Faculty of Engineering & Applied Science

SOFE4790U – Distributed Systems

Group Work

Lab 2 – Deploying a Request Splitting Ambassador and Load Balancer with Kubernetes

Due Date: 10/09/2022

First Name	Last Name	Student ID
Abdul	Bhutta	100785884
Alexander	Campbell	100703650
Mamun	Hossain	100553073
Owen	Musselman	100657709

Objectives:

- Learn how to configure and run a request splitter using Nginx.
- Be familiar with the ConfigMap tool in Kubernetes.
- Learn how to use the curl commands for requesting an HTTP method.
- Learn how to configure load balancer services.
- Get familiar with load balancing pattern.

Read the following document (Gateway Routing pattern - Azure Architecture Center | Microsoft Learn). Focus on the problem being solved by the pattern, how is it solved? and the requirements needed for the solution.

The problem is that when a client/user is accessing the various services at once, various instances, or a combination, then the client must be updated whenever there is a change in the service. E-commerce as an example, needs multiple services like searching for an item, checking out, adding an item to a cart. If a new service is added, like order history, it needs to be updated on the client side as well. The below table goes through more specific examples.

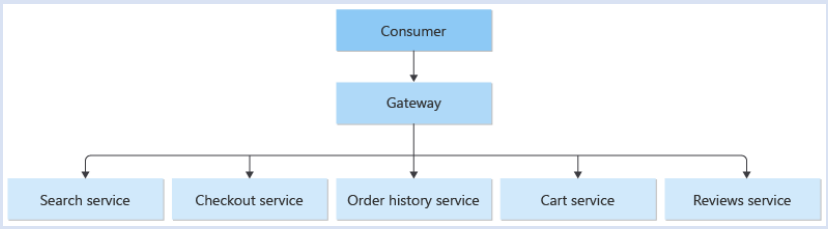
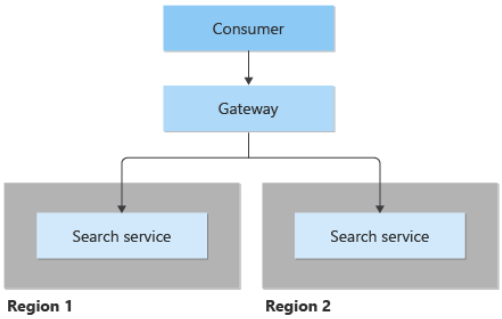
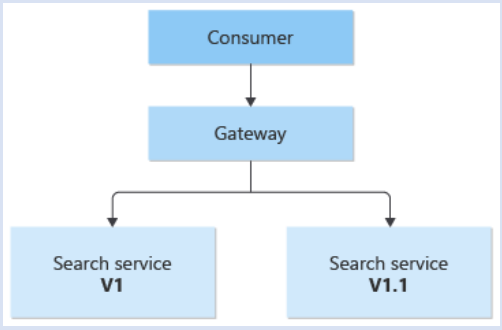
Scenario	Description
Multiple disparate services	Each service requires an API call that the client will have to communicate with, while each endpoint will be different. Each time a change is made in the code, the client and the service must be updated.
Multiple instances of the same service	The same services can be run multiple times for load balancing, or the service can be offered in different regions.
Multiple versions of the same service	A new version of the service can be deployed with the existing version.

Solution:

The above problem can be solved by the use of a gateway that is placed between the client and the services (applications or deployments). This is done at layer 7 (application layer). The client only needs a single endpoint and it will communicate with a single endpoint.

Requirements:

- Elasticity
- Availability
- Reliability
- Latency
- Greater than 1 or 2 services

Scenario	Solution Diagrams
Multiple disparate services	
Multiple instances of the same service	
Multiple versions of the same service	

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.

i. Create web server and experiment server while making them accessible.

```
bhutta_abdul@cloudshell:~ (sofe4790u-lab2)$ kubectl delete deployment web-deployment
deployment.apps "web-deployment" deleted
bhutta_abdul@cloudshell:~ (sofe4790u-lab2)$ kubectl delete deployment experiment-deployment
deployment.apps "experiment-deployment" deleted
bhutta_abdul@cloudshell:~ (sofe4790u-lab2)$ kubectl delete services experiment-deployment
service "experiment-deployment" deleted
bhutta_abdul@cloudshell:~ (sofe4790u-lab2)$ kubectl create -f web-deployment.yaml
deployment.apps/web-deployment created
bhutta_abdul@cloudshell:~ (sofe4790u-lab2)$ kubectl expose deployment web-deployment --port=80 --type=ClusterIP --name web- deployment
Error from server (Invalid): Service "web-" is invalid: metadata.name: Invalid value: "web-": a DNS-1035 label must consist of lower case alphanumeric characters or '-', start with an alphabetic ch
aracter, and end with an alphanumeric character (e.g. 'my-name',  or 'abc-123', regex used for validation is '[a-z]([-a-z0-9]*[a-z0-9])?')
Error from server (NotFound): deployments.apps "deployment" not found
bhutta_abdul@cloudshell:~ (sofe4790u-lab2)$ kubectl expose deployment web-deployment --port=80 --type=ClusterIP --name web-deployment
service/web-deployment exposed
bhutta_abdul@cloudshell:~ (sofe4790u-lab2)$ kubectl create -f experiment-deployment.yaml
deployment.apps/experiment-deployment created
bhutta_abdul@cloudshell:~ (sofe4790u-lab2)$  kubectl expose deployment experiment-deployment --port=80 --type=ClusterIP --name experiment-deployment
service/experiment-deployment exposed
bhutta_abdul@cloudshell:~ (sofe4790u-lab2)$
```

ii. Generate configMap

```
bhutta_abdul@cloudshell:~$ kubectl create configmap ambassador-config --from-file=conf.d
configmap/ambassador-config created
```

iii. Deploy the ambassador service

```
bhutta_abdul@cloudshell:~$ kubectl create -f ambassador-deployment.yaml
deployment.apps/ambassador-deployment created
bhutta_abdul@cloudshell:~$
```

iv. Check if the pods, deployments, and services are running

```
bhutta_abdul@cloudshell:~$ kubectl get pods
NAME                                    READY   STATUS    RESTARTS   AGE
ambassador-deployment-66db4f7766-br4vq  1/1     Running   0          113s
ambassador-deployment-66db4f7766-nmhzk  1/1     Running   0          113s
experiment-deployment-7b47cbd668-jt4b4  1/1     Running   0          10m
experiment-deployment-7b47cbd668-t6sgs  1/1     Running   0          10m
mongodb-deployment-5c6cb86f45-8lqqq     1/1     Running   0          4d21h
mysql-deployment-5496fdc956-994jm       1/1     Running   0          4d21h
web-deployment-6fdbb5c6bb-5zjvc         1/1     Running   0          33m
web-deployment-6fdbb5c6bb-pwp7f         1/1     Running   0          33m
bhutta_abdul@cloudshell:~$ kubectl get deployments
NAME                   READY   UP-TO-DATE   AVAILABLE   AGE
ambassador-deployment  2/2     2            2           2m6s
experiment-deployment  2/2     2            2           10m
mongodb-deployment     1/1     1            1           14d
mysql-deployment       1/1     1            1           21d
web-deployment         2/2     2            2           33m
bhutta_abdul@cloudshell:~$ kubectl get services
NAME                   TYPE           CLUSTER-IP    EXTERNAL-IP      PORT(S)          AGE
ambassador-deployment  LoadBalancer   10.84.6.158   34.171.252.47    80:32216/TCP     91s
experiment-deployment  ClusterIP      10.84.3.33    <none>           80/TCP           11m
kubernetes             ClusterIP      10.84.0.1     <none>           443/TCP          23d
mongodb-service        LoadBalancer   10.84.8.48    35.226.185.35    3306:30647/TCP   14d
mysql-service          LoadBalancer   10.84.0.54    34.171.222.236   3306:30136/TCP   21d
web-deployment         ClusterIP      10.84.9.181   <none>           80/TCP           15m
bhutta_abdul@cloudshell:~$
```

Finally, check that all pods, deployments, and services are running with no issues. What commands should you use? What's the external IP address associated with the **ambassador-deployment** service?

1. Check pods are running -> **kubectl get pods**
2. Check deployments are running -> **kubectl get deployments**
3. Check services are running -> **kubectl get services**
4. External IP address to the ambassador deployment -> **34.171.252.47**

v. View the output

vi. Run the script to call the server many times and view the output.txt file.

```
bhutta_abdul@cloudshell:~$ for _ in {1..20}; do curl http://34.171.252.47 -s > output.txt; done
bhutta_abdul@cloudshell:~$
bhutta_abdul@cloudshell:~$
```

output.txt

```
<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="M41.9,11.368A11.929,11.929,0,0,0,20.3,5.061a9.444,9.444,0,0,0-14.932,9.8A8.969,8.969,0,0,0,0,9.064,32H39.442A10.463,10.463,0,0,0,41.9,11.368Z" transform="t
    <path d="M41.9,11.368A11.929,11.929,0,0,0,20.3,5.061a9.444,9.444,0,0,0-14.932,9.8A8.969,8.969,0,0,0,0,9.064,32H39.442A10.463,10.463,0,0,0,41.9,11.368Z" transform=
    <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="M33,25H15V47H35V25ZM21,45H17V27h4Zm6,0H23V27h4Zm6,0H29V27h4Z" transform="translate(-0.1 -0.1)" fill="#b92025" opacity="0.3" style="isolation:isolate"/>
    <path d="M33,25H15V47H35V25ZM21,45H17V27h4Zm6,0H23V27h4Zm6,0H29V27h4Z" transform="translate(-0.1 -0.1)" fill="#fff" style="isolation:isolate"/>
</svg>
</div>
```

vii. Read logs to view how many times each server was called.

```
bhutta_abdul@cloudshell:~$ kubectl logs -l run=web-deployment
::ffff:10.80.3.6 - - [07/Oct/2022:17:59:14 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.80.3.6 - - [07/Oct/2022:17:59:14 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.80.3.6 - - [07/Oct/2022:17:59:14 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.80.2.10 - - [07/Oct/2022:17:59:14 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.80.3.6 - - [07/Oct/2022:17:59:14 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.80.3.6 - - [07/Oct/2022:17:59:14 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.80.2.10 - - [07/Oct/2022:17:59:15 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.80.3.6 - - [07/Oct/2022:17:59:15 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.80.3.6 - - [07/Oct/2022:17:59:15 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.80.2.10 - - [07/Oct/2022:17:59:15 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.80.3.6 - - [07/Oct/2022:17:49:59 +0000] "POST / HTTP/1.0" 404 140 "-" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit
::ffff:10.80.3.6 - - [07/Oct/2022:17:51:46 +0000] "GET /favicon.ico HTTP/1.0" 404 150 "http://34.171.252.47/" "Mozilla/5.0 (
ri/605.1.15"
::ffff:10.80.3.6 - - [07/Oct/2022:17:52:26 +0000] "GET / HTTP/1.0" 200 1663 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64; r
::ffff:10.80.2.10 - - [07/Oct/2022:17:59:14 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.80.2.10 - - [07/Oct/2022:17:59:14 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.80.2.10 - - [07/Oct/2022:17:59:14 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.80.3.6 - - [07/Oct/2022:17:59:15 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.80.3.6 - - [07/Oct/2022:17:59:15 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.80.3.6 - - [07/Oct/2022:17:59:15 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.80.3.6 - - [07/Oct/2022:17:59:15 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
bhutta_abdul@cloudshell:~$
bhutta_abdul@cloudshell:~$
 bhutta_abdul@cloudshell:~$ kubectl logs -l run=experiment-deployment
 listening on port 80
 ::ffff:10.80.2.10 - - [07/Oct/2022:17:59:15 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
 listening on port 80
 ::ffff:10.80.3.6 - - [07/Oct/2022:17:59:14 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
 ::ffff:10.80.3.6 - - [07/Oct/2022:17:59:15 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
```

Part 3:

I. Creating the load balancer deployment by using the yaml file given in the github.

```
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$ kubectl create -f loadbalancer-deployment.
aml
deployment.apps/loadbalancer-deployment created
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$
```

