Faculty of Engineering and Applied Science

SOFE 4790U Distributed Systems

Lab #2

CRN: 43525

Date: 2022-10-09

Name	Student #
Owen Musselman	100657709

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:

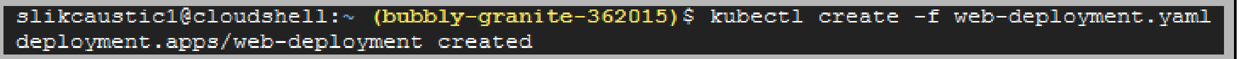
- Gateway routing pattern

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

Part 2.

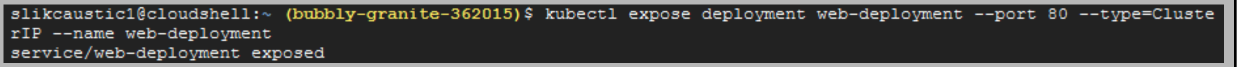
1. Check if pods are running: kubectl get pods
2. Check if deployments are running: kubectl get deployments
3. Check if services are running: kubectl get services
4. Getting the externa IP: 34.152.63.218

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



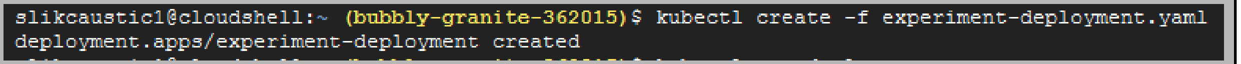
```
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$ kubectl create -f web-deployment.yaml
deployment.apps/web-deployment created
```

Creating the web deployment.



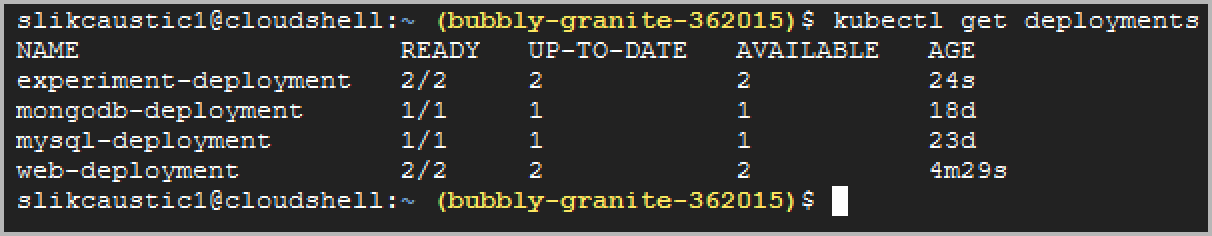
```
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$ kubectl expose deployment web-deployment --port 80 --type=Cluste
rIP --name web-deployment
service/web-deployment exposed
```

Creating a clusterIP, and exposing the web-deployment.



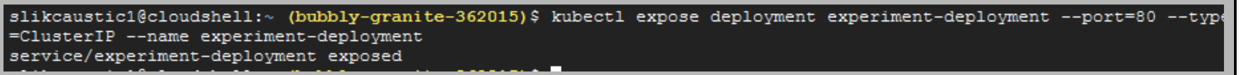
```
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$ kubectl create -f experiment-deployment.yaml
deployment.apps/experiment-deployment created
```

Creating experiment deployment.



