, for use by the Query and Analytics services.
* **Search**: Create indexes specially purposed for *Full Text Search*. This supports language-aware searching; allowing users to search for, say, the word beauties, and additionally obtain results for beauty and beautiful.
* **Analytics**: Supports join, set, aggregation, and grouping operations; which are expected to be large, long-running, and highly consumptive of memory and CPU resources.
* **Eventing**: Supports near real-time handling of changes to data: code can be executed both in response to document-mutations, and as scheduled by timers.
* **Backup**: Supports the scheduling of full and incremental data backups, either for specific individual buckets, or for all buckets on the cluster. Also allows the scheduling of *merges* of previously made backups.
* These services can be deployed, maintained, and provisioned independently of one another, by means of *Multi-Dimensional Scaling*, to ensure the most effective ongoing response to changing business conditions and emergent workload-requirements

## Bucket

**A bu**cket is the fundametal space for storing data in CouchBase Server. Each bucket contain a hierachy of scope and collection to group keys and values logically.

### Bucket Types

A maximum of 30 buckets can be created in a cluster. Each bucket must be specified as one of the following three types.

* **Couchbase buckets:** These store data persistently, as well as in memory. They 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:** These are an alternative to Couchbase buckets, to be used whenever persistence is not required: for example, when repeated disk-access involves too much overhead. This allows highly consistent in-memory performance, without disk-based fluctuations. It also allows faster node rebalances and restarts.

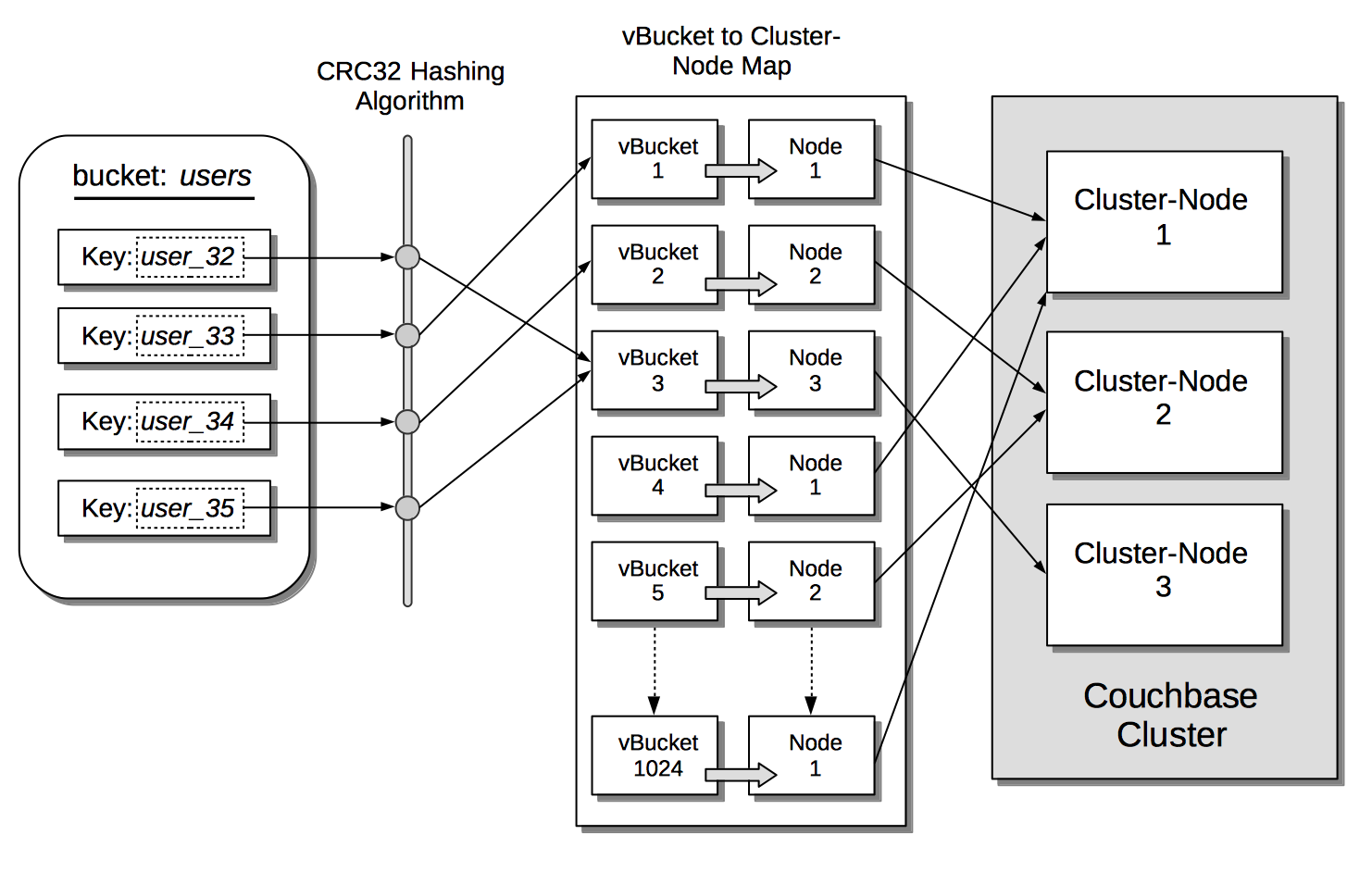
If an Ephemeral bucket’s RAM-quota is exceeded, one of the following occurs, based on configuration performed at the time of bucket-creation:

