# Activity: Adding Resource Requests and Limits, and Autoscaling

## Introduction

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

# Open the Deployment Example

- In Cloud Shell, change into the statefulset-demo folder

  ```
  cd ~/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 (in the `/eventsapp/HPA-demo` folder) in the editor 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%/50%     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 `/eventsapp/HPA-demo` folder
  - Ensure you understand it
- Apply the service.yaml:

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

- View the service and wait for the external IP to be available

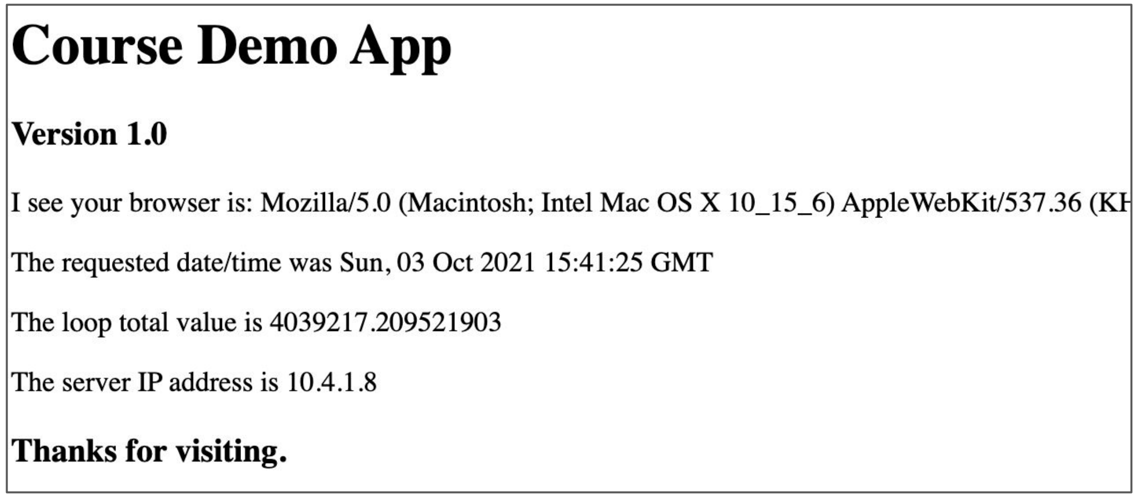
```
kubectl get service
```

- Record the External IP of the service
  - You will need it several times in this activity

# Creating a Service to Test Autoscaling

- Once the external IP is available, try viewing it in a browser
  - Try reloading the page
  - You may notice the page does not load instantly

    It takes a second or two 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



**Course Demo App**

**Version 1.0**

I see your browser is: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_6) AppleWebKit/537.36 (KH

The requested date/time was Sun, 03 Oct 2021 15:41:25 GMT

The loop total value is 4039217.209521903

The server IP address is 10.4.1.8

**Thanks for visiting.**

# 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

- Open 2 cloud shell terminal tabs
- In the first tab, run the following command to keep requesting the app's page in a loop:

  ```
  while true; do curl http://SERVICE-IP-HERE/; done;
  ```
- Switch to the other cloud shell tab 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 tp scale
- You can also open a 3rd cloud shell tab and run the while loop again

## Autoscaling the App

- The app should scale the number of pods until it can 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 any cloud shell window you have a loop running
  - 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 minutes the pods should scale back to the minimum number

## Clean Up

- 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 Horizontal Pod AutoScaler
  - 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