NJIT

High-Availability Postgres in Kubernetes

Ryan Tolboom

New Jersey Institute of Technology

Goals

Arrow in a target is in the public domain

- implement a primary / standby replication setup for PostgreSQL
- two *Services* will be provided: read/write and read
- Kubernetes will handle the initialization and monitoring
- we will be using YAML files to pass objects to `kubectl apply`

PersistentVolumeClaim

- lets the cluster know that you are expecting certain storage resources
- we are looking for a place to store our primary database files
- claims are fulfilled by a StorageClass that is built-in (details)

PersistentVolumeClaim Example

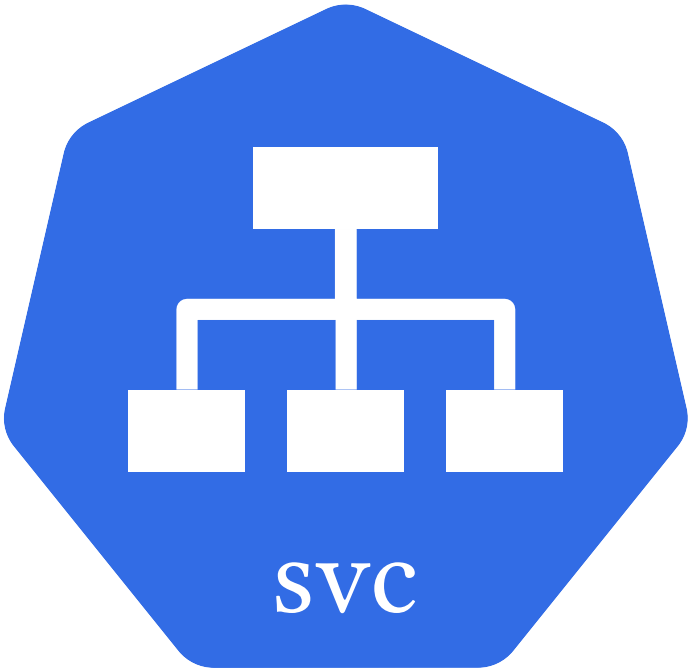

```yaml
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: db-primary-pv-claim
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 512M
---
```

Supported accessModes

This claim will be used by our one db-rw pod, so we don't have to worry about shared access. The supported accessModes are:

- ReadWriteOnce - can be mounted read / write by only one pod
- ReadOnlyMany - can be mounted read-only by many pods
- ReadWriteMany - can be mounted as read-write by many pods

Service



- exposes an application on a group of pods
- two *Services*
  - db-rw Service which connects to our primary PostgreSQL instance
  - db-r Service which connects to our standby PostgreSQL instances
- Unlike Docker Compose, even on our internal network we have to explicitly state which ports we make available

Service Example

Read / Write

```yaml
---
apiVersion: v1
kind: Service
metadata:
  name: db-rw
  labels:
    app: db-rw
  spec:
    selector:
      app: db-rw
    ports:
      - protocol: TCP
        port: 5432
```

Read Only

```yaml
---
apiVersion: v1
kind: Service
metadata:
  name: db-r
  labels:
    app: db-r
spec:
  selector:
    app: db-r
  ports:
    - protocol: TCP
      port: 5432
```

Selectors

- the selector field above defines how a *Service* knows which pods to utilize
- all pods with the app label db-r are used by the db-r
- a similar rule is applied to the db-rw *Service*
- both services accept incoming connections on port 5432 and route those connections to 5432
- if there are multiple pods in a *Service* a load balancing scheme is used by default

Service Discovery

- Kubernetes *Services* make things easier
- You want to connect to a read-only database instance? Use the hostname db-r
- You want to connect to a read-write database instance? use the hostname db-rw
- DNS resolution, load-balancing, and routing are set up automatically

Radar screen vector image is in the public domain

Deployment

- tells Kubernetes how to create and monitor pods
- the bulk of our work will be done in the db-r and db-rw *Deployment*
- let's take a look at them on GitHub