Scalability & Availability

Learning Outcomes

After completing the lab, you will be able to:

1. Configure Memory and CPU Quotas for a Namespace
2. Scale application vertically & horizontally
3. Implement storage using Persistent Volume and Persistent Volume Claims

Configuring resource quota for a namespace

1. To set the desired hard limits for each named resource, create a resource quota object using below specification.

```
name -> resource-quota
namespace -> [student-name]
limits:
cpu -> 2
memory -> 8Gi
pods -> 10
```

Solution

Create `deployment/resource-quota.yaml` file with the following content. Replace `[student-name]` with your namespace name.

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: resource-quota
  namespace: [student-name]
spec:
  hard:
    cpu: 2
    memory: 8Gi
    pods: 10
```

2. Ensure kubectl context is set to minikube with the right namespace
3. Run the below command

```
kubectl apply -f deployment/resource-quota.yaml
```

4. Verify that the quota is created in the appropriate namespace and the limits are set as specified in the specification

```
kubectl get quota
kubectl describe quota resource-quota
```

5. Delete deployments and pods

```
kubectl delete deploy pages
kubectl delete deploy mysql
kubectl delete job flyway-job
```

6. Create mysql deployment

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

7. Check the status of mysql deployment. Have the pods been scheduled? How do you debug and troubleshoot this error?
8. Check the logs and events of mysql

   `kubectl describe deploy mysql`

   `kubectl get events --sort-by=.metadata.creationTimestamp`

9. Error might look something like this:

   `failed quota: resource-quota: must specify cpu,memory`

10. In order to fix this, we need to specify the compute requests in all resources needing them. This applies to all pods and deployments which has compute resource requirements. In our case we have 2 deployments and 1 job .
11. We need to add the resource requests to the container section of `pages-deployment` , `mysql-deployment` and `flyway-job` to include the resource requests. Note that, if you request for more than available resources, your pods will not get scheduled.

```
resources:
      requests:
        memory: 500Mi
        cpu: 0.25
      limits:
        memory: 900Mi
        cpu: 1
```

12. Create the deployments and run the flyway job.

```
kubectl apply -f deployment/mysql-deployment.yaml
kubectl apply -f deployment/flyway-job.yaml
kubectl apply -f deployment/pages-deployment.yaml
```

13. Pages deployment will take some time to startup. Confirm the pod has started successfully before moving on to the next step.

14. Test the pages application by navigating to http://localhost:8080/pages after port-forwarding using kubectl on port 8080.

Vertical Scaling

1. Scale the pages application vertically, by adding 100 Mi to memory.

2. Update the pages-deployment.yaml file with the new request values and re-deploy pages.

```
resources:
      requests:
        memory: 600Mi  #old value 500Mi
        cpu: 0.25
      limits:
        memory: 900Mi
        cpu: 1
```

   Alternatively, you can edit the deployment imperatively like:

   `kubectl edit deploy pages`

   vi-editor opens up automatically with the live deployment file which can be edited. Use this option if you are well versed in using vi editor and save it to see the configuration updated.

Horizontal Scaling

1. Scale out the pages application horizontally, by increasing the number of replicas to 2.

2. There are 2 ways of acheiving this:

   a. Update the `Deployment.spec.replicas` field of pages-deployment.yaml file and then using the command `kubectl apply -f deployment/pages-deployment.yaml`

   b. Imperative way: `kubectl scale deploy pages --replicas=2`

3. Let's use the imperative way, which is much faster.

4. Verify the number of running pods for pages after scaling. This might take a while.

