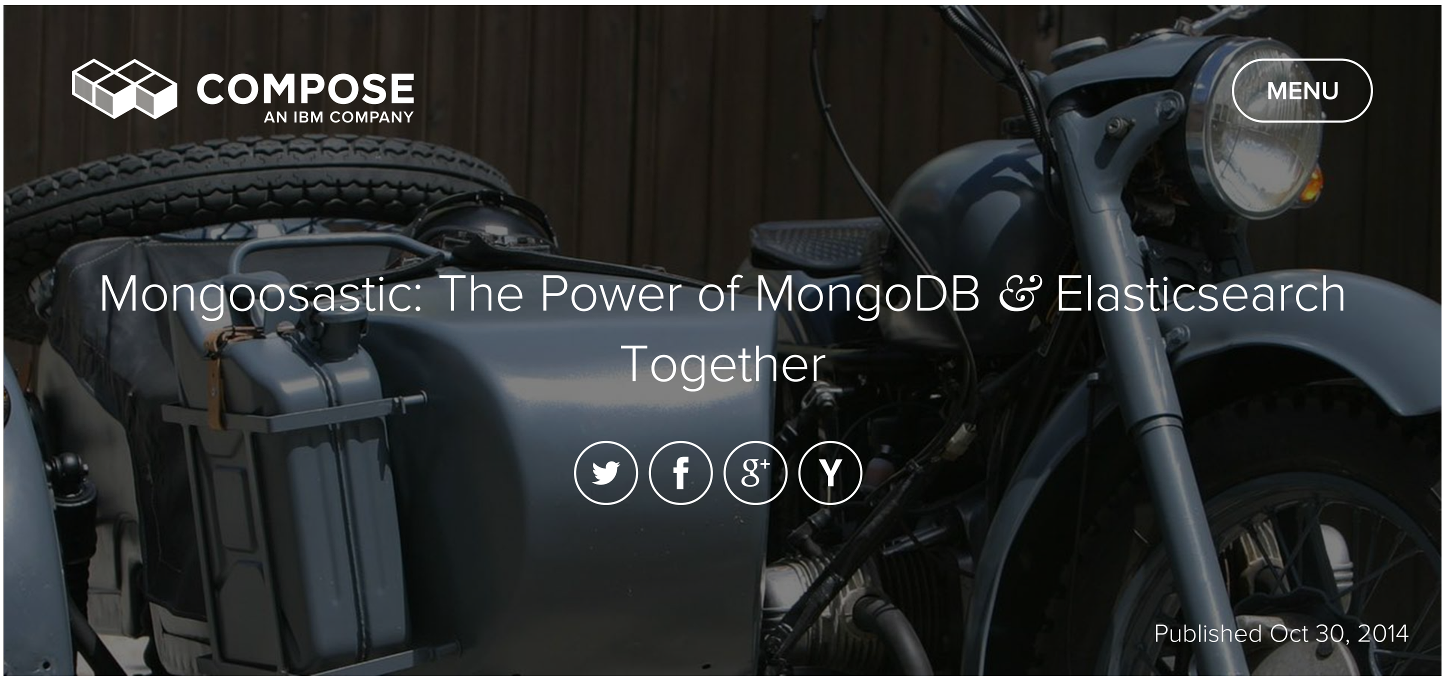


Mongoosastic: The Power of MongoDB & Elasticsearch Together

Published Oct 30, 2014

TL;DR: *Two databases can be better as one as we show how you can use Mongoosastic to user MongoDB and Elasticsearch at the same time leveraging each one's strengths without rebuilding your code.*

At Compose we're all about using the right database for the job, and sometimes that can mean using more than one database harnessed to one another. If that sounds strange or over complex, let's show you an example where Elasticsearch can help overcome some of the inadequacies of MongoDB.

MongoDB is a great general purpose database but one place where its limitations show up is with its full text searching. We only got a finalised full text search mechanism in MongoDB 2.6 and even then you can only have one field per collection indexed for searching in that way. If you also want richer full text search capabilities than what MongoDB offers then you will need an alternative. Elasticsearch gives you a much more extensive and powerful search capability, being built upon the Lucene text search library, but you may not want to move your application over to another database. Apart from anything else, the other database may not be suitable for the rest of your application workload.

Splitting your database between two databases can seem like a complex problem, but there are tools to assist in the process. To demonstrate one way we can approach the problem, we're going to talk about a simple book database, written using Node.js, Express and using the Mongoose library to access the MongoDB database. If you've not encountered Mongoose before, we've got some background articles on it, but in short its a schema-defined object layer for working with MongoDB in JavaScript which makes working with well-defined data sets easier.

The Book Database

Our starting point is ridiculously simple and you can find it in our Compose Examples repository. At the core is our Book schema definition for Mongoose:

```
var mongoose = require("mongoose");

mongoose.connect(process.env.COMPOSE_URL);

var bookSchema = new mongoose.Schema({
  title: String,
  author: String,
  description: String
});

var Book = mongoose.model("Book", bookSchema);
```

This snippet of code sets up the MongoDB connection, defines our Book object and creates a model we can use for creating, manipulating, saving, querying and deleting them.