* 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.

### Vbuckets

vBuckets are virtual buckets that help distribute data effectively across a cluster, support replication across multiple nodes.

* Both CouchBase and Ephermal buckets are implemented as Vbuckets, up to 1024 of which are created for every bucket.
* Buckets’s items are distributed evenly across its vBukets
* The 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 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.
* 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.
* The relationships between a bucket, its keys (and their associated values), the hashing algorithm, vBuckets, server-mappings, and servers, is illustrated below:



Picture ‑

Thus, 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. The 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.

* In Couchbase Server Version 7.0+, documents within a bucket are organized into Scopes and Collections. Scopes and collections do not affect the way in which keys are allocated to vBuckets.

### Scope and Collections

* Collection: A data container, within a bucket whose type is either CouchBase or Ephemeral.
  + Item-names must be unique within their collections
  + Items can optionally be assigned to different collections according to content-type
* Scope: a mechanism for grouping of multiple collections.
* Collection-names must be unique within their scope
* Collection might be assigned to different scope according to content-type, or deployment-phase

A diagram of a cluster

Description automatically generated

Picture ‑

### Indexes

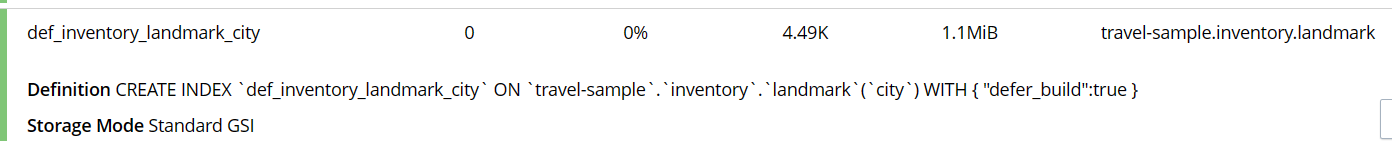
* Indexes enhance the performance of querry and search operation
* Indexes are used by certain services, such as Query, Analytics, and Search, as targets for search-routines. Each index makes a predefined subset of data available for the search.
* The Query service relies on indexes provided by the Index service. The Search and Analytics services both provide their own indexes, internally.
* The following forms of index are available:
* Primary: Provided by the Index Service, this is based on the unique key of every item in a specified collection. Every primary index is maintained asynchronously. A primary index is intended to be used for simple queries, which have no filters or predicates.

A close up of a computer screen

Description automatically generated

Picture ‑

* Secondary: Provided by the Index Service, this is based on an attribute within a document. The value associated with the attribute can be of any type: scalar, object, or array. A Secondary Index is frequently referred to as a Global Secondary Index, or GSI. This is the kind of index used most frequently in Couchbase Server, for queries performed with SQL++



Picture ‑

* Full Text: Provided by the Search Service, this is a specially purposed index, which contains targets derived from the textual contents of documents within one or more specified keyspaces. Text-matches of different degrees of exactitude can be searched for. Both input and target text-values can be purged of irrelevant characters (such as punctuation marks or html tags).