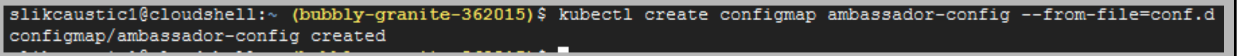
```
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$ kubectl get deployments
NAME                    READY   UP-TO-DATE   AVAILABLE   AGE
experiment-deployment   2/2     2            2           24s
mongodb-deployment      1/1     1            1           18d
mysql-deployment        1/1     1     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

Viewing the deployments created.



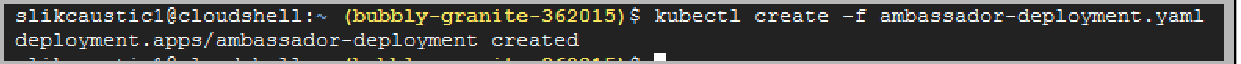
```
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$ kubectl expose deployment experiment-deployment --port=80 --typ
=ClusterIP --name experiment-deployment
service/experiment-deployment exposed
```

ClusterIP for the experiment deployment.



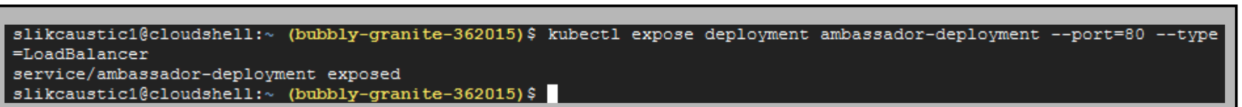
```
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$ kubectl create configmap ambassador-config --from-file=conf.d
configmap/ambassador-config created
```

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



