Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

Architectural Overview of Couchbase Server

A. Messina

Rapporto Tecnico N.:
RT-ICAR-PA-18-02 **Gennaio 2018**

Consiglio Nazionale delle Ricerche, Istituto di Calcolo e Reti ad Alte Prestazioni (ICAR)
– Sede di Cosenza, Via P. Bucci 41C, 87036 Rende, Italy, URL: *www.icar.cnr.it*
– Sede di Napoli, Via P. Castellino 111, 80131 Napoli, URL: *www.na.icar.cnr.it*
– Sede di Palermo, Viale delle Scienze, 90128 Palermo, URL: *www.pa.icar.cnr.it*

Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

Architectural Overview of Couchbase Server

A. Messina[1]

Rapporto Tecnico N.:
RT-ICAR-PA-18-02 **Gennaio 2018**

---

[1] Istituto di Calcolo e Reti ad Alte Prestazioni, ICAR-CNR, Sede di Palermo, Via Ugo La Malfa n. 153, 90146 Palermo.

Index

1 Introduction

The business world is undergoing massive change as industry after industry shifts to the Digital Economy. It's an economy powered by the Internet and other 21st century technologies – the cloud, mobile, social media, and big data. At the heart of every Digital Economy business are its web, mobile, and Internet of Things (IoT) applications: they're the primary way companies interact with customers today, and how companies run more and more of their business. The experiences that companies deliver via those apps largely determine how satisfied customers will be.

How are these applications different from legacy enterprise applications like ERP, HR, and financial accounting? Today's web, mobile, and IoT applications share one or more (if not all) of the following characteristics. They need to:

- Support large numbers of concurrent users (tens of thousands, perhaps millions)
- Deliver highly responsive experiences to a globally distributed base of users
- Be always available, which means no downtime
- Handle semi- and unstructured data
- Rapidly adapt to changing requirements with frequent updates and new features

Building and running these web, mobile, and IoT applications has created a new set of technology requirements. The new enterprise technology architecture needs to be far more agile than ever before, and requires an approach to real-time data management that can accommodate unprecedented levels of scale, speed, and data variability. Relational databases are unable to meet these new requirements, and enterprises are therefore turning to NoSQL database technology.

2 Overview of Couchbase Server

2.1 Introduction

Couchbase Server is a scale-out NoSQL database. The core architecture is designed to simplify building modern applications with a flexible data model, powerful SQL-based query language, and a secure core database platform that provides high availability, scalability, and performance.

Couchbase Server consists of a single package that is installed on all nodes within a cluster. Through the SDKs (also known as client libraries), developers can write applications in the language of their choice (Java, Node.js, .NET or others) and connect to a Couchbase Server cluster to perform read and write operations and queries across key-value records and/or JSON documents with low latencies (sub-millisecond) at high throughput rates (millions of operations per second).

To understand Couchbase's runtime behavior it is important to understand the high-level Server Architecture, Data Model, Client Connectivity, Management infrastructure and tooling, and Security.

2.2 Database Cluster Architecture

The basic database architecture consists of one or more Couchbase Servers (also called nodes) organized as a cluster. Each node contains a set of configurable services including a Managed Cache, Storage, Cluster Management as well as Data, Index, and Query Services. This scale-out architecture is one of the core aspects of Couchbase. It enables customers to grow the data management capacity and throughput of the system by simply adding more nodes, unlike more traditional RDBMS systems which require purchasing larger and larger servers in order to grow the capacity of the system.