II. Displaying the load balancing pods created (they are highlighted in yellow).

```
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$ kubectl get pods --output=wide
NAME                                    READY   STATUS    RESTARTS   AGE    IP
ODE                                      NOMINATED NODE   READINESS GATES
ambassador-deployment-66db4f7766-pjww8  1/1     Running   0          14h    10.104.1.11
ke-testcluster-default-pool-565f249a-9lyo  <none>           <none>
ambassador-deployment-66db4f7766-skhqq  1/1     Running   0          14h    10.104.0.6
ke-testcluster-default-pool-565f249a-u5l2  <none>           <none>
experiment-deployment-7b47cbd668-5f4gk  1/1     Running   0          14h    10.104.2.6
ke-testcluster-default-pool-565f249a-iga2  <none>           <none>
experiment-deployment-7b47cbd668-ngkhk  1/1     Running   0          14h    10.104.1.10
ke-testcluster-default-pool-565f249a-9lyo  <none>           <none>
loadbalancer-deployment-6676f9ccf6-2q6js  1/1     Running   0          94s    10.104.0.7
ke-testcluster-default-pool-565f249a-u5l2  <none>           <none>
loadbalancer-deployment-6676f9ccf6-7tqpl  1/1     Running   0          94s    10.104.2.7
ke-testcluster-default-pool-565f249a-iga2  <none>           <none>
loadbalancer-deployment-6676f9ccf6-whkc8  1/1     Running   0          94s    10.104.1.12
ke-testcluster-default-pool-565f249a-9lyo  <none>           <none>
mongodb-deployment-7945646c67-r94d6     1/1     Running   0          4d2h   10.104.1.2
ke-testcluster-default-pool-565f249a-9lyo  <none>           <none>
mysql-deployment-5496fdc956-8g9rw       1/1     Running   0          4d2h   10.104.0.2
ke-testcluster-default-pool-565f249a-u5l2  <none>           <none>
web-deployment-6fdbb5c6bb-6fmvm         1/1     Running   0          14h    10.104.1.9
ke-testcluster-default-pool-565f249a-9lyo  <none>           <none>
web-deployment-6fdbb5c6bb-phd4f         1/1     Running   0          14h    10.104.2.5
ke-testcluster-default-pool-565f249a-iga2  <none>           <none>
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$
```

III. Exposing the load balancer.

```
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$ kubectl expose deployment loadbalancer-dep
oyment --port=8080 --type=LoadBalancer
service/loadbalancer-deployment exposed
```

IV. Querying the load balancer by using curl with the external IP.

```
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$ curl http://35.203.71.168:8080/story


A set of rooms on the same floor or level; a floor, or thespace between two floors. Also, a h
rizontal division of a building'sexterior considered architecturally, which need not correspo
dexactly with the stories within. [Written also storey.]slikcaustic1@cloudshell:~ (bubbly-gsl
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$
```

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 usage of a gateway pattern is able to route the various instances/services to a single endpoint, which is done through placing the gateway in front of the services so that the traffic is directed to the proper instance/service.

For part 2, the ambassador-deployment is used as a request splitter, which will split the traffic between the experiment-deployment and the web-deployment, where 10% is sent to the experiment-deployment and the remaining 90% is sent to the web-deployment. The client communicates with a single endpoint, and the gateway/splitter manages the routes taken. This in essence is Multiple versions of the same service.

For part 3, load balancing is used, where the load balancer, distributes the loads between 3 pods, and this is done through the readinessProbe within the loadbalancer-deployment.yaml file, without this, traffic would be sent even if a new pod is not ready to receive traffic, and this will occur every 5 seconds.

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 from load balancing and request splitter?

Autoscaling is used to make sure that a desired application is currently operating at the required performance level. It allows the service to have greater availability, reliability and elasticity, even when there are hardware failures. With auto scaling a more optimized application environment is achieved through a dynamic hardware environment, where the amount of resources scales up or down depending on the workload of the service, which will lower the cost of operation when usage is low since less hardware is assigned to the service.

In google cloud, auto scaling is implemented through the use of kubernetes (k8s). When implementing through k8s, you need to specify the minimum, maximum replicas to make, and the average cpu usage of the pods as a whole. Auto scaling is different from load balancing and request splitting as auto scaling scales the resources up or down according to the workload, while load balancing distributes the incoming traffic to the various locations that they can be served. A request splitter will split the incoming traffic according to a user specified distribution, like our 10% and 90% in part 2, which can be used to help test between versions before they are deployed.

The implementation done in this section was accomplished through the k8s documentation which used a horizontal pod autoscaling image. A php apache server listened for requests. The average cpu usage over all the pods was set to 50%, with the number of replicas having a minimum of 1 and a maximum of 10. So whenever a workload would increase, the cpu usage would increase and the replicas would increase too. When the workload subsides, the cpu usage would decrease, and the replicas would decrease to reflect the lower workload.

Implementation of horizontal autoscaling:

i. Yaml file that holds the deployment and service information for the horizontal pod autoscaler. Initially start with 3 replicas, 500millicores of cpu, with a minimum of 200 millicores of processing power for the application.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hpa-experiment
spec:
  selector:
    matchLabels:
      run: hpa-experiment
  replicas: 3 # Start with 3 replicas initially.
  template:
    metadata:
      labels:
        run: hpa-experiment
    spec:
      containers:
      - name: hpa-experiment
        image: k8s.gcr.io/hpa-example # Utilize example image for horizontal pod autoscaling.
        ports:
        - containerPort: 80 # Set the port to 80.
        resources:
          limits:
            cpu: 500m # Limits the container to use half of a cpu core for the application. A full core is would be cpu: 1000m where m is millicores.
          requests:
            cpu: 200m # Requests the minimum compute capacity. In this case set to 200millicores.
---
  # Service information.
apiVersion: v1
kind: Service
metadata:
  name: hpa-experiment
  labels:
    run: hpa-experiment
spec:
  ports:
  - port: 80
  selector:
    run: hpa-experiment
```

ii. Creating the deployments and the services that are to be autoscaled.

```
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$ kubectl apply -f horizontal-experiment.yaml
deployment.apps/hpa-experiment created
service/hpa-experiment created
```

iii. Apply autoscaling to the hpa-experiment deployment, with the aim of having an average of 50% cpu usage over all the current pods. The minimum replicas of the pods is set to 1 and the max for this test is set to 10 replicas. The current status of the workload is 0% out of 50% which is accurate as there is currently no work being done. Additionally, there are currently 3 replicas.

```
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$ kubectl autoscale deployment hpa-experiment --cpu-percent=50 --min=1 --max=10
horizontalpodautoscaler.autoscaling/hpa-experiment autoscaled
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$ kubectl get hpa
NAME             REFERENCE                   TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
hpa-experiment   Deployment/hpa-experiment   0%/50%    1         10        3          55s
```

iv. Now, after some time, there has been no workload and the replicas have been scaled down, as intended.

```
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$ kubectl get hpa
NAME             REFERENCE                   TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
hpa-experiment   Deployment/hpa-experiment   0%/50%    1         10        3          55s
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$ kubectl get hpa
NAME             REFERENCE                   TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
hpa-experiment   Deployment/hpa-experiment   0%/50%    1         10        1          10m
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$
```

v. Create a workload by querying the hpa-experiment service infinitely. After a while, the replicas should increase accordingly.

```
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$ kubectl run -i --tty load-generator --rm --image=busybox --restart=Never -- /bin/sh -c "while sleep 0.01; do wget -q -O- http://hpa-experimen
t; done"
If you don't see a command prompt, try pressing enter.
OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!
```

vi. After a few minutes of executing the replicas have been increased to 8 to fit the workload, and the average cpu usage has been increased from 0% to 44% now as well.

```
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$ kubectl get hpa
NAME             REFERENCE                    TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
hpa-experiment   Deployment/hpa-experiment    0%/50%    1         10        3          55s
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$ kubectl get hpa
NAME             REFERENCE                    TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$ kubectl get hpa
NAME             REFERENCE                    TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
hpa-experiment   Deployment/hpa-experiment    44%/50%   1         10        8          25m
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$
```

Conclusion:

After completing the steps for this lab, we have now learned how to configure and run a simple request splitter utilizing nginx. An increased familiarity with the ConfigMap tool within Kubernetes. Additionally, we have a better understanding of using simple curl commands after completing the steps. Another configuration that is now understood is load balancing services and load balancing patterns. While doing the design section of this lab some struggles were finding resources on horizontal auto scaling that were applicable to the task provided but were achievable. And now due to looking at many resources about horizontal auto-scaling in Kubernetes, we can confidently say we have a better understanding of horizontal auto-scaling, and how to implement it into projects.

======================== END OF GROUP WORK ========================

Homework 2: Network and Virtualization
Member: Abdul Bhutta
Changes: RED

1.0 Part 1 – Gateway Routing Pattern

Read the following document (Gateway Routing pattern - Azure Architecture Center | Microsoft Learn). Focus on the problem being solved by the pattern, how is it solved? and the requirements needed for the solution.

The problem is that when a client is accessing multiple services at once, numerous service instances, or both, the client must be updated when there is a change in the services (Add or remove). An example is when the user is accessing an e-commerce website, they will have various services such as searching for an item, adding an item to the cart, and checkout. If a new service is added, such as order history, it must also be updated on the client side. In the table, you can see the different scenarios that may occur,

Scenario	Description
Multiple disparate services	Each service requires an API call that the client will have to communicate with, while each endpoint will be different. Each time a change is made in the code, the client and the service must be updated.
Multiple instances of the same service	The same services can be run multiple times for load balancing, or the service can be offered in different regions.
Multiple versions of the same service	A new version of the service can be deployed with the existing version.

The solution

One of the solutions to the problem is to place a gateway between the client and services where it will route the request to the desired service. The client only requires one endpoint and will communicate with the gateway. Since there is only one endpoint, the client side will communicate with a single endpoint. The table below shows the solutions to the different scenarios and how to integrate the gateway.

Multiple disparate services

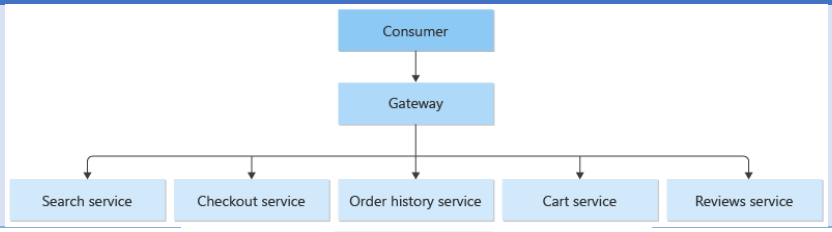
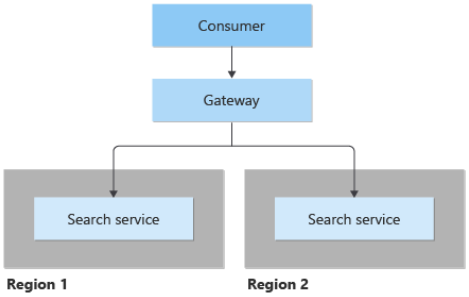
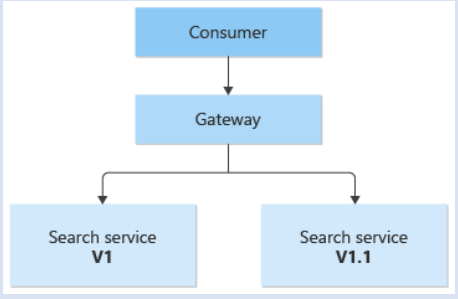
When the gateway routing protocol is applied, the client will send a request to the gateway, which routes it to the correct instance. If a change is made in service or removed, the client does not need to change; only the routing will be updated.

Multiple instances of the same service

Elasticity in cloud computing is extremely useful as services are never consistent throughout the year and fluctuate between various times and regions. The gateway routing protocol allows an increase in the number of services to meet the demand or a decrease in the number of services to save money while keeping it hidden from the user. The gateway will route the client to the best endpoint for their region.

Multiple versions of the same service

Multiple versions can be deployed, which helps the developer release them in parallel or incremental updates, while the gateway routing allows us to choose which service can be presented to the client. If the new deployments have issues, they can be changed back to the old version without affecting the clients.

Scenario	Solution Diagrams
Multiple disparate services	
Multiple instances of the same service	
Multiple versions of the same service	

2.0 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.

i. Create web server and experiment server while making them accessible.

