Phoenix with Ecto and MongoDb

Posted on February 9, 2017

How to use MongoDb with Phoenix and Ecto.

This might sound confusing:

- Use newest version of Phoenix
- Use newest version of Ecto
- Don’t use mongodb_ecto! Instead use the newest version of mongodb driver (mongodb).

This post explains techniques on how to leverage some of the features from Ecto, e.g. Changesets, but directly with the Mongodb driver. They work well together!

Also this post shows you how to use MongoDb driver. There’s not much documentation available on the mongodb driver site. This post shows you how to do things like indexes, and find, and updates with $set, etc.

Get started

How to start a project:

1. Create a new Phoenix project without a database specified. Ecto is included by default.
2. Now, adjust the mix, etc, by adding directly the mongodb driver.
3. Follow the other instructions in this post about startup, etc.

Background

Ecto 1.0 supported MongoDb, Ecto 2.0 & 3.0 don't. This has created a lot of confusion for developers who want to use MongoDb with Phoenix web applications. Don't downgrade to earlier versions of Ecto because you want to use MongoDb with Phoenix! You can use the newest versions of Ecto with the MongoDb drivers with a small amount of code.

This post's strategy lets you use many of Ecto's cool features, like Changesets, and also directly using the MongoDb driver. This might sound hacky, but it is not. The MongoDb drivers (thanks @ericmj and all) are really excellent quality, and if you have used MongoDb from Ruby or Javascript then you'll find this makes a lot of sense.

In this blog you will learn:

- How to use the MongoDb drivers with Elixir, Phoenix, and Ecto,
- How to use Ecto 2 and 3 with the MongoDb drivers,
- Example code,
- Snippits to help you get started.

Strategy

You want to build a Phoenix application that uses MongoDB. Here's how to do it.

1. Use the latest and greatest version of *Phoenix* and *Ecto* 3
2. Use the MongoDb driver directly (not the Ecto version)

Why Ecto? Ecto has a lot of great stuff, like Changesets, that are great. And they work really well with MongoDb (I'll explain), but we're not going to use the Repo part of Ecto.

One of the great things about Elixir and Phoenix/Ecto is that there really aren't very many deep dependencies, so it's not difficult.

Here's what my mix.exs looks like. Your's may differ, but the point is to get MongoDb and Ecto, but not the database specific Ecto drivers.

```
defp deps do
  [{:phoenix, "~> 1.2.1"},
   {:phoenix_pubsub, "~> 1.0"},
   {:phoenix_html, "~> 2.6"},
   {:phoenix_live_reload, "~> 1.0", only: :dev},
   {:gettext, "~> 0.11"},
   {:cowboy, "~> 1.0"},
   {:mongodb, ">= 0.0.0"},
   {:poolboy, ">= 0.0.0"},
   {:phoenix_ecto, "~> 3.0"},
   {:httpoison, "~> 0.9.0"},
   {:uuid, "~> 1.1" },
   {:logger_file_backend, "0.0.8"},
   {:exrm, "~> 1.0.8" },
 ]
end
```

And the applications section:

```
applications: [:phoenix, :phoenix_pubsub, :phoenix_html,
               :phoenix_ecto, :cowboy, :logger,
               :gettext, :mongodb, :poolboy, :logger_file_backend,
               :uuid, :httpoison]]
```

Quick Start

Configure databases

I added this to my prod.exs, dev.exs, test.exs:

In prod.exs:

```
config :my_app, :db, name: "my_mongo_prod"
```

In dev.exs:

```
config :my_app, :db, name: "my_mongo_dev"
```

In test.exs:

```
config :my_app, :db, name: "my_mongo_test"
```

Start MongoDb

In my lib/my_app.ex

```
defmodule MyApp do
  use Application

  def start(_type, _args) do
    import Supervisor.Spec


    # Define workers and child supervisors to be supervised
    children = [
      supervisor(MyApp.Endpoint, []),

      # 1. Start mongo
      worker(Mongo, [[database:
        Application.get_env(:my_app, :db)[:name], name: :mongo]])
    ]

    opts = [strategy: :one_for_one, name: MyApp.Supervisor]
    result = Supervisor.start_link(children, opts)

    # 2. Indexes
    MyApp.Startup.ensure_indexes
    result
  end

  def config_change(changed, _new, removed) do
    MyApp.Endpoint.config_change(changed, removed)
    :ok
  end
end
```

1. Mongodb is started with the database pool.
2. I also take this opportunity to ensure any indexes I need.

My ensure_indexes function looks like this. You won’t find a lot of documentation for this! But ‘command’ is like a back door into MongoDb and let’s you do just about anything.

```
defmodule MyApp.Startup do
  def ensure_indexes do
    IO.puts "Using database #{Application.get_env(:my_app, :db)[:name]}"
    Mongo.command(:mongo, %{createIndexes: "users",
      indexes: [ %{ key: %{ "email": 1 },
                    name: "email_idx",
                    unique: true} ] })
  end
end
```

Ecto

Changesets

Ecto uses the schema to validate the changeset. I use the Ecto schema for fields I know I will always have. This is really handy.

