PostgreSQL mit Zalando in Kubernetes bereitstellen

In dieser Anleitung erfahren Sie, wie Sie mit dem Zalando Postgres-Operator Postgres-Cluster in bereitstellen.

PostgreSQL ist ein leistungsstarkes, objektrelationales Open-Source-Datenbanksystem, das über mehrere Jahrzehnte hinweg aktiv entwickelt wurde und sich einen guten Ruf für Zuverlässigkeit, Robustheit von Features und Leistung verdient hat.

Vorteile

Zalando bietet folgende Vorteile:

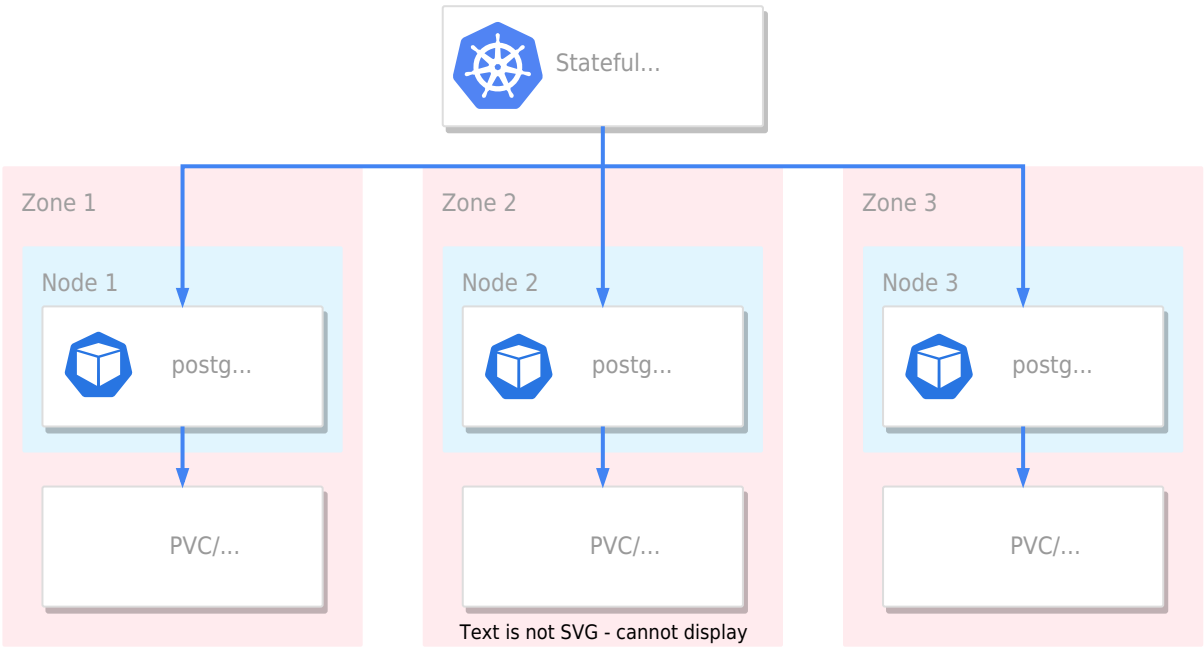
- Eine deklarative und Kubernetes-native Methode zum Verwalten und Konfigurieren der PostgreSQL-Cluster
- Hochverfügbarkeit von Patroni
- Unterstützung der Sicherungsverwaltung mithilfe von Cloud Storage-Buckets
- Rolling Updates für Postgres-Clusteränderungen, einschließlich schneller Nebenversionsaktualisierungen
- Deklarative Nutzerverwaltung mit Passworterstellung und -rotation mithilfe benutzerdefinierter Ressourcen
- Unterstützung für TLS, Zertifikatsrotation und Verbindungspools
- Clusterklonen und Datenreplikation

Bereitstellungsarchitektur

In dieser Anleitung verwenden Sie den Zalando Postgres-Operator, um einen hochverfügbaren Postgres-Cluster bereitzustellen und zu konfigurieren. Der Cluster hat ein Leader-Replikat und zwei schreibgeschützte Standby-Replikate, die von Patroni verwaltet werden. Patroni ist eine von Zalando verwaltete Open-Source-Lösung, um Postgres Hochverfügbarkeits- und automatische Failover-Funktionen zu bieten. Wenn ein Leader ausfällt, wird ein Standby-Replikat automatisch zu einer Leader-Rolle hochgestuft.

Außerdem stellen Sie einen hochverfügbaren Kubernetes-Cluster für Postgres bereit, wobei mehrere Kubernetes-Knoten über verschiedene Verfügbarkeitszonen verteilt sind. Diese Konfiguration sorgt für Fehlertoleranz, Skalierbarkeit und geografische Redundanz. Damit können Rolling Updates und Wartungen durchgeführt werden, während SLAs für Verfügbarkeit und Verfügbarkeit bereitgestellt werden.

