Best Load Balancer for Kubernetes

Project Elective under Prof.Thangaraju B

Naveen Kumar V R
IMT2017029
International Institute of Information Technology
Bangalore
naveenkumar.vr@iiitb.ac.in

***Abstract*—This document is a research on which load balancer is best for kubernetes(k8). This talks about the load balancer, how to deploy it and trigger the traffic to check the best among of all.**

I. Introduction

The most widely used tools in the modern cloud-native container ecosystem are Docker and Kubernetes. Docker is utility for packaging and running container. Docker helps build standard containers that include all the necessary components required for them to function in isolation. Kubernetes, also known as K8s, is an open-source system for automating deployment, scaling, and management of containerized applications. Docker is about building individual containers while K8s is about managing and orchestrating large number of them. Load distribution is the most basic type of load balancing in K8s. The Kubernetes load balancer sends connections to the first server in the pool until it is at capacity, and then sends new connections to the next available server. In this paper, we are going to compare the load balancers, nginx, HaPorxy and Traefik by deploying and triggering the traffic for each load balancer.

II. Load Balancers

A. NGINX

NGINX is a reverse proxy and load balancing tool that performs multiple roles. NGINX provides a suite of products which run within Kubernetes environments:

- NGINX Plus – A reverse proxy and load balancer that can perform multiple roles:
  - Sidecar in NGINX Service Mesh
  - Ingress controller for Kubernetes clusters managing both ingress and egress traffic
  - Per-service and per-pod application firewall proxy when deployed with NGINX App Protect
  - Service-to-service API gateway between containers and pods
- NGINX Service Mesh – A lightweight but full-featured service mesh designed around NGINX Plus, providing enterprise-ready data plane security, scalability, and complete cluster-wide traffic management.
- NGINX Ingress Controller – An enterprise ingress and egress controller for Kubernetes cluster traffic management and API gateway use cases.

B. HAProxy

HAProxy Kubernetes Ingress Controller, unlocks access to the raw HAProxy configuration language for power users to gain more control. You can also enable mutual TLS authentication between the ingress controller and services, enforce Basic authentication, and return custom error pages to users. This also enhances the controller's internals, resulting in a more efficient HAProxy configuration.It also provides a way to run the ingress controller outside of your Kubernetes cluster, but still, monitor the cluster for changes to pods. Some teams prefer this approach as they transition their applications to the container platform. You can launch the controller on a separate server and give it access to the pod network.

C. Traefik

Configuration discovery in Traefik is achieved through Providers. The providers are infrastructure components, whether orchestrators, container engines, cloud providers, or key-value stores. The idea is that Traefik queries the provider APIs in order to find relevant information about routing, and when Traefik detects a change, it dynamically updates the routes.While each provider is different, you can think of each as belonging to one of four categories:

- Label-based: each deployed container has a set of labels attached to it
- Key-Value-based: each deployed container updates a key-value store with relevant information
- Annotation-based: a separate object, with annotations, defines the characteristics of the container
- File-based: uses files to define configuration

III. Tools and Installation

A. Docker

First we have to uninstall old versions of docker.

```
$ sudo apt-get remove docker docker-engine docker.io containerd runc
```

Update the apt package index, and install the latest version of Docker Engine, containerd, and Docker Compose, or go to the next step to install a specific version:

Identify applicable funding agency here. If none, delete this.

```
$ sudo apt-get update
$ sudo apt-get install docker-ce docker-ce
-cli containerd.io docker-compose-plugin
```

To install a specific version of Docker Engine, list the available versions in the repo, then select and install:

- List the versions available in your repo:
  `$ apt-cache madison docker-ce`
- Install a specific version using the version string from the second column:
  `$ sudo apt-get install docker-ce=<VERSION_STRING> docker-ce-cli=<VERSION_STRING> containerd.io docker-compose-plugin`
- Verify that Docker Engine is installed correctly by running the hello-world image.
  `$ sudo docker run hello-world`

