Using the CowBoy HTTP server from an Elixir program

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

First publication of this article on 17 September 2019

https://www.bortzmeyer.org/cowboy-elixir.html

Among the people who use the CowBoy <https://ninenines.eu/> HTTP server, some do it from an Erlang program, and some from an Elixir program. The official documentation <https://ninenines.eu/docs/en/cowboy/2.6/guide/> only cares about Erlang. You can find some hints online about how to use CowBoy from Elixir but they are often outdated (CowBoy changed a lot), or assume that you use CowBoy with a library like Plug <https://hexdocs.pm/plug/readme.html> or a framework like Phoenix <https://hexdocs.pm/phoenix/>. Therefore, I document here how I use plain CowBoy, from Elixir programs, because it may help. This is with Elixir 1.9.1 and CowBoy 2.6.3.

I do not find a way to run CowBoy without the mix tool <https://hexdocs.pm/mix/>. So, I start with mix :

```
% mix new myserver
...
Your Mix project was created successfully.
```

I then add CowBoy dependencies to the `mix.exs` file :

```
 defp deps do
    [
      {:cowboy, "~> 2.6.0"}
    ]
 end
```

(Remember that CowBoy changes a lot, and a lot of CowBoy examples you find online are for old versions. Version number is important. I used 2.6.3 for the examples here.) Then, get the dependencies :

```
% mix deps.get
...
  cowboy 2.6.3
  cowlib 2.7.3
  ranch 1.7.1
...
```

We can now fill `lib/myserver.ex` with the main code :

```
 defmodule Myserver do

  def start() do
    dispatch_config = build_dispatch_config()
    { :ok, _ } = :cowboy.start_clear(:http,
      [{:port, 8080}],
      %{ env: %{dispatch: dispatch_config}}
    )
  end

  def build_dispatch_config do
    :cowboy_router.compile([
      { :_,
        [
          {"/", :cowboy_static, {:file, "/tmp/index.html"}}
        ]}
    ])
  end

end
```

And that's all. Let's test it :

```
% iex -S mix
Erlang/OTP 22 [erts-10.4.4] [source] [64-bit] [smp:4:4] [ds:4:4:10] [async-threads:1]

Compiling 1 file (.ex)
Interactive Elixir (1.9.1) - press Ctrl+C to exit (type h() ENTER for help)

iex(1)> Myserver.start()
{:ok, #PID<0.197.0>}

iex(2)>
```

If you have HTML code in `/tmp/index.html`, you can now use any HTTP client such as curl, lynx or another browser, to visit `http://localhost:8080/`.

The `start_clear` routine (which was `start_http` in former versions) starts HTTP (see its documentation `<https://ninenines.eu/docs/en/cowboy/2.6/manual/cowboy.start_clear/>`.) If you want explanation about the behaviour `:cowboy_static` and its parameters like `:file`, see the CowBoy documentation `<http://ninenines.eu/docs/en/cowboy/2.6/manual/cowboy_static/>`. If you are interested in routes (the argument of `:cowboy_router.compile`, directives for CowBoy telling it "if the request is for `/this`, then do `that`"), see also the documentation `<https://ninenines.eu/docs/en/cowboy/2.6/manual/cowboy_router/>`. There are many other possibilities, for instance, we could serve an entire directory :

```
def build_dispatch_config do
   :cowboy_router.compile([

     { :_,
       [
         # Serve a single static file on the route "/".
         {"/", :cowboy_static, {:file, "/tmp/index.html"}},

         # Serve all static files in a directory.
         # PathMatch is "/static/[...]" -- string at [...] will be used to look up the file
         # Directory static_files is relative to the directory where you run the server
         {"/static/[...]", :cowboy_static, {:dir, "static_files"}}
       ]}
   ])
 end
```

You can now start from this and improve :
— Use `start_tls` instead of `start_clear`, to provide security through HTTPS,
— Replace `def start() do` by `def start(_type, _args) do` (or `def start(type, args) do` if you want to use the parameters) to follow OTP conventions, in order for instance to run the HTTP server under a supervisor (see this example `<https://gist.github.com/mpugach/9093f092f63e5b91f08a8ac116684d66>` - untested), or simply to create a standalone application,
— Serve dynamic content by using Elixir (or Erlang) modules to produce content on the fly.