Blog

Written words that are often helpful

Doing Elixir Development in Windows using Docker

Posted by chathaway on April 12, 2018, 2:23 a.m.

This guide documents the steps I took to setup my Elixir/Phoenix/Docker environment on my Windows 10 machine.It might be worth nothing that I'm running Windows 10 Educational, which includes the Hyper-V supervisor; this is required for Docker.

Install Docker

Rather than repeat stuff here: go search Google, and follow the official guide.

This will probably require a restart.

Setup a Dockerfile and docker-compose file

To get started, we need a Dockerfile to build the web host.In addition to elixir, we will need NodeJS and some extras.This should do it:

```
FROM elixirADD . /codeWORKDIR /codeRUN mix archive.install --force https://github.com/phoenixframework/archives/raw/master/phx
```

This will build a basic Docker image for us to work out of; we will have to add the commands to download and install dependencies later.

And we need the docker-compose.yml file:

```
version: '3'services:  web:    build: .    ports:      - "4000:4000"    volumes:      - .:/code    depends_on:      - db  db:
```

Good? Good. Now, build the image

```
< from the directory with the Dockerfile >C:\Users\Charles\Programming\phoenix> docker build .Sending build context to Docker
```

Bring up the instance in one terminal, then connect to it in another.

```
C:\Users\Charles\Programming\phoenix>docker-compose up< Lots of output >web_1  | Interactive Elixir (1.6.4) - press Ctrl+C to
```

Great! Now, let's generate our Phoenix project and get started!

Connect to the docker instance and follow the "Up and running" guide

To connect to the Docker instance in a shell, first run this command to find the instance ID:

```
C:\Users\Charles\Programming\mtb>docker psCONTAINER ID        IMAGE               COMMAND                  CREATED
```

We are interested in the phoenix_web image (the name may vary), specifically the CONTAINER ID (in this case, 3f50...)

```
C:\Users\Charles\Programming\mtb> docker exec -ti 3f50c1a9ba4a /bin/bashroot@3f50c1a9ba4a:/code##
```

Huzzah! A Linux prompt! Let the fun begin; follow the Phoenix setup directions, with one key caveat; when running `mix phx.new`, use this: `mix phx.new . --module MyApp` and type 'y' when prompted.

Once the app is generated, lets modify the Dockerfile to pull in the dependencies as needed.At the time of writing, Phoenix was still using Brunch, but promises a transition to webpack soon.

new Dockerfile:

```
FROM elixirADD . /codeWORKDIR /codeRUN mix archive.install --force https://github.com/phoenixframework/archives/raw/master/phx
```

Bring down the old instance by hitting "Ctrl+C" in that terminal, then rerun `docker build .`.

Setup the database

Last thing, I swear.By default, Phoenix is configred to look at localhost for the database.In Docker, the database can be found at hostname db; we need to update config/dev.exs (towards the bottom, under the comment saying "Configure your database").We also need to remove the password, since the Docker postgres image doesn't ship with one.

After your image is up, you should run `mix ecto.create` from inside the instance.This can not be done in the Dockerfile since it requires the database host to be alive.Migrating your database will also become part of your day-to-day activities, so plan on learning the commands anyway :).

Change the default run command

Right now, the image is setup to IEX when it comes up.We need it to do a bit more than that.We want it to run `iex -S mix phx.server` when it comes online.To do this, modify docker-compose.yml:

```
version: '3'services:  web:    build: .    command: iex -S mix phx.server    ports:      - "4000:4000"    volumes:      - .:/c
```

Hopefully you now have a prompt to IEX>, and if you open a web browser, you can access http://localhost:4000/

The Good Stuff

One of the really cool things Elixir can do that others can not is hot-code reloading, and allowing multiple command interfaces for a single node (to inspect state and what not).To get that working, you need to setup a session name and a cookie for each session which is going operate in the same "universe".To do this, pass the `--sname <somename>` and `--cookie <some secret>` options to iex, like so (in docker-compose.yml):

```
    command: iex --sname compose --cookie nom -S mix phx.server
```

Then, from a shell, run:

```
root@d04082431ec9:/code## iex --sname client --cookie nom --remsh compose@d04082431ec9Erlang/OTP 20 [erts-9.3] [source] [64-bi
```

And you can do cool stuff.

Thanks for reading, feel free to comment!
