nimble

Elixir Project Template and Conventions

Ros - Tulsi - An - Micky - Olivier

Growth Session #30 - August 21 2020

Intro

Create our Elixir project template 🚀

Document our Elixir conventions on Compass

Goals

CLI Templatization

Unit testing

UI testing

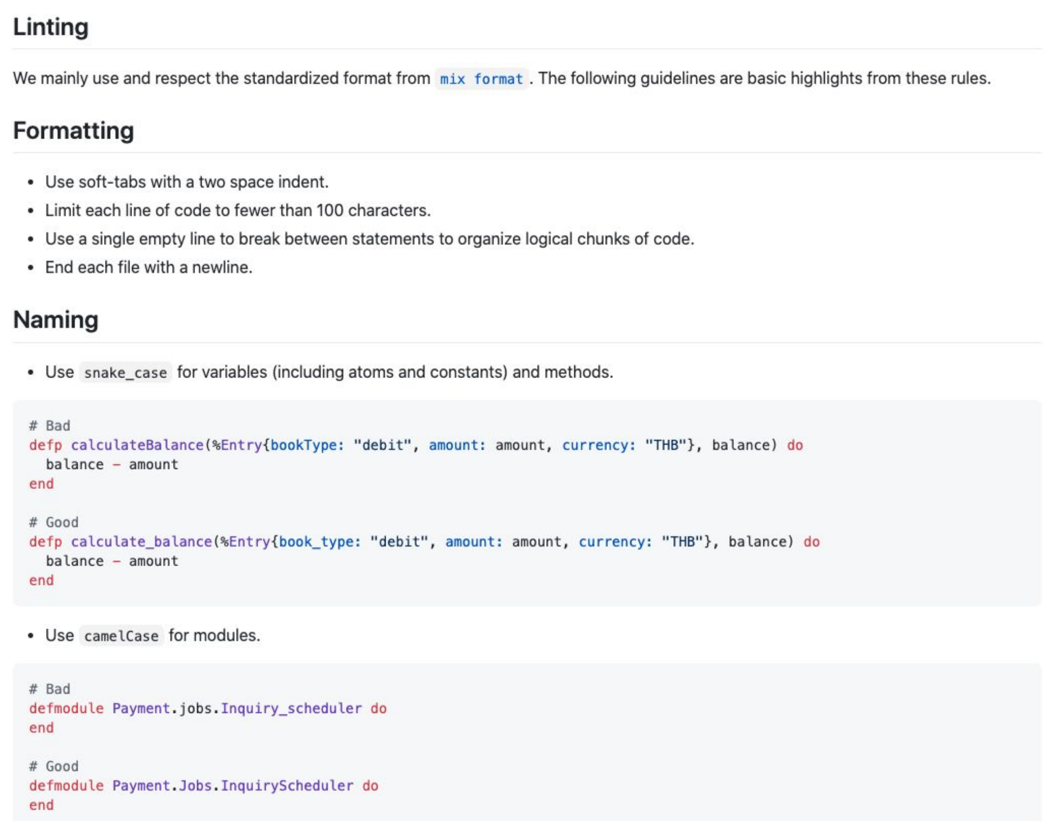
Docker + Docker Compose

Github Actions

Achievement and Progress

What we achieved in this session

Compass



Linting

We mainly use and respect the standardized format from `mix format`. The following guidelines are basic highlights from these rules.

Formatting

- Use soft-tabs with a two space indent.
- Limit each line of code to fewer than 100 characters.
- Use a single empty line to break between statements to organize logical chunks of code.
- End each file with a newline.

Naming

- Use `snake_case` for variables (including atoms and constants) and methods.

