ISSN No:-2456-2165

An Efficient and Scalable Traffic Load Balancing Based on Webserver Container Resource Utilization using Kubernetes Cluster

Ashok L Pomnar[1,]
[1]Student of ME Computer,
Department of Computer Engineering,
AVCOE Sangamner 422 605,
Maharashtra, India.

Dr. S.K. Sonkar[2]
Assistant Professor,
Department of Computer Engineering,
AVCOE Sangamner 422 605,
Maharashtra, India

Abstract:- A Today digital transformation era, traditional web based systems and application architecture become the bottleneck of scaling, high availability, portability, and many more downtime and performance-related challenges so the world is started adopting the cloud and cloud-based services to get its benefits. Virtualization technology played a very crucial role to migrate applications from traditional physical systems to virtualized systems but again because of limitations and challenges associated with virtualized system scalability, high availability, and application portability, the Container technique is gaining increasing attention in recent years and has become an alternative to traditional virtual machines to run the application.

Few of the primary motivations for the enterprise to adopt container technology include its convenience to encapsulate and deploy applications, lightweight, as well as efficiency and flexibility in resource sharing. Considering current vertical and horizontal application scaling challenges for the high traffic websites, in this paper we explains the benefits of cloud technology, virtualization, container and Kubernetes clustering features. Also, discuss the algorithm for building dynamic scaling large traffic platforms and applications, which will be runtime, scaled up and down as per user request and traffic demand.

Recent growth in e-business, online web application and mobile user accessibility are exponentially grown with the widespread of Internet of Things, Data Processing and data analytic and many more online portal. To improve better web application performance, web application availability and scalability resource utilization many service provider start design services using micro services and deploy them on the container using Kubernetes cluster. However, the exiting approaches fail to address service availability, handling high traffic load result to dissatisfaction to end users and business impact. To address these issues, in proposed work evaluate many interesting aspects of running high traffic website applications on containerized dynamic scaling and high availability cluster. Proposed Setup run on the cloud again, such as how convenient the execution environment can be set up, what are makes pans of different workloads running in each setup, how efficient the hardware resources, such as CPU and memory, utilized, and how well each environment can scale. The results show that compared with virtual machines, containers are more easy-to-deploy and scalable environment for high traffic workloads.

Keywords:- *Docker, Cloud Computing, Kubernetes Cluster, K8S, Micro service, Web Server, Haproxy Load Balancer, NFS Storage.*

I. INTRODUCTION

In today's digital transformation area, Docker and Kubernetes have revolutionized the way of DevOps consulting and both are leading container orchestration tools[1]. There is always a challenge to control an increase in the demand for scaling and auto-healing of the network and virtual instances. Managing the containers is always a task for any company because microservicesthat are running on the containers do not communicate with each other. They work independently as a separate entity. This is where Kubernetes steps in. Kubernetes is nothing but a platform to manage containers. These containers can be Docker containers or any other alternative containers. Kubernetes orchestrates, manages, and forms a line of communication between these containers [5].

- **The first Section covers Docker:** Containerization Using Docker, Docker for networking, The Docker File, and hosting a web server using Docker.
- **In the Second Section,** we have covered Kubernetes: The Role and Architecture of Kubernetes, basic concepts, features, Kubernetes clusters, scaling and deploying applications, and hosting a Web Server Using Helm.
- **In the third section,** I have planned to deploy the Cloud platform to deploy the complete. Kubernetes and Dockers on Virtual machines which run on either Xen or VMware Cloud. Its compatibility, services, features, and how dockets Kubernetes and Cloud go hand in hand and last.
- **The last section** deploys the back-end script to monitor the traffic on the load balancer server and then automatically scales up and down the webserver containers to distribute the traffic. To summarize, the project will be by taking advantage of each of the cloud and cloud services emerging technology to deploy the highly scalable and high availability platform to run high traffic applications.