Imagine though, if we were asked to add full text search to the description, and for good measure, we were going to add the content of the book and other rich text fields to the schema. If it had just been the description field, we might have been able to get away with MongoDB's full text search, but we need more and that's where Mongoosastic comes in.

Mongoosastic introduced

Mongoosastic is a Mongoose plugin which lets you select fields to be indexed in Elasticsearch. At its most basic, it's a matter of simply marking the fields in the schema like this:

```
var mongoosastic=require("mongoosastic");
var bookSchema = new mongoose.Schema({
  title: String,
  author: String,
  description: { type:String, es_indexed:true },
  content: { type:String, es_indexed:true }
});
```

By default, Mongoosastic will replicate the all the fields in a schema into the Elasticsearch database, so by flagging only the fields you want you can reduce duplication. Once the schema is defined, you can plugin Mongoosastic. If your Elasticsearch database was local, then it would simply be

```
bookSchema.plugin(mongoosastic);
```

But if you are working with Compose's hosted Elasticsearch, you'll need to provide the options to tell it where to connect to.

```
bookSchema.plugin(mongoosastic,{
  host:"haproxy2.dblayer.com",
  port: 10293,
  protocol: "https",
  auth: "username:password"
//  ,curlDebug: true
});
```

The value for host and port you can get from the Compose Elasticsearch overview and the username and password are defined by you in the users section of the Elasticsearch dashboard. Obviously, this is an example and you should adopt a more secure way of storing such credentials in production. Note the protocol is set to 'https' as Compose only allows encrypted connections into the database so the default of 'http' won't work (and doesn't report an error as it never responds). We've commented out the "curlDebug" option - uncomment that and you'll see the requests being made to Elasticsearch being printed on the console.

There's one other step needed and that's to create a mapping on Elasticsearch for our index. We use this snippet...

```
Book.createMapping(function(err, mapping){
  if(err){
    console.log('error creating mapping (you can safely ignore this)');
    console.log(err);
  }else{
    console.log('mapping created!');
    console.log(mapping);
  }
});
```

...in our example - an error may occur if the mapping already exists so it can be ignored. This is merely a stub for more extensive error checking in production code. What it does do though is give up a mapping in Elasticsearch.

To save a book

Saving a Book object in Mongoose is simple:

```
var book = new Book({
  title: req.body.title,
  author: req.body.author,
  description: req.body.description,
  content: req.body.content
});
book.save(function(err) {
  res.redirect("/");
});
```

And in Mongoosastic it is exactly the same. The plugin takes care of all the work behind the scenes. If you want to wait until the document is indexed then you can do:

```
book.on('es-indexed', function(err,res) {}
```

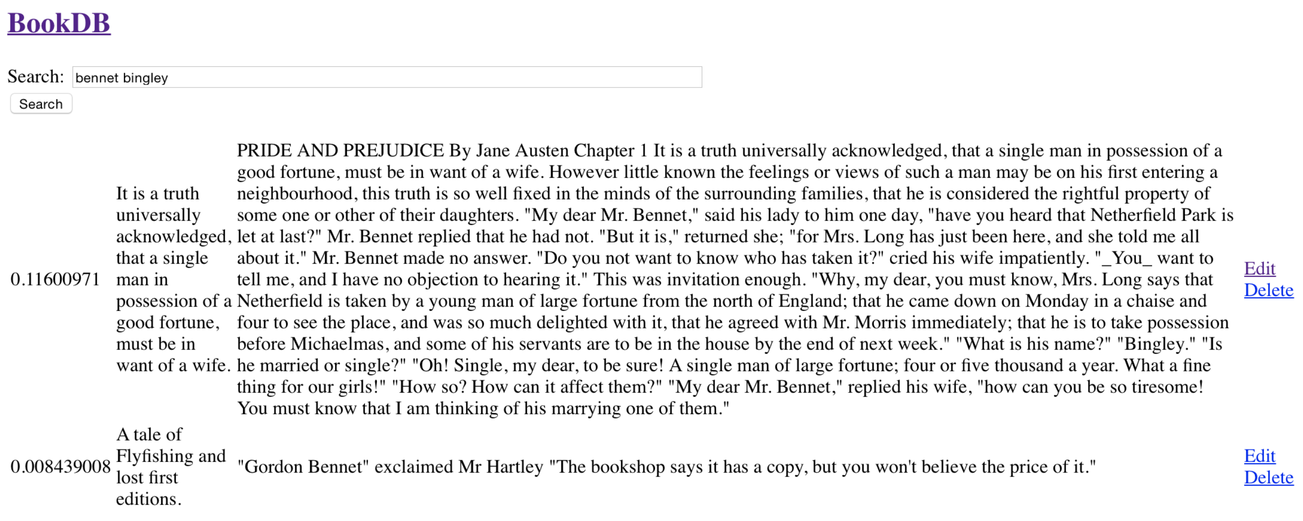
in the save callback and execute your own code when it has been indexed (or errored).