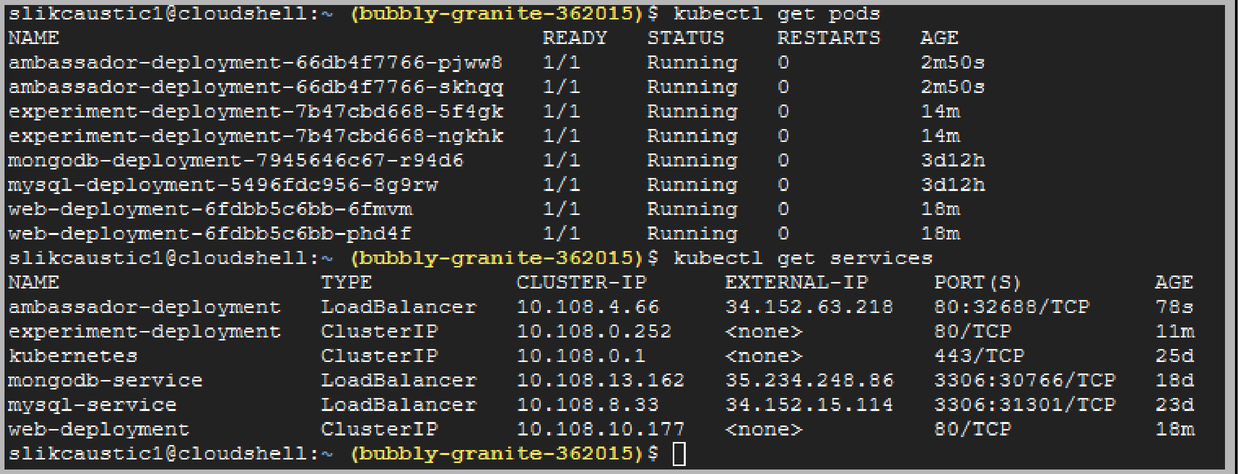
```
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$ kubectl create -f ambassador-deployment.yaml
deployment.apps/ambassador-deployment created
```

Creating ambassador-deployment deployment.



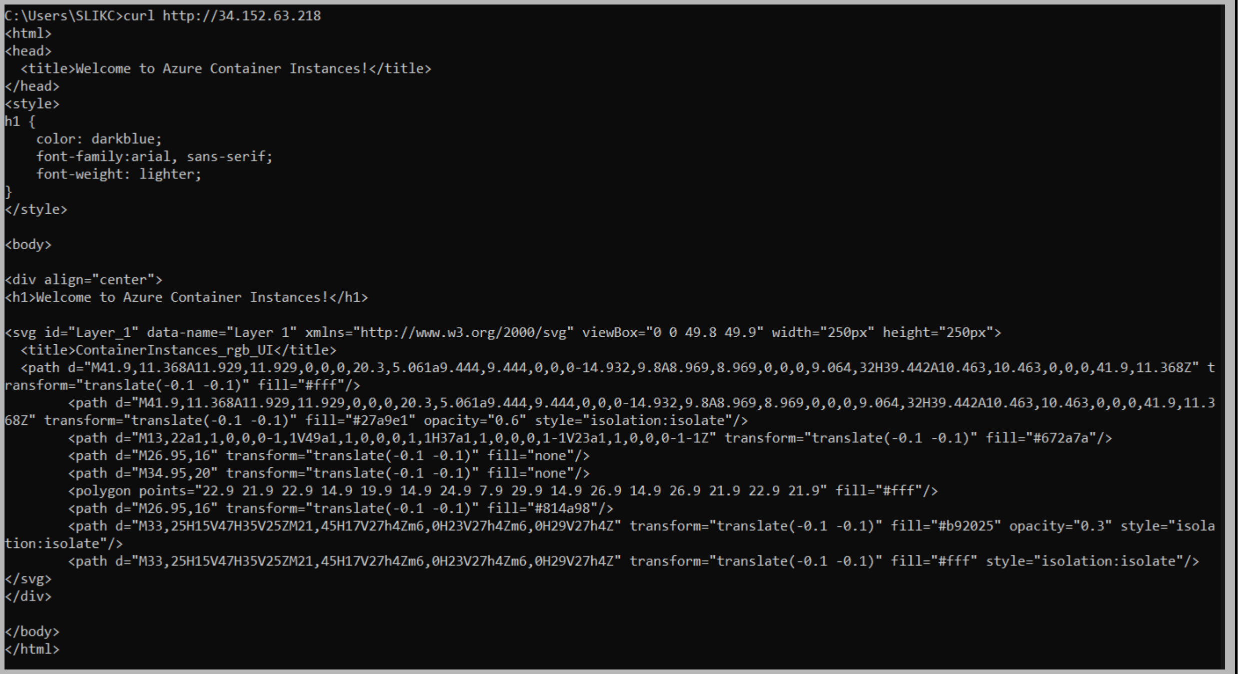
```
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$ kubectl expose deployment ambassador-deployment --port=80 --type
=LoadBalancer
service/ambassador-deployment exposed
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$
```

Assign a load balancer to the ambassador deployment.



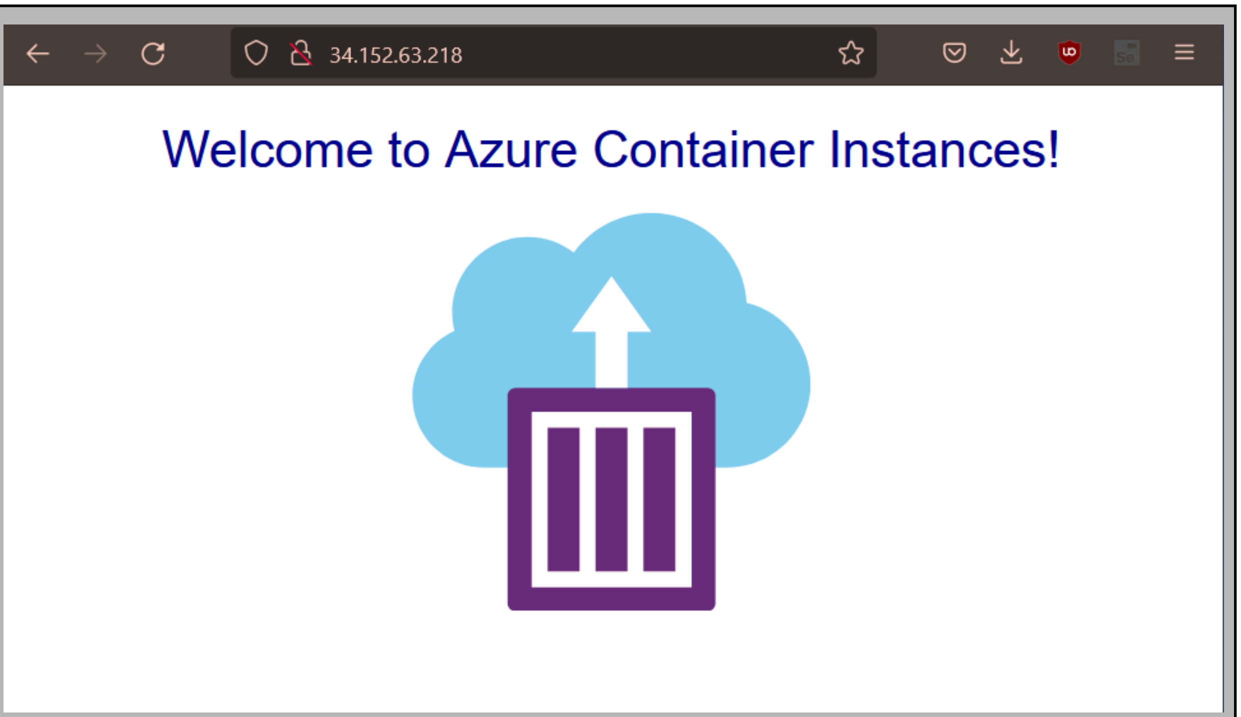
```
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$ kubectl get pods
NAME                                     READY   STATUS    RESTARTS   AGE
ambassador-deployment-66db4f7766-pjww8   1/1     Running   0          2m50s
ambassador-deployment-66db4f7766-skhqq   1/1     Running   0          2m50s
experiment-deployment-7b47cbd668-5f4gk   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.

