**ECGR 4090/5090 Cloud Native Application Architecture**

**Lab 11 : Kubernetes Storage**

Any data stored in the application is only valid when the pod exists. When it terminates, the data is lost. This is okay for stateless applications. However, if state needs to be persisted (for example, a database), K8s has a feature called Persistent Volume (PV). The infrastructure team creates the PV, and application developers access the PV through a Persistent Volume Claim (PVC). Pods that use the PVC are created using a StatefulSet. Pods in a StatefulSet have a unique ordinal index and a stable network identity. Although individual Pods in a StatefulSet are susceptible to failure, the persistent Pod identifiers make it easier to match existing volumes to the new Pods that replace any that have failed.

All these are specified using appropriate YAML manifest files.

Different types of PVs are supported by K8s. For a complete list see - <https://kubernetes.io/docs/concepts/storage/volumes/>

In this lab, we will use *hostpath* as a PV to use the local disk for storage for MongoDB. With hostPath volumes, your data lives in /tmp on the node the Pod is scheduled onto and does not move between nodes. If a Pod dies and gets scheduled to another node in the cluster, or the node is rebooted, the data is lost. However, for the purposes of this lab, and the single node cluster that we are using, *hostpath* works well. Alternatively, you could use a Network File System (NFS) for storage. In a public cloud, vendor specific storages could be used (for example, AWS Elastic Block Storage).

We first define a YAML manifest for the storage class. A StorageClass provides a way for administrators to describe the "classes" of storage they offer. Different classes might map to quality-of-service levels, or to backup policies, or to other policies determined by the cluster administrators. Kubernetes itself is un-opinionated about what classes represent. Each StorageClass contains the fields provisioner, parameters, and reclaimPolicy, which are used when a PersistentVolume belonging to the class needs to be dynamically provisioned.

The name of a StorageClass object is significant, and is how users can request a particular class.

Under a new lab11 directory, copy and paste the YAML below to a file named storageclass.yaml

*kind: StorageClass*

*apiVersion: storage.k8s.io/v1*

*metadata:*

*name: microk8s-hostpath*

*annotations:*

*storageclass.kubernetes.io/is-default-class: "true"*

*provisioner: microk8s.io/hostpath*

Next we define the Persistent Volume (PV) for this storage class that captures the details of the implementation of the storage. We also define a Persistent Volume Claim (PVC) for the application to request storage.

Copy and paste the following to a file named storage.yaml

*kind: PersistentVolume*

*apiVersion: v1*

*metadata:*

*name: mongodb-pv*

*labels:*

*app: mongodb*

*type: local*

*spec:*

*storageClassName: microk8s-hostpath*

*capacity:*

*storage: 1Gi*

*accessModes:*

*- ReadWriteOnce*

*hostPath:*

*path: "/var/data"*

*---*

*kind: PersistentVolumeClaim*

*apiVersion: v1*

*metadata:*

*name: mongodb-pv-claim*

*labels:*

*app: mongodb*

*spec:*

*storageClassName: microk8s-hostpath*

*capacity:*

*accessModes:*

*- ReadWriteOnce*

*resources:*

*requests:*

*storage: 1Gi*

We define a ConfigMap to store non-confidential data in key-value pairs. Pods can consume ConfigMaps as environment variables, command-line arguments, or as configuration files

Note: Kubernetes has a separate feature called Secrets to store confidential data.

Copy and paste the following to file named configmap.yaml

*apiVersion: v1*

*kind: ConfigMap*

*metadata:*

*name: mongodb-configuration*

*labels:*

*app: mongodb*

*data:*

*MONGODB: testdb*

Next we define a StatefulSet with a single replica to deploy MongoDB.

Copy and paste the following to a file named statefulset.yaml

*apiVersion: apps/v1*

*kind: StatefulSet*

*metadata:*

*name: mongodb-statefulset*

*labels:*

*app: mongodb*

*spec:*

*serviceName: "mongodb"*

*replicas: 1*

*selector:*

*matchLabels:*

*app: mongodb*

*template:*

*metadata:*

*labels:*

*app: mongodb*

*spec:*

*containers:*

*- name: mongodb*

*image: mongo:latest*

*envFrom:*

*- configMapRef:*

*name: mongodb-configuration*

*ports:*

*- containerPort: 27017*

*name: mongodb*

*volumeMounts:*

*- name: mongodb-data*

*mountPath: /var/lib/mongodb/data*

*volumes:*

*- name: mongodb-data*

*persistentVolumeClaim:*

*claimName: mongodb-pv-claim*

And finally we define a service to make MongoDB available as a service.

Copy and paste the following to a file name service.yaml

*apiVersion: v1*

*kind: Service*

*metadata:*

*name: mongodb-service*

*labels:*

*app: mongodb*

*spec:*

*ports:*

*- port: 27017*

*name: mongodb*

*type: NodePort*

*selector:*

*app: mongodb*

We are ready to launch. Apply the YAML manifest sequentially.

*$ kubectl apply -f storageclass.yaml*

*$ kubectl apply -f configmap.yaml*

*$ kubectl apply -f storage.yaml*

*$ kubectl apply -f statefulset.yaml*

*$ kubectl apply -f service.yaml*

Make sure everything is up and running

*$ kubectl get pods*

*$ kubectl get services*

*$ kubectl get storageclass*

*$ kubectl get statefulset*

Access MongoDB shell from within the cluster

*$ kubectl get pods*

*$ kubectl exec -it <mongo pod name> -- mongo*

Try a few of the MongoDB shell commands from Lab8 before exiting the shell.

Let’s access the MongoDB service from outside the cluster using the Go MongoDB driver from Lab 8 (main.go).

Find the cluster IP

*$ ls*

Find the MongoDB port (the second port number)

*$ kubectl get services*

Edit mongodbEndpoint in mongo.go with clusterIP:MongoDBport

*$ go run mongo.go*

You should see the same messages as Lab 8

**To do**

In the Lab 8 To-do you had MongoDB and the web server running as two separate containers. Run the two containers under Kubernetes. You should be able to curl to the webserver from outside K8s to test your implementation.

**Stateful vs deployment**

Stateful pods are used when we need to use query language.

Deployments I usually apps.

End deployments pods can be randomly created and deleted because every pod is similar whereas in stateful pods every pod is different that means consider 3 mySQL pods each my SQL pod will be different than the other so we cannot randomly create create or delete apart in stateful

1. export NODE\_PORT=$(kubectl get services/mongodb-service -o go-template='{{(index .spec.ports 0).nodePort}}'); echo NODE\_PORT=$NODE\_PORT

32658

1. *export NODE\_IP=$(kubectl describe nodes | grep InternalIP | awk '{print $2}');echo $NODE\_IP*

**192.168.85.128**

1. sudo docker build . -t 192.168.85.128:32000/webserver:registry

sudo docker push 192.168.85.128:32000/webserver:registry

Text

Description automatically generated

curl <http://192.168.85.128:30001/add?item=bag&price=100>

**curl** [**http://192.168.85.128:30001/list**](http://192.168.85.128:30001/list)

if we use it on command line

curl “<http://192.168.85.128:30001/add?item=bag&price=100>”