What does MongoDB do?

A database for the people, and the companies

Justin
Jun 29, 2021

(MongoDB paid me to write this post. But I was going to write it anyway.)

The TL;DR

MongoDB is a highly popular **document database** (i.e. NoSQL) for powering your applications.

- **Every app** (give or take a few) is powered by a **production database**
- These days, you can choose between something **structured** (SQL) or **unstructured** (NoSQL)
- Document databases are **schemaless** – you just throw data in, and worry about it later
- MongoDB provides **an ecosystem** around its document DB – managed services, an IDE, analytics, and search (among other things)

MongoDB is one of the OGs of what I consider the modern tech ecosystem (IPOd back in 2017), and powers apps at some of the world's largest enterprises. But they're not a stodgy shop – their cloud product (Atlas) has been growing like a weed recently, and their website actually (gasp) explains what the product does. So let's dive in and see what they're all about.

A refresher – what's NoSQL?

Although their marketing has moved away from the NoSQL moniker in recent years, the key to understanding MongoDB – and why they had such a major impact on the ecosystem – is to understand NoSQL.

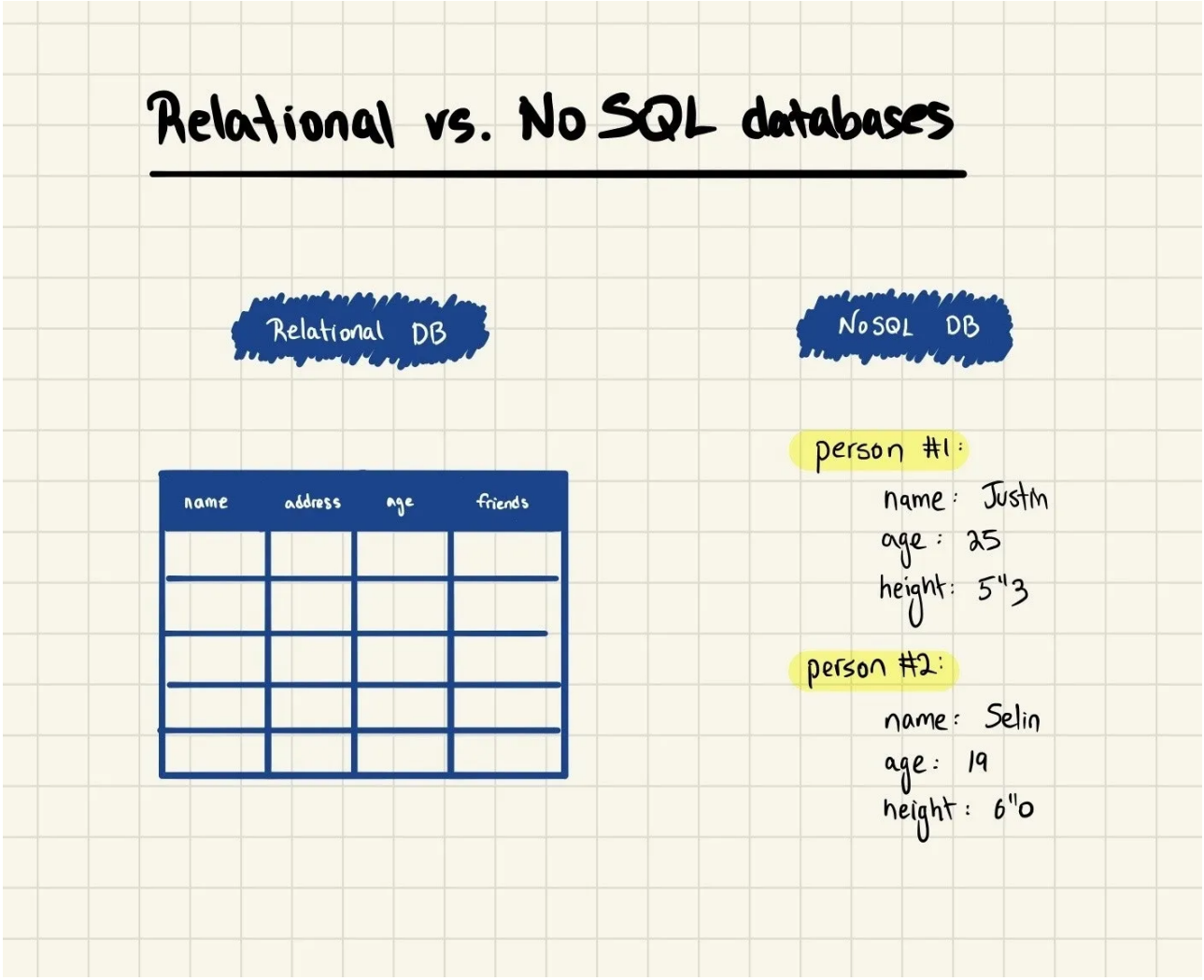
> 🔮 **Dependencies** 🔮
>
> If you haven't yet, check out the Technically posts on relational databases and then NoSQL. This section will jog your memory, but to go in depth you'll