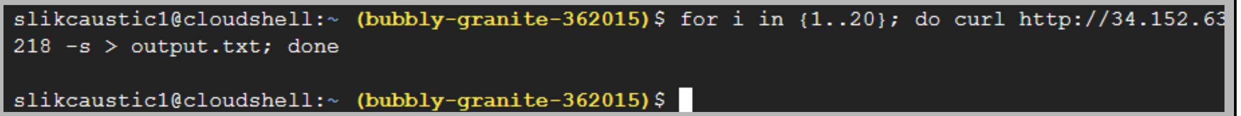


```
C:\Users\SLIKC>curl http://34.152.63.218
<html>
<head>
  <title>Welcome to Azure Container Instances!</title>
</head>
<style>
h1 {
    color: darkblue;
    font-family:arial, sans-serif;
    font-weight: lighter;
}
</style>

<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="M41.9,11.368A11.929,11.929,0,0,0,20.3,5.061a9.444,9.444,0,0,0-14.932,9.8A8.969,8.969,0,0,0,9.064,32H39.442A10.463,10.463,0,0,0,41.9,11.368Z" t
ransform="translate(-0.1 -0.1)" fill="#fff"/>
        <path d="M41.9,11.368A11.929,11.929,0,0,0,20.3,5.061a9.444,9.444,0,0,0-14.932,9.8A8.969,8.969,0,0,0,9.064,32H39.442A10.463,10.463,0,0,0,41.9,11.3
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.

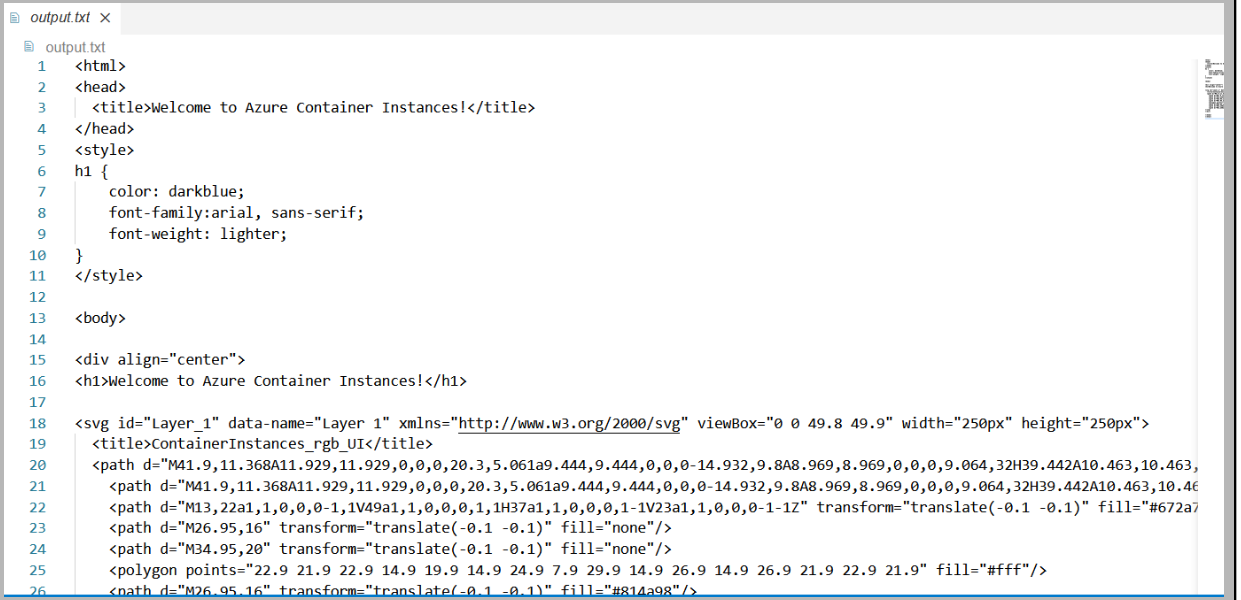


Above is what the output looks like in a browser when using the external IP for the ambassador service.



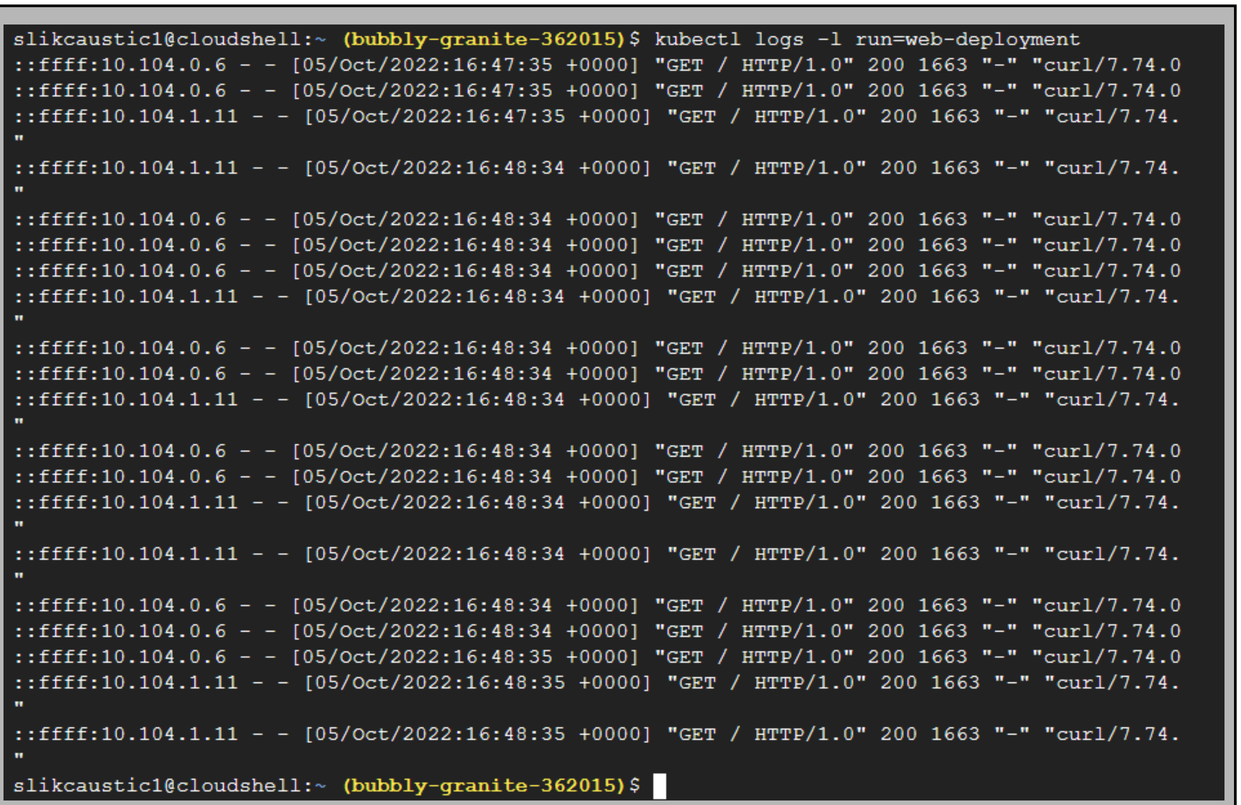
```
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$ for i in {1..20}; do curl http://34.152.63
218 -s > output.txt; done