Finding a book with Elasticsearch

Mongoosastic doesn't touch the Mongoose MongoDB `find` methods for searching MongoDB data and instead adds a `search` method which gives you access to the Elasticsearch DSL. At its simplest this can be just the terms we are looking for:

```
Book.search({ query:terms }, function(err,results) {
```

The results that come back from a query like this are in the [Elasticsearch search result format] with metadata about the search execution, shards queried and then a hits object. Within hits are some values regarding the success of matching and then a hits array which contains the relevancy score, id and a set of fields retrieved from Elasticsearch for that match. What that actually means is that when you do a search as above, your results will be in an array `results.hits.hits` and for each found document in that array there'll be the relevancy score in `_score` and the document's fields in `_source`. In the example code, that's the first Elasticsearch option and that will give results like this:



BookDB

Search: bennet bingley

Search

0.11600971	It is a truth universally acknowledged, that a single man in possession of a good fortune, must be in want of a wife.	PRIDE AND PREJUDICE By Jane Austen Chapter 1 It is a truth universally acknowledged, that a single man in possession of a good fortune, must be in want of a wife. However little known the feelings or views of such a man may be on his first entering a neighbourhood, this truth is so well fixed in the minds of the surrounding families, that he is considered the rightful property of some one or other of their daughters. "My dear Mr. Bennet," said his lady to him one day, "have you heard that Netherfield Park is let at last?" Mr. Bennet replied that he had not. "But it is," returned she; "for Mrs. Long has just been here, and she told me all about it." Mr. Bennet made no answer. "Do you not want to know who has taken it?" cried his wife impatiently. "_You_ want to tell me, and I have no objection to hearing it." This was invitation enough. "Why, my dear, you must know, Mrs. Long says that Netherfield is taken by a young man of large fortune from the north of England; that he came down on Monday in a chaise and four to see the place, and was so much delighted with it, that he agreed with Mr. Morris immediately; that he is to take possession before Michaelmas, and some of his servants are to be in the house by the end of next week." "What is his name?" "Bingley." "Is he married or single?" "Oh! Single, my dear, to be sure! A single man of large fortune; four or five thousand a year. What a fine thing for our girls!" "How so? How can it affect them?" "My dear Mr. Bennet," replied his wife, "how can you be so tiresome! You must know that I am thinking of his marrying one of them."	Edit Delete
0.008439008	A tale of Flyfishing and lost first editions.	"Gordon Bennet" exclaimed Mr Hartley "The bookshop says it has a copy, but you won't believe the price of it."	Edit Delete

As you can see, a single mention of "Bennet" scores very low while both terms appearing in a block of text scores much higher. It's likely though, that often you'll want all the other MongoDB stored data with your found Elasticsearch documents. For that, Mongoosastic has the hydrate option (but see the update at the end):

```
Book.search({ query:terms }, { hydrate:true }, function(err,results) {
```

With hydrate set, once the results come back from Elasticsearch, the plugin does a second trip to MongoDB to unite those fields with the MongoDB data. It then reformats the contents of the inner `hits` array, adding in the MongoDB data and removing most of the metadata so that the contents look more like a MongoDB JSON object. That does mean that you lose the relevancy score though, so its down to what you need to do with the results as to whether you hydrate or not.

We have examples of hydrated and un-hydrated searches in the Bookdbtastic github repository so you can see the differences.

Just the start

This has been just a quick glimpse of how you can harness two different databases together. Mongoosastic opens up numerous possibilities for a developer – The query system gives you extensive access to the Elasticsearch Query DSL while the mapping facility allows you to tune your Elasticsearch indexes for faster more precise responses. And you get all this without giving up the convenience of MongoDB. And you'll find both MongoDB and Elasticsearch hosted at Compose.

Update: And just as after we published this article, Mongoosastic 2.0.0 has been released. It now uses the official Elasticsearch driver and has a number of changes in the syntax for a hydrating query which would change our last example line to:

```
Book.search( query_string:{ query:terms }, { hydrate:true }, function(err,resul
```

Hydration options can now control which fields are included in the hydrated records and hydration can be turned on by default. It's a great update to a great library.

SUBSCRIBE

DJ WALKER-MORGAN is Compose's resident Content Curator, and has been both a developer and writer since Apples came in II flavors and Commodores had Pets. Love this article? Head over to Dj Walker-Morgan's author page and keep reading.

COMPANY

About Us
We're Hiring
Articles
Write Stuff
Plans & Pricing
Database as a Service

SUPPORT

System Status
Support
Docs
Security
Privacy Policy
Terms of Service

DATABASES

MongoDB
Elasticsearch
RethinkDB
Redis
PostgreSQL
Enhanced
etcd
RabbitMQ

DEPLOYMENTS

AWS
DigitalOcean
SoftLayer