> want to read these bad boys.
>
> **🔮 Dependencies 🔮**

For most of technical history, apps were powered by relational databases. A relational database is all about **structure** – before you put any data into it, you plan out what your data is going to look like. What are my column names going to be? How will my tables relate to each other? What type of data will each column store? All of that information together is called a **schema**, and it's the *sine a qua non* of relational data. You'll usually hear relational databases referred to as **SQL databases**.

(By the way, **SQL** stands for Structured Query Language, and it's a category of mini-programming languages that people use to get data in and out of these databases. Rigid structure means that querying data is easy and straightforward.)

NoSQL, on the other hand, is exactly what you think it is – **unstructured**. Instead of rigidly defining what your data looks like before storing it, you just kind of throw it in there and worry about it later. Querying it is more difficult, but it's a lot easier to scale horizontally to multiple servers. Here's a quick snapshot of how these storage patterns differ:



The thing about NoSQL is that those two people (Justin and Selin) don't even need to have the same attributes. Justin can have a "favorite food" value, while Selin has a "hair color" one. In SQL, if one record has a column, all records in the table need to have that same column. You can imagine how that **flexibility** that NoSQL offers is convenient – but it also means that down the road, you have less predictability around what your data will look like, and that can screw things up.

There are all different kinds of NoSQL databases – key value stores, wide column stores, etc. The most popular type, though, is called a **document store**, and it's the core of how MongoDB works. The canonical "unit" – the main thing at play – is a document, which is just a collection of different keys and values. Here's a document that represents my band:

```
{
    guitar_players: {
```

```
        justin_gage: {
            height: 5"3,
            weight: "wouldn't you like to know"
            },
        jake_mendel: {
            height: 5"11,
            weight: 170,
            axe: “mary kaye”
            }
        },
    drum_players: {
        chris_behrens: {
           height: 6"0,
           weight: 185,
           cymbal: “zildjian”
        }
    }
}
```

Each entry has a key (what does this data refer to?) and a value (the actual data). For Jake, height is a key, and 5”11 is a value. At a higher level, guitar_players is a key, and the entire entries for both Justin and Jake are the values. Notice how there’s really no structure at all to this – values and keys can be anything, data types don’t need to match, etc. This is the double edged sword of NoSQL.

The basic MongoDB product

MongoDB originally started as a company called 10gen – and when they open sourced their document database in 2008, the idea of using a NoSQL database in production was still not quite widely accepted. It took until 2013 for them to rebrand to MongoDB, and by then, they had built out a proper product suite with cloud, backups, monitoring, and an enterprise offering. And that’s the MongoDB you’ve probably heard of today. We’ll start by running through the basic open source product, and then explore the broader suite and where MongoDB fits into the ecosystem.

MongoDB is a document database

MongoDB basically invented the modern document database as we know it (commercially, at least). So let's dive into what that is.

Like we covered above, a document database is just a series of keys and values. MongoDB has their own terminology for the **taxonomy** of data (i.e. what contains what):

- A **document** is a series of keys and values. This might correspond to a single user.
- A **collection** is a group of multiple documents. This might correspond to all of your users. It's like a table in a SQL database.
- A **database** is a group of collections, like the same name means in SQL databases.

You can read more about the information hierarchy in their docs here.