```
bhutta_abdul@cloudshell:~ (sofe4790u-lab2)$ kubectl delete deployment web-deployment
deployment.apps "web-deployment" deleted
bhutta_abdul@cloudshell:~ (sofe4790u-lab2)$ kubectl delete deployment experiment-deployment
deployment.apps "experiment-deployment" deleted
bhutta_abdul@cloudshell:~ (sofe4790u-lab2)$ kubectl delete services experiment-deployment
service "experiment-deployment" deleted
bhutta_abdul@cloudshell:~ (sofe4790u-lab2)$ kubectl create -f web-deployment.yaml
deployment.apps/web-deployment created
bhutta_abdul@cloudshell:~ (sofe4790u-lab2)$ kubectl expose deployment web-deployment --port=80 --type=ClusterIP --name web- deployment
Error from server (Invalid): Service "web-" is invalid: metadata.name: Invalid value: "web-": a DNS-1035 label must consist of lower case alphanumeric characters or '-', start with an alphabetic ch
aracter, and end with an alphanumeric character (e.g. 'my-name',  or 'abc-123', regex used for validation is '[a-z]([-a-z0-9]*[a-z0-9])?')
Error from server (NotFound): deployments.apps "deployment" not found
bhutta_abdul@cloudshell:~ (sofe4790u-lab2)$ kubectl expose deployment web-deployment --port=80 --type=ClusterIP --name web-deployment
service/web-deployment exposed
bhutta_abdul@cloudshell:~ (sofe4790u-lab2)$ kubectl create -f experiment-deployment.yaml
deployment.apps/experiment-deployment created
bhutta_abdul@cloudshell:~ (sofe4790u-lab2)$  kubectl expose deployment experiment-deployment --port=80 --type=ClusterIP --name experiment-deployment
service/experiment-deployment exposed
bhutta_abdul@cloudshell:~ (sofe4790u-lab2)$
```

ii. Generate configMap

```
bhutta_abdul@cloudshell:~$ kubectl create configmap ambassador-config --from-file=conf.d
configmap/ambassador-config created
```

iii. Deploy the ambassador service

```
bhutta_abdul@cloudshell:~$ kubectl create -f ambassador-deployment.yaml
deployment.apps/ambassador-deployment created
bhutta_abdul@cloudshell:~$
```

iv. Check if the pods, deployments, and services are running

```
bhutta_abdul@cloudshell:~$ kubectl get pods
NAME                                    READY   STATUS    RESTARTS   AGE
ambassador-deployment-66db4f7766-br4vq  1/1     Running   0          113s
ambassador-deployment-66db4f7766-nmhzk  1/1     Running   0          113s
experiment-deployment-7b47cbd668-jt4b4  1/1     Running   0          10m
experiment-deployment-7b47cbd668-t6sgs  1/1     Running   0          10m
mongodb-deployment-5c6cb86f45-8lqqq     1/1     Running   0          4d21h
mysql-deployment-5496fdc956-994jm       1/1     Running   0          4d21h
web-deployment-6fdbb5c6bb-5zjvc         1/1     Running   0          33m
web-deployment-6fdbb5c6bb-pwp7f         1/1     Running   0          33m
bhutta_abdul@cloudshell:~$ kubectl get deployments
NAME                   READY   UP-TO-DATE   AVAILABLE   AGE
ambassador-deployment  2/2     2            2           2m6s
experiment-deployment  2/2     2            2           10m
mongodb-deployment     1/1     1            1           14d
mysql-deployment       1/1     1            1           21d
web-deployment         2/2     2            2           33m
bhutta_abdul@cloudshell:~$ kubectl get services
NAME                   TYPE           CLUSTER-IP    EXTERNAL-IP     PORT(S)          AGE
ambassador-deployment  LoadBalancer   10.84.6.158   34.171.252.47   80:32216/TCP     91s
experiment-deployment  ClusterIP      10.84.3.33    <none>          80/TCP           11m
kubernetes             ClusterIP      10.84.0.1     <none>          443/TCP          23d
mongodb-service        LoadBalancer   10.84.8.48    35.226.185.35   3306:30647/TCP   14d
mysql-service          LoadBalancer   10.84.0.54    34.171.222.236  3306:30136/TCP   21d
web-deployment         ClusterIP      10.84.9.181   <none>          80/TCP           15m
bhutta_abdul@cloudshell:~$
```

Finally, check that all pods, deployments, and services are running with no issues. What commands should you use? What's the external IP address associated with the **ambassador-deployment** service?

5. Check pods are running -> **kubectl get pods**
6. Check deployments are running -> **kubectl get deployments**
7. Check services are running -> **kubectl get services**
8. External IP address to the ambassador deployment -> **34.171.252.47**

v. View the output

vi. Run the script to call the server many times and view the output.txt file.

```
bhutta_abdul@cloudshell:~$ for _ in {1..20}; do curl http://34.171.252.47 -s > output.txt; done
bhutta_abdul@cloudshell:~$
bhutta_abdul@cloudshell:~$
```

output.txt

```
<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="M41.9,11.368A11.929,11.929,0,0,0,20.3,5.061a9.444,9.444,0,0,0-14.932,9.8A8.969,8.969,0,0,0,0,9.064,32H39.442A10.463,10.463,0,0,0,41.9,11.368Z" transform="t
    <path d="M41.9,11.368A11.929,11.929,0,0,0,20.3,5.061a9.444,9.444,0,0,0-14.932,9.8A8.969,8.969,0,0,0,0,9.064,32H39.442A10.463,10.463,0,0,0,41.9,11.368Z" transform=
    <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="M33,25H15V47H35V25ZM21,45H17V27h4Zm6,0H23V27h4Zm6,0H29V27h4Z" transform="translate(-0.1 -0.1)" fill="#b92025" opacity="0.3" style="isolation:isolate"/>
    <path d="M33,25H15V47H35V25ZM21,45H17V27h4Zm6,0H23V27h4Zm6,0H29V27h4Z" transform="translate(-0.1 -0.1)" fill="#fff" style="isolation:isolate"/>
</svg>
</div>
```

vii. Read logs to view how many times each server was called.

```
bhutta_abdul@cloudshell:~$ kubectl logs -l run=web-deployment
::ffff:10.80.3.6 - - [07/Oct/2022:17:59:14 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.80.3.6 - - [07/Oct/2022:17:59:14 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.80.3.6 - - [07/Oct/2022:17:59:14 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.80.2.10 - - [07/Oct/2022:17:59:14 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.80.3.6 - - [07/Oct/2022:17:59:14 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.80.3.6 - - [07/Oct/2022:17:59:14 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.80.2.10 - - [07/Oct/2022:17:59:15 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.80.3.6 - - [07/Oct/2022:17:59:15 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.80.3.6 - - [07/Oct/2022:17:59:15 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.80.2.10 - - [07/Oct/2022:17:59:15 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.80.3.6 - - [07/Oct/2022:17:49:59 +0000] "POST / HTTP/1.0" 404 140 "-" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit
::ffff:10.80.3.6 - - [07/Oct/2022:17:51:46 +0000] "GET /favicon.ico HTTP/1.0" 404 150 "http://34.171.252.47/" "Mozilla/5.0 (
ri/605.1.15"
::ffff:10.80.3.6 - - [07/Oct/2022:17:52:26 +0000] "GET / HTTP/1.0" 200 1663 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64; r
::ffff:10.80.2.10 - - [07/Oct/2022:17:59:14 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.80.2.10 - - [07/Oct/2022:17:59:14 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.80.2.10 - - [07/Oct/2022:17:59:14 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.80.3.6 - - [07/Oct/2022:17:59:15 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.80.3.6 - - [07/Oct/2022:17:59:15 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.80.3.6 - - [07/Oct/2022:17:59:15 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.80.3.6 - - [07/Oct/2022:17:59:15 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
bhutta_abdul@cloudshell:~$
bhutta_abdul@cloudshell:~$
 bhutta_abdul@cloudshell:~$ kubectl logs -l run=experiment-deployment
 listening on port 80
 ::ffff:10.80.2.10 - - [07/Oct/2022:17:59:15 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
 listening on port 80
 ::ffff:10.80.3.6 - - [07/Oct/2022:17:59:14 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
 ::ffff:10.80.3.6 - - [07/Oct/2022:17:59:15 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
```

3.0 Part 3

i. Create the deployments.

```
bhutta_abdul@cloudshell:~$ kubectl create -f loadbalancer-deployment.yaml
deployment.apps/loadbalancer-deployment created
bhutta_abdul@cloudshell:~$
```

ii. Check the status of the created pods.

```
bhutta_abdul@cloudshell:~$ kubectl get pods --output=wide
NAME                                      READY   STATUS    RESTARTS   AGE     IP           NODE                                                NOMINATED NODE   READINESS GATES
ambassador-deployment-66db4f7766-br4vq    1/1     Running   0          46m     10.80.2.10   gke-openfaas-default-pool-98fbf41c-xgnr             <none>           <none>
ambassador-deployment-66db4f7766-nmhzk    1/1     Running   0          46m     10.80.3.6    gke-openfaas-default-pool-98fbf41c-07wg             <none>           <none>
experiment-deployment-7b47cbd668-jt4b4    1/1     Running   0          55m     10.80.2.9    gke-openfaas-default-pool-98fbf41c-xgnr             <none>           <none>
experiment-deployment-7b47cbd668-t6sgs    1/1     Running   0          55m     10.80.1.5    gke-openfaas-default-pool-98fbf41c-l7x6             <none>           <none>
loadbalancer-deployment-6676f9ccf6-59ct7  1/1     Running   0          46s     10.80.1.6    gke-openfaas-default-pool-98fbf41c-l7x6             <none>           <none>
loadbalancer-deployment-6676f9ccf6-7q7k4  1/1     Running   0          46s     10.80.2.11   gke-openfaas-default-pool-98fbf41c-xgnr             <none>           <none>
loadbalancer-deployment-6676f9ccf6-gt712  1/1     Running   0          46s     10.80.3.7    gke-openfaas-default-pool-98fbf41c-07wg             <none>           <none>
mongodb-deployment-5c6cb86f45-81qqq       1/1     Running   0          4d22h   10.80.3.3    gke-openfaas-default-pool-98fbf41c-07wg             <none>           <none>
mysql-deployment-5496fdc956-994jm         1/1     Running   0          4d22h   10.80.2.2    gke-openfaas-default-pool-98fbf41c-xgnr             <none>           <none>
web-deployment-6fdbb5c6bb-5zjvc           1/1     Running   0          78m     10.80.2.8    gke-openfaas-default-pool-98fbf41c-xgnr             <none>           <none>
web-deployment-6fdbb5c6bb-pwp7f           1/1     Running   0          78m     10.80.1.4    gke-openfaas-default-pool-98fbf41c-l7x6             <none>           <none>
bhutta_abdul@cloudshell:~$
```

iii. Associating a load balancer.

```
bhutta_abdul@cloudshell:~$ kubectl expose deployment loadbalancer-deployment --port=8080 --type=LoadBalancer
service/loadbalancer-deployment exposed
bhutta_abdul@cloudshell:~$
```

iv. Get external IP of the load balancer.

```
bhutta_abdul@cloudshell:~$ kubectl get services --watch
NAME                        TYPE           CLUSTER-IP     EXTERNAL-IP      PORT(S)          AGE
ambassador-deployment       LoadBalancer   10.84.6.158    34.171.252.47    80:32216/TCP     50m
experiment-deployment       ClusterIP      10.84.3.33     <none>           80/TCP           59m
kubernetes                  ClusterIP      10.84.0.1      <none>           443/TCP          24d
loadbalancer-deployment     LoadBalancer   10.84.7.236    35.232.122.111   8080:30458/TCP   58s
mongodb-service             LoadBalancer   10.84.8.48     35.226.185.35    3306:30647/TCP   14d
mysql-service               LoadBalancer   10.84.0.54     34.171.222.236   3306:30136/TCP   21d
web-deployment              ClusterIP      10.84.9.181    <none>           80/TCP           64m
```

v. Running the dictionary application through the replicated load-balancing service.

```
bhutta_abdul@cloudshell:~$ curl http://35.232.122.111:8080/dog
A quadruped of the genus Canis, esp. the domestic dog (C.familiaris).bhutta_abdul@cloudshell:~$ curl http://35.232.122.111:8080/storey
See Story.bhutta_abdul@cloudshell:~$
bhutta_abdul@cloudshell:~$ curl http://35.232.122.111:8080/story
A set of rooms on the same floor or level; a floor, or thespace between two floors. Also, a horizontal division of a building'sexterior cons
idered architecturally, which need not correspondexactly with the stories within. [Written also storey.]bhutta_abdul@cloudshell:~$
```


