Two tier app setup on K8s using Minikube

Table of Contents

Objectives

Setup kubernetes environment for deploying two tier application using minikube.

Overview

Minikube is a free tool used with kubectl tool for testing kubernetes infrastructure for dev/qa environment. It saves the resource consumption and managing different VMs for multiple applications or using too much cloud resources which makes you pay a lot. It also enhance the functionality as it only uses minimum resource and management is easy. However, this is not suited for production environments.

Pre requisites/System Requirements

1) Ubuntu machine with root previleges.
2) Atleast 2GB RAM, 2 core CPU or more and 20GB free space available on machine.
3) Docker installed on machine for running containers
4) Docker hub account for managing and maintaining images same as Git.

Installation:

Installation of minikube and Kubectl

Note: We are creating document with ubuntu 22.04 version please choose correct version while installing as steps maybe differ for Windows or mac machine.

Check OS version using below cmds:

lsb_release -a

cat /*etc*/os-release

First check os version and follow guide as per recommendation.

Use below cmds to start installation of minikube (install as per os version):

```
curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
sudo install minikube-linux-amd64 /usr/local/bin/minikube && rm minikube-linux-amd64
```

verify installation using "minikube status" and start the tool.

Use below cmds to start installation of kubectl (install as per os version):

Install dendencies using:

```
curl -LO https://dl.k8s.io/release/$(curl -L -s
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl
```

To download a specific version, replace the $(curl -L -s https://dl.k8s.io/release/stable.txt) portion of the command with the specific version.

Install kubectl using:

```
sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl
```

Verify installation using:

```
kubectl version --client
```

Now, as both the required tools installed, build the docker file using below cmd:

docker build –t <username>/<repo name>:<tag name> <path to docker file>

example:

docker build –t raagrawal/demorepo:latest . (dot in the last considering we are in same directory as dockerfile)

After building the image lets push it over docker hub using below cmd:

docker push <username>/<repo name>:<tag name>

Now, we need to create deployment.yaml to deploy our application to pod and service.yaml file for load balancing. You can refer to provided link in the end to get tamplate for the same or refer to SS below.

Deployment.yaml for backend:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: tomcat
spec:
  replicas: 1
  selector:
    matchLabels:
      app: tomcat
  template:
    metadata:
      labels:
        app: tomcat
    spec:
      containers:
      - name: tomcat-service
        image: agrawalram/k8s-tomcat-app:latest
        ports:
        - containerPort: 8080
#started resource alocator limit
        resources:
          limits:
            cpu: "1"
            memory: "1Gi"
          requests:
            cpu: "500m"
            memory: "512Mi"
#ended resource alocator limit
        env:
        - name: POSTGRES_HOST
          value: postgres-service
        envFrom:
        - secretRef:
            name: postgres-secret
```

Deployment.yaml for database:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: postgres
spec:
  replicas: 1
  selector:
    matchLabels:
      app: postgres
  template:
    metadata:
      labels:
        app: postgres
    spec:
      containers:
      - name: postgres
        image: agrawalram/k8s-postgres-app:latest
        ports:
        - containerPort: 5432
#started resource alocator limit
        resources:
          limits:
            cpu: "1"
            memory: "1Gi"
          requests:
            cpu: "500m"
            memory: "512Mi"
#ended resource alocator limit
        envFrom:
        - secretRef:
            name: postgres-secret
        volumeMounts:
        - name: postgres-storage
          mountPath: /var/lib/postgresql/data
      volumes:
      - name: postgres-storage
        persistentVolumeClaim:
          claimName: postgres-pvc
```

Service.yaml for frontend:

```yaml
apiVersion: v1
kind: Service
metadata:
  name: tomcat-service
spec:
  type: NodePort      # Added NodePort type
  selector:
    app: tomcat
  ports:
  - port: 8080
    targetPort: 8080
    nodePort: 30080
```

Service.yaml for database:

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

Here we are using persistant volume and secret for database credentials to prevent our data from accidental deletion due to pod restart. So we need to create the same before applying deployment.yaml file. Refer to below SS for persistant volume

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: postgres-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

Create postgres secret using cmd:

```
kubectl create secret generic postgres-secret --from-literal=POSTGRES_DB=<your_database> --from-literal=POSTGRES_USER=<your_user> \ --from-literal=POSTGRES_PASSWORD=<your_password>
```

Now apply our storage file then deployment file and then service file respectively using cmd:

kubectl apply –f <yaml filename>

Bonus:

To manage configuration files using ConfigMaps in the Kubernetes setup refer below:

For example Tomcat configuration:

tomcat-configmap.yaml:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: tomcat-config
data:
  server.xml: |
    <?xml version="1.0" encoding="UTF-8"?>
    <Server port="8005" shutdown="SHUTDOWN">
      <Service name="Catalina">
        <Connector port="8080" protocol="HTTP/1.1"
                   connectionTimeout="20000"
                   redirectPort="8443"
                   maxThreads="300"
                   minSpareThreads="25"
                   maxSpareThreads="75"
                   acceptCount="100"/>
        <Engine name="Catalina" defaultHost="localhost">
          <Host name="localhost" appBase="webapps"
                unpackWARs="true" autoDeploy="true">
          </Host>
        </Engine>
      </Service>
    </Server>

  context.xml: |
    <?xml version="1.0" encoding="UTF-8"?>
    <Context antiResourceLocking="false" privileged="true">
      <ResourceLink name="jdbc/myDataSource"
                    global="jdbc/myDataSource"
                    type="javax.sql.DataSource"/>
    </Context>

  catalina.properties: |
    tomcat.util.scan.StandardJarScanFilter.jarsToSkip=\
    bootstrap.jar,commons-daemon.jar,tomcat-juli.jar
    org.apache.catalina.startup.EXIT_ON_INIT_FAILURE=true
    org.apache.catalina.startup.STRICT_SERVLET_COMPLIANCE=false
    org.apache.catalina.startup.MAX_PARALLEL_STARTUP_THREADS=4
```

Apply configmap.yaml file

kubectl apply -f tomcat-configmap.yaml

tomcat-deployment.yaml

```yaml
kind: Deployment
metadata:
  name: webapp
spec:
  replicas: 1
  selector:
    matchLabels:
      app: webapp
  template:
    metadata:
      labels:
        app: webapp
    spec:
      containers:
      - name: tomcat
        image: your-dockerhub-username/tomcat-app:latest
        ports:
        - containerPort: 8080
        env:
        - name: POSTGRES_HOST
          value: postgres-service
        envFrom:
        - secretRef:
            name: postgres-secret
        volumeMounts:
        - name: tomcat-config
          mountPath: /usr/local/tomcat/conf/server.xml
          subPath: server.xml
        - name: tomcat-config
          mountPath: /usr/local/tomcat/conf/context.xml
          subPath: context.xml
        - name: tomcat-config
          mountPath: /usr/local/tomcat/conf/catalina.properties
          subPath: catalina.properties
      volumes:
      - name: tomcat-config
        configMap:
          name: tomcat-config
```

Update the deployment

kubectl apply -f tomcat-deployment.yaml

Useful Commands

kubectl get pods –o wide

kubectl get all

kubectl logs -f deployment/webapp

Troubleshooting

NA

Useful Links

https://kubernetes.io/docs/tasks/tools/

https://minikube.sigs.k8s.io/docs/start/?arch=%2Fwindows%2Fx86-64%2Fstable%2F.exe+download

https://kubernetes.io/docs/concepts/workloads/controllers/deployment/

https://kubernetes.io/docs/concepts/services-networking/service/