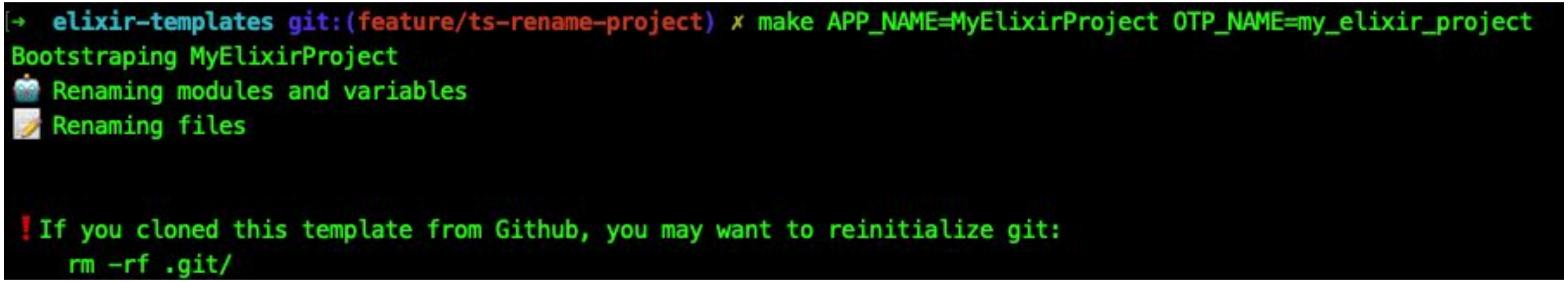
```
# Bad
defp calculateBalance(%Entry{bookType: "debit", amount: amount, currency: "THB"}, balance) do
  balance - amount
end

# Good
defp calculate_balance(%Entry{book_type: "debit", amount: amount, currency: "THB"}, balance) do
  balance - amount
end
```

- Use `camelCase` for modules.

```
# Bad
defmodule Payment.jobs.Inquiry_scheduler do
end

# Good
defmodule Payment.Jobs.InquiryScheduler do
end
```

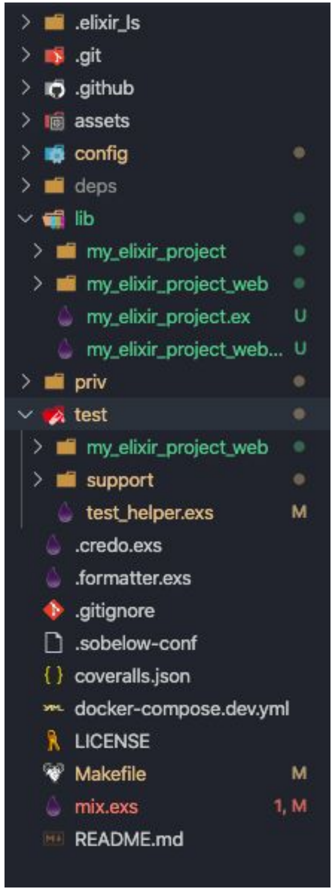
https://github.com/nimblehq/compass/pull/327

CLI Templatization

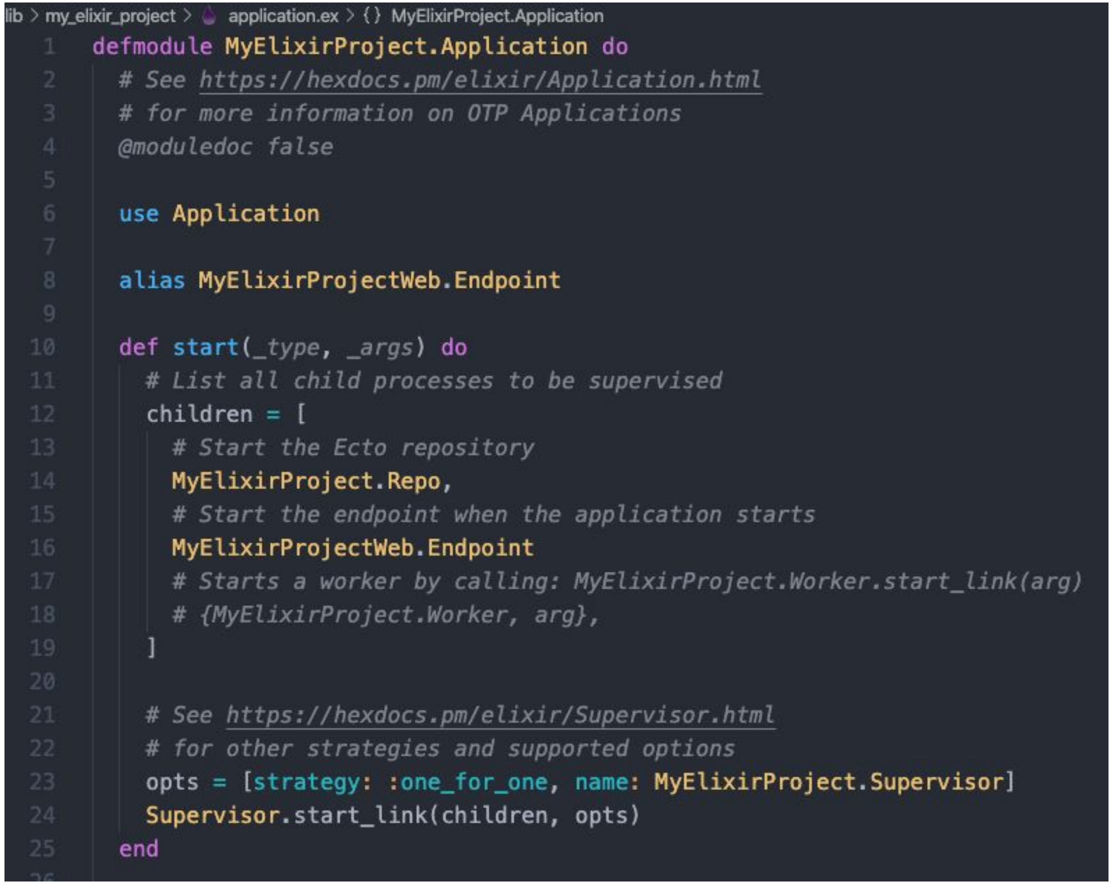


```
→  elixir-templates git:(feature/ts-rename-project) ✗ make APP_NAME=MyElixirProject OTP_NAME=my_elixir_project
Bootstraping MyElixirProject
🤖 Renaming modules and variables
📝 Renaming files


❗If you cloned this template from Github, you may want to reinitialize git:
    rm -rf .git/
```

