Passo a passo - Deploy Rails + Docker

Docker com NGINX, Postgres, Puma e Rails

======= NGINX ========

Criar a estrutura de pastas e arquivos abaixo:

- `nginx_postgres_puma_rails/`
  - `docker-compose.yml`

Adicionar a imagem do NGINX
environments/development/docker-compose-yml

```
version: '3'

services:
  web:
    image: nginx
    ports:
    - 80:80
```

Testar a imagem:

`docker-compose up -d`
- acessar http://127.0.0.1 e ver NGINX
`docker-compose down`

Ajustar o arquivo Hosts

- configurar o hosts para **http://myapp.local/**
>> https://www.howtogeek.com/howto/27350/beginner-geek-how-to-edit-your-hosts-file/
- acessar **http://myapp.local/** e ver NGINX

======= POSTGRES ========

Adicionar a imagem do Postgres

environments/development/docker-compose-yml

```
version: '3'

services:
  web:
    image: nginx
    ports:
    - 80:80
  db:
    image: postgres
    restart: always
    environment:
      POSTGRES_PASSWORD: mypassword
    volumes:
      - ./postgres:/var/lib/postgresql/data
    ports:
      - 5432
```

Acessar o Postgres

- Acessar via pgAdmin **myapp.local:<PORTA MAPEADA>**

======= RAILS =======

Criar a estrutura de pastas e arquivos abaixo:

- `nginx_postgres_puma_rails`
  - `postgres/`
  - `rails_app/`
    - `Gemfile`
    - `Gemfile.lock`
  - `Dockerfile`
  - `docker-compose.yml`

Adicionar o conteúdo abaixo ao arquivo **railsapp/Gemfile**

```
source 'https://rubygems.org'
gem 'rails', '5.1.4'
```

Adicionar o conteúdo abaixo ao arquivo **Dockerfile**

```
FROM ruby:2.4-jessie
RUN apt-get update -qq && apt-get install -y build-essential libpq-dev
nodejs
RUN mkdir /rails_app
WORKDIR /rails_app
COPY rails_app/Gemfile /rails_app/Gemfile
COPY rails_app/Gemfile.lock /rails_app/Gemfile.lock
RUN bundle install
COPY rails_app /rails_app
EXPOSE 3000
CMD [ "bundle", "exec", "puma", "-C", "config/puma.rb" ]
```

Adicionar o build do Ruby/Rails

docker-compose-yml

```
version: '3'

services:
  web:
    image: nginx
    ports:
    - 80:80
  db:
    image: postgres
    restart: always
    environment:
      POSTGRES_PASSWORD: mypassword
    volumes:
      - ./postgres:/var/lib/postgresql/data
```

```
    ports:
      - 5432
  app:
    build: .
    volumes:
      - ./rails_app:/rails_app
    ports:
      - 3000:3000
    depends_on:
      - db
```

Criar a aplicação

```
docker-compose run app bundle exec rails new . -d postgresql
```

Fazer um novo build com o novo Gemfile

```
docker-compose build
```

Alterar o **rails_app/config/database.yml**

```
default: &default
  adapter: postgresql
  encoding: unicode
  host: db
  username: postgres
  password: mypassword
  pool: 5

development:
  <<: *default
  database: myapp_development

test:
  <<: *default
  database: myapp_test

production:
  <<: *default
  database: myapp_production
```

Criar os BD

```
docker-compose run app bundle exec rails db:create
```

Criar um scaffold User

```
docker-compose run app bundle exec rails g scaffold User name:string email:string
```

Fazer a migração

```
docker-compose run app bundle exec rails db:migrate
```

Criar um controller Home

```
docker-compose run app bundle exec rails g controller home index
```

Configurar a rota padrão

```
root to: 'home#index'
```

Alterar a **views/home/index.html** adicionando...

```
<h1>Seja Bem-Vindo!</h1>
<hr/>
<ul>
  <li>
    <%= link_to "Cadastro de Usuários", users_path %>
  </li>
</ul>
<hr/>
<p>Você está em ambiente de [<strong><%= Rails.env %></strong>]</p>
```

Fazer um novo build

```
docker-compose build
```

Testar em ambiente de desenvolvimento **localhost:3000** / **myapp.local:3000** / Verificar que está em modo de **desenvolvimento**

```
docker-compose up -d
docker-compose down
```

======= Arquivo .env ========

Criar arquivo **.env** (https://docs.docker.com/compose/environment-variables/#the-env-file) com conteúdo abaixo

- nginx_postgres_puma_rails
  - postgres/
  - rails_app/
    - Gemfile
    - Gemfile.lock
  - .env
  - Dockerfile
  - docker-compose.yml

Adicionar ao arquivo .env o conteúdo abaixo

```
RAILS_ENV=production
```

Substituir no arquivo docker-compose.yml

docker-compose-yml

```
version: '3'

services:
  web:
    image: nginx
    ports:
    - "80:80"
  db:
    image: postgres
    restart: always
    environment:
      - POSTGRES_PASSWORD: mypassword
    volumes:
      - ./postgres:/var/lib/postgresql/data
    ports:
      - "5432:5432"
  app:
    build: .
    volumes:
      - ./rails_app:/rails_app
    ports:
      - “3000:3000”
    environment:
      - RAILS_ENV=${RAILS_ENV}
    depends_on:
      - db
```