If you have fields which are dynamic then the Schema will not really help you in this case! But that’s fine. I like to have a bit of both.

The trick to using the changeset is to realize that there is a `changes` function that will give you *exactly the syntax you need to insert into mongodb*! What could be sweeter.

Schema

I can’t include the whole thing, but enough so you get the idea:

```
defmodule MyApp.User do
  use MyApp.Web, :model
  require Logger

  @primary_key {:id, :binary_id, autogenerate: true}  # the id maps to uuid
  schema "users" do
    field :email,         :string
    field :phone_number,  :string
    field :first_name,    :string
    field :last_name,     :string
    field :status,        :string
  end

  def changeset_new_user(user, params \\ %{}) do
    params = scrub_params(params)  # change "" to nil
    user
      |> cast(params, [:email, :phone_number, :first_name, :last_name])
      |> validate_required([:email])
      |> validate_phone_number
      |> put_change(:status, 1)
  end
```

Example: Creating a New User

Here's an example of How I create a new users (in my Controller)

```
defmodule MyApp.UserController do

...

  changeset_new_user = MyApp.User.changeset_new_user(%MyApp.User{}, params)

  {:ok, user} = Mongo.find_one_and_replace(:mongo, "users", %{}, changeset_new_u
  Logger.info("created new user #{inspect(user)}")
```

Example: Finding a Single (One) User

```
cursor = Mongo.find(:mongo, "users",
              %{"email" => "someone@somewhere.com"}, limit: 1)
list = Enum.to_list(cursor)
if length(list) == 1 do
  {:ok, hd(list)}
else
  {:error, nil}
end
```

I've got a pull request in for a `find_one` version of this, so it may become this in the near future :) :

```
user = Mongo.find_one(:mongo, "users",
                %{"email" => "someone@somewhere.com"]})
```

Example: Updating a Document

Here I just use $set to make the changes.

```
 {:ok, user_after} = if Map.size(user_update_changeset.changes) >= 1 do
  Mongo.find_one_and_update(:mongo, "users",
    %{"email" => "someone@somewhere.com"},
    %{"$set" => user_update_changeset.changes},
    [return_document: :after])
```

Example: Updating a Document no Changeset

For my applications it's like 80% of fields *are* known ahead of time, and I use Changesets for those.

The whole point of MongoDb is to have a flexible schema. That means you will not know all the fields (key names) ahead of time. I think it's a good strategy to separate your logic for the dynamic updates. Here's the code I use.

This is a snippit from a controller that takes an id for the user and a bunch of updates.

WARNING: The parameters are *not* validated. The service I have is working internally with already validated! You probably want to validate things before you stuff them in MongoDb.

```
def put_client(conn, params) do
  user_id = params["id"]

  # need to merge to x the attributes
  attributes = params |> Map.delete("id")

  reduced = Enum.into(attributes, %{}, fn({key,value}) ->
    {"$.#{key}", value}
  end)

  {:ok, user} = Mongo.find_one_and_update(:mongo, "users",
    %{"email" => "someone@somewhere.com"},
    %{"$set" => reduced }, [return_document: :after])
```

Example: Embedded Documents

Quite often you want to have an array of “models” that are embedded in a document. This is how I did it.

I created another model object as above. Then I just use the changeset and directly manage inserting it into an array like this:

```
{:ok, result} = Mongo.find_one_and_update(:mongo, "users",
  %{"email" => "someone@somewhere.com"},
  %{"$push" => %{ addresses: one_address },
  [return_document: :after])
```

Removing is pull:

```
Mongo.find_one_and_update(:mongo, "users",
  %{"email" => "someone@somewhere.com"},
  %{"$pull" => %{ addresses: one_address } } )
```

Example: Find One and Update

In this case the actual fields are in my schema, so I can use a changeset and changes.

```
user_update_changeset = MyApp.User.changeset_update_user(%MyApp.User{},
                                                            params)

{:ok, user_after} = Mongo.find_one_and_update(:mongo, "users",
  %{"email" => "someone@somewhere.com"},
  %{"$set" => user_update_changeset.changes},
          [return_document: :after])
```

Note the “changes”. Ecto has a struct and ‘changes’ gives you a map of the changes.

