Postgres replication setup on k8s local cluster

Table of Contents

Objectives

Setup Postgresql replication on kubernetes pod for data syncronization over all replicas.

Overview

Postgresql replication feature provided for data synchronization over all replicas servers. We can achieve this using postgresql build-in replication feature so that whenever/whatever changes made over primary server will reflect over all replicas. There are basically two types of replication synchronous and asynchronous. Below document guide us to perform this on k8s environment.

Pre requisites/System Requirements

1) Kubernetes and docker installed and master/slave cluster setup.

Configuration Files:

Storageclass:

Storage class needed to allocate disk space to persistent volume, which we will assign to pod/container to store data on local. We can create storage class of basically two types of VolumeBindingMode: WaitForFirstConsumer and Immidiate. This define wheather our volume bind should be immediate or assign one by one. Here as we have single master and slave configuration we will use immediate.

Config storageclass.yaml

```yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: local-path
provisioner: kubernetes.io/no-provisioner
volumeBindingMode: Immediate
```

Persistent Volume:

Persistent volume is needed and assigned to pod/container to keep data on local storage. This basically used for databases container. Also, for this setup we are need to create two statefulsets.yaml file one for primary and one for all replicas. Also, we need to create number of persistent volume based on number of replicas we needed because statefulsets uses saperate PVs. In our scenario we are creating one primary and one replica, that's why we need atleast two PVs.

Config postgres-pv.yaml

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: postgres-primary-pv-0
spec:
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  volumeMode: Filesystem
  persistentVolumeReclaimPolicy: Retain
  storageClassName: local-path
  hostPath:
    path: /mnt/data/postgres-primary-0
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - key: kubernetes.io/hostname
          operator: In
          values:
          - svm-cmtit-vm5
---
apiVersion: v1
kind: PersistentVolume
metadata:
  name: postgres-replica-pv-0
spec:
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  volumeMode: Filesystem
  persistentVolumeReclaimPolicy: Retain
  storageClassName: local-path
  hostPath:
    path: /mnt/data/postgres-replica-0
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - key: kubernetes.io/hostname
          operator: In
          values:
          - svm-cmtit-vm5
```

FYI: In above config we need to update nodeSelectorTerms values field according to our slave node name.

Persistent Volume Claim:

Same as two PVs we need to create two PVCs, one for primary and one for replica statefulset. PVCs claim PVs mapped to localstorage to our statefulsets.

Config postgres-pvc.yaml

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: postgres-primary-data-postgres-primary-0
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
  storageClassName: local-path
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: postgres-replica-data-postgres-replica-0
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
  storageClassName: local-path
```

Postgres ConfigMap:

We need to create this configmap file to update postgresql configuration for both accordingly as we need to open connection from primary to replica for data synchronization.

Config postgres-configmap.yaml

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: postgres-config
data:
  primary.conf: |
    listen_addresses = '*'
    archive_mode = 'ON'
    wal_keep_segments = '10'
    max_connections = 100
    wal_level = 'replica'
    max_wal_senders = 10
    max_replication_slots = 10
    hot_standby = on
  replica.conf: |
    primary_conninfo = 'host=postgres-primary port=5432 user=replicator password=replicator application_name=postgres_replica'
    hot_standby = on
```

Postgres Secret:

We need to create postgres secret file so that we can securely pass our database credentials in encrypted form to our setup. (Postgres cred provided are base64 encoded)

Config postgres-secret.yaml

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: postgres-secret
type: Opaque
data:
  POSTGRES_USER: cG9zdGdyZXM=    # base64 encoded 'your_user'
  POSTGRES_PASSWORD: YWRtaW4xMjM=    # base64 encoded 'your_password'
  POSTGRES_DB: c2FtcGxlZGI=    # base64 encoded 'your_database'
```

Primary service:

As we are creating two statefulset for both primary and replica, We need to create service file as well for service discovery.

Config service.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: postgres-primary
spec:
  selector:
    app: postgres
    role: primary
  ports:
  - port: 5432
    targetPort: 5432
---
apiVersion: v1
kind: Service
metadata:
  name: postgres-replica
spec:
  selector:
    app: postgres
    role: replica
  ports:
  - port: 5432
    targetPort: 5432
```

Primary Statefulset:

We need to create statefulset file defining volume mount, appname for service discovery, config file, storageclass name, secret filename, number of replicas and image.

Config primary-statefulset.yaml

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: postgres-primary
spec:
  serviceName: postgres-primary
  replicas: 1
  selector:
    matchLabels:
      app: postgres
      role: primary
  template:
    metadata:
      labels:
        app: postgres
        role: primary
    spec:
      containers:
      - name: postgres
        image: postgres:latest
        envFrom:
        - secretRef:
            name: postgres-secret
        ports:
        - containerPort: 5432
        volumeMounts:
        - name: postgres-primary-data
          mountPath: /var/lib/postgresql/data
        - name: postgres-config
          mountPath: /etc/postgresql/postgresql.conf
          subPath: primary.conf
      volumes:
      - name: postgres-config
        configMap:
          name: postgres-config
  volumeClaimTemplates:
  - metadata:
      name: postgres-primary-data
    spec:
      accessModes: [ "ReadWriteOnce" ]
      storageClassName: local-path
      resources:
        requests:
          storage: 10Gi
```

Replica Statefulset:

For replica statefulset we need to additionally define pg_basebackup cmd on runtime.

Config replica-statefulset.yaml

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: postgres-replica
spec:
  serviceName: postgres-replica
  replicas: 1
  selector:
    matchLabels:
      app: postgres
      role: replica
  template:
    metadata:
      labels:
        app: postgres
        role: replica
    spec:
      containers:
      - name: postgres
        image: postgres:latest
        envFrom:
        - secretRef:
            name: postgres-secret
        command:
            - bash
            - "-c"
            - |
              if [ ! -s /var/lib/postgresql/data/PG_VERSION ]; then
                echo "Initializing replica from primary..."
                pg_basebackup -h postgres-primary -U replicator -p 5432 -D /var/lib/postgresql/data/ -Fp -Xs -P -R
              fi
              exec docker-entrypoint.sh postgres
        env:
        - name: REPLICA_ROLE
          value: "true"
        ports:
        - containerPort: 5432
        volumeMounts:
        - name: postgres-replica-data
          mountPath: /var/lib/postgresql/data
        - name: postgres-config
          mountPath: /etc/postgresql/postgresql.conf
          subPath: replica.conf
      volumes:
      - name: postgres-config
        configMap:
          name: postgres-config
  volumeClaimTemplates:
  - metadata:
      name: postgres-replica-data
    spec:
      accessModes: [ "ReadWriteOnce" ]
      storageClassName: local-path
      resources:
        requests:
          storage: 10Gi
```

Configuration steps:

First of all create folder and provide appropriate access which we'll map to our persistent volume for out setup using below cmds:

```
sudo mkdir -p /data/postgres-primary-0
sudo mkdir -p /data/postgres-replica-0
sudo chmod 777 /data/postgres-primary-0
sudo chmod 777 /data/postgres-replica-0
```

Secondly apply storageclass, PVs and PVCs verify these are created successfully and mounted successfully to respective using below cmds:

```
kubectl apply –f storageclass.yaml
kubectl apply –f postgres-pv.yaml
kubectl apply –f postgres-pvc.yaml
```

After successful creation of above, apply postgres-configmap.yaml and postgres-secret.yaml.

Once above steps successfully completed, proceed with creation of primary statefulset by applying primary-statefulset.yaml file.

As primary container is up, login to primary pod and verify postgres.conf as mentioned in postgresql-configmap.yaml file. Also update pg_hba.conf file to update connection for replica pod by adding below content to EOF.

"host replication replicator all md5"

Also, create replica user in primary postgresql server by logging in to postgres server and executing below query:

```
CREATE USER replicator WITH REPLICATION ENCRYPTED PASSWORD 'replicator';
```

Also, execute below sql query to update/enable server values for replication:

```
ALTER SYSTEM SET wal_level TO 'replica';
ALTER SYSTEM SET archive_mode TO 'ON';
ALTER SYSTEM SET max_wal_senders TO '10';
ALTER SYSTEM SET wal_keep_size TO '10';
ALTER SYSTEM SET listen_addresses TO '*';
ALTER SYSTEM SET hot_standby TO 'ON';
ALTER SYSTEM SET archive_command TO 'test ! -f /mnt/server/archivedir/%f && cp %p
/mnt/server/archivedir/%f';
```

Now, restart primary pod by deleting pod (as we have defined replicas it will automatically create new pod for same inshort restart our primary server).

Once successfully restart primary pod apply replica statefulset and login to replica pod once started successfully.

After login go to postgres default directory and copy existing data directory or main file to same folder and remove content of the directory. Now, you need to executed "pg_basebackup" cmd which will definitely going to fail because it'll interepted as it will restart postgres and since we are using postgres image it will stop pod and recreate

using replica-statefulset.yaml file with automatically execution of pg_basebackup cmd as mentioned at entrypoint in replica-statefulset.yaml file

Once replica pod restart successfully login to replica pod and you will see standby.signal file this ensure successful execution of pg_basebackup cmd (if not repeat previous step).

Verification of replication:

Verify the replication from primary using below sql query, it'll result replica server details on successful replication:
SELECT * FROM pg_stat_replication;
Verify the replication from replica using below sql query, it'll result true on successful replication:
SELECT pg_is_in_recovery();
Also, Verify by creating database on primary and checking on replica.

Useful Commands

kubectl exec –it <pod name> -- bash

kubectl get all

Troubleshooting

NA

Useful Links

https://www.percona.com/blog/setting-up-streaming-replication-postgresql/

Important Note: This setup will create replica servers read-only which means we can't make any changes from replica server.