Couchbase Essentials

Harness the power of Couchbase to build flexible and scalable applications

John Zablocki
In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 1 ‘Getting Comfortable with Couchbase’
- A synopsis of the book’s content
- More information on Couchbase Essentials

About the Author

**John Zablocki** is the director of information technology at EF High School Exchange Year in Cambridge, Massachusetts, USA. Previously, he worked at Couchbase Inc. as a developer advocate, maintaining the .NET SDK and delivering training to customers and users alike. John is the author of O'Reilly's Orchard CMS. He is a frequent presenter at community events and has run Code Camps and user groups. He holds a Master's degree in computer science from Rensselaer at Hartford, where he became an enthusiast of open source technology. John can be approached online at [http://about.me/johnzablocki](http://about.me/johnzablocki) and around Cambridge with his daughter, Mary Katherine; his dog, Lady; and his Fender Jaguar.
Couchbase Essentials

Not too long ago, I was fortunate enough to have worked for Couchbase Inc. with the developer solutions team. In my role as a developer advocate, I had two primary responsibilities: maintaining the Couchbase .NET SDK and training Couchbase users on how to develop for Couchbase Server.

During my tenure on the SDK team, I worked with hundreds of developers around the world who were using Couchbase Server for a wide variety of solutions. Some were using Couchbase Server for its distributed caching abilities, while others needed a model that could support near-real-time analytics over a flexible schema. I was always impressed with Couchbase Server's ability to handle such a vast number of development scenarios.

Through the countless meetings I had with the development community and customers alike, it became clear to me that NoSQL is far from a technology fad. Along with cloud computing and mobile services, NoSQL has become a part of the fabric from which modern applications are woven.

As with relational databases before them, NoSQL databases such as Couchbase Server are quickly nearing the "required knowledge" status for application developers. Modern applications that need to reach a massive scale or require greater data model flexibility have found success with non-relational systems.

It is a tremendous opportunity to be able to share my experience at Couchbase with you, the reader. This is an exciting technology, and this book contains the tools you need to get started with Couchbase development.

What This Book Covers

Chapter 1, Getting Comfortable with Couchbase, introduces Couchbase Server and provides details on obtaining and installing it. It also walks you through setting up Couchbase Server for the first time.

Chapter 2, Using Couchbase CRUD Operations, provides an overview of basic Couchbase Server operations. Basic SDK usage is demonstrated while exploring the various CRUD API methods.

Chapter 3, Creating Secondary Indexes with Views, explains in detail the programming model of MapReduce. After this exploration, the basics of using MapReduce within Couchbase Server are explored.

Chapter 4, Advanced Views, explores common view patterns for Couchbase development, following on the previous chapter's discussion of MapReduce.
Chapter 5, *Introducing N1QL*, introduces the prerelease Couchbase query language, N1QL.

Chapter 6, *Designing a Schema-less Data Model*, discusses many of the design options that must be considered when building Couchbase Server applications. Both key/value and document schemas are covered.

Chapter 7, *Creating a To-do App with Couchbase*, provides an overview on how to convert Couchbase Server to a basic to-do application.

Appendix, *Couchbase SDKs*, contains a brief introduction to the official Couchbase SDKs, including installation and basic usage.
Couchbase Server has quickly emerged as one of the leading NoSQL databases. Known for powering apps and sites such as Viber, PayPal, LinkedIn, and eBay, Couchbase Server easily serves up terabytes to petabytes of data. Whether used as a distributed cache or a document database, Couchbase Server has become a significant contributor to the growth of the Internet as a whole.

Long before the term NoSQL started to grace the pages of blogs, tech journals, and investor balance sheets, a technology called Memcached was providing life support for relational databases. As these systems attempted to reach the scale demanded by modern, Internet-based applications, it was clear that Memcached could help. Still widely used today, Memcached is a distributed key/value store used to provide a caching layer for applications.

Some of the developers on the open source Memcached project saw the potential to take the system beyond a simple cache. They introduced new features such as a binary protocol, better cluster management, and most importantly, persistence. This new and durable offshoot of Memcached became known as Membase. A company of the same name was formed to support the project (it is still open source) and provide customers with support in their production environments.