Bson Date Format

Here’s a routine to get time_now into BSON format for MongoDb.

It’s always a good idea to use Date formats in BSON format so that MongoDB understands them. This will help when doing queries, etc.

```
defmodule MyApp.BsonTime do

  epoch = {{1970, 1, 1}, {0, 0, 0}}
  @epoch :calendar.datetime_to_gregorian_seconds(epoch)

  def from_erl_timestamp_to_ecto(erl_ts) do
    timestamp = from_os_timestamp_to_usec(erl_ts)
    ts = trunc(timestamp / 1000000)
    ts = ts + @epoch
    ts_tuple = :calendar.gregorian_seconds_to_datetime(ts)

    # now we need remainder
    us = rem(timestamp,1000) * 1000
    {{y, m, d}, {h, mn, s}} = ts_tuple
    {{y, m, d}, {h, mn, s, us}}
  end

  def from_os_timestamp_to_usec({megasecs,secs,microsecs}) do
        (megasecs*1000000 + secs)*1000000 + microsecs
  end

  def bson_time_now() do
    now = :os.timestamp()
    now_ecto = from_erl_timestamp_to_ecto(now)
    BSON.DateTime.from_datetime(now_ecto)
  end
end
```

And if you have a BSON date returned from Mongo, you probably want to render it in a view, or serialize it to JSON. Here's what I do:

```
updated_at = BSON.DateTime.to_iso8601(user["updated_at"])
```

In my schemas, I didn't have a BSON.DateTime, so I specified a string.

```
  field :created_at,   :string
```

and use it like this:

```
user[:created_at] = MyApp.BsonTime.bson_time_now()
```

Since we are not using Ecto's persistence, you won't get the automatic timestamp features of Ecto. In this case the easiest thing is just to follow the pattern of using a business logic layer to do this.

If we can make a Mongoid like driver, we could have automatic management of the inserted_at, and updated_at, for example. But then again MongoDb's Document _id has a timestamp already embedded in it.

Mix Integrations

Migrations

Just kidding! ;) MongoDb doesn't have them.

Collections come into existence the instant you try to insert to one. And fields are present on any document where you write one.

The only thing you should do on startup is ensureIndex (see above)

Reset

I needed database reset and seed. The ecto mix tasks are not going to work, so I just made my own.

```
defmodule Mix.Tasks.MyApp.Reset do
  use Mix.Task

  @shortdoc "Resets all data in database"

  @moduledoc """
    There might be a better way to do this!

    You have to list all your collections listed.
  """

  def run(_args) do
    Mix.Task.run "app.start", []
    Mix.shell.info "Reset all data"
    Mongo.delete_many(:mongo, "users", %{})
    Mongo.delete_many(:mongo, "ponies", %{})
  end
end
```

This will not drop the indexes. So if you need to drop an index, you can do it from the MongoDb console with db.users.dropIndexes()

Seed

For seeding, again I just create a little mix task. Our devops guy hates this because now we have to special case seeding the services that use MongoDb instead of PostgreSQL. But it's just one line of code.

```
  mix my_app.reset
  mix my_app.seed
```

and this

```
defmodule Mix.Tasks.MyApp.Seed do
  use Mix.Task

  @shortdoc "Seeds "
  @moduledoc """

  """


  def run(_args) do
    Mix.shell.info "Seeding users -- starting application--"
    Mix.Task.run "app.start", []

    user = %{
      email: "someone@somewhere.com"
    }

    Mongo.insert_one(:mongo, "users", user)
  end
end
```

Caveats

Don’t mix atoms and string-keys in maps. The driver doesn’t like that.

That’s another reason to use Ecto Changesets, everything will end up as atoms, even if you have string keys.

Otherwise resort to walking the Map and getting it all fixed. There are examples in Phoenix on how to Enum a Map and convert atoms to strings and vice-versa.

Some notes

Thanks friends for posting comments. There were actually a bunch of things I forgot to mention.

- remove the mongodb_ecto from the `mix.exs` and then add the elixir mongodb driver as I’ve shown above.

