TU Dublin (Tallaght), Department of Computing
M.Sc. in DevOps
2019/2020
Enterprise Architecture Design
CA Project Report

Mubasher Mohammed

1. Part-1

We are building and deploying two type of microservices applications (synchronous & asynchronous) separately then we are running asynchronous application from within the Synchronous application.

1.1. Building and deploying Synchronous application on

We are building microservices applications that includes the main service (*atn-service* or all the news) that has two branches or helper services *wf* (weather fetcher) and nf (news fetcher). We are also using *atn-rdis* to store the number of page visits.

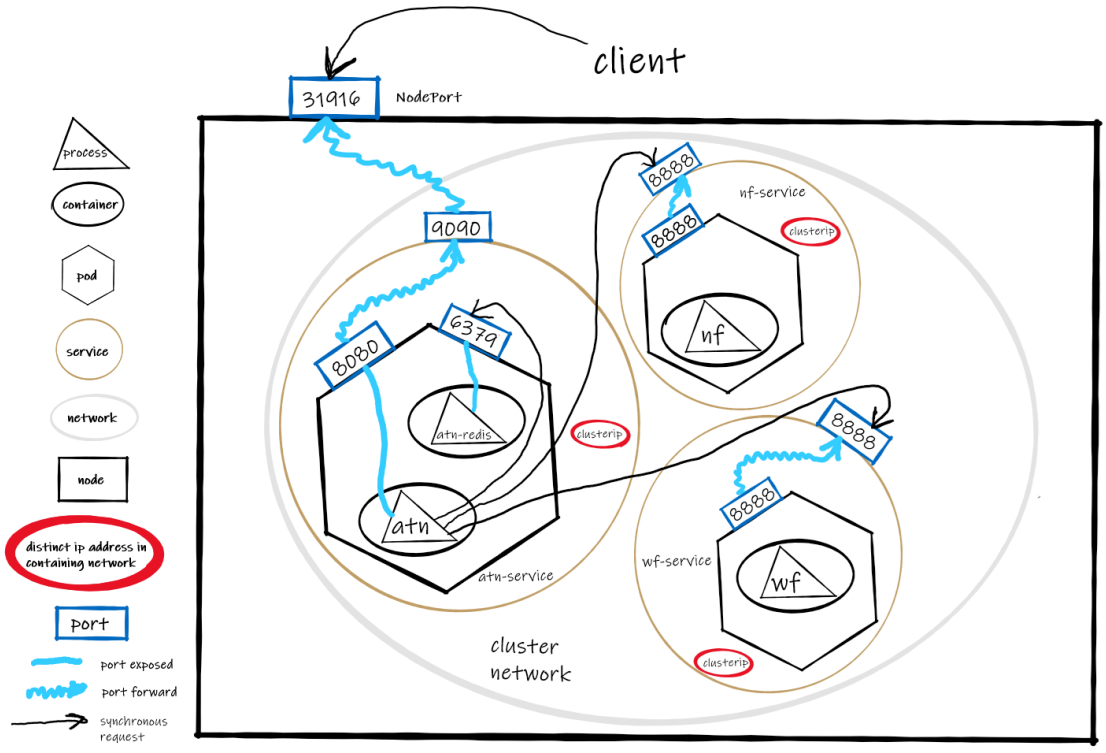

Figure 1 Synchronous microservices applications image

As you can see on Figure-1 the services (news fetcher and weather fetcher) communicate with the main service via port 8888 on cluster network. Redis is communicating with the main service on port 6379. The main service atn has two port exposed one with the network cluster (9090) and the second one which is the Node-port which is in our case it is (31826) that connect to the client.

We are using the materials provided for the lab for Enterprise Architecture Design, to build these services. The materials can be found on this link in GitHub.

1.1.1. Tools used

There many tools and options available but we used these tools part of this project:

- Docker in GCP for building containers.
- Docker Hub for storing containers in repositories Docker Hub ID “mubasherm”
- Kubernetes in GKE for managing the container Cluster. Figure-2 shows the detail of the cluster created on GCP in the location Europe-north1-a.