CLI Templatization



```
> .elixir_ls
> .git
> .github
> assets
> config
> deps
v lib
  > my_elixir_project
  > my_elixir_project_web
    my_elixir_project.ex          U
    my_elixir_project_web...      U
> priv
v test
  > my_elixir_project_web
  > support
    test_helper.exs               M
  .credo.exs
  .formatter.exs
  .gitignore
  .sobelow-conf
  coveralls.json
  docker-compose.dev.yml
  LICENSE
  Makefile                        M
  mix.exs                         1, M
  README.md
```



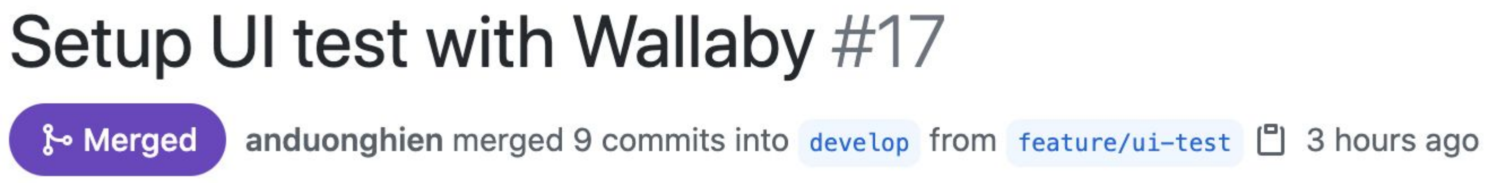
lib > my_elixir_project > application.ex > { } MyElixirProject.Application

```elixir
defmodule MyElixirProject.Application do
  # See https://hexdocs.pm/elixir/Application.html
  # for more information on OTP Applications
  @moduledoc false

  use Application

  alias MyElixirProjectWeb.Endpoint

  def start(_type, _args) do
    # List all child processes to be supervised
    children = [
      # Start the Ecto repository
      MyElixirProject.Repo,
      # Start the endpoint when the application starts
      MyElixirProjectWeb.Endpoint
      # Starts a worker by calling: MyElixirProject.Worker.start_link(arg)
      # {MyElixirProject.Worker, arg},
    ]

    # See https://hexdocs.pm/elixir/Supervisor.html
    # for other strategies and supported options
    opts = [strategy: :one_for_one, name: MyElixirProject.Supervisor]
    Supervisor.start_link(children, opts)
  end
```