B. Kubernetes

- Since you are downloading Kubernetes from a non-standard repository, it is essential to ensure that the software is authentic. This is done by adding a signing key.
  `$ sudo apt-get install curl`
  `$ curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg`
  `$ sudo apt-key add`
- Kubernetes is not included in the default repositories. To add them, enter the following:
  `$ sudo apt-add-repository "deb http://apt.kubernetes.io/ kubernetes-xenial main"`
- Kubeadm (Kubernetes Admin) is a tool that helps initialize a cluster. It fast-tracks setup by using community-sourced best practices. Kubelet is the work package, which runs on every node and starts containers. The tool gives you command-line access to clusters.
  `$ sudo apt-get install kubeadm kubelet kubectl`
  `$ sudo apt-mark hold kubeadm kubelet kubectl`

C. Apache Bench

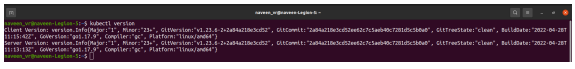
To install Apache Bench:

```
$ sudo apt-update
$ sudo apt-get install -y apache2-utils
```

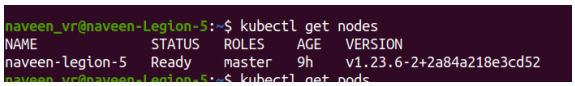
IV. Experiment

A. NGINX

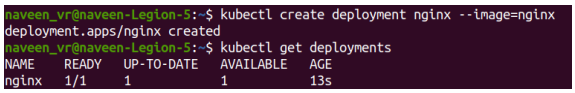
- First we have to check whether kubectl version is up to date,
  `$ kubectl version`
  this is will return



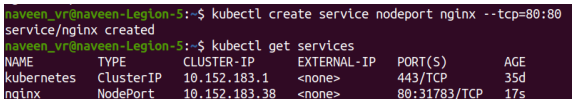
- We have to check whether we have a master node, that can be done listing out all the nodes
  `$ kubectl get nodes`



- We are going to take docker image of NGINX from docker hub and deploy it in the local-host. To deploy and run the image and to check the deployed images, we are going to use the commands,
  `$ kubectl create deployment nginx --image=nginx`
  `$ kubectl get deployments`



- Create service for the nginx deployment. We will create the service in the localhost. To create and check the services we can use terminal commands,
  `$ kubectl create service nodeport nginx --tcp=80:80`
  `$ kubectl get services`



- Since now there is port for nginx image, we can use the website page which is bascially localhost:port(nginx). To trigger traffic in the page, we will be using apache bench. The command to trigger the traffic is,
  `$ ab -n 10000 -c 1000 http://localhost:31783/`
  In this,
  - `-n` is number of page request that you want to make to server.
  - `-c` concurrent requests(at most how many request should be made).

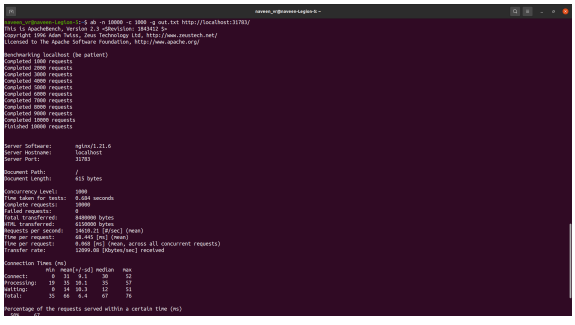


Fig. 1. number of page request to server are 10000 and concurrent requests are 1000

