SQL Server mit Kubernetes auf Linux (Teil 2)

Kathegorie
SQL Server

Simon
Schlaipfer

Wir haben erfolgreich einen SQL Server Container auf Linux bereitgestellt, der in einem Pod auf Kubernetes ausgeführt wird. Den Pod haben wir mithilfe eines Service verfügbar gemacht. Dies ist für den Anfang auch nicht schlecht, allerdings können wir noch persistenten Speicher für unsere Daten bereitstellen. Weiterhin könnten wir mittels YAML ein "geheimes" Objekt erstellen, das unser Datenbank Passwort speichert. In diesem Folgebeitrag wollen wir also unsere bisheriges Kubernetes Setup-Kenntnisse um die eben genannten Features erweitern.

Persistenter Speicher

Als Erstes wollen wir uns um den persistenten Speicher für unsere Daten kümmern. Hierfür verwenden wir ein von Kubernetes bereitgestelltes Objekt namens **Persistent Volume**. Hierbei gibt es, je nachdem welche Umgebung genutzt wird und für welchen Zweck es eingesetzt werden soll, verschiedene Volume Typen. Da wir in unserem Beispiel MicroK8s als ein Single Node Cluster verwenden, wählen wir den **HostPath** Typ.

Unser **Persistent Volume** Objekt erstellen wir erneut mithilfe von YAML. Hier müssen wir nun die Kapazität, den Pfad und den Zugriffsmodus für unser Volume konfigurieren:

```
apiVersion: v1
kind: PersistentVolume
metadata:
name: sqldata
spec:
capacity:
storage: 500Mi
storageClassName: sqlserver
accessModes:
- ReadWriteMany
hostPath:
path: "/tmp/sqldata"
```

Hier spezifizieren wir unser Objekt als PersistentVolume, nennen es sqldata und weisen ihm 500 Megabytes an Speicher zu. Als Pfad, auf dem wir dann unser PersistentVolume Objekt "mounten" möchten, definieren wir /tmp/sqldata. Dies entspricht einem Pfad auf dem Host, der von unserem Objekt genutzt wird, um dort Daten zu speichern.

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

Damit unser PersistentVolume Objekt auch wirklich funktionieren kann, müssen wir noch ein weiteres Objekt erstellen, einen sogenannten **Persistent Volume Claim**, oder kurz: **Claim**. Dieser **Claim** erlaubt unserem Pod, das **Persistent Volume** Objekt zu verwenden. Um einen **Claim** zu definieren, müssen wir zum einen die Art des Volumes spezifizieren, das mittels des **Claims** verwendet werden soll, sowie dessen Kapazität angeben.

Fertig konfiguriert sieht unser **Claim** so aus:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
name: dbclaim
spec:
accessModes:
- ReadWriteMany
storageClassName: sqlserver
resources:
requests:
storage: 500M
```

Wie auch in unserem ersten Beitrag werden wir beide Objekt in einem YAML File konfigurieren und sie mit drei Bindestrichen trennen. Wir erstellen das File mit dem Kommando

```
touch volume.yaml
```

Um es anschließend mit einem Texteditor zu öffnen, das Kommando

```
vi volume.yaml
```

ausführen. Um den Editor wieder zu verlassen und Änderungen zu speichern drücke **ESCAPE** und zwei mal **SHIFT + Z**.

Das gesamte File sieht dann wie folgt aus:

```
apiVersion: v1
kind: PersistentVolume
metadata:
name: sqldata
spec:
capacity:
storage: 500Mi
storageClassName: sqlserver
accessModes:
- ReadWriteMany
hostPath:
path: "/tmp/sqldata"
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
name: dbclaim
spec:
accessModes:
- ReadWriteMany
storageClassName: sqlserver
resources:
requests:
storage: 400Mi
```

Mit dem folgenden Kommando können wir unser YAML File ausführen und das Volumen erstellen.

```
microk8s.kubectl apply -f storage.yaml
```


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

Um zu überprüfen, ob auch wirklich alles erfolgreich war und wir das Volume sowie den **Claim** erstellen konnten, führen wir dieses Kommando aus:

```
microk8s.kubectl get pv
microk8s.kubectl get pvc
```

Das Volume:

```
simon@kubsql:~$ microk8s.kubectl get pv
NAME                                       CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM
                          STORAGECLASS       REASON   AGE
pvc-74343a16-e932-487a-9d27-92aa072a9b7a   20Gi       RWX            Delete           Bound    conta
iner-registry/registry-claim   microk8s-hostpath           28h
sqldata                                    500Mi      RWX            Retain           Bound    defau
lt/dbclaim                     sqlserver                   86s
```

Der Claim:

```
simon@kubsql:~$ microk8s.kubectl get pvc
NAME      STATUS   VOLUME    CAPACITY   ACCESS MODES   STORAGECLASS   AGE
dbclaim   Bound    sqldata   500Mi      RWX            sqlserver      2m47s
```

Wie wir sehen können, wurde beides erfolgreich erstellt. Als letzten Schritt müssen wir unserem Pod die Möglichkeit geben, den eben erstellten **Claim** zu verwenden. Hierfür müssen wir unser Volume zu unserem Pod hinzufügen und es an unserem SQL Server Container an der Stelle **/var/opt/mssql** "mounten".

Abschließend erstellen wir noch einen **Init-Container**, dem Benutzer **mssql** mit einer UID von 10001 Zugriff auf das Volumen gewährt. Dadurch erhalten SQL Server-Container, die nicht als Root-Benutzer ausgeführt werden, die Berechtigung, auf das Volume zu schreiben.

Um nun dem Pod die Möglichkeit zu geben, den **Claim** zu verwenden, müssen wir das sql-server.yaml File aus unserem ersten Beitrag wie folgt erweitern:

```
apiVersion: v1
kind: Pod
metadata:
labels:
run: mydb
name: mydb
spec:
volumes:
- name: sqldata-storage
persistentVolumeClaim:
claimName:
dbclaim initContainers:
- name: volume-permissions
image: busybox
command: ["sh", "-c", "chown -R 10001:0 /var/opt/mssql"]
volumeMounts:
- mountPath: "/var/opt/mssql"
name: sqldata-storage
containers:
- image: mcr.microsoft.com/mssql/server
name: mydb
env:
- name: ACCEPT_EULA
value: "Y"
- name: SA_PASSWORD
value: TestingPassword1
- name: MSSQL_PID
value: Developer
ports:
- containerPort: 1433
name: mydb
volumeMounts:
- mountPath: "/var/opt/mssql"
name: sqldata-storage
---
apiVersion: v1
```


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

```
kind: Service
metadata:
name: mydb
spec:
type: NodePort
ports:
- port: 1433
nodePort: 31433
selector:
run: mydb
```

Da Volumes und VolumeMounts nicht über Kubernetes aktualisiert werden können, müssen wir unseren vorherigen SQL Server löschen und anschließend einen Neuen erstellen, der dann über das Volume verfügt.

Dies tun wir mit dem Kommando:

```
microk8s.kubectl delete -f sql-server.yaml
microk8s.kubectl apply -f sql-server.yaml
```

Das geheime Passwort

Bisher haben wir unser Passwort direkt in unserem Manifest erstellt, was offensichtlich nicht besonders sicher ist. Um unsere Sicherheit zu erhöhen, wollen wir nun noch ein zusätzliches "Geheimes" Objekt erstellen, das unser Datenbanken-Passwort speichert.

Wir erstellen hierfür ein sogenanntes **Kubernetes Secret Object** und nennen es **sql-password**. Hierfür führen wir folgendes Kommando aus:

```
microk8s.kubectl create secret generic sql-password --from-
literal=sa_password=TestingPassword1
```

Jetzt müssen wir erneut unser sql-server.yaml File bearbeiten und unser Secret Object als Umgebungsvariable hinzufügen:

```
apiVersion: v1
kind: Pod
metadata:
labels:
run: mydb
name: mydb
spec:
volumes:
- name: sqldata-storage
persistentVolumeClaim:
claimName: dbclaim
initContainers:
- name: volume-permissions
image: busybox
command: ["sh", "-c", "chown -R 10001:0 /var/opt/mssql"]
volumeMounts:
- mountPath: "/var/opt/mssql"
name: sqldata-storage
containers:
- image: mcr.microsoft.com/mssql/server
name: mydb
env:
- name: ACCEPT_EULA
value: "Y"
- name: SA_PASSWORD
valueFrom:
secretKeyRef:
name: sql-password
key: sa_password
- name: MSSQL_PID
value: Developer
ports:
- containerPort: 1433
```


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

```
name: mydb
volumeMounts:
- mountPath: "/var/opt/mssql"
name: sqldata-storage
---
apiVersion: v1
kind: Service
metadata:
name: mydb
spec:
type:
NodePort ports:
- port: 1433
nodePort: 31433
selector:
run: mydb
```

Und erneut müssen wir unseren SQL Server löschen und neu erstellen, um nun auch das **Secret Object** nutzen zu können.

Nun haben wir unserem Kubernetes Single Node Cluster erfolgreich einen persistenten Speicher und ein geheimes Passwort Objekt hinzugefügt.

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