```
See Story.bhutta_abdul@cloudshell:~$ curl http://35.232.122.111:8080/story
A set of rooms on the same floor or level; a floor, or thespace between two floors. Also, a horizontal division of a building'sexterior considered architecturally, which need not correspondexactly with the stories wit
hin. [Written also storey.]bhutta_abdul@cloudshell:~$
bhutta_abdul@cloudshell:~$ curl http://35.232.122.111:8080/dog
A quadruped of the genus Canis, esp. the domestic dog (C.familiaris).bhutta_abdul@cloudshell:~$
```


4.0 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?

A gateway routing pattern can route multiple services or instances to a single endpoint by placing a gateway in front of the services. This solves the issue with multiple disparate services, multiple instances of the same service, and multiple versions of the same service. The solution allows the request from the client to reach the gateway, which routes to the appropriate instance or services. In part 2, the ambassador deployment is used as a gateway to split the traffic between the web server and the experiment server. The client will communicate with a single endpoint while the gateway will manage the routes. This is an example of multiple versions of the same service. A new pod should not receive traffic until it is entirely up, which is why the readiness probe is used in part 3 to wait until the application is fully up and running before sending the traffic every 5 seconds. The readiness probe acts as a gateway before routing the traffic.

5.0 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 from load balancing and request splitter?

Autoscaling is helpful as it can help the user manage the resource load, such as using more resources during peak hours and less during downtime. Also, it gives the user more availability in case of a hardware or application failure. Autoscaling in google cloud is implemented using Kubernetes, where you apply the autoscaling option to the deployment. You must specify the resource utilization target and the minimum and maximum pods. Autoscaling allows the user to scale the resources at a lower cost. In contrast, load balancing is used to distribute the server workload for efficiency and performance. Autoscaling and load balancer work together to provide efficiency and performance. A request splitter will always split the traffic based on the user's specification to another server. In part 2, we sent 90% of the traffic to the main server while sending 10% of the traffic to the experimental server. This allows the user to conduct testing on the experimental server between versions before deploying them. In this lab, to demonstrate autoscaling, we will use a deployment which runs a container with the image "autoscaling-example" and use that as a service. The YAML file deployed was found through the Kubernetes documentation website, which runs a container of the image "hpa-example" and uses it as a service. The PHP-Apache server will listen to incoming requests. The horizontal pod scaler was integrated to manage the CPU utilization to 50 percent while allowing for autoscaling to increase or decrease the number of pods (1 – 10). Multiple requests were sent to the server through a different terminal infinitely while monitoring the horizontal scaling done by Kubernetes. Once the requests were stopped, the auto-scaling took the replicas back to 1 as CPU utilization was 0. An example of autoscaling is shown below.

- It allows the service to have greater availability, reliability and elasticity, even when there are hardware failures. With auto scaling a more optimized application environment is achieved through a dynamic hardware environment

i. Create deployment and service of the YAML file.

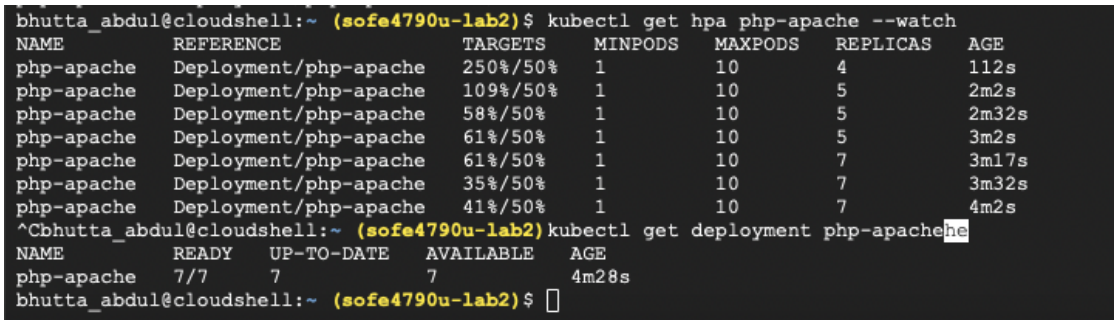
```
bhutta_abdul@cloudshell:~ (sofe4790u-lab2)$ kubectl apply -f https://k8s.io/examples/application/php-apache.yaml
deployment.apps/php-apache created
service/php-apache created
bhutta_abdul@cloudshell:~ (sofe4790u-lab2)$
```

ii. Create an auto scaler to maintain CPU utilization at 40% and balance the load between 1 and 10 pods.

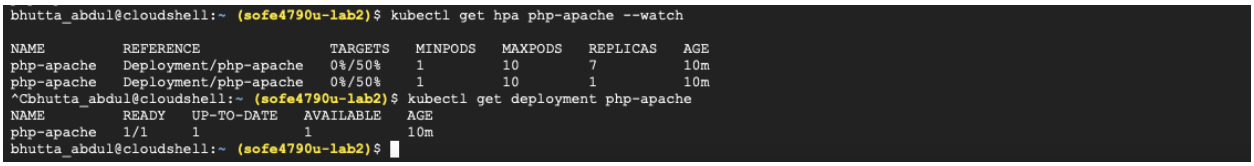
```
bhutta_abdul@cloudshell:~ (sofe4790u-lab2)$ kubectl autoscale deployment php-apache --cpu-percent=60 --min=1 --max=10
horizontalpodautoscaler.autoscaling/php-apache autoscaled
bhutta_abdul@cloudshell:~ (sofe4790u-lab2)$
bhutta_abdul@cloudshell:~ (sofe4790u-lab2)$ kubectl get hpa
NAME         REFERENCE                     TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
php-apache   Deployment/php-apache   0%/50%    1         10        1          34s
```

iii. Send unlimited requests to the PHP server, which will increase the workload on the CPU.

iv. Check-in real-time the usage of the CPU and the autoscaling.



v. Kubernetes scaling down will bring the replica's pod back down to 1 as the CPU utilization returns to 0 because the infinite requests were stopped.



Homework 2: Network and Virtualization
Member: Owen Musselman

Note: Changes made after group discussion are highlighted in red.

Part 1.

Objectives:

- Learn how to configure and run a request splitter using Nginx.
- Be familiar with the ConfigMap tool in Kubernetes.
- Learn how to use the curl commands for requesting an HTTP method.
- Learn how to configure load balancer services.
- Get familiar with load balancing pattern.

Problem:

If a client needs to utilize/take multiple services/instances, the client must then be updated in conjunction with addition or removal of services. E-commerce for example, needs multiple services like checking out, searching for an item, or adding items to a cart. If a new service is added, like order history for example, it needs to be updated on the client's side as well.

- *Multiple disparate services*: Each service has a different API in which the client interacts with, and this client must know each endpoint to connect and communicate with the services. If an API were to change, each of the clients would need to be updated as well. Additionally, if a service were to be refactored into multiple services, the code needs to change both within the service and the client.
- *Multiple instances of the same services*: System may require to have multiple instances of the same service running that are a different or the same region. Having various instances can be done to meet any availability requirements, or for load balancing means. Any time that an instance is increased or decreased (in instances) in order to meet demand, the client needs to be updated accordingly.
- *Multiple versions of the same service*: New versions of services are able to be deployed with pre existing versions. This is called "blue and green deployments." Here, the client needs to be updated whenever there are booms or busts in the traffic directed to the new version and the existing endpoints.

Solution:

The above problem is solved by placing a gateway in front of a set of applications, or deployments. Layer 7 (the application layer) is used to route the requests to the appropriate instances. This pattern allows for the client to only be aware of one endpoint and can communicate with this single endpoint.

Requirements:

- Gateway routing pattern
- Elasticity
- Geode pattern
- Availability
- Latency
- Reliability
- Greater than 1 or 2 services

Part 2.

1. Check if pods are running: kubectl get pods
2. Check if deployments are running: kubectl get deployments
3. Check if services are running: kubectl get services
4. Getting the externa IP: 34.152.63.218

NOTE: Using the provided yaml files via the github for parts 2 and 3.

```
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$ kubectl create -f web-deployment.yaml
deployment.apps/web-deployment created
```

Creating the web deployment.

```
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$ kubectl expose deployment web-deployment --port 80 --type=Cluste
rIP --name web-deployment
service/web-deployment exposed
```

Creating a clusterIP, and exposing the web-deployment.

```
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$ kubectl create -f experiment-deployment.yaml
deployment.apps/experiment-deployment created
```

Creating experiment deployment.

```
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$ kubectl get deployments
NAME                   READY   UP-TO-DATE   AVAILABLE   AGE
experiment-deployment  2/2     2            2           24s
mongodb-deployment     1/1     1            1           18d
mysql-deployment       1/1     1            1           23d
web-deployment         2/2     2            2           4m29s
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$
```

Viewing the deployments created.

```
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$ kubectl expose deployment experiment-deployment --port=80 --type
=ClusterIP --name experiment-deployment
service/experiment-deployment exposed
```

ClusterIP for the experiment deployment.

```
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$ kubectl create configmap ambassador-config --from-file=conf.d
configmap/ambassador-config created
```

Generating the config map using nginx-ambassador.conf that is in conf.d folder.

```
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$ kubectl create -f ambassador-deployment.yaml
deployment.apps/ambassador-deployment created
```

Creating ambassador-deployment deployment.

```
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$ kubectl expose deployment ambassador-deployment --port=80 --type
=LoadBalancer
service/ambassador-deployment exposed
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$
```

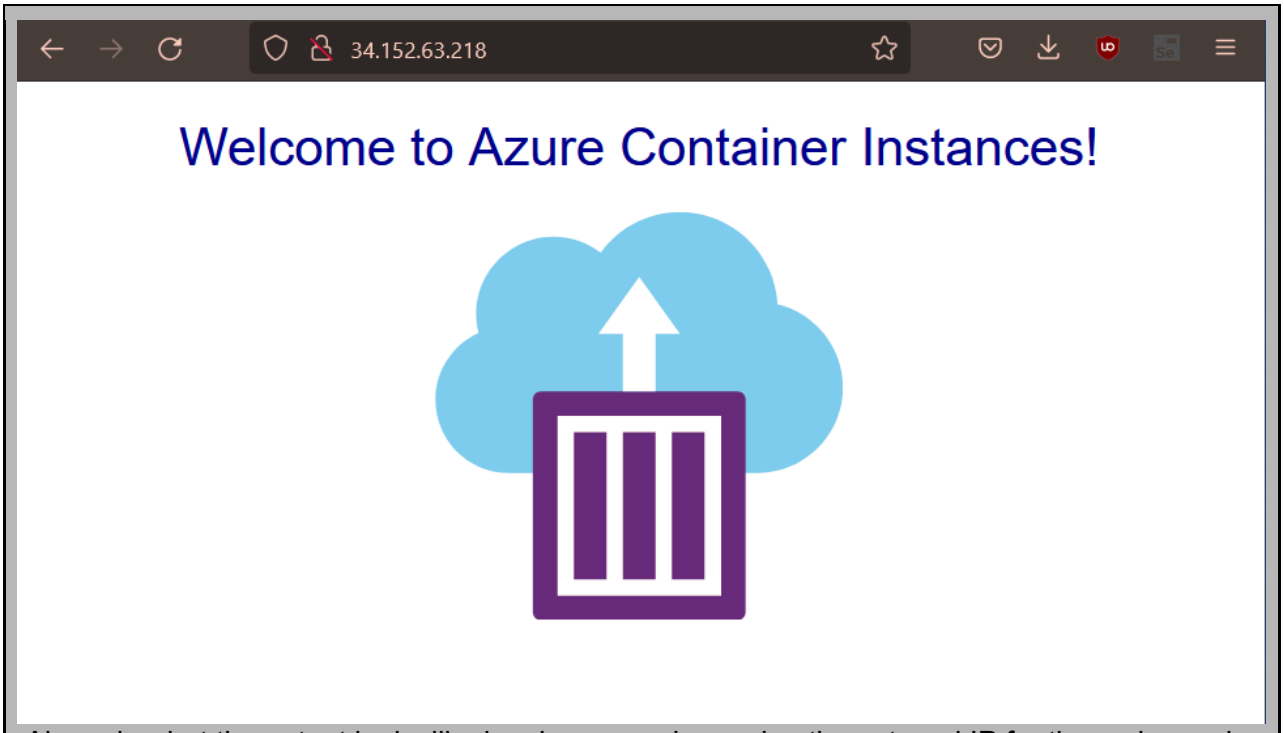
Assign a load balancer to the ambassador deployment.

```
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$ kubectl get pods
NAME                                    READY   STATUS    RESTARTS   AGE
ambassador-deployment-66db4f7766-pjww8  1/1     Running   0          2m50s
ambassador-deployment-66db4f7766-skhqq  1/1     Running   0          2m50s
experiment-deployment-7b47cbd668-5f4gk  1/1     Running   0          14m
experiment-deployment-7b47cbd668-ngkhk  1/1     Running   0          14m
mongodb-deployment-7945646c67-r94d6     1/1     Running   0          3d12h
mysql-deployment-5496fdc956-8g9rw       1/1     Running   0          3d12h
web-deployment-6fdbb5c6bb-6fmvm         1/1     Running   0          18m
web-deployment-6fdbb5c6bb-phd4f         1/1     Running   0          18m
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$ kubectl get services
NAME                    TYPE           CLUSTER-IP      EXTERNAL-IP     PORT(S)          AGE
ambassador-deployment   LoadBalancer   10.108.4.66     34.152.63.218   80:32688/TCP     78s
experiment-deployment   ClusterIP      10.108.0.252    <none>          80/TCP           11m
kubernetes              ClusterIP      10.108.0.1      <none>          443/TCP          25d
mongodb-service         LoadBalancer   10.108.13.162   35.234.248.86   3306:30766/TCP   18d
mysql-service           LoadBalancer   10.108.8.33     34.152.15.114   3306:31301/TCP   23d
web-deployment          ClusterIP      10.108.10.177   <none>          80/TCP           18m
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$
```

Checking to see if the pods are running and then finding the external IP of the ambassador deployment which is 34.152.63.218 in this case.

```
C:\Users\SLIKC>curl http://34.152.63.218
<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="M41.9,11.368A11.929,11.929,0,0,0,20.3,5.061a9.444,9.444,0,0,0-14.932,9.8A8.969,8.969,0,0,0,0,9.064,32H39.442A10.463,10.463,0,0,0,41.9,11.368Z" t
ransform="translate(-0.1 -0.1)" fill="#fff"/>
        <path d="M41.9,11.368A11.929,11.929,0,0,0,20.3,5.061a9.444,9.444,0,0,0-14.932,9.8A8.969,8.969,0,0,0,0,9.064,32H39.442A10.463,10.463,0,0,0,41.9,11.3
68Z" 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="M33,25H15V47H35V25ZM21,45H17V27h4Zm6,0H23V27h4Zm6,0H29V27h4Z" transform="translate(-0.1 -0.1)" fill="#b92025" opacity="0.3" style="isola
tion:isolate"/>
        <path d="M33,25H15V47H35V25ZM21,45H17V27h4Zm6,0H23V27h4Zm6,0H29V27h4Z" transform="translate(-0.1 -0.1)" fill="#fff" style="isolation:isolate"/>
</svg>
</div>

</body>
</html>
```

Using curl to see the output of the nginx load balancing implementation.



Above is what the output looks like in a browser when using the external IP for the ambassador service.