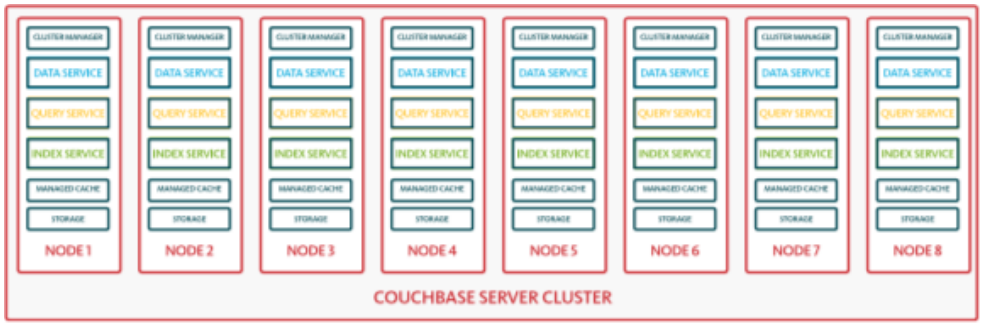

Figure 1 - Coucgbase Server Cluster

A **node** is a single Couchbase Server instance running on a physical server, virtual machine, or a container. All nodes are identical in the sense that they consist of the same components and services and provide the same interfaces. However, each node can be individually configured to maximize the available hardware and scalability of the system. This configurability allows customers to de ne cluster topologies that match their requirements, rather than having the database vendor decide, a priori, the kind of cluster topology that is best for all of their users.

A **cluster** is a collection of nodes that are accessed and managed as a single group. Each node is an equal partner in orchestrating the cluster to provide operational information and management of the cluster, including membership and the health of nodes. Clusters can be expanded by adding new nodes or shrunk by removing nodes, during which time the system remains online, and read and write operations continue uninterrupted. The "Cluster Manager" is the main component that orchestrates the cluster level operations.

All nodes within a cluster include a set of common services, a local in-memory Cache, and persistent Storage.

2.2.1 Cache

Unlike other database platforms, Couchbase Server does not require a caching tier in front of it. Couchbase uses a memory-first architecture, ensuring that all data operations are performed through a configurable high-speed in-memory cache. Internally, Couchbase moves data to and from disk as needed, thereby automatically acting as both a read-through and a write-through cache, which facilitates an extremely high read/write rate. With Couchbase Server, application configuration and deployment is simplified as applications don't have to deal with complex cache coherency issues or varying performance capabilities across technologies.

2.2.2 Storage

Couchbase uses a highly efficient append-only storage system to persist data to files that are placed in the local file system. Append-only storage systems are a technique used by many modern database architectures to provide optimized I/O throughput, especially for write-intensive applications. A back- ground compaction process (automated or manually controlled) cleans up orphaned and fragmented space within the files caused by ongoing data mutations to the append-only file system. The compaction process is designed to minimize the impact to front end I/O operations. Although high performance storage systems

(such as SSDs) can be used on Couchbase Server nodes, they are not required.

Couchbase Server uses two optimized storage engines, Couchstore and ForestDB. Couchstore is used to store the data items and local indexes, while ForestDB is used to store the Secondary Indexes.

Each node also includes the Data, Indexing and Query configurable services which enable Couchbase's revolutionary Multi-Dimensional Scaling architecture (MDS). Via MDS, system architects can implement a cluster that reflects and supports the technical business requirements using the hardware that is currently available. MDS supports both symmetrical and asymmetrical scaling topologies as shown below.

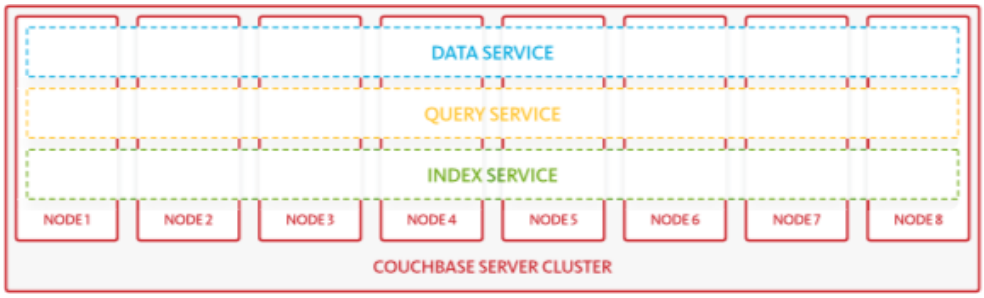

Figure 2 - Symmetrical Scaling Topology: all services run on every node, commonly used when all of the nodes have a similar hardware configuration and capacity. Nodes are "interchangeable".

