SOFE 4790U Distributed Systems

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

2 October 2022

Group 1- Michelle Cheng 100696572

Introduction
This Lab introduces how to deploy a request splitting ambassador and load balancer with Kubernetes in Google Cloud Platform (GCP).

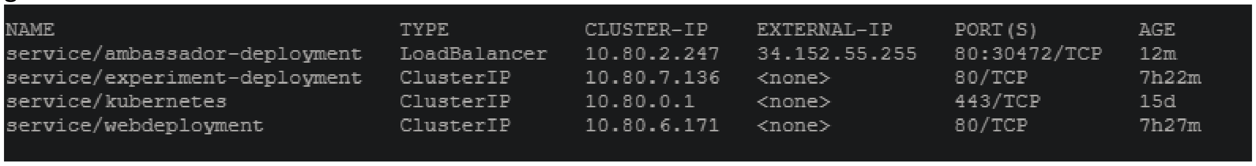
Main Objectives:
1. Learn how to configure and run a request splitter using Nginx.
2. Be familiar with the ConfigMap tool in Kubernetes.
3. Learn how to use the curl command for requesting an HTTP method.
4. Learn how to configure load balancer services.
5. Get Familiar with load balancing pattern.

Discussion
Part 1
Clients may consume multiple services, multiple instances of the same service or multiple version of the same service. During this process, the client needs to know what services are added and removed. For example, an application may have multiple services that require a different API that the client can interact with to search items, review, checkout and more. In another scenario, the system may need to have multiple instance of the same services for load balancing or to meet availability requirements. New versions of a service can also be released while the existing versions are still used. All of these examples require the client to be updated on the changes that are occurring.

The general solution for this would be to route the request to the appropriate instances in the application Layer 7. The Gateway Routing Pattern ensures that the client only need to interact at a single endpoint in order to communicate their request in. For multiple disparate services this pattern allows the client to have multiple access and keeps client calls simple. If a service was to be removed or replaced the client can continue to make requests as normal to the gateway and only the routing would change. Services can also be reorganized behind the gateway as needed. This type of pattern also allows for multiple instance of the same service to exist easier without the user being aware of the increase or decrease in the number of services. This pattern is also beneficial for deployments when new versions of an application is deployed in parallel to the existing. Any issues discovered after the service is deployed can be reverted by making a change at the gateway without affecting clients.

Some key requirements of setting up this gateway service include ensuring that the availability requirements are met as this pattern has a single point of failure and limiting public access to the backened services. Other requirements needed include meeting scalibility reuqirements, having load balancing functionaliteis within the gateway, differentiating between regional or global services.

Part 2
Request splitting ambassador was created to split 10% of incoming HTTP requests to experimental server and 90% to web deployment server. The ambassador-deployment had a external IP as seen below and CURL http://34.152.55.255 was used. To check all of the pods, deployments and services the kubectl get command was used.

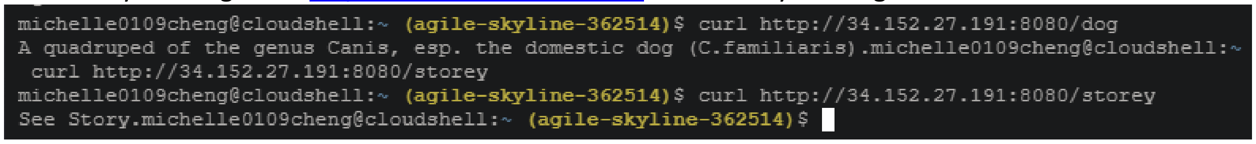


```
NAME                           TYPE           CLUSTER-IP    EXTERNAL-IP     PORT(S)        AGE
service/ambassador-deployment  LoadBalancer   10.80.2.247   34.152.55.255   80:30472/TCP   12m
service/experiment-deployment  ClusterIP      10.80.7.136   <none>          80/TCP         7h22m
service/kubernetes             ClusterIP      10.80.0.1     <none>          443/TCP        15d
service/webdeployment          ClusterIP      10.80.6.171   <none>          80/TCP         7h27m
```

Figure 1.0 Screenshot of kubectl get services for lab procedure Part 2

Part 3

A replicated load balancing service to process requests for the definition of English words was created. The NodeJS application takes the HTTP request paths with a word such as dog and storey and returns the definition of the word. 3 pods were replicated for this deployment and the external IP can be accesed by running CURL http://34.1452.27.191:8080 followed by the English word.