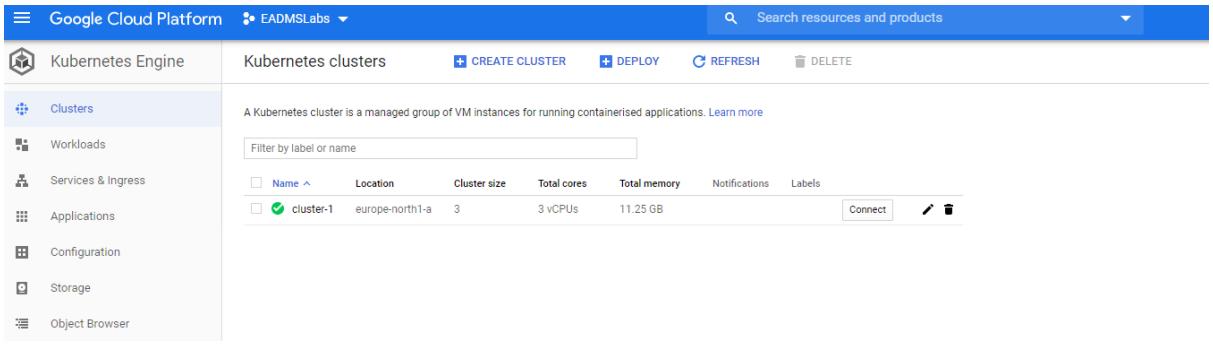

Figure 2 Kubernetes Cluster-1 details

1.1.2. Building the images and image repositories in Docker Hub

These steps are followed to build tag and push all the images to Docker Hub:

1. We build all the images using docker build command
   We tagged all the images with our Docker Hub ID, like $`docker tag allthenews:v3 mubasherm/allthenews:v3`
3. Push all the images to Docker Hub using docker push command. Figure-3 shows all the images repository in Docker Hub.

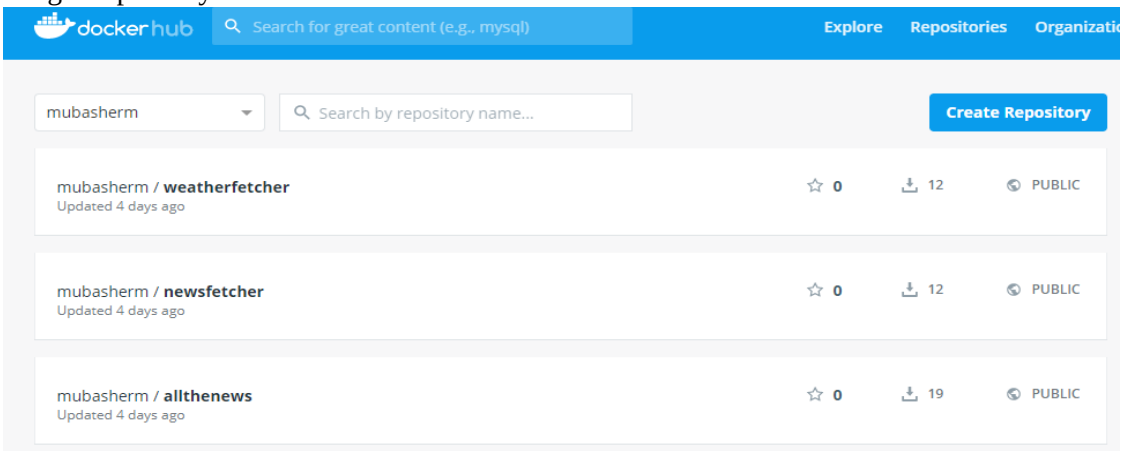

Figure 3 images repository in Docker Hub

1.1.3. Deployment and getting Kubernetes pods up and running

Using kubectl create command we create pods for all the deployments like: `kubectl create -f deployment_atn3.yaml.` Figure-4 shows the list of pods created for each container including the ones for this project.

```
mubasher_mohammed@cloudshell:~/api-21lab-k8s (eadmslabs)$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
atn-deployment-5d7bb58fb5-vkl24     2/2     Running   0          44h
atnt-deployment-84cf557bc-kvnwr     2/2     Running   0          126m
door1-deployment-66f8bfcb44-4n8lf   1/1     Running   0          4d21h
door2-deployment-998fd69b5-kz6hd    1/1     Running   0          4d21h
door3-deployment-6df59c6d6b-g5m49   1/1     Running   0          4d21h
nf-deployment-8588b5c47f-4zjrs      1/1     Running   0          3d21h
nft-deployment-bd89f44f8-b9vsx      1/1     Running   0          26h
redis-deployment-5f849646f9-xhq5p   1/1     Running   0          4d21h
seccon-deployment-75d698884b-tc9rn  1/1     Running   0          4d21h
wf-deployment-96bcfbdc4-xt7pb       1/1     Running   0          3d21h
wft-deployment-568d6b84f8-7c29s     1/1     Running   0          19h
mubasher_mohammed@cloudshell:~/api-21lab-k8s (eadmslabs)$
```