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

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



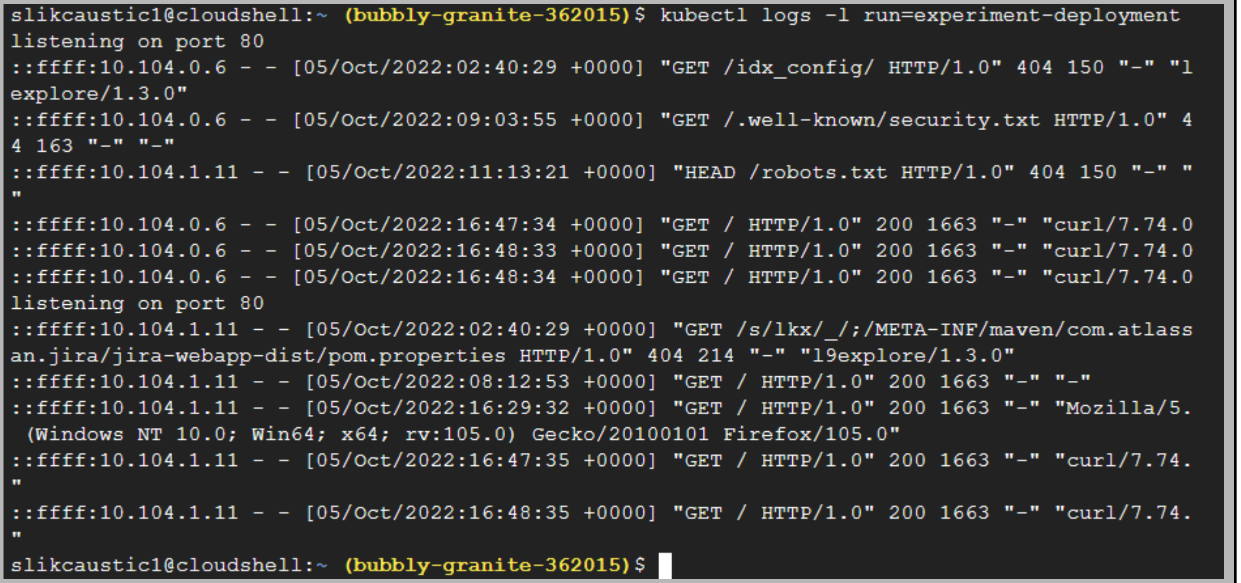
```
<html>
<head>
  <title>Welcome to Azure Container Instances!</title>
</head>
<style>
h1 {
    color: darkblue;
    font-family:arial, sans-serif;
    font-weight: lighter;
}
</style>

<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)$
```