Interacting with MongoDB

Once you've got your database provisioned and ready to go, there are a few different ways you can actually interact with it. This is true of most databases (SQL too), but we'll cover it here too:

- **The mongo Shell** – you can write commands directly via your terminal, in JavaScript. Generally useful for setup, quick commands, and administrative tasks.
- **Drivers** – drivers let you write MongoDB queries in your favorite programming languages like Python and JavaScript. Synonymous with *client libraries* (if you've seen that term around).
- **GUIs** – you can interact with your MongoDB data through a graphical user interface on your local machine. MongoDB provides a free GUI called Compass, but there are other popular ones like Studio3T.

It's all the same data behind the scenes, but depending on what context you're interacting with your database in (testing, production, etc.) you might use one or all of these methods.

What using MongoDB looks like

The lifecycle of a basic app is usually called **CRUD** – it stands for **Create**, **Read**, **Update**, **Delete**. These are the 4 things that most apps do, give or take. For the utmost simplicity, imagine a note taking app backed by MongoDB. Notes are stored in the database, and the app lets you:

- **C**reate a new note (insert a new value into Mongo)
- **R**ead an existing note (find an existing value in Mongo)
- **U**pdate an existing note (edit an existing value in Mongo)
- **D**elete an existing note (remove an existing value in Mongo)

To actually *interact* with your MongoDB instance, you can use their CLI directly from your Terminal, or use a client library in your favorite programming language like JavaScript. If we're building our note taking app, here's a sample of Mongo commands we might use:

- **C**reate a new note with `insertOne()`
- **R**ead an existing note with `findOne()`
- **U**pdate an existing note with `updateOne()`
- **D**elete an existing note with `deleteOne()`

There are hundreds of these commands with all different types of filters and parameters, and these are just scratching the surface.

> 🔍 **Deeper Look** 🔍
>
> Mongo has commands for aggregating data, too. If you want to find how many documents you have in a collection that match some criteria, or you want to add up revenue for products sold in the past 3 months, you'd build what they call an aggregation pipeline. This is analogous to GROUP BY in SQL.
>
> 🔍 **Deeper Look** 🔍

The MongoDB ecosystem

So now that you understand the *database* that is MongoDB, we can get into what makes the company so valuable – all of the supporting products they've

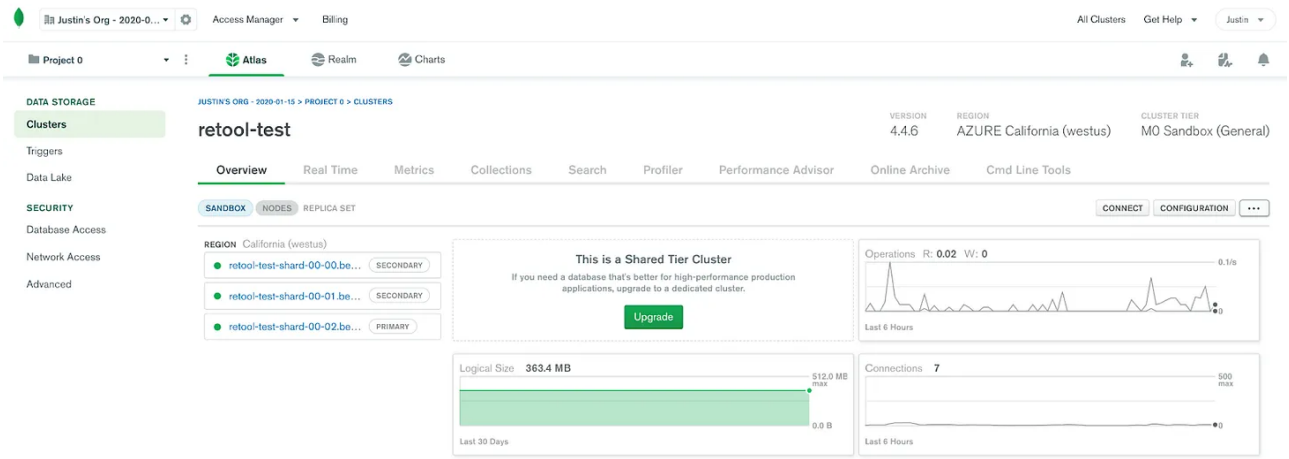
built around that database. Like most mature companies in developer tools, Mongo makes most of their money off of managed services (hosting your database for you), and use ancillary services like monitoring to make the product suite more compelling as a whole.

