Run Pgpool-II on Kubernetes

This documentation describes how to run Pgpool-II to achieve read query load balancing and connection pooling on Kubernetes.

Introduction

Because PostgreSQL is a stateful application and managing PostgreSQL has very specific requirements (e.g. backup, recovery, automated failover, etc), the built-in functionality of Kubernetes can't handle these tasks. Therefore, an Operator that extends the functionality of the Kubernetes to create and manage PostgreSQL is required.

There are several PostgreSQL operators, such as Crunchy PostgreSQL Operator, Zalando PostgreSQL Operator and KubeDB. However, these operators don't provide query load balancing functionality.

This documentation describes how to combine PostgreSQL Operator with Pgpool-II to deploy a PostgreSQL cluster with query load balancing and connection pooling capability on Kubernetes. Pgpool-II can be combined with any of the PostgreSQL operators mentioned above.

Prerequisites

Before you start the configuration process, please check the following prerequisites. - Make sure you have a Kubernetes cluster, and the `kubectl` is installed. - Kebernetes 1.15 or older is required. - PostgreSQL Operator and a PostgreSQL cluster are installed. For the installation of each PostgreSQL Operator, please see the documentation below: - Crunchy PostgreSQL Operator - Zalando PostgreSQL Operator - KubeDB

Architecture

Deploy Pgpool-II

Pgpool-II's health check, automated failover, watchdog and online recovery features aren't required on Kubernetes. You need to only enable load balancing and connection pooling.

The Pgpool-II pod should work with the minimal configuration below:

```
backend_hostname0 = '<primary service name>'
backend_hostname1 = '<replica service name>'
backend_port0 = '5432'
backend_port1 = '5432'
backend_flag0 = 'ALWAYS_PRIMARY|DISALLOW_TO_FAILOVER'
backend_flag1 = 'DISALLOW_TO_FAILOVER'
backend_weight0 = '<load balance ratio>'
```

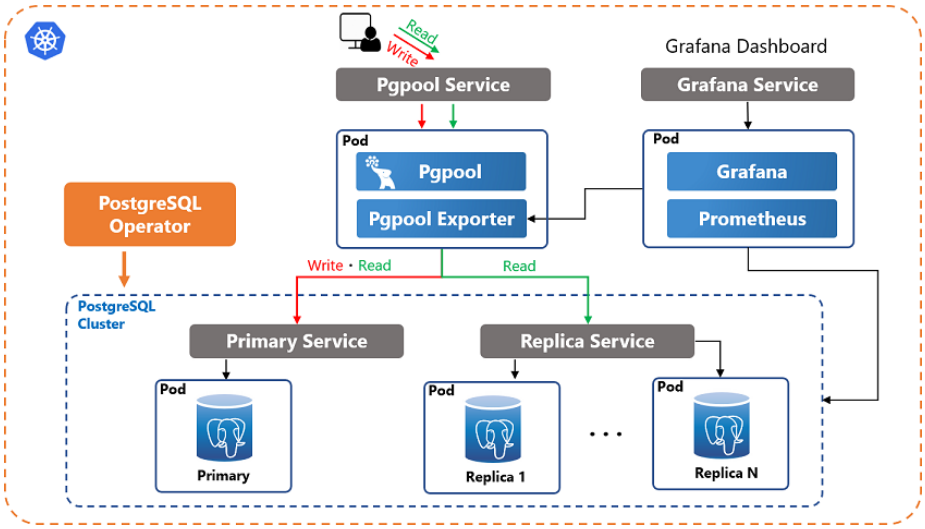

Figure 1: pgpool-on-k8s

```
backend_weight1 = '<load balance ratio>'

failover_on_backend_error = off

sr_check_period = 10                          (when using streaming replication check)
sr_check_user='username used for streaming replication check'  (when using streaming replica

load_balance_mode = on
connection_cache = on
listen_addresses = '*'
```

There are two ways to configure Pgpool-II.