Unit testing

UI Testing

- We are using Wallaby for UI test


Setup UI test with Wallaby #17

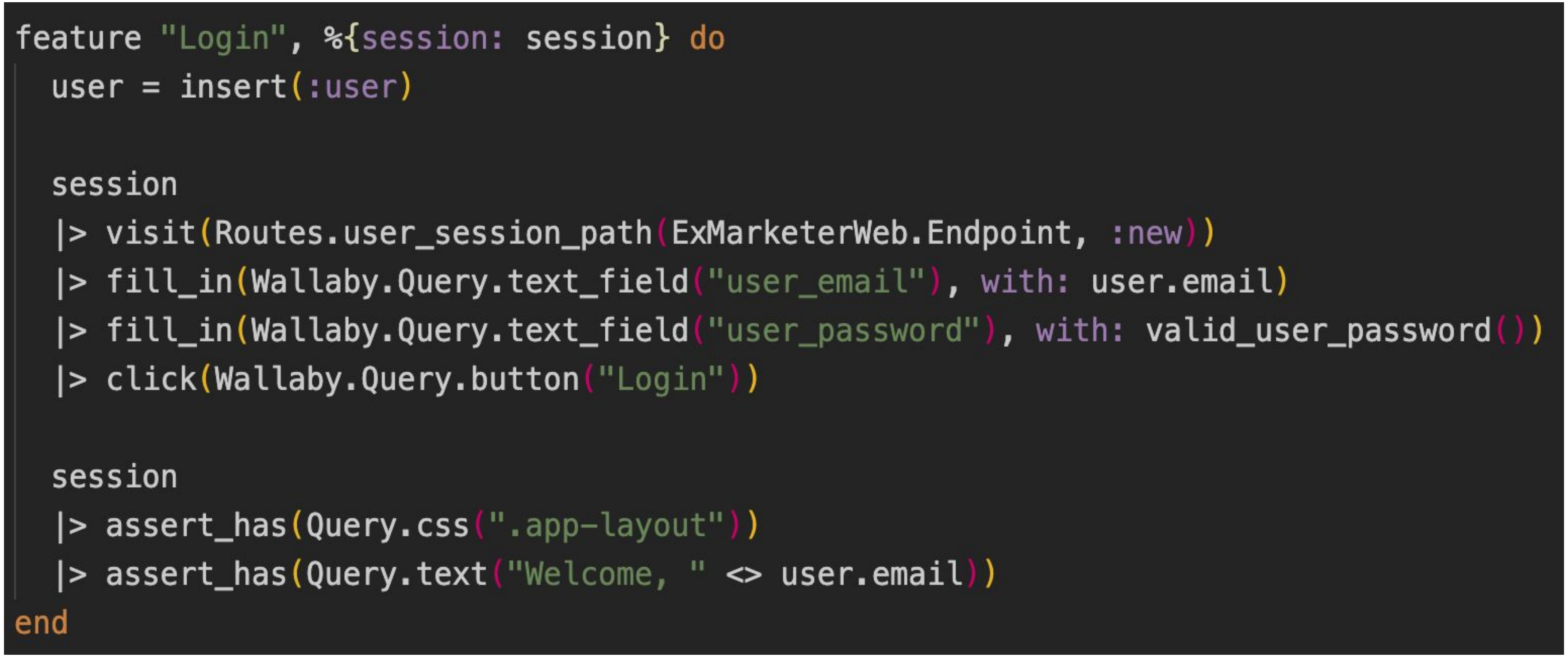
Merged anduonghien merged 9 commits into develop from feature/ui-test 3 hours ago

- As Elixir is a concurrency language so the UI test can be run as concurrency, each UI test process will have it own database connection

- Wallaby supports multiple browser sessions in one test case, eg: Testing the chatting feature, we could open 2 or 3 browser sessions.