Das folgende Diagramm zeigt einen Postgres-Cluster, der auf mehreren Knoten und Zonen in einem Kubernetes-Cluster ausgeführt wird:



Im Diagramm wird die Postgres-`Stateful Set` auf drei Knoten in drei verschiedenen Zonen bereitgestellt. Sie können steuern, wie Kubernetes auf Knoten bereitgestellt wird. Dazu legen Sie den erforderlichen Pod-Regeln für Affinität und Anti-Affinität für die postgresql benutzerdefinierte Ressourcenspezifikation. Wenn eine Zone gemäß der empfohlenen Konfiguration ausfällt, verschiebt Kubernetes die Pods auf andere verfügbare Knoten in Ihrem Cluster. Zum Speichern von Daten verwenden Sie SSD-Laufwerke (`premium-rwo` StorageClass). Diese werden in den meisten Fällen für stark ausgelastete Datenbanken aufgrund ihrerniedrigen Latenz und hohen IOPS empfohlen.

Zalando-Operator in Ihrem Cluster bereitstellen

Stellen Sie den Zalando-Operator mithilfe eines Helm-Diagramms in Ihrem Kubernetes-Cluster bereit.

1. Fügen Sie das Helm-Diagramm-Repository des Zalando-Operators hinzu:

```
helm repo add postgres-operator-charts
https://opensource.zalando.com/postgres-operator/charts/postgres-
operator
```

2. Erstellen Sie einen Namespace für den Zalando-Operator und den Postgres-Cluster:

```
kubectl create ns postgres
kubectl create ns zalando
```

3. Stellen Sie den Zalando-Operator mit dem Helm-Befehlszeilentool bereit:

```
helm install postgres-operator postgres-operator-charts/postgres-
operator -n zalando \
    --set configKubernetes.enable_pod_antiaffinity=true \
    --set
configKubernetes.pod_antiaffinity_preferred_during_scheduling=true \
    --set
```

```
configKubernetes.pod_antiaffinity_topology_key="topology.kubernetes.io/zone" \
    --set configKubernetes.spilo_fsgroup="103"
```

Sie können die `podAntiAffinity`-Einstellungen nicht direkt für die benutzerdefinierte Ressource konfigurieren, die den Postgres-Cluster darstellt. Legen Sie stattdessen global die `podAntiAffinity`-Einstellungen für alle Postgres-Cluster in den Operatoreinstellungen fest.

4. Prüfen Sie den Bereitstellungsstatus des Zalando-Operators mit Helm:

```
helm ls -n zalando
```

Die Ausgabe sieht in etwa so aus:

```
NAME                NAMESPACE    REVISION    UPDATED
STATUS      CHART                       APP VERSION
postgres-operator    zalando     1           2023-10-13 16:04:13.945614
+0200 CEST    deployed    postgres-operator-1.10.1    1.10.1
```

Postgres bereitstellen

Die grundlegende Konfiguration für die Postgres-Clusterinstanz umfasst die folgenden Komponenten:

- Drei Postgres-Replikate: ein Leader und zwei Standby-Replikate.
- CPU-Ressourcenzuweisung: eine CPU-Anfrage und zwei CPU-Limits mit 4 GB Speicheranfragen und -Limits.
- Toleranzen, nodeAffinities und topologySpreadConstraints, die für jede Arbeitslast konfiguriert sind und eine ordnungsgemäße Verteilung auf Kubernetes-Knoten mithilfe ihrer jeweiligen Knotenpools und verschiedenen Verfügbarkeitszonen gewährleisten.

Diese Konfiguration stellt die minimale Einrichtung dar, die zum Erstellen eines produktionsfertigen Postgres-Clusters erforderlich ist.

Das folgende Manifest beschreibt einen Postgres-Cluster: databases/postgres-zalando/manifests/01-basic-cluster/my-cluster.yaml

```
apiVersion: "acid.zalan.do/v1"
kind: postgresql
metadata:
  name: my-cluster
spec:
  dockerImage: ghcr.io/zalando/spilo-15:3.0-p1
  teamId: "my-team"
  numberOfInstances: 3
  users:
    mydatabaseowner:
    - superuser
    - createdb
    myuser: []
```

```
  databases:
    mydatabase: mydatabaseowner
  postgresql:
    version: "15"
    parameters:
      shared_buffers: "32MB"
      max_connections: "10"
      log_statement: "all"
      password_encryption: scram-sha-256
  volume:
    size: 5Gi
    storageClass: premium-rwo
  enableShmVolume: true
  podAnnotations:
    cluster-autoscaler.kubernetes.io/safe-to-evict: "true"
  tolerations:
  - key: "app.stateful/component"
    operator: "Equal"
    value: "postgres-operator"
    effect: NoSchedule
  nodeAffinity:
    preferredDuringSchedulingIgnoredDuringExecution:
    - weight: 1
      preference:
        matchExpressions:
        - key: "app.stateful/component"
          operator: In
          values:
          - "postgres-operator"
  resources:
    requests:
      cpu: "1"
      memory: 4Gi
    limits:
      cpu: "2"
      memory: 4Gi
  sidecars:
    - name: exporter
      image: quay.io/prometheuscommunity/postgres-exporter:v0.14.0
      args:
      - --collector.stat_statements
      ports:
      - name: exporter
        containerPort: 9187
        protocol: TCP
      resources:
        limits:
          cpu: 500m
          memory: 256M
        requests:
```

```
        cpu: 100m
        memory: 256M
    env:
    - name: "DATA_SOURCE_URI"
      value: "localhost/postgres?sslmode=require"
    - name: "DATA_SOURCE_USER"
      value: "$(POSTGRES_USER)"
    - name: "DATA_SOURCE_PASS"
      value: "$(POSTGRES_PASSWORD)"
```