## SQL++ Language

* CouchBase Server can be queried using SQL++.
* SQL++ is an expressive, powerful, and complete SQL dialect for querying, transforming, and manipulating JSON data. Based on SQL, it’s immediately familiar to developers who can quickly start developing rich applications.
* The SQL++ language is composed of statements, expressions and comments
* Statement:
  + Data Definition Language(DDL) statement to create indexes, modify indexes and drop indexes
  + Data Manipulation Language(DML) statement to select,insert,update,delete, and upsert data into JSON documents
* Expression:
  + Literal value
  + Identifiers
  + Arithmetic terms
  + Comparison terms
  + Concatenation terms
  + Logical terms
  + Conditional expressions
  + Collection expressions
  + Construction expressions
  + Nested expressions
  + Function calls
  + Subqueries
* Comments: block comments and line comments

A close up of a screen

Description automatically generatedA screenshot of a computer code

Description automatically generated

Picture ‑

## Local and Remote Replication

CouchBase Server ensure the availability of data across the node of a cluster; across group of nodes within the cluster; and across separate clusters, potentially located in different data-centers.

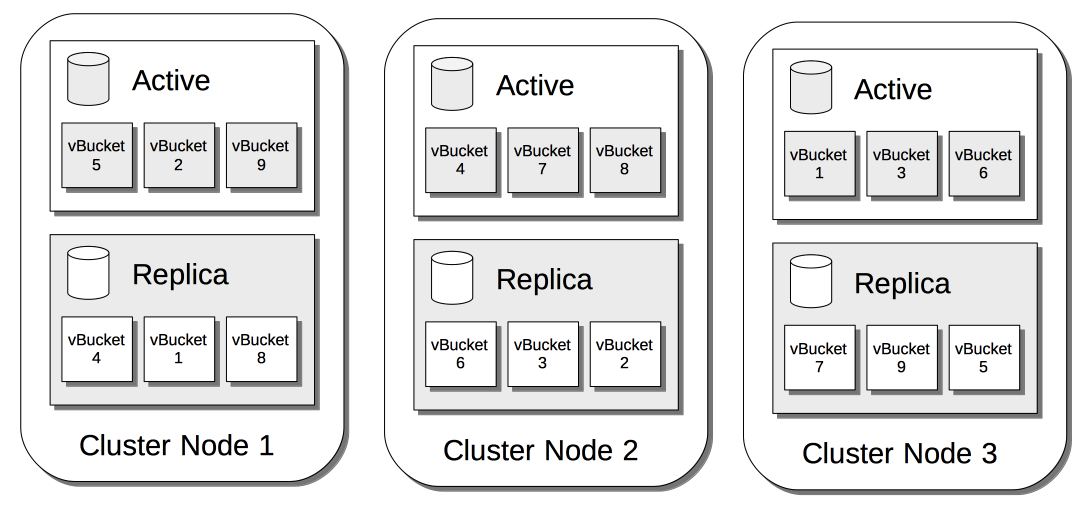
* Local, or intra-cluster replication
* Remote, or Cross Data Center Replication (XDCR)

### Intra-Cluster Replication

Intra-cluster replication involves replicating data across the nodes of a cluster.

Couchbase Data is defined logically to reside in Buckets. Each bucket, when defined, is implemented by Couchbase Server as vBuckets, up to 1024 of which thereby exist in memory (and, in the case of Couchbase Buckets, on disk); the exact number at any given time depending on the number of items to be stored. Items are associated with vBuckets by means of a hashing algorithm, and buckets are assigned to nodes according to a fixed mapping, determined and updated by the Master Services of the ns-server component of the Cluster Manager.