Atlas

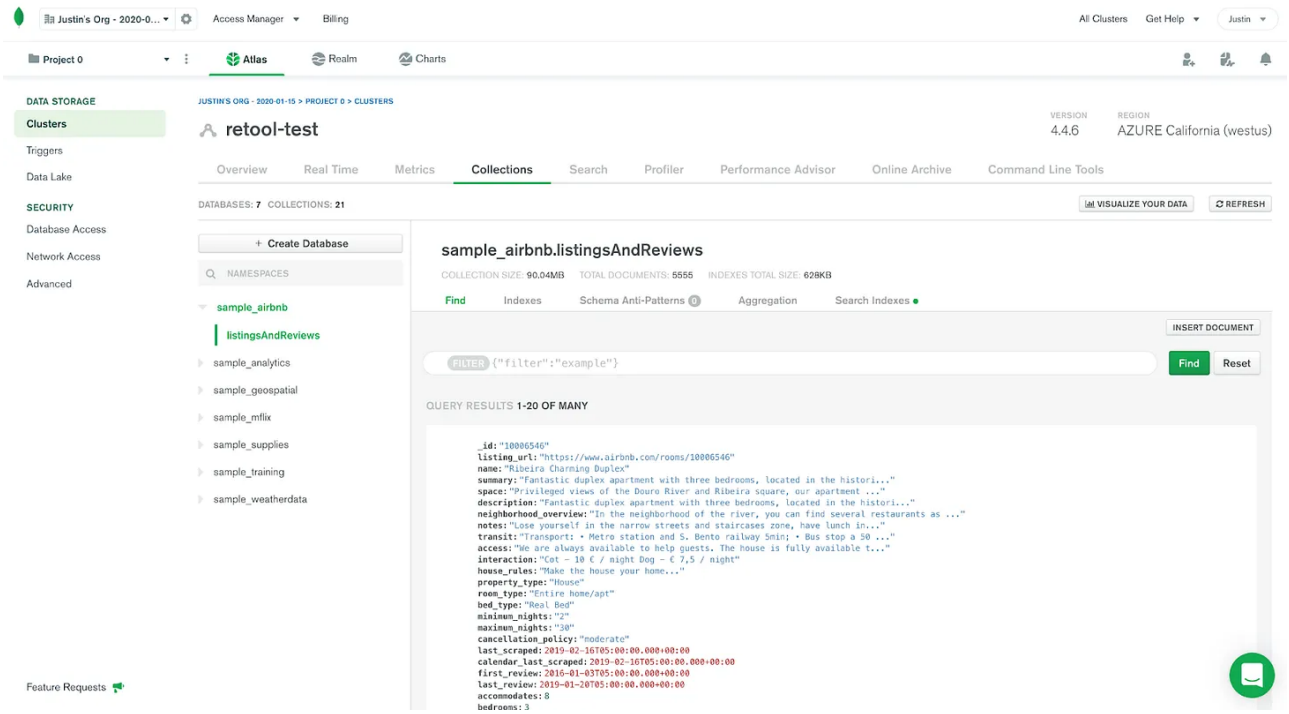
Atlas is Mongo's fully managed cloud database – it's MongoDB, but you don't need to host it yourself. When you set up Atlas, you choose between 3 different plans and any additional services you want to add on. And when you're deploying, you can choose which cloud you want to use (along with any specific regions in those clouds) – AWS, GCP, or Azure. You can read more about pricing here.

When MongoDB started, they made most of their money off of deploying the DB in their customers' data centers (e.g. enterprise deals). But since Atlas was officially released in 2016, it has been growing like a weed, and now represents more than 50% of their revenue (!). It's a testament to a great product and tides shifting towards more workloads in the cloud.