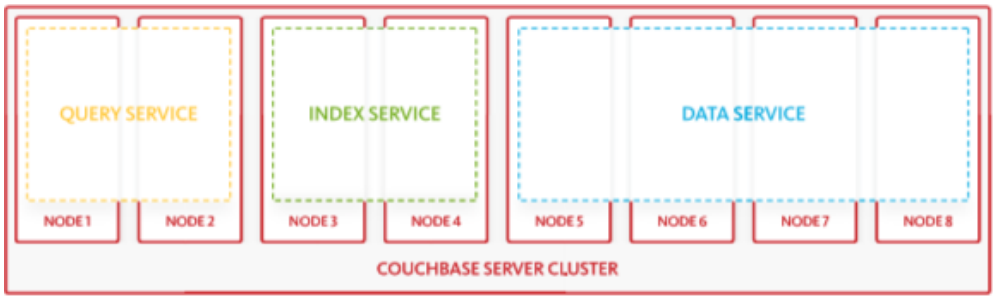

Figure 3 - Asymmetrical Scaling Topology: services run on specialized nodes, commonly used when leveraging specific hardware configuration and capacity (amount of memory, storage, or processors) for specific services. Nodes are more capacity- or service-specific.

By separating the core data management functions into these three services, Couchbase enables users to de ne the scalability characteristics of their system based on their actual technical business requirements. As those requirements change, Couchbase users can expand or shrink their cluster, adding data management resources to the precise function where they are needed most in order to address the changing Data, Indexing and Query Service requirements.

2.2.3 Data Service

The Data Service manages the data records (stored in binary or as JSON documents) via the Key-Value API (KV API) which provides high speed, direct read/write CRUD operations based on the primary key. The Data Service also manages the MapReduce View Indexes and MapReduce Spatial Indexes. These indexes enable application-specific JavaScript functions to de ne the index key and value that are derived from the data records. These local indexes are distributed and co-located with the data on each node, thereby providing low latency node-local access to the source data records. MapReduce indexes provide a powerful mechanism for building application-specific indexes, which can be especially useful for reporting and interactive analytical workloads. MapReduce View Indexes are accessed from both the View API and the N1QL query language. The MapReduce Spatial Indexes are accessed via the Spatial View API that can query data based on a bounding box (rectangle with coordinates).

The Data Service forms the base data management layer of Couchbase and is leveraged by the Indexing and Query Services.

2.2.4 Indexing Service

In addition to the aforementioned MapReduce Indexes, the Couchbase Server provides an indexing service to manage Global Secondary Indexes (GSIs). Global Secondary Indexes are aggregated and stored on a given Index Service node and can be independently partitioned by using a WHERE clause in the index definition. GSIs provide fast lookups and range scans which are especially useful for interactive applications that use secondary keys. Global Secondary Indexes are defined via the N1QL query language and are extremely flexible, enabling index creation on a specific subset of fields, on a subset of records, and on JSON arrays using built-in SQL functions. Additionally, Global Secondary Indexes can be declared to use in-memory storage optimization, enabling enhanced index performance.

Couchbase indexes are automatically maintained asynchronously from the core Data Service write operations, using high speed in-memory queues to forward mutations (writes) to the appropriate indexer using Couchbase's advanced Database Change Protocol (DCP). This asynchronous indexing architecture provides two fundamental benefits:

a) write operations within the Data Service can be completed extremely quickly, and
b) developers can create as many indexes as are needed by the application, no

longer having to pay the typical performance and latency penalty that is common in traditional RDBMS systems.

If needed, read operations can optionally request to wait for any pending index updates to be completed prior to performing the index scan if this is an application requirement for a given query.

2.2.5 Query Service

The Query Service takes a N1QL query and performs the necessary functions to retrieve, filter, and/or project the data in order to resolve the application request. Actual data and index lookups and scans may occur in the Data Service or in the Index Service, depending on the data and the indexes that are required to resolve the query. Applications and developers can use the N1QL REST API, the `cbq` shell, the `cbc` command line tool, or the Query Workbench to query records (JSON documents) using SQL-like syntax. With N1QL, developers can run ad hoc or prepared queries using standard SQL syntax (including WHERE clauses and aggregate functions) with JSON-specific extensions to filter, process, and format query output. N1QL queries will automatically leverage existing MapReduce View and GSI indexes to resolve GROUP BY, ORDER BY, and WHERE clauses.

