Faculty of Engineering and Applied Science

SOFE 4790U Distributed Systems

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

Group 19

Monil Patel 100727400

Group GitHub Link: https://github.com/sunilt4/Distributed-Systems/tree/main/Lab%202

Individual Video 1:
https://drive.google.com/file/d/1vNrN3ng9wyUgwcj7CJKqcz1JKOfPDRFx/view?usp=sharing
Individual Video 2:
https://drive.google.com/file/d/1-DD8B98RdTi_fHiQv-vPLt3bZESh6H7i/view?usp=sharing

Table of Contents

Video Links

Individual Video 1:
https://drive.google.com/file/d/1vNrN3ng9wyUgwcj7CJKqcz1JKOfPDRFx/view?usp=sharing

Individual Video 2:
https://drive.google.com/file/d/1-DD8B98RdTi_fHiQv-vPLt3bZESh6H7i/view?usp=sharing

Objectives

1. Learn how to configure and run a request splitter using Nginx
2. Become 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 a Load Balancer services
5. Get familiar with Load Balancing pattern

Link to GitHub repository containing files utilized in the lab:
https://github.com/GeorgeDaoud3/SOFE4790U-lab2

Procedure

Part 2:
You will be guided through the steps to deploy a request-splitting ambassador that will split 10% of the incoming HTTP requests to an experimental server.

Deploying YAML files:

```
monilp01@cloudshell:~ (lab2-364710)$ kubectl create -f web-deployment.yaml
deployment.apps/web-deployment created
monilp01@cloudshell:~ (lab2-364710)$ kubectl expose deployment web-deployment --port=80 --type=ClusterIP --name web-deployment
service/web-deployment exposed
monilp01@cloudshell:~ (lab2-364710)$ kubectl create -f experiment-deployment.yaml
deployment.apps/experiment-deployment created
monilp01@cloudshell:~ (lab2-364710)$ kubectl expose deployment experiment-deployment --port=80 --type=ClusterIP --name experiment-deployment
service/experiment-deployment exposed
monilp01@cloudshell:~ (lab2-364710)$ kubectl create configmap ambassador-config --from-file=conf.d
configmap/ambassador-config created
monilp01@cloudshell:~ (lab2-364710)$ kubectl create -f ambassador-deployment.yaml
deployment.apps/ambassador-deployment created
monilp01@cloudshell:~ (lab2-364710)$ kubectl expose deployment ambassador-deployment --port=80 --type=LoadBalancer
service/ambassador-deployment exposed
```

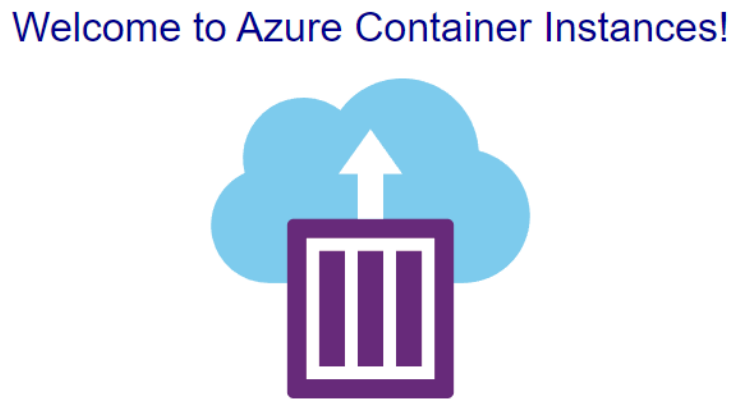
Get ambassador external IP:

```
monilp01@cloudshell:~ (lab2-364710)$ kubectl get services
NAME                    TYPE           CLUSTER-IP     EXTERNAL-IP   PORT(S)        AGE
ambassador-deployment   LoadBalancer   10.12.9.85     35.203.6.1    80:31520/TCP   49s
experiment-deployment   ClusterIP      10.12.8.154    <none>        80/TCP         97s
kubernetes              ClusterIP      10.12.0.1      <none>        443/TCP        4m15s
web-deployment          ClusterIP      10.12.13.125   <none>        80/TCP         107s
```