Up to three replica buckets can be defined for every bucket. Each replica itself is also implemented as 1024 vBuckets. A vBucket that is part of the original implementation of the defined bucket referred to as an active vBucket. Therefore, a bucket defined with two replicas has 1024 active vBuckets and 2048 replica vBuckets. Typically, only active vBuckets are accessed for read and write operations: although replica vBuckets are able to support read requests. Nevertheless, vBuckets receive a continuous stream of mutations from the active vBucket by means of the Database Change Protocol (DCP), and are thereby kept constantly up to date. To ensure maximum availability of data in case of node-failures, the Master Services for the cluster calculate and implement the optimal vBucket Distribution across available nodes: consequently, the chance of data-loss through the failure of an individual node is minimized, since replicas are available on the nodes that remain. This is shown by the following illustration:



Picture ‑

The illustration shows active and replica vBuckets that correspond to a single, user-defined bucket, for which a single replication-instance has been specified. The first nine active vBuckets are shown, along with their nine corresponding, replica vBuckets; distributed across three server-nodes. The distribution of vBuckets indicates a likely distribution calculated by Couchbase Server: no replica resides on the same node as its active equivalent: therefore, should any one of the three nodes fail, its data remains available.

### Cross Data Center Replication(XDCR)

* Cross Data Center Replication (XDCR) replicates data between a source bucket and a target bucket. The buckets may be located on different clusters, and in different data centers: this provides protection against data-center failure, and also provides high-performance data-access for globally distributed, mission-critical applications.
* Data from the source bucket is pushed to the target bucket by means of an XDCR agent, running on the source cluster, using the Database Change Protocol. Any bucket (Couchbase or Ephemeral) on any cluster can be specified as a source or a target for one or more XDCR definitions.

#### Different between XDCR and ICR

* Cross Data Center Replication differs from intra-cluster replication in the following, principal ways:
  + As indicated by their respective names, intra-cluster replication replicates data across the nodes of a single cluster; while Cross Data Center Replication replicates data across multiple clusters, each potentially in a different data center.
  + Whereas intra-cluster replication is configured and performed with reference to only a single bucket (to which all active and replica vBuckets will correspond), XDCR requires two buckets to be administrator-specified, for a replication to occur: one is the bucket on the source cluster, which provides the data to be replicated; the other is the bucket on the target cluster, which receives the replicated data.
  + Whereas intra-cluster replication is configured at bucket-creation, XDCR is configured following the creation of both the source and target buckets.
* The starting, stopping, and pausing of XDCR all occur independently of whatever intra-cluster replication is in progress on either the source or target cluster. While running, XDCR continuously propagates mutations from the source to the target bucket.

#### XDCR Direction and Topology

XDCR allows replication to occur between source and target clusters in either of the following ways:

* **Unidirectionally:** The data contained in a specified source bucket is replicated to a specified target bucket. Although the replicated data on the source could be used for the routine serving of data, it is in fact intended principally as a backup, to support disaster recovery.

A diagram of a data center

Description automatically generated

Picture ‑

* **Bidirectionally:** The data contained in a specified source bucket is replicated to a specified target bucket; and the data contained in the target bucket is, in turn, replicated back to the source bucket. This allows both buckets to be used for the serving of data, which may provide faster data-access for users and applications in remote geographies.

A diagram of data center

Description automatically generated

Picture ‑

Note that XDCR provides only a single basic mechanism from which replications are built: this is the unidirectional replication. A bidirectional topology is created by implementing two unidirectional replications, in opposite directions, between two clusters; such that a bucket on each cluster functions as both source and target.

Used in different combinations, unidirectional and bidirectional replication can support complex topologies; an example being the ring topology, where multiple clusters each connect to exactly two peers, so that a complete ring of connections is formed:

A diagram of data center

Description automatically generated

Picture ‑

# INSTALLATION AND SET UP

## Installation on one node

### Create a Cluster

* Connect to CouchBase Web Console via: [**http://localhost:8091**](http://localhost:8091)

A screenshot of a computer

Description automatically generated

Picture ‑

* Click on: Setup New Cluster

A screenshot of a computer

Description automatically generated

Picture ‑

The field display on the screen are:

* Cluster Name: Name for the cluster to be created
* Create Admin Username: your choice of username, Full Administrator: read-write access to all CouchBase Server resources, including the ability to create new users with defined roles and corresponding privileges.
* Create Password: choice of password
* When you have enter appropriate data, left click on Next: Accept Term