II. LITERATURE REVIEW

- Ruchika Muddinagiri, Shubham Ambavane, Simran Bayas [3] In this paper,the author has deployed the containerization application using docker and minikube tools on the local system, so there is future scope to build Kubernetes cloud-deployed scalable application on same which can provide availability as well as scalability.
- RobertBotez,Calin-MarianIurian,Iustin-AlexandruIvanciu,VirgilDobrota [5]: In this paper, authors have evaluated a solution for monitoring vehicles in real-time using containerized applications. However, there is the future scope for work on evaluating the Docker performance and security as well set up a Kubernetes cluster on a private cloud. The cluster nodes are made up of multiple Raspberry Pi platforms that will collect more sensor information.
- Jay Shah, Dushyant Dubaria [4]: In this paper,the WordPress blog application is deployed using containerization technology with a standalone and static resource on the Google Cloud Platform. The major advantage of using Kubernetes is orchestrating between many applications. It is also very useful for scaling many applications in very less time. By using this, we can save our costs and time.

III. PROPOSED METHODOLOGY

A. *Architecture*

In this paper, three virtual Machines need to deploy the Virtual Machin on the Cloud Service Provider (ESDS eNlight360/AWS/Microsoft Azure) with high available Cloud. Install and set up the Kubernetes cluster on all these three-node with one master and 2 worker/slave kinds of architecture where we will get the benefit of container failover advanced in case any VM is failed/rebooted.

We can deploy any PHP application container using apache-PHPDocker container using Kubernetes deployment,pods, services, and endpoints features and make the application accessible with IP address and port number using Kubernetes networking. A persistent volume is used to save and store application data after failover of VM, Kubernetes nodes, or container restarting.

We can deploy another HaProxy container to distribute the traffic of web server traffic among different application containers, which are dynamically scaling up and down as container resource utilization and traffic increase. Kubernetes scheduler and monitoring continuously monitor the traffic utilization and CPU and memory utilization of the container and increasing application container using application deployment, which quickly gets available behind the HaProxy load balancer to server the web traffic.

When web request, CPU, and memory utilization are decreased, the Kubernetes scheduler and Kubernetes manager are reducing the container and back to the default min container count.

We do use the Round Robin algorithm to equally balance the load among all application containers.

B. *Main System Components*

- **Centos VM:** Setup the three CentOS-based Virtual machines on any Cloud platform.
- **Setup and Configuration Kubnernet Cluster:** Setup and configure the three-node one Master node and 2 Worker-Slave nodes Kubernetes cluster on these three VM.
- **Deploy Application Container:** Deploy the apache-PHP web servers, MySQL database Docker container, and installed on the Kubernetes cloud.
- **Application Accessibility:** With help of Kubernetes networking, services and endpoint make the application accessible with a web serverURL.
- **Web Server Load Balancer:** Deploy the haproxy Docker container on Kubernetes cloud with dynamical container scaling feature automatically detect to enable or disable the application container whenever traffic demand is increasing or decreasing.
- **Web Traffic generating tool:** Using the Siege or curl method to generate real-time traffic on the webserver.

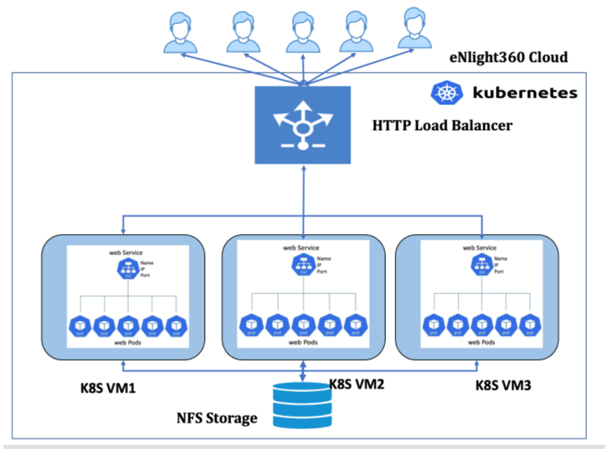

Fig. 1: Solution Architecture Diagram

The proposed system will be comprised of the following modules:

- **Module 1: Setup phase:** Build and set up the Virtual machine on the cloud to set up the Kubernetes clustering with required resources.
- **Module 2: Build Kubernetes Cloud and Containers:** Build and set up the Kubernetes clustering with high availability option.
- **Module 3: Deploy Application:** Deploy containerized services like web container, database container, and load balancer container to run the WordPresscontainerized application.
- **Module 4: Deploy backend scripts:** Deploy the proposed algorithm in the form of script, whichmonitor real-time traffic patterns, and as per container traffic handling limit/threshold set it will dynamically scale up, scale down the web servers container, and add the same behind load balance too to balance the traffic.
- **Module 5: Dynamic scaling and High availability:** Demonstrate the container scale up whenever the traffic spike or increased and scale down when traffic is reduced. Also, demonstrate the application's high availability in terms of any web server container failure, any

Kubernetesnode (VM) failure, VM failure, or any cloud node failure.

- **Module 6: Analysis Reports:** Prepare the details analysis report to consider the traffic pattern, traffic balancing among several web servers, container scale up and scale down as per traffic pattern, and high availability report in terms of components failure.

IV. ALGORITHMS

A. Web Request Balancing Algorithm: Round robin

We do use the Round Robin algorithm to balance the load among all application containers. The round-robinload-balancing algorithm is one of the popular methods for distributing web traffic across a group of web servers. Going down the list of servers in the group, the round-robin load balancer forwards a request to each server in turn. When it reaches the end of the list, the load balancer loops back and goes down the list again (sends the next request to the first listed server, the one after that to the second server, and so on).

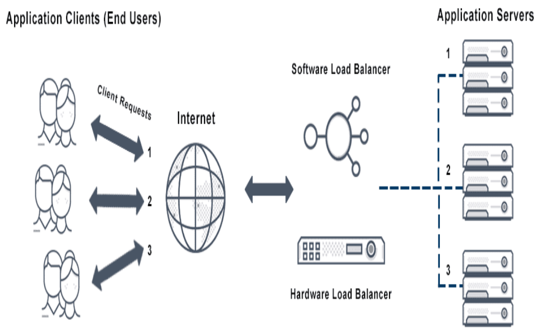

Fig. 2: Web Request balancing using Round Robin Algorithm

B. Horizontal Container Scaling Algorithm:

Desired App Replicas = ceil [current App Replicas* (current Scaling Metric Value / desired Scaling Metric Value)]

Desired App Replicas: Application Replicas pod count that will sent to the controller after calculations.

Ceil (): This function that rounds a fractional number. For example,ceil (11.28) is 12.

Current App Replicas: Current number of Application Replicas pods for a given deployment or any other superset of "scale" object type.

Current Scale Metric Value: Current value of metric for a given scaling factor metric. Can be 800m or 1.5Gi, for custom metrics, it can be 500 events per second, etc.

Desired Scale Metric Value: Metric that has been set to maximum scalability of application pod. Eventually, with all mechanisms provides, your app runs at this metric value. This value should not be too low or too high.

Let us say we have a scaling configuration with a target CPU usage of 70%, a minimum pod count of 4, and a maximum pod count of 20.

The current deployment status issix pods averaging 95% usage.

Desired Replicas = ceil [6*(85/70)] = ceil (8.14) = 9

Scaling down Algorithm:

Let us say we have a pod-scaling configuration with a target CPU usage of 70%, a minimum pod count of 4, and a maximum pod count of 20.

The current application pod deployment status is: There are 10 total pods. 10 pods averaging 45% usages.

The same algorithm we used to scale down the application pod:

The formulais applied to all normal pods.

Desired Replicas = ceil [10*(45/70)] = ceil (6.42) = 7 > 10

With the above calculation three application pods are extra, so as per the scaling downtime set in the configuration it will scale down the application pod to seven.

