Docker und SQL Server: Erstellen, Updaten und .NET Anbindung

Kathegorie
SQL Server

Aaron
Priesterroth

In diesem Beitrag möchte ich Dir eine kleine Einführung in die Verwendung von MSSQL mit Docker Containern geben. Die primären Ziele dabei sind folgende:

- das Ausführen eines SQL Servers mithilfe von Docker
- das Ausführen mehrerer SQL Server mithilfe von Docker
- das Wiederherstellen einer Datenbank auf dem zuvor erstellten SQL Server Container
- das Verbinden des MSSQL Containers mit dem .NET Core

Vorbereitungen

Als erstes muss das SQL Server Image bezogen werden. Mithilfe der folgenden PowerShell-Eingabe kann eine Übersicht der von Microsoft zur Verfügung gestellten SQL Server Images erzeugt werden:

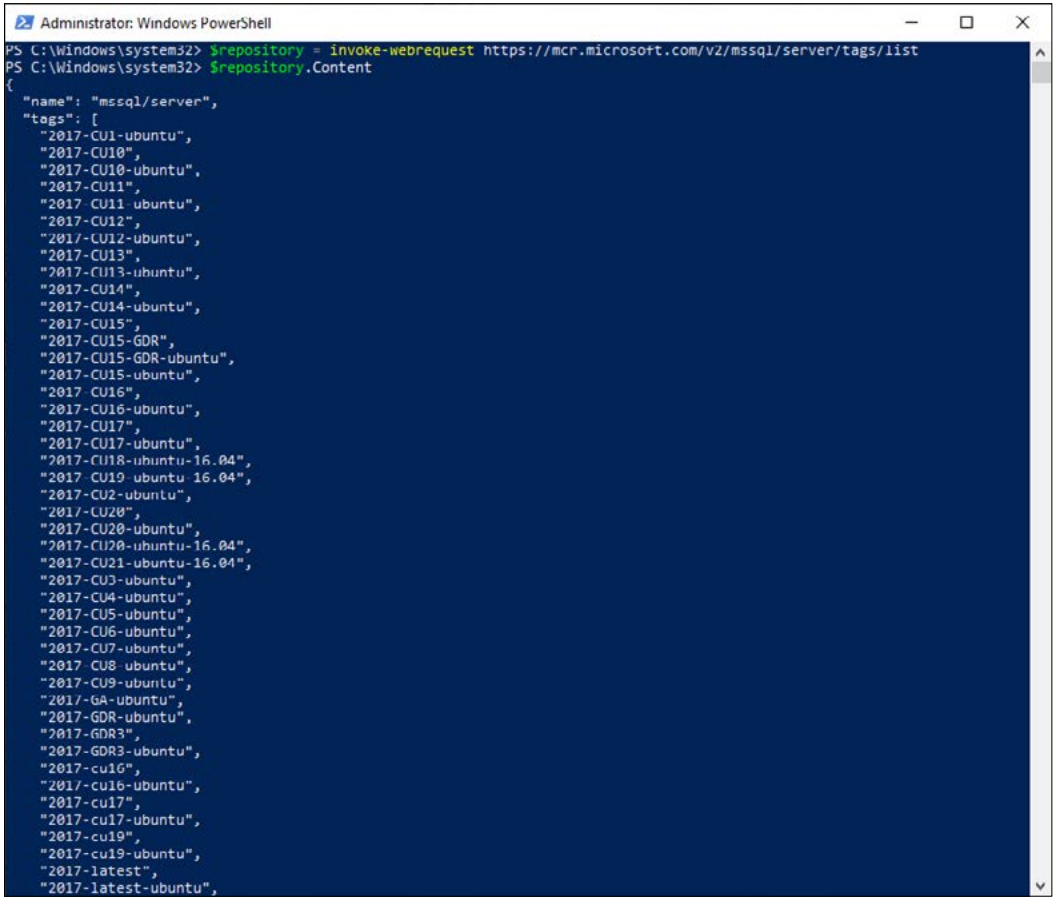
```
invoke-webrequest https://mcr.microsoft.com/v2/mssql/server/tags/list
```