- You can delete Repo.ex since you won't need it.
- Make sure you add the startup code above in my_app.ex. (note: Don't start (remove) MyApp.Repo!)
- You also can remove `Ecto.Adapters.SQL.Sandbox` stuff from test/support/conn_case.ex, etc.
- For tests I just use delete_many before (or after) I setup a test since there is no transaction feature in Mongodb anyways. (the database is set to test and started automatically if you follow my instructions above).

Conclusion

The MongoDb drivers for Elixir have proven very reliable. It also support replica sets now. And it also works well with Phoenix.

All you need is a little bit of configuration and you'll be good to go.

If you liked this article, have some corrections or additions, send me an email or pull-request. I'm sure other developers have done some similar solutions.

Proposal

It would be great if there were something akin to Rails's Mongoid drivers in Elixir/Phoenix. It would grab the best parts of Ecto (or be a branch) and provide handy features for dealing with the various times, dates, timestamps, ids etc., and embedded documents.

Tags: Elixir, Phoenix, Ecto2, MongoDb

(ht… (ht… (ht…

← **PREVIOUS POST (/2017-02-07-RIGHTS-MANAGEMENT-API/)**

NEXT POST → (/2017-02-09-MERGING-TWO-LISTS-IN-ELIXIR/)

20 Comments **tomjoro-github-io** 1 Login

Recommend 1 Tweet Share Sort by Best

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Vyacheslav Voronchuk • 2 years ago

Hi, we decided just to map Ecto.Repo pattern backed with low-level driver: https://github.com/eyrmedic...

1 • Reply • Share ›

tomjoro Mod → Vyacheslav Voronchuk • 2 years ago

Thanks, very nice - I hadn't seen that. Yours is a similar approach, but you have done much better with integrating with Repo and queries. Also, would be interesting to look at how to do flexible schema + schema, i.e. some changes are in changeset, others are not. I also have examples of find_and_update, etc. and adding indexes, etc. which are common questions. As I mentioned in conclusion I think it would be really cool to create some sort of driver, similar to Mongoid in Ruby (would this need to be an Ecto fork?)

• Reply • Share ›

Vyacheslav Voronchuk → tomjoro • 2 years ago

I think that it should be possible even without forking Ecto, because in custom Ecto.Repo methods we can decide how to map data in / out of MongoDb, so we can ignore schemaless data on Ecto level and manage it directly with low level driver, possibly writing special helper module for this. Then we were migrating our data from Meteor Mongo collections to Phoenix, that's how we did that. I can't give any input on Mongoid itself because I'm not familiar with it's infrastructure.

• Reply • Share ›

Shaun Chua • 10 months ago

Hi, I'm using phoenix 1.3 to execute this. I'm getting this error:

```
=INFO REPORT==== 7-Jan-2018::21:41:58 ===
application: logger
exited: stopped
type: temporary
** (Mix) Could not start application mongo_two: exited in: MongoTwo.Application.start(:normal, [])
** (EXIT) exited in: GenServer.call(:mongo, :topology, 5000)
** (EXIT) no process: the process is not alive or there's no process currently associated with the given name, possibly because its application isn't started
```

I'm not sure what this means.

Also there is another file called application.ex generated by phoenix 1.3 that seems to house the code that you wrote for my_app.ex in your example. So that's where I put your my_app.ex code snippet

• Reply • Share ›

muthu hari • 2 years ago

Hello Tom,

Is it possible to have schema like this instead of array i want to store as document

schema "users" do
field :name, :string
field :email, :string
field :address :document
end

As u told if create separate module called address when user update some info i have to check both changesets validation separately before update, Is there is any way to get check changeset errors as same time??

• Reply • Share ›

Marcos Benatti • 2 years ago

Nice post, thank you,

But what's the problem with mongodb_ecto adapter?

• Reply • Share ›

tomjoro Mod → Marcos Benatti • 2 years ago

Ecto 1.0 supported MongoDb, Ecto 2.0 & 3.0 don't. This has created a lot of confusion for developers who want to use MongoDb with Phoenix web applications. Don't downgrade to earlier versions of Ecto because you want to use MongoDb with Phoenix! You can use the newest versions of Ecto with the MongoDb drivers with a small amount of code.

• Reply • Share ›

Marcos Benatti → tomjoro • 2 years ago

But we have an updated repo with mongodb ecto adapter updated to v 2.0 https://github.com/michalmu...

