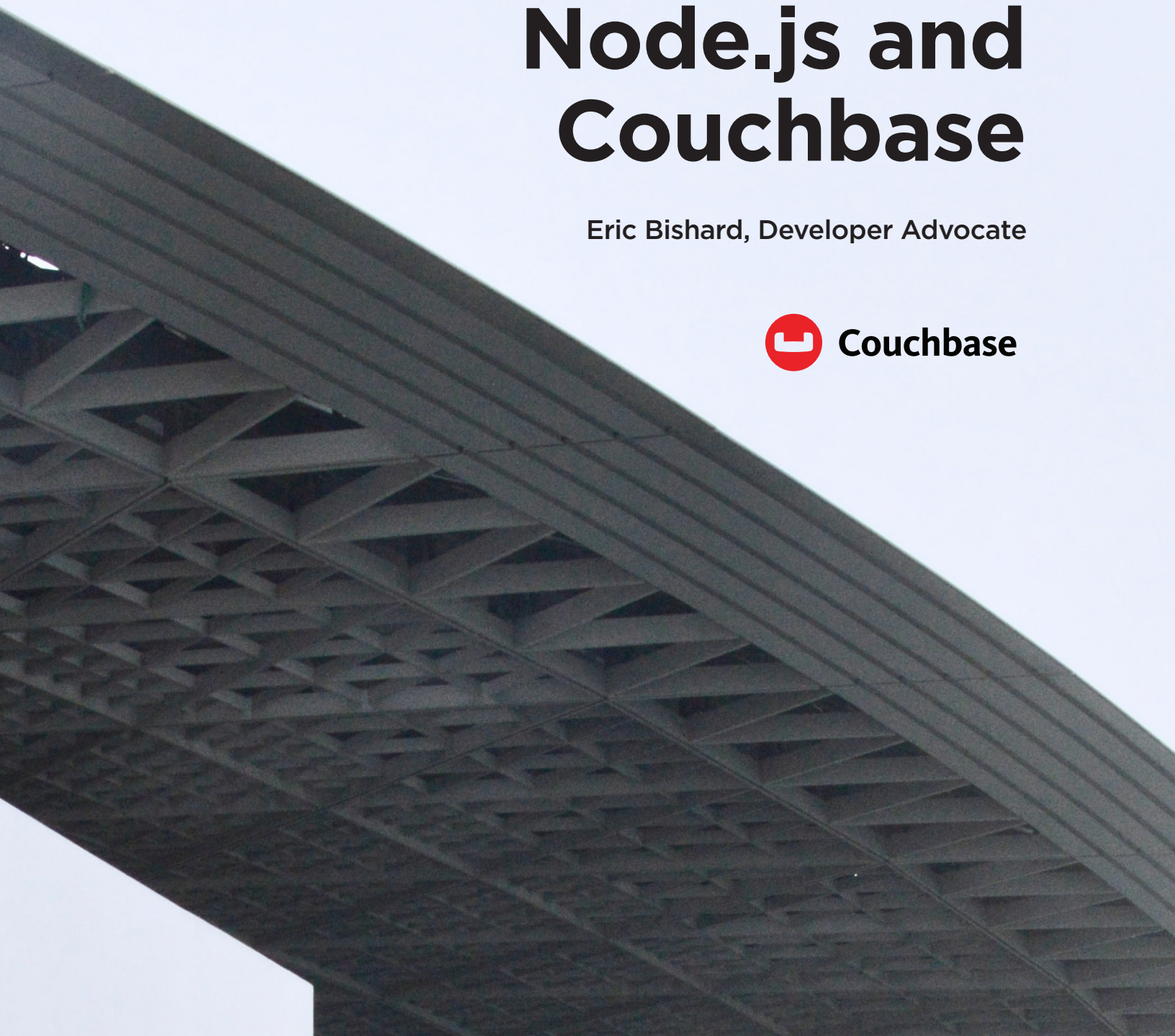
DEVELOPER GUIDE

Build World-Class Web Apps Using Node.js and Couchbase

Eric Bishard, Developer Advocate



Couchbase

Contents

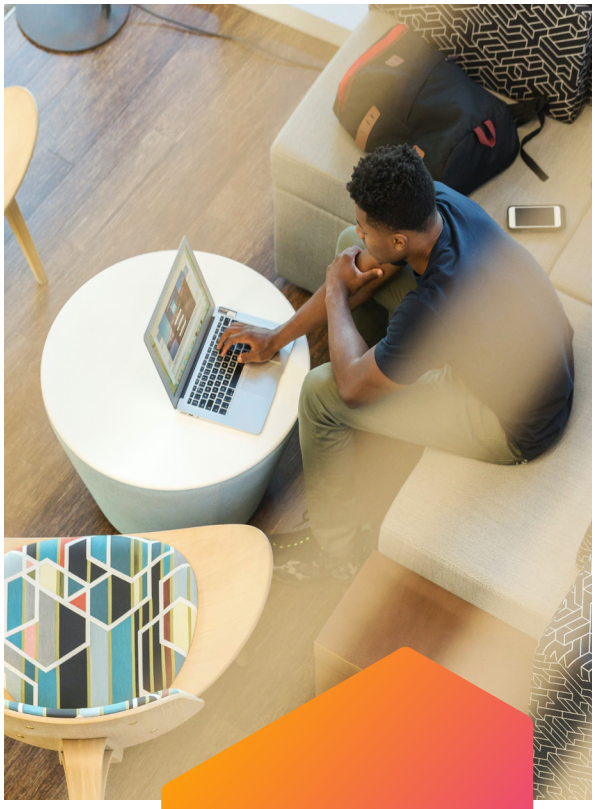
Introduction

As a Node.js developer, the last thing you want to have to worry about is your application database. After all, there are more important issues demanding your time and attention.

Fortunately, NoSQL databases such as Couchbase are a natural fit for JavaScript applications because of their native use of JSON documents. With JavaScript from front-end to backend, development is integrated, leaving you free to tackle more interesting challenges.

If you're looking to build a world-class web app using Node.js, then Couchbase is the perfect fit for your backend database. The Couchbase ecosystem offers a premium set of tools and the perfect IDE for Node.js developers, so the only limit to what you can build is your imagination.

In this Developer Guide, we'll take a closer look at how to develop high-performing web apps using Node.js and Couchbase. Let's get started.



Why Couchbase?

Couchbase is a distributed JSON document database with all the capabilities of a key-value store, a search engine and an RDBMS, including SQL, distributed ACID transactions and more.

On one end, Couchbase is built for microservices and serverless consumption-based computing in the cloud; on the other end, it's designed for edge computing on occasionally and locally connected mobile/IoT devices. Bottom line: Couchbase is perfect for multicloud deployments.

Because Couchbase manages JSON documents, it eliminates the need for a hard-coded schema in the database, although Couchbase does include logical structure to make organizing those documents as intuitive as old-school schema. The application object definition, available within JSON, is the editable schema you get to control as a developer.

You only write the JSON into the database once and then you can apply multiple data processing capabilities on it, including:

- Full SQL programmability including ACID transactions
- In-memory caching processing speeds
- Key-value store capabilities