- Using environment variables
- Using a ConfigMap

You may need to configure client authentication and more parameters to setup your production-ready environment. We recommend using a `ConfigMap` to configure `pgpool.conf` and `pool_hba.conf` to setup a production-ready database environment.

The following sections describe how to configure and deploy Pgpool-II pod using environment variables and ConfigMap respectively. These sections are using minimal configuration for demonstration purposes. We recommend that you read section Pgpool-II configuration to see how to properly configure Pgpool-II.

You can download the example manifests used for deploying Pgpool-II from here. Note, we provide the example manifests as an example only to simplify the

installation. All configuration options are not documented in the example manifests. you should consider updating the manifests based on your Kubernetes environment and configuration preferences. For more advanced configuration of Pgpool-II, please refer to the Pgpool-II docs.

Configure Pgpool-II using environment variables

Kubernetes environment variables can be passed to a container in a pod. You can define environment variables in the deployment manifest to configure Pgpool-II's parameters. `pgpool-deploy-minimal.yaml` is an example manifest including the minimal settings of environment variables. You can download `pgpool-deploy-minimal.yaml` and modify the environment variables in this manifest.

```
curl -LO https://raw.githubusercontent.com/pgpool/pgpool2_on_k8s/master/pgpool-deploy-minima
```

Environment variables starting with `PGPOOL_PARAMS_` can be converted to Pgpool-II's configuration parameters and these values can override the default settings.

On kubernetes, you need to specify only two backend nodes. Update `pgpool-deploy-minimal.yaml` based on your Kubernetes and PostgreSQL environment.

- `backend_hostname`: Specify the primary service name to `backend_hostname0` and the replica service name to `backend_hostname1`.
- `backend_flag`: Because failover is managed by Kubernetes, specify `DISALLOW_TO_FAILOVER` flag to `backend_flag` for both of the two nodes and `ALWAYS_PRIMARY` flag to `backend_flag0`.
- `backend_data_directory`: The setting of `backend_data_directory` is not required.

For example, the following environment variables defined in manifest,

```
env:
- name: PGPOOL_PARAMS_BACKEND_HOSTNAME0
  value: "mypostgres"
- name: PGPOOL_PARAMS_BACKEND_PORT0
  value: "5432"
- name: PGPOOL_PARAMS_BACKEND_WEIGHT0
  value: "1"
- name: PGPOOL_PARAMS_BACKEND_FLAG0
  value: "ALWAYS_PRIMARY|DISALLOW_TO_FAILOVER"
- name: PGPOOL_PARAMS_BACKEND_HOSTNAME1
  value: "mypostgres-replica"
- name: PGPOOL_PARAMS_BACKEND_PORT1
  value: "5432"
- name: PGPOOL_PARAMS_BACKEND_WEIGHT1
  value: "1"
```

```
- name: PGPOOL_PARAMS_BACKEND_FLAG1
  value: "DISALLOW_TO_FAILOVER"
```

will be convert to the following configuration parameters in `pgpool.conf`.

```
backend_hostname0 = 'mypostgres'
backend_port0 = '5432'
backend_weight0 = '1'
backend_flag0 = 'ALWAYS_PRIMARY|DISALLOW_TO_FAILOVER'
backend_hostname1 = 'mypostgres-replica'
backend_port1 = '5432'
backend_weight1 = '1'
backend_flag1 = 'DISALLOW_TO_FAILOVER'
```

Then, you need to define environment variables that contain the `username` and `password` for the PostgreSQL users for client authentication. For more details, see section Register password to pool_passwd.

After updating the manifest, run the following command to deploy Pgpool-II.

```
kubectl apply -f pgpool-deploy-minimal.yaml
```

Configure Pgpool-II using ConfigMap

Alternatively, you can use a Kubernetes `ConfigMap` to store the entire `pgpool.conf` and `pool_hba.conf`. The `ConfigMap` can be mounted to Pgpool-II's container as a volume. If `pool_hba.conf` isn't configured, Pgpool-II will generate it automatically.