The Query Service is where the N1QL query parser, planner, optimizer, and executor reside and it is the recipient of incoming N1QL queries. It provides the basis for how N1QL queries are executed across the Couchbase Server, leveraging the facilities that are available in the Data and Indexing Services.

2.2.6 Full Text Service

The Full Text Service (FTS) is available as part of the Couchbase 4.5 Developer Preview. FTS provides built-in management of full text indexes and queries, based on the Bleve open source project. FTS automatically maintains full text indexes based on the JSON documents stored in the Couchbase cluster. It also provides a REST-based API to perform rich Full Text Search queries over one or more attributes.

2.3 Disaster Recovery & High Availability

Replication is the primary internal mechanism used within the Couchbase Server in order to provide data that is highly available, even in the event of multiple

failures within a Couchbase cluster. Replication achieves this goal by creating up to three additional copies of the data on alternate nodes. When replication is enabled, Couchbase Server will create multiple copies of each record, placing them on different nodes. One copy will be designated as active, while the remaining copies will be designated as replicas. By default, read and write operations will be directed to the active copy of the data, although the application can optionally request that reads be directed to one of the replicas. For more information on how replication and the organization of the data on each Data Service node, please see the Data Organization section a little farther down.

Couchbase replication is Node and Rack-aware, ensuring that the additional copies of the data are placed on different Nodes/Racks than the active data. Couchbase replication uses the in-memory DCP protocol to asynchronously forward data mutations to the replicas. This ensures that mutations are streamed to the replicas in real time, while also avoiding additional latency on the node with the active data.

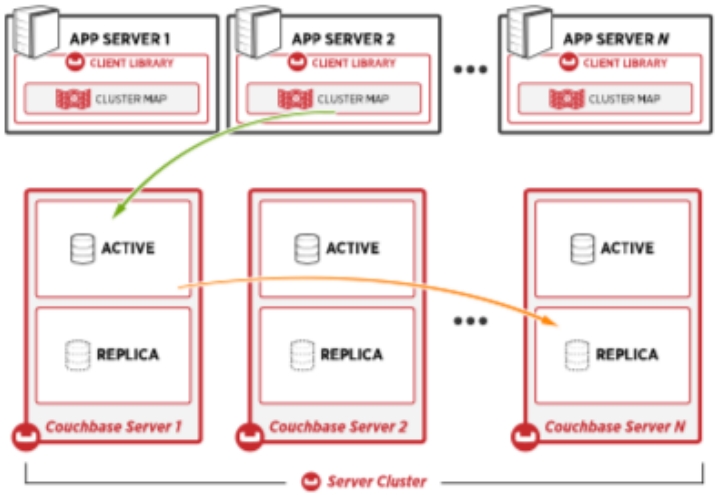

Figure 4 - Replication

2.3.1 Multiple Data Center and Streaming Data Support

Two of the big challenges in managing today's complex applications for the Digital Economy are a) the global distribution of data, often across multiple data centers, and b) the ability to stream data in real time to other services and application systems.

In addition to the replication that occurs within a Couchbase cluster, Couchbase DCP streams also enable bi-directional, high throughput replication between Couchbase clusters (which can be located in remote data centers) as well as connectors to external downstream systems (using Couchbase-provided or

application-specific connectors).

DCP streams enable:

- Cross Datacenter Replication (XDCR) providing geographical bi-directional distribution of data across Couchbase clusters
- Automatic backups with full or incremental datasnapshots
- Integration with down stream systems and services like Hadoop, Flume, Kafka, Spark, Storm and other third-party tools as well as external text search engines like Elasticsearch

3 Data Access

3.1 Introduction

There are some basic concepts to explore when discussing how Couchbase provides data access: a) how data is organized, b) how data is written, and c) how data is read.

3.2 Data Organization

The basic unit of data in a Couchbase Server is an **item** (also called a *document*). An item is a key-value pair where each stored value is identified by a unique key. Values for an item can be anything, including a single bit, a binary value, a decimal measurement or a JSON document. Storing data as a JSON document allows the Couchbase Server to provide extended features such as indexing and querying.