```
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$ for i in {1..20}; do curl http://34.152.63
218 -s > output.txt; done

slikcaustic1@cloudshell:~ (bubbly-granite-362015)$
```

The for loop to run curl command 20 times and then write the output to output.txt.

output.txt

```
<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,
    <path d="M41.9,11.368A11.929,11.929,0,0,0,20.3,5.061a9.444,9.444,0,0,0-14.932,9.8A8.969,8.969,0,0,0,9.064,32H39.442A10.463,10.46
    <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="#672a7
    <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"/>
```

After the for loop is run and the output file contains the above contents.

```
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$ kubectl logs -l run=web-deployment
::ffff:10.104.0.6 - - [05/Oct/2022:16:47:35 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0
::ffff:10.104.0.6 - - [05/Oct/2022:16:47:35 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0
::ffff:10.104.1.11 - - [05/Oct/2022:16:47:35 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.
"
::ffff:10.104.1.11 - - [05/Oct/2022:16:48:34 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.
"
::ffff:10.104.0.6 - - [05/Oct/2022:16:48:34 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0
::ffff:10.104.0.6 - - [05/Oct/2022:16:48:34 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0
::ffff:10.104.0.6 - - [05/Oct/2022:16:48:34 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0
::ffff:10.104.1.11 - - [05/Oct/2022:16:48:34 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.
"
::ffff:10.104.0.6 - - [05/Oct/2022:16:48:34 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0
::ffff:10.104.0.6 - - [05/Oct/2022:16:48:34 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0
::ffff:10.104.1.11 - - [05/Oct/2022:16:48:34 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.
"
::ffff:10.104.0.6 - - [05/Oct/2022:16:48:34 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0
::ffff:10.104.0.6 - - [05/Oct/2022:16:48:34 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0
::ffff:10.104.1.11 - - [05/Oct/2022:16:48:34 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.
"
::ffff:10.104.1.11 - - [05/Oct/2022:16:48:34 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.
"
::ffff:10.104.0.6 - - [05/Oct/2022:16:48:34 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0
::ffff:10.104.0.6 - - [05/Oct/2022:16:48:34 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0
::ffff:10.104.0.6 - - [05/Oct/2022:16:48:35 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0
::ffff:10.104.1.11 - - [05/Oct/2022:16:48:35 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.
"
::ffff:10.104.1.11 - - [05/Oct/2022:16:48:35 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.
"
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$
```

Output of the kubectl logs -l run=web-deployment. 90% of the traffic directed to the web-deployment.

```
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$ kubectl logs -l run=experiment-deployment
listening on port 80
::ffff:10.104.0.6 - - [05/Oct/2022:02:40:29 +0000] "GET /idx_config/ HTTP/1.0" 404 150 "-" "l
explore/1.3.0"
::ffff:10.104.0.6 - - [05/Oct/2022:09:03:55 +0000] "GET /.well-known/security.txt HTTP/1.0" 4
4 163 "-" "-"
::ffff:10.104.1.11 - - [05/Oct/2022:11:13:21 +0000] "HEAD /robots.txt HTTP/1.0" 404 150 "-" "
"
::ffff:10.104.0.6 - - [05/Oct/2022:16:47:34 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0
::ffff:10.104.0.6 - - [05/Oct/2022:16:48:33 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0
::ffff:10.104.0.6 - - [05/Oct/2022:16:48:34 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0
listening on port 80
::ffff:10.104.1.11 - - [05/Oct/2022:02:40:29 +0000] "GET /s/lkx/_/;/META-INF/maven/com.atlass
an.jira/jira-webapp-dist/pom.properties HTTP/1.0" 404 214 "-" "l9explore/1.3.0"
::ffff:10.104.1.11 - - [05/Oct/2022:08:12:53 +0000] "GET / HTTP/1.0" 200 1663 "-" "-"
::ffff:10.104.1.11 - - [05/Oct/2022:16:29:32 +0000] "GET / HTTP/1.0" 200 1663 "-" "Mozilla/5.
 (Windows NT 10.0; Win64; x64; rv:105.0) Gecko/20100101 Firefox/105.0"
::ffff:10.104.1.11 - - [05/Oct/2022:16:47:35 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.
"
::ffff:10.104.1.11 - - [05/Oct/2022:16:48:35 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.
"
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$
```

Output of the kubectl logs -l run=experiment-deployment. Can see how it was accessed by curl and my Mozilla Firefox browser. 10% of the requests are sent here to the experiment deployment.

Part 3:

```
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$ kubectl create -f loadbalancer-deployment.
aml
deployment.apps/loadbalancer-deployment created
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$
```

Creating the load balancer deployment using the given yaml file.

```
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$ kubectl get pods --output=wide
NAME                                     READY   STATUS    RESTARTS   AGE    IP
ODE                                       NOMINATED NODE   READINESS GATES
ambassador-deployment-66db4f7766-pjww8   1/1     Running   0          14h    10.104.1.11
ke-testcluster-default-pool-565f249a-9lyo  <none>           <none>
ambassador-deployment-66db4f7766-skhqq   1/1     Running   0          14h    10.104.0.6
ke-testcluster-default-pool-565f249a-u5l2  <none>           <none>
experiment-deployment-7b47cbd668-5f4gk   1/1     Running   0          14h    10.104.2.6
ke-testcluster-default-pool-565f249a-iga2  <none>           <none>
experiment-deployment-7b47cbd668-ngkhk   1/1     Running   0          14h    10.104.1.10
ke-testcluster-default-pool-565f249a-9lyo  <none>           <none>
loadbalancer-deployment-6676f9ccf6-2q6js  1/1     Running   0          94s    10.104.0.7
ke-testcluster-default-pool-565f249a-u5l2  <none>           <none>
loadbalancer-deployment-6676f9ccf6-7tqpl  1/1     Running   0          94s    10.104.2.7
ke-testcluster-default-pool-565f249a-iga2  <none>           <none>
loadbalancer-deployment-6676f9ccf6-whkc8  1/1     Running   0          94s    10.104.1.12
ke-testcluster-default-pool-565f249a-9lyo  <none>           <none>
mongodb-deployment-7945646c67-r94d6      1/1     Running   0          4d2h   10.104.1.2
ke-testcluster-default-pool-565f249a-9lyo  <none>           <none>
mysql-deployment-5496fdc956-8g9rw        1/1     Running   0          4d2h   10.104.0.2
ke-testcluster-default-pool-565f249a-u5l2  <none>           <none>
web-deployment-6fdbb5c6bb-6fmvm          1/1     Running   0          14h    10.104.1.9
ke-testcluster-default-pool-565f249a-9lyo  <none>           <none>
web-deployment-6fdbb5c6bb-phd4f          1/1     Running   0          14h    10.104.2.5
ke-testcluster-default-pool-565f249a-iga2  <none>           <none>
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$
```

After executing the kubectl get pods –output=wide there are 3 replicas highlighted.

```
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$ kubectl expose deployment loadbalancer-dep
oyment --port=8080 --type=LoadBalancer
service/loadbalancer-deployment exposed
```

Now the load balancer service has been exposed.

```
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$ curl http://35.203.71.168:8080/story


A set of rooms on the same floor or level; a floor, or thespace between two floors. Also, a h
rizontal division of a building'sexterior considered architecturally, which need not correspo
dexactly with the stories within. [Written also storey.]slikcaustic1@cloudshell:~ (bubbly-gsl
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$
```

Confirmation that the load balancer service is working as using the external IP to the server allows us to connect, query, and get output from the input dictionary word.

Discussion:

For part one, the problem is that if a client needs to use multiple services, or multiple service instances (or both), then the client needs to update accordingly with the addition or removal of services. The solution to this problem is to utilize a gateway that is in front of various applications, this directs the incoming requests to the proper instances.

For request splitting this problem (part 2) is solved in part three through the ambassador-deployment.yaml, nginx-ambassador.conf, with the command: kubectl create configmap ambassador-config --from-file=conf.d, which creates the configmap, and kubectl create -f ambassador-deployment.yaml, to create the deployment. The deployment is then exposed to allow access. The nginx-ambassador.conf sets the experiment-deployment to get 10% of the incoming requests, and the other 90% is directed to the web-deployment.

In part 3 the procedure guides us through load balancing, where the load balancer distributes the load between the three pods which is done through the readinessProbe that is within the loadbalancer-deployment.yaml file in which the traffic will not be sent until everything is running and then it is checked every 5 seconds.

Design:

Autoscaling is used to ensure that your desired application is operating at the your required performance level. Autoscaling also assists having your application operate in a more resource optimized environment. This is done by scaling up the resources used for your application when traffic requests are higher, and scaling down the resources when traffic requests are lower.

Horizontal pod scaling is implemented with kubernetes, where the resources you are scaling are specified, which could be memory or in this case, cpu usage. There is an upper bound on how much cpu cores you want to use for the pods, which is done through limiting the millicores the replica can use. There is also a lower bound set by requesting millicores for the workload. For the implementation that was done here an example [1] from the kubernetes about horizontal pod autoscaling was used. This implementation of auto scaling uses a command that sets the minimum and maximum replicas to be made for matching the working load, along with the setting what the average cpu usage should be across the pods. An infinite loop doing some querying and displaying an “OK” message was used too in order to see if the scaling capabilities were working with respect to the workload. In this implementation, a php apache server listens for requests. The average cpu usage across the pods is set for 50%, the minimum replicas is set to 1 and the maximum replicas is set to 10. With a higher work load the cpu usage will increase, and the replicas will increase with it. If the workload is lower the cpu usage will decrease and the replicas will decrease with it too, reflecting the workload.

Auto scaling and load balancing/request splitting as with autoscaling the resources being used are being scaled up or scaled down to match the usage of the application/service. While load balancing is responsible for distributing the traffic coming into various other locations to serve them.

Implementation of horizontal scaling:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hpa-experiment
spec:
  selector:
    matchLabels:
      run: hpa-experiment
  replicas: 3 # Start with 3 replicas initially.
  template:
    metadata:
      labels:
        run: hpa-experiment
    spec:
      containers:
      - name: hpa-experiment
        image: k8s.gcr.io/hpa-example # Utilize example image for horizontal pod autoscaling.
        ports:
        - containerPort: 80 # Set the port to 80.
        resources:
          limits:
            cpu: 500m # Limits the container to use half of a cpu core for the application. A full core is would be cpu: 1000m where m is millicores.
          requests:
            cpu: 200m # Requests the minimum compute capacity. In this case set to 200millicores.
---
  # Service information.
apiVersion: v1
kind: Service
metadata:
  name: hpa-experiment
  labels:
    run: hpa-experiment
spec:
  ports:
  - port: 80
  selector:
    run: hpa-experiment
```

Yaml file that holds the deployment and service information for the horizontal pod autoscaler. Initially start with 3 replicas, 500millicores of cpu, with a minimum of 200 millicores of processing power for the application.

```
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$ kubectl apply -f horizontal-experiment.yaml
deployment.apps/hpa-experiment created
service/hpa-experiment created
```

Create the deployment(s) and service(s) that will be autoscaled.

```
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$ kubectl autoscale deployment hpa-experiment --cpu-percent=50 --min=1 --max=10
horizontalpodautoscaler.autoscaling/hpa-experiment autoscaled
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$ kubectl get hpa
NAME             REFERENCE                   TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
hpa-experiment   Deployment/hpa-experiment   0%/50%    1         10        3          55s
```

Apply autoscaling to the hpa-experiment deployment, with the aim of having an average of 50% cpu usage over all the current pods. The minimum replicas of the pods is set to 1 and the max for this test is set to 10 replicas. The current status of the workload is 0% out of 50% which is accurate as there is currently no work being done. Additionally, there are currently 3 replicas.

```
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$ kubectl get hpa
NAME             REFERENCE                   TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
hpa-experiment   Deployment/hpa-experiment   0%/50%    1         10        3          55s
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$ kubectl get hpa
NAME             REFERENCE                   TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
hpa-experiment   Deployment/hpa-experiment   0%/50%    1         10        1          10m
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$
```

Now, after 10 minutes, you can now see that since there has been no work load the replicas have been scaled down, as intended.

```
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$ kubectl run -i --tty load-generator --rm --image=busybox --restart=Never -- /bin/sh -c "while sleep 0.01; do wget -q -O- http://hpa-experiment; done"
If you don't see a command prompt, try pressing enter.
OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!
```

Create a workload by querying the hpa-experiment service infinitely. After a while, the replicas should increase accordingly.

```
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$ kubectl get hpa
NAME             REFERENCE                   TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
hpa-experiment   Deployment/hpa-experiment   0%/50%    1         10        3          55s
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$ kubectl get hpa
NAME             REFERENCE                   TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$ kubectl get hpa
NAME             REFERENCE                   TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
hpa-experiment   Deployment/hpa-experiment   44%/50%   1         10        8          25m
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$
```

After a few minutes of executing the replicas have been increased to 8 to fit the workload, and the average cpu usage has been increased from 0% to 44% now as well.