You can download the example manifest files that define the `ConfigMap` and `Deployment`.

```
curl -LO https://raw.githubusercontent.com/pgpool/pgpool2_on_k8s/master/pgpool-configmap.yam
curl -LO https://raw.githubusercontent.com/pgpool/pgpool2_on_k8s/master/pgpool-deploy.yaml
```

The `ConfigMap` is in the following format. You can update it based on your configuration preferences.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: pgpool-config
  labels:
    name: pgpool-config
data:
  pgpool.conf: |-
    listen_addresses = '*'
    port = 9999
    socket_dir = '/var/run/pgpool'
    pcp_listen_addresses = '*'
```

```
    pcp_port = 9898
    pcp_socket_dir = '/var/run/pgpool'
    backend_hostname0 = 'mypostgres'
...
  #If pool_hba.conf isn't configured, Pgpool-II will generate it automatically.
  #Note that to use pool_hba.conf you must set enable_pool_hba = on.
  #pool_hba.conf: |-
  #  local      all    all           trust
  #  hostssl    all    all    all    scram-sha-256
```

Note, to use the `pool_hba.conf` for client authentication, you must turn on `enable_pool_hba`. For more details on client authentication, please refer to Pgpool-II docs.

Then, you need to define environment variables that contain the `username` and `password` for the PostgreSQL users for client authentication. For more details, see section Register password to pool_passwd.

Run the following commands to create `ConfigMap` and Pgpool-II pod that references this `ConfigMap`.

```
kubectl apply -f pgpool-configmap.yaml
kubectl apply -f pgpool-deploy.yaml
```

After deploying Pgpool-II, you can see the Pgpool-II Pod and Services using `kubectl get pod` and `kubectl get svc` command.

Pgpool-II configuration

Environment variables

Below is the list of environment variables available in the Pgpool-II container.

- `PGPOOL_ENABLE_POOL_PASSWD`: If true, generate `pool_passwd` file. Default is `true`. If false, Pgpool-II will use `password` authentication between client and Pgpool-II and force SSL on all connections in `pool_hba.conf`:

  ```
  hostssl    all    all    all    password
  ```

- `PGPOOL_PASSWORD_ENCRYPTION_METHOD`: Password encryption method used to generate `pool_passwd`. Either `scram-sha-256` or `md5`. Default is `scram-sha-256`. This parameter is valid only if `PGPOOL_ENABLE_POOL_PASSWD` is `true`.
- `PGPOOL_SKIP_PASSWORD_ENCRYPTION`: If true, skip password encryption. Default is `false`. It is used for the passwords that are already encrypted.
- `<some string>_USERNAME`: The username for PostgreSQL users. No defaults. (See Register password to pool_passwd)
- `<some string>_PASSWORD`: The password for PostgreSQL users. No defaults. (See Register password to pool_passwd)

- PGPOOL_PCP_USER: The username to use for PCP command. No defaults. (See Generating pcp.conf)
- PGPOOL_PCP_PASSWORD: The password to use for PCP command. No defaults. (See Generating pcp.conf)
- PGPOOL_PARAMS_<Pgpool-II configuration parameters>: Configure the Pgpool-II parameters. Environment variables starting with PGPOOL_PARAMS_ can be converted to Pgpool-II's configuration parameters and these values can override the default settings. (See Configure Pgpool-II using environment variables)

Backend settings

On kubernetes, you need to specify only two backend nodes. Specify the primary service name to backend_hostname0, replica service name to backend_hostname1.

```
backend_hostname0 = '<primary service name>'
backend_hostname1 = '<replica service name>'
backend_port0 = '5432'
backend_port1 = '5432'
```

Automated failover

Pgpool-II has the ability to periodically connect to the configured PostgreSQL backends and check the state of PostgreSQL. If an error is detected, Pgpool-II will trigger the failover. In Kubernetes, Kubernetes monitors the PostgreSQL pods, if a pod goes down, Kubernetes will restart a new one. You need to disable Pgpool-II's automated failover, becuase Pgpool-II's automated failover is not required in Kubernetes.

Specify PostgreSQL node 0 as primary (ALWAYS_PRIMARY), because the primary Service always connects to the primary pod and the replica Service always connects to the standby pods, even if the primary or replica pod is sacled, restarted or failover occurred.