Membase quickly gained popularity with developers who needed massive scalability. From start-ups to stalwarts, this new database was becoming one of the disruptive technologies that would forever change the way applications store data. Around the same time, developers were starting to demand more flexibility from their databases. A seemingly infinite number of web applications were built using Object Relational Mappers (ORM) such as ActiveRecord, Hibernate, and SQLAlchemy.
ORMs attempt to simplify the object-to-relational mapping problems often associated with working with a highly normalized database. The basic problem is that the relational model does not always look like an object-oriented model. ORMs hide the underlying data model from the application layer, often by way of a significant amount of configuration. ORMs also provide relational databases with a new lifeline.

One open source project that attempted to solve the object-to-relational mapping problem by doing away with the relational side of things was **CouchDB**. The developers of CouchDB built a database that, in their own words, was for developers and by developers. Tables, columns, and rows were replaced by documents stored as **JSON**. The net result was a system that stored data in structures similar to those found in the application layer.

Eventually, as both Membase and CouchDB matured, the developers of both systems came together for what is one of the most important chocolate-meets-peanut-butter moments in database history. The extremely scalable and reliable Membase would eventually be married to the ever-flexible and developer-friendly CouchDB. Each database would take part of its maiden name in the merger, which was called **Couchbase**.

Today, Couchbase is responsible for developing and supporting **Couchbase Server**. The combined products still remain open source but are no longer tied to their parent projects. While many of the features of Couchbase were inspired by CouchDB and Memcached, the code is anything but a "copy-and-paste" from the parent projects. Make no mistake about it! Couchbase is a standalone product optimized to be better than two otherwise great projects.

**The NoSQL landscape**

In the crowded market of NoSQL databases, Couchbase Server is one of the dominant players. Its performance sets the bar high for its competitors. The rich feature set of Couchbase Server also sets a new standard for what is expected from NoSQL databases. As NoSQL is still a nascent field, Couchbase Server seems destined to influence its future.

All relational databases tend to be the same animal. Whether you're using SQL Server or MySQL, you could expect to find the same basic set of features. You store your data in rows with strictly defined columns inside a table. You then modify your data using SQL's **INSERT** and **UPDATE** statements. You retrieve your data using SQL queries. In contrast, NoSQL databases vary wildly from one system to the next. However, there are some features you would expect to find across various NoSQL taxonomies.
Perhaps the most common feature in NoSQL databases is the lack of an imposed structure on your data. While in practice, structures tend to be defined by your application layer, it is permissible that your NoSQL records are like snowflakes—no two records are the same. This flexibility has made NoSQL databases popular with developers, who no longer have to work within the constraints of a relational schema and ORMs.

Another feature (or lack of a feature) that you could expect to find in NoSQL databases is the lack of explicit ACID transactions. In other words, you won't be able to wrap a series of insertions or updates within a transaction. However, this does not mean that ACID properties are not supported in NoSQL databases.

**Atomicity** is widely supported in NoSQL databases. Partial writes are not possible. Either an entire record is written or nothing is written. **Consistency** in NoSQL ranges from eventual (delayed) consistency to strict consistency. **Isolation** is implicit, which means that a read will never return values from an update in progress. Like consistency, **durability** varies within NoSQL databases and is generally tunable.

The importance of full ACID compliance in NoSQL is somewhat diminished. Often, the need for transactions is dictated by the relational model, where related data is stored in one or more tables. In NoSQL databases, it is common to write related data to a single structure or record. In other words, a single NoSQL update or insertion might require several updates or inserts in the relational world.

This modeling difference also reduces the need for features such as joins or strict referential integrity. When records are stored in a denormalized fashion, a single query may bring back the required object graph.

Of course, it's likely that you will still need to make use of relational concepts in your NoSQL data model. Full denormalization is often impractical in NoSQL. In these cases, the applications that consume the data face an increased burden of being responsible for handling the details that a relational database typically would have dealt with.