V. WORKING

A. Kubernetes Cloud and Container Status:

We can set up and create three Centos VM virtual Machines on the eNlight360 Cloud to build the High Available Kubernetes Master-Slave clustering. Below is the Kubernetes cluster node status

```
[root@k8snode1 ~]# kubectl get node
NAME                   STATUS   ROLES                  AGE    VERSION
k8snode1.alexparker.com   Ready    control-plane,master   201d   v1.22.2
k8snode2.alexparker.com   Ready    <none>                 201d   v1.22.2
k8snode3.alexparker.com   Ready    <none>                 195d   v1.22.2
[root@k8snode1 ~]#
```

Fig. 3: Kubernetes cluster status

B. Deploy the apache-PHP base application container:

We can deploy and install the PHP-Apache web server application container using k8s deployment components which run on the respective worker nodes with a minimum number of the replica set and maximum, minimum replica scaling number when pod CPU and memory demands increased because of a total number of web request increases.

```
[root@k8snode1 ~]# kubectl get pod,deploy -n mydemo
NAME                                   READY   STATUS    RESTARTS       AGE
pod/centos-loadtest-966f54885-8g4mx    1/1     Running   1 (161d ago)   194d
pod/mydemo-app-6bf8bdc664-f29hb        1/1     Running   0              28d

NAME                              READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/centos-loadtest   1/1     1            1           194d
deployment.apps/mydemo-app        1/1     1            1           194d
[root@k8snode1 ~]#
```

Fig. 4: K8s pod, deploy status

```
[root@k8snode1 ~]# kubectl get service -n myproject haproxy-kubernetes-ingress  -o wide
NAME                        TYPE       CLUSTER-IP       EXTERNAL-IP   PORT(S)
             AGE    SELECTOR
haproxy-kubernetes-ingress  NodePort   10.106.165.170   <none>        80:30000/TCP,443:30001/TCP,1024
:30002/TCP   185d   app.kubernetes.io/instance=haproxy,app.kubernetes.io/name=kubernetes-ingress
[root@k8snode1 ~]#
```

Fig. 5: Load balancer deployment status

172.16.23.16:30002

mydemo-mydemo-app-80

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings		
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status
SRV_1	0	0	-	0	3		0	1	-	441	441	25m31s	37 407	80 262		0		0	0	0	0	24m33s MAINT
SRV_2	0	0	-	0	1		0	1	-	315	315	25m18s	25 764	57 330		0		0	0	0	0	24m33s MAINT
SRV_3	0	0	-	0	1		0	1	-	326	326	25m27s	26 648	59 332		0		0	0	0	0	9h48m UP
SRV_4	0	0	-	1	2		0	2	-	305	305	28d20h	24 945	55 510		0		0	0	0	0	24m33s MAINT
SRV_5	0	0	-	1	3		0	1	-	570	570	28d20h	47 148	103 740		0		0	0	0	0	24m33s MAINT
SRV_6	0	0	-	1	1		0	1	-	223	223	30d17h	18 308	40 586		0		0	0	0	0	30d17h MAINT
SRV_7	0	0	-	1	1		0	1	-	175	175	30d17h	14 175	31 850		0		0	0	0	0	30d17h MAINT
SRV_8	0	0	-	1	1		0	1	-	174	174	30d17h	14 339	31 668		0		0	0	0	0	30d17h MAINT
SRV_9	0	0	-	1	1		0	1	-	173	173	30d17h	14 013	31 486		0		0	0	0	0	30d17h MAINT

Fig. 6: Load balance traffic status

C. Generate the web request and check application pod scaling stats:

We can use the siege /curl command to generate the web request on the application server, which equally get a balance among the number of application pod, whichis automatically added behind the load balancer. When the total number of web requests demand increases, the application pod CPU and memory utilization start increasing, once it crossesdesired limit it scalesthe replica threshold value, and the system automatically starts deploying additional application pods to sustain the web request demand and enhance system performance.

```
[root@centos-loadtest-966f54885-8g4mx /]# for i in {1..1000}; do curl http://10.106.165.170;sleep 1;do
ne
OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!
```

