

Images haven't loaded yet. Please exit printing, wait for images to load, and try to print again.

Compose: montando rapidamente um ambiente para uso



Renato Groffe

Follow

Jan 6 · 4 min read



Em um artigo recente abordei a criação de **containers Docker** para a execução do **pgAdmin 4** e de uma instância do **PostgreSQL**:

PostgreSQL + Docker: executando uma instância e o pgAdmin 4 a partir de containers

A solução em questão permite, sem sombra de dúvidas, a rápida montagem de um ambiente que dependa das tecnologias de bancos de dados mencionadas (**PostgreSQL** e **pgAdmin 4**). Contudo, 3 comandos foram executados neste procedimento a fim de criar as estruturas necessárias: 2 containers, além de uma network que possibilitasse a comunicação entre tais elementos.

E como tornar tudo isto mais fácil ainda, se possível através de um único comando e gerando até mesmo um script reutilizável?

O **Docker Compose** é a resposta para simplificar esse processo. Sua utilização dependerá primeiramente da criação de um arquivo

chamado **docker-compose.yml**, o qual conterá as configurações para a geração de containers e networks.


Importante destacar que o **Docker Compose** é um serviço do próprio **Docker** voltado à criação e execução conjunta dos múltiplos containers de uma solução. Tal capacidade contribui para facilitar o deployment de um projeto em diferentes ambientes.

Na listagem a seguir está o conteúdo do arquivo **docker-compose.yml** que permitirá a criação do ambiente citado (**PostgreSQL + pgAdmin 4**). Os testes descritos neste artigo acontecerão no **Ubuntu Desktop 18.04**:

- O serviço **teste-postgres-compose** se refere à instância do **PostgreSQL** a ser criada para acesso na porta **15432**;
- Já o serviço **teste-pgadmin-compose** corresponde ao container que permitirá a execução do **pgAdmin 4** (imagem **dpape/pgadmin4**) na porta **16543**;
- Nas seções **environment** de **teste-pgadmin-compose** e **teste-postgres-compose** foram definidas configurações (variáveis de ambientes) necessárias para a geração dos 2 containers;
- As imagens referenciadas serão baixadas caso ainda não existam no ambiente a partir do qual o **Docker Compose** foi executado;
- Foi especificado ainda um volume para **teste-postgres-compose**, indicando assim o diretório no **Ubuntu Desktop** em que serão gravados os arquivos de dados (**/home/renatogroffe/Desenvolvimento/Docker-Compose/PostgreSQL**);
- Por meio da network **postgres-compose-network** acontecerá a comunicação entre os containers **teste-pgadmin-compose** e **teste-postgres-compose**.

```
1  version: '3'
2
3  services:
4    teste-postgres-compose:
5      image: postgres
6      environment:
7        POSTGRES_PASSWORD: "Postgres2019!"
8      ports:
9        - "15432:5432"
10     volumes:
11       - /home/renatogroffe/Desenvolvimento/Docker-Comp
12     networks:
13       - postgres-compose-network
14
15     teste-pgadmin-compose:
16       image: dpage/pgadmin4
17       environment:
18         PGADMIN_DEFAULT_EMAIL: "renatogr@yahoo.com.br"
19         PGADMIN_DEFAULT_PASSWORD: "PgAdmin2019!"
20     ports:
```

O comando **docker-compose up -d** procederá com a criação da network e dos containers esperados, efetuando inclusive o download das imagens se as mesmas ainda não existirem na máquina considerada (não foi o caso deste exemplo):



```
renatogroffe@renatogroffe-Virtual-Machine: ~/Desenvolvimento/AmbientePostgreSQL$ sudo docker-compose up -d
Creating network "ambientepostgresql_postgres-compose-network" with driver "bridge"
Creating ambientepostgresql_teste-postgres-compose_1_6a0c24476ab1 ... done
Creating ambientepostgresql_teste-pgadmin-compose_1_5dfe0808afee ... done
renatogroffe@renatogroffe-Virtual-Machine: ~/Desenvolvimento/AmbientePostgreSQL$
```

Com a instrução **docker network ls** podemos confirmar que a rede **postgres-compose-network** foi criada com sucesso (como **ambientepostgresql_postgres-compose-network**, resultado da concatenação com o nome do diretório em que se encontra o arquivo **docker-compose.yml**):

```

renatogroffe@renatogroffe-Virtual-Machine: ~/Desenvolvimento/AmbientePostgreSQL$ sudo docker network ls
NETWORK ID          NAME                                     DRIVER          SCOPE
4e1007ba5325        ambientepostgresql_postgres-compose-network bridge          local
1d4fb7949e07        apicontagem_default                   bridge          local
0b440195be01        bridge                                bridge          local
10656615c66f        host                                  host            local
79ee81f27008        none                                  null            local
430b9f116f50        postgres-network                       bridge          local
renatogroffe@renatogroffe-Virtual-Machine: ~/Desenvolvimento/AmbientePostgreSQL$