```
backend_flag0 ='ALWAYS_PRIMARY|DISALLOW_TO_FAILOVER'
backend_flag1 ='DISALLOW_TO_FAILOVER'
failover_on_backend_error = off
```

Load balancing

To enable load balancing, you must set load_balance_mode = on. In Kubernetes environment, even if there are multiple replicas, Pgpool-II connects to them via a single replica service. If you are using two or more replicas, set backend_weight1 using the number of replicas so that the READ queries can be evenly distributed among all PostgreSQL pods.

For example, if you have two replicas, set backend_weight1 to 2.

```
backend_weight0 = 1
backend_weight1 = 2
```

Register password to pool_passwd

Pgpool-II performs authentication using `pool_passwd` file which contains the `username:encrypted password` for PostgreSQL users.

At Pgpool-II pod startup, Pgpool-II automatically generates pool_passwd based on the environment variables in `<some string>_USERNAME` and `<some string>_PASSWORD` format and encrypt the password using the encryption method set in `PGPOOL_PASSWORD_ENCRYPTION_METHOD`.

The environment variables that represent the username and password for PostgreSQL users must be defined in the following format:

```
username: <some string>_USERNAME
password: <some string>_PASSWORD
```

Define the environment variables using secret is the recommended way to keep user credentials secure. In most PostgreSQL Operators, several secrets which define the PostgreSQL user's redentials will be automaticlly created when creating a PostgreSQL cluster. Use `kubectl get secret` command to check the existing Secrets.

For example, if `mypostgres-postgres-secret` is created to store the username and password for `postgres` user, to reference this secret, you can define the environment variables as below. You may need to modify `key` to match the settings of your secrets.

```
env:
- name: POSTGRES_USERNAME
  valueFrom:
     secretKeyRef:
       name: mypostgres-postgres-secret
       key: username
- name: POSTGRES_PASSWORD
  valueFrom:
     secretKeyRef:
       name: mypostgres-postgres-secret
       key: password
```

When Pgpool-II Pod is started, `pool_passwd` is automatically generated under `/opt/pgpool-II/etc`.

```
$ kubectl exec <pgpool pod> -it -- cat /opt/pgpool-II/etc/pool_passwd
postgres:AESHs/pWL5rtXy2IwuzroHfqg==
```

TLS settings

If you wish to enable the SSL connections, turn on `ssl`.

```
ssl = on
```

You can use your own TLS certificate. If your own TLS certificate isn't specified, Pgpool-II will automatically generate the private key and certificate under `/opt/pgpool-II/tls/`. Pgpool-II's configuration parameters `ssl_key` and `ssl_cert` will be automatically configured with the path of private key file and certificate file.

To use your own TLS certificates, you will need to create a secret and mount it to Pgpool-II pod. For example, you can run the following command to generate TLS secret.

```
kubectl create secret tls pgpool-tls --cert=tls.crt --key=tls.key
```

You will need to mount the secret as a volume.

```
    spec:
    ...
        volumeMounts:
        - name: pgpool-tls
          mountPath: /config/tls
      volumes:
      - name: pgpool-tls
        secret:
          secretName: pgpool-tls
```

Generating pcp.conf

If you wish to use Pgpool-II's PCP command, you will need to set the `PGPOOL_PCP_USER` and `PGPOOL_PCP_PASSWORD` environment variables. To enable the following settings, you will need to define a secret that stores the username and password for the PCP user.

```
env:
- name: PGPOOL_PCP_USER
  valueFrom:
     secretKeyRef:
       name: pgpool-pcp-secret
       key: username
- name: PGPOOL_PCP_PASSWORD
  valueFrom:
     secretKeyRef:
       name: pgpool-pcp-secret
       key: password
```

When Pgpool-II Pod is started, `pcp.conf` will be automatically generated under `/opt/pgpool-II/etc/`.