Gerar um secret para o ambiente de produção e coloque-o em **config/secrets.yml**

```
docker-compose run app bundle exec rails secret
```

Precompilar assets para produção

```
docker-compose run app bundle exec rails assets:precompile RAILS_ENV=production
```

Criar base em produção

```
docker-compose run app bundle exec rails db:create RAILS_ENV=production
```

Migrar em produção

```
docker-compose run app bundle exec rails db:migrate RAILS_ENV=production
```

Testar em ambiente de desenvolvimento e produção (alterando o arquivo .env) **localhost:3000** / **myapp.local:3000**

```
docker-compose build
```

```
docker-compose up -d
```

======= LIGANDO O NGINX AO RAILS ========

Adicionar à estrutura de pastas o arquivo app.conf:

- nginx_postgres_puma_rails/
  - .gitignore
  - postgres/
  - nginx/
    - app.conf
    - Dockerfile
  - rails_app/
    - Gemfile
    - Gemfile.lock
  - Dockerfile
  - docker-compose.yml

Adicionar ao arquivo Dockerfile
nginx/Dockerfile

```
FROM nginx
RUN apt-get update -qq && apt-get -y install apache2-utils
ENV RAILS_ROOT /var/www/rails_app
WORKDIR $RAILS_ROOT
RUN mkdir -p log
COPY nginx/app.conf /tmp/docker_example.nginx
RUN envsubst '$RAILS_ROOT' < /tmp/docker_example.nginx >
/etc/nginx/conf.d/default.conf
EXPOSE 80
CMD [ "nginx", "-g", "daemon off;" ]
```

Adicionar ao arquivo app.conf
nginx/app.conf

```
upstream puma_rails_app {
  server app:3000;
}
server {
  listen       80;
  proxy_buffers 64 16k;
  proxy_max_temp_file_size 1024m;
  proxy_connect_timeout 5s;
  proxy_send_timeout 10s;
  proxy_read_timeout 10s;
  location / {
    try_files $uri $uri/ @nginx_rails_app;
  }
  location @nginx_rails_app {
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
```

```
    proxy_set_header Host $http_host;
    proxy_redirect off;
    proxy_pass http://puma_rails_app;
    # limit_req zone=one;
    access_log /var/www/rails_app/log/nginx.access.log;
    error_log /var/www/rails_app/log/nginx.error.log;
  }
}
```

Substituir no arquivo docker-compose.yml
docker-compose-yml

```
version: '3'

services:
  web:
    image: nginx
    build:
      context: .
      dockerfile: ./nginx/Dockerfile
    depends_on:
     - app
    ports:
      - 80:80
  db:
    image: postgres
    restart: always
    environment:
      - POSTGRES_PASSWORD: mypassword
    volumes:
      - ./postgres:/var/lib/postgresql/data
    ports:
      - "5432:5432"
  app:
    build: .
    volumes:
      - ./rails_app:/rails_app
    ports: # remover
      - "3000:3000" # remover
    environment:
      - RAILS_ENV=${RAILS_ENV}
    depends_on:
      - db
```

Fazer um novo build

```
docker-compose build
```

Levantar e acessar a aplicação (lembre-se que o .env controla o ambiente) `docker-compose up -d`

========== DIGITAL OCEAN ==========

Gerar SSH keys

```
ssh-keygen -t rsa -b 4096 -C "your_email@example.com"
```

Alterar .env para production

Adicionar o gitignore à estrutura de pastas e arquivos :

- `nginx_postgres_puma_rails/`
  - `.gitignore`
  - `postgres/`
  - `nginx/`
    - `app.conf`
    - `Dockerfile`
  - `rails_app/`
    - `Gemfile`
    - `Gemfile.lock`
  - `Dockerfile`
  - `docker-compose.yml`

Adicionar o conteúdo abaixo ao arquivo **.gitignore**

```
postgres/
?? rails_app/
```

Criar repositório e efetuar primeiro commit.
Verificar se a pasta realmente foi ignorada.

Acessar Digital Ocean e criar um droplet com Docker

Logar via SSH no Digital Ocean

ssh root@<ip>

Configurar acesso via UFW

```
ufw status
ufw disable
ufw enable
ufw default deny incoming
ufw default allow outgoing
ufw allow 80/tcp
ufw allow 443/tcp
ufw allow 5432/tcp
```

(sudo ufw **deny** 5432/tcp)

Mais informações:

https://www.digitalocean.com/community/tutorials/how-to-setup-a-firewall-with-ufw-on-an-ubuntu-and-debian-cloud-server

Fazer **clone** do repositório git

Rodar comandos de Produção

```
docker-compose run app bundle exec rails assets:precompile RAILS_ENV=production
```

```
docker-compose run app bundle exec rails db:create RAILS_ENV=production
```

```
docker-compose run app bundle exec rails db:migrate RAILS_ENV=production
```

Fazer um build

```
docker-compose build
```

Rodar o `docker-compose up -d`
Acessar o IP do servidor para testar o NGINX
Acessar o servidor via pgAdmin para testar o Postgres

========== WORKFLOW ==========

Desenvolva em (development)
Altere o .env para produção
Faça um build
Faça commit e push

Pare os containers: `docker-compose down`
Faça um pull
Rode as migrações:

```
docker-compose run app bundle exec rails db:migrate RAILS_ENV=production
```

Faça um novo build: `docker-compose build`
Levante os containers: `docker-compose up -d`