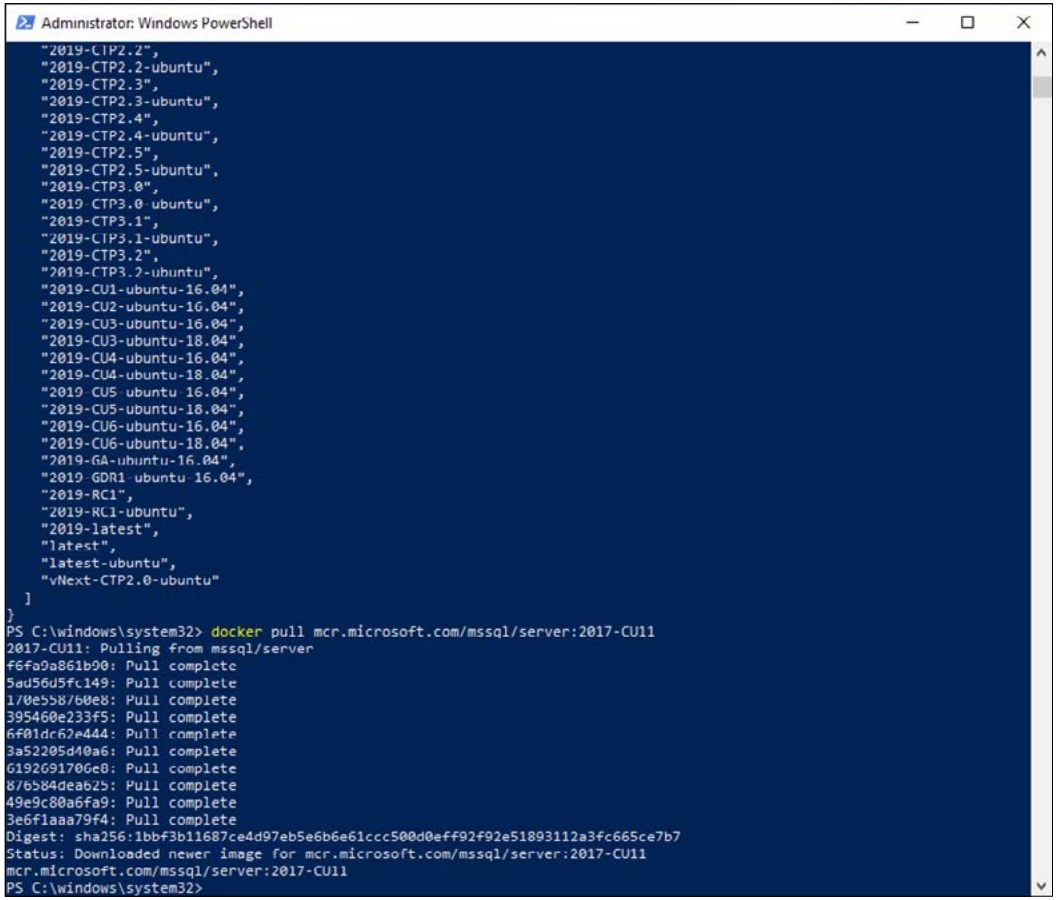
Als nächstes, können wir die Ausgabe des zuvor ausgeführten Kommandos in einer Variable zwischenspeichern und anschließend nur den "Content" ausgeben lassen:

```
$repository = invoke-webrequest https://mcr.microsoft.com/v2/mssql/server/tags/
list
$repository.Content
```

Die folgende Ausgabe sollte nun zu sehen sein:

Mainzer Datenfabrik GmbH
Rotterdamer Platz 2, 55120 Mainz
info@madafa.de
madafa.de
+49 6131 3331612





Um nun anhand der zuvor ausgelesenen Informationen ein SQL Server Image zu downloaden, kann folgender Befehl ausgeführt werden (hier am Beispiel von SQL Server 2017 mit Cumulative Update 11):

```
docker pull mcr.microsoft.com/mssql/server:2017-CU11
```

Nach einiger Zeit (abhängig von der Geschwindigkeit der Internetverbindung) ist der Download beendet.

Mainzer Datenfabrik GmbH
Rotterdamer Platz 2, 55120 Mainz
info@madafa.de
madafa.de
+49 6131 3331612

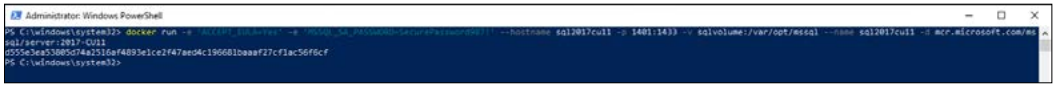
SQL Server mit Docker ausführen

Um einen Container mit SQL Server nutzen zu können müssen zuvor einige Parameter zur richtigen Konfiguration des Containers angegeben werden. Mit Hilfe des nächste Befehls werden die EULA akzeptiert, das Passwort für den SA Benutzer gesetzt, ein Hostname, ein Port, ein zugehöriges Volumen und der Name des Containers konfiguriert:

```
docker run -e 'ACCEPT_EULA=Yes' -e 'MSSQL_SA_PASSWORD=SecurePassword987!'
--hostname sql2017cu11 -p 1401:1433 -v sqlvolume:/var/opt/mssql --name
sql2017cu11 -d mcr.microsoft.com/mssql/server:2017-CU11
```

Der Befehl verwendet die folgenden Parameter zur Angabe dieser Informationen:

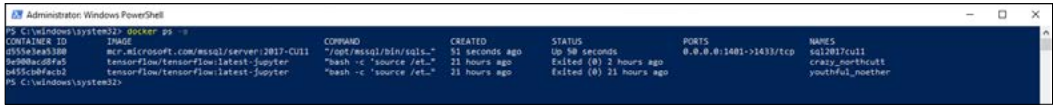
- **-e**: zum setzen von Umgebungsvariablen (also das SA Passwort und das Akzeptieren der Docker EULA)
- **-hostname**: zum Setzen des Hostnamen des Containers
- **-p**: zum Veröffentlichen des Ports des Containers an den Host
- **-v**: zum Mounten eines Volumen. Auf diesem Volumen werden die Daten, die für die SQL Server Datenbanken benötigt werden, gespeichert.
- **-name**: zum Setzen eines Namens für den Container
- **-d**: gibt an, dass der Container im Hintergrund ausgeführt werden soll



Nach der Ausführung des obigen Befehls sollte nach nur ein paar Sekunden ein Container erzeugt werden. Wir können den Status des erzeugten Containers mit dem Befehl

```
docker ps -a
```

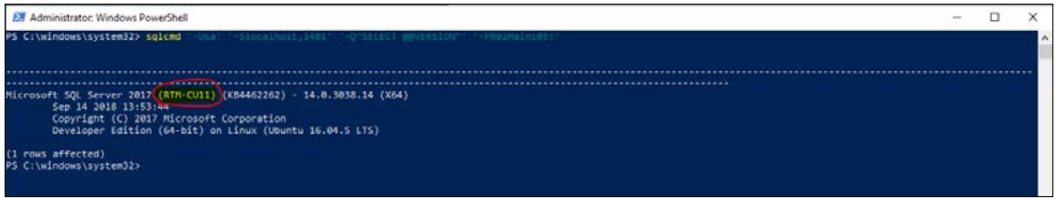
auslesen. An dieser Stelle muss sicher gestellt sein, dass der "STATUS" des zuvor erzeugten Containers "Up" ist.



Sobald der Container erzeugt wurde und im Hintergrund läuft, können wir T-SQL Abfragen gegen den Container ausführen. Hier eine beispielhafte Abfrage zum Auslesen der Version des SQL Servers:

```
sqlcmd '-Usa' '-Slocalhost,1401' '-Q"SELECT @@VERSION"' '-PSecurePassword987!'
```

Hinweis: Das **sqlcmd** Programm kann hier gefunden werden.



Die erfolgreich verarbeitete Abfrage zeigt, dass der Container läuft und von außen erreichbar ist.

Wiederherstellen einer Datenbank

Als Nächstes können wir das Backup einer Datenbank auf den Container kopieren, um diese anschließend wiederherstellen zu können. Das Kopieren einer Datei in den Container kann mit dem **docker cp** Befehl umgesetzt werden:

```
docker cp C:\Backups\WideWorldImporters.bak sql2017cu11:/var/opt/mssql
```

Nachdem die Datei in den Container kopiert wurde, kann ein T-SQL Befehl verwendet werden, um das Backup auf dem bei der Erstellung des Containers angegebenen Volumen wiederherzustellen:

Mainzer Datenfabrik GmbH
Rotterdamer Platz 2, 55120 Mainz
info@madafa.de
madafa.de
+49 6131 3331612

```
docker exec sql2017cu11 /opt/mssql-tools/bin/sqlcmd -S localhost -U sa -P
'SecurePassword987!' -Q"RESTORE DATABASE WideWorldImporters FROM DISK = '/
var/opt/mssql/WideWorldImporters.bak' WITH MOVE 'WWI_Primary' TO '/var/opt/
mssql/data/WideWorldImporters.mdf', MOVE 'WWI_UserData' TO '/var/opt/mssql/
data/WideWorldImporters_userdata.ndf', MOVE 'WWI_Log' TO '/var/opt/mssql/data/
WideWorldImporters.ldf', MOVE 'WWI_InMemory_Data_1' TO '/var/opt/mssql/data/
WideWorldImporters_InMemory_Data_1'"
```

Bei Erfolg wird nach einer kurzen Zeit die Ausgabe "RESTORE DATABASE successfully processed…" ausgegeben. Jetzt ist unsere Datenbank wiederhergestellt!

Mehrere SQL Server Instanzen gleichzeitig verwenden

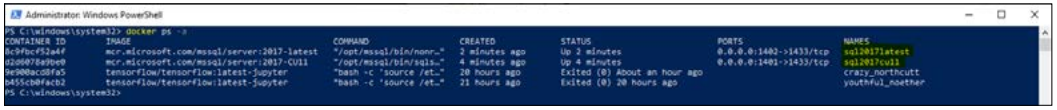
Als Nächstes wollen wir versuchen, mehrere SQL Server Instanz gleichzeitig unter Docker zu verwenden. Mit dem folgenden Befehl erzeugen wir einen weiteren Container:

```
docker run -e 'ACCEPT_EULA=Yes' -e 'MSSQL_SA_PASSWORD=SecurePassword987!'
--hostname sql2017latest -p 1402:1433 -v sqlvolume2:/var/opt/mssql --name
sql2017latest -d mcr.microsoft.com/mssql/server:2017-latest
```

Hierbei sind einige Änderungen gegenüber dem ersten Befehl zu bemerken:

- Der "Hostname" des neuen Containers wurde angepasst
- Der angegebene Port, um den Container von außen zu erreichen, wurde verändert (vorher 1401)
- Ein anderes Volumen wird verwendet
- Der "Name" des neuen Containers wurde angepasst

Nachdem der Befehl ausgeführt wurde, haben wir zwei Container gleichzeitig laufen. Mit dem **docker ps -a** Befehl kann der Status der Container wie zuvor überprüft werden.



Mithilfe von SSMS mit den Containern verbinden

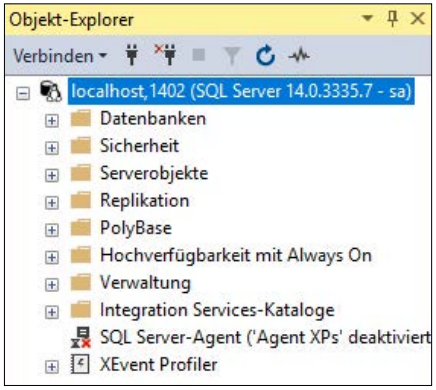
Um zu zeigen, dass sich die erzeugten Container wie eine übliche SQL Server Installation unter Windows verhalten, werden wir uns im nächsten Schritt mit Hilfe des SQL Server Management Studios (SSMS) mit den Containern verbinden.
Nachdem SSMS gestartet wurde, muss eine Verbindung zu einer "Database Engine" mit den folgenden Konfigurationen aufgebaut werden:

Server name: localhost,1401 (oder localhost,1402)
Login: sa
Password: SecurePassword987!

Wir bauen also eine Verbindung zu dem lokalen Container auf, der auf dem Port 1401 zuhört. Nachdem die Eingaben mit einem Klick auf "Verbinden" bestätigt wurden, ist eine Verbindung zu dem Container aufgebaut.

Mainzer Datenfabrik GmbH
Rotterdamer Platz 2, 55120 Mainz
info@madafa.de
madafa.de
+49 6131 3331612

Einer der Unterschiede zu einer "gewöhnlichen" Installation ist das zugehörige Icon im "Object Explorer" in SSMS. Da Docker Linux-basiert ist, ist der Pinguin zu sehen.



Datenbanken Updates unterschiedlicher Cumulative Update-Levels und SQL Server Upgrade

Mit unseren beiden zuvor erstellten Containern haben wir zwei SQL Server mit unterschiedlichen Update-Leveln. Als Nächstes möchte ich eine Situation simulieren, in der beispielsweise Kompatibilitätstests gegen eine Datenbank einer höheren SQL Server Version durchgeführt werden müssen.
Um einen Docker Container auf dem ein SQL Server läuft zu updaten, müssen wir folgende Schritte nacheinander durchführen:

- Der erste Container muss gestoppt werden
- Der zweite Container muss gestoppt werden
- Das Volumen mit der Datenbank muss an den zweiten Container angebunden werden
- Die Tests an der Datenbank können durchgeführt werden

Als Erstes stoppen wir also den SQL Server 2017 CU11 Container. Dafür kann der folgende Befehl verwendet werden:

```
docker stop sql2017cu11
```

Jetzt können wir einen neuen Container erstellen, der das selbe Volumen wie der Container benutzt, den wir zuvor gestoppt haben. Mit folgendem Befehl wird eine neuer Container erzeugt:

```
docker run -e 'ACCEPT_EULA=Y' -e 'MSSQL_SA_PASSWORD=SecurePassword987!'
--hostname sql2017test -p 1403:1433 -v sqlvolume:/var/opt/mssql --name
sql2017test -d mcr.microsoft.com/mssql/server:2017-latest
```

Nachdem dieser Container erzeugt wurde, dauerte es ein paar Minuten bis der Container erreichbar ist. Der SQL Server führt unmittelbar nach seinem Erzeugen ein Update aus, um die Datenbank auf die selbe Version wie die "master" Datenbank zu bringen. Nachdem das Update durchgeführt wurde, ist der Container wieder erreichbar und wir haben spielend leicht ein neues CU aufgespielt.

Genau so einfach ist es auch die SQL Server Version zu upgraden. Im Grunde läuft das Verfahren analog zum vorherigen CU Update, nur dass der neue Container die Version SQL Server 2019 benutzt:

```
docker run -e 'ACCEPT_EULA=Y' -e 'MSSQL_SA_PASSWORD=SecurePassword987!'
--hostname sql2019 -p 1404:1433 -v sqlvolume:/var/opt/mssql --name sql2019 -d
mcr.microsoft.com/mssql/server:2019-latest
```


Mainzer Datenfabrik GmbH
Rotterdamer Platz 2, 55120 Mainz
info@madafa.de
madafa.de
+49 6131 3331612

Docker Container richtig entfernen

Nachdem wir Docker Container erzeugt haben, ist es auch wichtig zu wissen, wie sie wieder entfernt werden können, wenn wir sie nicht mehr benötigen. Zu aller erst sollten wir alle lokalen Container auflisten. Dies kann mit dem folgenden Befehl umgesetzt werden:

```
docker ps -a
```

Die aufgelisteten Container können dann mit der Angabe einer Container-ID oder eines Namen entfernt werden:

```
docker rm my_container_name
docker rm c183ced2fe37
```

Wenn nach dem Ausführen dieses Befehls keine weitere Ausgabe erscheint, bedeutet das, dass der Befehl erfolgreich ausgeführt wurde.
Sollte der gelöschte Container über ein Volumen verfügt haben, muss dieses ebenfalls entfernt werden. Zum Auflisten aller Volumen kann folgender Befehl verwendet werden:

```
docker volume ls
```