- Full-text search (information retrieval)
- Data analytics (ad-hoc querying)
- Event-driven (reactive) programming and change data capture

As a result, Couchbase serves as a reliable system of record, while concurrently handling key-value operations of microsecond latency, SQL queries and text searches in milliseconds, and ad-hoc analytical queries spanning tens of seconds, without any query impeding another.

Traditional databases simply don't offer these capabilities out of the box.

When you use Couchbase, you reduce data and database sprawl, improve security, decrease administration and lower overall cost. But most importantly, Couchbase enables you to build apps quickly and deploy them at scale.

New to Couchbase? Start with our developer docs

Or, try out Couchbase Cloud for free when you sign up today.

Using Node.js with Couchbase

There are three main ways to use Node.js with Couchbase Server:

- The Node.js SDK
- The Node Ottoman Object Data Modeler (ODM)
- The RAGE Stack with Couchbase

Let's take a closer look at each one.

Node.js SDK

The Couchbase Node.js SDK is the default choice for JavaScript developers building applications with Node.js.

The SDK 3.x version is a complete rewrite of the 2.x API, providing a simpler surface area and adding support for future Couchbase Server features like Collections and Scopes.

The 3.x SDK also brings in promises to reduce the complexity of asynchronous JavaScript in client applications, as well as extending the management APIs and bringing better debugging and logging options for the developer.

For more information on how to use the Node.js SDK, read Install and Start Using the Node.js SDK with Couchbase Server or visit our Node.js Developer Portal.

Node Ottoman ODM

Ottoman is an Object Data Modeler (ODM) for Couchbase's Node.js SDK providing JSON schema and validation for NoSQL. It is designed to eliminate most boilerplate code necessary to build Node.js apps with Couchbase so you can build systems that are easy to design, maintain and scale.

With Ottoman, you declare schema in your code. Although Couchbase has no schema enforcement for your documents, most applications need some level of schema even in NoSQL.