```
kubectl get deploy pages
```

```
kubectl port-forward svc/pages 8080:8080
```

```
curl http://localhost:8080/pages
```

5. Scale in the pages application to 1 replica.

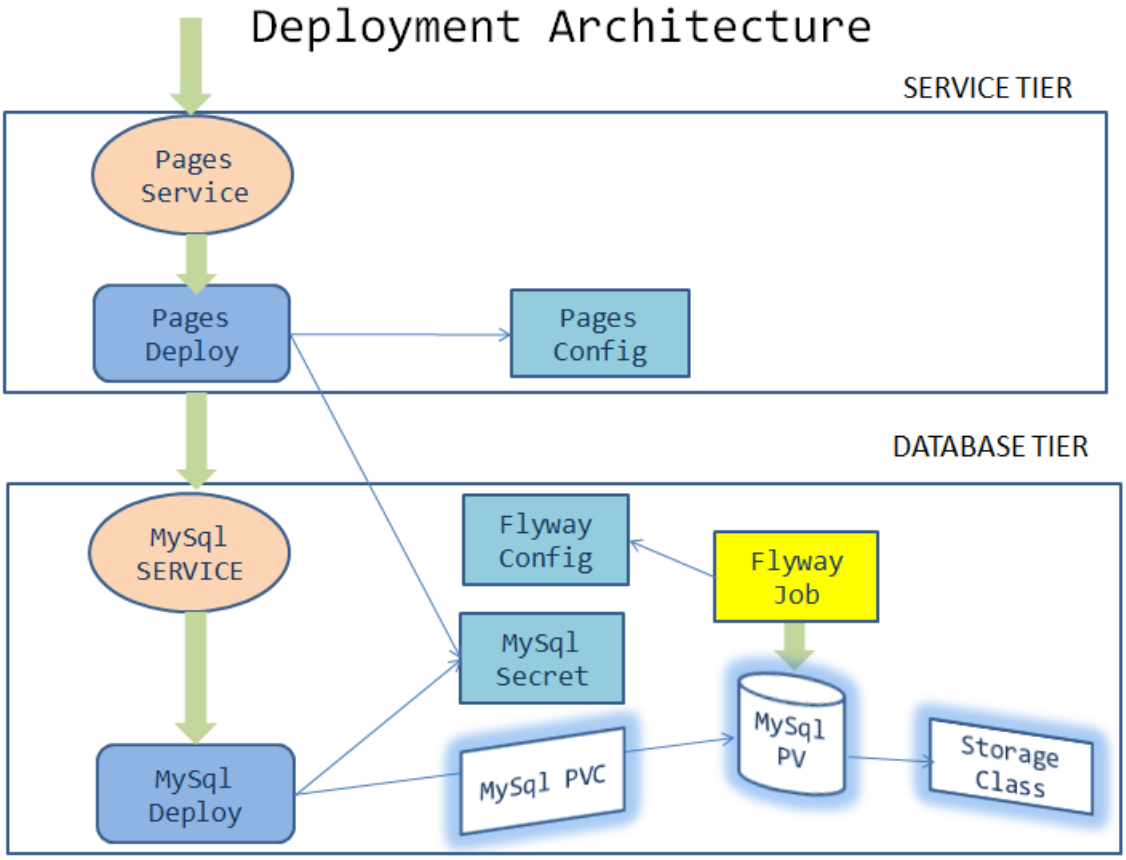
```
kubectl scale deploy pages --replicas=1
```

Peristent volume and persistent volume claims

Understanding the scenario

1. POST few entries into pages application.
2. Delete the mysql pod. Since we have the replicaset, the pod will be recreated to maintain the desired state of the deployment.
3. Wait till the pod gets recreated. Issue a GET request to the pages controller.
4. What happened? You need to run the migration job again.
5. Wait till the job is completed. Issue a GET request to the pages controller.
6. What happened to the data that was POST'ed?

Need to refactor the design for persistence of data



We have used volumes in our labs. They allowed us to mount a storage unit such as file system folder on a node or a cloud storage bucket and also share information between nodes.These regular volumes are evicted when the pod is evicted.

Kubernetes provides a persistent storage solution for containers called Persistent Volumes (PV). Persistent Volume can remain alive for as long as necessary for ongoing operations.

1. Persistence Volume normally makes use of `StorageClass` objects in K8s, creating an abstraction to connect to various storage solutions for multiple storage requirements.

   For more information, refer https://kubernetes.io/docs/concepts/storage/persistent-volumes

2. Create storage class object definition in `deployment/mysql-storage-class.yaml` as given below.

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: database
  labels:
    addonmanager.kubernetes.io/mode: EnsureExists
provisioner: k8s.io/minikube-hostpath
reclaimPolicy: Retain
volumeBindingMode: Immediate
```

3. Create persistent volume definition in `deployment/mysql-pv.yaml` as given below. Replace `[student-name]` with your namespace name at 2 places.

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: mysql-pv-[student-name]
  labels:
    type: local
spec:
  storageClassName: database
  capacity:
    storage: 3Gi
  accessModes:
    - ReadWriteMany
  hostPath:
    path: "/var/lib/mysql/[student-name]"
```

4. We can now request for the persistent volume, by creating a Persistent Volume Claim object with below specification in `deployments/mysql-pvc.yaml` . Replace `[student-name]` with your namespace name.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mysql-pvc
  namespace: [student-name]
spec:
  storageClassName: database
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
```

5. Edit and update the volume and volume mount section of `mysql-deployment.yaml` file to use the persistent volume claim defined earlier.

```
    ---

        volumeMounts:
          - name: mysql-persistent-storage
            mountPath: /var/lib/mysql
    volumes:
      - name: mysql-persistent-storage
        persistentVolumeClaim:
```

```
            claimName: mysql-pvc
      ---
```

6. Update the environment variable for spring datasource url in `pages-deployment.yaml` file with the below value

```
    - name: SPRING_DATASOURCE_URL
      value: jdbc:mysql://pages-mysql/pages?allowPublicKey
Retrieval=true&useSSL=false
```

7. Delete deployments and job

```
kubectl delete deploy pages
kubectl delete deploy mysql
kubectl delete job flyway-job
```

8. Create storage related objects in minikube

```
kubectl apply -f deployment/mysql-storage-class.yaml
kubectl apply -f deployment/mysql-pv.yaml
kubectl apply -f deployment/mysql-pvc.yaml

kubectl get storageclasses
kubectl get pv
kubectl get pvc
```

9. Create all the deployments and job in minikube

```
kubectl apply -f deployment/mysql-deployment.yaml
kubectl apply -f deployment/flyway-job.yaml
kubectl apply -f deployment/pages-deployment.yaml
```

10. Testing on minikube

```
kubectl get deploy
kubectl get jobs
kubectl get pods


#Delete the job as you dont need it after completion.
kubectl delete job flyway-job

kubectl port-forward svc/pages 8080:8080

curl -i -XPOST -H"Content-Type: application/json" localhos
t:8080/pages -d"{\"businessName\": \"Uber\", \"address\":
```

```
\"SanFrancisco, CA, USA\", \"categoryId\": 123, \"contactN
umber\": \"0045987869\"}"

curl localhost:8080/pages

#Stop the port forwarding by presing CTRL+C

#Let's delete the mysql deployment and recreate it

kubectl delete deploy mysql

curl localhost:8080/pages

#What's the error?
#Pages app will be DOWN, as the health checks will fail, a
nd the kubelet will kill and restart the container. See th
e logs and events.

#Let's create the mysql deployment. This time, we will not
run the flyway migration job.

 kubectl apply -f deployment/mysql-deployment.yaml

 kubectl get deploy

 kubectl get pods

 #Verify that all the pods are running.

 kubectl port-forward svc/pages 8080:8080

 curl localhost:8080/pages

 # Was the data persistent? Were you able to persist all t
he entries added?
```

Clean up and deploy

1. Delete the resource quota that we created earlier `kubectl delete quota resource-quota`
2. Delete the manifest file `deployment/resource-quota.yaml`
3. Remove resource requests added to the container section of `pages-deployment`, `mysql-deployment` and `flyway-job` as we dont need to use it any further.

```
-resources:
-      requests:
-        memory: 500Mi
-       cpu: 0.25
-      limits:
-        memory: 900Mi
-        cpu: 1
```

4. Update the `pipeline.yaml` with the appropriate tag
5. Update `scripts/deploy-to-k8s.sh` to include pv and pvc support for the pages application.

```
kubectl apply -f deployment/mysql-pv.yaml
kubectl apply -f deployment/mysql-pvc.yaml
```

6. Does the storage class `database` exist in the production cluster?

```
kubectl get storageclasses
```

7. Skip this step if the storage class `database` is already created either by you or the cluster admin.

   Manually create the storage class, without adding it to the pipeline. This would normally be done by the operations team/cluster admins.

```
kubectl apply -f deployment/mysql-storage-class.yaml

kubectl get storageclasses
```

8. Push the code to github repository

Additional tasks to explore:

1. Performing a rolling update
2. Updating the deployment with a new version
3. Roll back to a previous version of deployment

   You can do the tasks given in this link to understand the concepts of the rolling updates and roll back https://kubernetes.io/docs/tutorials/kubernetes-basics/update/update-interactive
4. Automatically scale your deployment using HPA. For more information. refer the documentation for HPA https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale-walkthrough