Project 10: Kubernetes

Goal of this project

The goal of this project is to containerise the deployment of the Sparta test app (which uses NodeJS v20) and database using a Kubernetes (Minikube) cluster.

GitHub repo link

The GitHub repository for this project is accessible here

Prerequisites

1. Docker Desktop
2. Docker Hub container image for Sparta test app

Research

Kubernetes

- **Kubernetes (or K8s)**: an open-source container orchestration system that automates the deployment, scaling, & management of apps running in containers
- Helpful because it can self-heal pods, automatically scale resources to meet changing needs based on traffic, & apply updates to apps with zero down-time using Rollout Strategies
- It can automate:
  - starting new apps when needed
  - restarting apps if they crash
  - spreading out work so that no one part of the system is overloaded
  - scaling up or down based on demand
- Companies that use K8s:
  - Spotify
  - Booking.com
  - Netflix
- **Cluster**: a Kubernetes environment, which includes one or more nodes; has two elements:
  - **Control Plane or master/control node**: the centralised brain of a cluster; governs and coordinates the cluster's operations, schedules new containers onto nodes, monitors the cluster's health, and provides an API that we interact with
  - **Data plane**: made up of worker nodes and their pods; carries out the policies set on the Control Plane **- Parts of the Control Plane (non-exhaustive):**
  - **kube-apiserver:** essentially the cluster's front door; the part of the Control Plane that allows us to interact with a running K8s cluster; all administrative commands & resource requests pass thru this
  - **etcd**: stores all config data and the current state of the cluster
  - **kube-controller-manager:** starts and runs K8s's built-in controllers; continually adjusts the cluster's state to ensure it matches the desired state; creates, scales, & deletes objects in response to API requests or load changes
  - **kube-scheduler:** assigns new pods (i.e. containers) onto the nodes in a cluster based on resource requirements, constraints, & policies **- Parts of the data plane (i.e. worker nodes):**
  - **kubelet:** the administrative agent that runs on each node; it communicates with the Control Plane to receive instructions; ensures pods are running and reports their statuses to Control

Plane; responsible for pulling container images and starting them in response to scheduling requests
    - **kube-proxy:** configures host's networking system on each node in a cluster so that traffic can reach the cluster's services
    - **container runtime**: software that runs & manages containers (e.g. Docker or Containerd); creates and manages containerised apps within Pods
- **kubectl**: the CLI tool used to interact with clusters
- **Manifest file**: a YAML file that describes the desired state of the K8s objects you want to create/manage (i.e. it's declarative); can be reused across environments; you can group multiple related resources in one file and delineate them with `---`; these resources are then created with a `kubectl apply -f <file path>` command
- **Minikube**: a development/learning tool that allows you to run a single-node K8s cluster on your local machine
- **Ingress**: if you have multiple services that need to be accessed via the same external IP, routes traffic to a service based on hostnames or paths; see example here

Managed K8s services

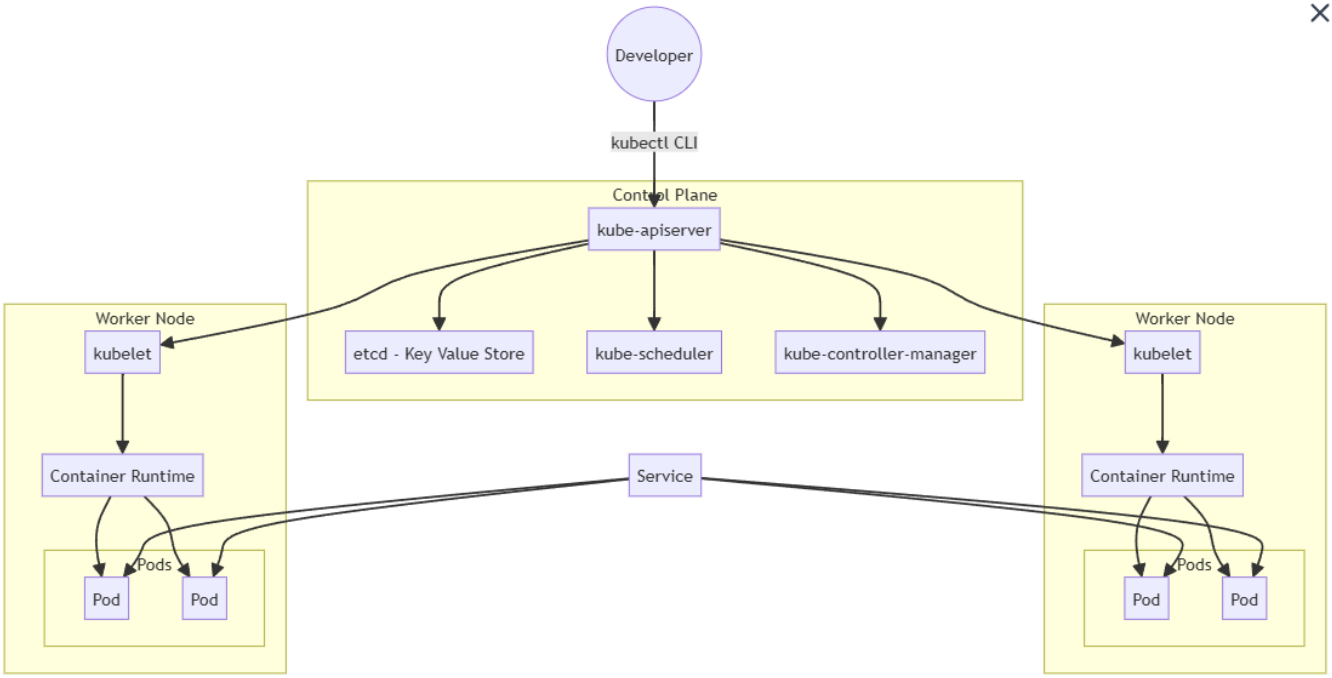
- a paid service in which a third party (e.g. AWS's EKS, Azure's AKS, GCP's GKE) handles the lifecycle of Kubernetes clusters to simplify containerised environments for the client
- **Benefits of using a managed K8s service**:
    - reduced need for in-house expertise, which is usually required when you're running a K8s environment at scale
    - frees up teams from configuring/managing K8s so they can focus less on the infrastructure and more on the product
    - usually offer enhanced security protocols
- **Downsides of using a managed K8s service:**
    - they come with an added cost
    - you may have less control over some aspects of the cluster environment

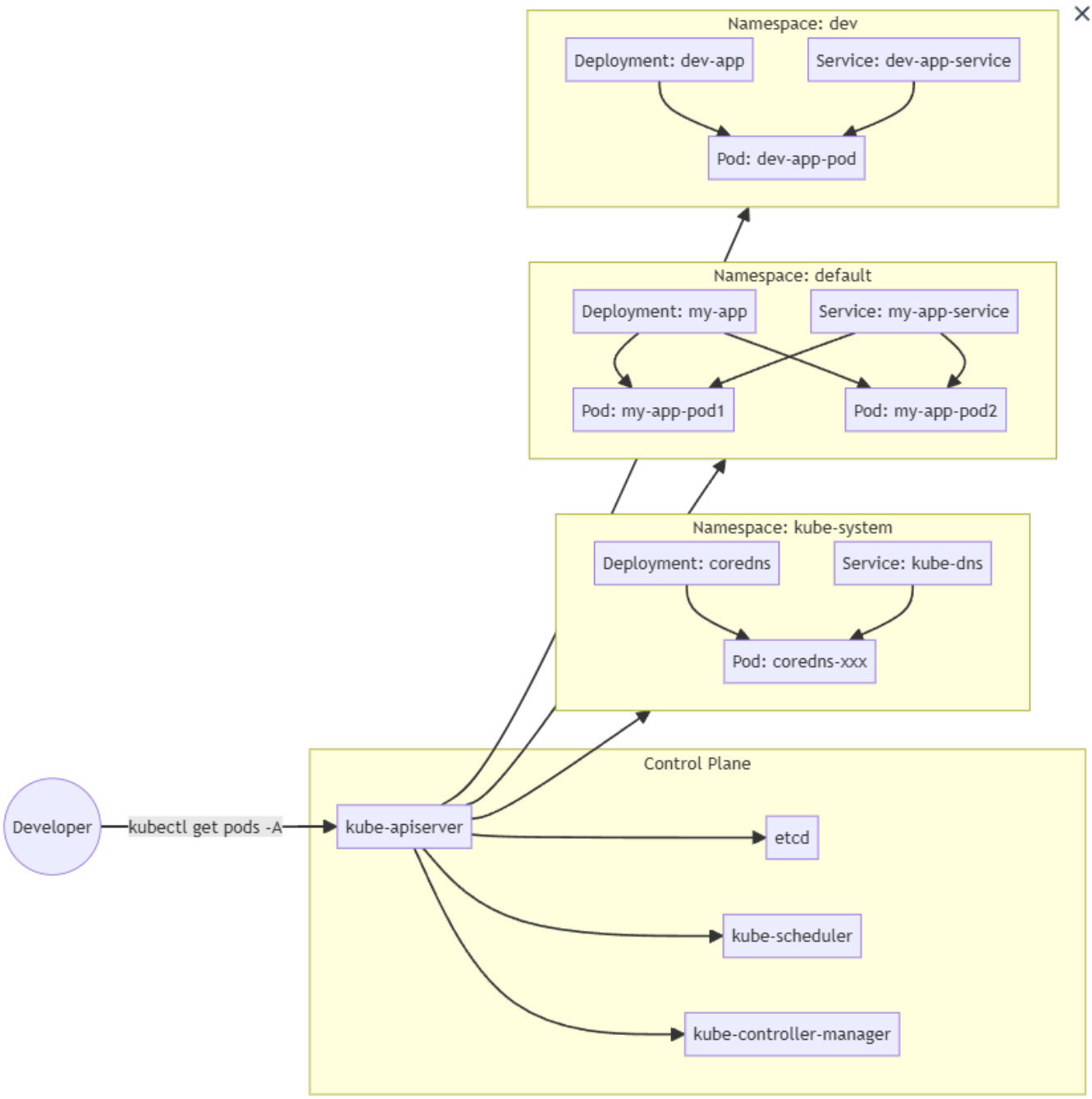
Container security

- you should store containers on secure registries/repositories with access control, otherwise they can be modified, deleted, or be subject to data breaches
- you should also sign images e.g. through Docker Content Trust or Notary to verify them
- use vulnerability scanners on images
- to reduce attack surfaces, you should include only the components the application needs in the container
- use trustworthy (e.g. official/maintained) images
- isolate container networks
- **Pros of using maintained images as base container images**:
    - reliable
    - have clear documentation
    - best practices are usually implemented already
    - easy to pull
- **Cons**:
    - less flexibility in terms of customisation

- you have no control over their size (e.g. if you only need some features), which can increase performance overhead
- you may become too dependent on the developer which could introduce issues in the long run
- if you're deploying at scale, there may be costs incurred

Kubernetes architecture diagrams by Labex:



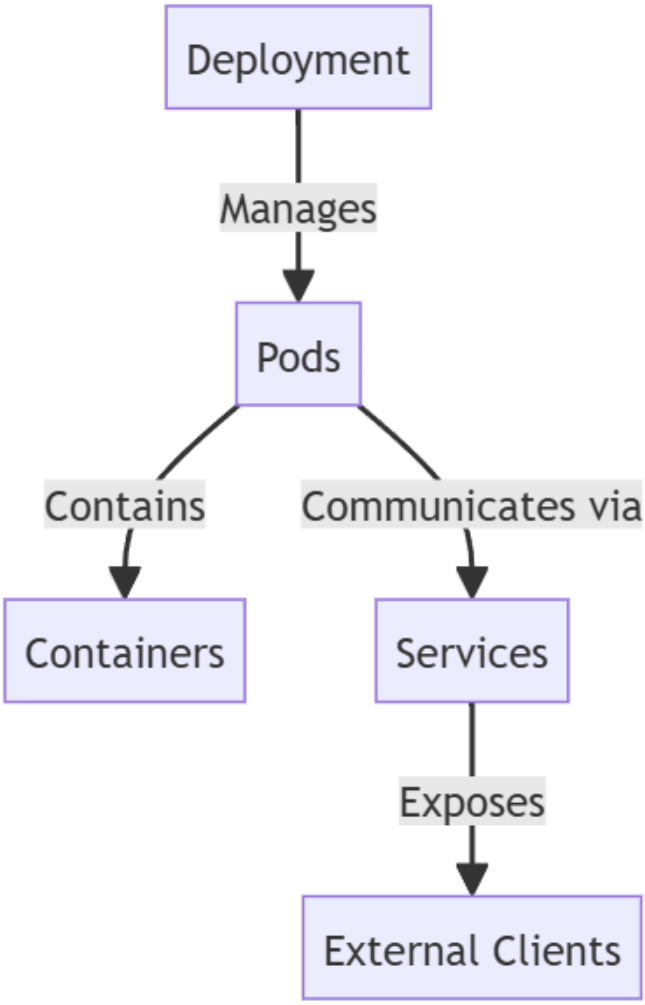


Kubernetes objects

- **Pod**: essentially a box that can hold one or more containers which share a network and storage and work together to run an app; usually represents a single instance of a running app; smallest deployable unit in K8s
- **Replicas**: identical Pods running within a cluster at the same time for high availability and scaling as one replica/pod can only handle a certain amount of concurrent requests before the app becomes slow/unresponsive; replicas can be managed by a deployment or a **ReplicaSet**, which creates new replicated pods if one fails/is deleted (but provides fewer features than a deployment)
- **Deployment:** an object that manages a ReplicaSet; has a rolling update mechanism as well as rollback feature
- **Service**: a method for exposing an application on a pod internally or externally; it provides a fixed IP address and DNS name that always point to the set of pods it manages, which is important because pods' actual IP addresses change when created & destroyed so you would otherwise not always be able

to connect to them; it defines a policy for how to reliably access the pods by correctly routing external & internal traffic
    - **Types of services:**
        - **ClusterIP** (the default): creates an internal service with an IP address that allows ONLY internal (i.e. in-cluster) network traffic to pods
        - **NodePort**: exposes pods internally as above, but also exposes the same port in each node (which will be within the range 30000-32767; doesn't need to be manually defined) to a `targetPort` (e.g. 8080) for traffic outside of the cluster
        - **LoadBalancer**: does what both of the above do but also creates external network infrastructure to distribute network requests across all of the pods to prevent any from being overwhelmed (via an even **round robin approach**) ; this is the best choice when pods need to be exposed to predictable URLs, though they are more expensive and require a cloud provider to offer load balancing services as they aren't a native K8s object; see more explanation here
- **Node**: a worker node; a physical or virtual machine that will host and run your pods; these offer the computational and storage resources needed to run pods
- **Namespaces**: logical partitions within a cluster that help organise and manage resources; allow you to apply policies, access controls, & resource quotas on a granular level
    - **Benefits of namespaces:**
        - helpful when you need to **group related workloads** (e.g. by team or environment)
        - **enhance security** and access control by restricting which users can view/modify resources in a namespace
        - **simplify resource management** by effectively applying limits, network policies & other cluster-wide configs
- **Context:** the combination of a cluster, user, and namespace

Diagram by labex.io



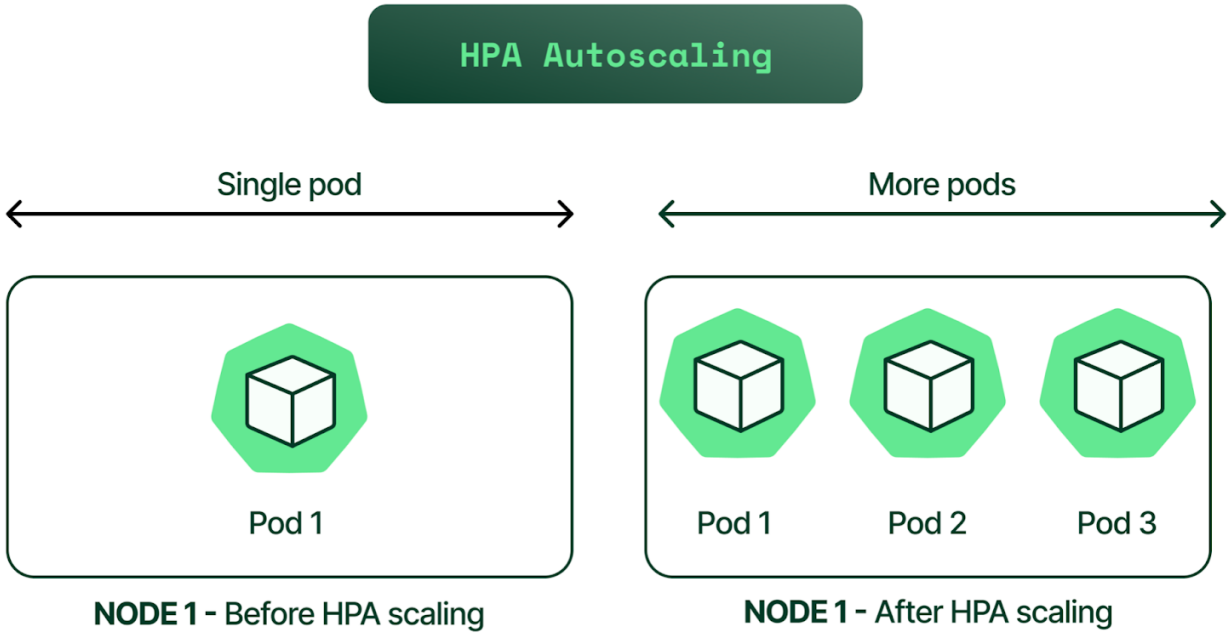
Lifecycle

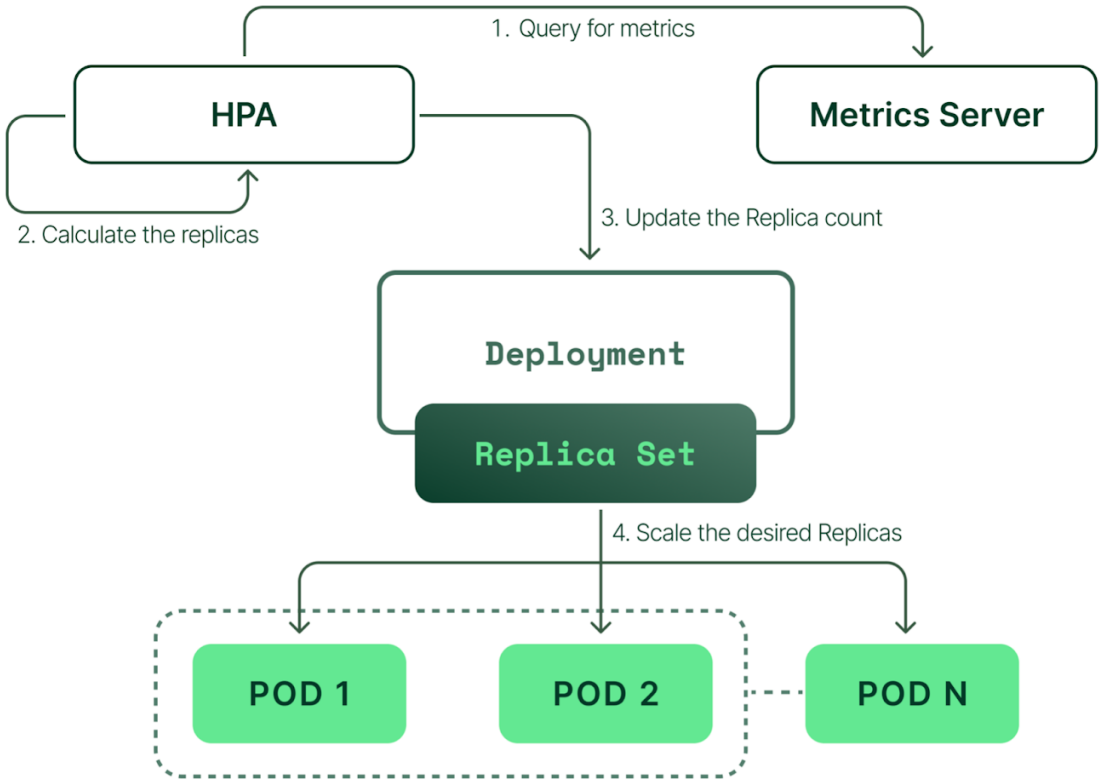
1. When a new app is deployed, the Control Plane places the pods on appropriate nodes
2. The kubelet on each node ensures the pods are healthy and running as instructed
3. Services route traffic to the correct pods, allowing clients to access apps

Autoscaling

- **Autoscaling:** a K8s feature allowing a cluster to increase/decrease the number of nodes or pod resources in response to demand
- 3 methods of autoscaling with K8s:
    1. **Horizontal Pod Autoscaler (HPA)**: automatically adds/removes pods to handle changing traffic loads and improve performance & availability; by default, it checks resource metrics via the Metrics Server add-on every 15 seconds (the default metric being average CPU usage, though the metric can be customised) — note that it CAN'T add new nodes, as only a Cluster Autoscaler can do this
    2. **Vertical Pod Autoscaler (VPA)**: automatically assigns/unassigns CPU & memory resources (i.e. reservations) for pods
    3. **Cluster Autoscaler**: automatically adds/removes nodes in a cluster based on pods' requested resources

Diagrams by kubecost.com:





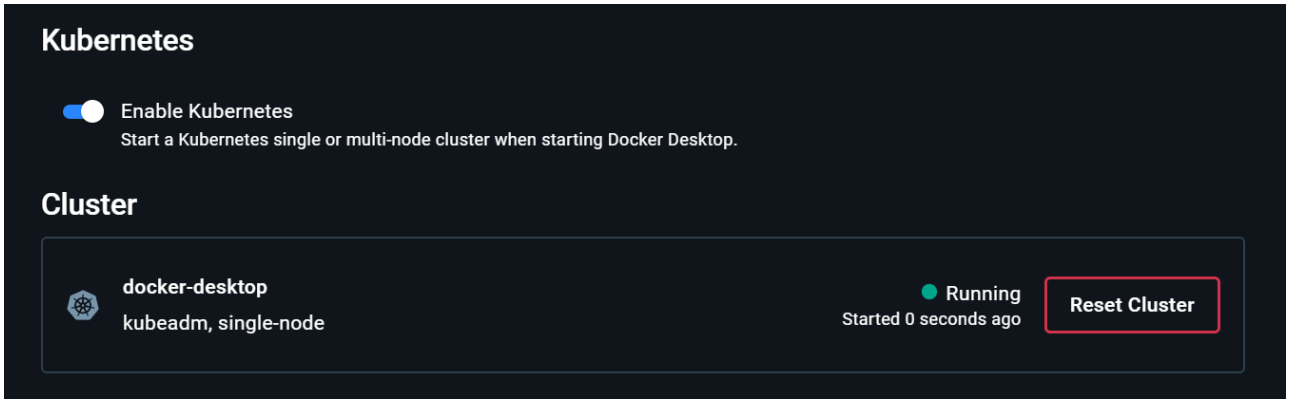
Day 1 Tasks

Get Kubernetes running using Docker Desktop

1. I ran `kubectl get service` on Git Bash and got this expected error

```
farah@Farah-laptop MINGW64 ~
$ kubectl get service
E0306 13:41:48.041425   62456 memcache.go:265] "Unhandled Error" err=<
        couldn't get current server API group list: <html><head><meta http-equiv
='refresh' content='1;url=/login?from=%2Fapi%3Ftimeout%3D32s'/><script id='redir
ect' data-redirect-url='/login?from=%2Fapi%3Ftimeout%3D32s' src='/static/dbffb37
1/scripts/redirect.js'></script></head><body style='background-color:white; colo
r:white;'>
        Authentication required
        <!--
        -->

        </body></html>
```

2. To solve this, I enabled Kubernetes in Docker Desktop



3. I re-ran `kubectl get service` and saw that Kubernetes was now running

```
farah@Farah-laptop MINGW64 ~
$ kubectl get service
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   15m
```

Create Nginx deployment only

1. Created a deployment file
2. Ran `kubectl apply -f nginx-deploy.yml` to create the deployment

```
farah@Farah-laptop MINGW64 ~/OneDrive - Sparta Global/Documents/Github/tech501-b
ench-udemy-pathways/Project 10/Kubernetes (main)
$ kubectl apply -f nginx-deploy.yml
deployment.apps/nginx-deployment created
```

3. Ran `kubectl get deployment nginx-deployment` to get details on the deployment

```
$ kubectl get deployment nginx-deployment
NAME               READY   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment   3/3     3            3           6m52s
```

4. Ran `kubectl get replicaset` to get details on the ReplicaSets

```
$ kubectl get replicaset
NAME                         DESIRED   CURRENT   READY   AGE
nginx-deployment-68d98fd8fc  3         3         3       7m33s
```

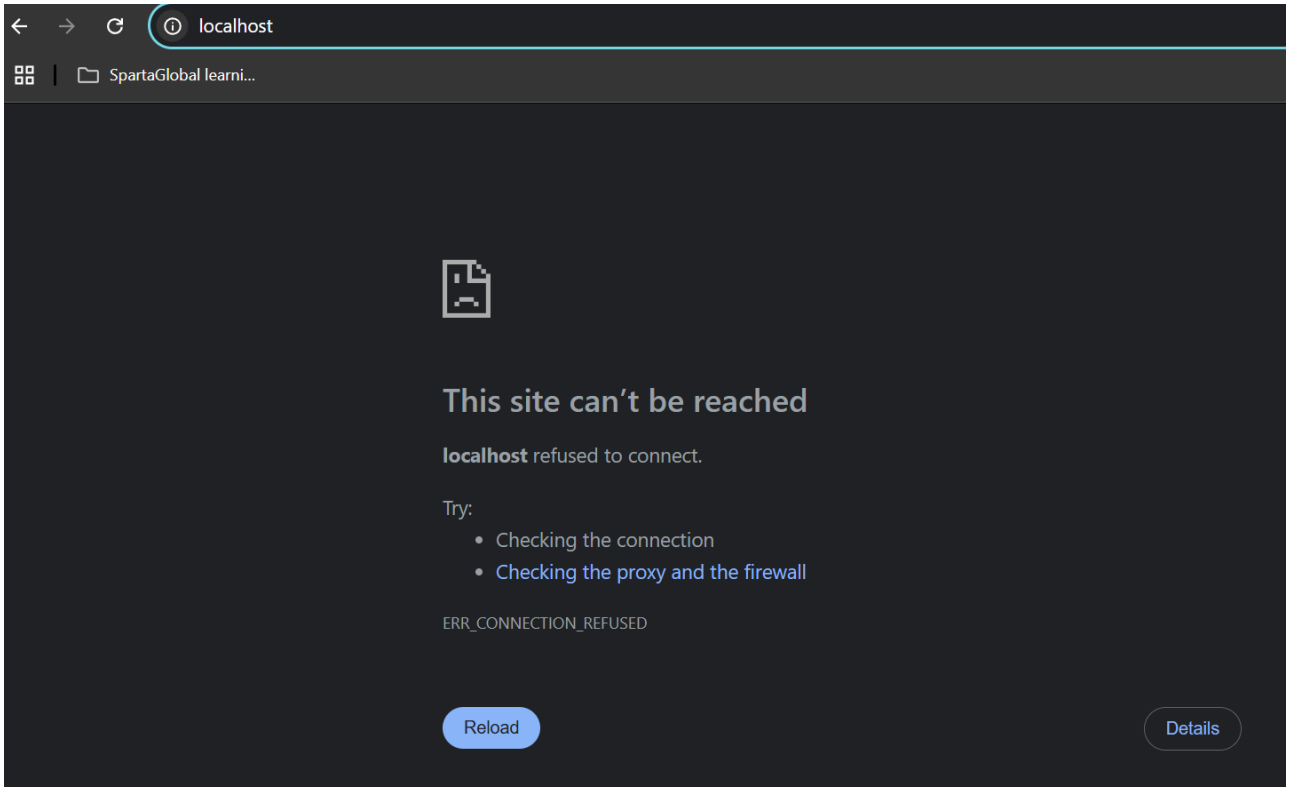
5. Ran `kubectl get pods` to get details on the pods

```
$ kubectl get pods
NAME                                    READY   STATUS    RESTARTS   AGE
nginx-deployment-68d98fd8fc-2j8ht       1/1     Running   0          9m15s
nginx-deployment-68d98fd8fc-2zmrm       1/1     Running   0          9m15s
nginx-deployment-68d98fd8fc-95jww       1/1     Running   0          9m15s
```

6. Ran `kubectl get all -l app=nginx` to see details on all three with one command

```
farah@Farah-laptop MINGW64 ~/OneDrive - Sparta Global/Documents/Github/tech501-b
ench-udemy-pathways/Project 10/k8s-yaml-definitions/local-nginx-deploy (main)
$ kubectl get all -l app=nginx
NAME                                    READY   STATUS    RESTARTS   AGE
pod/nginx-deployment-68d98fd8fc-m8vtw   1/1     Running   0          3s

NAME                               READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/nginx-deployment   1/1     1            1           3s

NAME                                          DESIRED   CURRENT   READY   AGE
replicaset.apps/nginx-deployment-68d98fd8fc   1         1         1       3s
```

7. Tried to access my deployment via my web browser at localhost but this didn't work (as expected), because the ClusterIP is a private IP address only accessible within the cluster



Get a NodePort service running

- We do this to expose the app deployment outside of the cluster, making it visible in a browser to the outside world

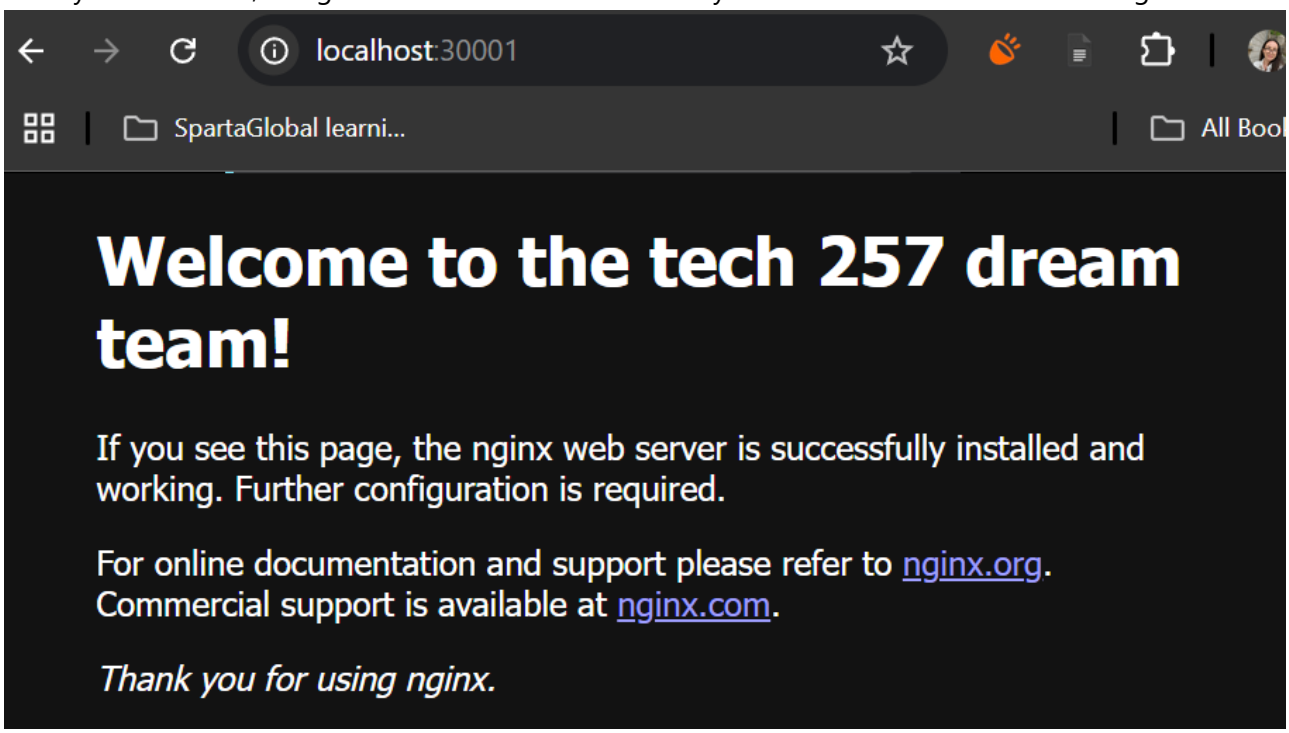
1. Created a service file
2. Created the service with `kubectl apply -f nginx-service.yml`

```
farah@Farah-laptop MINGW64 ~/OneDrive - Sparta Global/Documents/Github/tech501-b
ench-udemy-pathways/Project 10/k8s-yaml-definitions/local-nginx-deploy (main)
$ kubectl apply -f nginx-service.yml
service/nginx-svc created
```

3. Ran `kubectl get svc nginx-svc` to get details on the service

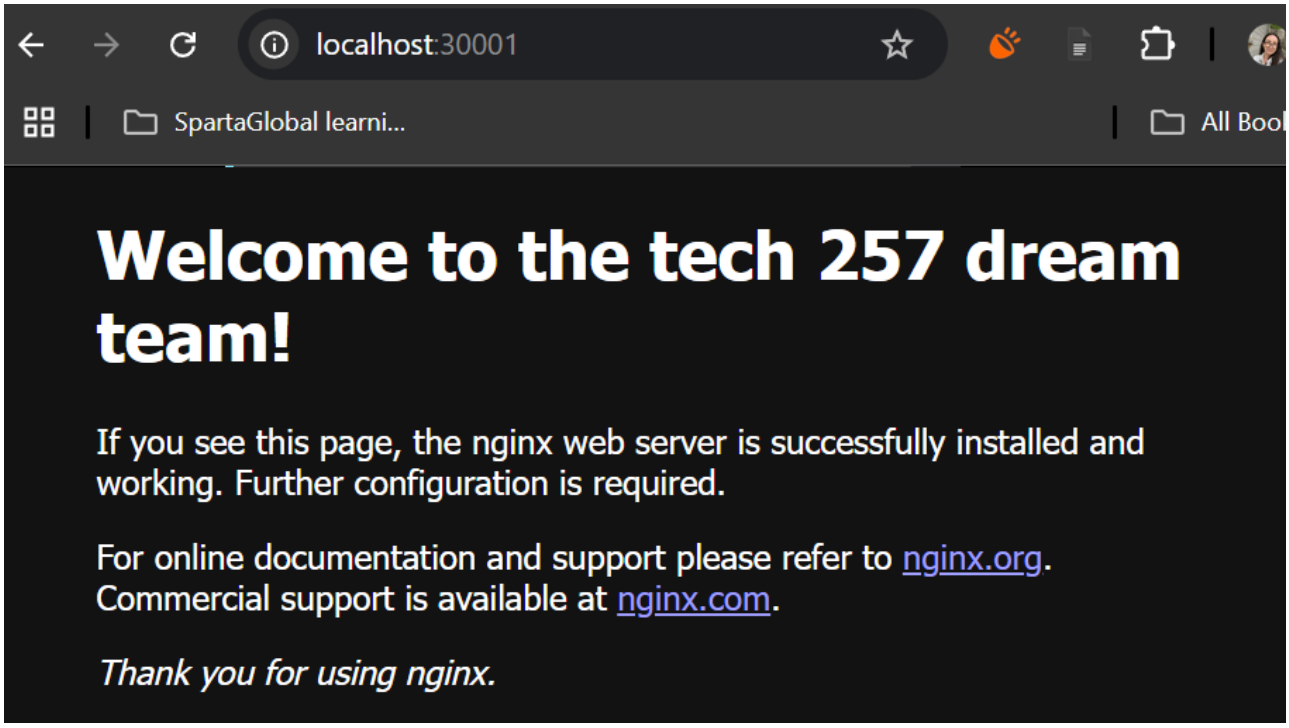
```
$ kubectl get svc nginx-svc
NAME        TYPE       CLUSTER-IP     EXTERNAL-IP   PORT(S)        AGE
nginx-svc   NodePort   10.104.86.19   <none>        80:30001/TCP   82s
```

4. On my web browser, navigated to *localhost:30001* to verify the NodePort service was running



See what happens when we delete a pod

1. Ran `kubectl get pods` to list the pods



2. Ran `kubectl delete pod nginx-deployment-68d98fd8fc-2j8ht` to delete one of the pods (in this case, the first pod) in the above list

```
$ kubectl delete pod nginx-deployment-68d98fd8fc-2j8ht
pod "nginx-deployment-68d98fd8fc-2j8ht" deleted
```

3. Re-ran `kubectl get pods` and saw that Kubernetes had automatically recreated a pod to satisfy the requirements in my deployment file

```
$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-68d98fd8fc-2zmrm   1/1     Running   0          24m
nginx-deployment-68d98fd8fc-4pbt2   1/1     Running   0          7s
nginx-deployment-68d98fd8fc-95jww   1/1     Running   0          24m
```

4. To get detailed information about the newest pod, I:
   1. Ran `kubectl get pods --sort-by=.metadata.creationTimestamp` to get a list of pods by their time of creation

```
$ kubectl get pods --sort-by=.metadata.creationTimestamp
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-68d98fd8fc-2zmrm   1/1     Running   0          25m
nginx-deployment-68d98fd8fc-95jww   1/1     Running   0          25m
nginx-deployment-68d98fd8fc-4pbt2   1/1     Running   0          90s
```

   2. Ran `kubectl describe pod nginx-deployment-68d98fd8fc-4pbt2` to get detailed information on the newest pod
   3. Then automated this process by running `kubectl describe pod $(kubectl get pods --sort-by=.metadata.creationTimestamp -o jsonpath='{.items[-1].metadata.name}')` to get details on the newest pod — this command uses the output of the parenthetical command as a variable, with `items[-1]` getting the last item in the list of pods ordered by their creation time (which runs from oldest-newest)

```
farah@Farah-laptop MINGW64 ~/OneDrive - Sparta Global/Documents/Github/tech501-b
ench-udemy-pathways/Project 10/k8s-yaml-definitions/local-nginx-deploy (main)
$ kubectl describe pod $(kubectl get pods --sort-by=.metadata.creationTimestamp
-o jsonpath='{.items[-1].metadata.name}')
Name:             nginx-deployment-68d98fd8fc-4pbt2
Namespace:        default
Priority:         0
Service Account:  default
Node:             docker-desktop/192.168.65.3
Start Time:       Fri, 07 Mar 2025 11:44:18 +0000
Labels:           app=nginx
                  pod-template-hash=68d98fd8fc
Annotations:      <none>
Status:           Running
IP:               10.1.0.9
IPs:
  IP:           10.1.0.9
Controlled By:  ReplicaSet/nginx-deployment-68d98fd8fc
Containers:
  nginx:
    Container ID:   docker://c4fd7a37e2e55d1b4c511a805f94aef8988eb6193820bdf85ec
06ffcaec37b5f
    Image:          daraymonsta/nginx-257:dreamteam
    Image ID:       docker-pullable://daraymonsta/nginx-257@sha256:aca9f1774d817
86850052224e68a247259ec8d1790d14694aea373feaf57c03f
    Port:           80/TCP
    Host Port:      0/TCP
    State:          Running
      Started:      Fri, 07 Mar 2025 11:44:19 +0000
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
```

Increase replicas with no downtime

- We want to be able to increase the number of replicas (pods) in our deployment in real-time, without needing to destroy and re-create our deployment

Method 1: editing the deployment file in real-time

1. I ran `kubectl edit deployment nginx-deployment` — this opened up an editable version of the in-use manifest file in Notepad, so I changed the number of replicas to 4, saved, and exited the file

```
kubectl.exe-edit-1438167428.yaml
File  Edit  View
,"ports":[{"containerPort":80}]}]}}}}
  creationTimestamp: "2025-03-07T11:20:16Z"
  generation: 1
  name: nginx-deployment
  namespace: default
  resourceVersion: "29999"
  uid: b692b3be-8ce2-4839-9c78-e1cdb1df8f24
spec:
  progressDeadlineSeconds: 600
  replicas: 4
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      app: nginx
```

2. I then ran `kubectl get pods` again and verified that there were now 4 pods running

```
$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-68d98fd8fc-2zmrm   1/1     Running   0          34m
nginx-deployment-68d98fd8fc-4pbt2   1/1     Running   0          10m
nginx-deployment-68d98fd8fc-95jww   1/1     Running   0          34m
nginx-deployment-68d98fd8fc-px42b   1/1     Running   0          8s
```

Method 2: Apply a modified deployment file

1. Edited the *nginx-deploy.yml* file in Nano to change the number of replicas to 5
2. Applied the updated file by running `kubectl apply -f nginx-deploy.yml` again
3. Ran `kubectl get pods` again to verify that there were now 5 pods running

```
$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-68d98fd8fc-2zmrm   1/1     Running   0          57m
nginx-deployment-68d98fd8fc-4pbt2   1/1     Running   0          33m
nginx-deployment-68d98fd8fc-95jww   1/1     Running   0          57m
nginx-deployment-68d98fd8fc-px42b   1/1     Running   0          22m
nginx-deployment-68d98fd8fc-w8f9w   1/1     Running   0          6s
```

Method 3: Use the scale command

- This is a quick way of scaling up/down, but isn't persistent (unlike editing the YAML files or using a HPA)

1. I ran `kubectl scale deployment nginx-deployment --replicas=6` to scale this deployment by 1 more replica

```
$ kubectl scale deployment nginx-deployment --replicas=6
deployment.apps/nginx-deployment scaled
```

2. Ran `kubectl get pods` again to verify that there were now 6 pods running

```
$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-68d98fd8fc-2zmrm   1/1     Running   0          58m
nginx-deployment-68d98fd8fc-4pbt2   1/1     Running   0          34m
nginx-deployment-68d98fd8fc-95jww   1/1     Running   0          58m
nginx-deployment-68d98fd8fc-p5kwr   1/1     Running   0          3s
nginx-deployment-68d98fd8fc-px42b   1/1     Running   0          23m
nginx-deployment-68d98fd8fc-w8f9w   1/1     Running   0          87s
```

Delete K8s deployments and services

1. Ran `kubectl delete -f nginx-deploy.yml` and `kubectl delete -f nginx-service.yml` to delete the Nginx deployment and service

```
$ kubectl delete -f nginx-deploy.yml
deployment.apps "nginx-deployment" deleted

farah@Farah-laptop MINGW64 ~/OneDrive - Spar
ench-udemy-pathways/Project 10/k8s-yaml-def
$ kubectl delete -f nginx-service.yml
service "nginx-svc" deleted
```

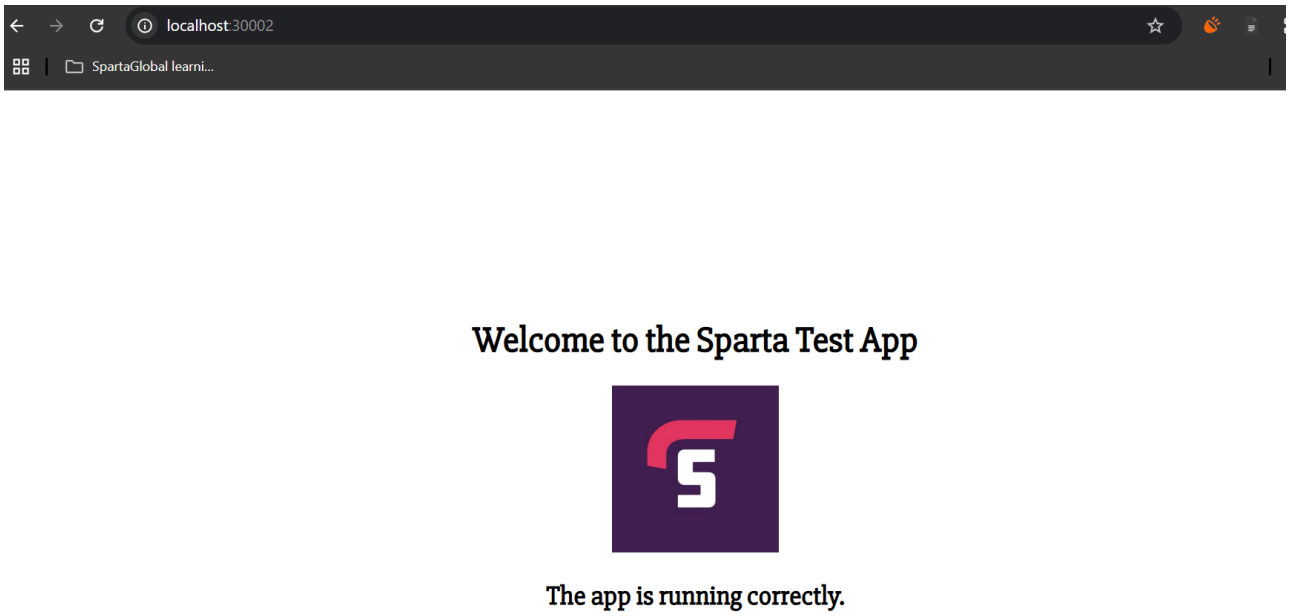
2. Ran `kubectl get all` to verify that they were deleted

```
$ kubectl get all
NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   19h
```

K8s deployment of NodeJS Sparta test app

architecture diagram

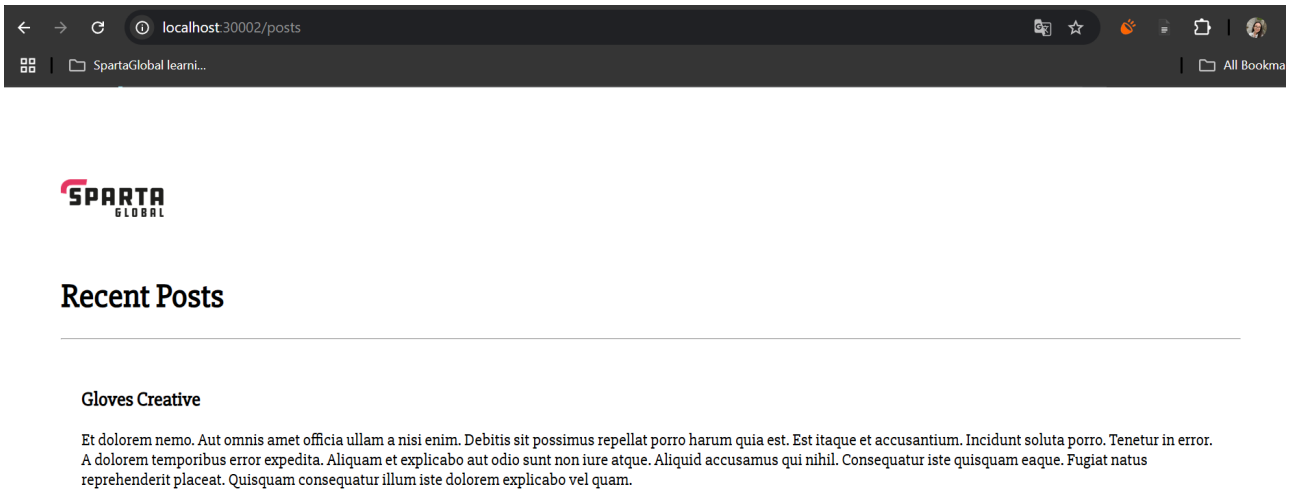
1. Created my *app-deploy.yml* and *app-service.yml* manifest files
2. Ran`kubectl apply -f app-deploy.yml` and `kubectl apply -f app-service.yml` to create the resources
3. Navigated to *localhost:30002* (the port specified in my *app-service.yml* file) to verify that this was successful



4. Created my *mongodb-service.yml* and *mongodb-deploy.yml* files
5. Ran `kubectl apply -f mongodb-deploy.yml` and `kubectl apply -f mongodb-service.yml` to create the resources

```
$ kubectl apply -f mongodb-deploy.yml
deployment.apps/mongodb-deployment created

farah@Farah-laptop MINGW64 ~/OneDrive - Spa
ench-udemy-pathways/Project 10/k8s-yaml-def
cal-mongodb-deploy (main)
$ kubectl apply -f mongodb-service.yml
service/mongodb-svc created
```

6. Navigated to *localhost:30002/posts* to verify that the deployment and database connection was successful



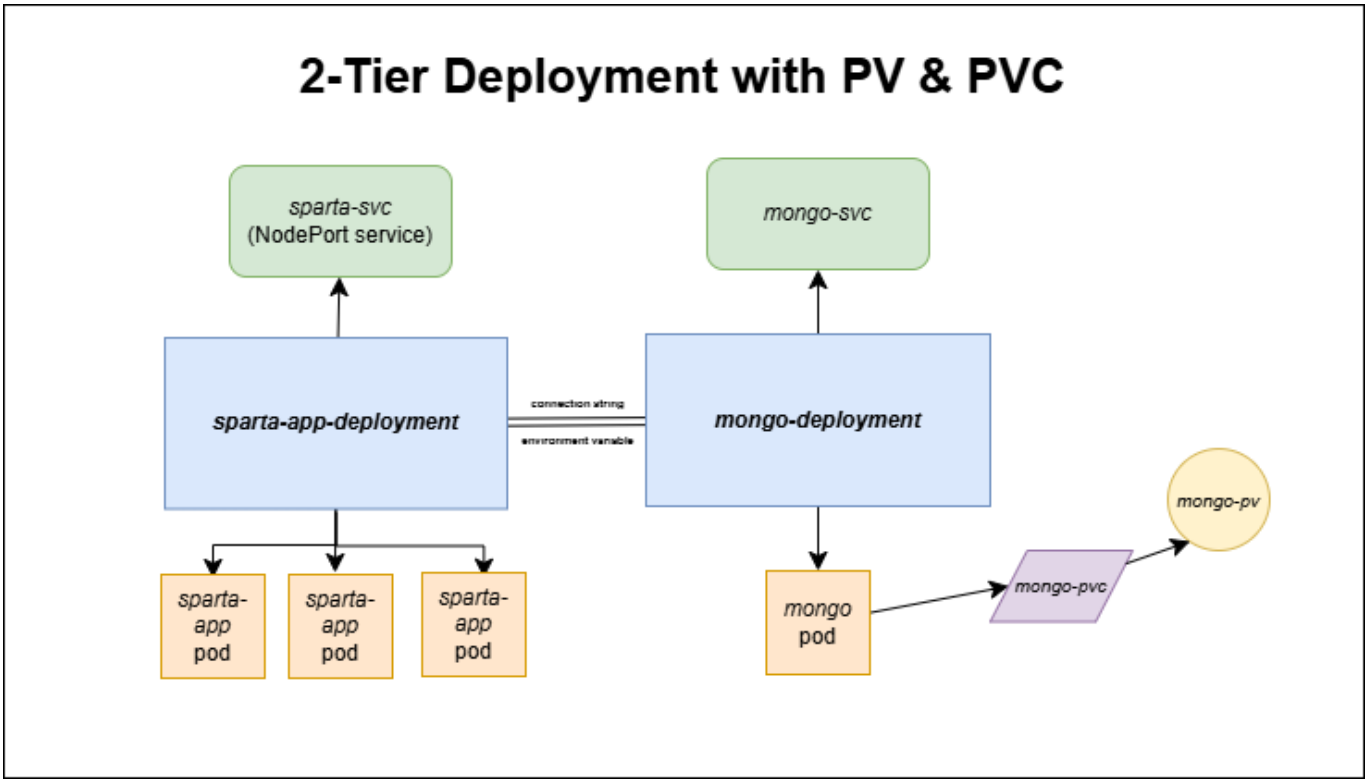
Blockers

1. I had issues enabling Kubernetes with Docker Desktop as it hung on "Starting Kubernetes" for a long time; this was solved by manually installing Kubernetes via Chocolatey and then enabling Kubernetes again via Docker Desktop

Day 2 Tasks

Create 2-tier deployment with a PersistentVolume (PV) for the database

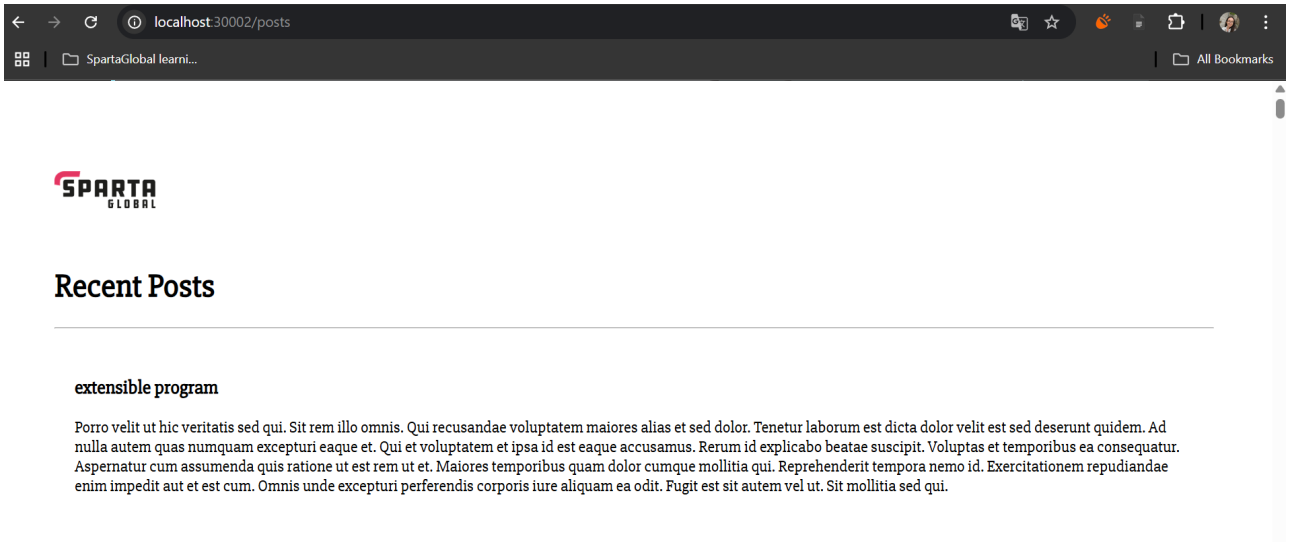
- using a PersistentVolume ensures the data stored on the PV will be retained even if a pod restarts
- a PersistentVolumeClaim (PVC) requests a certain amount of storage from the PV for a resource



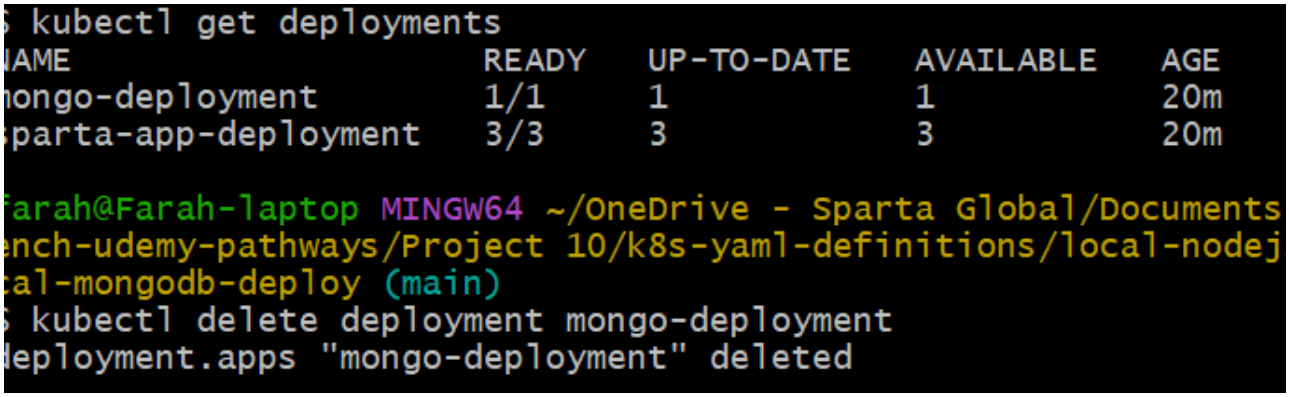
1. Created a *mongo-pv.yml* and a *mongo-pvc.yml* file (both found here), with 100Mb in both the PV and the claim (since I only have one MongoDB pod)
2. Created the PV with `kubectl apply -f mongo-pv.yml`
3. Created the PVC with `kubectl apply -f mongo-pv.yml`
4. Verified they were both created successfully

```
$ kubectl get pv
NAME        CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM   STORAG
ECLASS   VOLUMEATTRIBUTESCLASS   REASON   AGE
mongo-pv    100Mi      RWO            Retain           Available
         <unset>                          5m34s

farah@Farah-laptop MINGW64 ~/OneDrive - Sparta Global/Documents/Github/tech501-b
ench-udemy-pathways/Project 10/k8s-yaml-definitions/local-nodejs20-app-deploy/lo
cal-mongodb-deploy (main)
$ kubectl get pvc
NAME        STATUS    VOLUME     CAPACITY   ACCESS MODES   STORAGECLASS   VOLUME
ATTRIBUTESCLASS   AGE
mongo-pvc   Pending   mongo-pv   0                         hostpath       <unset
>                 44s
```

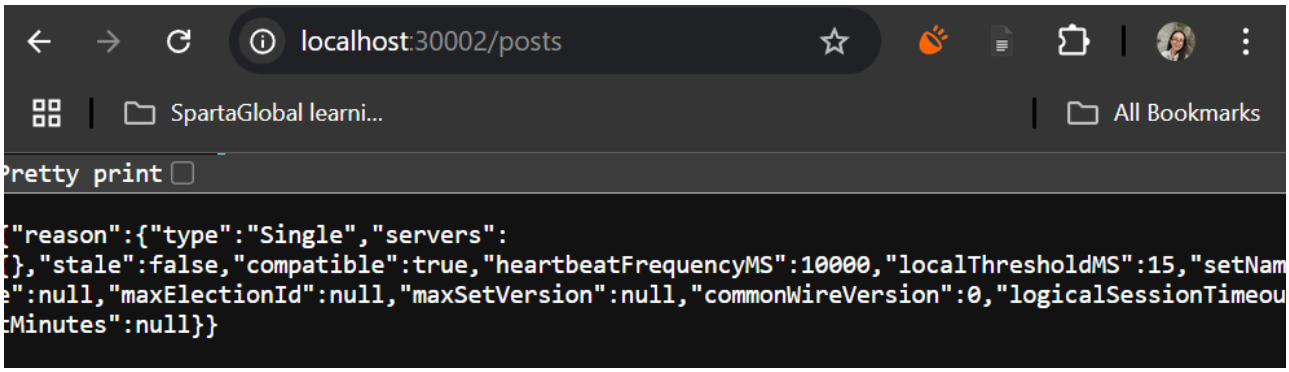
- For reference, this is what my *localhost:30002/posts* page looked like at this point



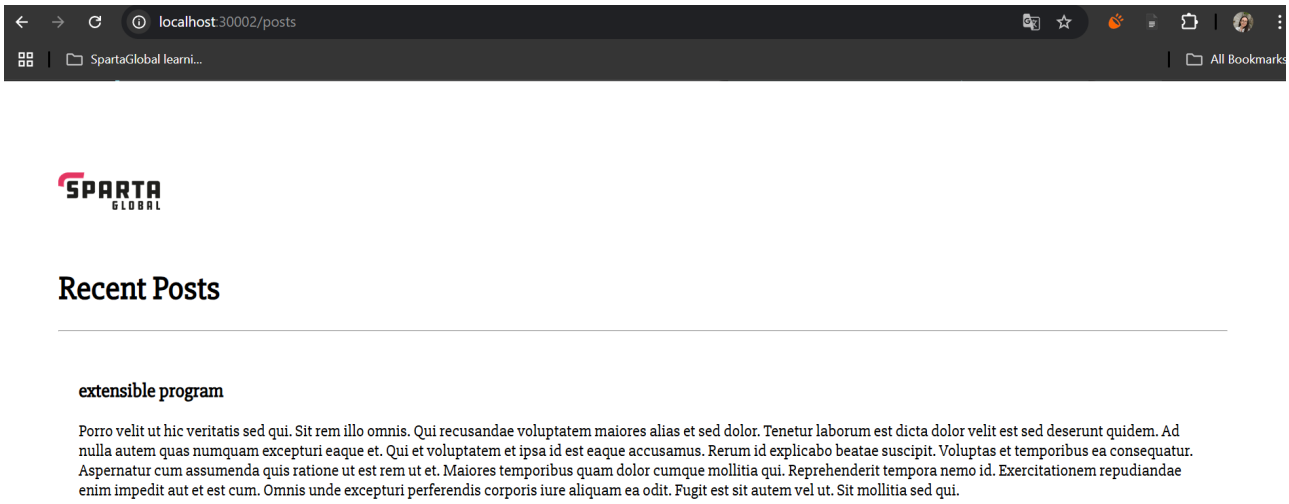
1. Deleted my database deployment with `kubectl delete deployment mongo-deployment`



2. Verified that my *posts* page no longer worked



3. Recreated the above deployment with `kubectl apply -f mongo-deploy.yml` and visited *localhost:30002/posts* again to verify that the records were still the same



Use Horizontal Pod Autoscaler (HPA) to scale the app

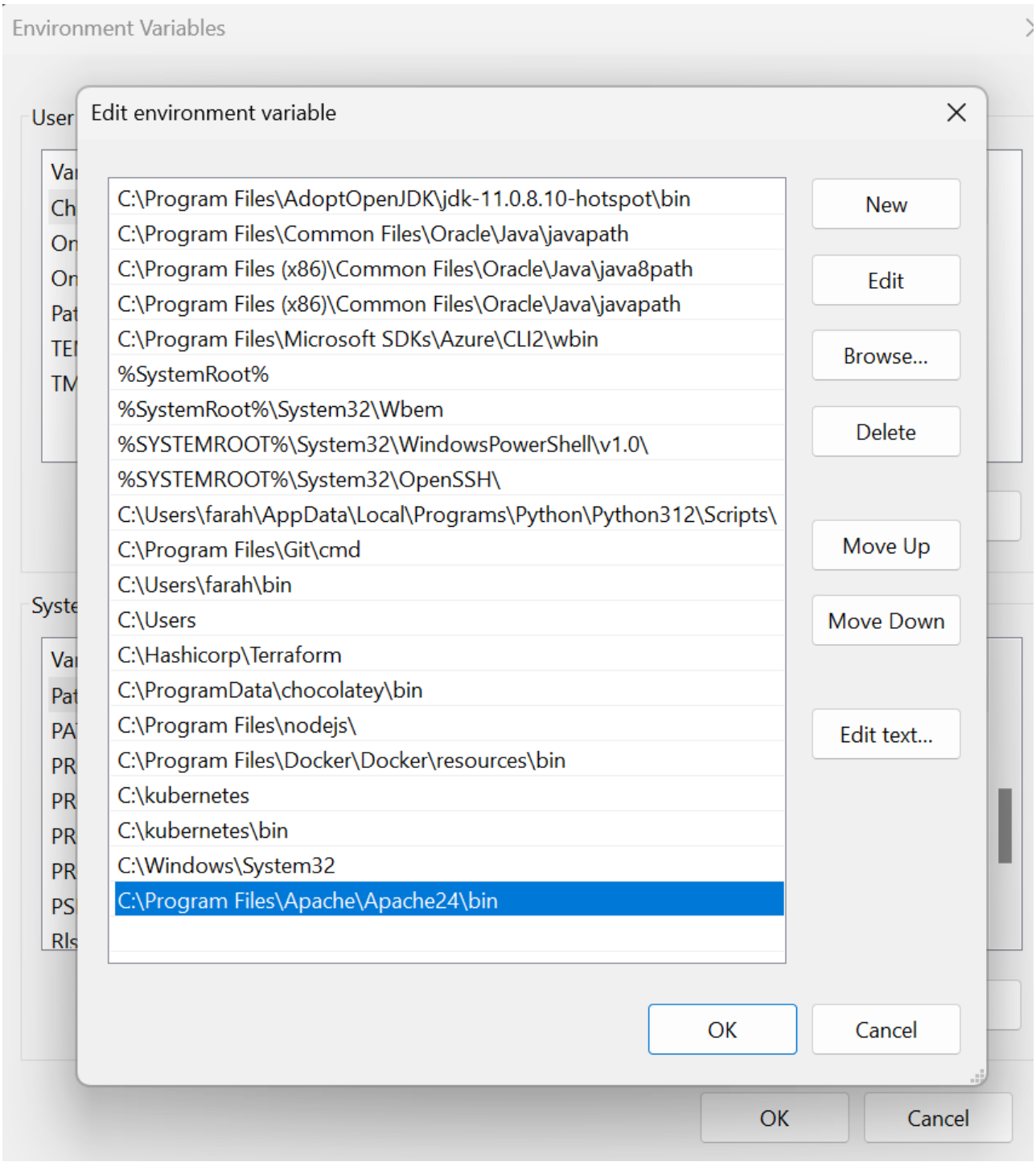
1. Installed the Metrics Server add-on (a requirement for HPAs) with `kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml`
2. Created a HPA with one command `kubectl autoscale deployment sparta-app-deployment --cpu-percent=5 --min=2 --max=10` (note that this can also be done via a YAML file, which I included here for illustration purposes)

```
farah@Farah-laptop MINGW64 ~/OneDrive - Sparta Global/Documents/Github/tech501-b
ench-udemy-pathways/Project 10/k8s-yaml-definitions (main)
$ kubectl autoscale deployment sparta-app-deployment --cpu-percent=5 --min=2 --m
ax=10
horizontalpodautoscaler.autoscaling/sparta-app-deployment autoscaled
```

3. Verified the HPA was running with `kubectl get hpa`

```
farah@Farah-laptop MINGW64 ~/OneDrive - Sparta Global/Documents/Github/tech501-b
ench-udemy-pathways/Project 10/k8s-yaml-definitions/local-nodejs20-app-deploy (m
ain)
$ kubectl get hpa
NAME                    REFERENCE                        TARGETS       MINPODS
  MAXPODS   REPLICAS   AGE
sparta-app-deployment   Deployment/sparta-app-deployment   cpu: 1%/5%   2
  10        2          140m
```

4. Installed Apache Benchmark (a load-testing tool) on Windows by downloading *Apache 2.4.63-250207 Win64* from Apache Lounge and adding it to the PATH environment variable on my local machine



- I verified that Apache Benchmark was installed with `ab -V` !

```
farah@Farah-laptop MINGW64 ~
$ ab -V
This is ApacheBench, Version 2.3 <$Revision: 1923142 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/
```

5. Ran this command to send 10000 requests (1000 at a time) to my app `ab -n 10000 -c 1000 http://localhost:30002/`
6. Watched the CPU activity with `kubectl get hpa -w`

```
$ kubectl get hpa -w
NAME                    REFERENCE                          TARGETS        MINPODS
   MAXPODS   REPLICAS   AGE
sparta-app-deployment   Deployment/sparta-app-deployment   cpu: 49%/5%    2
   10        2          141m
```

7. Verified that new pods were being created to meet the increased demand with with `kubectl get pods`

```
$ kubectl get pods
NAME                                    READY   STATUS    RESTARTS      AGE
mongo-deployment-747dcd8f54-5gqw7       1/1     Running   3 (13m ago)   2d20h
sparta-app-deployment-59f9f97f7b-4v8b7  1/1     Running   0             2m7s
sparta-app-deployment-59f9f97f7b-cbptj  1/1     Running   0             67s
sparta-app-deployment-59f9f97f7b-gvrsv  1/1     Running   0             6m39s
sparta-app-deployment-59f9f97f7b-hxsvw  1/1     Running   0             2m7s
sparta-app-deployment-59f9f97f7b-m9lz8  1/1     Running   0             67s
sparta-app-deployment-59f9f97f7b-n646v  1/1     Running   0             2m7s
sparta-app-deployment-59f9f97f7b-ndpb2  1/1     Running   0             6m36s
sparta-app-deployment-59f9f97f7b-npjdr  1/1     Running   0             2m7s
sparta-app-deployment-59f9f97f7b-smqcl  1/1     Running   0             3m7s
sparta-app-deployment-59f9f97f7b-sxrgz  1/1     Running   0             3m7s
```

   - and also with `kubectl describe deployment sparta-app-deployment` (note the instances of "Scaled up")

```
  Normal  ScalingReplicaSet  3m43s                deployment-controller  Scaled
up replica set sparta-app-deployment-65978f45cd to 1
  Normal  ScalingReplicaSet  3m39s                deployment-controller  Scaled
down replica set sparta-app-deployment-59f9f97f7b to 1 from 2
  Normal  ScalingReplicaSet  3m39s                deployment-controller  Scaled
up replica set sparta-app-deployment-65978f45cd to 2 from 1
  Normal  ScalingReplicaSet  3m36s                deployment-controller  Scaled
down replica set sparta-app-deployment-59f9f97f7b to 0 from 1
  Normal  ScalingReplicaSet  3m8s                 deployment-controller  Scaled
up replica set sparta-app-deployment-65978f45cd to 4 from 2
  Normal  ScalingReplicaSet  2m8s                 deployment-controller  Scaled
up replica set sparta-app-deployment-65978f45cd to 8 from 4
  Normal  ScalingReplicaSet  68s                  deployment-controller  Scaled
up replica set sparta-app-deployment-65978f45cd to 10 from 8
```

8. I waited a little while for the HPA's cooldown period to be over so that the HPA started scaling down again

```
farah@Farah-laptop MINGW64 ~/OneDrive - Sparta Global/Documents/Github/tech501-b
ench-udemy-pathways/Project 10/k8s-yaml-definitions/local-nodejs20-app-deploy (m
ain)
$ kubectl get pods -w
NAME                                    READY   STATUS        RESTARTS      AGE
mongo-deployment-747dcd8f54-5gqw7       1/1     Running       3 (15m ago)   2d2
0h
sparta-app-deployment-59f9f97f7b-4v8b7  1/1     Terminating   0             4m8
s
sparta-app-deployment-59f9f97f7b-cbptj  1/1     Terminating   0             3m8
s
sparta-app-deployment-59f9f97f7b-gvrsv  1/1     Running       0             8m4
0s
sparta-app-deployment-59f9f97f7b-hxsvw  1/1     Terminating   0             4m8
s
sparta-app-deployment-59f9f97f7b-m9lz8  1/1     Terminating   0             3m8
s
sparta-app-deployment-59f9f97f7b-n646v  1/1     Terminating   0             4m8
s
sparta-app-deployment-59f9f97f7b-ndpb2  1/1     Running       0             8m3
7s
sparta-app-deployment-59f9f97f7b-npjdr  1/1     Terminating   0             4m8
s
sparta-app-deployment-59f9f97f7b-smqcl  1/1     Running       0             5m8
```

9. I also verified this with `kubectl describe deployment sparta-app-deployment`

```
  Normal  ScalingReplicaSet  89s (x10 over 12m)  deployment-controller  (combine
d from similar events): Scaled down replica set sparta-app-deployment-59f9f97f7b
 to 2 from 4
```

   - Note that HPA minimums take precedence over the number of replicas defined in my *app-deploy.yml* file, which is why I now only have 2 app pods

```
farah@Farah-laptop MINGW64 ~/OneDrive - Sparta Global/Documents/Github/tech501-b
ench-udemy-pathways/Project 10/k8s-yaml-definitions/local-nodejs20-app-deploy (m
ain)
$ kubectl get pods
NAME                                    READY   STATUS    RESTARTS      AGE
mongo-deployment-747dcd8f54-5gqw7       1/1     Running   3 (19m ago)   2d20h
sparta-app-deployment-59f9f97f7b-gvrsv  1/1     Running   0             12m
sparta-app-deployment-59f9f97f7b-ndpb2  1/1     Running   0             12m
```

Blockers

1. I had blockers getting my Metrics Server and HPA to work, because it would say `unknown/5%` (which was the CPU metric I set); I got around this by adding a resource block to my *app-deploy.yml*, deleting the pods, and then trying again

Day 3 Tasks

Setup Minikube on a cloud instance running Ubuntu 22.04 LTS

1. I created a *t3a.small* EC2 using the Ubuntu 22.04 LTS OS named *tech501-farah-kubernetes-minikube-ec2*, with inbound network access allowed on port 9000 from anywhere

Name	Instance ID	Instance state	Instance type
tech501-farah-kubernetes-minikube-ec2	i-000a0ead7af2779d4	Running	t3a.small

2. I logged into the EC2 via SSH and ran `sudo apt update && sudo apt upgrade -y`
3. Installed Nginx with `sudo apt install nginx -y`
4. Installed and configured Docker (a dependency):

```
sudo apt install -y apt-transport-https curl virtualbox docker.io
sudo systemctl enable docker
sudo systemctl restart docker
```

5. Installed and configured Minikube

```
sudo curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-
linux-amd64
sudo mv minikube-linux-amd64 /usr/local/bin/minikube
sudo chmod +x /usr/local/bin/minikube
```

6. Verified Minikube was installed with `minikube version`

```
ubuntu@ip-172-31-50-112:~$ minikube version
minikube version: v1.35.0
commit: dd5d320e41b5451cdf3c01891bc4e13d189586ed-dirty
```

7. Added myself to the Docker group with `sudo usermod -aG docker $USER && newgrp docker` so that I could run the next command
8. Installed `kubectl` following these steps

```
  sudo curl -LO "https://dl.k8s.io/release/$(curl -L -s
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
  sudo curl -LO "https://dl.k8s.io/release/$(curl -L -s
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl.sha256"
  echo "$(cat kubectl.sha256)  kubectl" | sha256sum --check
  sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl
```

9. Installed Metrics server with `kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml`

Deploy three apps on one cloud instance running Minikube

First app

1. Started Minikube with `minikube start` (at this point, I would have to do this whenever I rebooted the EC2 instance)
2. Created manifest files for an Nginx deployment (with 5 replicas, using the *daraymonsta/nginx-257:dreamteam* image) and NodePort service using port 30001 (both found here)
3. Created the above resources with `minikube kubectl -- apply -f first-deployment.yml` and `minikube kubectl -- apply -f first-service.yml`

```
ubuntu@ip-172-31-50-112:~/project10/minikube-task$ minikube kubectl -- apply -f
first-deployment.yml
deployment.apps/nginx-deployment created
ubuntu@ip-172-31-50-112:~/project10/minikube-task$ minikube kubectl -- apply -f
first-service.yml
service/nginx-app-service created
```

4. Noted Minikube's IP address with `minikube ip`

```
ubuntu@ip-172-31-50-112:~/project10/minikube-task$ minikube ip
192.168.49.2
```

5. Replaced the *try_files* line with a *proxy_pass* line in Nginx's *sites-available/default* file so it looked like this

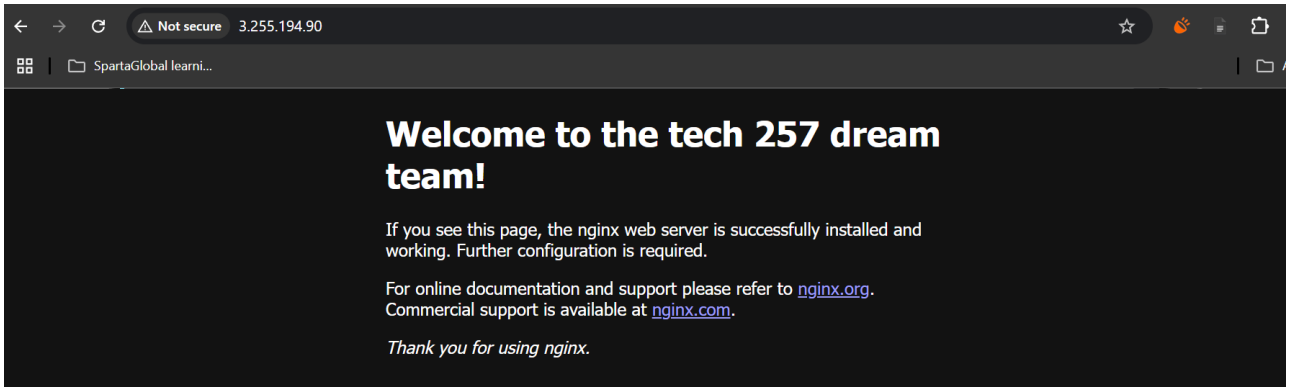
```
ubuntu@ip-172-31-50-112: /etc/nginx/sites-available
GNU nano 6.2                    default

        root /var/www/html;

        # Add index.php to the list if you are using PHP
        index index.html index.htm index.nginx-debian.html;

        server_name _;

        location / {
                # First attempt to serve request as file, then
                # as directory, then fall back to displaying a 404.
                proxy_pass http://192.168.49.2:30001;
        }
```

6. Verified the syntax of the above file was okay with `sudo nginx -t`

```
ubuntu@ip-172-31-50-112:/etc/nginx/sites-available$ sudo nginx -t
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
```

7. Reloaded Nginx with `sudo systemctl reload nginx`

8. In my browser, I navigated to this EC2's public IP and verified that the app was running



Second app

1. Created the *second-deployment.yml* manifest file for a deployment (with 2 replicas, using the *daraymonsta/tech201-nginx-auto:v1* image)
2. Created these resources with `minikube kubectl -- apply -f second-deployment.yml`

```
ubuntu@ip-172-31-50-112:~/project10/minikube-task/second-app$ minikube kubectl -
- apply -f second-deployment.yml
```

3. Created a *second-service.yml* manifest file for a LoadBalancer service using port 30002
4. Created this service with `minikube kubectl -- apply -f second-service.yml`

```
ubuntu@ip-172-31-50-112:~/project10/minikube-task/second-app$ minikube kubectl -
- apply -f second-service.yml
service/nginx-tech201-app-service created
```

5. Added a new *server* block to Nginx's *default* file to create a reverse proxy for this app

```
server {
    listen 9000;
    server_name _;

    location / {
        proxy_pass http://192.168.49.2:30002;
    }
}
```

