SQL Server mit Kubernetes auf Linux (Teil 1)

Kathegorie
SQL Server

Simon
Schlaipfer

In diesem Beitrag wollen wir uns mit der Bereitstellung von SQL Server in Kombination mit Kubernetes auf einem Ubuntu Server auseinandersetzen. Hierfür wollen wir anhand eines praktischen Beispiels zwei Kubernetes Objekte, einen Pod und einen SQL Server Service erstellen. Haben wir das alles auf einem Cluster zum Laufen gebracht, werden wir uns mittels des Azure Data Studios mit diesem verbinden. Als Umgebung werden wir einen Ubuntu 18.04 Server mit **MicroK8s** verwenden.

MicroK8s installieren und Konfigurieren

Bevor wir mit der Bereitstellung unseres Kubernetes Clusters beginnen können, müssen wir zunächst **MicroK8s** installieren und konfigurieren. Bei **MicroK8s** handelt es sich um eine zertifizierte Kubernetes Distribution von Canonical, den Schöpfern von Ubuntu. **MicroK8s** stellt ein einziges Paket bereit, mit dem sich eine komplette Kubernetes Umgebung installieren lässt.

MicroK8s ist kompatibel mit Linux, Windows und macOS. In unserem Beispiel nutzen wir einen Ubuntu 18.04 Server.

Um **MicroK8s** zu installieren, loggen wir uns zunächst auf unserer Ubuntu Maschine ein und führen anschließend folgendes Kommando aus:

```
sudo snap install microk8s --classic
```

```
simon@kubernetes_sql:~$ sudo snap install microk8s --classic
[sudo] password for simon:
microk8s v1.18.6 from Canonical♦ installed
simon@kubernetes_sql:~$
```

Nach erfolgreicher Installation kann mit dem folgenden Kommando überprüft werden, ob **MicroK8s** auch tatsächlich fehlerfrei auf der Maschine läuft:

```
microk8s.status
```


Mainzer Datenfabrik GmbH
Rotterdamer Platz 2, 55120 Mainz

info@madafa.de
madafa.de

+49 6131 3331612

Weiterhin kann mit diesem Kommando geprüft werden, welche Addons für **MicroK8s** aktiviert sind:

```
simon@kubsql:~$ microk8s.status
microk8s is running
addons:
ambassador: disabled
cilium: disabled
dashboard: disabled
dns: disabled
fluentd: disabled
gpu: disabled
helm: disabled
helm3: disabled
host-access: disabled
ingress: disabled
istio: disabled
jaeger: disabled
knative: disabled
```

Hinweis: Es ist im Allgemeinen darauf zu achten, dass der Computername der Ubuntu Maschine, auf der **MicroK8s** ausgeführt werden soll, weder Großbuchstaben noch Unterstriche enthält. Andernfalls kann Kubernetes die Maschine nicht als Node registrieren.

Nun haben wir erfolgreich ein Single-Node Kubernetes Cluster erzeugt. Jetzt können wir einfache kubectl Kommandos ausführen und uns beispielsweise einer Liste aller Nodes innerhalb unseres Clusters anzeigen lassen oder uns einen Überblick über die ausgeführten Services verschaffen.

Liste der Nodes:

```
microk8s.kubectl get nodes
```

```
simon@kubsql:~$ microk8s.kubectl get nodes
NAME     STATUS   ROLES    AGE     VERSION
kubsql   Ready    <none>   9m34s   v1.18.6-1+64f53401f200a7
```

Liste der Services:

```
microk8s.kubectl get services
```

```
simon@kubsql:~$ microk8s.kubectl get services
NAME         TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.152.183.1   <none>        443/TCP   13m
```

Es besteht nun die Möglichkeit, gewisse Dienste wie ein Dashboard, DNS oder Prometheus für unser Cluster zu aktivieren.

Wir führen hierfür folgendes Kommando aus:

```
sudo microk8s.enable dns dashboard registry prometheus metrics-server
```

Damit wir unser Dashboard ansehen können, müssen wir uns mit einem sogenannten **Token** authentifizieren. Diesen erhalten wir mit folgenden Kommandos:

```
token=$(microk8s.kubectl -n kube-system get secret | grep default-token | cut -d
" " -f1)
microk8s.kubectl -n kube-system describe secret $token
```

Die Ausgabe des letzten Kommandos entspricht dem **Token**. Diesen kopieren wir. Nun benötigen wir noch die Adresse für unser Dashboard. Hierfür führen wir dieses Kommando aus:

```
microk8s.kubectl get services -n kube-system -l k8s-app=kubernetes-dashboard
```

```
NAME                   TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
kubernetes-dashboard   ClusterIP   10.152.183.77   <none>        443/TCP   46m
```


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

Hieraus können wir nun entnehmen, welche IP Adresse wir zum Verbinden mit unserem Dashboard benötigen. In unserem Fall müssen wir https://10.152.183.77:443 in der Internet Suchleiste eingeben und anschließend die aufkommende Warnung akzeptieren. Mit dem vorher kopierten Token können wir uns nun über die Login Seite mit dem Dashboard verbinden.

Das Kubernetes Manifest erstellen

Nachdem wir nun erfolgreich unsere Ubuntu Maschine erstellt und **MicroK8s** als Kubernetes Cluster konfiguriert haben, können wir mit dem Erstellen eines Kubernetes Manifestes beginnen.