B. HAProxy

- First we have to check whether kubectl version is up to date,
  `$ kubectl version`
- We have to check whether we have a master node, that can be done listing out all the nodes
  `$ kubectl get nodes`
- Create a namespace for haproxy controller, which will be called haproxy-controller-devops in this experiment.
  `$ kubectl create namespace haproxy-controller-devops`

```
naveen_vr@naveen-Legion-5:~$ kubectl create namespace haproxy-controller-devops
namespace/haproxy-controller-devops created
naveen_vr@naveen-Legion-5:~$ kubectl get namespace
NAME                      STATUS   AGE
kube-system               Active   36d
kube-public               Active   36d
kube-node-lease           Active   36d
default                   Active   36d
shpod                     Active   13d
nginx-ingress             Active   19h
ingress-nginx             Active   18h
haproxy-controller-devops   Active   10s
```

- Create a yaml file for deployment and service. yaml file is in GitHub repo.b7
- To delpoy and start the service we use commands,
  `$ kubectl get deploy -n haproxy-controller-devops`
  `$ kubectl get services -n haproxy-controller-devops`

```
naveen_vr@naveen-Legion-5:~$ kubectl get deploy -n haproxy-controller-devops
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
haproxy-ingress-devops   1/1     1            1           2m39s
backend-deployment-devops   1/1     1            1           2m39s
naveen_vr@naveen-Legion-5:~$ kubectl get services -n haproxy-controller-devops
NAME                    TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)                                  AGE
service-backend-devops   ClusterIP   10.152.183.22   <none>        8080/TCP                                 3m4s
ingress-service-devops   NodePort    10.152.183.135  <none>        80:32456/TCP,443:32567/TCP,1024:32678/TCP   3m3s
```

- Since now there is port for docker image. To trigger traffic in the page, we will be using apache bench. The command to trigger the traffic is,
  `$ ab -n 10000 -c 1000 http://localhost:32567/`

Fig. 2. number of page request to server are 10000 and concurrent requests are 1000

C. Traefik

- First we have to check whether kubectl version is up to date,
  `$ kubectl version`
- We have to check whether we have a master node, that can be done listing out all the nodes
  `$ kubectl get nodes`
- We are going to take docker image of Traefik from docker hub and deploy it in the local-host. To deploy and run the image and to check the deployed images, we are going to use the commands,
  `$ kubectl create service nodeport traefik --tcp=80:80`
  `$ kubectl get services`

```
naveen_vr@naveen-Legion-5:~$ kubectl create service nodeport nginx --tcp=80:80
service/nginx created
naveen_vr@naveen-Legion-5:~$ kubectl get services
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
kubernetes   ClusterIP   10.152.183.1    <none>        443/TCP        35d
nginx        NodePort    10.152.183.38   <none>        80:31783/TCP   17s
```

- Since now there is port for traefik image, we can use the website page which is bascially localhost:port(traefik). To trigger traffic in the page, we will be using apache bench. The command to trigger the traffic is,
  `$ ab -n 15000 -c 100 http://localhost:32323/`

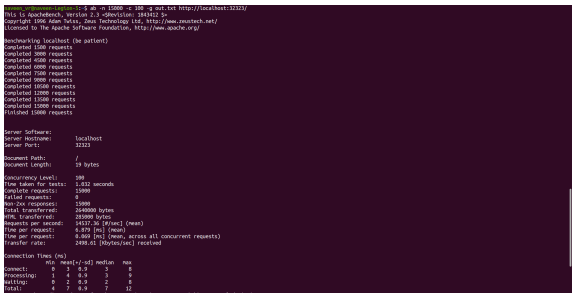


Fig. 3. number of page request to server are 15000 and concurrent requests are 100

V. Comparison

We are going compare the top three load balancers, NGINX, HAProxy and Traefik, by the results given by apache bench. The same basic application of each load balancer is used for comparison. Apache bench, we give parameters like number of page request to the server and concurrent requests, and it gives results like,

- Concurrency level
- Time taken for tests
- Complete requests
- Failed requests
- Non-2xx responses
- Total transferred
- HTML transferred
- Requests per second
- Time per request
- Transfer data
- min, mean, median, max

We are going to use the parameters like,

- Concurrency level
- Time taken for tests
- Requests per second

- Time per request
- longest request time

to compare the load balancers

Each time, the values of the result will be different so I have taken the average among 5 times.