6. Verified the syntax was okay with `sudo nginx -t`

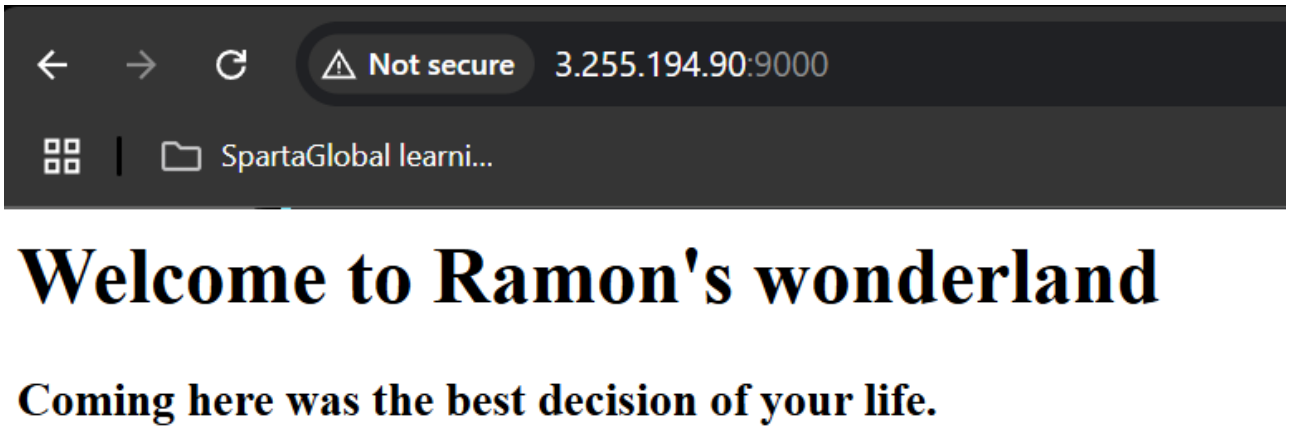
```
ubuntu@ip-172-31-50-112:~/project10/minikube-task/second-app$ sudo nginx -t
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
```

7. Reloaded Nginx with `sudo systemctl reload nginx`
8. In a separate terminal window, I ran `minikube tunnel` and kept this process running throughout this task and the next

```
ubuntu@ip-172-31-50-112:~/project10/minikube-task/second-app$ minikube tunnel
Status:
        machine: minikube
        pid: 33756
        route: 10.96.0.0/12 -> 192.168.49.2
        minikube: Running
        services: [nginx-tech201-app-service]
    errors:
                minikube: no errors
                router: no errors
                loadbalancer emulator: no errors
```

- This is to create a tunnel between the EC2 and the service that is running in the Minikube cluster, which allows traffic external to the Minikube cluster to reach the NodePort via the LoadBalancer service

9. Verified this process was successful by navigating to <EC2's publici IP>:9000/ in a browser



Third app

1. Ran `sudo snap install kubectl --classic` to enable me to install `kubectl` and follow the steps on the hello-minikube tutorial

```
ubuntu@ip-172-31-50-112:/etc/nginx/sites-available$ sudo snap install kubectl --
classic
kubectl 1.32.2 from Canonical✓ installed
```

2. Ran `kubectl create deployment hello-node --image=registry.k8s.io/e2e-test-images/agnhost:2.39 -- /agnhost netexec --http-port=8080`

```
ubuntu@ip-172-31-50-112:/etc/nginx/sites-available$ kubectl create deployment he
llo-node --image=registry.k8s.io/e2e-test-images/agnhost:2.39 -- /agnhost netexe
c --http-port=8080
deployment.apps/hello-node created
```

3. Verified that the deployment was created with `kubectl get deployments`

```
ubuntu@ip-172-31-50-112:/etc/nginx/sites-available$ kubectl get deployments
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
hello-node          1/1     1            1           51s
nginx-deployment    5/5     5            5           39m
second-deployment   2/2     2            2           31m
```

4. Verified that the pod had been created with `kubectl get pods`

```
ubuntu@ip-172-31-50-112:/etc/nginx/sites-available$ kubectl get pods
NAME                                READY   STATUS    RESTARTS        AGE
hello-node-c74958b5d-wwdsb          1/1     Running   0               82s
nginx-deployment-5bc95b6d5-6x9xk    1/1     Running   2 (4m25s ago)   39m
nginx-deployment-5bc95b6d5-jjzlc    1/1     Running   2 (4m25s ago)   39m
nginx-deployment-5bc95b6d5-lzwcm    1/1     Running   2 (4m25s ago)   39m
nginx-deployment-5bc95b6d5-tmxht    1/1     Running   2 (4m25s ago)   39m
nginx-deployment-5bc95b6d5-xvqth    1/1     Running   2 (4m25s ago)   39m
second-deployment-5bc56d8d59-fjksg  1/1     Running   2 (4m25s ago)   31m
second-deployment-5bc56d8d59-wnwvp  1/1     Running   2 (4m25s ago)   31m
```

5. Exposed the pod to port 8080 with a LoadBalancer service via `kubectl expose deployment hello-node --type=LoadBalancer --port=8080`

```
ubuntu@ip-172-31-50-112:/etc/nginx/sites-available$ kubectl expose deployment h
llo-node --type=LoadBalancer --port=8080
service/hello-node exposed
```

6. Verified that the service had been created with `kubectl get services`

```
ubuntu@ip-172-31-50-112:/etc/nginx/sites-available$ kubectl get services
NAME                        TYPE           CLUSTER-IP       EXTERNAL-IP     PORT
(S)          AGE
hello-node                  LoadBalancer   10.98.44.179     <pending>       8080
:32194/TCP   28s
kubernetes                  ClusterIP      10.96.0.1        <none>          443/
TCP          51m
nginx-app-service           NodePort       10.107.245.208   <none>          80:3
0001/TCP     40m
nginx-tech201-app-service   LoadBalancer   10.99.226.151    10.99.226.151   9000
:30002/TCP   31m
```

7. Ran `minikube service hello-node`

```
ubuntu@ip-172-31-50-112:~$ minikube service hello-node
|-----------|------------|-------------|---------------------------|
| NAMESPACE |    NAME    | TARGET PORT |            URL            |
|-----------|------------|-------------|---------------------------|
| default   | hello-node |        8080 | http://192.168.49.2:31654 |
|-----------|------------|-------------|---------------------------|
```

(normally this would open a browser window to serve the app, but it didn't because I don't have a browser installed on this VM)

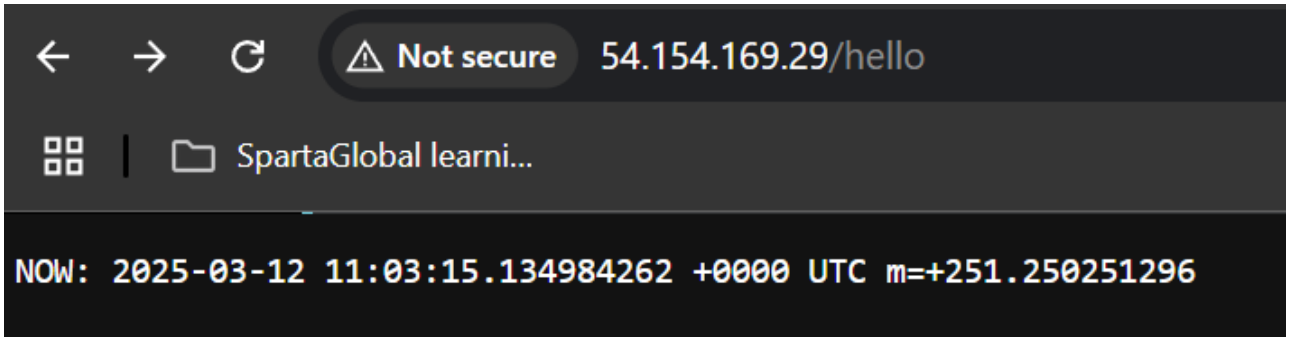
8. Ran a `curl` command to verify that the app was running on the URL in above command's output

```
ubuntu@ip-172-31-50-112:~$ curl http://192.168.49.2:31654
NOW: 2025-03-11 09:42:45.207094641 +0000 UTC m=+2407.996616240ubuntu@ip-172-31-5
0-112:~$
```

9. Edited my Nginx *default* file to add a new `location` block under the server listening on port 80 (which contained the reverse proxy for the first app; see here for the full file)

```
server_name _;

location / {
        # First attempt to serve request as file, then
        # as directory, then fall back to displaying a 404.
        proxy_pass http://192.168.49.2:30001;
}

location /hello {
        proxy_pass http://192.168.49.2:31654;
}
```

10. Ran `sudo nginx -t` and `sudo systemctl restart nginx` and verified that this was working on the browser



11. After this, I cleaned up all the resources by running `kubectl delete all --all --namespace default`

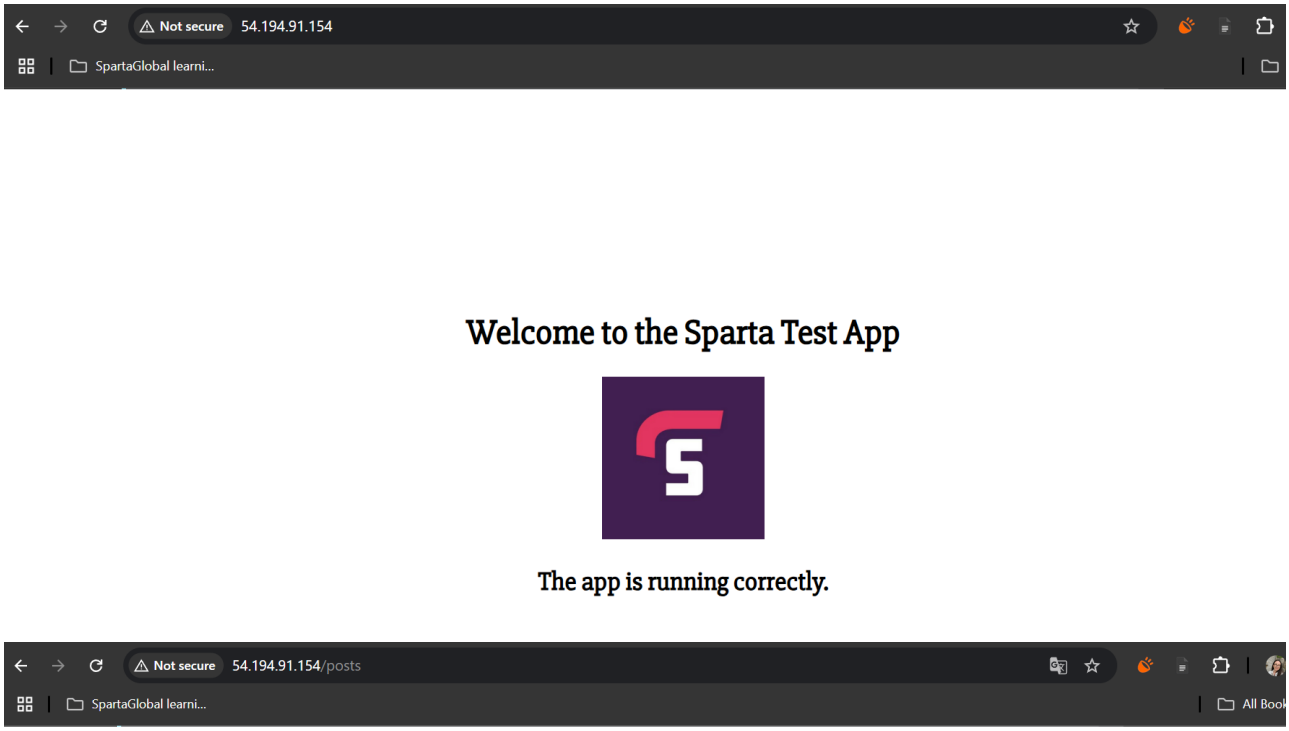
Use Kubernetes to deploy the Sparta test app in the cloud

- For this task, I created a new EC2 named *tech501-farah-kubernetes-2tier-minikube-ec2*

1. Created the deployment manifest files (the exact same files that were used in the Day 2 2-tier deployment task)
2. Ran `kubectl apply -f` on all of the above files
3. Configured the Nginx reverse proxy using Minikube's IP

```
ubuntu@ip-172-31-58-36: /etc/nginx/sites-available
  GNU nano 6.2
#
server {
        listen 80 default_server;
        listen [::]:80 default_server;

        # SSL configuration
        #
        # listen 443 ssl default_server;
        # listen [::]:443 ssl default_server;
        #
        # Note: You should disable gzip for SSL traffic.
        # See: https://bugs.debian.org/773332
        #
        # Read up on ssl_ciphers to ensure a secure configuration.
        # See: https://bugs.debian.org/765782
        #
        # Self signed certs generated by the ssl-cert package
        # Don't use them in a production server!
        #
        # include snippets/snakeoil.conf;

        root /var/www/html;

        # Add index.php to the list if you are using PHP
        index index.html index.htm index.nginx-debian.html;

        server_name _;

        location / {
                # First attempt to serve request as file, then
                # as directory, then fall back to displaying a 404.
                proxy_pass http://192.168.49.2:30002;
```

