Laboratório 9

Sistemas Distribuídos
Prof Adailton de Jesus Cerqueira Junior

Neste laboratório, daremos continuidade à criação de uma aplicação simples utilizando NodeJS com o framework Express. Agora, vamos executar nossa aplicação em containers Docker.

Introdução:

1 – Os serviços que iremos utilizar neste laboratório são: Banco de dados e aplicação NodeJs.

2 – Um bom container Docker executa apenas um único serviço, mas executar cada container de forma separada pode ser perigoso por diversas questões.

3 – Para baixar e instalar o Docker é só acessar: https://www.docker.com/get-started/

4 – Verifique se o Docker está instalado: Abra o terminal ou prompt de comando e digite "*docker --version*"

5 – Vamos baixar uma imagem: "*docker pull hello-world*"

6 – Teste a execução do container hello-world: "*docker run hello-world*"

NOTA: *Deve ser exibida a mensagem "Hello from Docker".*

7 – Para listar os contêineres em execução use o comando: "*docker ps*"

8 – Para interagir com o container: "*docker exec -it* <ID_CONTAINER> */bin/bash*"

9 – Para parar o container: "*docker stop* <ID_CONTAINER>"

Database:

1 – Será necessário configurar nossa aplicação para o SGBD (Sistema Gerenciador de Banco de Dados) MySQL.

2 – No package.json, vamos trocar "sqlite3": "^5.1.7" por "mysql2": "^3.14.1".

NOTA: *Isso pode ser feito também com o comando npm install mysql.*

NOTA 2: *Será necessário recriar o package-lock.json.*

3 – No .env, será necessário adicionar as configurações para conectar no MySQL:

```
DB_NAME="jogo_api"
DB_HOST="172.23.0.2"
DB_USER="root"
DB_PASSWORD="123456"
```

```
DB_PORT=3306
```

4 – Também será necessário modificar o arquivo *db.js* e alterar a classe Database, conforme exemplo abaixo:

```
const mysql = require("mysql2");
require("dotenv").config();

class Database {
 constructor() {
   if (!Database.instance) {
     this._connect();
     Database.instance = this;
   }
   return Database.instance;
 }

 _connect() {
   this.db = mysql.createConnection({
     host: process.env.DB_HOST || "localhost",
     user: process.env.DB_USER || "root",
     password: process.env.DB_PASSWORD || "",
     database: process.env.DB_NAME || "jogo_api",
     port: process.env.DB_PORT || 3306,
   });
   this.db.connect((err) => {
     if (err) {
       console.error("Erro ao conectar no MySQL: ", err.message);
     } else {
       console.log("Conectado ao MySQL.");
       this._createTable();
     }
   });
 }

 _createTable() {
   // Criação da tabela empresas
   const tbEmpresa = `
     CREATE TABLE IF NOT EXISTS empresas (
       id INT PRIMARY KEY AUTO_INCREMENT,
       nome VARCHAR(255) NOT NULL UNIQUE
     );
   `;
   this.db.query(tbEmpresa, (err) => {
     if (err) console.error("Erro ao criar tabela: ", err.message);
     else {
       console.log("Tabela 'empresas' verificada/criada.");
       this._seed();
     }
```

```
    });

    // Criação da tabela jogos
    const tbJogo = `
      CREATE TABLE IF NOT EXISTS jogos (
        id INT AUTO_INCREMENT PRIMARY KEY,
        nome VARCHAR(255) NOT NULL UNIQUE,
        categoria VARCHAR(255) NOT NULL,
        ano INT NOT NULL,
        fk_empresa INT NOT NULL,
        FOREIGN KEY(fk_empresa) REFERENCES empresas(id)
      );
    `;
    this.db.query(tbJogo, (err) => {
      if (err) console.error("Erro ao criar tabela: ", err.message);
      else console.log("Tabela 'jogos' verificada/criada.");
    });
  }

  _seed() {
    const query = "INSERT IGNORE INTO empresas (nome) VALUES (?)";
    this.db.query(query, ["Nintendo"], (err) => {
      if (err) console.error("Erro ao criar empresa: ", err.message);
      else console.log("Empresa criada.");
    });
    this.db.query(query, ["Ubisoft"], (err) => {
      if (err) console.error("Erro ao criar empresa: ", err.message);
      else console.log("Empresa criada.");
    });
    this.db.query(query, ["Dumativa"], (err) => {
      if (err) console.error("Erro ao criar empresa: ", err.message);
      else console.log("Empresa criada.");
    });
    this.db.query(query, ["Bethesda"], (err) => {
      if (err) console.error("Erro ao criar empresa: ", err.message);
      else console.log("Empresa criada.");
    });
  }
}

// Exporta uma única instância do banco
module.exports = new Database();
```

