Если вы видите что-то необычное, просто сообщите мне.

Postgres connection pool для Kubernetes

Проблема

Если вы разрабатываете приложения используя такой фреймворк как Django или RoR, вы, скорей всего, сталкивались со следуюущей проблемой:

```
FATAL: sorry, too many clients already
```

Как вы знаете, этот тип фрейморка использует пул подключений к базе данных, с целью сократить время работы с базой данных.

Это всегда здорово когда ваша бд настроениа обслуживать множество подключений.

Как вы можете понять, это не случай с Postgres.

Каждое подключение postgres использует порядка 10mb, так же большую часть времени они находятся в ожидании.

Вместе с бумом gRPC потоков, всё стало хуже. У нас есть кучество подключений в ожидании, но Postgres запрашивает больше ресурсов ни на что, чтобы перевести подключения в состояние ожидания.

PgBouncer на помощь

Есть несколько решений для проблем с множественными подключениями но все из них используют один и тот же шаблон, прокси по середине.

Идея такова, вы подключаете потребителя к прокси, который позволяет множество дешовых подключений, и этот прокси подключает к Postgres базе даных только когда ваше приложение действительно необходимо произвести действие в дб.

Одно из этих решений - **PgBouncer**.

Это старейшее решение и оно широко используется.

Pgbouncer в вашем кластере K8S

Запустить pgbouncer в кластер проще паренной репы.

Мы будем использовать этот образ: edoburu/pgbouncer

Для начала определим configmap, указав следующие настройки для подключения к Postgres дб:

- DB_HOST: адрес Postgres дб
- DB_USER: Postgres пользоатель
- DB_PASSWORD: Postgres пользователь базы данных (его необходимо хранить в закрытом виде).
- POOL_MODE: указываем параметр в режиме "transaction", так как нужно подключаться к Postgres дб, когда нам реально это нужно.
- SERVER_RESET_QUERY: прошлые подключения могут быть использованы другими клиентами. Очень важно скдиывать предыдущие сессии. `DISCARD ALL` используется для версий Postgres 8.3<

```
pgbouncer-configmap.yaml
apiVersion: v1
```

```
kind: ConfigMap
metadata:
  name: pgbouncer-env
  namespace: test

data:
  DB_HOST:
  DB_PASSWORD:
  DB_USER:
  POOL_MODE: transaction
  SERVER_RESET_QUERY: DISCARD ALL
```

Настройки деплоймента должны выглядеть следующим образом:

```
#pgbouncer-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: pgbouncer-deployment
  namespace: test
  labels:
    app: pgbouncer-app
spec:
  selector:
    matchLabels:
      app: pgbouncer-app
  template:
    metadata:
      labels:
        app: pgbouncer-app
    spec:
      containers:
        -  image: edoburu/pgbouncer:1.9.0
           name: pgbouncer-pod
           ports:
           - containerPort: 5432
             name: pgbouncer-p
           securityContext:
             allowPrivilegeEscalation: false
             capabilities:
```

```
            drop:
            - all
          lifecycle:
            preStop:
              exec:
                command:
                - /bin/sh
                - -c
                - killall -INT pgbouncer && sleep 120
          envFrom:
          - configMapRef:
              name: pgbouncer-env
```

Мы примеяем конфи используя стандартную команду(не забываем сначала создать `configmap`, а затем деплоймент):

```
$ kubectl apply -f pgbouncer-configmap.yaml
$ kubectl apply -f pgbouncer-deployment.yaml
```

Осталось только создать сервис для протребителя.

```
#pgbouncer-service.yaml
apiVersion: v1
kind: Service
metadata:
  name: pgbouncer-service
  namespace: test
spec:
  type: ClusterIP
  selector:
    app: pgbouncer-app
  ports:
  - name: pgbouncer
    port: 5432
    targetPort: pgbouncer-p
```

Применяем конфиг сервиса:

```
$ kubectl apply -f pgbouncer-service.yaml
```

Вот и всё!

Можно пользоваться, только не забудьте изменить `DB_HOST` переменную **в вашем деплойменте** с postgres адреса на `pgbouncer-service`(в нашем случае)

Revision #3

Created 2021-09-02 15:38:46 UTC by gasick

Updated 2023-04-16 19:36:18 UTC by gasick