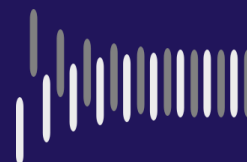
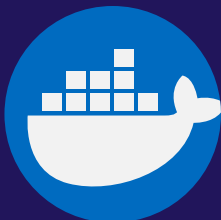


# Docker Grundlagen

## Kursheft

### Kapitel 5: Praxisprojekt

### Datenbank



DATAMICS  
machine intelligence consulting services

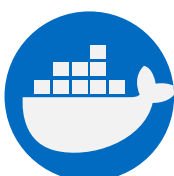
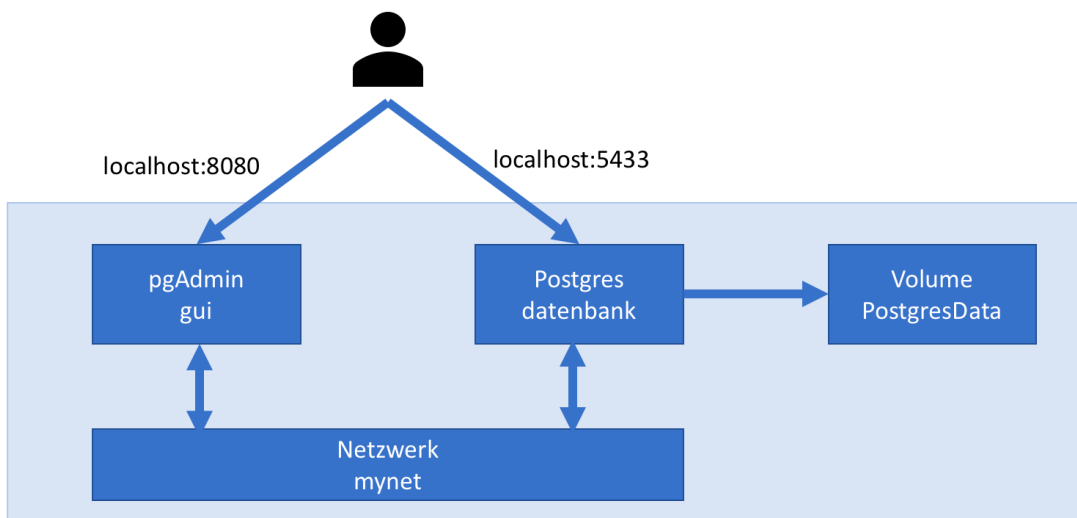
## Praxisprojekt SQL Datenbank

Seitdem ich Docker kenne, installiere ich kaum noch meine eigene direkte Installation von Entwicklungssoftware auf meinem lokalen Computer. Egal ob Datenbankserver (d.h. Postgres, pgAdmin) oder Messaging-Systeme (d.h. Kafka), Big Data System (d.h. Spark, Hadoop) – ich versuche fast immer, ein geeignetes Docker-Image zu finden oder zu erstellen, das ich während der Entwicklung verwenden kann.

Die Installation von Software ist schwierig. Und es hat nichts mit deiner Expertise als Entwickler zu tun. Wir alle haben schon erlebt, wie viel Zeit durch Versionskonflikte, kryptische Build Fehlermeldungen und Fehler zur fehlenden Abhängigkeit verschwendet wird, jedes Mal, wenn wir mit der Installation einer neuen Software begonnen haben.

Dann haben wir unzählige Stunden damit verbracht, Codeausschnitte von Stack Overflow auf unser Terminal zu kopieren und sie in der Hoffnung auszuführen, dass einer von ihnen Probleme bei der Installation magisch löst und die Software laufen lässt. Das Ergebnis sind vor allem Verzweiflung, Frustration und Produktivitätsverlust.

Docker bietet einen Ausweg aus diesem Schlamassel, indem die Aufgabe der Installation und Ausführung von Software auf nur zwei Befehle reduziert wird (Docker Run und Docker Pull). In diesem Kapitel werden wir den Prozess für das Aufsetzen eines komplizierten Datenbanksystems mit Frontend in Aktion sehen, indem wir Schritt für Schritt betrachten, wie einfach es ist, eine Postgres-Installation mit Docker einzurichten. Das Ziel System soll dann folgendermaßen aussehen:



Erstelle zuerst ein Netzwerk, das die GUI und die Datenbank verbindet.

