

PostgreSQL Docker & Windows Server 2019

Einführung

PostgreSQL, auch Postgres genannt, ist ein objektrelationales Open Source Datenbankmanagementsystem. Mit MySQL gehört PostgreSQL zu den beliebtesten relationalen Datenbankmanagementsystemen.

Heute ist Postgres eines der häufigsten verwendeten Docker-Images, das in Form von Containern ausgeführt wird. Die Beliebtheit von containerisierten Datenbanken geht auf die Einfachheit zurück, mit der sie bereitgestellt werden. Anstatt eine zentrale Datenbank zu haben, können Entwickler außerdem für jede Anwendung einen PostgreSQL-Container verwenden.

Wir werden uns hier damit beschäftigen, wie man PostgreSQL auf einem Docker Container ausführen und wie man eine PostgreSQL-Datenbank sichern kann.

Voraussetzungen

- × Zugriff auf Windows Powershell
- × Ein Benutzerkonto mit **Administrator**-Berechtigung
- × Eine vorhandene Docker-Installation

PostgreSQL auf Docker Containern ausführen

Das Bereitstellen eines Postgres-Containers ist einfach. Das Postgres-Image zum Erstellen dieser Datenbankcontainer findest Du im offiziellen Docker Hub. Wir zeigen Dir zwei Möglichkeiten, dies zu tun.

Die erste Option verwendet **Docker Compose**, ein Tool zum Verwalten von Docker-Anwendungen mit mehreren Containern. Du kannst Docker Compose verwenden, um Postgres als Dienst, der in einem Container ausgeführt wird, zu konfigurieren. In diesem Fall erstellst Du eine yaml-Datei mit allen Spezifikationen.

Alternativ kannst Du als zweite Option einen **einzelnen Docker-Befehl** mit allen erforderlichen Informationen zum Bereitstellen eines neuen PostgreSQL-Containers verwenden.

Option 1: Postgres mit Docker Compose ausführen

Um einen Postgres-Container mit Docker bereitzustellen, solltest Du diesen auf Deinem System installiert haben.

- Um eine einfache und saubere Installation zu gewährleisten, wollen wir zuerst ein Arbeitsverzeichnis mit dem Namen **postgres** erstellen und in dieses Verzeichnis wechseln. Das machst Du mit diesem Befehl:

```
mkdir postgres
cd postgres/
```

- Als Nächstes verwendest Du Docker Compose, um das Postgres-Image herunterzuladen und den Dienst zum Laufen zu bringen. Hierzu erstellst Du mit einem **Editor** Deiner Wahl eine neue **docker-compose.yml**-Datei mit folgendem Inhalt:

```
version: '3.8'
services:
  my_service:
    image: postgres:latest
    ports:
      - 5432:5432
    environment:
      - POSTGRES_PASSWORD=Start123
```

Die yaml-Konfigurationsdatei beschreibt, dass es einen **my_service**-Dienst gibt, der auf **latest postgres image** basiert. Es bleibt Dir überlassen, welche Version Du verwenden möchtest. Wir haben uns hier für die neueste **'3.8'** Postgres-Version entschieden.

Schließlich musst Du die Ports definieren, über die der Container kommuniziert. **5432** ist die Standardportnummer für PostgreSQL und die Umgebungsvariable für das Passwort des Postgres-Superuser setzen (der Wert "Start123" kann beliebig angegeben werden).

- Speichere die Datei.

- Nachdem Du nun die yaml-Konfigurationsdatei hast, kannst Du den Postgres-Dienst starten und den Container ausführen. Verwende den **docker-compose up**-Befehl mit der **-d**-Option um ihn in den Detach-Modus zu versetzen (damit Du weiterhin Befehle von der aktuellen Shell ausführen können):

```
docker-compose up -d
```

```
Administrator: Windows PowerShell
PS C:\Users\denise\desktop\postgres> docker-compose up -d
postgres_my_service_1 is up to date
PS C:\Users\denise\desktop\postgres> docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS                    NAMES
6bc7e45df503   postgres:latest "docker-entrypoint.s..." About a minute ago Up About a minute   0.0.0.0:5432->5432/tcp   postgres_my_service_1
PS C:\Users\denise\desktop\postgres>
```