4. Verified that the app and the */posts* page were working on the browser



5. Created a HPA with `kubectl autoscale deployment sparta-app-deployment --cpu-percent=5 --min=2 --max=10`

```
ubuntu@ip-172-31-58-36:/etc/nginx/sites-available$ kubectl autoscale deployment
sparta-app-deployment --cpu-percent=5 --min=2 --max=10
horizontalpodautoscaler.autoscaling/sparta-app-deployment autoscaled
```

6. Verified it was running with `kubectl get hpa`

```
ubuntu@ip-172-31-58-36:/etc/nginx/sites-available$ kubectl get hpa
NAME                    REFERENCE                         TARGETS              M
INPODS   MAXPODS   REPLICAS   AGE
sparta-app-deployment   Deployment/sparta-app-deployment   cpu: <unknown>/5%    2
         10        3          48s
```

7. Installed Apache Benchmark with `sudo apt install apache2-utils`
   - I then verified that it was installed with `ab -V`

```
ubuntu@ip-172-31-58-36:/etc/nginx/sites-available$ ab -V
This is ApacheBench, Version 2.3 <$Revision: 1879490 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/
```

8. Ran a load-testing command `ab -n 6000 -c 100 http://<EC2's public IP>/`

9. Watched the CPU activity with `kubectl get hpa -w`

```
ubuntu@ip-172-31-58-36:~/2-tier-deployment$ kubectl get hpa -w
NAME                    REFERENCE                          TARGETS       MINPODS
  MAXPODS   REPLICAS   AGE
sparta-app-deployment   Deployment/sparta-app-deployment   cpu: 1%/5%    2
  10        2          22m
sparta-app-deployment   Deployment/sparta-app-deployment   cpu: 175%/5%   2
    10        2          23m
```

10. Verified that new pods were being created with `kubectl get pods`

```
ubuntu@ip-172-31-58-36:~$ kubectl get pods
NAME                                    READY   STATUS    RESTARTS   AGE
mongo-deployment-79c96444db-g97gj       1/1     Running   0          108m
sparta-app-deployment-7956fc8954-7vwgx  1/1     Running   0          117s
sparta-app-deployment-7956fc8954-f5768  1/1     Running   0          2m27s
sparta-app-deployment-7956fc8954-fq9cz  1/1     Running   0          14m
sparta-app-deployment-7956fc8954-jhs8j  1/1     Running   0          117s
sparta-app-deployment-7956fc8954-jxvsj  1/1     Running   0          2m27s
sparta-app-deployment-7956fc8954-lbd7x  1/1     Running   0          14m
sparta-app-deployment-7956fc8954-p4pph  1/1     Running   0          2m12s
sparta-app-deployment-7956fc8954-wgzcv  1/1     Running   0          2m12s
sparta-app-deployment-7956fc8954-wm67w  1/1     Running   0          2m12s
sparta-app-deployment-7956fc8954-xnl88  1/1     Running   0          2m12s
```

- and also with `kubectl describe deployment sparta-app-deployment` (note the mentions of "Scaled up")

```
Events:
  Type    Reason             Age    From                   Message
  ----    ------             ----   ----                   -------
  Normal  ScalingReplicaSet  15m    deployment-controller  Scaled up replica set
 sparta-app-deployment-7956fc8954 from 0 to 3
  Normal  ScalingReplicaSet  8m41s  deployment-controller  Scaled down replica s
et sparta-app-deployment-7956fc8954 from 3 to 2
  Normal  ScalingReplicaSet  2m55s  deployment-controller  Scaled up replica set
 sparta-app-deployment-7956fc8954 from 2 to 4
  Normal  ScalingReplicaSet  2m40s  deployment-controller  Scaled up replica set
 sparta-app-deployment-7956fc8954 from 4 to 8
  Normal  ScalingReplicaSet  2m25s  deployment-controller  Scaled up replica set
 sparta-app-deployment-7956fc8954 from 8 to 10
```

11. Waited a little while for the HPA's cooldown period to be over so that the HPA started scaling down again, which I verified with `kubectl describe deployment sparta-app-deployment`

```
  Normal  ScalingReplicaSet  48s    deployment-controller  Scaled down replica s
et sparta-app-deployment-7956fc8954 from 10 to 4
  Normal  ScalingReplicaSet  2s     deployment-controller  Scaled down replica s
et sparta-app-deployment-7956fc8954 from 4 to 2
```

- I also verified this by running `kubectl get pods` and noting that I now only had 2 app pods, which is what was expected (again, given that HPA minimums take precedence over the number of replicas defined in my deployment file)

```
ubuntu@ip-172-31-58-36:/etc/nginx/sites-available$ kubectl get pods
NAME                                    READY   STATUS    RESTARTS   AGE
mongo-deployment-79c96444db-g97gj       1/1     Running   0          115m
sparta-app-deployment-7956fc8954-fq9cz  1/1     Running   0          22m
sparta-app-deployment-7956fc8954-lbd7x  1/1     Running   0          22m
```

Make Minikube start automatically on boot

1. To make Minikube start on the reboot of my instance, I created this file *`/etc/systemd/system/minikube.service`*
2. I then ran the following commands to put it into action:

```
sudo systemctl daemon-reload
sudo systemctl enable minikube
sudo systemctl restart minikube
sudo systemctl status minikube
```

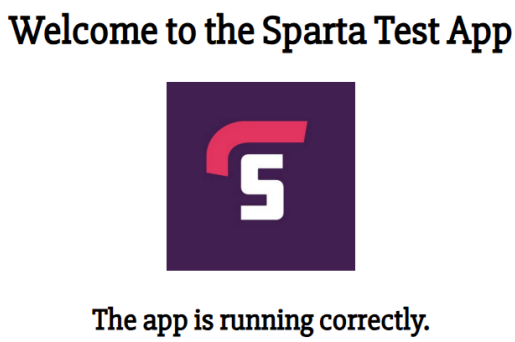
3. After verifying that the app was still accessible on the browser at this point, I rebooted the VM, logged in again, and ran `sudo systemctl status minikube`

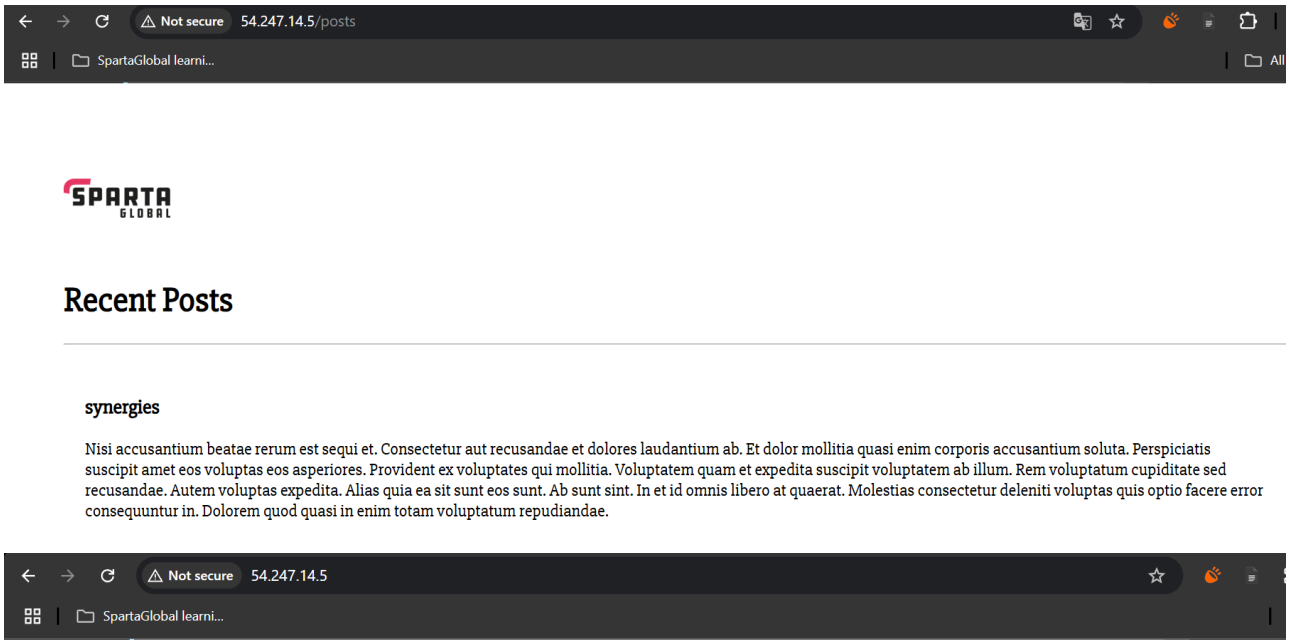
```
ubuntu@ip-172-31-58-36:/etc/systemd/system$ sudo systemctl status minikube
● minikube.service - Minikube
     Loaded: loaded (/etc/systemd/system/minikube.service; enabled; vendor pres>
     Active: active (exited) since Wed 2025-03-12 10:25:02 UTC; 5s ago
    Process: 7840 ExecStart=/usr/local/bin/minikube start --driver=docker (code>
   Main PID: 7840 (code=exited, status=0/SUCCESS)
        CPU: 1.733s

Mar 12 10:24:39 ip-172-31-58-36 minikube[7840]: * Suggestion: Start minikube wi>
Mar 12 10:24:39 ip-172-31-58-36 minikube[7840]: * Starting "minikube" primary c>
Mar 12 10:24:39 ip-172-31-58-36 minikube[7840]: * Pulling base image v0.0.46 ...
Mar 12 10:24:39 ip-172-31-58-36 minikube[7840]: * Restarting existing docker co>
Mar 12 10:24:50 ip-172-31-58-36 minikube[7840]: * Preparing Kubernetes v1.32.0 >
Mar 12 10:24:52 ip-172-31-58-36 minikube[7840]: * Verifying Kubernetes componen>
Mar 12 10:24:52 ip-172-31-58-36 minikube[7840]:   - Using image gcr.io/k8s-mini>
Mar 12 10:25:01 ip-172-31-58-36 minikube[7840]: * Enabled addons: default-stora>
Mar 12 10:25:01 ip-172-31-58-36 minikube[7840]: * Done! kubectl is now configur>
Mar 12 10:25:02 ip-172-31-58-36 systemd[1]: Finished Minikube.
```

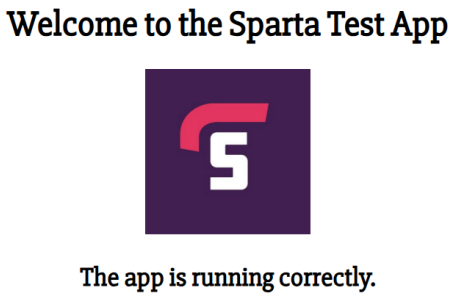
4. I also ran `minikube status` to verify that each component was running successfully

```
ubuntu@ip-172-31-58-36:/etc/systemd/system$ minikube status
minikube
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured
```

5. I then verified that my app was available on the browser







6. I then stopped the VM, started it again, and, after giving it a few minutes for everything to start up without logging in this time, I verified that my app was working on the browser again (note the new IP address as the URL, indicating that the EC2 was rebooted)

Welcome to the Sparta Test App

The app is running correctly.

SPARTA GLOBAL

Recent Posts

Checking Account

Modi reiciendis inventore ut sequi iusto. Consequatur est eius dolorem accusamus consequatur ex omnis et quo. Sequi voluptas consequatur iure ut sunt modi eius. Corrupti numquam ipsam voluptates ipsum iste aperiam facilis id saepe. Ut voluptas consequatur quidem et dicta est quo omnis vel. Accusamus quia ipsum sequi nihil voluptatem id. Non aspernatur est adipisci. Adipisci quas minus illum blanditiis officia est sint. Ut fugit aspernatur atque temporibus et impedit. Doloremque saepe temporibus tempore dolor consectetur nostrum et. Quia quaerat eius. Quo rerum sed animi in fugit. Rem consectetur sed labore vitae veritatis qui incidunt atque fugiat.

Blockers

1. On my first attempt to enable Minikube, I didn't use `Type=oneshot` or the `Wants:` field, and my `After:` and `Requires` fields only contained `docker.service`

- when rebooting the VM, this seemed to cause issues as I received connection refusals and lots of errors about conflicts/context changes when running any `kubectl` commands, so to undo this step, I ran:

```
sudo systemctl stop minikube
sudo systemctl disable minikube
sudo rm /etc/systemd/system/minikube.service
sudo systemctl daemon-reload

sudo systemctl restart docker

minikube delete
```

```
minikube start --driver=docker

minikube status
```

- I then edited the file so that it was how it appears here and it worked

What I learnt

- I learnt in-depth about how Kubernetes's architecture works
- I learnt about Kubernetes objects and how they relate to one another (e.g. deployments and ReplicaSets, and services)
- I learnt how powerful Kubernetes is for container orchestration

Benefits I personally saw from the project

- I appreciate that K8s can handle all of the scaling to meet traffic demands
- I like that objects like deployments can self-heal in case something goes wrong with a pod
- I understood more about the benefits of using Kubernetes to orchestrate many containers in production scenarios

Helpful links

- My notes on K8s commands are here
- Guide on which K8s API version to use depending on the kind of object
- Official Kubernetes documentation