Figure 4 list of the pods

Once we have the pods created and running. For the application consisting of the multiple pods to be useful, they need to communicate with each other and for this purpose we need to expose them as services. To do this we need to create services, the services can be created using the kubectl command like: `$ kubectl create -f service_atnt.yaml` once we created all the service we should be able to list them as shown in Figure-5.

```
mubasher_mohammed@cloudshell:~/api-21lab-k8s (eadmslabs)$ kubectl get services
NAME              TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
atn-service       NodePort                    <none>        9090:31916/TCP   6d19h
atnt-service      NodePort                    <none>        9090:31826/TCP   27h
kubernetes        ClusterIP                   <none>        443/TCP          6d19h
nf-service        ClusterIP                   <none>        8888/TCP         6d19h
nft-service       ClusterIP                   <none>        8888/TCP         27h
redis-service     ClusterIP                   <none>        6379/TCP         4d21h
seccon-service    NodePort                    <none>        9090:31080/TCP   4d21h
wf-service        ClusterIP                   <none>        8888/TCP         6d19h
wft-service       ClusterIP                   <none>        8888/TCP         27h
```

Figure 5 list of services

Since we have all the services created we should be able to access the main service on the external IP of the instance on the specified port. The Figure-6 shows that content from news fetcher, weather and also the number of times the page visited all displayed on the main page. This the link for the web server http://35.228.239.159:31826/allthenews?style=colourful which is running on port 31826.

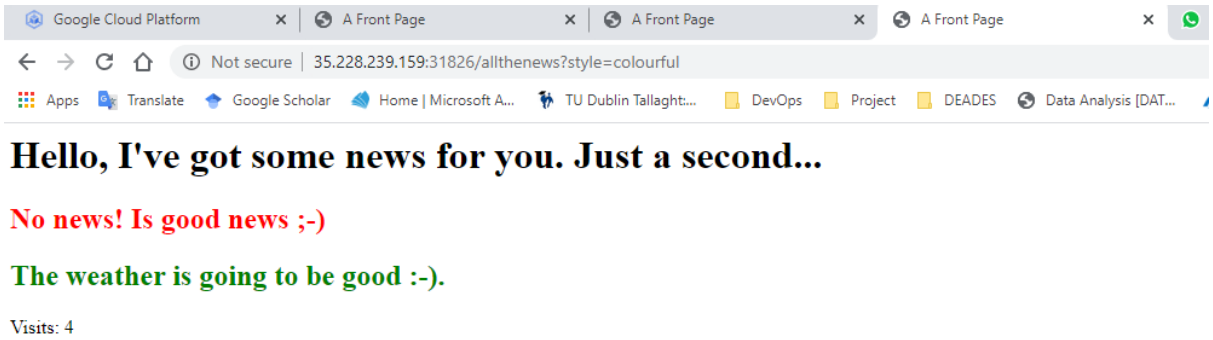

Figure 6 the microservices front page

1.2.Building and deploying Asynchronous services.

We are building and deploying microservices asynchronous communication between services. In this part as well we using the lab material provided in GitHub this link.