Conclusion:

After completing the steps for this lab, we have now learned how to configure and run a simple request splitter utilizing nginx. An increased familiarity with the ConfigMap tool within kubernetes. Additionally, we have a better understanding of using simple curl commands after completing the steps. Another configuration that is now understood is load balancing services and load balancing patterns. While doing the design section of this lab some struggles were finding resources on horizontal auto scaling were applicable to the task provided, but was achievable. And now due to looking at many resources about horizontal auto scaling in kubernetes, we can confidently say we have a better understanding of horizontal auto scaling, and how to implement it into projects.

Works Cited

[1] “Horizontalpodautoscaler walkthrough,” *Kubernetes*, 06-Aug-2022. [Online]. Available: https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale-walkthrough/. [Accessed: 07-Oct-2022].

Homework 2: Network and Virtualization
Member: Alexander Campbell

Part 1:

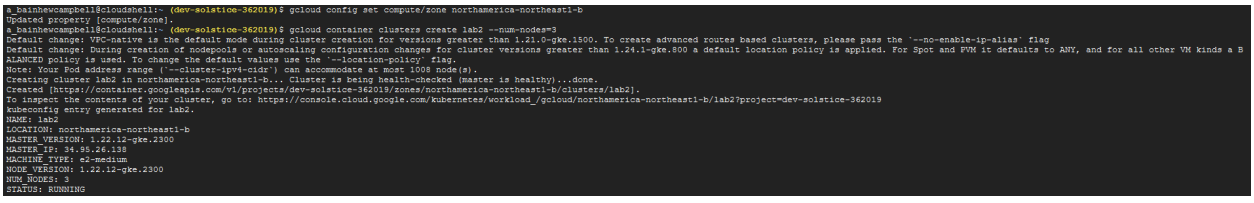
The Gateway Routing pattern is a pattern where a single endpoint can be used to serve multiple services or multiple requests. This is done by placing a gateway in front of your application so that the connected client is only aware of the single endpoint occupied by the gateway. This is useful for three main reasons: exposing several versions of one service, having multiple instances of the same service, or when there's multiple services on one end point.

We can explore the usefulness by taking an e-commerce store as an example. Say a product owner would like to do some AB Testing for a new user interface, you may change the user interface on one version of the service to test if it increases sales by exposing the two versions of the service on one end point for your traffic to use. The gateway would then automatically distribute clients to each version of the website. Having multiple instances is also very useful for e-commerce as there are many regional differences you may want to consider such as high traffic times changing depending on the time zone. In this case, we would want to balance the load to ensure that servers located in high traffic areas have the appropriate amount of availability, while low traffic areas can be decreased and save on computational costs. The gateway would automatically redirect clients towards the endpoint that best suits their region. E-commerce stores also have many services such as search or checkout. Using this pattern, we can expose each end point so that the client is aware of each of the services available to them through the gateway.

The problems discussed in this can be boiled down to the fact that clients need to be easily redirected to different services without them having to have knowledge about the backend of the application. To utilize this pattern properly one must ensure that the gateway is correctly designed as otherwise it can lead to failure or a bottleneck of the entire system.

Part 2:

First the pods were initialized, then the web, experiment, and ambassador deployment yaml files were created, deployed, and exposed successfully



web-deployment.yaml | experiment-deployment.yaml | nginx-ambassado

experiment-deployment.yaml > ...

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: experiment-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      run: experiment-deployment
  template:
    metadata:
      labels:
        run: experiment-deployment
    spec:
      containers:
      - name: experiment-depoyment
        image: mcr.microsoft.com/azuredocs/aci-helloworld
        ports:
        - containerPort: 80
```

web-deployment.yaml | experiment-deployment.yaml | nginx-ambassad

web-deployment.yaml > ...

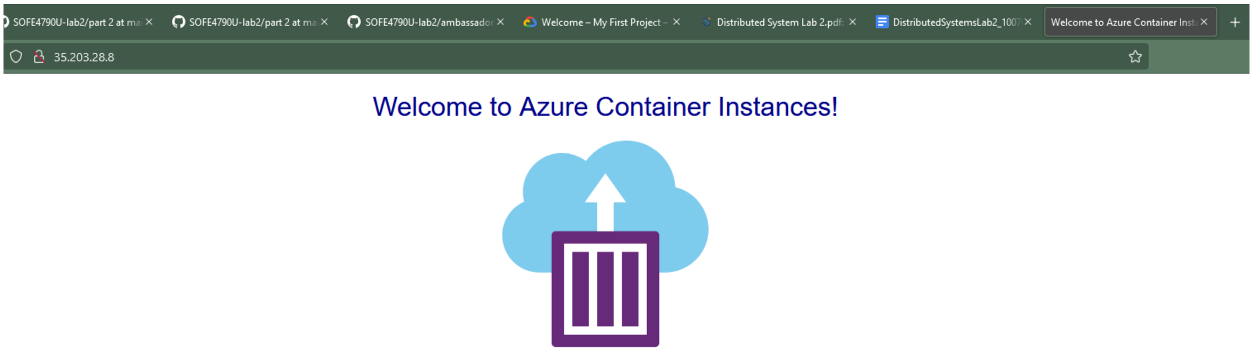
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      run: web-deployment
  template:
    metadata:
      labels:
        run: web-deployment
    spec:
      containers:
      - name: web-depoyment
        image: mcr.microsoft.com/azuredocs/aci-helloworld
        ports:
        - containerPort: 80
```

```
a_bainhewcampbell@cloudshell:~ (dev-solstice-362019)$ kubectl create -f web-deployment.yaml
deployment.apps/web-deployment created
a_bainhewcampbell@cloudshell:~ (dev-solstice-362019)$ kubectl get pods
NAME                              READY   STATUS    RESTARTS   AGE
web-deployment-6fdbb5c6bb-752fj   1/1     Running   0          10s
web-deployment-6fdbb5c6bb-nfdf9   1/1     Running   0          10s
a_bainhewcampbell@cloudshell:~ (dev-solstice-362019)$ kubectl expose deployment web-deployment --port=80 --type=ClusterIP --name web-deployment
service/web-deployment exposed
a_bainhewcampbell@cloudshell:~ (dev-solstice-362019)$ kubectl create -f experiment-deployment.yaml
deployment.apps/experiment-deployment created
a_bainhewcampbell@cloudshell:~ (dev-solstice-362019)$ kubectl expose deployment experiment-deployment --port=80 --type=ClusterIP --name experiment-deployment
service/experiment-deployment exposed
a_bainhewcampbell@cloudshell:~ (dev-solstice-362019)$ kubectl create configmap ambassador-config --from-file=conf.d
configmap/ambassador-config created
a_bainhewcampbell@cloudshell:~ (dev-solstice-362019)$ kubectl create -f ambassador-deployment.yaml
deployment.apps/ambassador-deployment created
a_bainhewcampbell@cloudshell:~ (dev-solstice-362019)$ kubectl expose deployment ambassador-deployment --port=80 --type=LoadBalancer
service/ambassador-deployment exposed
```

Then the pods, deployments, and services were checked to ensure that they were running without any errors

```
a_bainhewcampbell@cloudshell:~ (dev-solstice-362019)$ kubectl get deployments
NAME                    READY   UP-TO-DATE   AVAILABLE   AGE
ambassador-deployment   2/2     2            2           22s
experiment-deployment   2/2     2            2           10m
web-deployment          2/2     2            2           11m
a_bainhewcampbell@cloudshell:~ (dev-solstice-362019)$ kubectl get pods
NAME                                    READY   STATUS    RESTARTS   AGE
ambassador-deployment-66db4f7766-f9rzv  1/1     Running   0          30s
ambassador-deployment-66db4f7766-xthpz  1/1     Running   0          30s
experiment-deployment-7b47cbd668-l81fq  1/1     Running   0          11m
experiment-deployment-7b47cbd668-nqdv9  1/1     Running   0          11m
web-deployment-6fdbb5c6bb-752fj         1/1     Running   0          12m
web-deployment-6fdbb5c6bb-nfdf9         1/1     Running   0          12m
a_bainhewcampbell@cloudshell:~ (dev-solstice-362019)$ kubectl get services
NAME                    TYPE           CLUSTER-IP     EXTERNAL-IP   PORT(S)        AGE
ambassador-deployment   LoadBalancer   10.12.6.8      35.203.28.8   80:31372/TCP   66s
experiment-deployment   ClusterIP      10.12.7.229    <none>        80/TCP         11m
kubernetes              ClusterIP      10.12.0.1      <none>        443/TCP        15m
web-deployment          ClusterIP      10.12.7.194    <none>        80/TCP         12m
```

After the pods, deployments, and services were confirmed to be running correctly I opened up the ambassador-deployment IP address in my browser to find the website running and then ran the curl command many times and viewed the logs.




```
a_bainhewcampbell@cloudshell:~ (dev-solstice-362019)$ curl http://35.203.28.8
<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="M33,25H15V47H35V25ZM21,45H17V27h4Zm6,0H23V27h4Zm6,0H29V27h4Z" transform="translate(-0.1 -0.1)" fill="#b92025" opacity="0.3" style="isolation:isolate"/>
        <path d="M33,25H15V47H35V25ZM21,45H17V27h4Zm6,0H23V27h4Zm6,0H29V27h4Z" transform="translate(-0.1 -0.1)" fill="#fff" style="isolation:isolate"/>
</svg>
</div>

</body>
</html>
a_bainhewcampbell@cloudshell:~ (dev-solstice-362019)$ curl http://35.203.28.8
<html>
<head>
  <title>Welcome to Azure Container Instances!</title>
</head>
<style>
h1 {
    color: darkblue;
    font-family:arial, sans-serif;
    font-weight: lighter;
}
</style>

<body>
```

```
a_bainhewcampbell@cloudshell:~ (dev-solstice-362019)$ kubectl logs -l run=web-deployment
::ffff:10.8.1.9 - - [05/Oct/2022:22:57:25 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.8.0.5 - - [05/Oct/2022:22:57:26 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.8.1.9 - - [05/Oct/2022:22:57:27 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.8.1.9 - - [05/Oct/2022:22:57:29 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.8.0.5 - - [05/Oct/2022:22:57:32 +0000] "GET / HTTP/1.0" 304 - "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:105.0) Gecko/20100101 Firefox/105.0"
::ffff:10.8.0.5 - - [05/Oct/2022:22:57:33 +0000] "GET / HTTP/1.0" 304 - "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:105.0) Gecko/20100101 Firefox/105.0"
::ffff:10.8.0.5 - - [05/Oct/2022:22:57:33 +0000] "GET / HTTP/1.0" 304 - "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:105.0) Gecko/20100101 Firefox/105.0"
::ffff:10.8.0.5 - - [05/Oct/2022:22:57:33 +0000] "GET / HTTP/1.0" 304 - "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:105.0) Gecko/20100101 Firefox/105.0"
::ffff:10.8.0.5 - - [05/Oct/2022:22:57:33 +0000] "GET / HTTP/1.0" 304 - "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:105.0) Gecko/20100101 Firefox/105.0"
::ffff:10.8.0.5 - - [05/Oct/2022:22:57:34 +0000] "GET / HTTP/1.0" 304 - "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:105.0) Gecko/20100101 Firefox/105.0"
listening on port 80
::ffff:10.8.0.5 - - [05/Oct/2022:22:56:42 +0000] "GET / HTTP/1.0" 200 1663 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:105.0) Gecko/20100101 Firefox/105.0"
::ffff:10.8.0.5 - - [05/Oct/2022:22:56:42 +0000] "GET /favicon.ico HTTP/1.0" 404 150 "http://35.203.28.8/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:105.0) Gecko/20100101 Firefox/105.0"
::ffff:10.8.1.9 - - [05/Oct/2022:22:57:22 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.8.0.5 - - [05/Oct/2022:22:57:25 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.8.0.5 - - [05/Oct/2022:22:57:26 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.8.0.5 - - [05/Oct/2022:22:57:33 +0000] "GET / HTTP/1.0" 304 - "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:105.0) Gecko/20100101 Firefox/105.0"
::ffff:10.8.0.5 - - [05/Oct/2022:22:57:34 +0000] "GET / HTTP/1.0" 304 - "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:105.0) Gecko/20100101 Firefox/105.0"
::ffff:10.8.0.5 - - [05/Oct/2022:22:57:35 +0000] "GET / HTTP/1.0" 304 - "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:105.0) Gecko/20100101 Firefox/105.0"
a_bainhewcampbell@cloudshell:~ (dev-solstice-362019)$ kubectl logs -l run=experiment-deployment
listening on port 80
::ffff:10.8.0.5 - - [05/Oct/2022:22:57:29 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.8.0.5 - - [05/Oct/2022:22:57:35 +0000] "GET / HTTP/1.0" 304 - "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:105.0) Gecko/20100101 Firefox/105.0"
listening on port 80
```