Beyond these basic features, NoSQL systems tend to become more and more disparate. Instead, you will be more likely to find similar features between databases in the same NoSQL category. For example, CouchDB and MongoDB are both document stores. While they are fundamentally very different databases, they are more similar to each other than either of them is to a graph database such as Neo4j or a column database such as Cassandra. In the next section, I'll discuss the different categories of NoSQL databases and describe how Couchbase fits into the big picture.
**NoSQL taxonomies**

There are many different categories of NoSQL database. A broad definition of NoSQL might consider everything from XML databases to cloud-based BLOB storage as parts of the NoSQL landscape. However, in practice only a few NoSQL databases are widely used, with the vast majority of developer mind share belonging to only two categories, key/value and document stores.

Key/value stores are popular because of their simplicity. Records are stored and retrieved via a key much like programmers use hash tables or dictionary structures to store data in the memory. These systems tend to be highly performant.

Document stores are arguably the most popular of NoSQL databases, driven primarily by the flexibility they offer. Documents are typically stored in a JSON or JSON-like structure. JSON, being a notation for describing object graphs, is a natural fit for object-oriented applications.

While nearly all popular NoSQL databases fall into one category or another, Couchbase is both a key/value and a document store. Records are written to and read from Couchbase using a key/value API. When those records are stored as JSON documents, Couchbase provides document indexing, allowing queries on arbitrary properties in the document structure.

Importantly, Couchbase does not sacrifice features to achieve its duplicity. Though it might seem that such a hybrid system would necessarily be lacking in either its key/value or document capabilities, Couchbase feels complete. As a key/value store, Couchbase offers a rich API based on its Membase heritage. As a document store, Couchbase supports the most important features from its "pure document relative" — CouchDB.

Two data storage models also provide developers with a great deal of flexibility. Applications may be optimized using different approaches for different features; for example, a social game might make use of Couchbase's key/value interface to achieve scaling when collecting or serving vast amounts of data. That same application could then use the document interface to retrieve aggregate statistics on players.

**Installing Couchbase**

There are two editions of Couchbase available for download — Community and Enterprise. While both editions are largely the same, there are two key differences. The Community Edition is free to use for development and in your production systems. However, there is no guarantee that patches (critical or small) will be made to this build in a timely manner. This edition is intended primarily for development, or for those developers who are okay with relying on free support (that is, the Couchbase forums).
The Couchbase Server Enterprise Edition requires the acceptance of an End User License Agreement (EULA), with the user agreeing to install it on no more than two production nodes. Use of more than two nodes requires the purchase of a support license. There are a variety of support levels available. Enterprise Edition also receives priority patches and new features ahead of the Community Edition. It is recommended for use in mission-critical systems.

For the examples in this book, there are no meaningful differences between the two editions. As such, we’ll use the Community Edition. How you install Couchbase Server will depend on your operating system. Once it is installed, maintaining and developing the server is generally the same experience on both Windows and Linux.

To get started, open your browser and go to http://www.couchbase.com/download. Here, you’ll find the latest binaries. At the time of writing this book, the latest Enterprise Edition is 2.5.1 and the latest Community Edition is 2.2.0.

**Installing Couchbase on Linux**

The Couchbase team maintains 32-bit and 64-bit builds for Ubuntu, CentOS, and Red Hat Linux. After downloading the package on Ubuntu, run the following command to install it:

```
sudo dpkg -i couchbase-server-enterprise_2.2.0_x86_64.deb
```

For CentOS or Red Hat installation, run this command:

```
sudo rpm --install couchbase-server-enterprise_2.2.0_x86_64.rpm
```

**Installing Couchbase on Windows**

For Windows 7, Windows 8, and Windows Server, there is a setup program. Simply download the installer, run the .exe file, and follow the steps of the wizard. When you install Couchbase on a Windows machine, you’ll see the following prompt:

![Ephemeral port warning](image_url)
By default, the highest port number that TCP may assign to an application requesting a user port is 5000 on Windows systems. This value is generally sufficient for development purposes, but in production deployments, Couchbase requires a greater number. For the purpose of this book, leaving your default settings as they are is safe.

### Installing Couchbase on Mac OS X

Finally, if you're developing on a Mac, there is a development-only build available. After downloading the Mac package, double-click to unzip it. Then drag the contents into your Applications directory. For obvious reasons, the Mac release is for development only.