- Du kannst die dabei ausgegebenen Protokolle mit dem folgenden Befehl überprüfen:

```
docker-compose logs -f
```

```
Auswählen Administrator: Windows PowerShell
PS C:\Users\denise\desktop\postgres> docker-compose logs -f
Attaching to postgres_my_service_1
postgres_my_service_1 The files belonging to this database system will be owned by user "postgres".
postgres_my_service_1 This user must also own the server process.
postgres_my_service_1
postgres_my_service_1 The database cluster will be initialized with locale "en_US.utf8".
postgres_my_service_1 The default database encoding has accordingly been set to "UTF8".
postgres_my_service_1 The default text search configuration will be set to "english".
postgres_my_service_1
postgres_my_service_1 Data page checksums are disabled.
postgres_my_service_1
postgres_my_service_1 fixing permissions on existing directory /var/lib/postgresql/data ... ok
postgres_my_service_1 creating subdirectories ... ok
postgres_my_service_1 selecting dynamic shared memory implementation ... posix
postgres_my_service_1 selecting default max_connections ... 100
postgres_my_service_1 selecting default shared_buffers ... 128MB
postgres_my_service_1 selecting default time zone ... ftc/UTC
postgres_my_service_1 creating configuration files ... ok
postgres_my_service_1 running bootstrap script ... ok
postgres_my_service_1 performing post-bootstrap initialization ... ok
postgres_my_service_1 initdb: warning: enabling "trust" authentication for local connections
postgres_my_service_1 You can change this by editing pg_hba.conf or using the option -A, or
postgres_my_service_1 -auth local and --auth-host, the next time you run initdb.
postgres_my_service_1 syncing data to disk ... ok
postgres_my_service_1
postgres_my_service_1
postgres_my_service_1 Success. You can now start the database server using:
postgres_my_service_1
postgres_my_service_1 pg_ctl -D /var/lib/postgresql/data -l logfile start
postgres_my_service_1
postgres_my_service_1 waiting for server to start...2020-08-20 13:29:49.654 UTC [45] LOG: starting PostgreSQL 12.4 (Debian 12.4-1.pgdg100+1) on x86_64-pc-linux-gnu, compiled by gcc (Debian 8
3.0-6) 8.3.0, 64-bit
2020-08-20 13:29:49.659 UTC [45] LOG: listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
2020-08-20 13:29:49.697 UTC [46] LOG: database system is shut down at 2020-08-20 13:29:48 UTC
2020-08-20 13:29:49.707 UTC [45] LOG: database system is ready to accept connections
done
server started
postgres_my_service_1 /usr/local/bin/docker-entrypoint.sh: ignoring /docker-entrypoint-initdb.d/*
postgres_my_service_1
postgres_my_service_1 waiting for server to shut down...2020-08-20 13:29:49.737 UTC [45] LOG: received fast shutdown request
2020-08-20 13:29:49.743 UTC [45] LOG: aborting any active transactions
2020-08-20 13:29:49.746 UTC [45] LOG: background worker "logical replication launcher" (PID 52) exited with exit code 1
2020-08-20 13:29:49.747 UTC [47] LOG: shutting down
2020-08-20 13:29:49.793 UTC [45] LOG: database system is shut down
done
server stopped
postgres_my_service_1
postgres_my_service_1 PostgreSQL init process complete; ready for start up.
postgres_my_service_1
2020-08-20 13:29:49.859 UTC [1] LOG: starting PostgreSQL 12.4 (Debian 12.4-1.pgdg100+1) on x86_64-pc-linux-gnu, compiled by gcc (Debian 8.3.0-6) 8.3.0, 64-bit
2020-08-20 13:29:49.860 UTC [1] LOG: listening on IPv4 address "0.0.0.0", port 5432
2020-08-20 13:29:49.860 UTC [1] LOG: listening on IPv6 address "::::", port 5432
2020-08-20 13:29:49.870 UTC [1] LOG: listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
2020-08-20 13:29:49.901 UTC [54] LOG: database system was shut down at 2020-08-20 13:29:49 UTC
2020-08-20 13:29:49.909 UTC [1] LOG: database system is ready to accept connections
```

Mit **CTRL+C** kehrst Du wieder zur Shell zurück.

Option 2: Postgres mit einem einzelnen Docker-Befehl ausführen

Eine weitere Möglichkeit, PostgreSQL in einem Container bereitzustellen, besteht darin, einen einzelnen Docker-Befehl auszuführen.

Du kannst einen Postgres-Container herunterladen und ausführen, indem Du alle erforderlichen Informationen in einem Befehl angibst.

Folgender Befehl weist Docker an, einen neuen Container unter einem bestimmten Containernamen auszuführen, definiert das Postgres-Passwort und lädt die neueste Postgres-Version herunter:

```
docker run --name [container_name] -e POSTGRES_PASSWORD=[your_password] -d postgres
```

Bestätige, dass Dein PostgreSQL-Container jetzt aktiv ist, indem Du Docker mit dem unten stehenden Befehl aufforderst, alle laufenden Container aufzulisten:

```
docker ps
```

In unserem Beispiel haben wir einen Container **"postgres"** erstellt, den wir leicht unter anderen laufenden Containern finden können.