CURL output:

```
monilp01@cloudshell:~ (lab2-364710)$ curl http://35.203.6.1
<html>
<head>
  <title>Welcome to Azure Container Instances!</title>
</head>
<style>
h1 {
    color: darkblue;
    font-family:arial, sans-serif;
    font-weight: lighter;
}
</style>

<body>

<div align="center">
<h1>Welcome to Azure Container Instances!</h1>

<svg id="Layer_1" data-name="Layer 1" xmlns="http://www.w3.org/2000/svg" viewBox="0 0 49.8 49.9" width="250px" height="250px">
  <title>ContainerInstances_rgb_UI</title>
  <path d="M41.9,11.368A11.929,11.929,0,0,0,20.3,5.061a9.444,9.444,0,0,0-14.932,9.8A8.969,8.969,0,0,0,9.064,32H39.442A10.463,10.463,0,0,0,41.9,11.368Z" transform="translate(-0.1 -0.1)" fill="#fff"/>
        <path d="M41.9,11.368A11.929,11.929,0,0,0,20.3,5.061a9.444,9.444,0,0,0-14.932,9.8A8.969,8.969,0,0,0,9.064,32H39.442A10.463,10.463,0,0,0,41.9,11.368Z" transform="translate(-0.1 -0.1)" fill="#27a9e1" opacity="0.6" style="isolation:isolate"/>
        <path d="M13,22a1,1,0,0,0-1,1V49a1,1,0,0,0,1,1H37a1,1,0,0,0,1-1V23a1,1,0,0,0-1-1Z" transform="translate(-0.1 -0.1)" fill="#672a7a"/>
        <path d="M26.95,16" transform="translate(-0.1 -0.1)" fill="none"/>
        <path d="M34.95,20" transform="translate(-0.1 -0.1)" fill="none"/>
        <polygon points="22.9 21.9 22.9 14.9 19.9 14.9 24.9 7.9 29.9 14.9 26.9 14.9 26.9 21.9 22.9 21.9" fill="#fff"/>
        <path d="M26.95,16" transform="translate(-0.1 -0.1)" fill="#814a98"/>
        <path d="M33,25H15V47H35V25Z21,45H17V27h4Zm6,0H23V27h4Zm6,0H29V27h4Z" transform="translate(-0.1 -0.1)" fill="#b92025" opacity="0.3" style="isolation:isolate"/>
        <path d="M33,25H15V47H35V25Z21,45H17V27h4Zm6,0H23V27h4Zm6,0H29V27h4Z" transform="translate(-0.1 -0.1)" fill="#fff" style="isolation:isolate"/>
</svg>
</div>

</body>
</html>
```

Physical webpage:



