



**Faculty of Engineering and Applied Science**

**SOFE 4790U Distributed Systems**

**CRN 44425**

**Lab # 2**

**Mamun Hossain - 100553073**

## **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 Docker 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-deployment.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 services 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.

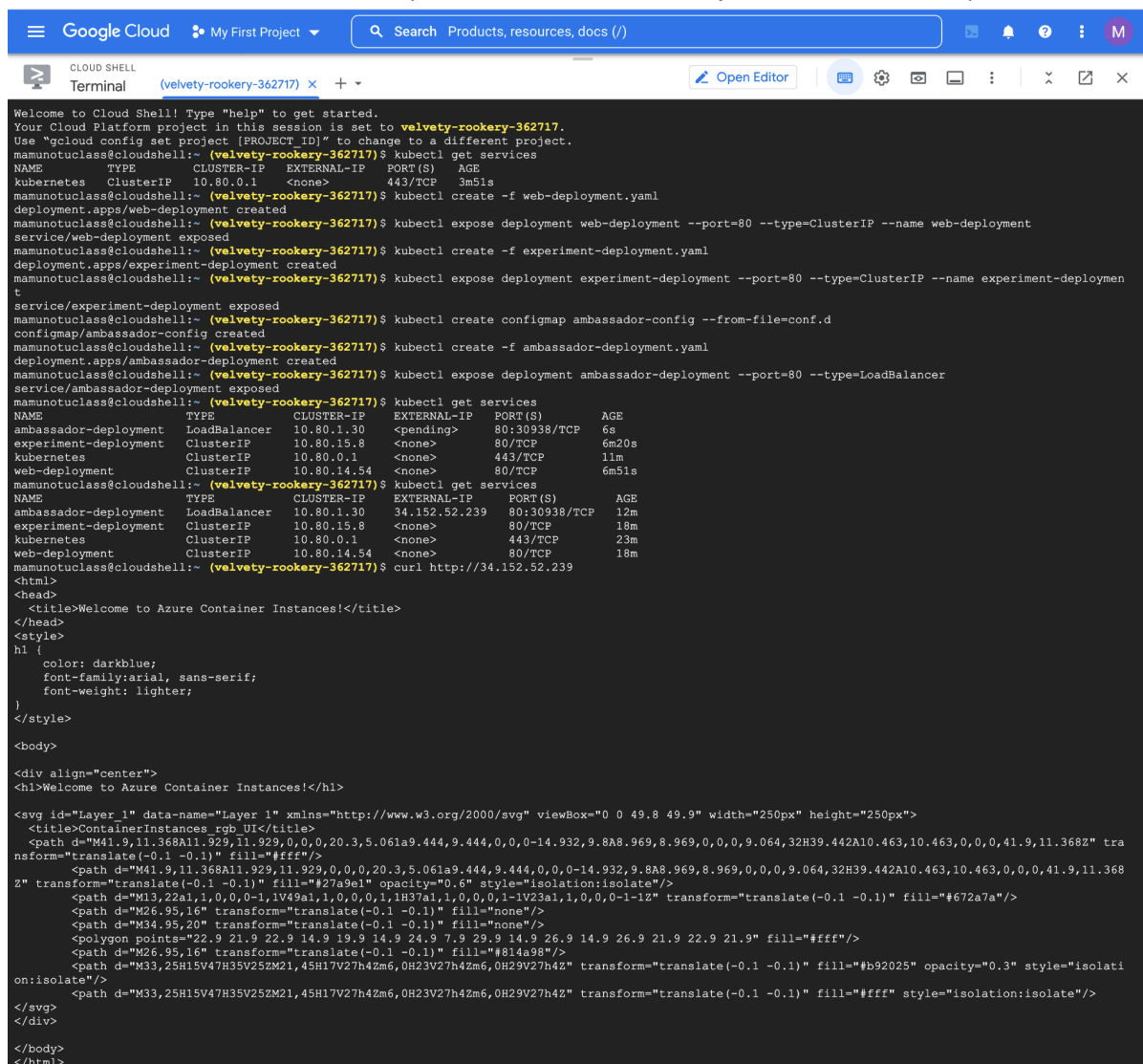
The main reason we use Auto Scaling is to ensure that our application is working at the desired levels we want, and can help us monitor in doing so. For example, when an application requires more resources, then the auto scaling feature can increase the capacity of the constrained resources so the service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



```
Google Cloud My First Project Search Products, resources, docs (/)
Terminal (velvety-rookery-362717) x + Open Editor

Welcome to Cloud Shell! Type "help" to get started.
Your Cloud Platform project in this session is set to velvety-rookery-362717.
Use "gcloud config set project [PROJECT_ID]" to change to a different project.
mamunotucass@cloudshell:~ (velvety-rookery-362717)$ kubectl get services
NAME      TYPE      CLUSTER-IP    EXTERNAL-IP    PORT(S)    AGE
kubernetes ClusterIP  10.80.0.1      <none>         443/TCP     3m51s
mamunotucass@cloudshell:~ (velvety-rookery-362717)$ kubectl create -f web-deployment.yaml
deployment.apps/web-deployment created
mamunotucass@cloudshell:~ (velvety-rookery-362717)$ kubectl expose deployment web-deployment --port=80 --type=ClusterIP --name web-deployment
service/web-deployment exposed
mamunotucass@cloudshell:~ (velvety-rookery-362717)$ kubectl create -f experiment-deployment.yaml
deployment.apps/experiment-deployment created
mamunotucass@cloudshell:~ (velvety-rookery-362717)$ kubectl expose deployment experiment-deployment --port=80 --type=ClusterIP --name experiment-deployment
service/experiment-deployment exposed
mamunotucass@cloudshell:~ (velvety-rookery-362717)$ kubectl create configmap ambassador-config --from-file=conf.d
configmap/ambassador-config created
mamunotucass@cloudshell:~ (velvety-rookery-362717)$ kubectl create -f ambassador-deployment.yaml
deployment.apps/ambassador-deployment created
mamunotucass@cloudshell:~ (velvety-rookery-362717)$ kubectl expose deployment ambassador-deployment --port=80 --type=LoadBalancer
service/ambassador-deployment exposed
mamunotucass@cloudshell:~ (velvety-rookery-362717)$ kubectl get services
NAME      TYPE      CLUSTER-IP    EXTERNAL-IP    PORT(S)    AGE