A screenshot of a computer screen

Description automatically generated

Picture ‑

* Configure CouchBase Server

A screenshot of a computer

Description automatically generated

Picture ‑

A screenshot of a computer

Description automatically generated

Picture ‑

* Host Name/ IP Address: The name will be used for this node. You can use loopback address 127.0.0.1, when a second node is added to the cluster, the name will be changed to the IP of the underlying host. If you wish, you can substitute the IP address of the underlying host now, or you can substitute the fully qualified hostname of the underlying host
* enable node ecryption:  Check the checkbox to enable node-to-node encryption for the cluster
* IP Family Preferences: Select IP V4(default)
* Service Memory Quotas: Allows you to specify how much memory should be allocated to each service you select for both current node and node subsequently add to the cluster
* Data Service: 256Mib or more
* Index Service: Selection and RAM-allocation to support Global Secondary Indexes. This should be 256 MiB or more
* Query Service: No RAM-allocation
* Search Service: Selection and RAM-allocation for the Full Text Search Service. This should be 256 MiB or more.
* Data and Index Disk Path: default location
* Then Click Save & Finish

A screenshot of a computer

Description automatically generated

Picture ‑

### Add Sample Bucket

* From the Settings screen, select the Sample Buckets tab. The Sample Buckets screen now appears, as follows:

A screenshot of a computer

Description automatically generated

Picture ‑

* Choose travel-sample and click Load Sample Data

## Installation on cluster of nodes

### Add A Node and Rebalance

1. Access Server Screen



Picture ‑

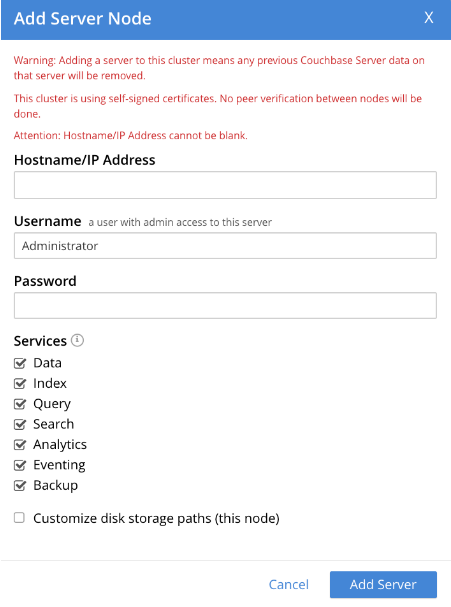
1. A screenshot of a computer

   Description automatically generatedEnsure that the node to be added has been started. This can be accomplished by checking the IP address and port number for the new node in the address bar of the browser. The following interface is displayed:

Picture ‑

This indicates that Couchbase Server is installed and running on the new node, but has not yet been provisioned. Do not use this interface: instead, return to Couchbase Web Console for the cluster, **10.142.181.101.**

1. Click ADD SERVER button



Picture ‑

Note the warning provided at the top of the dialog: if the node to be added has already been provisioned, the results of such provisioning will be eliminated and replaced on the node’s addition to the current cluster. (In fact, the node to be added in this example, has neither been initialized nor provisioned.)

1. Specify the IP address of the node to be added. A placeholder password must be specified, even though the node has not yet been provisioned with one. Uncheck all of the Services check-boxes except Data. The dialog now appears as follows:

A screenshot of a computer

Description automatically generated

Picture ‑

Left-click on the Add Server button to save the settings. The Servers screen is redisplayed, with the following appearance:

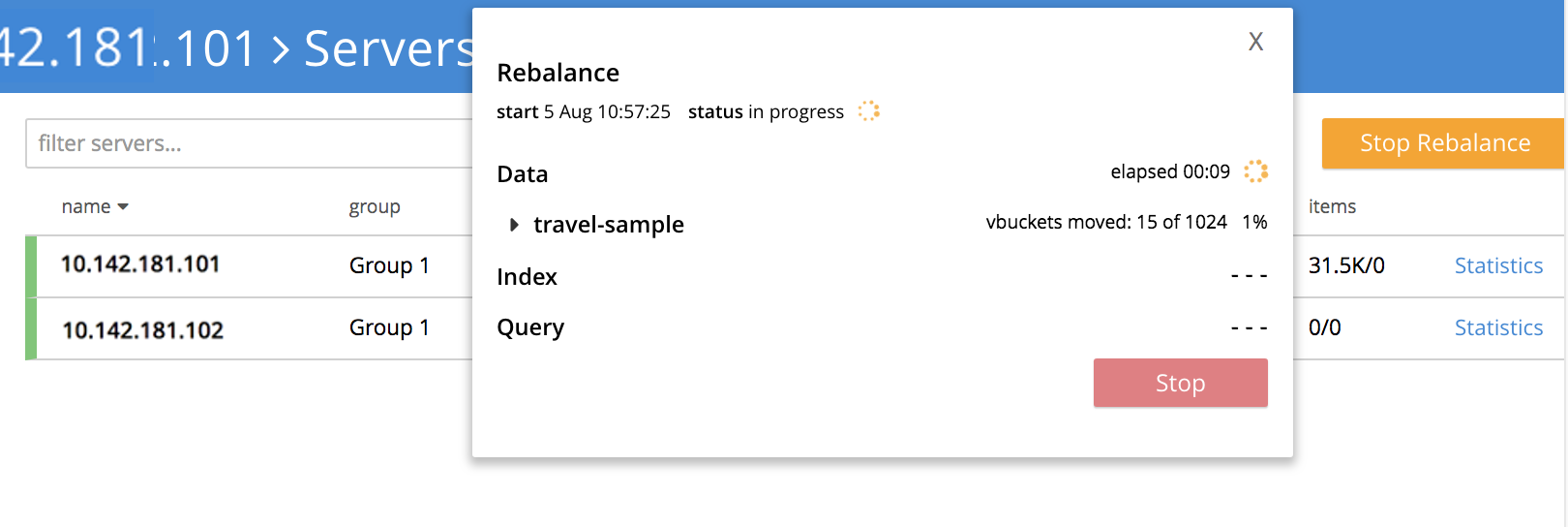
A screenshot of a computer

Description automatically generated

This indicates that the new node, 10.142.181.102 has been successfully added. However, it is not yet taking traffic, and will be added following a *rebalance*. Note, at this point, the figure under the Items column for for 10.142.181.101: this is 63.1 K/0, which indicates that the node contains 63.1 K items in *active* vBuckets, and 0 items in *replica* vBuckets. Meanwhile, the Items figure for 10.142.181.102 is 0/0, indicating that no items are yet distributed onto that node in either active or replica form.

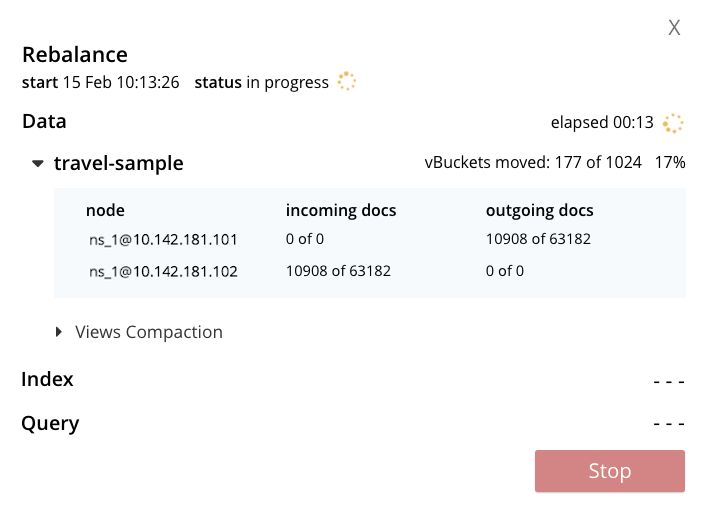
1. To perform a rebalance, left-click on the Rebalance button, at the upper right

The new node is rebalanced into the cluster, meaning that whatever active and replica vBuckets were previously distributed across the original cluster nodes are redistributed across the superset of nodes created by the addition. Additionally, a Rebalance dialog is displayed:



Picture ‑

The dialog indicates rebalance progress for each of the services on the cluster. To see more information on the progress related to the Data Service, left-click on the **travel-sample** tab, under **Data**: The pane expands to provide additional information on the progress of data-transfer for the **travel-sample** bucket:



Picture ‑

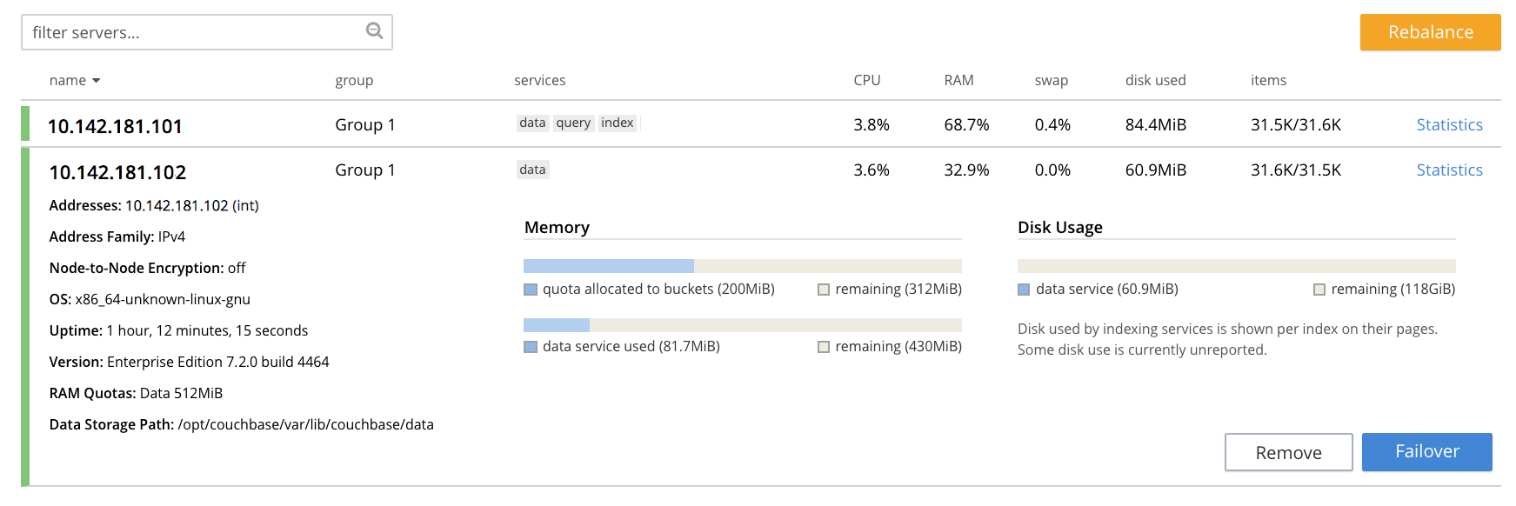
When the rebalance is complete, the panel at the bottom of the screen disappears, and the dialog appears as follows:



Picture ‑

Left-click on the **X** at the upper-right of the dialog, to dismiss the dialog.

The **Servers** screen now displays two rows — one each for servers 10.142.181.101 and 10.142.181.102. Left-click on the row for 10.142.181.102, to open it. The screen now appears as follows:



Picture ‑

Note that the figure in the **Items** column for node 10.142.181.101 is 31.5 K/31.6 K, which indicates that 31.5 K items are stored on the node in *active* vBuckets, and 31.6 K in *replica* vBuckets. The figure for 10.142.181.102 indicates the converse. Therefore, replication has successfully distributed the contents of travel-sample across both nodes, providing a single replica vBucket for each active vBucket.

# Perform Distributed Stimulation

## Description of problem posed

Simulating a distributed database system managing information about classes in an educational organization, used and accessed by multiple different applications. In each branch of the organization, there are applications that regularly access information about the classes conducted at that branch. Due to the geographical distance between branches, to optimize the overall system performance, classes will be distributed across nodes corresponding to the locations where they take place. The challenge is that there is a central application that needs to access information about all classes, even though the class information is distributed across multiple locations.

Details: Create 3 server nodes to store about class and subjects:

* Node A will store distributed data about class taught at branch A
* Node B will store distributed data about class taught at branch B
* Node C will access data from both A and B

## Description of data structure used

**Bucket: ClassesDetails**

* classId: Class code.
* className: Class name.
* beginDate: Start date.
* finalDate: End date.
* location
* room
* site: Branch code, location where the class takes place.