```

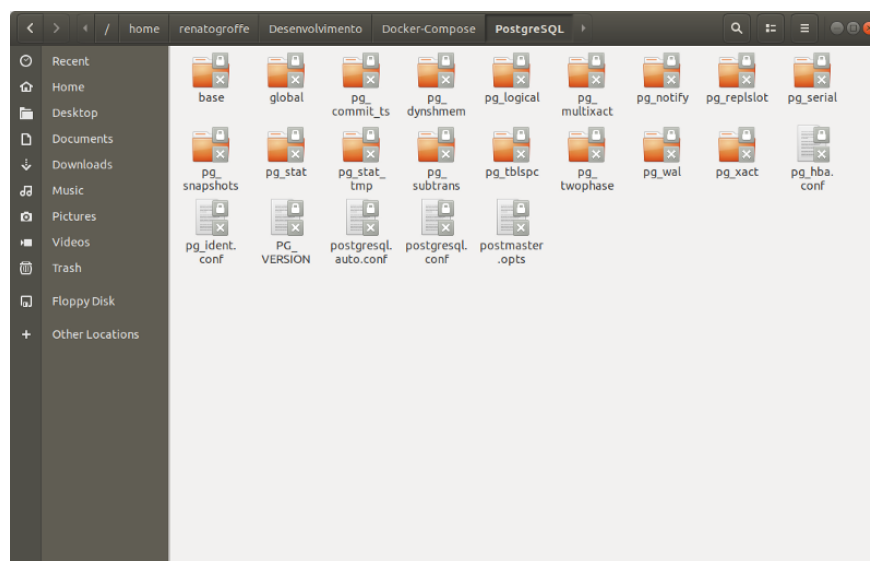
Já o comando **docker-compose ps** indicará que os containers do PostgreSQL (porta 15432) e do pgAdmin 4 (porta 16543) foram gerados corretamente e se encontram em execução:

```

renatogroffe@renatogroffe-Virtual-Machine: ~/Desenvolvimento/AmbientePostgreSQL$ sudo docker-compose ps
Name                                Command                                State          Ports
-----                                -
ambientepostgresql_teste-pgadmin-   /entrypoint.sh                        Up             443/tcp, 0.0.0.0:16543->80/tcp
compose_1_a78364dbdb0d
ambientepostgresql_teste-postgres-   docker-entrypoint.sh postgres        Up             0.0.0.0:15432->5432/tcp
compose_1_6c3d8511ef1e
renatogroffe@renatogroffe-Virtual-Machine: ~/Desenvolvimento/AmbientePostgreSQL$

```

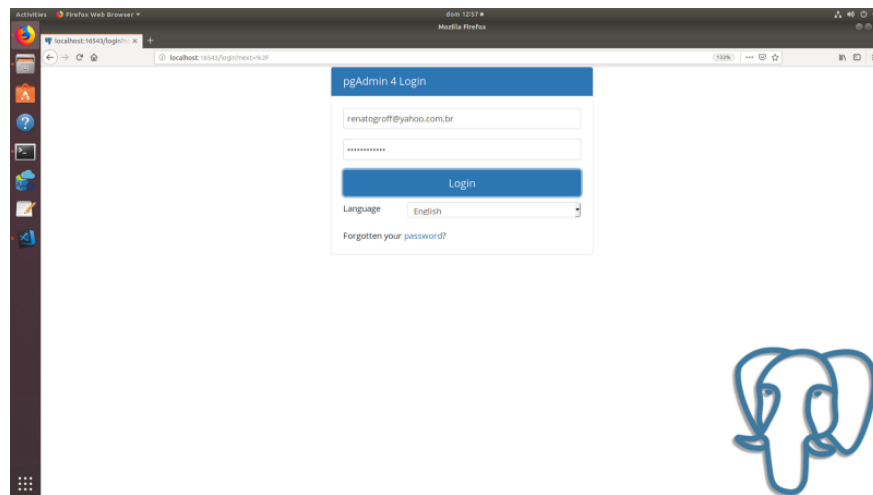
Na próxima imagem estão os arquivos e diretórios criados para o volume definido no arquivo **docker-compose.yml**:



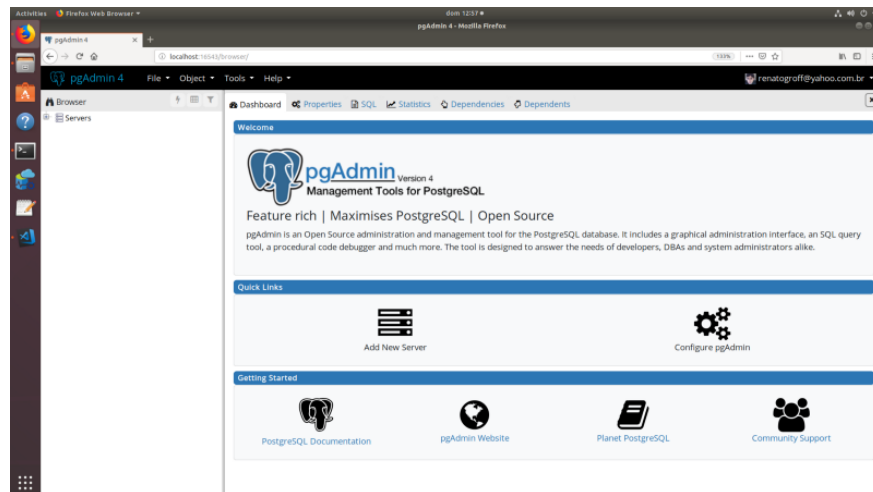
Testando o ambiente

Um teste de acesso via browser ao pgAdmin 4

(<http://localhost:16543>) exibirá a tela inicial desta solução:



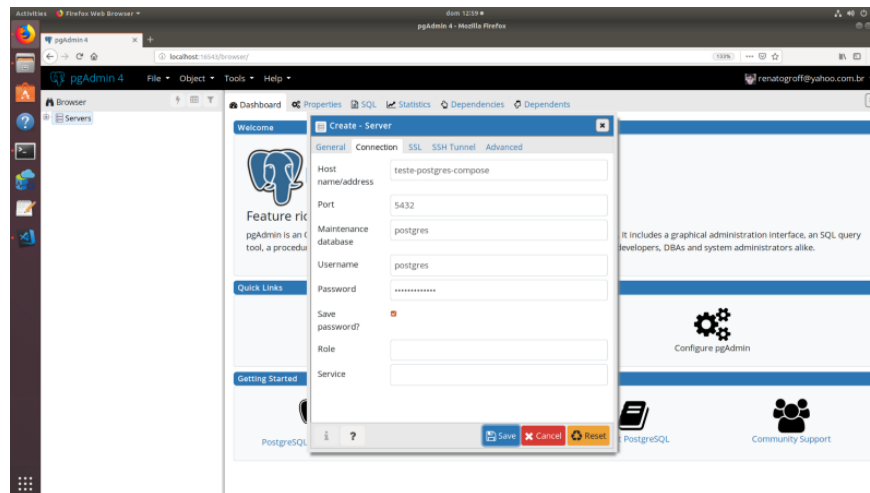
Fornecendo as credenciais de acesso que estavam no arquivo **docker-compose.yml** aparecerá então o painel de gerenciamento do **pgAdmin 4**:



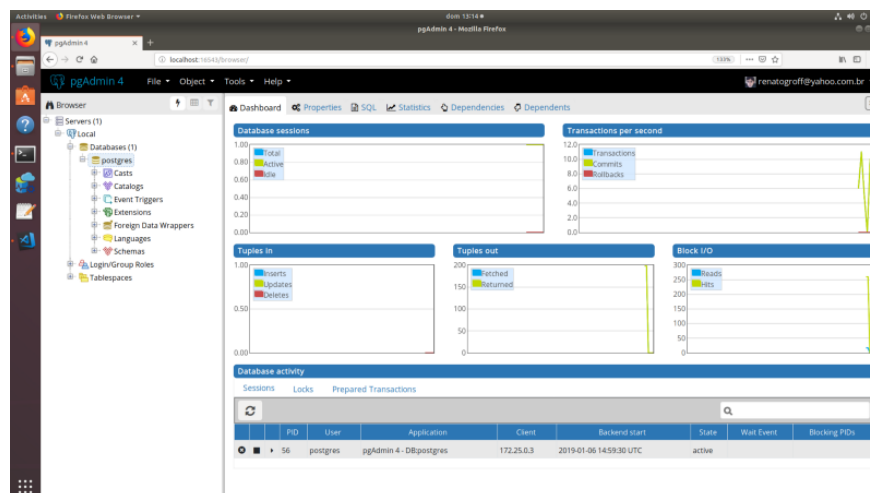
Ao criar a conexão para acesso à instância do **PostgreSQL** levar em conta as seguintes considerações:

- Em **Host name/address** informar o nome do container que corresponde à instância do **PostgreSQL** (**teste-postgres-compose**);
- Em **Port** definir o valor **5432** (porta default de acesso ao container e disponível a partir da rede **postgres-compose-network**; não informar a porta em que o **PostgreSQL** foi mapeado no host);

- No atributo **Username** será informado o usuário default do **PostgreSQL (postgres)**, bem como a senha correspondente em **Password (Postgres2019!)**.



Na próxima imagem é possível observar que a conexão ao **PostgreSQL** via **pgAdmin 4** ocorreu com sucesso:



E aproveito este espaço para deixar aqui ainda um convite.

Dia **08/01/2019 (terça-feira)** às **21h30—horário de Brasília—** teremos o **primeiro hangout do ano de 2019** no **Canal .NET**. Desta vez receberemos o **MVP Luiz Carlos Faria**, que fará uma apresentação

justamente sobre o **uso do Portainer como solução de administração e troubleshooting de containers Docker**.

Para efetuar a sua inscrição acesse a [página do evento no Meetup](#). A transmissão acontecerá via **YouTube**, em um link a ser divulgado em breve.

. . .

Referências

[Docker para Desenvolvedores .NET - Guia de Referência](#)

[PostgreSQL - Docker Hub](#)

[pgAdmin 4 - Docker Hub](#)