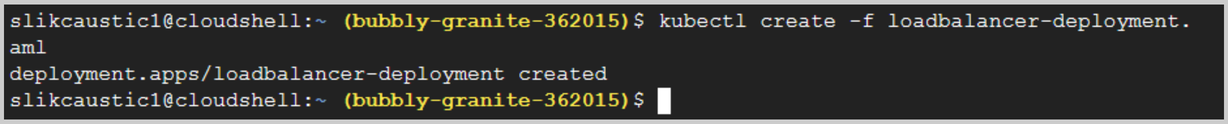
Output of the kubectl logs -l run=web-deployment. 90% of the traffic directed to the web-deployment.



```
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)$
```

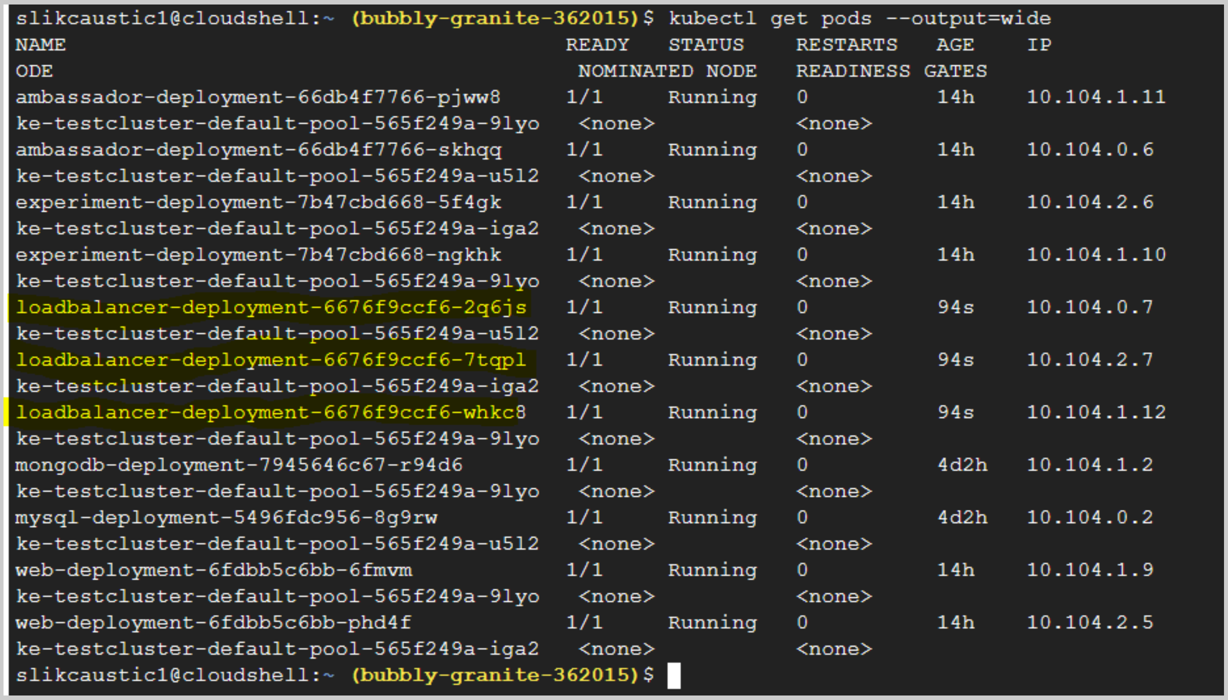
Output of the kubectl logs -l run=experiment-deployment. Can see how it was accessed by curl and my Mozilla Firefox browser. 10% of the requests are sent here to the experiment deployment.