Items are grouped into **buckets**. The key for any given item must be unique within the bucket to which it belongs. Buckets can contain a mix of key-value pairs, binary records, and JSON documents. There is no implied pre-defined structure or schema within a bucket. Buckets are automatically partitioned (using a built-in hash function on the item's unique key) into 1024 Virtual Buckets (*vBuckets*) and evenly distributed across the Couchbase cluster (within the Data Service). This allows Couchbase to provide a very flexible, as well as highly scalable data management system.

If you need to store something with a different structure, just add it to the bucket. If you need to manage more data or provide more read/ write throughput, just add more nodes to the cluster. Additionally, buckets provide the resource management facility for the group of data that they contain, including configuration parameters to control a) cache and I/O management, b) authentication, c) indexing and views, and d) replication and Cross-Datacenter Replication.

When replication has been enabled, the Couchbase Server will automatically create both active and replica vBuckets, where the replica vBuckets will contain the additional copies of the data. The replica vBuckets will be placed on different Data Service nodes than the active vBuckets, thus ensuring optimal data load balancing and high availability of the data as additional copies of the data are still accessible in the event of a node failure.

This is different from relational databases which store records (rows) in tables grouped into database instances. Tables have a strict pre-defined schema (set of columns) and the data being stored must conform to that schema. Furthermore, relational databases

are centrally managed on a given physical server or virtual machine. If you need to store something with a different structure, you need to modify the schema first (usually requiring coordination with a Database Administrator). If you need to manage more data or provide more read/write throughput, you need to buy a bigger server (usually requiring a lengthy procurement and installation process).

3.3 Writing Data to the Couchbase Server

Couchbase Server uses the runtime behaviors that we have been discussing to provide high throughput, low latency, reliable write operations. Specifically, it brings all of this together thusly:

Writes are sent to the high-speed memory-first cache (directly on the node that contains the appropriate vBucket for the item being written).

The new or changed record is placed on several asynchronous high-speed in-memory queues, including the Indexer Queue, the Persistent I/O Queue, and the Replication Queue, as well as any additional external Queues such as XDCR, Kafka, Spark, etc., leveraging DCP.

The application can optionally specify stronger durability assurances for a given write operation, re- questing that the write be replicated to the memory-first cache on another node(s) or persisted to disk if required by the application logic.

Write operations are both highly predictable and extremely low latency because they occur within the in-memory cache and they don't have to wait for downstream asynchronous actions to occur. Write operations are also highly scalable because they are distributed to the different nodes hosting the appropriate vBuckets. Therefore, adding throughput is as simple as adding more nodes to the cluster.

3.4 Reading Data from the Couchbase Server

Couchbase Server fundamentally supports two kinds of reads: key-based and index-based. When retrieving records based on the unique key for the item, the Couchbase Server provides full read consistency (sometimes referred to as read-your-own-write semantics) by ensuring direct access to the active vBucket for that key. The application can optionally request access to a replica vBucket (also known as replica reads) if the application logic allows it.

When retrieving data based on an indexed key, the Couchbase Server searches the

appropriate index for any matching values and will then fetch the original primary record unless the query can be resolved within the index. Because indexes are asynchronously maintained, they may be slightly out of date. The application can optionally request that the index be brought up to date before performing the index scan if the application logic requires it.

A simplified example of the read and write behavior can be found below.