Um Objekte zu definieren, verwendet Kubernetes, wie Docker, YAML. Die einzelnen Objekte werden dann in einem Ordner, dem Manifest, gespeichert. In unserem Beispiel werden wir mehrere Objekte in einem Manifest speichern, das die Objekte, den Pod und den Service erstellt.

Wir beginnen also mit dem Erstellen unseres Manifests, das wir **sql-server.yaml** nennen. Das erfolgt mit diesem Kommando:

```
touch sql-server.yaml
```

Um unsere YAML File nun zu bearbeiten, führen wir diesen Befehl aus:

```
vi sql-server.yaml
```

Um nach dem Bearbeiten die File zu speichern und den Editor zu verlassen, klicken wir einmal **ESCAPE** und anschließend zweimal **SHIFT + Z**.

Als Erstes erstellen wir unseren Pod:

```
apiVersion: v1
kind: Pod
metadata:
labels:
run: mydb
name: mydb
spec:
containers:
- image: mcr.microsoft.com/mssql/server
name: mydb
```

Wir geben ihm den Namen mydb und verwenden den von Microsoft bereitgestellten SQL Server Container. Für den Container ist es notwendig Umgebungsvariablen für das Passwort, das Akzeptieren der EULA und das Definieren einer SQL Server Produktversion zu setzen. Mit den gesetzten Variablen sieht unsere YAML File so aus:

```
apiVersion: v1
kind: Pod
metadata:
labels:
run: mydb
name: mydb
spec: 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
```


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

Nachdem wir nun einen Pod erstellt haben, müssen wir einen Service erstellen, der unsere Datenbank auf unserem Cluster verfügbar macht. Wir konfigurieren ihn wie folgt:

```
apiVersion: v1
kind: Service
metadata:
name: mydb
spec:
type: NodePort
ports:
- port: 1433
nodePort: 31433
selector:
run: mydb
```

Unseren Service nennen wir ebenfalls mydb und definieren seinen Typ als **NodePort**. Den internen Port spezifizieren wir mit 1433 und unseren NodePort mit 31433. Anschließend müssen wir einen Selektor definieren, um unserem Service mitzuteilen, welchem Port er zugeordnet werden muss. Dies können wir zu unserer Pod-Definition hinzufügen und diese beiden Objekte durch drei Bindestriche trennen.

Komplett sieht unser YAML File so aus:

```
apiVersion: v1
kind: Pod
metadata:
labels:
run: mydb
name: mydb
spec: containers:
- image: mcr.microsoft.com/mssql/server name: mydb
env:
- name: ACCEPT_EULA
value: "Y"
- name: SA_PASSWORD
value: passwort123
- name: MSSQL_PID
value:
Developer ports:
- containerPort: 1433
name: mydb
---
apiVersion: v1
kind: Service
metadata:
name: mydb
spec:
type: NodePort
ports:
- port: 1433
nodePort: 31433
selector:
run: mydb
```

Das Kubernetes Manifest ausführen

Jetzt wurden alle wichtigen Ressourcen in unserem YAML File definiert, also können wir unseren SQL Server und unseren Service erstellen. Hierfür prüfen wir zunächst, welche Pods auf unserem Cluster definiert sind. Hierfür führen wir das Kommando aus:

```
microk8s.kubectl get pods
```

Als Ausgabe erhalten wir die Nachricht, dass im Standard Namespace unseres **MicroK8s** Clusters keine Pods definiert sind. Da das Kommando aber erfolgreich ausgeführt wurde, zeigt uns, dass unser Cluster läuft.

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

Als Nächstes überprüfen wir die Services mit:

```
microk8s.kubectl get svc
```

```
simon@kubsql:~$ microk8s.kubectl get svc
NAME         TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.152.183.1   <none>        443/TCP   22h
```

Wie zu erwarten läuft unser Kubernetes Service. Nun erstellen wir mithilfe unseres YAML Files unseren SQL Server und den Service. Hierfür führen wir dieses Kommando aus:

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

```
simon@kubsql:~$ microk8s.kubectl apply -f sql-server.yaml
pod/mydb unchanged
service/mydb created
```

Überprüfen wir nun erneut unsere Pods und Services, erhalten wir:

Pods:

```
simon@kubsql:~$ microk8s.kubectl get pods
NAME    READY   STATUS    RESTARTS   AGE
mydb    1/1     Running   0          23m
```

Services:

```
simon@kubsql:~$ microk8s.kubectl get svc
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
kubernetes   ClusterIP   10.152.183.1    <none>        443/TCP          23h
mydb         NodePort    10.152.183.85   <none>        1433:31433/TCP   16m
```

Wir können sehen, dass sowohl unser SQL Server Pod als auch unser Service innerhalb unseres Clusters laufen.

Verbinden mit Azure Studio

Als letzten Schritt wollen wir uns mithilfe von Azure Data Studio mit unserem SQL Server verbinden. Hierfür müssen wir Azure Studio zunächst auf unserer Ubuntu Maschine installieren. Dies tun wir mit dem Kommando:

```
wget https://github.com/microsoft/azuredatastudio/releases/download/1.14.1/
azuredatastudio-linux-1.14.1.deb
sudo dpkg -i azuredatastudio-linux-1.14.1.deb
```


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