Part 3:

Since the pods were set up from part 2, I created the yaml file, deployed, and exposed it. Then I verified that it was working correctly by running the specified commands.

```
a_bainhewcampbell@cloudshell:~ (dev-solstice-362019)$ kubectl create -f loadbalancer-deployment.yaml
deployment.apps/loadbalancer-deployment created
a_bainhewcampbell@cloudshell:~ (dev-solstice-362019)$ kubectl get pods --output=wide
NAME                                       READY   STATUS    RESTARTS   AGE   IP           NODE                                       NOMINATED NODE   READINESS GATES
loadbalancer-deployment-6676f9ccf6-58w6x   1/1     Running   0          93s   10.8.2.8     gke-lab2-default-pool-2a786950-3rxq   <none>           <none>
loadbalancer-deployment-6676f9ccf6-g8xm6   1/1     Running   0          93s   10.8.0.8     gke-lab2-default-pool-2a786950-73bh   <none>           <none>
loadbalancer-deployment-6676f9ccf6-q4d8t   1/1     Running   0          93s   10.8.1.13    gke-lab2-default-pool-2a786950-pv4f   <none>           <none>
a_bainhewcampbell@cloudshell:~ (dev-solstice-362019)$ kubectl expose deployment loadbalancer-deployment --port=8080 --type=LoadBalancer
service/loadbalancer-deployment exposed
```

```
NAME                     TYPE           CLUSTER-IP    EXTERNAL-IP   PORT(S)          AGE
kubernetes               ClusterIP      10.12.0.1     <none>        443/TCP          5m48s
loadbalancer-deployment  LoadBalancer   10.12.6.39    35.203.28.8   8080:32388/TCP   104s
^Ca_bainhewcampbell@cloudshell:~ (dev-solstice-362019)$ curl http://35.203.28.8:8080/dog
A quadruped of the genus Canis, esp. the domestic dog (C.familiaris).a_bainhewcampbell@cloudshell:~ (dev-solstice-362019)$
a_bainhewcampbell@cloudshell:~ (dev-solstice-362019)$ curl http://35.203.28.8:8080/storey
See Story.a_bainhewcampbell@cloudshell:~ (dev-solstice-362019)$
```

Discussion:

As discussed in part 1, the major issues that the Gateway Routing pattern solve are when you want to expose multiple services on one end point, when there are multiple versions of the same service you want to expose to various users through one end point, and when you want multiple instances of a service on one endpoint. The solution that the Gateway Routing pattern provides is having a gateway service that will route the client to the various instances that an application may have.

The requirements of this pattern are that you have multiple deployments of some service(s) and a gateway service that will route the client to a deployment of their requested service(s).

In part 2 of the lab, this pattern was seen where we had the web deployment and the experimental deployment being kept behind the gateway of the ambassador deployment. The ambassador was used as a gateway such that you would be routed to one of the web or experimental deployments. This was seen after repeatedly using the curl command and viewing the logs of each of the deployments.

In part 3 of the lab, we deploy 3 images of the dictionary-server and use the readiness probe as a gateway. This gateway checks the readiness of all 3 deployed images every 5 seconds to ensure that they are healthy.

Design:

Autoscaling is a service that initiates new instances of your deployment automatically when your existing deployments are becoming overwhelmed or conversely closes deployments automatically when too many are idle or underutilized. This allows a platform to optimize the server utilization without having client performance degrade.

This differs from load balancing such that load balancers will balance the requests being taken in throughout existing deployments, while the autoscaler will create more deployments that the load balancer can route requests towards. Autoscaling is used when traffic to the platform is elastic as it can automatically scale up to meet the demands that a load balancer cannot do on its own, or scale down to save on computational costs.

How autoscaling is implemented can change depending on the business, however generally it involves the platform having a baseline amount of deployments that are scaled up

by a load balancing service when it finds there are not enough resources for the incoming requests.

Utilizing a kubernetes autoscaler guide, it was possible to set up an autoscaler and experiment with it and see the scaling done in real-time.

php-apache.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: php-apache
spec:
  selector:
    matchLabels:
      run: php-apache
  replicas: 1
  template:
    metadata:
      labels:
        run: php-apache
    spec:
      containers:
      - name: php-apache
        image: registry.k8s.io/hpa-example
        ports:
        - containerPort: 80
        resources:
          limits:
            cpu: 500m
          requests:
            cpu: 200m
---
apiVersion: v1
kind: Service
metadata:
  name: php-apache
  labels:
    run: php-apache
spec:
  ports:
  - port: 80
  selector:
    run: php-apache
```

```
a_bainhewcampbell@cloudshell:~ (dev-solstice-362019)$ kubectl apply -f https://k8s.io/examples/application/php-apache.yaml
deployment.apps/php-apache created
service/php-apache created
```

Configuration and deployment of yaml file

```
a_bainhewcampbell@cloudshell:~ (dev-solstice-362019)$ kubectl get hpa
NAME          REFERENCE                TARGETS                       MINPODS   MAXPODS   REPLICAS   AGE
php-apache    Deployment/php-apache    <unknown>/1 (avg), <unknown>/60%   1         20        1          6m53s
a_bainhewcampbell@cloudshell:~ (dev-solstice-362019)$ kubectl get hpa php-apache --watch
NAME          REFERENCE                TARGETS                       MINPODS   MAXPODS   REPLICAS   AGE
php-apache    Deployment/php-apache    <unknown>/1 (avg), 0%/60%     1         20        1          7m27s
php-apache    Deployment/php-apache    <unknown>/1 (avg), 250%/60%   1         20        1          8m
php-apache    Deployment/php-apache    <unknown>/1 (avg), 250%/60%   1         20        4          8m15s
php-apache    Deployment/php-apache    <unknown>/1 (avg), 250%/60%   1         20        5          8m30s
^Ca_bainhewcampbell@cloudshell:~ (dev-solstice-362019)$ kubectl get deployment php-apache
NAME          READY   UP-TO-DATE   AVAILABLE   AGE
php-apache    5/5     5            5           2m49s
```

Using a command that generates load onto the deployments so that the autoscaler kicks in and creates more replicas to meet the need

```
a_bainhewcampbell@cloudshell:~ (dev-solstice-362019)$ kubectl get hpa php-apache --watch
NAME          REFERENCE                TARGETS                       MINPODS   MAXPODS   REPLICAS   AGE
php-apache    Deployment/php-apache    <unknown>/1 (avg), 43%/60%    1         20        5          9m54s
php-apache    Deployment/php-apache    <unknown>/1 (avg), 0%/60%     1         20        5          10m
^Ca_bainhewcampbell@cloudshell:~ (dev-solstice-362019)$ kubectl get deployment php-apache
NAME          READY   UP-TO-DATE   AVAILABLE   AGE
php-apache    5/5     5            5           4m24s
```

After stopping the command we can see CPU Utilization go down

Homework 2: Network and Virtualization
Member: Mamun Hossain

Introduction:

The objective of this lab was to learn how to configure and run a request using Nginx file compilation. In doing so, we reapplied our knowledge from the last lab using various .yaml files and configured the ConfigMap tool in Kubernetes. We were also able to familiarize ourselves with using the curl command to request an HTTP method. Finally we were able to configure load balancer services and get familiar with the load balancing patterns that were involved.

Experiment:

For this lab, we reused our knowledge with the Google Cloud platform to increase our knowledge of Docker while using Kubernetes. Previously we learned that Docket is an open platform that allowed for development, shipment, and running application. Now we want to understand how to decouple environment-specific configurations in the container images that we create so we can make the applications more portable in our approach.

In this lab, we ran Kubernetes once again to recreate GKE clusters again and from there, we were needed to create a web server with the provided YAML file configuration called web-depolyment.yaml. Once that has been inputted, we will see two replicas of aci-helloworld images be created. From here on, we create the deployment for the web deployment. Afterwards we retrieve the deployment information. Then we repeat these steps but instead of deploying the web server we will deploy and experimental deployment. With these steps we can generate a config map by creating a sub folder and create a file called nginx-ambassador.conf which in term give our previous deployment servers with weights. Once we repeat the steps one more time with an ambassador-deployment.yaml, we can receive the service information and then use the ambassador-IP address to see the web browser and the NGINX implementation.

Discussion:

Now we could have used simple backend information and received a similar output if we manipulate the code the way we wanted, so why did we do it like this? What benefit do we have with web deployments do we have over storing our own backend information? We need to consider that a client can potentially have multiple service instances or it might need to consume multiple services, so gateway routing patterns become crucial when needing to expose these services or instances to route traffic.

The best solution is by providing a gateway at the start of an application, right when the consumer uses it. This also has an application with the OSI layers, with layer 7 proving its usefulness when needing to route the request at the best times possible. This pattern for gateway routing works very well especially for rolling out deployments and how the updates are sent to the users on the same network. And even if any issues are found during the updates, a new service can change the update and revert it back to the original.

Some requirements that are needed to use the Gateway Routing Pattern are as follows: A client will need to consume multiple servic es before it can be accessed behind a gateway. A client will also need to have to consume a number of different instances in order for it to work, as well as being able to route different requests from one endpoint to another. An example of this is by having the ports of a Virtual Machine and creating a GKE cluster to virtual IP addresses.

Now if we look at part 2 and 3 of our lab, we can see that we can use gateway routing right after the deployments of the .YAML files, especially for part 2. The reasons that would be greatly beneficial is because of the following:

- Multiple services being accessed (web-deployment, experimental-deployment, ambassador-deployment)

- Simplifying client applications to single accesspoint (nginx-ambassador.conf)

- Needing to consumer variable number of service instance (Retrieving IP addresses to access HTTP services)

Now if we look at part 3, we can see Gateway Routing Pattern with the following requirements:

- Client consuming multiple services (3 replicas of dictionary-server)

- Simplifying client applications by a single endpoint (loadbalancer-deployment.yaml)

- Deployment where clients can access multiple versions at the same time (using the Server-IP to retrieve information from the word search but use the same port)

With this information, we can see how Gateway Routing Patterns can be achieved with Part 2 and Part 3 with the requirements listed because when we do it manually it takes a lot more time and a lot more resources are being expedited to deploy each web server manually one by one.

Design:

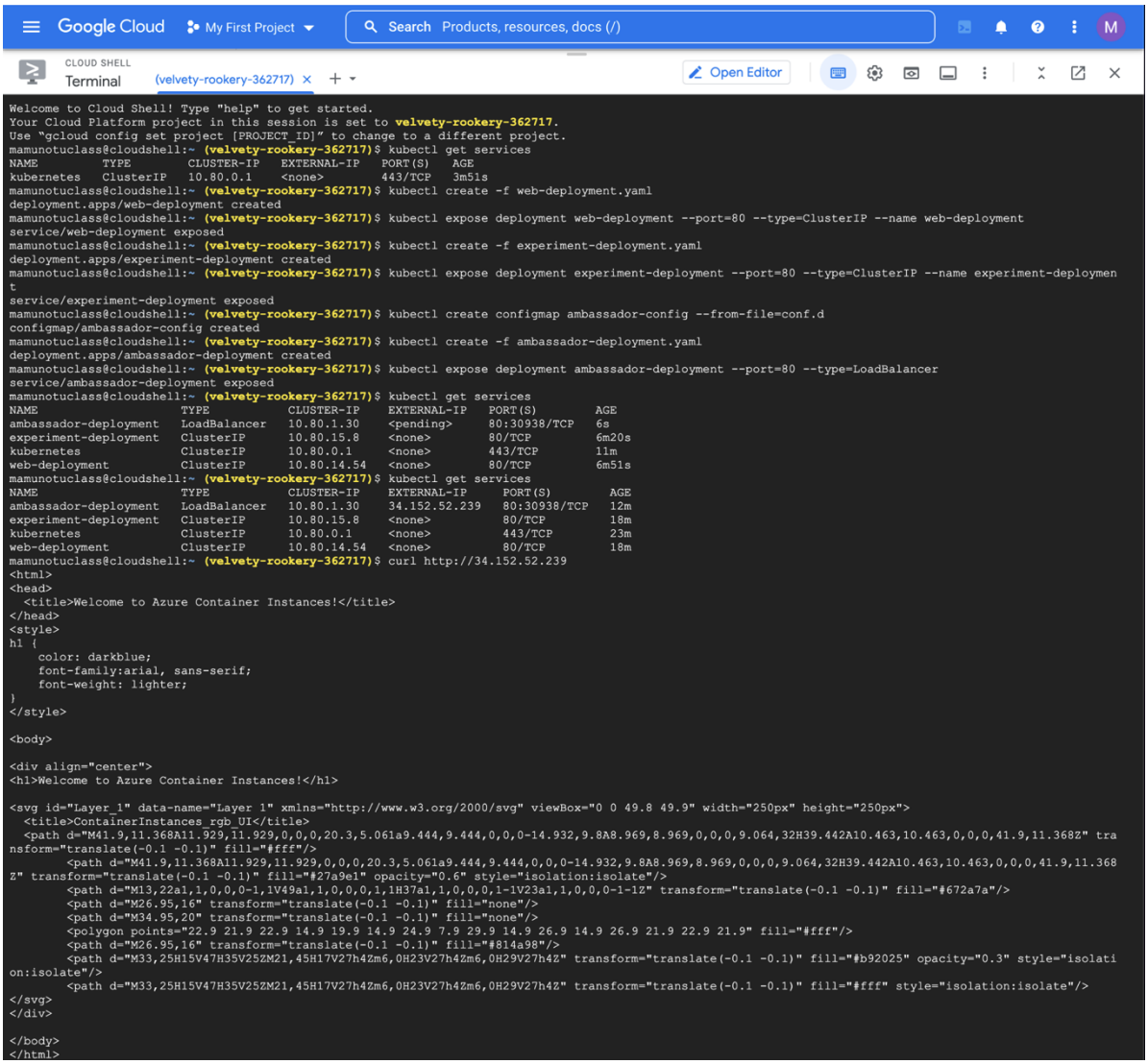
Below screenshots will be provided with the information regarding on how auto scaling has been implemented. That being said, we will discuss why autoscaling is used.

The main reason we use Auto Scaling is to ensure that our application is working at the desired levels we want, and can help us monitor in doing so. For example, when an application requires more resources, then the auto scaling feature can increase the capacity of the constrained resources so the service is still high quality, and vice versa.

The reason why autoscaling is different from load balancing and request splitting is because autoscaling will initiate new instances and load balancing will re reroute connections from unhealthy instances and attach connections instead of creating new ones, and request splitting will parse the request message and split the corrupted requests instead of creating new ones.

Design (Screenshots):

Deployment of web-deployment.yaml, experiment-deployment.yaml, and ambassador-deployment.yaml alongside configuration of nginx-ambassador.conf file, whilst deploying the pods and services to retrieve IP information. Using external IP to see NGINX implementation and then running script to call server and check how many times each server was called (all the .YAML files are provided in the lab).



Checking how many times each server was called for both web-deployment and experiment-deployment.