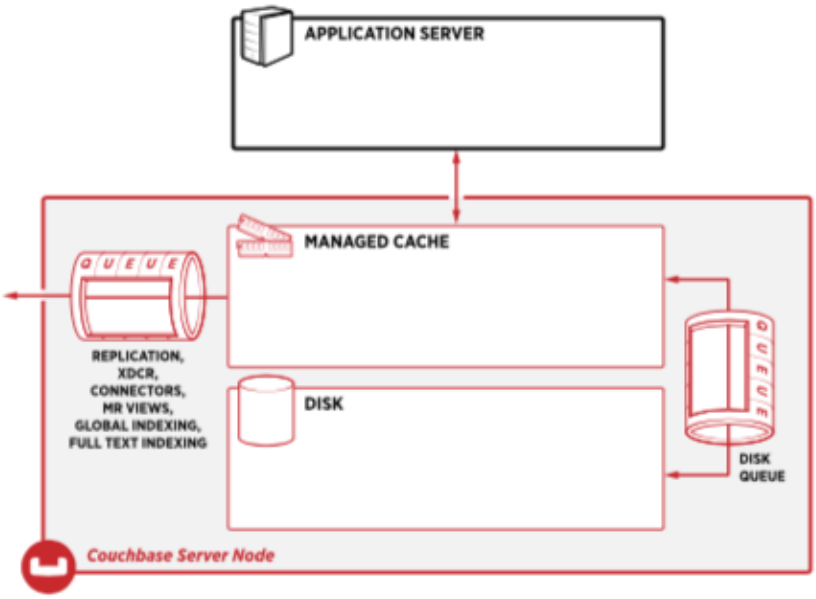

Figure 5 - Write Operations

4 Connectivity Architecture

4.1 Introduction

Couchbase Server is a fully distributed database, thus connection management and effcient and secure communication are key components of the architecture. There are fundamentally four aspects of connectivity and communications that occur in the Couchbase Server:

4.2 Client to Cluster Communication

Client applications communicate with Couchbase Server on a set of ports and protocols appropriate to the API that they are using (Admin, View, N1QL or KV API).

Each access point supports both clear text and encrypted communications. When the client first connects to the cluster, it authenticates and gets back a copy of the cluster map indicating which nodes are hosting which vBuckets.

Once the client SDK has the cluster map, it has the information it needs in order to authenticate and establish communications with the appropriate node and vBucket(s) based on the API being used by the client.

4.3 Node to Node Communication

Nodes of the cluster communicate with each other to replicate data, maintain indexes, check the health of other nodes, communicate changes to the configuration of the cluster and more.

Node to node communication is optimized for high efficiency operations and may not go through all the client-to-cluster connectivity phases (Authentication, Discovery, and Service connection).

4.4 Cluster to Cluster Communication

Couchbase Server clusters can communicate with each other uni- or bi-directionally using the Cross-Datacenter Replication (XDCR) capability.

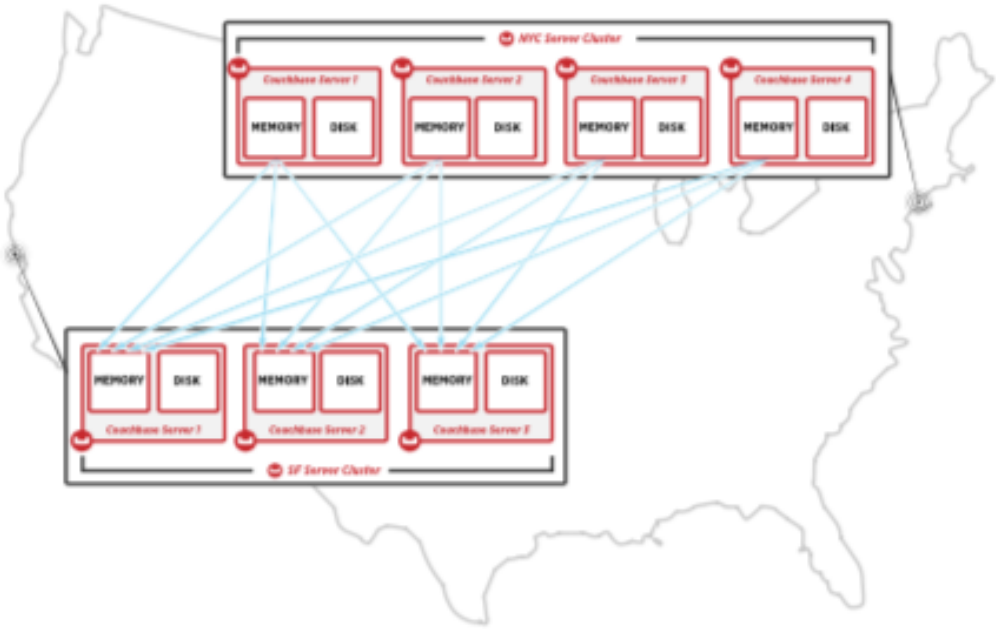

Figure 6 - Cross Datacenter Replication (XDCR)

4.5 External Connector Communication

Couchbase Server also communicates with external products through connectors. There are a number of connectors that are built and supported by Couchbase (including Spark, Kafka, Elasticsearch, Solr, etc.) and many that are built by the community or other companies (ODBC driver, JDBC driver, Flume, Storm, Nagios, etc.). External connectors are typically built using existing client SDKs or admin APIs, or feed directly from the internal APIs such as XDCR or the DCP API.