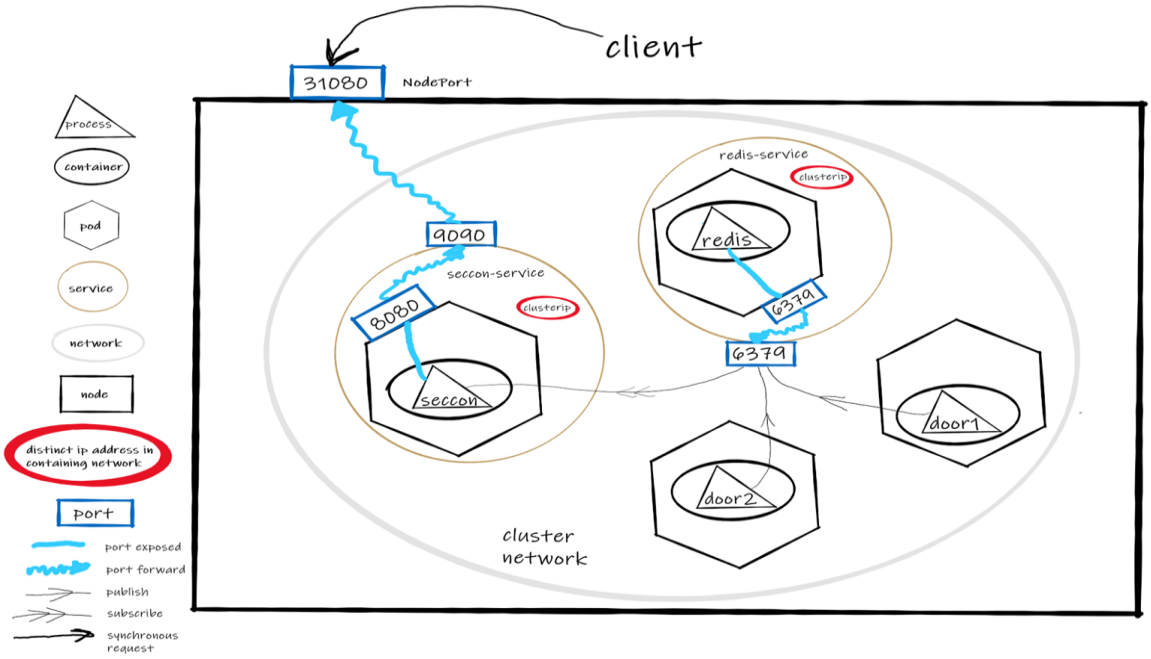

Figure 7 Asynchronous communications of microservices

As shown on Figure-7 we have microservices communicating on the internal network and with the client via node port and external IP. Door1, door2 and door3 which is not on the Figure-7 all communicating internally with the redis on port 6379 and redis is communicating with the main service seccon with the same port. Seccon exposed port 8080 to seccon service and forwarding port 9090 to the network cluster and connected to the client via node-port 31080.

1.2.1. Building the images and image repositories in Docker Hub

Like section 1.1. we are using docker build command to build seccon, door1, door2, and door3 services and push them to our Docker Hub. Figure-8 shows the repositories created in dockerhub.

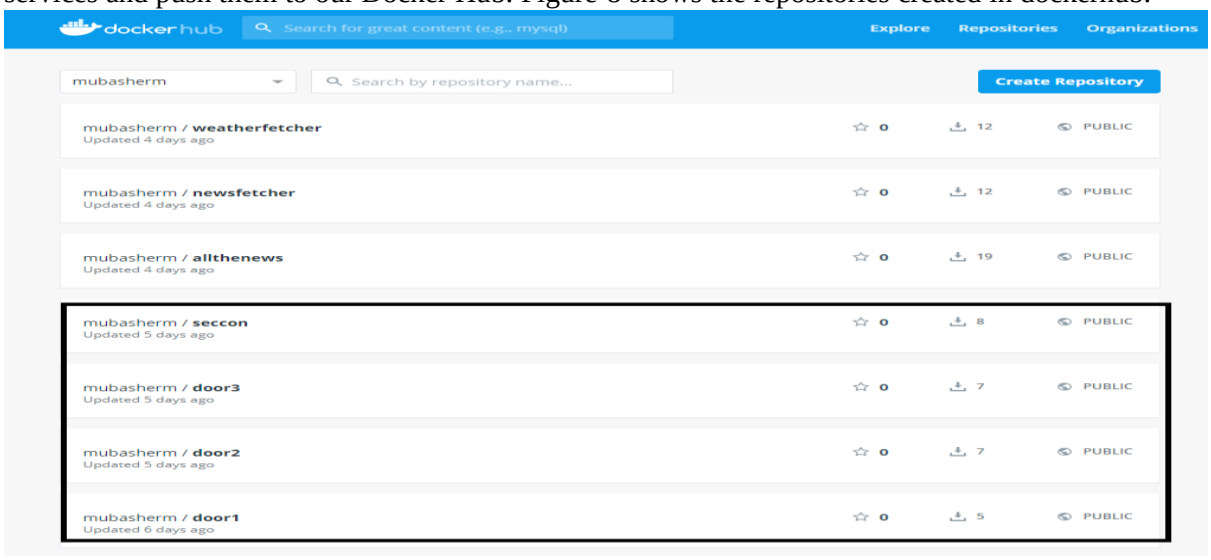

Figure 8 repositories in Docker Hub

1.2.2. Deployment and getting Kubernetes pods up and running

We create the services using kubectl create command like below:

```
$ kubectl create -f manifests/redis.yaml
```

```
$ kubectl create -f manifests/seccon.yaml
```