Fig. 7: Generate the web traffic using the curl command

Total number of application pod status with normal stats

```
[root@k8snode1 ~]# kubectl get pod -n mydemo -o wide
NAME                              READY   STATUS    RESTARTS       AGE    IP           NODE
centos-loadtest-966f54885-8g4mx   1/1     Running   1 (161d ago)   194d   10.44.0.8    k8snode2.alexparker.com
mydemo-app-6bf8bdc664-f29hb       1/1     Running   0              28d    10.44.0.15   k8snode2.alexparker.com
[root@k8snode1 ~]#
```

Fig. 8: Total no of pod running at normal traffic scenario

The total number of application pod status after web requests started increasing

```
NAME         REFERENCE                TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
mydemo-app   Deployment/mydemo-app    30%/20%   1         100       4          194d
```

Fig. 9: Container (pod) scaling status

```
[root@k8snode1 ~]# kubectl get pod,deploy -n mydemo
NAME                                  READY   STATUS    RESTARTS       AGE
pod/centos-loadtest-966f54885-8g4mx   1/1     Running   1 (161d ago)   194d
pod/mydemo-app-6bf8bdc664-7nbsp       1/1     Running   0              55s
pod/mydemo-app-6bf8bdc664-bdr9v       1/1     Running   0              55s
pod/mydemo-app-6bf8bdc664-f29hb       1/1     Running   0              28d
pod/mydemo-app-6bf8bdc664-x9g9v       1/1     Running   0              70s

NAME                              READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/centos-loadtest   1/1     1            1           194d
deployment.apps/mydemo-app        4/4     4            4           194d
[root@k8snode1 ~]#
```

Fig. 10: Total of no pod scale after web traffic load increased

```
NAME         REFERENCE                TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
mydemo-app   Deployment/mydemo-app    13%/20%   1         100       5          194d
```

Fig. 11: The total no of the pod that got auto deployed after load increased

172.16.23.16:30002

mydemo-mydemo-app-80

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings		
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status
SRV_1	0	0	-	0	3		0	1	-	451	451	3s	38 217	82 082		0		0	0	0	0	1m41s UP
SRV_2	0	0	-	0	1		0	1	-	322	322	2s	26 331	58 604		0		0	0	0	0	1m26s UP
SRV_3	0	0	-	1	1		0	1	-	346	346	1s	28 268	62 972		0		0	0	0	0	9h50m UP
SRV_4	0	0	-	0	2		0	2	-	311	311	8s	25 431	56 602		0		0	0	0	0	1m25s UP
SRV_5	0	0	-	0	3		0	1	-	572	572	5s	47 310	104 104		0		0	0	0	0	24s UP

Fig. 12: Load balance web traffic status

Check application pod stats after web request demand reduce: Once web request demand decrees

The total number of application pod statuses after web request demand decreased.

```
NAME         REFERENCE                TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
mydemo-app   Deployment/mydemo-app    0%/20%    1         100       5          194d
```

Fig. 13: Pod scaling down after web traffic decreased

```
[root@k8snode1 ~]# kubectl get pod,deploy -n mydemo
NAME                                  READY   STATUS        RESTARTS       AGE
pod/centos-loadtest-966f54885-8g4mx   1/1     Running       1 (161d ago)   194d
pod/mydemo-app-6bf8bdc664-7nbsp       1/1     Terminating   0              7m36s
pod/mydemo-app-6bf8bdc664-f29hb       1/1     Running       0              28d
pod/mydemo-app-6bf8bdc664-hd5d2       1/1     Running       0              6m35s
pod/mydemo-app-6bf8bdc664-x9g9v       1/1     Running       0              7m51s

NAME                              READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/centos-loadtest   1/1     1            1           194d
deployment.apps/mydemo-app        3/3     3            3           194d
[root@k8snode1 ~]#
```

