ROI Training

Lab:
Adding Resource Requests and Limits, and Autoscaling

Introduction

- In this lab, you will deploy a simple Node.js demo container to demonstrate resource limits and scaling
  - The events app case study will not be used

Verify Minikube Is Running

- From a codespaces terminal, check if Minikube is still running with:
  `minikube status`
- If it is stopped, start it again with:
  `minikube start`
- Verify the tunnel and port forward from the last lab is stopped:
  - In the terminal window running the port forward, press **CTRL+C** to stop it
  - Run the following command to stop the tunnel:
    `pkill -f "minikube tunnel"`

Kubernetes Metrics Server

- The Kubernetes Metrics Server collects resource usage metrics (CPU and memory) from nodes and pods exposes them via the Metrics API
    - Required for features like Horizontal Pod Autoscaling (HPA)
- Minikube does not have a metrics server by default but is available as an add-on
- Run the following command to enable the metrics server

```
minikube addons enable metrics-server
```

Open the Deployment Example

- In your codespaces terminal, change into the `HPA-demo` folder
  ```
  cd /workspaces/eventsapp/HPA-demo/
  ```
  - This folder was created when the Git repo was pulled earlier in the course
- Open the `deployment.yaml` file in the editor and answer the questions on the following slide

Investigate the YAML

- How many replicas will be created?
- What is the name of the image deployed?
- What is the memory limit?
- What is the CPU limit?

Deploying the `deployment.yaml`

- Deploy the deployment:
  ```
  kubectl apply -f deployment.yaml
  ```
- Verify it was deployed and wait for all replicas to be ready:
  ```
  kubectl get deployments
  kubectl get pods
  ```
- View the current resource usage for each pod:
  ```
  kubectl top pods
  ```

Open the Autoscale Example

- Open the `autoscale.yaml` file and answer the following questions:
  - What kind of object is this YAML creating?
  - Which deployment will it affect?
  - What is the min and max replicas?
  - What metric limit will be used for scaling?

Deploying the `autoscale.yaml`

- Deploy the horizontal pod autoscaler (HPA):
  `kubectl apply -f autoscale.yaml`
- Verify it was deployed and wait for the target % to be known:
  `kubectl get hpa`
- View the current resource usage for each pod:
  `kubectl top pods`

```
$ kubectl get hpa
NAME                REFERENCE                    TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
autoscale-app-hpa   Deployment/autoscale-app   0%/60%    2         6         2          33s
$
$ kubectl top pods
NAME                                CPU(cores)   MEMORY(bytes)
autoscale-app-545d96b96c-4ztzg   0m           19Mi
autoscale-app-545d96b96c-qb2g9   0m           19Mi
```

Creating a Service to Test Autoscaling

- A service is needed to be able to route traffic to the pods
  - The service will automatically route traffic to any new pod created by the HPA
- Investigate the `service.yaml` file in the `HPA-demo` folder
  - Ensure you understand it
- Apply the service.yaml:

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

- Open a new terminal (+ button) and create the port forward and tunnel:

```
minikube tunnel & kubectl port-forward service/autoscale-app-svc 8080:80
```

- Click the **Open in Browser** button

Creating a Service to Test Autoscaling

- Try reloading the page
  - You may notice the page does not load instantly; it takes a second or so to load

- This application contains logic to cause it to utilize CPU resources
  - It is written in Node.js and we will now investigate the code
- Switch back to the first terminal session and view the service

```
kubectl get service
```

- Record the External IP of the **autoscale-app-svc** service
  - You will need it several times in this lab

Investigating the Demo App

- Retrieve the list of pods:
  `kubectl get pods`
  - From the list, copy one of the `autoscale-app` pod names
- Exec inside that pod and list the files:
  `kubectl exec -i -t POD-NAME-HERE -- /bin/bash`
  `ls`
- View the app.js file
  `cat app.js`
- Investigate the code and locate the for loop
  - This loop performs some simple math operations 4,000,000 times
- When done, exit the pod:
  `exit`

Autoscaling the App

- In a codespaces terminal, run the following command to keep requesting the app's page in a loop:

```
while true; do curl http://SERVICE-IP-HERE/; done;
```

- Open another terminal session (you will have three now) and investigate the HPA, and pods using the following commands:

```
kubectl get hpa
kubectl top pods
kubectl get pods
```

- Keep executing these commands and watch for changes
  - The application should start to scale

Autoscaling the App (continued)

- The app should scale the number of pods to handle all the traffic
- This is a very simple load test; there are better load testing tools such as Apache Bench that could be used
- When done, press **CTRL+C** in the session running the loop
  - This will stop the load
- Investigate the HPA, and pods using the following commands:

```
kubectl get hpa
kubectl top pods
kubectl get pods
```

- After a few (5) minutes, the pods should scale back to the minimum number

Clean Up

- Stop the port forward:
  - In the terminal window running the port forward, press **CTRL+C** to stop it
  - Then run the following command to stop the tunnel:

    ```
    pkill -f "minikube tunnel"
    ```

- Delete the autoscale demo with the following commands:

  ```
  kubectl delete deployment autoscale-app
  kubectl delete hpa autoscale-app-hpa
  kubectl delete service autoscale-app-svc
  ```

Success

- **Congratulations**! You have successfully used resource limits and a HorizontalPodAutoscaler
  - Set CPU and memory limits for a container
  - Created an HPA to scale the number of pods
  - Load tested the service and caused the HPA to scale