Anschließend kann mit dem folgenden Befehl eines der aufgelisteten Volumen entfernt werden:

```
docker volume rm my_volume_name
```

Zu guter Letzt können die lokalen Images, die verwendet wurden, entfernt werden. Um alle lokalen Images aufzulisten, muss folgender Befehl ausgeführt werden:

```
docker images
```

Um nun eines der aufgelisteten Images zu entfernen, muss die ID des Images verwendet werden:

```
docker rmi ba266fae5320
```

MSSQL Container mit .NET Core verwenden

Um SQL Server Container mit .NET Core Anwendungen zu benutzen, werden wir das "**entity framework**", genauer die **DBContext** Klasse, verwenden, um einen Kontext zu erzeugen. Eine typische Implementation der Klasse sieht dabei wie folgt aus:

```
using Microsoft.EntityFrameworkCore;

public class MyDbContext: DbContext {
    public MyDbContext (DbContextOptions options) : base (options) { }
    public DbSet<User> Users {get; set;}
}
```

Das **DbSet** vom Typ **User** repräsentiert dabei eine Entität, für die ein Äquivalent in der MSSQL Datenbank existiert. Zur besseren Veranschaulichung hier noch die Implementation dieser **User** Klasse:

```
public class User {
    public int Id { get; set; }
    public string Name { get; set; }
}
```

Jetzt müssen wir noch dafür sorgen, dass die .NET Core Anwendung und der SQL Server Container miteinander kommunizieren können. Mit einer simplen **PackageReference** lässt sich auch dieses Problem lösen:

```
<PackageReference Include="Microsoft.EntityFrameworkCore.SqlServer"
Version="2.2.0" />
```

Zum Schluss muss nun das **DbContext** Objekt benutzt werden, um eine Verbindung zum Container aufzubauen. Dafür benötigen wir einen sogenannten Connection-String. Das ist eine Zeichenkette, die alle für den Verbindungsaufbau wichtigen Informationen enthält:

```
var connection = @"Server=127.0.0.1,1433;Database=my_
db;User=sa;Password=SecurePassword987!;";
```


Mainzer Datenfabrik GmbH
Rotterdamer Platz 2, 55120 Mainz
info@madafa.de
madafa.de
+49 6131 3331612

Dabei repräsentieren die Parameter folgende Informationen:

Parameter	Wert	Beschreibung
Server	127.0.0.1,1433	Die Angabe über IP/Port des Containers. Der Port muss dabei mit dem zuvor in der Erzeugung des Containers angegebenen Ports übereinstimmen.
Database	my_db	Der Name der Datenbank, mit der die Verbindung aufgebaut werden soll.
User	sa	Der Name des Benutzers, mit dem die Verbindung aufgebaut wird.
Password	SecurePassword987!	Das Passwort des Benutzers, mit dem die Verbindung aufgebaut wird.

Der erzeugte Datenbank-Kontext kann nun in der **ConfigurationServices** Methode der **Startup.cs** Datei verwendet werden:

```
public void ConfigureServices(IServiceCollection services)
        {
            // Database connection string.
            var connection = @"Server=127.0.0.1,1433;Database=main_
db;User=sa;Password=qwertY12@3;";

            services.AddDbContext<MyDbContext>(options => options.
UseSqlServer(connection));
...
```

Zu guter Letzt können wir überprüfen, dass alles funktioniert. Dafür erzeugen wir eine Migration mithilfe des **entity frameworks**, um die Tabelle **User** zu erzeugen:

```
dotnet ef migrations add InitDB
```

Dann führen wir sie aus:

```
dotnet ef database update
```

Hat alles funktioniert, sollte nun die Tabelle **User** in der in der Verbindung angegebenen Datenbank existieren.

Fazit

Wie vermutlich leicht zu erkennen ist, bieten Docker Container eine enorm flexible und rapide Möglichkeit der Entwicklung auf unterschiedlichen SQL Server Versionen an. Der ermüdende und aufwendige Prozess, eine virtuelle Maschine bereitzustellen und zu konfigurieren, entfällt vollständig.

Mainzer Datenfabrik GmbH
Rotterdamer Platz 2, 55120 Mainz
info@madafa.de
madafa.de
+49 6131 3331612