```
$ kubectl exec <pgpool pod> -it -- cat /opt/pgpool-II/etc/pcp.conf
<pcpuser>:<md5 encrypted password>
```

Streaming replication check

Pgpool-II has the ability to periodically connect to the configured PostgreSQL backends and check the replication delay. To use this feature, sr_check_user and sr_check_password are required. If `sr_check_password` isn't set, Pgpool-II will try to get the password from `pool_passwd`.

Below is an example that connects to PostgreSQL using `postgres` user every 10s to perform streaming replication check. Because `sr_check_password` isn't set here, Pgpool-II will try to get the `postgres` user's password from `pool_passwd`.

```
sr_check_period = 10
sr_check_user = 'postgres'
```

Create a secret to store the `username` and `password` for `sr_check_user` and configure the environment variables to reference the created secret. In most PostgreSQL Operators, several secrets which define the PostgreSQL user's redentials will be automaticlly created when creating a PostgreSQL cluster. Use `kubectl get secret` command to check the existing secrets.

For example, the environment variables below reference the Secret `mypostgres-postgres-secret`.

```
env:
- name: POSTGRES_USERNAME
  valueFrom:
     secretKeyRef:
       name: mypostgres-postgres-secret
       key: username
- name: POSTGRES_PASSWORD
  valueFrom:
     secretKeyRef:
       name: mypostgres-postgres-secret
       key: password
```

Note, in Kubernetes Pgpool-II connects to any of the replicas rather than connecting to all the replicas. Even if there are multiple replicas, Pgpool-II manages them as a single replica. Therefore, Pgpool-II may not be able to properly determine the replication delay.

To disable this feature, configure the following parameter:

```
sr_check_period = 0
```

Generating pool_hba.conf

ConfigMap allows you to customize the settings of `pool_hba.conf`. If the settings of `pool_hba.conf` isn't included in ConfigMap, Pgpool-II will automatically generate it if `enable_pool_hba = on`.

Pgpool-II will generate the following entries in `/opt/pgpool-II/etc/pool_hba.conf` and the authentication method is the value set in `PGPOOL_PASSWORD_ENCRYPTION_METHOD`.

```
# If ssl = on
local      all    all           trust
hostssl    all    all    all    <PGPOOL_PASSWORD_ENCRYPTION_METHOD>

# If ssl = off
local      all    all           trust
host       all    all    all    <PGPOOL_PASSWORD_ENCRYPTION_METHOD>
```

Pgpool-II with monitoring

Pgpool-II Exporter is a Prometheus exporter for Pgpool-II metrics.

The example manifest `pgpool-deploy-metrics.yaml` is used to deploy Pgpool-II pod with the Pgpool-II Exporter container.

```
spec:
  containers:
  - name: pgpool
    image: pgpool/pgpool
  ...
  - name: pgpool-stats
    image: pgpool/pgpool2_exporter
  ...
```

Download the sample manifest `pgpool-deploy-metrics.yaml`.

```
curl -LO https://raw.githubusercontent.com/pgpool/pgpool2_on_k8s/master/pgpool-deploy-metric
```

Then, configure Pgpool-II and Pgpool-II Exporter. For more details on configuring Pgpool-II, see the previous section Deploy Pgpool-II. Below is the settings of the environment variables used in Pgpool-II exporter container to connect to Pgpool-II.

```
env:
- name: POSTGRES_USERNAME
  valueFrom:
    secretKeyRef:
      name: mypostgres-postgres-secret
      key: username
- name: POSTGRES_PASSWORD
  valueFrom:
```

```
    secretKeyRef:
      name: mypostgres-postgres-secret
      key: password
- name: PGPOOL_DATABASE
  value: "postgres"
- name: PGPOOL_SERVICE
  value: "localhost"
- name: PGPOOL_SERVICE_PORT
  value: "9999"
- name: SSLMODE
  value: "require"
```

After configuring Pgpool-II and Pgpool-II Exporter, deploy Pgpool-II.

```
kubectl apply -f pgpool-configmap.yaml
kubectl apply -f pgpool-deploy-metrics.yaml
```