UI Testing

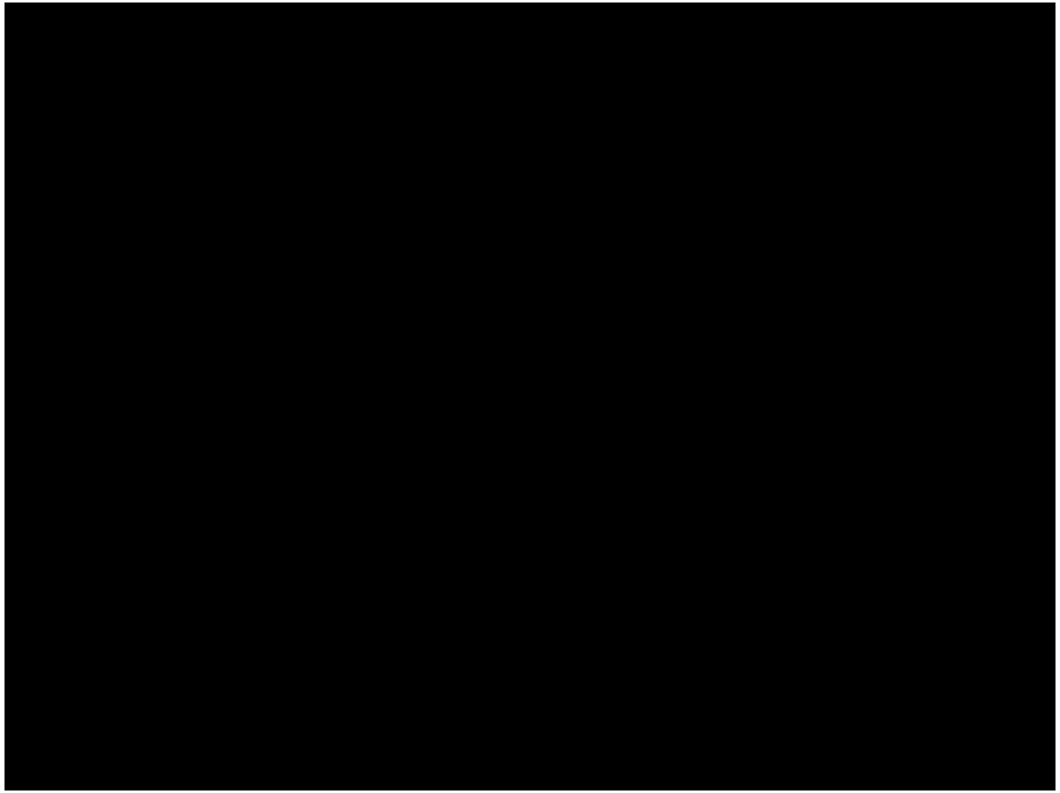
Wallaby Syntax



```elixir
feature "Login", %{session: session} do
  user = insert(:user)

  session
  |> visit(Routes.user_session_path(ExMarketerWeb.Endpoint, :new))
  |> fill_in(Wallaby.Query.text_field("user_email"), with: user.email)
  |> fill_in(Wallaby.Query.text_field("user_password"), with: valid_user_password())
  |> click(Wallaby.Query.button("Login"))

  session
  |> assert_has(Query.css(".app-layout"))
  |> assert_has(Query.text("Welcome, " <> user.email))
end
```

