Criando o script para o banco de dados

Antes de configurarmos o arquivo de **Docker Compose**, vamos preparar o arquivo de criação do banco de dados. Esse arquivo será lido automaticamente pelo **PostgreSQL** assim que o container subir. No Windows 11 crie a seguinte estrutura de pastas:

```
C:\workarea\docker\postgresql\demo-psql\init
```

Essa pasta será usada para armazenar os scripts SQL que o Docker executará na criação do banco. Dentro dela, crie um arquivo chamado `dbdemo.sql`. No arquivo, adicione o conteúdo abaixo:

```sql
-- Cria o DB usando ICU + pt-BR (coerente com POSTGRES_INITDB_ARGS)
CREATE DATABASE dbdemo
  WITH OWNER = postgres
    OWNER = postgres
    ENCODING = 'UTF8'
    LC_COLLATE = 'pt_BR.UTF-8'
    LC_CTYPE = 'pt_BR.UTF-8'
    TABLESPACE = pg_default
    CONNECTION LIMIT = -1;
```

Esse script cria um novo banco de dados chamado `dbdemo`. Ele define o usuário `postgres` como dono, configura a codificação **UTF-8** para suportar caracteres especiais, utiliza o provedor de localização **pt-BR** para seguir as regras do português do Brasil. Por fim, usa o `template0`, garantindo que o banco seja criado a partir de uma base limpa. Assim, teremos o arquivo `dbdemos.sql` no diretório correto. Quando executarmos o arquivo do **Docker Compose**, o **PostgreSQL** rodará esse script automaticamente, criando o banco de dados com as configurações necessárias.

Criando o arquivo Dockerfile

Com o script do banco já preparado, precisamos configurar um arquivo **`Dockerfile`**. Esse arquivo serve como uma “receita de construção” da imagem Docker, indicando quais pacotes instalar, como configurar o ambiente e quais arquivos deve ser adicionados. No Windows 11, dentro da pasta:

```
C:\workarea\docker\postgresql\demo-psql
```

Crie um arquivo chamado `Dockerfile` e adicione o conteúdo abaixo:

```
FROM postgres:16.9

# Instalar pacotes necessários para o locale
RUN apt-get update && apt-get install -y locales

# Gerar e configurar o locale pt_BR.UTF-8
RUN echo "pt_BR.UTF-8 UTF-8" >> /etc/locale.gen && \
    locale-gen pt_BR.UTF-8 && \
    update-locale LANG=pt_BR.UTF-8

# Configurar variáveis de ambiente
ENV LANG=pt_BR.UTF-8
ENV LC_ALL=pt_BR.UTF-8
ENV TZ=America/Sao_Paulo

# Copiar o script de criação para o diretório de entrada do PostgreSQL
COPY ./init/dbdemo.sql /docker-entrypoint-initdb.d/

# Expor a porta do PostgreSQL
EXPOSE 5432
```

No trecho acima, cada instrução **`Dockerfile`** tem uma função específica. A seguir estão as explicações de cada step:

- **`FROM postgres:16.9`**: Define que a imagem será baseada no PostgreSQL versão 16.9;
- **Instalação e configuração de locale**: Os comandos `apt-get install -y locales` e `locale-gen` habilitam o suporte ao idioma `pt_BT.UTF-8`, essencial para trabalhar corretamente com acentuação e ordenação de textos;
- **Variáveis de ambiente**: Configuram o idioma padrão (LANG e LC_ALL) e o fuso horário (TZ) para São Paulo;
- **`COPY ./init/dbdemo.sql /docker-entrypoint-initdb.d/`**: Copia o arquivo `dbdemo.sql`

criado anteriormente para o diretório especial `/docker-entrypoint-initdb.d/` todo arquivo `*.sql` colocado nesse diretório é executado automaticamente na primeira inicialização do container PostgreSQL.

- **EXPOSE 5432**: Expõe a porta padrão do PostgreSQL, permitindo que aplicações externas possam se conectar ao banco.

Criando o arquivo Docker Compose

O próximo passo é criar o arquivo `docker-compose.yml`, que descreve como o container PostgreSQL deve ser iniciado, quais portas serão expostas, quais variáveis de ambiente precisam ser configuradas e onde os dados serão armazenados. Dentro da pasta:

```
C:\workarea\docker\postgresql\demo-psql
```

Crie o arquivo chamado `docker-compose.yml` e adicione o conteúdo abaixo:

```yaml
services:
  postgres:
    build: .
    container_name: pg16
    restart: always
    ports:
      - "5432:5432"
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: mypassword
      POSTGRES_DB: dbdemo
    volumes:
      - psql-data:/var/lib/postgresql/data
      - ./init:/docker-entrypoint-initdb.d:ro

volumes:
  psql-data:
```

No trecho acima, cada configuração tem uma papel importante. Veja a explicação:

- `services`: Define os serviços que serão executados. Aqui temos apenas o serviço `postgres`;
- `build: .` : Indica que a imagem do container deve ser construída a partir do **Dockerfile** que criamos na mesma pasta;
- `container_name: pg16` : Dá um nome fixo ao container, facilitando sua identificação;
- `restart: always` : Faz com que o container seja reiniciado automaticamente em caso de falha ou quando o sistema for reiniciado;
- `ports → "5432:5432"` : Mapeia a porta **5432** do container para a porta **5432** da máquina host, permitindo acesso externo ao PostgreSQL;
- `environment` : Define variáveis de ambiente POSTGRES_USER que representa o usuário administrador do PostgreSQL, POSTGRES_PASSWORD senha do usuário definido e POSTGRES_DB banco de dados que será criado na primeira execução.
- `volumes` : A diretiva **`volumes`** faz dois mapeamentos importantes. O primeiro é `psql-data:/var/lib/postgresql/data`, que garante que os dados do banco sejam persistidos mesmo que o container seja removido. O segundo é `./init:/docker-entrypoint-initdb.d:ro`, que monta a pasta **init** (onde está o arquivo `dbdemo.sql`) dentro do diretório especial do PostgreSQL. Dessa forma, todo arquivo `.sql` colocado nessa pasta será executado automaticamente na primeira inicialização do container.
- `volumes` (no final do arquivo) : Cria o volume nomeado `psql-data`, usado para armazenar os dados do banco de forma persistente.

Removendo containers, imagens e volumes anteriores

Antes de subir o ambiente, é importante verificar se já existe algum container em execução com o nome pg16 ou se há volumes e imagens antigas que possam causar conflito. Para listar os containers em execução, use:

```
docker ps
```

Se encontrar um container com o mesmo nome pg16, pare e remova-o com:

```
docker stop pg16
docker rm pg16
```

Em seguida, verifique se existem imagens antigas do PostgreSQL criadas pelo seu **Dockerfile**. Para listar todas as imagens, execute:

```
docker images
```

Caso identifique imagens relacionadas ao seu projeto, remova-as com o comando abaixo:

```
docker rmi <IMAGE_ID>
```

Por fim, é recomendável remover os volumes antigos associados ao PostgreSQL para garantir que o banco será criado do zero. Para listar os volumes existentes:

```
docker volume ls
```

E para remover um volume específico:

```
docker volume rm <VOLUME_NAME>
```

Caso queira fazer uma limpeza geral de containers, imagens e volumes não utilizados, pode usar:

```
docker system prune -a --volumes
```

> ⚠ **Atenção**: Esse comando remove todos os containers parados, imagens não utilizadas e volumes não referenciados. Use com cuidado, pois a exclusão é definitiva, porém é o melhor comando para manter o seu ambiente de desenvolvimento otimizado.

Executando o Docker Compose no WSL2

Com todos os arquivos preparados, vamos agora iniciar o ambiente. Para isso, abra o WSL2 (Ubuntu) e navegue até a pasta do projeto:

```
cd /mnt/c/workarea/docker/postgresql/demo-psql
```

> **Lembre-se**: No WSL2, os caminhos do Windows são acessados via `/mnt/c/...`.

Uma vez dentro da pasta, execute o seguinte comando para construir a imagem e iniciar o container:

```
docker compose up -d
```

Verificando se o container e o banco foram criados corretamente

Depois de executar o `docker compose up -d`, o primeiro passo é confirmar que o container está rodando. Para isso use o comando:

```
docker ps
```

Se o container pg16 aparecer na lista com o status Up, significa que o PostgreSQL está em execução. Em seguida, vamos acessar o container e consultar as informações do banco dbdemo para garantir que ele foi criado com o locale correto. Execute:

```
docker exec -it pg16 psql -U postgres -d dbdemo -c \
"SELECT datname, datcollate, datlocprovider FROM pg_database WHERE datname =
current_database();"
```

Esse comando entra no container, abre o cliente `psql` e executa a query SQL que mostra o nome do banco atual, a configuração de collation e o provedor de locale.

A saída esperada deve ser semelhante a esta:

```
 datname | datcollate  | datlocprovider
---------+-------------+----------------
 dbdemo  | pt_BR.UTF-8 | c
(1 linha)
```

Conclusão

Chegamos ao final deste capítulo, no qual preparamos todo o ambiente para rodar o PostgreSQL no WSL2 com Docker. Neste percurso, criamos:

- O script `dbdemo.sql` para configurar o banco com locale em português do Brasil;
- O `Dockerfile` para personalizar a imagem do PostgreSQL;
- O `docker-compose.yml` para orquestrar o container;
- O processo para garantir que não havia containers, imagens e volumes antigos em conflito;
- E por fim, validamos que o banco `dbdemo` foi criado corretamente com o locale configurado.

No próximo capítulo da série, vamos aprender a acessar o banco de dados através do **pgAdmin** no Windows, criando uma interface gráfica para gerenciar suas bases de forma prática e intuitiva.