Introdução

Este tutorial mostrará como instalar e executar o PostgreSQL com Docker. Usaremos o Ubuntu 22.04.5 LTS rodando no WSL2 dentro do Windows 11 Pro. A ideia é mostrar de forma prática, como preparar o ambiente, configurar o contêiner e iniciar o banco de dados para uso imediato, sem precisar de instalações complexas diretamente no sistema operacional.

O uso do Docker traz várias vantagens, como:

- **Isolamento do ambiente:** O banco de dados roda dentro de um contêiner independente;
- **Facilidade de configuração:** Podemos iniciar ou remover a instância rapidamente;
- **Reprodutibilidade:** É possível recriar a mesma configuração em outros ambientes de forma consistente;
- **Praticidade no WSL2:** Aproveitamento a leveza e integração do Ubuntu dentro do Windows.

Ao final, você terá um contêiner PostgreSQL rodando no WSL2, pronto para receber conexões, criar banco de dados, usuários e rodar suas aplicações.

Verificando se há contêiner em execução no WSL2

Antes de criar uma nova instância do PostgreSQL no Docker, é importante verificar se já existe algum contêiner em execução. Isso evita conflitos de porta e ajuda a manter o ambiente organizado. Para listar os contêineres ativos, utilize o seguinte comando no terminal do Ubuntu 22.04 no WSL2:

```
docker ps
```

O resultado desse comando no meu ambiente pode ser visto na imagem abaixo:

> **Atenção:** O resultado exibido é baseado na minha máquina. No seu ambiente, os contêineres listados serão diferentes, dependendo do que você já tenha rodado no Docker.

No meu caso, o `docker ps` retornou três contêineres em execução:

Oracle XE

- **CONTAINER ID:** `55104ac3985b`
- **IMAGE:** `container-registry.oracle.com/database/express:21.3.0-xe`
- **PORTS:** `0.0.0.0:1521->5500/tcp`

PostgreSQL

- **CONTAINER ID:** `b64a9ce87db5`
- **IMAGE:** `db-postgresql-postgres`
- **PORTS:** `0.0.0.0:5432->5432/tcp`

Portainer (ferramenta de administração do Docker via interface web)

- **CONTAINER ID:** `31d0a79a5f93`
- **IMAGE:** `portainer/portainer-ce:latest`
- **PORTS:** `8000/tcp, 9000/tcp`

Com base nesse resultado, identificamos que já existem contêineres do Oracle XE e do PostgreSQL rodando, além do Portainer. Como o nosso objetivo é configurar uma nova instância do PostgreSQL, vamos remover apenas os contêineres do Oracle e do PostgreSQL, mantendo o Portainer ativo.

> **Atenção:** Não se preocupe caso você não tenha o **Portainer** instalado no seu ambiente. Neste momento, o nosso foco não é instalar ou configurar o Portainer, mas sim **remover os contêineres de banco de dados já existentes** para liberar espaço e evitar conflitos de portas quando criarmos uma nova instância do PostgreSQL.

Parando e removendo os contêineres de banco de dados existentes

Antes de removermos um contêiner do Docker, é necessário pará-lo primeiro. Isso acontece porque o Docker não permite excluir diretamente um contêiner que ainda está em execução. Assim, o processo é sempre feito em duas etapas: primeiro paramos o contêiner e depois o removemos.

No exemplo da minha máquina, os contêineres que precisamos excluir são os do PostgreSQL e do Oracle XE. Para isso, começamos interrompendo sua execução com o comando:

```
docker stop b64a9ce87db5 55104ac3985b
```

O primeiro ID corresponde ao contêiner do PostgreSQL e o segundo ao do Oracle XE. Após a execução desse comando, ambos deixarão de rodar e liberarão os recursos que estavam ocupando. Em seguida, podemos prosseguir com a remoção definitiva dos contêineres. Para isso, usamos:

```
docker rm b64a9ce87db5 55104ac3985b
```

Assim, os dois contêineres deixam de existir no ambiente. Para confirmar que foram realmente removidos, basta rodar novamente o comando `docker ps`. Neste ponto, apenas o **Portainer** (ou outro contêiner que você eventualmente tenha) deverá aparecer listado.

Listando e escolhendo as imagens corretas para remover

Depois de remover os contêineres do Oracle XE e do PostgreSQL, ainda precisamos verificar quais imagens continuam armazenadas localmente. Isso é importante porque as imagens ocupam bastante espaço em disco, mesmo sem nenhum contêiner ativo. No meu ambiente, ao executar o comando abaixo:

```
docker image ls
```

obtive o seguinte resultado (imagem abaixo):

> ⚠ **Atenção:** essa listagem reflete apenas o meu ambiente. No seu computador, o resultado será diferente, dependendo de quais imagens você já baixou ou utilizou.

No caso mostrado, temos três imagens armazenadas:

- **db-postgresql-postgres** (460 MB)
- **portainer/portainer-ce** (268 MB)

- **container-registry.oracle.com/database/express:21.3.0-xe** (11.4 GB)

Como nosso objetivo é liberar espaço e preparar o ambiente para uma nova instância do PostgreSQL, iremos remover apenas as imagens de **banco de dados** — ou seja, o `db-postgresql-postgres` e o `container-registry.oracle.com/database/express:21.3.0-xe`. A imagem do **Portainer** será mantida, pois ela não é um banco de dados e pode continuar sendo útil para administração do Docker.

Para remover as imagens, utilizamos o comando `docker rmi`, especificando o nome ou o IMAGE ID correspondente. No meu caso, os comandos seriam:

```
docker rmi db-postgresql-postgres container-registry.oracle.com/database/express:21.3.0-xe
```

Depois de executar a remoção, é sempre uma boa prática rodar novamente o `docker image ls` para confirmar se as imagens realmente foram excluídas.

Conclusão

Neste primeiro momento, aprendemos a verificar quais contêineres estavam em execução, entender a saída do comando `docker ps`, identificar os serviços de banco de dados já existentes e, por fim, removemos tanto os contêineres quanto as imagens antigas do PostgreSQL e do Oracle XE para liberar espaço no ambiente.

om o cenário agora limpo e preparado, estamos prontos para avançar. Na **Parte 2** desta série, iremos criar do zero um novo contêiner PostgreSQL no Docker, já configurado com senha, porta exposta e volume persistente para garantir que os dados não sejam perdidos.