**Hinweis:** Wir verwenden hier aus Trainingszwecken für Docker die Netzwerkkomponente. Du könntest auch die GUI bei dir lokal installieren und dich dann direkt auf die Datenbank verbinden.

```
docker network create mynet
```

### Was ist mit Postgres?

Das Anlegen und Arbeiten mit einem Container mit Postgres ist recht einfach und kann mit dem folgenden Befehl ausgeführt werden. Er erstellt einen Container und gibt den von Postgre verwendeten Port aus, damit vom Host aus darauf zugegriffen werden kann.

Das Problem bei diesem Ansatz ist, dass die Daten verloren gehen, wenn du den Container aus irgendeinem Grund jemals neu erstellen musst, z.B. wenn eine neue Version von Postgres veröffentlicht wird.

Zum Glück habe ich einen Blog-Bertrag gefunden ([diesen hier](#)), in dem gezeigt wird, wie ein zweiter Container für die Daten verwendet wird, sodass der Postgres-Container bei Bedarf zerstört und neu erstellt werden kann. Der folgende Befehl zeigt das Erstellen des Datencontainers:

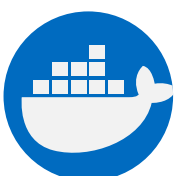
```
docker create -v /var/lib/postgresql/data --name PostgresData alpine
```

Der oben erstellte Container mit dem Namen PostgresData basiert auf einem Alpine Image. Es ist wichtig, dass der Parameter -v mit dem von Postgres erwarteten Pfad übereinstimmt.

Nachdem wir nun einen Container haben, der unsere Daten schützt, können wir den eigentlichen Postgres-Container mit dem folgenden Befehl erstellen.

```
docker run -p 5433:5432 --name datenbank --net mynet -e POSTGRES_PASSWORD=123 -d --volumes-from PostgresData postgres
```

Der einzige Unterschied zum Run-Befehl des ersten Beispiels besteht im Hinzufügen von `--volumes-from PostgresData`, wodurch der Container angewiesen wird, den PostgresData-Container zu verwenden.



Wenn du den Befehl `docker ps -a` ausführst, werden alle deine Container angezeigt.

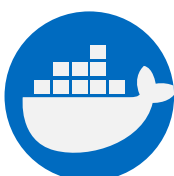
```
docker ps -a
```

```
docker exec -ti datenbank pg_isready -h localhost -p 5432
```

```
k docker ps -a
CONTAINER ID   IMAGE      COMMAND                  CREATED        STATUS        PORTS                    NAMES
f6a07cc31454   postgres  "docker-entrypoint.s..." 9 minutes ago  Up 3 seconds  0.0.0.0:5432->5432/tcp   Moon
71fe42ad8d8e   alpine    "/bin/sh"                12 minutes ago Created                                PostgresData
```

Wie du in diesem Beispiel sehen kannst, haben wir nun zwei Container, von denen nur einer tatsächlich ausgeführt wird.

**Hinweis:** Stelle sicher, dass du den Datencontainer nicht entfernst, nur weil er niemals als ausgeführt angezeigt wird!



## Pgadmin

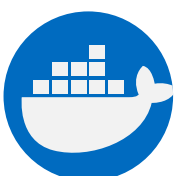
Manchmal ist es wünschenswert, dass Benutzer über einen Reverse-Proxy eine Verbindung zu pgAdmin herstellen und nicht direkt zu dem Container, in dem es ausgeführt wird. Die folgenden Beispiele zeigen, wie dies funktioniert. Bei herkömmlichen Reverse-Proxy-Servern wie [Nginx](#) wird pgAdmin in einem Container auf demselben Host ausgeführt, wobei Port 8080 auf dem Host dem Port 80 auf dem Container zugeordnet ist. Beispiel:

```
docker run -p 8080:80 \  
-e "PGADMIN_DEFAULT_EMAIL=user@domain.com" \  
-e "PGADMIN_DEFAULT_PASSWORD=234" \  
--name gui \  
-d --net mynet dpage/pgadmin4
```