4.5.1 Administration Communication

Couchbase Server provides an Administrative Web Console, REST API, and a set of Command Line Interface (CLI) tools for secure configuration, management, and monitoring of your Couchbase Server installation. The REST API and CLI tools can be used in conjunction with other third-party tools and used with your custom management and administration scripts to support different operations.

4.5.2 Couchbase Mobile Communication

Couchbase Mobile includes a Sync Gateway which enables secure, scalable, efficient data exchange between a Couchbase Server cluster and mobile devices running Couchbase Lite.

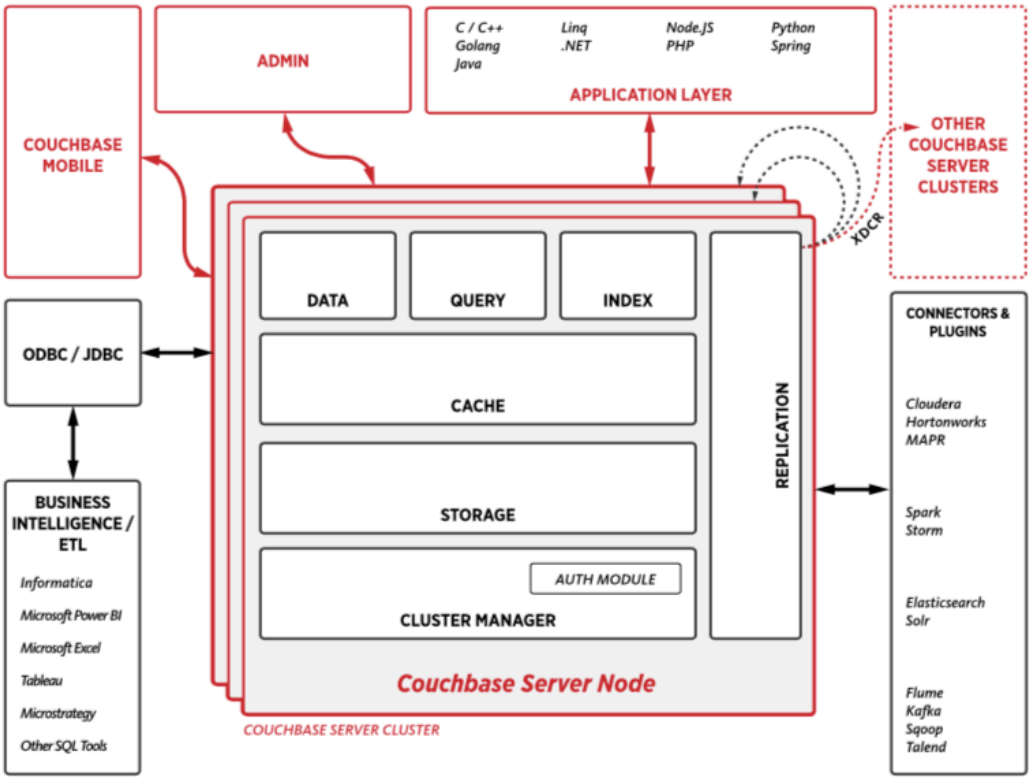

Figure 7 - Connectivity Architecture

5 Cluster Management

5.1 Introduction

Couchbase Server provides a comprehensive Management UI to visualize, monitor, and manage the individual nodes and the cluster as well as overall cluster status and statistics. Additionally, the REST- based Admin API allows developers to directly query and integrate Couchbase administration and management functionality into their existing tools.

The Couchbase Cluster Manager runs on all of the nodes of the cluster and orchestrates cluster-wide operations, including:

a) cluster topology and node membership,
b) data placement (distributing vBuckets to nodes),
c) topology map change communication,
d) centralized management, statistics, and logging,
e) connection authentication.