Atlas gives you a lovely management console on the web for interacting with your MongoDB data. Here's an overview of a test cluster I set up for work:



You can click into any of your collections to see your data and write basic queries, too:

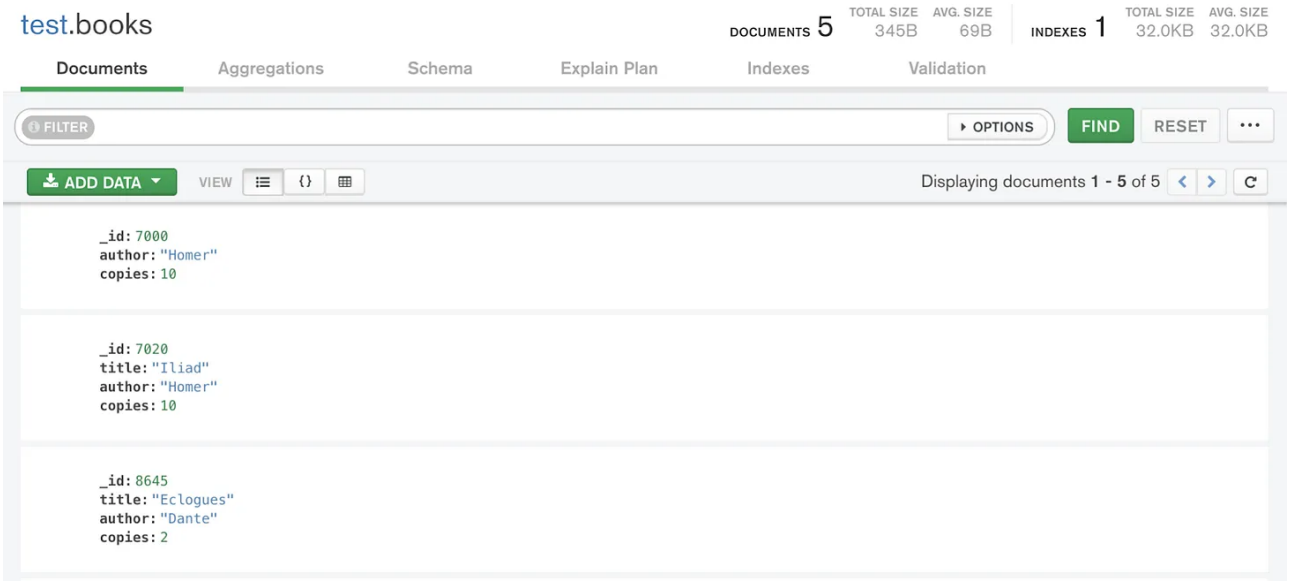


I only used it for a basic project, but setting up and using Atlas was, personally, a smooth experience. And it was free!

If you're a larger enterprise and you want to deploy MongoDB in your own data center, you can use their Enterprise Advanced product. It's a package of the MongoDB software, support, and services meant for truly giant companies.

Compass

For more fully featured querying, MongoDB has an IDE (integrated development environment) called Compass that you can download locally to your Mac or PC. It's completely free and, in my experience, the best (and most feature rich) way to query your MongoDB data locally.



Search

One of the great problems among document databases is **search** – teams need what Google built for themselves internally. The usual state of the art is Elasticsearch, another document database optimized for search and usually used by teams for storing performance data like logs. The normal workflow was for teams to basically duplicate their MongoDB data into a separate Elasticsearch cluster (which, by the way, is also hard to run and thus a public-company-selling-a-managed-service exists).

But as of a couple of years ago, MongoDB offers a search product for your data in Atlas.

Some other stuff too

Beyond Atlas, MongoDB on prem, Compass, and Search, MongoDB has a few other product lines aimed at making MongoDB your single, go to place for data. A few examples:

- Realm – basically Firebase, a suite of services for building mobile apps
- Realm Sync - out-of-the-box data synchronization service for building mobile apps
- Charts – service for visualizing your MongoDB data (and sharing that viz)
- Atlas Data Lake – lets you join your MongoDB and S3 data

I hinted at this before, but if you take a look at MongoDB's site, you'll see that they're slowly moving away from the "NoSQL" branding and more towards just being a good database for developers. And the buildup of these products around the ecosystem helps with that story.

Further reading

- It's hard to talk about "competition" for production databases, but MongoDB generally competes against relational databases (Postgres, MySQL, etc.) and other NoSQL DBs like DynamoDB or Firebase
- Couchbase, a very MongoDB-like product, just released their S-1

Comments

Write a comment...