Figure-9 shows all the services plus the services created for redis and seccon.

```
mubasher_mohammed@cloudshell:~/api-3lab-async (eadmslabs)$ kubectl get services
NAME              TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
atn-service       NodePort                    <none>        9090:31916/TCP   6d20h
atnt-service      NodePort                    <none>        9090:31826/TCP   28h
kubernetes        ClusterIP                   <none>        443/TCP          6d21h
nf-service        ClusterIP                   <none>        8888/TCP         6d20h
nft-service       ClusterIP                   <none>        8888/TCP         28h
redis-service     ClusterIP                   <none>        6379/TCP         4d22h
seccon-service    NodePort                    <none>        9090:31080/TCP   4d22h
wft-service       ClusterIP                   <none>        8888/TCP         28h
mubasher_mohammed@cloudshell:~/api-3lab-async (eadmslabs)$
```

Figure 9 services list on the cluster

Once the security console service is up and running, we deploy door1, door2 and door using kubectl create.

```
$ kubectl create -f manifests/deployment_d1.yaml
```

```
$ kubectl create -f manifests/deployment_d2.yaml
```

```
$ kubectl create -f manifests/deployment_d3.yaml
```

We should be able to see all the deployments including the one we created for this services Figure-10.

```
mubasher_mohammed@cloudshell:~/api-3lab-async (eadmslabs)$ kubectl get deployments
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
atn-deployment      1/1     1            1           6d21h
atnt-deployment     1/1     1            1           4h14m
door1-deployment    1/1     1            1           4d23h
door2-deployment    1/1     1            1           4d23h
door3-deployment    1/1     1            1           4d23h
nf-deployment       1/1     1            1           6d21h
nft-deployment      1/1     1            1           29h
redis-deployment    1/1     1            1           4d23h
seccon-deployment   1/1     1            1           4d23h
wf-deployment       1/1     1            1           6d21h
wft-deployment      1/1     1            1           29h
mubasher_mohammed@cloudshell:~/api-3lab-async (eadmslabs)$
```

Figure 10 list of the deployments

Once we have the services running we should be able to access the services on the specified port (31080) using external IP as you can see on Figure-11. The services can be access on this link: http://35.228.233.110:31080/

Figure 11 asynchronous services client side

1.3.Running asynchronous application with synchronous

To achieve this, we have to call the Async services which is counting number of people walked through the doors on our Synchronous services (ATN). As you can see on Figure-12 now the client will be able to see both services on the same URL http://35.228.239.159:31916/allthenews?style=colourful.