Schauen wir uns das Netzwerk an:

```
docker network inspect mynet
```

```
[  
  {  
    "Name": "mynet",  
    "Id": "8a76c46f19bf54eb969ca73dbbe299c06ec18a732ac724a51db0ca1ae53d8cdd",  
    "Created": "2019-12-06T15:01:57.6061781Z",  
    "Scope": "local",  
    "Driver": "bridge",  
    "EnableIPv6": false,  
    "IPAM": {  
      "Driver": "default",  
      "Options": {},  
      "Config": [  
        {  
          "Subnet": "172.22.0.0/16",  
          "Gateway": "172.22.0.1"  
        }  
      ]  
    }  
  }  
]
```

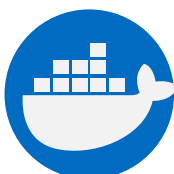


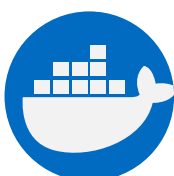
```
{
  "Internal": false,
  "Attachable": false,
  "Ingress": false,
  "ConfigFrom": {
    "Network": ""
  },
  "ConfigOnly": false,
  "Containers": {
    "472d752653a8a3424839f817224d59957002665596da7cf16b16311b924ed9ea": {
      "Name": "gui",
      "EndpointID": "ead9299b55d9a956ef3a7edb76f79c2c6bceef83743ba207556edcef44904bd2",
      "MacAddress": "02:42:ac:16:00:03",
      "IPv4Address": "172.22.0.3/16",
      "IPv6Address": ""
    },
    "785ae73cd135cbe81d4fa5ba814fea69f214033542a9bf714dcf4582b5f7aa70": {
      "Name": "datenbank",
      "EndpointID": "c8df20f766566b26f05d2c91e20b9fd2668b20add9502dbc76dd391e9e9ec843",
      "MacAddress": "02:42:ac:16:00:02",
      "IPv4Address": "172.22.0.2/16",
      "IPv6Address": ""
    }
  },
  "Options": {},
  "Labels": {}
}
```



```
docker exec -ti gui ping 172.22.0.2
```

```
docker exec -ti gui pg_isready -h 172.22.0.2 -p 5432
```

<http://localhost:8080>







 **Erstellen - Server** 

**Allgemein** Connection SSL SSH Tunnel Erweitert



**Name**

Datenbank



**Servergruppe**

 Servers 

**Background**



**Foreground**






 

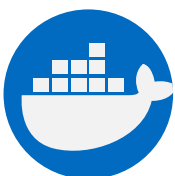
**Jetzt verbinden?**

☒

**Kommentare**

 Either Host name, Address or Service must be specified. 

   Abbrechen  Zurücksetzen  Speichern





## Erstellen - Server

Allgemein Connection SSL SSH Tunnel Erweitert

Hostname/-  
adresse

172.22.0.2

Port

5433

Wartungsdatenbank

postgres

Benutzername

user

Passwort

...


Passwort  
speichern?



Rolle

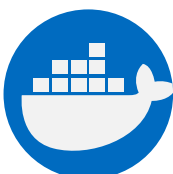
Service



 Abbrechen

 Zurücksetzen

 Speichern



## Erstellen - Server

Allgemein Connection SSL SSH Tunnel Erweitert

Hostname/-  
adresse

172.22.0.2

Port

5433

Wartungsdatenbank

postgres

Benutzername

postgres

Passwort

...


Passwort  
speichern?




Rolle

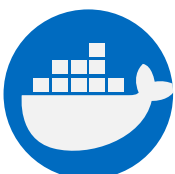
Service



 Abbrechen

 Zurücksetzen

 Speichern



Beenden der Container:

```
docker rm -f datenbank  
docker rm -f gui  
docker rm -f PostgresData
```