```
mamunotuclass@cloudshell:~ (velvety-rookery-362717)$ for _ in {1..20}; do curl http://34.152.52.239 -s > output.txt; done
mamunotuclass@cloudshell:~ (velvety-rookery-362717)$ kubectl logs -l run=web-deployment
::ffff:10.76.0.9 - - [08/Oct/2022:18:48:28 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.76.2.5 - - [08/Oct/2022:18:48:28 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.76.0.9 - - [08/Oct/2022:18:48:28 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.76.0.9 - - [08/Oct/2022:18:48:28 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.76.2.5 - - [08/Oct/2022:18:48:29 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.76.0.9 - - [08/Oct/2022:18:48:29 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.76.2.5 - - [08/Oct/2022:18:48:29 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.76.0.9 - - [08/Oct/2022:18:48:29 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.76.0.9 - - [08/Oct/2022:18:48:29 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.76.0.9 - - [08/Oct/2022:18:48:29 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.76.0.9 - - [08/Oct/2022:18:44:31 +0000] "GET /config/getuser?index=0 HTTP/1.0" 404 153 "-" "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:76.0) Geck
o/20100101 Firefox/76.0"
::ffff:10.76.2.5 - - [08/Oct/2022:18:47:28 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.76.2.5 - - [08/Oct/2022:18:48:28 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.76.0.9 - - [08/Oct/2022:18:48:28 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.76.2.5 - - [08/Oct/2022:18:48:28 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.76.2.5 - - [08/Oct/2022:18:48:28 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.76.2.5 - - [08/Oct/2022:18:48:29 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.76.0.9 - - [08/Oct/2022:18:48:29 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.76.0.9 - - [08/Oct/2022:18:48:29 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.76.0.9 - - [08/Oct/2022:18:48:29 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
mamunotuclass@cloudshell:~ (velvety-rookery-362717)$ kubectl logs -l run=experiment-deployment
listening on port 80
listening on port 80
::ffff:10.76.0.9 - - [08/Oct/2022:18:48:29 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
::ffff:10.76.2.5 - - [08/Oct/2022:18:48:29 +0000] "GET / HTTP/1.0" 200 1663 "-" "curl/7.74.0"
mamunotuclass@cloudshell:~ (velvety-rookery-362717)$
```

Initializing the loadbalancer-deployment.yaml and repeating the steps from Part 2 to retrieve service replication is working. Then, initializing the autoscaling deployment by creating the .YAML for the autoscaling file, we are able to create the deployment and pod of the autoscaling.

```
mamunotuclass@cloudshell:~ (velvety-rookery-362717)$ kubectl create -f loadbalancer-deployment.yaml
deployment.apps/loadbalancer-deployment created
mamunotuclass@cloudshell:~ (velvety-rookery-362717)$ kubectl get pods --output=wide
NAME                                        READY   STATUS              RESTARTS   AGE   IP          NODE                                        NOMINATED NOD
E   READINESS GATES
ambassador-deployment-66db4f7766-56566      1/1     Running             0          51m   10.76.2.5   gke-openfaas-default-pool-f9b771c9-kvjs     <none>
    <none>
ambassador-deployment-66db4f7766-9chql      1/1     Running             0          51m   10.76.0.9   gke-openfaas-default-pool-f9b771c9-gfs0     <none>
    <none>
experiment-deployment-7b47cbd668-q48xj      1/1     Running             0          57m   10.76.1.6   gke-openfaas-default-pool-f9b771c9-rgzw     <none>
    <none>
experiment-deployment-7b47cbd668-w86zb      1/1     Running             0          57m   10.76.0.8   gke-openfaas-default-pool-f9b771c9-gfs0     <none>
    <none>
loadbalancer-deployment-6676f9ccf6-7qsxs    0/1     ContainerCreating   0          7s    <none>      gke-openfaas-default-pool-f9b771c9-rgzw     <none>
    <none>
loadbalancer-deployment-6676f9ccf6-fhjqc    0/1     ContainerCreating   0          7s    <none>      gke-openfaas-default-pool-f9b771c9-kvjs     <none>
    <none>
loadbalancer-deployment-6676f9ccf6-wch4b    0/1     ContainerCreating   0          7s    <none>      gke-openfaas-default-pool-f9b771c9-gfs0     <none>
    <none>
web-deployment-6fdbb5c6bb-v9dqh             1/1     Running             0          58m   10.76.2.4   gke-openfaas-default-pool-f9b771c9-kvjs     <none>
    <none>
web-deployment-6fdbb5c6bb-zb8wl             1/1     Running             0          58m   10.76.0.7   gke-openfaas-default-pool-f9b771c9-gfs0     <none>
    <none>
mamunotuclass@cloudshell:~ (velvety-rookery-362717)$ kubectl expose deployment loadbalancer-deployment --port=8080 --type=LoadBalancer
service/loadbalancer-deployment exposed
mamunotuclass@cloudshell:~ (velvety-rookery-362717)$ kubectl get services --watch
NAME                      TYPE           CLUSTER-IP     EXTERNAL-IP     PORT(S)          AGE
ambassador-deployment     LoadBalancer   10.80.1.30     34.152.52.239   80:30938/TCP     51m
experiment-deployment     ClusterIP      10.80.15.8     <none>          80/TCP           57m
kubernetes                ClusterIP      10.80.0.1      <none>          443/TCP          62m
loadbalancer-deployment   LoadBalancer   10.80.14.172   <pending>       8080:31058/TCP   9s
web-deployment            ClusterIP      10.80.14.54    <none>          80/TCP           58m
loadbalancer-deployment   LoadBalancer   10.80.14.172   35.203.101.45   8080:31058/TCP   33s
^Z
[1]+  Stopped                 kubectl get services --watch
mamunotuclass@cloudshell:~ (velvety-rookery-362717)$ kubectl get services
NAME                      TYPE           CLUSTER-IP     EXTERNAL-IP     PORT(S)          AGE
ambassador-deployment     LoadBalancer   10.80.1.30     34.152.52.239   80:30938/TCP     52m
experiment-deployment     ClusterIP      10.80.15.8     <none>          80/TCP           58m
kubernetes                ClusterIP      10.80.0.1      <none>          443/TCP          63m
loadbalancer-deployment   LoadBalancer   10.80.14.172   35.203.101.45   8080:31058/TCP   71s
web-deployment            ClusterIP      10.80.14.54    <none>          80/TCP           59m
mamunotuclass@cloudshell:~ (velvety-rookery-362717)$ curl http://35.203.101.45:8080/dog
A quadruped of the genus Canis, esp. the domestic dog (C.familiaris).mamunotuclass@cloudshell:~ (velvety-rookery-362717)$ curl http://35.203.101.45:8080/st
orey
See Story.mamunotuclass@cloudshell:~ (velvety-rookerykubectl apply -f https://k8s.io/examples/application/php-apache.yamlpache.yaml
deployment.apps/php-apache created
service/php-apache created
mamunotuclass@cloudshell:~ (velvety-rookery-362717)$
```

```
mamunotuclass@cloudshell:~ (velvety-rookery-362717)$ kubectl create -f php-apache.yaml
Error from server (AlreadyExists): error when creating "php-apache.yaml": deployments.apps "php-apache" already exists
Error from server (AlreadyExists): error when creating "php-apache.yaml": services "php-apache" already exists
mamunotuclass@cloudshell:~ (velvety-rookery-362717)$ kubectl get services
NAME                      TYPE           CLUSTER-IP     EXTERNAL-IP     PORT(S)          AGE
ambassador-deployment     LoadBalancer   10.80.1.30     34.152.52.239   80:30938/TCP     24h
experiment-deployment     ClusterIP      10.80.15.8     <none>          80/TCP           24h
kubernetes                ClusterIP      10.80.0.1      <none>          443/TCP          24h
loadbalancer-deployment   LoadBalancer   10.80.14.172   35.203.101.45   8080:31058/TCP   23h
php-apache                ClusterIP      10.80.13.15    <none>          80/TCP           23h
web-deployment            ClusterIP      10.80.14.54    <none>          80/TCP           24h
mamunotuclass@cloudshell:~ (velvety-rookery-362717)$ kubectl apply -f https://k8s.io/examples/application/php-apache.yaml
deployment.apps/php-apache unchanged
service/php-apache unchanged
mamunotuclass@cloudshell:~ (velvety-rookery-362717)$ kubectl autoscale deployment php-apache --cpu-percent=50 --min=1 --max=10
horizontalpodautoscaler.autoscaling/php-apache autoscaled
mamunotuclass@cloudshell:~ (velvety-rookery-362717)$ kubectl get hpa
NAME         REFERENCE               TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
php-apache   Deployment/php-apache   0%/50%    1         10        1          17s
mamunotuclass@cloudshell:~ (velvety-rookery-362717)$ kubectl get hpa
NAME         REFERENCE               TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
php-apache   Deployment/php-apache   0%/50%    1         10        1          41s
mamunotuclass@cloudshell:~ (velvety-rookery-362717)$ kubectl run -i --tty load-generator --rm --image=busybox:1.28 --restart=Never -- /bin/sh -c "while sleep 0.01; do wget -q -O- http://php-apache; done"
If you don't see a command prompt, try pressing enter.
OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!
OK!OK!OK!OK!OK!OK!OK!OK!
OK!OK!OK!OK!OK!OK!
OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!^Cpod "load-generator" deleted
pod default/load-generator terminated (Error)
mamunotuclass@cloudshell:~ (velvety-rookery-362717)$ kubectl get hpa php-apache --watch
NAME         REFERENCE               TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
php-apache   Deployment/php-apache   0%/50%    1         10        1          80s
php-apache   Deployment/php-apache   64%/50%   1         10        1          91s
```

Code for Auto Scaling .YAML

php-apache.yaml > ...

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: php-apache
spec:
  selector:
    matchLabels:
      run: php-apache
  replicas: 1
  template:
    metadata:
      labels:
        run: php-apache
    spec:
      containers:
      - name: php-apache
        image: registry.k8s.io/hpa-example
        ports:
        - containerPort: 80
        resources:
          limits:
            cpu: 500m
          requests:
            cpu: 200m
---
apiVersion: v1
kind: Service
metadata:
  name: php-apache
  labels:
    run: php-apache
spec:
  ports:
  - port: 80
  selector:
    run: php-apache
```

Conclusion:

In conclusion, we were able to reutilize Kubernetes and understand the Gateway Routing Patterns with several deployment instances and use different config files and understand NGINX implementation as well as the usage of autoscaling. The lab had run into a minor issue, when trying to reiterate the original cluster from lab 1, but we needed to delete the clusters and create new ones. The lab was a success, and we were able to increase our knowledge about Kubernetes and its deployments and the Gateway Routing Pattern.