Request splitting output:

```
monilp01@cloudshell:~ (lab2-364710)$ for _ in {1..20}; do curl http://35.203.6.1/ -s > output.txt; done
monilp01@cloudshell:~ (lab2-364710)$ kubectl logs -l run=web-deployment kubectl logs -l run=experiment-deployment
error: selectors and the all flag cannot be used when passing resource/name arguments
monilp01@cloudshell:~ (lab2-364710)$ kubectl logs -l run=web-deployment
listening on port 80
::ffff:10.8.0.7 - - [08/Oct/2022:16:34:10 +0000] "GET /favicon.ico HTTP/1.0" 404 150 "http://35.203.6.1/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/105.0.0.0 Safari/537.36"
::ffff:10.8.2.6 - - [08/Oct/2022:16:34:52 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.8.2.6 - - [08/Oct/2022:16:34:52 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.8.0.7 - - [08/Oct/2022:16:34:52 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.8.0.7 - - [08/Oct/2022:16:34:52 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.8.2.6 - - [08/Oct/2022:16:34:52 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.8.0.7 - - [08/Oct/2022:16:34:53 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.8.2.6 - - [08/Oct/2022:16:34:53 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.8.2.6 - - [08/Oct/2022:16:34:53 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.8.0.7 - - [08/Oct/2022:16:34:52 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.8.2.6 - - [08/Oct/2022:16:34:52 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.8.2.6 - - [08/Oct/2022:16:34:52 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.8.0.7 - - [08/Oct/2022:16:34:52 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.8.2.6 - - [08/Oct/2022:16:34:52 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.8.0.7 - - [08/Oct/2022:16:34:52 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.8.0.7 - - [08/Oct/2022:16:34:53 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.8.0.7 - - [08/Oct/2022:16:34:53 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.8.0.7 - - [08/Oct/2022:16:34:53 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.8.2.6 - - [08/Oct/2022:16:34:53 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
monilp01@cloudshell:~ (lab2-364710)$ kubectl logs -l run=experiment-deployment
listening on port 80
listening on port 80
::ffff:10.8.0.7 - - [08/Oct/2022:16:34:52 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.8.2.6 - - [08/Oct/2022:16:34:52 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
monilp01@cloudshell:~ (lab2-364710)$
```

Part 3:

You will be guided through the steps to deploy a replicated load balancing service that will process requests for the definition of English words. The requests will be processed by a small NodeJS application that we will fire up in Kubernetes using a pre-existing Docker image

```
monilp01@cloudshell:~ (lab2-364710)$ kubectl create -f loadbalancer-deployment.yaml
deployment.apps/loadbalancer-deployment created
monilp01@cloudshell:~ (lab2-364710)$ kubectl get pods --output=wide
NAME                                      READY   STATUS              RESTARTS   AGE   IP       NODE                                      NOMINATED NODE   READINESS GATES
loadbalancer-deployment-6676f9ccf6-2f4vs  0/1     ContainerCreating   0          2s    <none>   gke-lab2-2-default-pool-edc54573-x26v   <none>           <none>
loadbalancer-deployment-6676f9ccf6-pb5ph  0/1     ContainerCreating   0          2s    <none>   gke-lab2-2-default-pool-edc54573-fz7r   <none>           <none>
loadbalancer-deployment-6676f9ccf6-x4s74  0/1     ContainerCreating   0          2s    <none>   gke-lab2-2-default-pool-edc54573-9j18   <none>           <none>
monilp01@cloudshell:~ (lab2-364710)$ kubectl expose deployment loadbalancer-deployment --port=8080 --type=LoadBalancer
service/loadbalancer-deployment exposed
monilp01@cloudshell:~ (lab2-364710)$ kubectl get services --watch
NAME                     TYPE          CLUSTER-IP    EXTERNAL-IP   PORT(S)          AGE
kubernetes               ClusterIP     10.96.0.1     <none>        443/TCP          4m44s
loadbalancer-deployment  LoadBalancer  10.96.8.113   <pending>     8080:31352/TCP   6s
^Cmonilp01@cloudshell:~ (lab2-364710)$ kubectl get services --watch
NAME                     TYPE          CLUSTER-IP    EXTERNAL-IP    PORT(S)          AGE
kubernetes               ClusterIP     10.96.0.1     <none>         443/TCP          5m18s
loadbalancer-deployment  LoadBalancer  10.96.8.113   35.203.61.81   8080:31352/TCP   40s
^Cmonilp01@cloudshell:~ (lab2-364710)$ curl http://35.203.61.81:8080/dog
A quadruped of the genus Canis, esp. the domestic dog (C.familiaris).monilp01@cloudshell:~ (lab2-364710)$ curl http://35.203.61.81:8080/storey
See Story.monilp01@cloudshell:~ (lab2-364710)$ curl http://35.203.61.81:8080/cat
An animal of various species of the genera Felis and Lynx. Thedomestic cat is Felis domestica. The European wild cat (Felis catus)is much larger than the domestic cat. In the United States the namewild cat is commonly applied to the bay lynx
monilp01@cloudshell:~ (lab2-364710)$
```

Discussion

1. Summarize the problem, the solution, and the requirements for the pattern given in part 1.

Using routing requests on multiple services or service instances using a single endpoint, we can expose multiple services on one endpoint and route to the appropriate service based on the input request. We can also expose multiple instances of the same service on for load balancing and availability. Finally, we can expose different versions of the same service on a single endpoint and route traffic as needed across the different versions.

