Introdução ao Docker

Em resumo, imagem é template que pode ser utilizado.

Container é o processo de criar e executar containers a partir de imagens Docker.

Iniciar o node

```
npm init -y
```

Criar o index.js

- Instalar o express, mysql2 e nodemon

```
npm install express mysql2 nodemon
```

Código index.js

```
const express = require('express')
const mysql2 = require('mysql2');

const PORT = 3000;
const HOST = '0.0.0.0' //Uma forma do docker entender que ele só precisa repassar a porta 3000

const connection = mysql2.createConnection({
    //host: 'database-mysql',
    host: 'localhost',
    user: 'root',
    password: '123',
    database: 'fiap',
  });

  connection.connect((err) => {
    if (err) {
      console.error('Error connecting to MySQL:', err);
      return;
    }
    console.log('Connected to MySQL database');
  });

const app = express()


app.get('/', (req, res) => {
    const query = 'SELECT * FROM products';

    connection.query(query, (err, results, fields) => {
      if (err) {
        console.error('Error executing SELECT query:', err);
        return;
      }

      //console.log('Query results:');
      res.send(results.map(item => ({ name: item.name, price: item.price })));

    });
})


app.listen(PORT, HOST)
```

Criar o dockerignore e o gitignore

Vamos criar tanto o .dockerignore quanto o dockignore para bloquearmos o envio da pasta nodemodule

- Conteudo:

```
node_modules
```

Criar a imagem docker

Criar o dockerfile

Dockerfile

```
# Usar uma imagem existente do node
FROM node:alpine

# Configurar o caminho dentro do container
WORKDIR /usr/src/app

# Copiar os arquivos
COPY package*.json ./

# Executar npm install
RUN npm install

# Copiar o resto da aplicação
COPY . .

# Expor a porta
EXPOSE 3000

CMD ["npm", "start"]
```

Criar a imagem

Vamos criar a imagem a partir do dockerfile. Precisamos executar o local do dockerfile

```
docker build -t app-node .
```

docker build: utilizado para executar o build do dockerfile **-t:**: nome da imagem **. (ponto)**: indica que exite um arquivo dockerfile

Criar o container node

Vamos criar um container baseado na imagem que acabamos de criar.

```
docker run -p 3000:3000 -d app-node
```

docker run vamos criar um container **-p** vamos indicar um mapeamento de porta **3000:3000** irá expor a porta interna. No exemplo a porta 3000 interna será exposta pela porta 3000 externa.

##Criar um container mysql

- Acessar o docker hub: https://hub.docker.com/_/mysql (https://hub.docker.com/_/mysql)

Para este exemplo é importante deixarmos claro que iremos utilizar uma imagem existente.

```
docker run --name database-mysql -e MYSQL_ROOT_PASSWORD=123 -p 3306:3306 -d mysql
```

docker run: já ira executar a o container, caso a imagem não exista, fará o download **name** indica o nome do container **e** utlizado para configurar o ambiente **MYSQL_ROOT_PASSWORD=123** utilizado para definir a senha da base **mysql**: a imagem padrão do mysql. Neste caso fará o pull do dockerhub.

Criar uma arquivo sql

```
CREATE DATABASE IF NOT EXISTS fiap;
USE fiap;

CREATE TABLE IF NOT EXISTS products (
  id INT(11) AUTO_INCREMENT,
  name VARCHAR(255),
  price DECIMAL(10, 2),
  PRIMARY KEY (id)
);

INSERT INTO products VALUE(0, 'Curso 1', 2500);
INSERT INTO products VALUE(0, 'Curso 2', 900);
```

Acessar o container e executar um bash

Vamos acessar o container para executar o script.

```
docker exec -it database-mysql /bin/bash
```

it: acessar de forma interativa **database-mysql:** nome do container **/bin/bash:** indica que acessaremos via bash

Após acessar vamos logar no mysql

```
mysql -uroot -p123
```

Vamos agora acessar a database

```
use fiap
```

Só lembrando que fiap foi a database criada no script, confirme o nome da sua.

```
select * from products;
```

Por último vamos fazer o select na tabela para confirmar a inserção dos dados

Para sair do container podemos utilizar exit.

Criar volumes

É possível compartilhar uma pasta do seu host com o container

Antes, vamos parar nosso container

```
docker stop database-mysql
```

stop é o comando para parar **database-mysql** o nome do container

No linux

```
docker run --name database-mysql -v $(pwd)/db/data:/var/lib/mysql  -e MYSQL_ROOT_PASSWORD=123 -p 3306:3306 -d mysql
```

No windows

```
docker run --name database-mysql -v /db/data:/var/lib/mysql  -e MYSQL_ROOT_PASSWORD=123 -p 3306:3306 -d mysql
```

db/data: o nome do volume **tag v**: indica o volume **rm**: remover o container caso exista **name**: parametro para criar um nome do Container **mysql-container**: o nome do container

Criar um link para Container

Para podermos conectar o container node ao container do mysql iremos expor um link entre eles.