ambassador-deployment LoadBalancer 10.80.1.30      <pending>      80:30938/TCP 6s
experiment-deployment ClusterIP     10.80.15.8      <none>         80/TCP       6m20s
kubernetes ClusterIP  10.80.0.1      <none>         443/TCP       11m
web-deployment ClusterIP     10.80.14.54     <none>         80/TCP       6m51s
mamunotucass@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 12m
experiment-deployment ClusterIP     10.80.15.8      <none>         80/TCP       18m
kubernetes ClusterIP  10.80.0.1      <none>         443/TCP       23m
web-deployment ClusterIP     10.80.14.54     <none>         80/TCP       18m
mamunotucass@cloudshell:~ (velvety-rookery-362717)$ curl http://34.152.52.239
<html>
<head>
<title>Welcome to Azure Container Instances!</title>
</head>
<style>
<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="M13.22a1,1,0,0,1,1V49a1,1,0,0,1,1H37a1,1,0,0,1,1V23a1,1,0,0,1,1V23a1,1,0,0,1,1Z" 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="M33,25H15V47H35V25ZM21,45H17V27H42M6,0H23V27H42M6,0H29V27H42" transform="translate(-0.1 -0.1)" fill="#b92025" opacity="0.3" style="isolation:isolate"/>
  <path d="M33,25H15V47H35V25ZM21,45H17V27H42M6,0H23V27H42M6,0H29V27H42" transform="translate(-0.1 -0.1)" fill="#fff" style="isolation:isolate"/>
</svg>
</div>
</body>
</html>
```

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

```
mamunotucass@cloudshell:~ (velvety-rookery-362717) $ for _ in {1..20}; do curl http://34.152.52.239 -s > output.txt; done
mamunotucass@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/2022:18:48:29 +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) Gecko/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"
mamunotucass@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"
mamunotucass@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.



```

php-apache.yaml > ...
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: php-apache
5  spec:
6    selector:
7      matchLabels:
8        run: php-apache
9    replicas: 1
10   template:
11     metadata:
12       labels:
13         run: php-apache
14     spec:
15       containers:
16       - name: php-apache
17         image: registry.k8s.io/hpa-example
18         ports:
19         - containerPort: 80
20         resources:
21           limits:
22             cpu: 500m
23           requests:
24             cpu: 200m
25   ---
26   apiVersion: v1
27   kind: Service
28   metadata:
29     name: php-apache
30     labels:
31       run: php-apache
32   spec:
33     ports:
34     - port: 80
35     selector:
36       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.