Cluster management operations often leverage common underlying runtime behaviors available in the Couchbase Server, including rebalancing and failover.

5.2 Rebalance

The topology of a cluster changes over time, as nodes are added or removed. As the number of nodes changes, cluster resources may need to be redistributed to ensure optimal reliability, performance, and load balancing.

The rebalance operation is used to redistribute the load and adapt to changes in the cluster topology. A rebalance also updates the cluster map as needed and forwards the updated map to all clients. The Cluster Manager coordinates the movement and handoff data and services during a rebalance operation. A rebalance occurs completely online, with minimal impact to the ongoing workload.

For example, adding or removing Data Service nodes can trigger a rebalance operation which may re- quire the incremental movement of vBuckets from one node to another. By moving vBuckets onto or off nodes, these nodes become responsible for more or less data and begin handling more or less query traffic from the applications.

5.3 Failover

Given that Couchbase Server distributes data, queries, and services across the nodes within the cluster, failover is the natural component of the overall architecture.

Failover is the process that automatically re-directs traffic away from one or more failing nodes to the remaining healthy nodes.

Failover can be done automatically by the Couchbase cluster, can be done manually by the administrator or by an external script. Failover is designed to ensure high availability of data and services even in the event of multiple failures within a cluster.

6 Security

6.1 Introduction

Security is a fundamental consideration in database management systems and NoSQL databases are no exception. However, security is not black and white – it is a continuum of features and functionality that are available within a database system, as new features are introduced as the product evolves. This is especially true for NoSQL databases, since security was not the first technical concern that was being addressed – performance, throughput, volume, and scalability were the first priority for NoSQL databases. As NoSQL databases have become more and more crucial to applications that power the Digital Economy, security has acquired a growing importance and focus.

At a high level, security includes the following capabilities: Authentication, Authorization, Encryption and Auditing. In order to comply with important security requirements like PCI-DSS and HIPAA, Couch- base Server provides a growing set of functionalities in all four security areas.

6.2 Authentication

Couchbase Server supports multiple authentication mechanisms (SASL, non-SASL, and CRAM) for Administrators and applications accessing a Couchbase cluster.

Administrator authentication via the web console can also be integrated with existing LDAP authentication services.

6.3 Authorization

Application read/write access is defined at the bucket level, enabling the ability to have data sets with varying degrees of authorized access.

Couchbase Server supports role-based authorization for Administrators, allowing the system to restrict what aspects of the cluster a given administrator can view and/or modify.

6.4 Encryption

Couchbase Server provides two levels of encryption capabilities:

- **Data in motion** – Couchbase Server supports end-to-end Secure Socket Layer (SSL) traffic both from applications to clusters, between nodes in a cluster and between clusters (XDCR). This encryption covers both data packets and administration traffic.

- **Data at rest** – Couchbase works with LUKS-based disk encryption on Linux, Bitlocker Drive Encryption on Windows Server 2008 and 2012, and with Vormetric Data Security platform, which provides disk, file-level, and application-level encryption.

7 Conclusion

Applications and technologies that power the Digital Economy have a unique set of requirements. If experience has taught us anything about the technical requirements for technologies in the Digital Economy, it is that

a) "*One size never fits all*" – flexibility across the technology spectrum is crucial because each application is different and the requirements evolve rapidly;

b) "*If it doesn't perform at scale, it's not going to work*" – a corollary to "*Go big or go home*," that is driven by the sheer volume of data and throughput that is growing exponentially.

Not surprisingly, the database management technology that these applications use also must exhibit these characteristics.

Couchbase Server employs an architecture that addresses both of these challenges. Support for key-value as well as JSON document data storage, the variety of indexing and query options, support for multiple APIs, programming models and languages, the availability of Couchbase and community supported connectors to external tools all demonstrate an innate understanding of the need for developer flexibility that is part of the "One size never fits all" principle.

Architecturally, Couchbase Server is designed with scalability and performance at its core: with features like memory-first caching, asynchronous downstream services (I/O, indexing, and replication), built-in load balancing, as well as Multi-Dimensional Scaling, Couchbase Server is designed to address the data management demands of the Digital Economy.