We can solve some problems by exposing the client to a single endpoint and having the request split from there. Some of the requirements can be summarized as following: This is useful when a client needs to consume multiple services. For instance, they may need multiple disparate services where each microservice has its own API that the client interacts with, and information is exchanged in between. Another use case would be if multiple instances of a service need to be run, where the requests are split amongst the multiple instances, or a service with different versions is being utilized as well.

By utilizing this gateway routing pattern, we can handle multiple services on a gateway, by only exposing the client to a single endpoint and routing external addresses to internal endpoints. We also want to make sure there are multiple services available, so that in the case there are multiple services that need access at the same time, the request can be split and balanced to maintain them all. All in all, we need some services that utilize APIs and endpoints to communicate with the client.

2. Which of these requirements can be achieved by the procedures shown in parts 2 and 3?

In part 2, we have two deployments of the same web page deployed onto one cluster. However, our goal is to simulate the case of two different servers that handle requests to the webpage. We have our deployments active in our cluster, which are being managed by an ambassador using nginx to manage and split incoming requests. We specify that we want the main deployment to handle 90% of requests while the experimental deployment 10%. The ambassador deployment used the load balancer to handle this 90-10 split. Once we have the external IP of the ambassador, we can send request to our deployments and see the results. When we run 20 requests, we can see the 90-10 split in the logs.

For part 3, we are starting with a load balancer deployment using 3 pods of the image and an extra readiness pod. We are checking the readiness of each pod every 5 seconds to see if it can be used again, and if so, the requests are sent to that pod. We can execute the requests based on the routing defined, for example ending with dog or cat for their definitions.

Design

Autoscaling is another pattern that can be implemented by GKE. Your task is to configure a Yaml file that shows autoscaling the deployment of a given Pod.

1. Why is autoscaling usually used?

Autoscaling allows resources to be scaled up or down based on utilization, traffic, or demand. It is a way to make sure that the costs of a cloud service are always as needed (pay per use) in a way that businesses pay for whatever is needed and not anymore. By using autoscaling, the resource utilization is automized to meet demand, which always helps to retain service availability in cases where traffic is high instantly.

2. How is autoscaling implemented?

Autoscaling monitors resource utilization in a cluster, and always aims to provide a certain threshold above the demand of resources. This way, service failures are avoided, and timing dependencies are met in instantaneous spikes and drops of resource/traffic,

3. How is autoscaling different than load balancing and request splitter?

Autoscaling aims to provide vertical scaling of resources of a cluster per demand within an environment, whereas load balancing aims to split requests among different pods based on current pod utilizations. A request splitter helps split incoming requests (traffic) based on the split to different pods, which is useful when utilizing multiple instances of a single endpoint.

References

[1]
Erjosito, “Gateway Routing pattern - Azure Architecture Center,” *Microsoft.com*, 2022. https://learn.microsoft.com/en-us/azure/architecture/patterns/gateway-routing (accessed Oct. 06, 2022).

[2]

“Docker Hub,” *Docker.com*, 2022. https://hub.docker.com/_/microsoft-azuredocs-aci-helloworld (accessed Oct. 06, 2022).

[3]

“Configuring horizontal Pod autoscaling | Google Kubernetes Engine (GKE) | Google Cloud,” *Google Cloud*, 2022. https://cloud.google.com/kubernetes-engine/docs/how-to/horizontal-pod-autoscaling (accessed Oct. 06, 2022).