Although Ottoman creates an abstraction over the Couchbase SDK, the benefits outweigh the drawbacks. A developer creates a lot of logic around creating and updating documents, writing pre- and post-lifecycle, working with data structures and validation.

For more information on how to use the Ottoman ODM with Couchbase, read our Developer Quick Start Guide for Ottoman or take a high-level tour of Ottoman in this blog post.

RAGE Stack with Couchbase

You also have the option of using the RAGE stack — React, Apollo Client, GraphQL and Express — alongside Couchbase Server for full-stack JavaScript application development.

You can use React, Apollo and GraphQL on the client side, and Express and Express GraphQL on the server side. The Express application also uses the Couchbase Node.js SDK, allowing you to connect to your Couchbase database, make queries, and return data to the API.

For a walkthrough demonstration of full-stack development with RAGE and Couchbase, watch this on-demand Couchbase Connect presentation (with its accompanying code repo).

Example Projects Using Couchbase & Node.js

Now that you're familiar with how to use Node.js and Couchbase together, it's time to dive into some example use cases and projects.

The most popular use case of Couchbase and Node.js for user profiles and session stores.

User Profile & Session Stores

One of the most common use cases for a NoSQL database is to create a user profile and session store. Why NoSQL specifically? Because profiles often need to be flexible and accept data changes frequently. While changes are possible with a traditional RDBMS, regular changes with a relational database require more work and impose larger performance penalties.

So, how do you create a user profile using Node.js and Couchbase Server? Here's a high-level look. For a detailed walkthrough, please consult this video: Develop a User Profile and Session Store with Node.js.

When it comes to managing user data, you need a way to create a user profile store and session store and then associate other JSON documents with them. Here are some important principles to remember when creating a user profile store:

- Store account data like usernames and passwords in a profile document.
- Pass sensitive user data with each user action request.
- Use a session that expires after a set amount of time.
- Store session documents with an expiry limit.

Video blog post: Develop a User Profile Store and Session Store with Node.js - Video

Old blog post: Create a User Profile Store with Node.js and a NoSQL Database

STEP #1 CREATE THE API WITH NODE AND EXPRESS

First, create a project directory for your Node.js app and install any dependencies.

This creates a working directory for your project and initializes a new Node project. Your dependencies include the Node.js SDK for Couchbase and Express Framework and other utility libraries like body-parser to accept JSON data via POST requests, uuid for generating unique keys and bcryptjs to hash your passwords to deter malicious users.

Next, bootstrap your application with a server.js file. You also need to create an index in Couchbase Server because you'll be using the N1QL query language for one of your endpoints. Primary indexes are not recommended fo production-level code.

STEP #2 SAVING A NEW USER TO THE PROFILE STORE

A user profile contains any information describing a user, such as address, phone number, social media profiles, and more. It's never a good idea to store account credentials in the same document as a user's basic profile information. You'll need a minimum of two documents for every user, so let's take a look at how you can structure those documents.

Your user profile document should have a key that you will use to refer to in related documents. Most often, this key is an auto-generated UUID.

Your Profile document will have a JSON value that includes two properties: email and a type property. The type property is an important indicator that describes our document similar to how a table organizes records in a relational database. This is a standard convention in a document database.

The account document associated with your user profile will have a key that is equal to a given user's email address. Your account document should also have a type, as well as a pid referring to the key of your profile document along with an email address and hashed password.

Now you have an established model for each document and a strategy for relating those documents without having created any database constraints.

STEP #3 AN ENDPOINT FOR ACCOUNT CREATION

The next step is to create an endpoint for account creation to your server.js file. Make sure all of the following items are included.

First, double-check that both an email and password exist in the request.

Next, your endpoint should create an account object and a profile object based on the data that was sent in the request. The pid that you're saving into the account object should be a unique key and should be set as the document key for your profile object.

The account document should use the email as its key. In the future, if other account details are needed (like alternate email, social login, etc.) you can associate other documents to the same profile.

Rather than saving the password in the account object as plain text, you should hash it using Bcrypt. The password should be stripped from the profile object for security. (For more information on password hashing, check out this tutorial.)

With the data ready, it's time to insert it into Couchbase. The goal of this save should be all or nothing. You want both the account and profile documents to both be created successfully or else the entire save should be rolled back. Depending on the success of the save, you'll return some information to the client.

You also could have used N1QL queries for inserting the data, but it's easier to use CRUD operations with no penalty on performance.