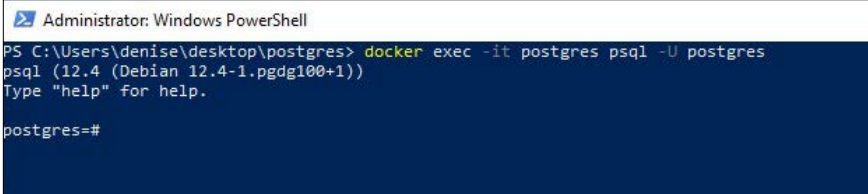
Der Start mit Postgres Containern

Eine Verbindung zu Postgres im Docker Container herstellen

Um in einen Postgres-Container hineinzukommen, muss der folgende Befehl ausgeführt werden:

```
docker exec -it [container_name] psql -U [postgres_user]
```

In diesem Beispiel haben wir uns mit dem Container **"postgres"** als **"postgres"**-Benutzer verbunden.



```
Administrator: Windows PowerShell
PS C:\Users\denise\desktop\postgres> docker exec -it postgres psql -U postgres
psql (12.4 (Debian 12.4-1.pgdg100+1))
Type "help" for help.

postgres=#
```

Eine Datenbank erstellen

In unserem Docker Postgres Container können wir mit dem folgenden Befehl eine Datenbank erstellen:

```
create database [db_name];
```

Tipp: Mit `\l` kannst Du Dir alle Datenbanken anzeigen lassen, die auf PostgreSQL ausgeführt werden.

Mit folgendem Befehl verbindest Du Dich als Postgres-Benutzer mit der Datenbank:

```
\c [db_name]
```

Wenn die Datenbank eingerichtet ist, musst Du im nächsten Schritt ein Schema erstellen, mit dem Du eine logische Darstellung der Datenbankstruktur erhältst:

```
create schema [db_schema_name]
```

Hiermit kannst Du eine Tabelle erstellen und Daten zur Tabelle hinzufügen:

```
create table [table_name] ([field_names] [values])
```

Tipp: Um den Postgres-Container zu verlassen benutze `\q`

So sicherst Du eine PostgreSQL-Datenbank mit Docker

Sichern einer lokalen oder einer remote PostgreSQL-Datenbank

1. Befehl zum Sichern einer lokalen oder einer remote PostgreSQL-Datenbank:

```
$ docker run -i postgres /usr/bin/pg_dump  
-h [POSTGRES_HOST]  
-U [POSTGRES_USER] [POSTGRES_DATABASE] > backup.sql
```

2. Befehl zum Sichern mehrerer PostgreSQL-Datenbanken:

```
$ docker run -i postgres /usr/bin/pg_dumpall  
-h [POSTGRES_HOST]  
-U [POSTGRES_USER] > backup.sql
```

3. Befehl zum Sichern einer lokalen oder einer remote PostgreSQL-Datenbank mit Komprimierung (mit gzip):

```
$ docker run -i postgres /usr/bin/pg_dump  
-h [POSTGRES_HOST]  
-U [POSTGRES_USER] [POSTGRES_DATABASE] | gzip -9 > backup.sql.  
gz
```

4. Voriger Befehl, der jedoch das PostgreSQL-Passwort als Umgebungsvariable liefert:

```
$ docker run -i -e PGPASSWORD=[POSTGRES_PASSWORD] postgres /usr/  
bin/pg_dump  
-h [POSTGRES_HOST]  
-U [POSTGRES_USER] [POSTGRES_DATABASE] | gzip -9 > backup.sql.  
gz
```

Sichern einer containerisierten PostgreSQL-Datenbank

1. Befehl zum Sichern einer containerisierten PostgreSQL-Datenbank, wobei eine komprimierte Datei erstellt wird (mit gzip):

```
$ docker exec [POSTGRESQL_CONTAINER] /usr/bin/pg_dump  
-U [POSTGRESQL_USER] [POSTGRESQL_DATABASE] | gzip -9 > backup.sql.  
gz
```

2. Voriger Befehl, der jedoch zusätzlich das PostgreSQL-Passwort als Umgebungsvariable bei vorhandenem Container festlegt:

```
$ docker exec [POSTGRESQL_CONTAINER] /bin/bash  
-c "export PGPASSWORD=[POSTGRESQL_PASSWORD]  
&& /usr/bin/pg_dump -U [POSTGRESQL_USER] [POSTGRESQL_DATABASE] "  
| gzip -9 > backup.sql.gz
```

Du hast nun zwei verschiedene Methoden zum Ausführen von PostgreSQL in einem Docker-Container kennengelernt und weißt wie man verschiedene Arten von Datenbanken sichern kann. Damit solltest Du jetzt mit der Erstellung Deiner Datenbanken mit allen erforderlichen Daten beginnen können

backup, database, docker, mssql,
powershell, sql server, t-sql, volume