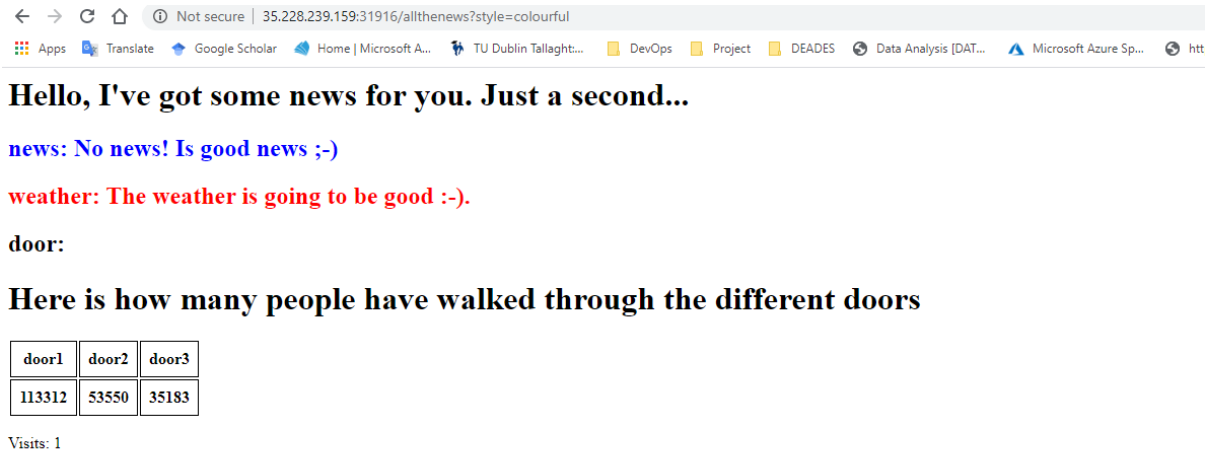

Figure 12 Async services running on Synchronous

We have to add the URL string of the Async services "http://35.228.233.110:31080/" to the aTN4 deployment file as you can see on the code snippet below and then replace the deployment file by running `$ kubectl replace -f deployment_atn4.yaml.`

```
spec:

          containers: [{name: atn, image: 'mubasherm/allthenews:v4', args: [
news, 'http://nf-service:8888', weather, 'http://wf-
service:8888', door, 'http://35.228.206.59:31080/'], ports: [{containerPort: 8
080}]}, {name: atn-redis, image: redis, ports: [{containerPort: 6379}]}]
```

2. Part-2

We are using APACHE JMeter for getting the average response time for Synchronous and Asynchronous services and comparing both results. Figure-13 shows the thread setup in JMeter we are making 100 users are making 5 times requests access to each service, in total we have 500 request to each service. Figures-14 and 15 shows how setup the http request in JMeter.