• Reply • Share ›

alan blount → Marcos Benatti • a year ago

I have just setup `mongodb_ecto` on a branch, and it worked with `phx 1.3-rc2`
```{:mongodb_ecto, github: "michalmuskala/mongodb_ecto", branch: "ecto-2.1"}```

Reply • Share ›

Frank McGeough • 2 years ago

Thanks for this write-up. I'm still trying to understand the details of the driver. The github repo seems to indicate that you need : , pool: DBConnection.Poolboy in order to have pooling. what does "Mongodb is started with the database pool" mean since you aren't passing in the pool parameter. Thanks again for the write-up.

Reply • Share ›

tomjoro Mod → Frank McGeough • 2 years ago

You're correct. I'll update the article. I'm not starting it with the pool

Reply • Share ›

Kabaji Egara Jr. • 2 years ago

Hello Tom,

What if i want to represent an array in the document for example

field :last_name, :string
field :status, :string
field :following :array

is this possible?

Reply • Share ›

tomjoro Mod → Kabaji Egara Jr. • 2 years ago

Yes. Read the section on "Embedded documents" above. In this case I'm saving an array of addresses like so:

{:ok, result} = Mongo.find_one_and_update(:mongo, "users",
%{"email" => "someone@somewhere.com"},
%{"$push" => %{ addresses: one_address },
[return_document: :after])

I'm identifying the user from their email in this case, and pushing a single address to the addresses array.

But you won't be able to do that "addresses" with an ecto mapping in Users. What I did is create a separate module (schema) for "Address" and then verify the changeset fields before pushing into the addresses array in a user.

The main point of the changeset is for you to verify the data in the Address. The fact that you don't have a one-to-many type of mapping really doesn't add much complexity in Mongo's case because Mongo doesn't have a join anyways!

Reply • Share ›

svsdehh • 2 years ago

>The MongoDb drivers for Elixir have proven very reliable. It also support replica sets now.

as far as I now, it does not support that up to now. Do you have an example?

^ | ˅ • Reply • Share ›

tomjoro Mod ➦ svsdehh • 2 years ago

Thanks, you're correct. I'll update the article. The feature is currently being tested, so hopefully it'll be released soon https://github.com/ericmj/m... . Help out if you have time.

^ | ˅ • Reply • Share ›

Mark Windrim • 2 years ago

Hi, great article. I've been following it, and ran into an issue when I tried to start up the application. You mention that we're not going to use the Repo part of Ecto, but I still have a repo.ex file that fails at startup. It is complaining about the lack of an adapter specification.

I started with phoenix.new app --database=mongodb

Should I also specify --no-ecto, or have I messed up something else?

Thanks

^ | ˅ • Reply • Share ›

tomjoro Mod ➦ Mark Windrim • 2 years ago

No, that's fine.

* remove the mongodb_ecto from the `mix.exs` and then add the elixir mongodb driver as I've shown above.
* You can delete Repo.ex since you won't need it.
* Make sure you add the startup code above in my_app.ex. (note: Don't start (remove) MyApp.Repo!)
* You also can remove `Ecto.Adapters.SQL.Sandbox` stuff from test/support/conn_case.ex, etc.
* For tests I just use delete_many before (or after) I setup a test since there is no transaction feature in Mongodb anyways. (the database is set to test and started automatically if you follow my instructions above).

Phoenix is really easy to `refactor` dependencies, so just dive in and don't worry. Remember we're not using the Repo features of Ecto.

I will update the instructions to include these changes, and maybe include some test cases, or perhaps an entire Github example would help the most. Just write back if you have any issues.

^ | ˅ • Reply • Share ›

Ricardo García Vega • 2 years ago

Great article, congrats! I have a question related to pooling. How do you configure it? I've tried like in the official docs https://github.com/ericmj/m..., but when I deploy to production, where I have a load balancer with three different instances, the mongo process dies eventually, and I'm not really sure about the reason yet.

• Reply • Share ›

tomjoro Mod ➦ Ricardo García Vega • 2 years ago

Thanks!
It's hard to say without help without some more details about your setup. Is it actual system process Mongo that dies? Or the process in the BEAM? You might try without the pool and see if the behavior changes. There is an open issue currently with disconnect, which you might hit if there is a dropped connection causes a reconnect.

1 • Reply • Share ›

Ricardo García Vega ➦ tomjoro • 2 years ago

Thanks for the response!
Being honest I'm not really sure, as I haven't had time enough to test it again, but I'll try to gather more info about the issue as soon as I can.

● (https://www.facebook.com/yourname) ● (https://github.com/tomjoro)
● (https://twitter.com/tomjoro) ● (mailto:tomjoro@gmail.com)

Thomas O'Rourke • 2018

Theme by beautiful-jekyll (http://deanattali.com/beautiful-jekyll/)