UI Testing

Multiple browser sessions per test case



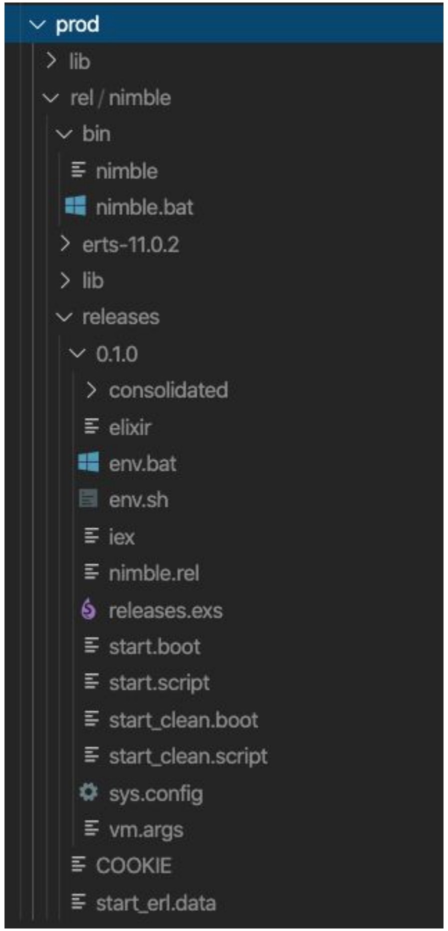
Docker

Environment	
Development	Only for database
Test	⛔⛔⛔
Production	✅✅✅

Docker for production

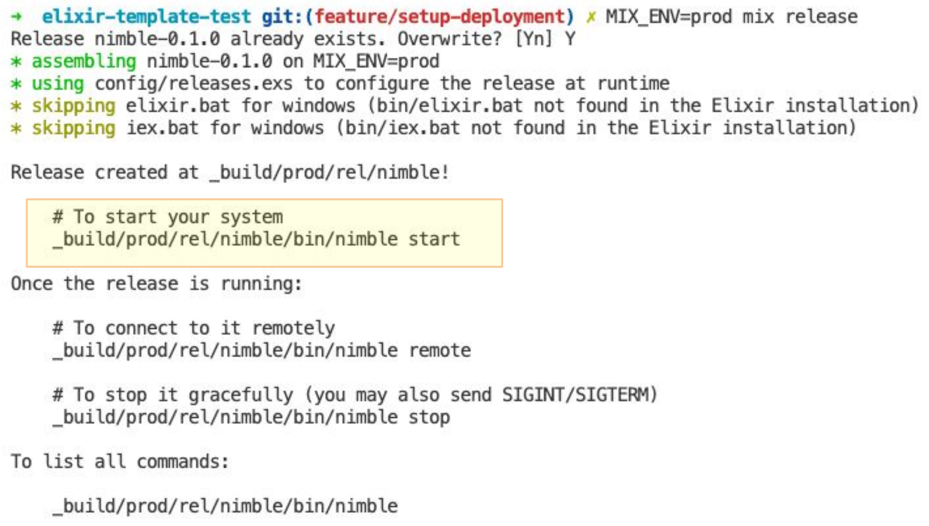
- Release on the machine that has the same OS as the deployment target machine
  - Erlang still depending on OS
  - Docker becomes useful to make sure that build environment is the same as running environment
- Easy to keep track with environment variables
- Easy to maintain and manage the production environment and tools

Releasing App



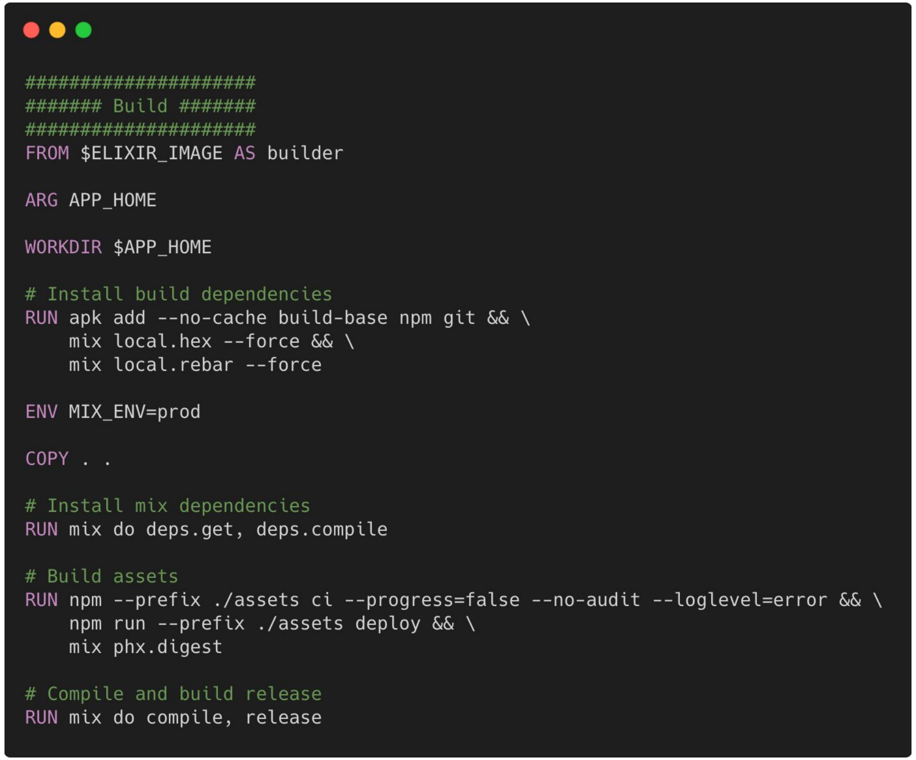
```
prod
  lib
  rel / nimble
    bin
      nimble
      nimble.bat
    erts-11.0.2
    lib
    releases
      0.1.0
        consolidated
        elixir
        env.bat
        env.sh
        iex
        nimble.rel
        releases.exs
        start.boot
        start.script
        start_clean.boot
        start_clean.script
        sys.config
        vm.args
      COOKIE
      start_erl.data
```

- Self-contained (Erlang + Erlang Runtime + Elixir + compiled code)
- Preload the code in embedded mode
- Provide useful scripts to manage the app

