Lab 2: Deploying a request splitting ambassador and a load balancer with Kubernetes

Course: SOFE 4790U
Date: 18/09/2022
Lab #: 1 (Individual Report)
Submitted by: Matheeshan Sivalingam(100703887)

Objective:

1. Learn how to configure and run a request splitter using Nginx.
2. Be familiar with the ConfigMap tool in Kubernetes.
3. Learn how to use the curl command for requesting an HTTP method.
4. Learn how to configure load balancer services
5. Get Familiar with load balancing pattern

Discussion:

Summarize the problem, the solution, and the requirements for the pattern given in part 1. Which of these requirements can be achieved by the procedures shown in parts 2 and 3?

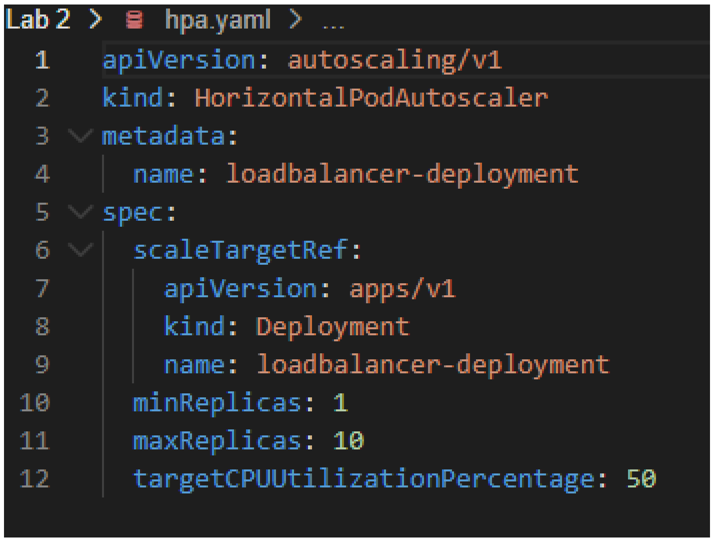
The problem that the Gateway Routing pattern intends to solve is the inability to consume multiple services without requiring the client to be updated whenever a service is added or removed. Rather than having the client be updated each time, the proposed solution provided by the pattern is to create a layer of abstraction or a gateway in which the client is able to consume multiple services. The requirements for implementing this pattern are needing multiple services for the client to consume, deploying multiple instances of the same service, or deploying multiple versions of the same service. It also requires application layer 7 routing as well as a single endpoint for the client to go communicate through.

In lab parts 2 and 3, the request-splitting ambassador and the load balancer serve as a single endpoint for the client to communicate through. Both clients in parts 2 and 3 communicate via layer 7 routing as they require the server IP address in order for them to use the service.

Design:

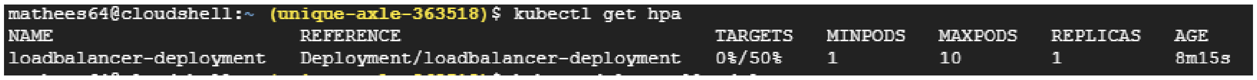
Autoscaling is another pattern that can be implemented by GKE. Your task is to configure a Yaml file that autoscaling the deployment of a given Pod. Why autoscaling is usually used? How autoscaling is implemented? How autoscaling is different than load balancing and request splitter?

Autoscaling is usually used as a way to autonomously control the number of networking resources allocated depending on the amount of traffic on the network. This essentially allows a dynamic allocation of resources which allows companies to save costs as they only deploy pods when necessary. It works by increasing the number of pods available depending on the specified metrics such as when the memory usage of a pod exceeds a certain threshold or on the rate of client requests per second. Autoscaling on GKE can be implemented in different ways. This includes deploying through configuring a YAML file, writing an autoscale command on Google Cloud CLI, or using the console. Autoscaling is different from load balancing as load balancing and request splitting distributes traffic to different nodes while autoscaling essentially horizontally increases the computational power.



```yaml
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: loadbalancer-deployment
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: loadbalancer-deployment
  minReplicas: 1
  maxReplicas: 10
  targetCPUUtilizationPercentage: 50
```

Figure 1 - YAML file of the Horizontal Pod Autoscaler configured



```
mathees64@cloudshell:~ (unique-axle-363518)$ kubectl get hpa
NAME                      REFERENCE                            TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
loadbalancer-deployment   Deployment/loadbalancer-deployment   0%/50%    1         10        1          8m15s
```

Figure 2 - Implementation of autoscaling use GKE