## Empirical steps

* Prepare:
  + Computer A (node A) – IP: 26.192.173.199
  + Computer B (node B) – IP: 26.121.249.103
  + Computer C (node C) – IP: 26.173.173.157
  + Turn off fire wall in all node
  + Node C will be an initial node and have a Bucket
  + Node A and B will be a new node
* **Config node C**
  + Add Bucket in node C:
* Choose Tab Buckets → Choose “ADD BUCKET” in the top right corner

A screenshot of a computer

Description automatically generated

Picture ‑

* A add form bucket will pop up → Fill out the information → Click “OK”

**A screenshot of a computer

Description automatically generated**

Picture ‑

* Bucket was added successfully

****

Picture ‑

* 1. Add Scope and Collection:
* In the end of the Bucket item → Choose “Scopes & Collections”

A close-up of a sign

Description automatically generated

Picture ‑

* List of Scopes will be shown → Choose “ADD SCOPE”

A screenshot of a computer

Description automatically generated

Picture ‑

* Choose Bucket and fill out the name of scope we want to creata (Ex: classScope)

A screenshot of a computer

Description automatically generated

Picture ‑

* Scope classScope was added successfully

A white background with black text

Description automatically generated

Picture ‑

* In the end of that Scope → Choose “Add Collection” → Fill out the name of Collection   
  (EX: classInfo) → Save

A screenshot of a computer

Description automatically generated

Picture ‑

1. **Add node A and B:**

* Check the connection between node A and C, node B and C

**A screenshot of a computer program

Description automatically generated**

Picture ‑

* Choose Tab Buckets → Choose “ADD SERVER” in the top right corner → Enter the IP of the node you want to connect (EX: 26.121.249.103 – node B) and account have admin access

**A screenshot of a computer

Description automatically generated**

Picture ‑

* Add node successfully → then start “Rebalance” → click “Rebalance” button

**A blue and white screen

Description automatically generated**

Picture ‑

* Notificate that Rebalance is starting

A screenshot of a computer

Description automatically generated

Picture ‑

* Rebalance successfully

A screenshot of a message

Description automatically generated

Picture ‑

* Go to node B, this is first UI we have seen

A computer screen shot of a computer

Description automatically generated

Picture ‑

* After node B was added, refresh the page → Sign in UI will be displayed

A computer screen shot of a computer

Description automatically generated

Picture ‑

* Use Admin account to sign in and check

A screenshot of a login

Description automatically generated

Picture ‑

* Bucket was synced from node C

A computer screen shot of a computer

Description automatically generated

Picture ‑

* And node A also

A screenshot of a computer

Description automatically generated

Picture ‑

* List of node in cluster

A white background with black dots

Description automatically generated

Picture ‑

1. Import document and query:
   1. Import document from node A and B:

* Choose tab “Documents”

A screenshot of a computer

Description automatically generated

Picture ‑

* Choose “Import” in the middle top

A screenshot of a computer

Description automatically generated

Picture ‑

* Choose “Select File to Import” button

A screenshot of a computer

Description automatically generated

Picture ‑

* Choose file type and bucket.scope.collection → Then click “import data” button in the bottom middle

A screenshot of a computer

Description automatically generated

Picture ‑

* Back to tab Document to check the data → data was imported successfully

A screenshot of a computer

Description automatically generated

Picture ‑

* Do the same on node B → Check the data

A screenshot of a computer

Description automatically generated

Picture ‑

* Check the data on node C

A screenshot of a computer

Description automatically generated

Picture ‑

* 1. Query data;
* We will query on node C,

A screenshot of a computer

Description automatically generated

Picture ‑

* Create Index at first

A screenshot of a computer

Description automatically generated

Picture ‑

A screenshot of a computer

Description automatically generated

Picture ‑

* Try again

A screenshot of a computer

Description automatically generated

Picture ‑

* Result

A screenshot of a computer code

Description automatically generated

Picture ‑

A screenshot of a computer program

Description automatically generated

Picture ‑

# References

[Couchbase Server - Wikipedia](https://en.wikipedia.org/wiki/Couchbase_Server)

[Couchbase Server | Couchbase Docs](https://docs.couchbase.com/home/server.html)

[Couchbase - Database of Databases (dbdb.io)](https://dbdb.io/db/couchbase)