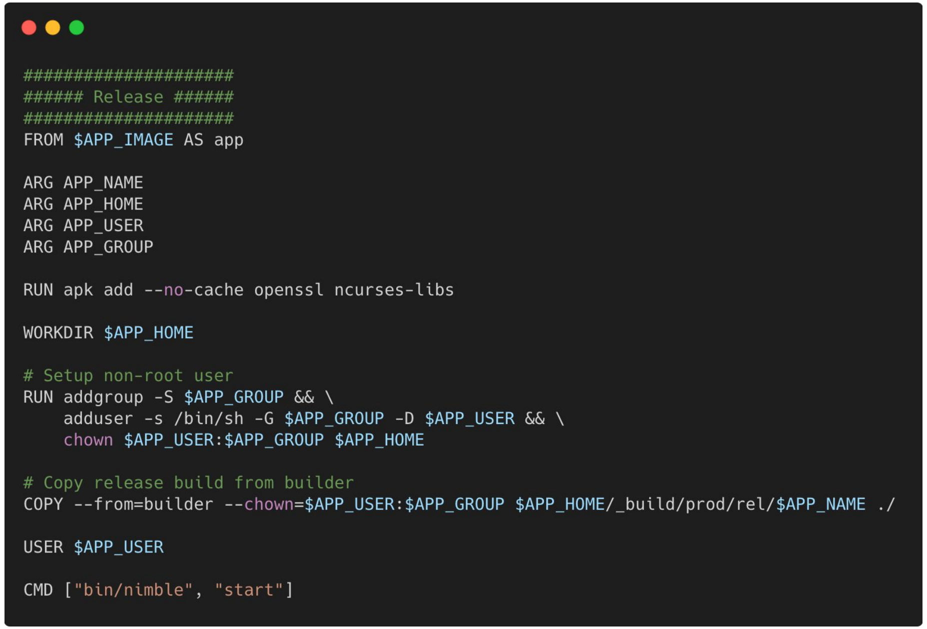


```
➜  elixir-template-test git:(feature/setup-deployment) ✗ MIX_ENV=prod mix release
Release nimble-0.1.0 already exists. Overwrite? [Yn] Y
* assembling nimble-0.1.0 on MIX_ENV=prod
* using config/releases.exs to configure the release at runtime
* skipping elixir.bat for windows (bin/elixir.bat not found in the Elixir installation)
* skipping iex.bat for windows (bin/iex.bat not found in the Elixir installation)

Release created at _build/prod/rel/nimble!

    # To start your system
    _build/prod/rel/nimble/bin/nimble start

Once the release is running:

    # To connect to it remotely
    _build/prod/rel/nimble/bin/nimble remote

    # To stop it gracefully (you may also send SIGINT/SIGTERM)
    _build/prod/rel/nimble/bin/nimble stop

To list all commands:

    _build/prod/rel/nimble/bin/nimble
```

Dockerfile



```
#####################
####### Build #######
#####################
FROM $ELIXIR_IMAGE AS builder

ARG APP_HOME

WORKDIR $APP_HOME

# Install build dependencies
RUN apk add --no-cache build-base npm git && \
    mix local.hex --force && \
    mix local.rebar --force

ENV MIX_ENV=prod

COPY . .

# Install mix dependencies
RUN mix do deps.get, deps.compile

# Build assets
RUN npm --prefix ./assets ci --progress=false --no-audit --loglevel=error && \
    npm run --prefix ./assets deploy && \
    mix phx.digest

# Compile and build release
RUN mix do compile, release
```



```
#####################
###### Release ######
#####################
FROM $APP_IMAGE AS app

ARG APP_NAME
ARG APP_HOME
ARG APP_USER
ARG APP_GROUP

RUN apk add --no-cache openssl ncurses-libs

WORKDIR $APP_HOME

# Setup non-root user
RUN addgroup -S $APP_GROUP && \
    adduser -s /bin/sh -G $APP_GROUP -D $APP_USER && \
    chown $APP_USER:$APP_GROUP $APP_HOME

# Copy release build from builder
COPY --from=builder --chown=$APP_USER:$APP_GROUP $APP_HOME/_build/prod/rel/$APP_NAME ./

USER $APP_USER

CMD ["bin/nimble", "start"]
```

Added best practices around Elixir through compass and template :)

Thanks!

Contact Nimble

nimblehq.co

hello@nimblehq.co

Bangkok

399 Interchange 21 Sukhumvit Road, Unit
#2402-03, Klong Toei, Wattana, Bangkok
10110, Thailand

Singapore

28C Stanley St, Singapore 068737

Hong Kong

20th Floor, Central Tower
28 Queen's Road, Central, Hong Kong

nimble