Para isso vamos parar o container:

```
docker ps

docker stop nome_container_ou_id
```

Executar o container

Vamos executar o container do node com a criação do link

```
docker run -p 3000:3000 -d --rm --name node-container --link database-mysql app-node
```

link indica um link entra o container **database-mysql** nome do container mysql **app-node** nome da imagem node criada

Alterar a conexão

Antes de subir é importante alterar o index e incluir o nome do link na conexão.

- Parar o container node novamente

```
docker ps

docker stop nome_container
```

- Alterar a conexão da base:

```
const connection = mysql2.createConnection({
    //host: 'database-mysql',
    host: 'localhost',
    user: 'root',
    password: '123',
    database: 'fiap',
  });
```

Build da imagem node novamente

```
docker build -t app-node .
```

Executar o container

```
docker run -p 3000:3000 -d --rm --name node-container --link database-mysql app-node
```

###Criar uma aplicação web

Agora vamos criar uma aplicação web para consumir a API criada em node. Este exemplo será desenvolvido em PHP.

Página index.php

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <title>Docker | Programador a Bordo</title>
  <link rel="stylesheet" href="vendor/bootstrap/css/bootstrap.min.css" />
</head>
<body>
  <?php
    $result = file_get_contents("http://node-container:3000/");
    $products = json_decode($result);
  ?>

  <div class="container">
    <table class="table">
      <thead>
        <tr>
          <th>Produto</th>
          <th>Preço</th>
        </tr>
      </thead>
      <tbody>
        <?php foreach($products as $product): ?>
          <tr>
            <td><?php echo $product->name; ?></td>
            <td><?php echo $product->price; ?></td>
          </tr>
        <?php endforeach; ?>
      </tbody>
    </table>
  </div>
</body>
</html>
```

#Vamos incluir também o dockerfile para criar a imagem

```
FROM php:7.4-apache

# configurar a pasta
WORKDIR /var/www/html

# Dependências
RUN apt-get update && apt-get install -y \
    libfreetype6-dev \
    libjpeg62-turbo-dev \
    libpng-dev \
    libzip-dev \
    libonig-dev \
    libxml2-dev \
    && docker-php-ext-configure gd --with-freetype --with-jpeg \
    && docker-php-ext-install pdo_mysql mysqli gd zip mbstring xml


RUN apt-get install -y zip unzip

# Copiar os arquivos
COPY . /var/www/html

# Alterar o proprietários dos arquivos
RUN chown -R www-data:www-data /var/www/html


# Expor a porta
EXPOSE 80

# Configurar o apache
RUN echo "ServerName localhost" >> /etc/apache2/apache2.conf

# Iniciar o server
CMD ["apache2-foreground"]
```

Criar a imagem

Vamos criar uma imagem da aplicação PHP

```
#1 - Acesse a pasta do dockerfile do app em PHP
#2 - Execute o comando abaixo

docker build -t web .

#3 - Após criar a imagem, vamos subir o container

docker run -p 8000:80 -d --rm --name front --link node-container web

#Lembrando que vamos criar um link com o app node para poder consumir a api internamente (dentro do container)

# Neste exemplo a porta 8000 será externa, a porta 80 será a interna (container)
```

Alguns comandos

Comando	Utilização	Parametros
sudo service docker start	start service docker	Nova York
docker run -d -p 80:80 docker/getting-started	Executar um container a partir de uma imagem já existente	"> -d: deamon, executação em background;> -p: porta"
docker container ps	verificar container em execução	Chicago
docker logs -f {{idContainer}}	verificar os logs do container	> -f vai manter em primeiro plano
docker stop {{idContainer}}	parar um container	
docker rm {{idContainer}}	remover um container	
docker container exec -it {{idContainer}} /bin/sh	Acessar o container	> -it iterativo
ps -ef	Dentro do container podemos ver tudo que está rodando	
docker build -t {{nomeImagem}} .	Build da imagem, o ponto ao final buscará o arquivo no local onde está sendo executado	> -t tag
docker run -d -p 3000:3000 getting-starded:latest	Executar uma imagem	
docker ps -a	Pesquisar todos os container	
docker stop $(docker ps -a -q)	Remover todos os containers em execução	
docker rm $(docker ps -a -q)	Remover todos os containers parados	

Referências:

- https://www.youtube.com/watch?v=AVNADGzXrrQ (https://www.youtube.com/watch?v=AVNADGzXrrQ)
- https://hub.docker.com/ (https://hub.docker.com/)
- https://www.youtube.com/watch?v=Kzcz-EVKBEQ&t=1115s (https://www.youtube.com/watch?v=Kzcz-EVKBEQ&t=1115s)
- https://www.freecodecamp.org/portuguese/news/como-remover-imagens-e-conteineres-no-docker/ (https://www.freecodecamp.org/portuguese/news/como-remover-imagens-e-conteineres-no-docker/)