Lab 02: Deploy Applications Using Kubernetes

Getting Started with kubectl

`kubectl` is the command line interface for interacting with a Kubernetes cluster. Let's explore some of it's features.

Step 1: introduction to kubectl

By executing kubectl you will get a list of options you can utilize kubectl for. kubectl allows you to control Kubernetes cluster manager.

```
kubectl
```

Step 2: use kubectl to understand an object

Use explain to get documentation of various resources. For instance pods, nodes, services, etc.

```
kubectl explain pods
```

Step 3: get more information on an object

Shortcut to object names:

```
kubectl describe
```

Step 4: autocomplete

Use TAB to autocomplete:

```
kubectl describe <TAB> <TAB>
```

Create Service Object for MySQL

Step 1: Analyze a Kubernetes Service object for the backend MySQL database

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

Let's inpsect the `gowebapp-mysql-service.yaml` file. You can do so be opening it in your favorite command line editor, use the built-in editor, or we have embedded the file directly in the lab instructions here for easy viewing. This kubernetes configuration file contains the definition of what the desired state should be for our Kubernetes Service object for the backend MySQL database should be.

gowebapp-mysql-service.yaml

```
1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: gowebapp-mysql
5    labels:
6      run: gowebapp-mysql
7      tier: backend
8  spec:
9    type: ClusterIP
10   ports:
11   - port: 3306
12     targetPort: 3306
13   selector:
14     run: gowebapp-mysql
15     tier: backend
```

Step 2: create a Service defined above

Use kubectl to create the service defined above

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

Step 3: test to make sure the Service was created

```
kubectl get service -l "run=gowebapp-mysql"
```

Create Deployment Object for MySQL

Step 1: Analyze a Kubernetes Deployment object for the backend MySQL database

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

Let's inpsect the `gowebapp-mysql-deployment.yaml` file. You can do so be opening it in your favorite command line editor, use the built-in editor, or we have embedded the file directly in the lab instructions here for easy viewing. This kubernetes configuration file contains the definition of what the desired state should be for our Kubernetes Deployment object for the backend MySQL database should be.

gowebapp-mysql-deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: gowebapp-mysql
  labels:
    run: gowebapp-mysql
    tier: backend
spec:
  replicas: 1
  strategy:
    type: Recreate
  selector:
    matchLabels:
      run: gowebapp-mysql
      tier: backend
  template:
    metadata:
      labels:
        run: gowebapp-mysql
        tier: backend
    spec:
      containers:
      - name: gowebapp-mysql
        env:
        - name: MYSQL_ROOT_PASSWORD
          value: mypassword
        image: localhost:5000/gowebapp-mysql:v1
        ports:
        - containerPort: 3306
```

Step 2: create the Deployment defined above

Use kubectl to create the service defined above

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

Step 3: test to make sure the Deployment was created

```
kubectl get deployment -l "run=gowebapp-mysql"
```

Create Service Object for frontend application: gowebapp

Step 1: Analyze a Kubernetes Service object for the frontend gowebapp

```
cd $HOME/gowebapp/gowebapp
```

Let's inpsect the `gowebapp-service.yaml` file. You can do so be opening it in your favorite command line editor, use the built-in editor, or we have embedded the file directly in the lab instructions here for easy viewing. This kubernetes configuration file contains the definition of what the desired state should be for our Kubernetes Service object for the frontend gowebapp.

gowebapp-service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: gowebapp
  labels:
    run: gowebapp
    tier: frontend
spec:
  type: NodePort
  ports:
  - port: 80
  selector:
    run: gowebapp
    tier: frontend
```

Step 2: create a Service defined above

Use kubectl to create the service defined above

```
kubectl apply -f gowebapp-service.yaml
```

Step 3: test to make sure the Service was created

```
kubectl get service -l "run=gowebapp"
```

Create Deployment Object for gowebapp

Step 1: Analyze a Kubernetes Deployment object for the frontend gowebapp

```
cd $HOME/gowebapp/gowebapp
```

Let's inpsect the `gowebapp-deployment.yaml` file. You can do so be opening it in your favorite command line editor, use the built-in editor, or we have embedded the file directly in the lab instructions here for easy viewing. This kubernetes configuration file contains the definition of what the desired state should be for our Kubernetes Deployment object for the frontend gowebapp.

gowebapp-deployment.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: gowebapp
  labels:
    run: gowebapp
    tier: frontend
spec:
  replicas: 2
  selector:
    matchLabels:
      run: gowebapp
      tier: frontend
  template:
    metadata:
      labels:
        run: gowebapp
        tier: frontend
    spec:
      containers:
      - name: gowebapp
        env:
        - name: MYSQL_ROOT_PASSWORD
          value: mypassword
        image: localhost:5000/gowebapp:v1
        ports:
        - containerPort: 80
```

Step 2: create the Deployment defined above

Use kubectl to create the service defined above

```
kubectl apply -f gowebapp-deployment.yaml
```

Step 3: test to make sure the Deployment was created

```
kubectl get deployment -l "run=gowebapp"
```

Test Your Application

Step 1: Access gowebapp through the NodePort service

Access your application at the NodePort Service endpoint: http://<EXTERNAL-IP>:<NodePort>.

Note

To get the EXTERNAL-IP of your host, run the command `lab-info` in your lab terminal

> **Note**
>
> To get the NodePort for your service, run the command `kubectl get svc gowebapp` in your lab terminal

```
*** EXAMPLE OUTPUT ****

$ kubectl get svc gowebapp

NAME       TYPE       CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
gowebapp   NodePort   10.107.169.105   <none>        80:30500/TCP   2m

# In this example, the NodePort is 30500
```

> **Note**
>
> Sometimes, a firewall may block the egress required for this to work. If you can't browse to it from your machine, then to try to curl `http://localhost:<nodeport>` from the lab machine terminal. Alternatively, you may be able to use your phone to access over the cellular network

You can now 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.

Lab 02 Conclusion

Congratulations! You have successfully deployed your applications using Kubernetes!