### Ports

Couchbase Server is constructed using a series of components, each requiring access to a different port. It's common to encounter errors when trying to use Couchbase for the first time, due to blocked ports. You're more likely to have fewer port restrictions on your development machine than on your production servers, but it's still important to make sure you have at least ports 8091, 8092, 11210, and 11211 open. Running a cluster requires more port access, but for development, you'll need to have the web admin accessible (8091) and the API and client endpoint ports open (8092, 11210, and 11211).

### Running Couchbase for the first time

One feature that really sets Couchbase apart from the other NoSQL databases is its administrative interface. When you install Couchbase, you also get this powerful web app to manage your server. Moreover, the admin tool is simply a wrapper over a RESTful management API supported by the server. In other words, any action you can perform with the admin GUI can also be performed via your favorite DevOps tools.
You can get to the Couchbase Server web admin by opening your browser and going to http://localhost:8091. If you’ve just completed installing Couchbase, there may be a brief delay between the startup of the server and the startup of the admin. Refresh a couple of times, and you should see something like this:

Couchbase Console for new install
The Couchbase Web Console provides a setup utility to get your cluster up and running. Click on the blue Setup button to begin. The following screenshot shows the configuration screen:

Configuring the server

The options found in step 1 of the configuration screen are fairly straightforward. The first two fields are the paths where Couchbase will store data and indexes. The server hostname uniquely identifies a node in a cluster. I'll discuss clusters in more detail later in this chapter. For now, you can think of a cluster as a collection of Couchbase server instances, or nodes, with the same buckets.

For development, it's generally useful to set this to 127.0.0.1. Basically, you want to ensure that whatever hostname you choose, it is not subject to change, as could be the case when running in the cloud or attaching the cluster to a network outside of your home or office.
The final option is whether to start a new cluster or connect to an existing cluster. In our case, we'll obviously be starting a new cluster. In a production environment, you'd want to maximize the amount of RAM available to your node. For development purposes, you are free to choose a lesser amount. The important thing to note here is that the amount of RAM you allocate will be required by each node in your cluster. If you click on the **Join a cluster now** option, you'll be asked to provide the address of a node in the cluster and the cluster credentials, as shown here:

![Connecting to a new cluster](image)

Click on **Next** to be brought to step 2, where you'll be asked whether you want to install one of the two available sample buckets. We'll dig into buckets in the next step, so for now, just check the **beer-sample** bucket. That's the sample data source we'll use as we explore the development APIs. The following screen shows the sample buckets to be selected:

![Sample buckets](image)
In step 3, you're prompted to configure the default bucket for the new cluster. Couchbase buckets are loosely analogous to databases in relational systems. If you've used MySQL, SQL Server, or any other relational database server, you know that you must create an object called a database in which you'll create your tables and other database objects. Similarly, with Couchbase Server, a bucket is a container for the documents and indexes you'll store.

You must have at least one bucket on your cluster, and during the setup you are required to create a bucket named default. As you can see in the next screenshot, you are not allowed to change the name of this first bucket. You do, however, have other decisions to make about the default bucket.

Bucket configuration
Couchbase, a being of Membase and therefore of Memcached lineage, fully supports the Memcached binary protocol. What this means is that Couchbase Server can be used as a stand-in replacement for a Memcached cluster. If you're currently using Memcached as a distributed cache for your application, you would be able to replace it with Couchbase Server and a Memcached bucket.

If you set the bucket type to Memcached, your bucket won't be persistent, and it won't be able to take advantage of the document capabilities that Couchbase provides. Even for use as a distributed cache, a Couchbase bucket is almost always the right choice. Couchbase disk writes are performed asynchronously, and it's unlikely that your application will be impeded by I/O problems. We'll stick to Couchbase buckets for this text, but it's important to understand the difference between these two bucket types.

Because Couchbase relies heavily on RAM to achieve its blazingly fast performance, it's important to allocate as much RAM as possible to your bucket. I'll discuss Couchbase Server's architecture towards the end of this chapter, but for now, know that more RAM generally means better performance. For development purposes, feel free to allocate the minimum amount of RAM required for each node (for instance, 100 MB).