A. Number of page request=10000; concurrent request=1000

Paramters	Load Balancer		
	NGINX	***HAProxy***	***Traefik***
Concurrency Level	1000	1000	1000
Time taken for tests	0.696secs	1.03secs	0.729secs
Requests per second	14610.21	9570.61	13726.52
Time per request	69.645ms	104.4ms	72.852ms
Longest request time	107	288	83

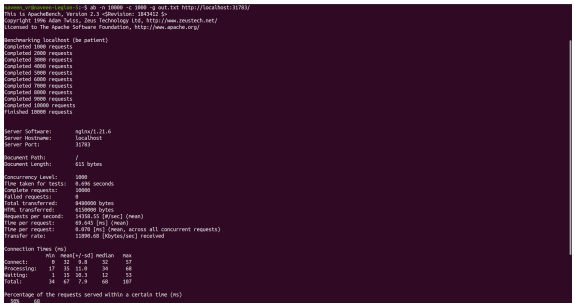


Fig. 4. NGINX

Fig. 5. HAProxy

Fig. 6. Traefik

B. Number of page request=30000; concurrent request=1000

Paramters	Load Balancer		
	NGINX	***HAProxy***	***Traefik***
Concurrency Level	1000	1000	1000
Time taken for tests	2.178secs	2.456	2.162secs
Requests per second	13610.21	12153.34	13826.52
Time per request	72.645ms	81.981ms	71.852ms
Longest request time	108	114	92

Fig. 7. NGINX

Fig. 8. HAProxy

Fig. 9. Traefik

C. Number of page request=50000; concurrent request=500

Paramters	Load Balancer		
	NGINX	***HAProxy***	***Traefik***
Concurrency Level	500	500	500
Time taken for tests	3.478secs	4.031secs	3.512secs
Requests per second	14110.21	12343.92	13686.52
Time per request	34.645ms	39.6ms	36.852ms
Longest request time	49	65	50

