8: Storage & Stateful Applications¶

8.1: Lab Goals¶

In this lab, you will be converting your MySQL deployment into a StatefulSet. You will statically provision a Persistent Volume that the StatefulSet will use. You will then demonstrate that the state of your MySQL database successfully persists after deleting and recreating the pod.

Let's get your lab environment setup by automatically performing necessary steps from previous labs in this course. We've automated it for you with the following command:

```
eval "$BOOTSTRAP_COMMAND"
```

8.2: Convert MySQL deployment to a StatefulSet¶

Step 1: Remove MySQL Deployment¶

Before converting the gowebapp-mysql deployment to a StatefulSet, you need to remove the existing deployment from the Kubernetes cluster.

```
kubectl delete deployment gowebapp-mysql
```

Step 2: Convert MySQL from Deployment to StatefulSet¶

```
cd $HOME/gowebapp/gowebapp-mysql
```

Convert the MySQL deployment to a StatefulSet. You can use the following as a starting point.

Note

Replace TODO comments with the appropriate commands

gowebapp-mysql-sts.yaml

```yaml
apiVersion: apps/v1
kind: # TODO: Set kind to StatefulSet
metadata:
  name: gowebapp-mysql
  labels:
    app: gowebapp-mysql
    tier: backend
spec:
  serviceName: # TODO: Set serviceName to gowebapp-mysql
  replicas: 1
  selector:
    matchLabels:
      app: gowebapp-mysql
      tier: backend
  template:
    metadata:
      labels:
        app: gowebapp-mysql
        tier: backend
    spec:
      securityContext:
        fsGroup: 1000
      containers:
        - name: gowebapp-mysql
          env:
            - name: MYSQL_ROOT_PASSWORD
              value: mypassword
          image: gowebapp-mysql:v1
          ports:
            - containerPort: 3306
          livenessProbe:
            tcpSocket:
              port: 3306
            initialDelaySeconds: 20
            periodSeconds: 5
            timeoutSeconds: 1
          readinessProbe:
            exec:
              command:
                [
                  "mysql",
                  "-uroot",
                  "-pmypassword",
                  "-e",
                  "use gowebapp; select count(*) from user",
                ]
            initialDelaySeconds: 25
            periodSeconds: 10
            timeoutSeconds: 1
          volumeMounts:
            - name: mysql-pv
              mountPath: # TODO: Set mountPath to
/var/lib/mysql
  volumeClaimTemplates:
```

```
      - metadata:
          name: mysql-pv
        spec:
          accessModes: ["ReadWriteOnce"]
          resources:
            requests:
              storage: # TODO: Set storage request to 5Gi
```

We are using a custom image that we created in a previous lab. Therefore we need to add the registry server to the `image:` line in the YAML so that Kubernetes knows which registry to pull the image from. Otherwise it will try to find the image on the public/default configured registry server.

```
sed -i s/"image: gowebapp"/"image: $REGISTRY_HOST\/gowebapp"/g gowebapp-mysql-sts.yaml
```

Step 3: Create the MySQL StatefulSet¶

Warning

Ensure that you deleted the gowebapp-mysql-deployment (Step 1) before applying the new StatefulSet.

```
kubectl apply -f gowebapp-mysql-sts.yaml
```

Step 4: Check that a persistent volume claim was created and bound¶

The persistent volume claim should bind to a pre-existing persistent volume in your workshop environment.

```
kubectl get pvc
```

8.3: Test your application¶

Step 1: Access your application¶

You should be able to access the application by running the following command in the terminal and then clicking the URL it produces.

```
echo "http://$SESSION_NAME-gowebapp-k8s.$INGRESS_DOMAIN"
```

Sign up for an account, login and use the Notepad.

Warning

Note: Your browser may cache an old instance of the gowebapp application from previous labs. When the webpage loads, look at the top right. If you see 'Logout', click it. You can then proceed with creating a new test account.

Step 2: Create a file on the persistent volume¶

```
kubectl exec -it  gowebapp-mysql-0 -- touch /var/lib/mysql/Persistence
```

Step 3: Verify the file on the persistent volume¶

```
kubectl exec -it  gowebapp-mysql-0 -- ls /var/lib/mysql/Persistence
```

8.4: Illustrate Failure and Recovery of Database State¶

Step 1: Monitor StatefulSet¶

Set up a watch on the stateful set to monitor it:

```
#open a new tab/terminal connected to your lab instance
kubectl get sts gowebapp-mysql -w
```

Step 2: Find the name of the MySQL pod¶

```
#Switch to the Original Terminal/Tab
kubectl get pod -l 'app=gowebapp-mysql'
```

Step 3: Kill the pod¶

The StatefulSet will automatically relaunch it

```
kubectl delete pod <mysql_pod_name>
```

Step 4: Monitor StatefulSet¶

- Switch to the watch terminal and view the results.
- Monitor the StatefulSet until it's ready again (AVAILABLE turns from 0 to 1).
- Press <ctrl> + c to return to the prompt and close this terminal/tab

8.5: Confirm Persistent Volume Kept Application State¶

Step 1: Find the name of the new MySQL pod and inspect it¶

```
kubectl get pod -l 'app=gowebapp-mysql'

kubectl describe pod <mysql_pod_name>
```

You should see that the volume has been re-mounted to the new pod.

Step 2: Ensure gowebapp can still access the pre-failure data¶

You should be able to access the application by running the following command in the terminal and then clicking the URL it produces.

```
echo "http://$SESSION_NAME-gowebapp-k8s.$INGRESS_DOMAIN"
```

Test that you can still login using the username and password you created earlier in this lab.

Step 3: Check the persistent volume location for the file you created called Pesistence¶

```
kubectl exec -it  gowebapp-mysql-0 -- ls /var/lib/mysql
```