Couchbase Server supports replication within your cluster. When you set up a bucket, you may choose to replicate the data to up to three other nodes. Replication will also be discussed at the end of this chapter. Since we are using a single-node cluster, uncheck the Enable option.

Couchbase allows you to specify the number of reader/writer workers to allocate for a bucket. This setting exists to allow administrators to optimize disk I/O. We'll leave the default value, 3, in place. If you enable Flush on your buckets, you'll have the ability to remove all documents from a bucket with a single command. This action is like truncating all the tables in your relational database, so obviously it should be set only when absolutely necessary.

In step 4, the wizard simply asks whether you wish to receive update notifications, and allows you to sign up for Couchbase's community update e-mails. Neither choice will affect the setup. The fifth and final step is to set up a username and password for cluster administration.
Getting Comfortable with Couchbase

After completing the wizard, you'll be presented with a Cluster Overview page. When this page first loads, it's possible that you'll see a brief notification that the node is down while the bucket is activated. You're also likely to see a notification that the sample bucket is being loaded. Once ready, your cluster should show as healthy with active buckets, as in shown in the following screenshot:

![Cluster Overview](image-url)
Exploring the Couchbase Console

At this point, we'll take a quick tour of the other tabs found in the Couchbase Console, starting with the Server Nodes tab. When you click on this view first, you'll see a list of all active servers in your cluster. In our case, we have only one active server. For each of the nodes, you'll also see its status (Up or Down) and some vital stats such as RAM and CPU usage. Note that in the following screenshot, I clicked on the arrow next to the node name to reveal additional details about the node.

You'll also notice a button labeled Pending Rebalance next to the active servers. Nodes that appear in this list are those that are part of the cluster, but will not be fully active until they've been rebalanced. I'll discuss rebalancing at the end of this chapter. You'll also see options to trigger a rebalance and add another node to the cluster.
The Data Buckets tab lists all the buckets for a cluster. At this point, you should see both the beer-sample and default buckets. I expanded the beer-sample bucket in the following screenshot to reveal more detailed information about the bucket. You'll see options for viewing bucket documents and views. You may edit your existing buckets or create new buckets. You'll also see important stats such as item count and RAM and disk usage. We'll explore these options in more detail in the rest of the book.

Chapter 3, Creating Secondary Indexes with Views, and Chapter 4, Advanced Views, will cover Views in detail, so for now we'll skip over this tab. Cross-data-center replication, or XDCR, allows you to create unidirectional or bidirectional replications of two clusters. XDCR is beyond the scope of this book, but know that you can manage it here. The Log tab shows the running server log. Some messages are only for information, while some expose failures on your server. On the Settings tab, you can perform a variety of tasks from adding a sample bucket to activating auto-failover.
Couchbase architecture

Before we move on to developing with Couchbase, it’s useful to understand the general Couchbase architecture. While coding against a single-node cluster should generally be no different than coding against a 10-node cluster, supporting a production application does require deeper understanding of what could go wrong, as your application needs to scale out. In the following sections, I’ll describe in more detail some of the concepts we’ve already seen, and the basics of how a Couchbase cluster operates.

Couchbase clusters

Fundamental to all Couchbase deployments is the notion of a cluster. This is a common term in the NoSQL world and generally refers to a collection of nodes performing operations on a data store in tandem. However, how nodes in a cluster behave varies significantly across NoSQL products. In some systems, all nodes are peers, with no differences. In others, clusters are set up in master-slave configurations.

In a Couchbase cluster, nodes are interchangeable. Each node contains a cluster manager, which is responsible for knowing the status of other nodes in the cluster, and for allowing other nodes to know its status. As each node has its own cluster manager component, this allows Couchbase Server to scale out linearly with no single point of failure.

Replication

One of the most important tasks of the cluster manager is to ensure that all of the data is available to clients. Couchbase Server replication works by making one node the master node for a given document, while up to three slave nodes maintain a replica of that document. In case the cluster manager detects a node failure, it is responsible for promoting replicas to the primary node.