Dieses Manifest hat die folgenden Felder:

- `spec.teamId` ist ein Präfix für die von Ihnen ausgewählten Clusterobjekte
- `spec.numberOfInstances` ist die Gesamtzahl der Instanzen für einen Cluster
- `spec.users` ist die Nutzerliste mit Berechtigungen
- `spec.databases` ist die Datenbankliste im Format dbname: ownername
- `spec.postgresql` sind die Postgres-Parameter
- `spec.volume` sind die Parameter für den nichtflüchtigen Speicher
- `spec.tolerations` ist die Toleranz-Pod-Vorlage, mit der Cluster-Pods auf pool-postgres-Knoten geplant werden können
- `spec.nodeAffinity` ist die Pod-Vorlage nodeAffinity, die GKE mitteilt, dass Cluster-Pods lieber auf pool-postgres-Knoten geplant werden sollen.
- `spec.resources` sind Anfragen und Limits für Cluster-Pods
- `spec.sidecars` ist eine Liste der Sidecar-Container, die postgres-exporter enthält

Weitere Informationen finden Sie in der Postgres-Dokumentation unter Referenz zu Clustermanifesten.

Einfachen Postgres-Cluster erstellen

1. Erstellen Sie mithilfe der grundlegenden Konfiguration einen neuen Postgres-Cluster:

   ```
   kubectl apply -n postgres -f manifests/01-basic-cluster/my-cluster.yaml
   ```

   Mit diesem Befehl wird eine benutzerdefinierte PostgreSQL-Ressource des Zalando-Operators erstellt:
   * CPU- und Speicheranforderungen und -limits
   * Markierungen und Affinitäten zum Verteilen der bereitgestellten Pod-Replikate auf Kubernetes-Knoten.
   * Eine Datenbank
   * Zwei Nutzer mit Datenbankinhaberberechtigungen
   * Ein Nutzer ohne Berechtigungen

2. Warten Sie, bis Kubernetes die erforderlichen Arbeitslasten gestartet hat:

   ```
   kubectl wait pods -l cluster-name=my-cluster --for condition=Ready --
   timeout=300s -n postgres
   ```

   Die Verarbeitung dieses Befehls kann einige Minuten dauern.

3. Prüfen Sie, ob Kubernetes die Postgres-Arbeitslasten erstellt hat:

```
kubectl get pod,svc,statefulset,deploy,pdb,secret -n postgres
```

Die Ausgabe sieht in etwa so aus: <code bash>NAME READY STATUS RESTARTS AGE

pod/my-cluster-0 1/1 Running 0 6m41s pod/my-cluster-1 1/1 Running 0 5m56s pod/my-cluster-2 1/1 Running 0 5m16s pod/postgres-operator-db9667d4d-rgcs8 1/1 Running 0 12m

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE service/my-cluster ClusterIP 10.52.12.109 <none> 5432/TCP 6m43s service/my-cluster-config ClusterIP None <none> <none> 5m55s service/my-cluster-repl ClusterIP 10.52.6.152 <none> 5432/TCP 6m43s service/postgres-operator ClusterIP 10.52.8.176 <none> 8080/TCP 12m

NAME READY AGE statefulset.apps/my-cluster 3/3 6m43s

NAME READY UP-TO-DATE AVAILABLE AGE deployment.apps/postgres-operator 1/1 1 1 12m

NAME MIN AVAILABLE MAX UNAVAILABLE ALLOWED DISRUPTIONS AGE poddisruptionbudget.policy/postgres-my-cluster-pdb 1 N/A 0 6m44s

NAME TYPE DATA AGE secret/my-user.my-cluster.credentials.postgresql.acid.zalan.do Opaque 2 6m45s secret/postgres.my-cluster.credentials.postgresql.acid.zalan.do Opaque 2 6m44s secret/sh.helm.release.v1.postgres-operator.v1 helm.sh/release.v1 1 12m secret/standby.my-cluster.credentials.postgresql.acid.zalan.do Opaque 2 6m44s secret/zalando.my-cluster.credentials.postgresql.acid.zalan.do Opaque 2 6m44s</code

Der Operator erstellt die folgenden Ressourcen: