# POSTGRESQL & K8S: RUN A STATEFUL LEGACY APP ON A STATELESS MICROSERVICE

In this blog post, we are going to expand on a previous article about statefulsets. I'll show how to run and work with a database application, such as PostgreSQL, in Kubernetes.

To follow along I assume you have a Kubernetes cluster running and are familiar with k8s Service, Statefulset, Configmap, PersistentVolume, PersistentVolumeClaim and Docker images. (If you don't, explore the K8s Guide, on the right). For us to deploy PostgresSQL on kubernetes, we need few things:

- Postgres Docker Image to deploy.
- Configmap for storing Postgres configurations.
- Postgres Statefulset to deploy the pods and to auto create the PV/PVC.
- Postgres Service to expose the statefulset.

## Setup

The first resource we need to create is the configurations we want to inject into postgres pod with a configmap. We want to pass in the username, password and the database.

*postgres-config.yaml*

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: postgres-config-demo
```

```
  labels:
    app: postgres
data:
  POSTGRES_DB: demopostgresdb
  POSTGRES_USER: demopostgresadmin
  POSTGRES_PASSWORD: demopostgrespwd
```

To create, simply run *"kubectl create-f postgres-config.yaml"*

Next resource to create is the postgres service so we can have multiple backends with a service that other services can connect with. Resource for deployment looks like:

*postgres-service.yaml*

```
apiVersion: v1
kind: Service
metadata:
  name: postgres
  labels:
    app: postgres
spec:
  ports:
  - port: 5432
    name: postgres
  clusterIP: None
  selector:
    app: postgres
```

Next resource to create is postgres statefulset. Resource for statefulset looks like:

*postgres-stateful.yaml*

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: postgres-demo
spec:
  serviceName: "postgres"
  replicas: 2
  selector:
    matchLabels:
      app: postgres
  template:
    metadata:
      labels:
        app: postgres
    spec:
      containers:
      - name: postgres
        image: postgres:latest
```

```
        envFrom:
          - configMapRef:
              name: postgres-config-demo
        ports:
        - containerPort: 5432
          name: postgredb
        volumeMounts:
        - name: postgredb
          mountPath: /var/lib/postgresql/data
          subPath: postgres
  volumeClaimTemplates:
  - metadata:
      name: postgredb
    spec:
      accessModes:
      storageClassName: gp2
      resources:
        requests:
          storage: 3Gi
```

In this yaml file, we can see that we are consuming the configmap we created earlier. We are also at the bottom of the file creating a volume claim automatically with the help of the storage class gp2. Refer to https://kubernetes.io/docs/concepts/storage/storage-classes/#aws-ebs for more info.

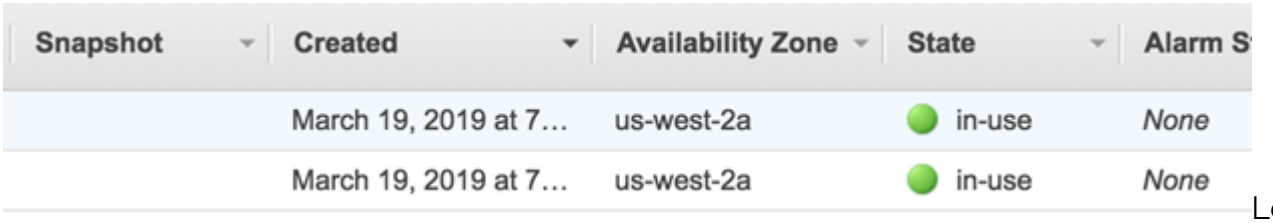
The neat thing about statefulset is that it will create a volume for each of the pods. If any of the pods get deleted, the volume will persist. Let's see what I mean:

Assuming the file above is created, if we describe any of the two pods, we will see in the event what is happening:

```
Events:
  Type    Reason                 Age   From
Message
  ----    ------                 ----  ----
-------
  Normal  Scheduled              1m    default-scheduler
Successfully assigned default/postgres-demo-0 to ip-172-20-33-219.us-
west-2.compute.internal
  Normal  SuccessfulAttachVolume  1m    attachdetach-controller
AttachVolume.Attach succeeded for volume
"pvc-61f5163b-4aac-11e9-8d84-0692704f033a"
  Normal  Pulling                1m    kubelet, ip-172-20-33-219.us-
west-2.compute.internal  pulling image "postgres:latest"
  Normal  Pulled                 1m    kubelet, ip-172-20-33-219.us-
west-2.compute.internal  Successfully pulled image "postgres:latest"
  Normal  Created                1m    kubelet, ip-172-20-33-219.us-
west-2.compute.internal  Created container
  Normal  Started                1m    kubelet, ip-172-20-33-219.us-
west-2.compute.internal  Started container
```

```
olatoyei01-mac
```

This line *"AttachVolume.Attach succeeded for volume "pvc-61f5163b-4aac-11e9-8d84-0692704f033a""* tells us that a PVC is attached and a volume is created in aws. Something like the image below:



| Snapshot | Created | Availability Zone | State | Alarm S |
|---|---|---|---|---|
| | March 19, 2019 at 7… | us-west-2a | in-use | *None* |
| | March 19, 2019 at 7… | us-west-2a | in-use | *None* |

Lets verify that the configmap actually got injected by first checking the service that fronts out statefulset.

```
kubectl get service
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
postgres     NodePort    100.64.200.90           5432:31556/TCP   13s
```

Now we can connect with node port 31556 and the public IP address of the node our pod is running on. Keep in mind, i am using node port just to show how to quickly connect. It is not recommended to expose your database to public. If we run this command *"psql -h <PUBLIC IP> -U demopostgresadmin --password -p 31556 demopostgresdb"*, it will prompt for a password. We can enter the password we defined in configmap earlier. If all goes well we should see something like

```
psql (11.2, server 10.4 (Debian 10.4-2.pgdg90+1))
Type "help" for help.

demopostgresdb=#
```

To conclude, we used statefulset to deploy our postgres image along with PV/PVC, injecting configs into the pods using configmap, then exposing the postgres pods using a service that we can connected to on the NodePort.

# Additional resources

For more on Kubernetes, explore these resources:

- Kubernetes Guide, with 20+ articles and tutorials
- BMC DevOps Blog
- Bring Kubernetes to the Serverless Party
- How eBay is Reinventing Their IT with Kubernetes & Replatforming Program