Fig. 14: Total no of POD reduce stats after traffic decreased

```
[root@k8snode1 ~]# kubectl get pod,deploy -n mydemo
NAME                                  READY   STATUS        RESTARTS       AGE
pod/centos-loadtest-966f54885-8g4mx   1/1     Running       1 (161d ago)   194d
pod/mydemo-app-6bf8bdc664-f29hb       1/1     Running       0              28d
pod/mydemo-app-6bf8bdc664-x9g9v       1/1     Terminating   0              8m19s

NAME                              READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/centos-loadtest   1/1     1            1           194d
deployment.apps/mydemo-app        1/1     1            1           194d
[root@k8snode1 ~]#
```

Fig. 15: Total no of POD terminate after traffic decreased

VI. CONCLUSION

By adopting the cloud native services benefit, Virtualization, containerization technology are used to build and migrate applications from a traditional physical system to virtualized system, the containerized system which convenient to encapsulate and deploy applications, lightweight operations, as well as efficient and flexible in resources scaling.

Considering current vertical and horizontal application scaling challenges for high traffic websites, in this paper taking the benefit of the cloud, virtualization, container, and Kubernetes clustering features we proposed a dynamic scaling algorithm considering backend latency for large traffic platforms and applications which will be dynamically scaled up and down as per user request and traffic demand.

REFERENCES

[1.] Ruchika Muddinagiri, Shubham Ambavane, Simran Bayas , ”Self-Hosted Kubernetes: Deploying Docker Containers Locally With Minikube” , IEEE Xplore, Conference,18 August 2020 ISBN Information: ,DOI: 10.1109/ICI- TAET47105.2019.9170208”

[2.] Jay Shah,Dushyant Dubaria”,Building Modern Clouds: Using Docker, Kubernetes Google Cloud Platform”,IEEE Xplore Conference: 14 March 2019 ,DOI: 10.1109/CCWC.2019.8666479

[3.] Robert Botez,Calin-Marian Iurian,Iustin-Alexandru Ivanciu,Virgil Dobrota ,”Deploying a Dockerized Application With Kubernetes on Google Cloud Platform”, IEEE Xplore Conference: 16 July 2020,DOI: 10.1109/COMM48946.2020.9142014

[4.] K. Matthias, and S.P. Kane, “Docker:Up and Running”, O’Reilly, 2015

[5.] Deploy on Kubernetes Documentation , Aug 2021 [Online], Available: https://docs.docker.com/desktop/kubernetes/

[6.] Kubernetes Cloud Setup Documentation, Aug 2021, [Online] Available: https://kubernetes.io/docs/setup/production-environment/

[7.] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, J. Wilkes, “Borg, omega, and kubernetes”, 2016.

[8.] Q. Zhang, L. Liu, C. Pu, Q. Dou, L. Wu and W. Zhou, "A Comparative Study of Containers and Virtual Machines in Big Data Environment," 2018 IEEE 11th International Conference on Cloud Computing (CLOUD), San Francisco, CA, 2018, pp. 178-185, doi: 10.1109/CLOUD.2018.00030.

[9.] ‘DevOps; Puppet, Docker and Kubernetes – Learning path’ by Thomas Uphill, Arundel, Khare, Saito, Lee and Carol Hsu, Packt Publications, First Edition, 2017

[10.] ‘Cloud Native Applications- The Intersection of Agile Development and Cloud Platforms’ by Douglas Bourgeois, David Kelly, Thomas Henry members of Deloitte Touche Tohmatsu Limited,2016

[11.] ‘Kubernetes from the ground up, deploy and scale performant and reliable containerized applications with Kubernetes’ by Level Up Kubernetes Program, Basit Mustafa, Tao W, James Lee, Stefan Thorpe, 2018