IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

DEPLOYING HIGH SCALABLE AND HIGH AVAILABLE CONTAINERIZED WEB APPLICATION USING KUBNERTES CLUSTER ON THE CLOUD

[1]Dr. S.K. Sonkar,
Assistant Professor, [1]Computer Engineering,
[1]AVCOE ,Sangamner , India

[2] Ashok L Pomnar
Student of ME computer
AVCOE ,Sangamner , India

Abstract: Today digital transformation era, traditional 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.
Some of the primary motivations for the enterprise to adopt container technology include its convenience to encapsulate and deploy applications, lightweight operations, as well as efficiency and flexibility in resource sharing. Considering current vertical and horizontal application scaling challenges for high traffic websites, in this paper explains the benefits of cloud technology, virtualization technology, container technology and Kubernetes clustering features, building dynamic scaling large traffic platforms and applications which will be dynamically scaled up and down as per user request and traffic demand.

I am going to evaluate many interesting aspects of running high traffic website applications on containerized dynamic scaling and high availability cluster which would be run on the cloud again, such as how convenient the execution environment can be set up, what are makespans of different workloads running in each setup, how efficient the hardware resources, such as CPU and memory, are utilized, and how well each environment can scale. The results show that compared with virtual machines, containers provide a more easy-to-deploy and scalable environment for high traffic workloads.

keywords - Docker,Cloud Computing, Kubernets Cluster ,K8S,Microservice,Web Server,Haproxy Load Balancer,NFS Storage

I. INTRODUCTION

Today's digital transformation area, Docker and Kubernetes have revolutionized the way of DevOps consulting and both are leading container orchestration tools. There is always a challenge to control an increase in the demand of scaling and auto- healing of the network and virtual instances. Managing the containers is always as task for any company because microservices which 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 man- age containers. These containers can be docker

containers or any other alternative containers. Kubernetes orchestrates, manages and forms a line of communication between these containers.

This paper is divided into four sections.
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 deployed the Cloud platform to deploy the complete. Kubernetes and dockers on Virtual machine which would be run on either Xen or VMware Cloud. It's compatibility, services, features and how dockets Kubernetes and Cloud go hand in hand and last.

The last section deploys the backend script which would be monitoring the traffic on the load balancer server and then automatically scale up and down the webserver containers to distribute the traffic. To summarize will be implementing the project by taking advanced 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

1. Ruchika Muddinagiri, Shubham Ambavane, Simran Bayas [3] : In this paper 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.

2. 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 future scope work on evaluating the Docker performance and security as well set up a Kubernetes cluster on a private cloud. The cluster nodes will be made up of multiple Raspberry Pi platforms that will collect more sensor information.

3. Jay Shah,Dushyant Dubaria [4] : In this paper WordPress blog application is deployed using containerization technology with 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

As mentioned in the proposed solution architect, three virtual Machine need to deploy the Virtual Machin on the Cloud Service Provider like (ESDS eNlight360/AWS/Microsoft Azure) with high available Cloud. Install and setup the kubernets cluster on all these three-node with one master and 2 worker/slave kinds of architecture where we will get benefit of container failover advanced in case any VM is failed/rebooted.
We can deploy any php application container using apache-php docker container using Kubernetes deployment,pod, services and endpoints features and make the application accessible with ip address and port number using kubernet networking. A persistent volume is being used to save and stored application data after failover of VM, Kubernetes nodes or container restarting.
We can deploy another HaProxy container to distribute the traffic web server traffic among different application containers which are dynamically scaling up and down as container resources utilization and traffic increase. Kubernetes schedular and monitoring are continuous 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 RoundRobin algorithm to equally balance the load among all application containers.