Part 3:



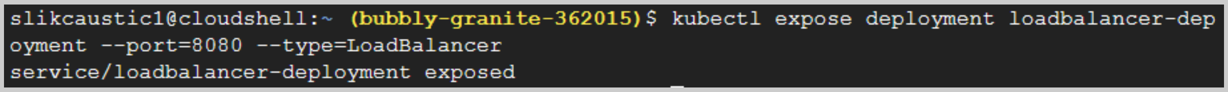
```
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$ kubectl create -f loadbalancer-deployment.
aml
deployment.apps/loadbalancer-deployment created
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$
```

Creating the load balancer deployment using the given yaml file.



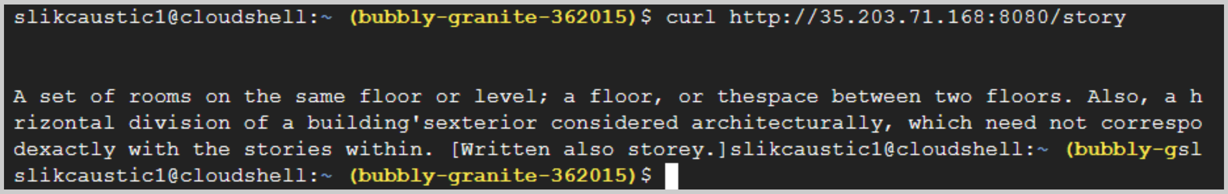
```
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$ kubectl get pods --output=wide
NAME                                           READY   STATUS    RESTARTS   AGE    IP
ODE                                             NOMINATED NODE   READINESS GATES
ambassador-deployment-66db4f7766-pjww8         1/1     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.

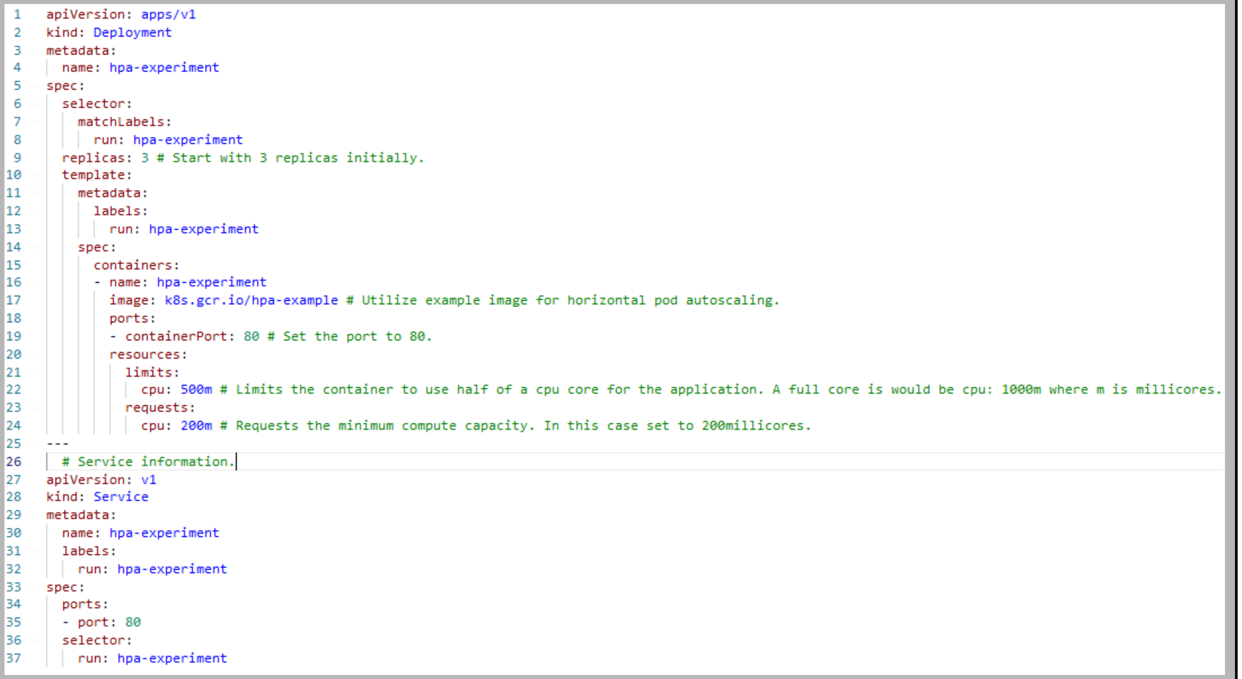
Design:

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.

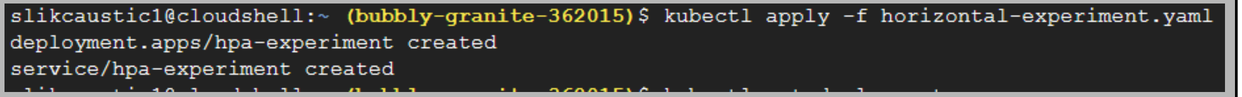
Auto scaling and load balancing/request splitting as with autoscaling the resources being used are being scaled up or scaled down to match the usage of the application/service. While load balancing is responsible for distributing the traffic coming into various other locations to serve them.

Implementation of horizontal scaling:



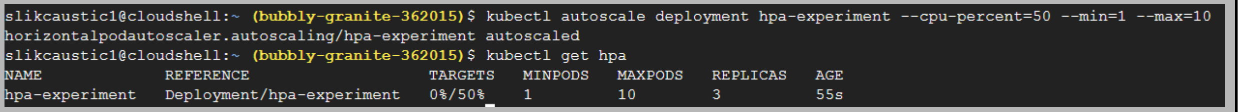
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hpa-experiment
spec:
  selector:
    matchLabels:
      run: hpa-experiment
  replicas: 3 # Start with 3 replicas initially.
  template:
    metadata:
      labels:
        run: hpa-experiment
    spec:
      containers:
      - name: hpa-experiment
        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.



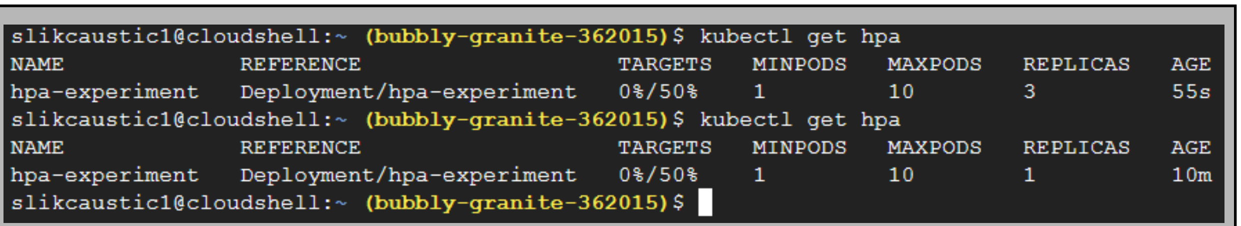
```
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$ kubectl apply -f horizontal-experiment.yaml
deployment.apps/hpa-experiment created
service/hpa-experiment created
```

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



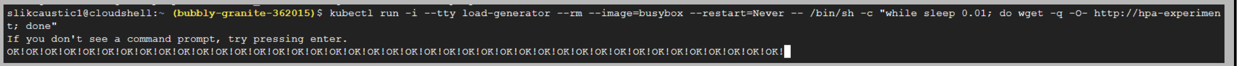
```
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$ kubectl autoscale deployment hpa-experiment --cpu-percent=50 --min=1 --max=10
horizontalpodautoscaler.autoscaling/hpa-experiment autoscaled
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$ kubectl get hpa
NAME             REFERENCE                   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.