Balancing and rebalancing

Sharding is the notion of distributing data evenly across the nodes of a cluster. In most sharded systems, the admin is responsible for picking a shard key to be used for data distribution. For example, a Users table might be sharded on a Username field. If the shard key turns out to be poorly distributed (imagine 30 percent of users having usernames starting with T), then the nodes will not be well balanced.
Couchbase, in contrast, is auto-sharded and guarantees balance. Recall that Couchbase documents are stored using a key/value approach. Though the user supplies the key, Couchbase SDKs use a strong and cryptographic hash on each key to guarantee that keys will be evenly distributed across a cluster. This hashing action considers the topology of the cluster, which means that whether there are 2 or 20 nodes, the keys will still be balanced.

Even though the SDKs and the server work together to ensure proper sharding, in case a node (or nodes) goes offline, that balance will temporarily be broken. This is because replicas are promoted. As nodes are added or removed from a cluster, the cluster manager will work to rebalance the data across the nodes. A newly added node may not be ready to fully join the cluster until a rebalance has been performed. As alluded to earlier, this task may be done using the Couchbase Console.

**Couchbase SDKs**

We’ll explore the Couchbase SDK and relevant APIs in detail over the next few chapters. But to complete our discussion on balancing and rebalancing, it’s useful to understand the process from client to cluster. When an SDK is initialized in a client application, it makes a persistent connection to the cluster over a RESTful API. This API broadcasts a JSON message containing the cluster’s topology. As nodes are added or removed, the cluster sends a new message with an updated topology. This behavior sets Couchbase apart from other databases, whether relational or nonrelational. Most database systems have a central point of communication that is responsible for client communications. Couchbase owes some of its massive throughput to its smart clients. Eliminating the bottleneck of a man-in-the-middle allows performance levels to reach a massive scale. On a cluster with only four nodes, Couchbase is capable of achieving nearly 1 million operations per second.

Returning to the idea of balancing data across nodes, there’s an additional detail that I didn't mention. The cluster maintains an abstraction known as **vBuckets**, which are used to direct a key to the correct server. Rather than mapping a key directly to a node, Couchbase SDKs map the key to one of the vBuckets. The endpoint for a vBucket is provided to the client as part of its topology message from the cluster. Regardless of the number of nodes, the number of vBuckets remains the same. The keys always hash to the same vBucket, even if the cluster changes the endpoint of the vBucket.
While you’ll generally not need to worry about the existence of vBuckets, it is important to understand what happens on the client as the cluster changes its topology. The client maintains a map of vBuckets to the nodes. If that map changes due to a node failure, brief client failures may appear while the map is updated.

The only case where you’re likely to care about vBuckets is if you are developing an application using Mac OS X. On this platform, Couchbase Server uses 64 vBuckets instead of the standard 1024. While this difference generally won’t impact your development, it will impede your ability to move data from your local server to another cluster running Linux or Windows.

RAM matters
Couchbase is a "RAM first, disk second" database. Both the reads and writes are optimized to use RAM. On the write side, documents are written to the memory first and then flushed asynchronously to the disk. While volatile memory might not seem optimal for a database, remember that Couchbase will replicate your data on up to three nodes. Additionally, there are API methods that require a disk write before a write to RAM is considered a success.

On the read side, Couchbase maintains metadata about documents in the RAM to provide faster retrieval. Couchbase will also attempt to store as many documents as it is able to in the memory for faster access. Less available RAM means that Couchbase will need to fetch more documents from the disk. Couchbase uses a most recently used (MRU) algorithm to determine which documents are cached and which are evicted. The current beta version, Couchbase Server 3.0, will allow caching and eviction strategies to be tuned.
Summary

As we saw, Couchbase is an extremely flexible and scalable database. It offers a set of complimentary key/value and document features not found in any other database. In the next few chapters, we'll explore these features in detail. You will learn how and when to use them.

We also set up our single-node Couchbase cluster. Our default and sample buckets were created. We explored the Couchbase Console and discussed cluster architecture. With this knowledge in hand, you're ready to dig into application development with Couchbase.

If you've used either Memcached or CouchDB, you'll find the next three chapters to be somewhat familiar. In the next chapter, we're going to dig deep into Couchbase's key/value API. As we'll see, at first it will look a lot like Memcached, but it'll quickly go above and beyond.
Where to buy this book

You can buy Couchbase Essentials from the Packt Publishing website.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

Click here for ordering and shipping details.