Docker:

1 - Crie uma pasta chamada lab09 e dentro desta pasta vamos copiar os arquivos criados no laboratório 8.

NOTA: *Você pode fazer esse laboratório direto na pasta do lab08, mas aconselho a criação de um novo diretório para manter o histórico dos laboratórios.*

2 – Dentro da pasta lab09, vamos criar dois arquivos:

- docker-compose.yml: O arquivo de configuração para o Docker Compose.
- Dockerfile: O arquivo de configuração para construir a imagem da aplicação NodeJS.

3 - Dentro do arquivo Dockerfile, vamos configurar nosso container NodeJS. Adicione os seguintes comandos:

```
FROM node:latest
WORKDIR /app
COPY package*.json ./
RUN npm install
EXPOSE 3000
CMD ["npm", "start"]
```

FROM node:latest: Esta instrução específica a imagem base que será usada para construir o contêiner.

WORKDIR /app: Define o diretório de trabalho dentro do contêiner. Todas as instruções subsequentes (como COPY, RUN, etc.) serão executadas a partir deste diretório.

COPY package*.json ./: Copia os arquivos package.json e package-lock.json do diretório atual do host (onde o Dockerfile está localizado) para o diretório de trabalho no contêiner (/app).

RUN npm install: Executa o comando npm install dentro do contêiner.

EXPOSE 3000: Informa ao Docker que o contêiner escutará na porta 3000 em tempo de execução.

CMD ["npm", "start"]: Define o comando padrão que será executado quando um contêiner for iniciado a partir da imagem. Neste caso, ele executa npm start, que inicia a aplicação Node.js.

4 - Dentro do arquivo docker-compose.yml, vamos configurar nossos serviços que iniciaram os containers.

```
services:
 mysql:
   image: mysql:latest
   container_name: sd-bd
   restart: "no"
   ports:
     - "3309:3306"
   environment:
     MYSQL_ROOT_PASSWORD: 123456
```

```
    MYSQL_DATABASE: jogo_api
    TZ: "America/Bahia"
  healthcheck:
    test: ["CMD", "mysqladmin", "ping", "-h", "localhost"]
    interval: 10s
    timeout: 10s
    retries: 3
  networks:
    sd-network:
      ipv4_address: 172.23.0.2

nodejs:
  build:
    context: .
    dockerfile: Dockerfile
  image: nodejs:latest
  container_name: sd-app
  restart: "no"
  ports:
    - "3000:3000"
  working_dir: /app
  volumes:
    - ./:/app
  networks:
    sd-network:
      ipv4_address: 172.23.0.3
  depends_on:
    mysql:
      condition: service_healthy


#Docker Networks
networks:
 sd-network:
  driver: bridge
  ipam:
    config:
      - subnet: 172.23.0.0/24
```

NOTA: *Arquivos .yml (YAML) usam indentação para estruturar os dados.*

NOTA 2: *Pesquise a função de cada uma das instruções do arquivo docker-compose.yml.*

4 – Para criar Execute o comando: *docker-compose up*.

5 – Realize os testes de conexão com a aplicação.