STEP #4 USING SESSION TOKENS FOR SENSITIVE DATA

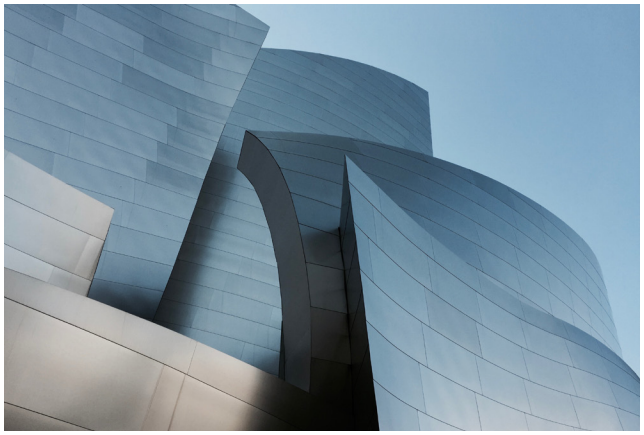
With the user profile and account created, you want the user to sign in and establish a session that will be stored in the database referencing their user profile. This session document should be set to eventually expire and be removed from the database.

The session data model, like the other documents, should have a different type. Just like with the account document, it should have a pid property that references the target user profile.

The code that makes this possible should be in your login endpoint.

This endpoint should validate the incoming data and then execute an account lookup by email address. If data comes back for the email, you can compare the incoming password with the hashed password returned in the account lookup. Provided this succeeds, your endpoint creates a new session for the user.

Unlike the previous insert operation, you should set a document expiration of an hour (3600 s). If the expiration isn't refreshed, the document should be set to disappear. This is good because it forces the user to sign in again and create a new session. This session token is then passed with every future request instead of the password.



STEP 5: MANAGING A USER SESSION WITH TOKENS

Eventually, you'll want to get information about a user's profile as well as associate new information with the profile. For this, you should confirm authority through the session.

You can confirm the session is valid using middleware. A Middleware function can be added to any Express endpoint. This validation should be a simple function that has access to your endpoint's HTTP request.

Your code should check the request for an authorization header. If you have a valid bearer token with a session id (sid), you can do a lookup. Your session document should have the profile id in it. If the session lookup is successful, save the profile id (pid) in the request.

Next, you'll need to refresh the session expiration and move through the middleware and back to the endpoint. If the session doesn't exist, no profile id will be passed and the request will fail.

After that, you can use your middleware to get information about the profile in your account endpoint. Note: the validation should happen before the rest of the request.

FINAL THOUGHTS

Those are the high-level steps to create a user profile and session store using Node.js and Couchbase.

Again, for a detailed walkthrough, please consult this video: Develop a User Profile and Session Store with Node.js. And for more advanced reading material, please consult this article: User Profile Store: Advanced Data Modeling Part 1. All code mentioned in the above articles can be found in the couchbaselabs / couchbase-nodejs-blog-api repo on GitHub.

Other Example Projects

While user profile and session stores are the most popular use case, Couchbase and Node.js can also be used for a variety of other apps and projects, including:

- Full-text search functionality (whether standalone or part of a bigger project)
- Chatbot applications
- Points-of-interest and travel apps
- Bitcoin and other cryptocurrency applications
- Analytics on massive datasets

Conclusion

You shouldn't have to think twice about which database to use for your next Node.js app. As a NoSQL document database, Couchbase is a perfect fit. In this Developer Guide, you learned:

- Why Couchbase is a natural choice when building with Node.js
- How to use Node.js alongside Couchbase, including the SDK, Ottoman ODM and RAGE stack
- What common projects use both Node.js and Couchbase to tackle today's development challenges
- For more information on how to use Node.js with Couchbase, check out our portal for Node.js developers.

Now, when it's time to build a new web app, your possibilities are endless. The only question is: What will you build?

Couchbase

About Couchbase

Unlike other NoSQL databases, Couchbase provides an enterprise-class, multicloud to edge database that offers the robust capabilities required for business-critical applications on a highly scalable and available platform. As a distributed cloud-native database, Couchbase runs in modern dynamic environments and on any cloud, either customer-managed or fully managed as-a-service. Couchbase is built on open standards, combining the best of NoSQL with the power and familiarity of SQL, to simplify the transition from mainframe and relational databases.

Couchbase has become pervasive in our everyday lives; our customers include industry leaders Amadeus, American Express, Carrefour, Cisco, Comcast/Sky, Disney, eBay, LinkedIn, Marriott, Tesco, Tommy Hilfiger, United, Verizon, as well as hundreds of other household names. For more information, visit www.couchbase.com.

3250 Olcott Street | Santa Clara, California 95054 | 1-650-417-7500 | www.couchbase.com