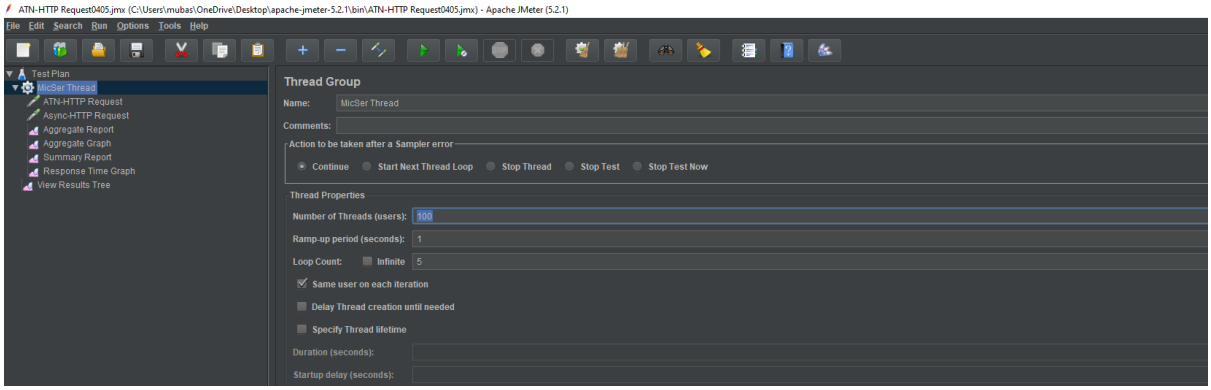

Figure 13 Thread setup in JMeter

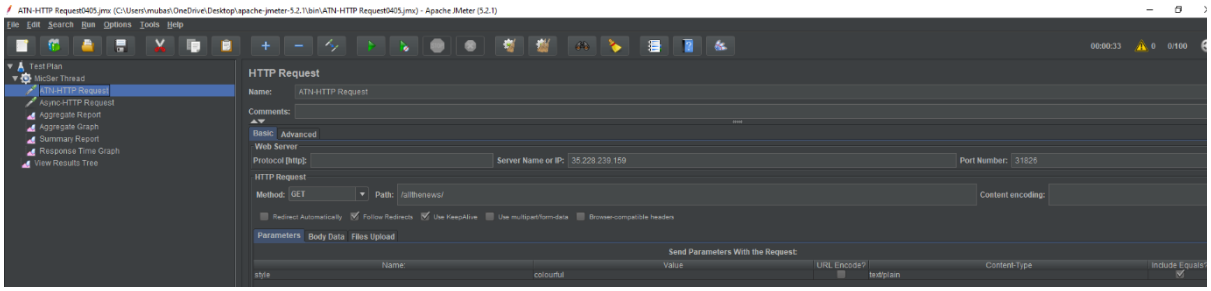

Figure 14 atn http request setup in Jmeter

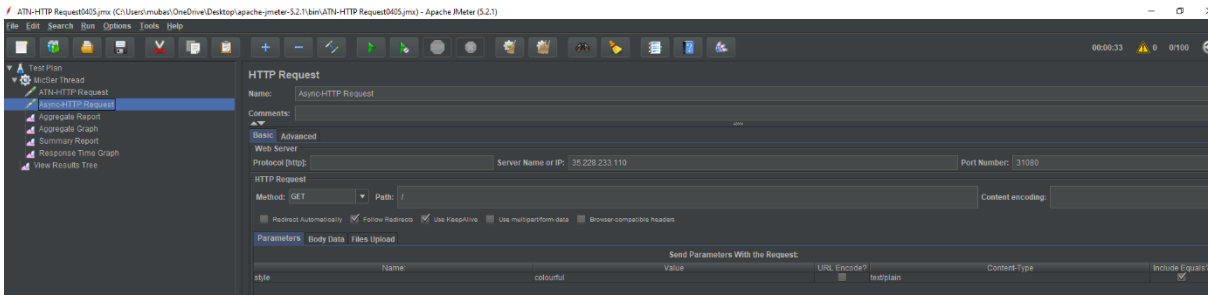

Figure 15 Async http request setup in JMeter

Figure-16 display the average response time in bar-chart for both async and synchronous services. Also Figure-17 shows the summary of the test run which was 500 request for to each service by 100 users.

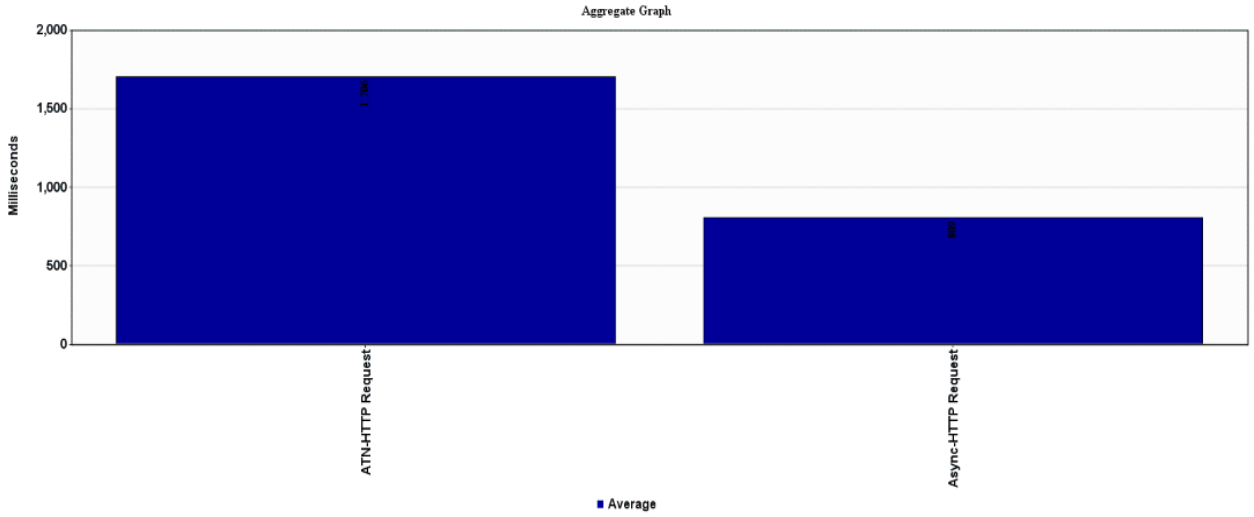

Figure 16 Average Response Time

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received KB/sec
ATN-HTTP Request	500	1706	1614	3130	3580	5736	131	21006	0.20%	15.1/sec	11.53
Async-HTTP Request	500	809	660	1548	2122	2740	61	6742	0.00%	15.1/sec	7.86
TOTAL	1000	1258	1023	2432	3136	5583	61	21006	0.10%	29.9/sec	19.19

Figure 17 Summary Report

3. References

Google Cloud Platform

https://console.cloud.google.com/kubernetes/list?project=eadmslabs

Google Kubernetes Engine

https://cloud.google.com/kubernetes-engine/?hl=en_GB&_ga=2.217743455.-89011285.1579868906&_gac=1.15013956.1588591988.Cj0KCQjw-r71BRDuARIsAB7i_QN6vAqOxSNBVEN2Tot2BPgM_UazZI4He6GUJh6OXyM1GNm3Oyo2LDoaAlelEALw_wcB#section-2

Kubernetes getting started

https://kubernetes.io/docs/setup/production-environment/turnkey/gce/

APACHE JMeter:

https://jmeter.apache.org/usermanual/component_reference.html#Aggregate_Graph