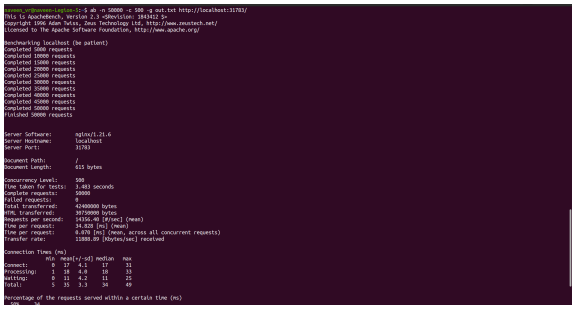


Fig. 10. NGINX

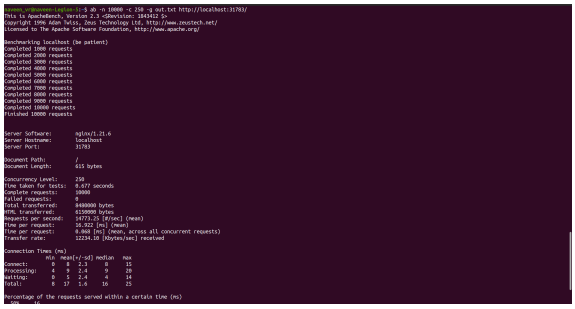


Fig. 13. NGINX

Fig. 11. HAProxy

Fig. 14. HAProxy

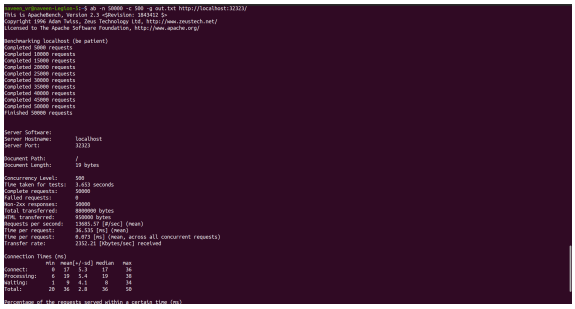


Fig. 12. Traefik

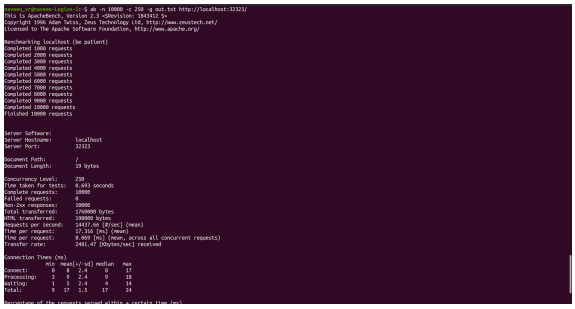


Fig. 15. Traefik

D. Number of page request=10000; concurrent request=250

E. Number of page request=100000; concurrent request=1000

Paramters	Load Balancer		
	NGINX	***HAProxy***	***Traefik***
Concurrency Level	250	250	250
Time taken for tests	0.678secs	0.762secs	0.692secs
Requests per second	14710.21	12765.43	14486.52
Time per request	16.645ms	19.61ms	17.852ms
Longest request time	25	34	24

Paramters	Load Balancer		
	NGINX	***HAProxy***	***Traefik***
Concurrency Level	1000	1000	1000
Time taken for tests	7.178secs	8.101secs	7.392secs
Requests per second	14110.21	12431.29	13786.52
Time per request	71.645ms	81.902ms	73.152ms
Longest request time	1085	138	117

Fig. 16. NGINX

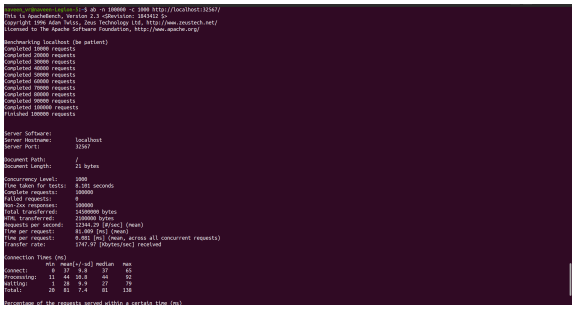


Fig. 17. HAProxy

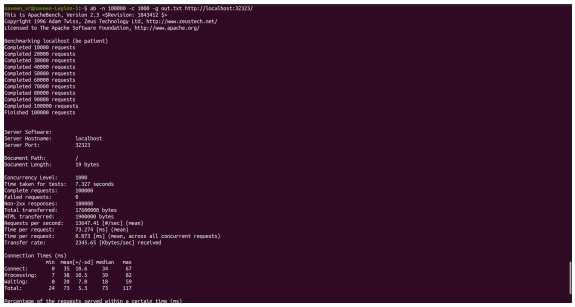


Fig. 18. Traefik

VI. Conclusion

As we can compare from the five tables, NGNIX is the best load balancer for the kuberentes. We can't always say the NGINX will perform better than other load balancer, there can be situations where other load balancer can perform better than NGINX. On an average, NGINX is the best load balancer as per the experiment that's shown in this paper.

References

[1] https://kubernetes.io/ .
[2] https://hub.docker.com//traefik?tab=description .
[3] https://hub.docker.com//nginx
[4] https://hub.docker.com//haproxy
[5] https://medium.com/microservices-for-net-developers/deploying-your-first-app-to-kubernetes-108d2c31ac3b
[6] https://www.bluematador.com/blog/running-haproxy-docker-containers-kubernetes
[7] https://github.com/naveen-kumar-vr/Best-Load-Balancer-for-Kubernets