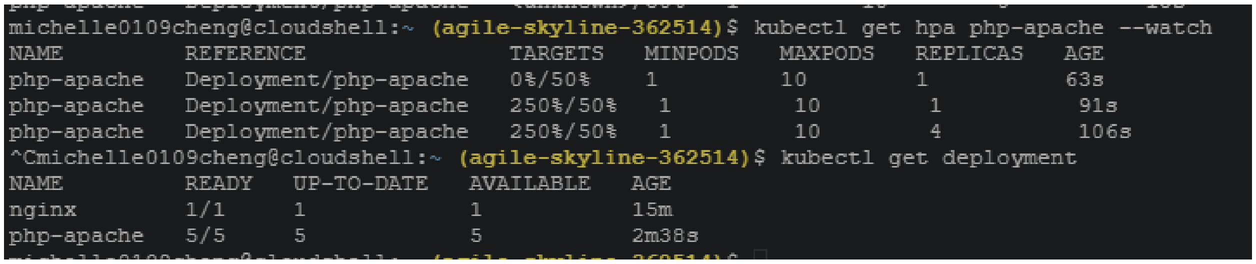
```
michelle0109cheng@cloudshell:~ (agile-skyline-362514)$ curl http://34.152.27.191:8080/dog
A quadruped of the genus Canis, esp. the domestic dog (C.familiaris).michelle0109cheng@cloudshell:~
 curl http://34.152.27.191:8080/storey
michelle0109cheng@cloudshell:~ (agile-skyline-362514)$ curl http://34.152.27.191:8080/storey
See Story.michelle0109cheng@cloudshell:~ (agile-skyline-362514)$
```

Figure 2.0 Results for Part 3

Design

Autoscaling allows for the user load to increase or decrease its resources as required. In Kubernetes, the Autoscaling can be applied to pods, nodes, and these can be scaled in and out as needed. Kubernetes Autoscaling has 3 methods Horizontal Pod Autoscaler (HPA) deploys extra pods when the load increases, Vertical Pod Autoscaler (VPA) resizes pods for CPU and memory resource optimization and Cluster Autoscaler (CA) increase and decrease size of cluster by adjusting number of nodes. The main difference between Autoscaling compared to Load balancer or request splitter is the automatic scaling up and down instead of distribution approach.

One way to set up autoscaling is to create the HorizontalPodAutoscaler that can hold a specified number of replicas as defined in the deployment. This is done using the public hp-example image and the Kubectl Autoscaler deployment command which allowed an average CPU utilization target to be created. As seen in the figure below, when the target exceeds the target CPU utilization of 50%, more replicas are created.



```
michelle0109cheng@cloudshell:~ (agile-skyline-362514)$ kubectl get hpa php-apache --watch
NAME         REFERENCE                TARGETS    MINPODS   MAXPODS   REPLICAS   AGE
php-apache   Deployment/php-apache   0%/50%     1         10        1          63s
php-apache   Deployment/php-apache   250%/50%    1         10        1          91s
php-apache   Deployment/php-apache   250%/50%    1         10        4          106s
^Cmichelle0109cheng@cloudshell:~ (agile-skyline-362514)$ kubectl get deployment
NAME         READY   UP-TO-DATE   AVAILABLE   AGE
nginx        1/1     1            1           15m
php-apache   5/5     5            5           2m38s
```

Figure 3.0 Autoscaling example

Deliverable

Video Submission Link:
https://drive.google.com/drive/folders/1KgcLQhA9NUC1Ulkv178chj_aUUEVG5xt?usp=sharing

Resources

- https://github.com/GeorgeDaoud3/SOFE4790U-lab2
- https://learn.microsoft.com/en-us/azure/architecture/patterns/gateway-routing
- https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale-walkthrough/
- https://www.densify.com/kubernetes-autoscaling