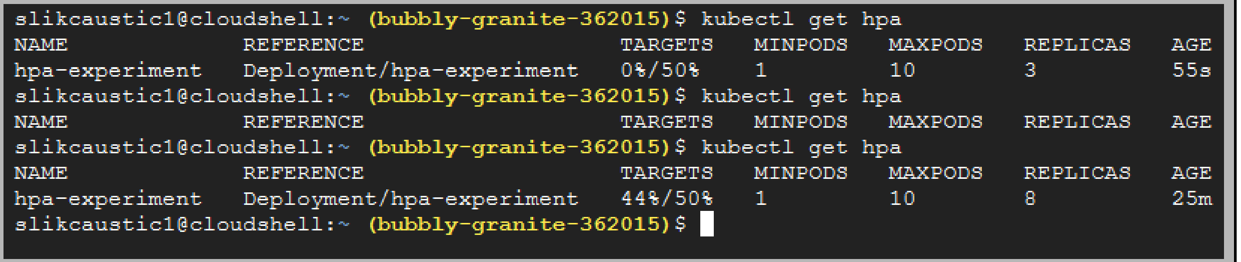
```
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$ kubectl get hpa
NAME             REFERENCE                  TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
hpa-experiment   Deployment/hpa-experiment   0%/50%    1         10        3          55s
slikcaustic1@cloudshell:~ (bubbly-granite-362015)$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

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].