Main System Components

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

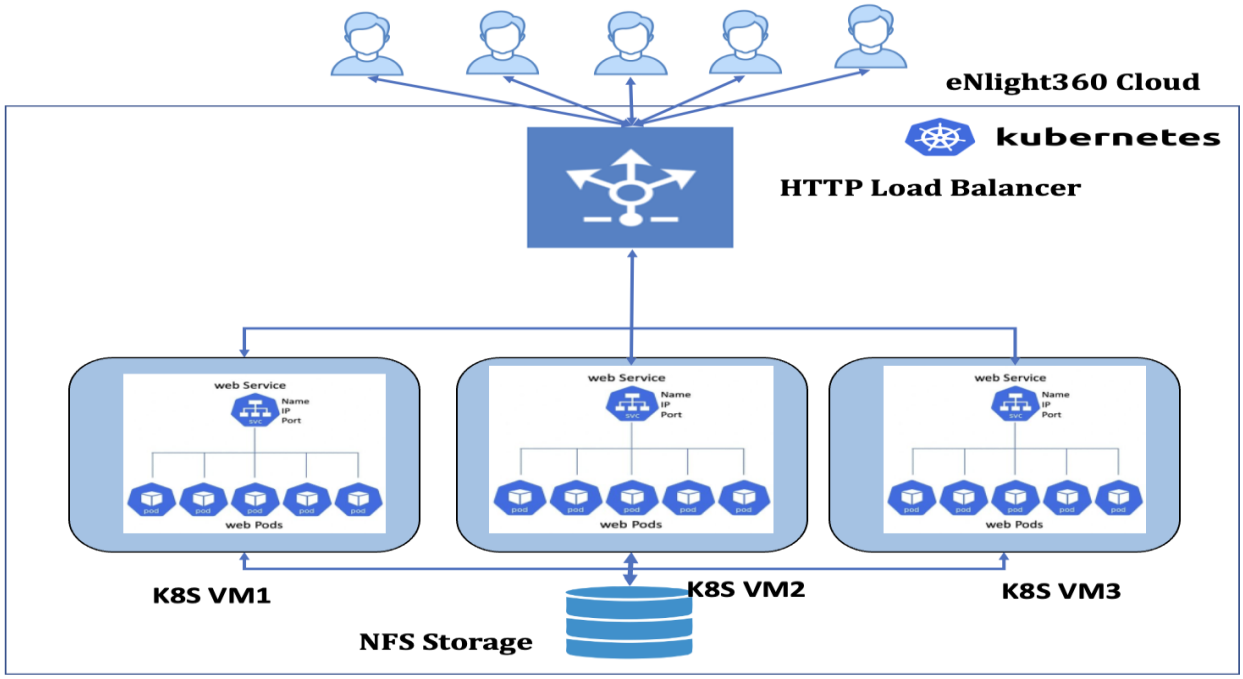

Fig 1. Solution Architecture Diagram

Proposed system will be comprising of following modules:

1. Module 1: Setup phase: build and setup the Virtual machine on the cloud to setup the Kubernetes clustering with required resources.
2. Module 2: Build Kubernetes cloud and containers: Build and setup the Kubernetes clustering with high availability option .
3. Module 3: Deploy Application: Deploy the require containerized services like web container, database container, load balancer container to run the wordpress containerized application.
4. Module 4: Deploy backend scripts: Deploy the backend script which would read real time traffic pattern and as per container traffic handling limit/threshold set it will dynamically scale up and scale down the web servers container and add same behind load balance too to balance the traffic .
5. Module 5: Dynamic scaling and High availability: Demonstrate the container scale up whenever the traffic are increases and scale down when traffic is re- duces. Also demonstrate the application high availability in term of any web server container failure , any kubernets node(VM) failure, VM failure or any cloud node failure.
6. Module 6: analysis Reports: Prepare the details analysis report consider the traffic pattern, traffic balancing among number of web servers, container scale up and scale down as per traffic pattern, high availability report in term of components failure.

B.Algorithms

Web Request Balancing Algorithm: Round robin

We do use the RoundRobin algorithm to equally balance the load among all application containers. Round robin load balancing is one of the popular methods for distributing web requests across a group of web servers. Going down the list of servers in the group, the round-robin load balancer forwards a client 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).

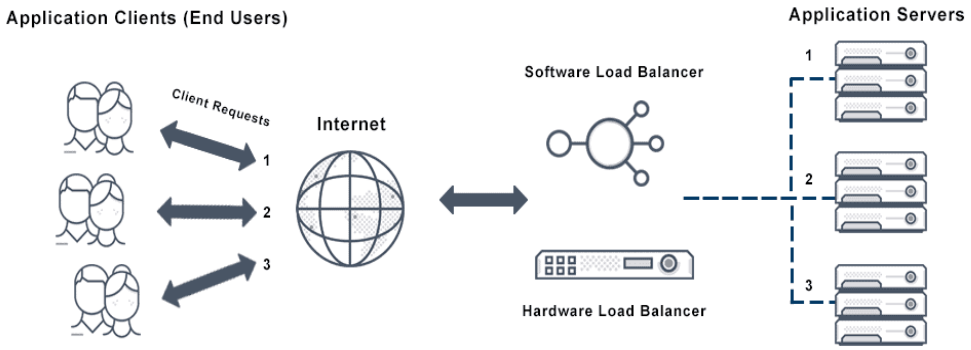

Fig 2. Web Request balancing using RoundRobin Algorithm

Horizontal Container Scaling Algorithm :

desiredAppReplicas = ceil[currentAppReplicas * (currentScalingMetricValue / desiredScalingMetricValue)]

desiredAppReplicas: Application Replicas pod count that will be sent to the controller after calculations.

ceil(): This is a function that rounds a fractional number upwards. For example ceil(12.18) is 13.

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

currentScaleMetricValue: 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.

desiredScaleMetricValue: 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's say we have scaling configuration with a target CPU usage of 70%, a minimum pod count of 4 and a maximum pod count of 20.
Current deployment status is: 6 pods averaging %95 usage.
desiredReplicas = ceil[6*(85/70)] = ceil(8.14) = 9
Scaling down Algorithm :
Let's say we have 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.
Same algorithm us used to scale down the application pod:

1. Formula is applied on all normal pods.

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

2. With above calculation 3 application pod are extra, so as per scaling down time set in the configuration it will scale down the application pod to 7.

IV. WORKING

Kubernetes Cloud and Container Status :
We can setup and create three Centos VM virtual Machine on the eNlight360 Cloud to build the High Available Kubernetes Master-Slave clustering. Below is the kubernet 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. Kubernet cluster status

Deploy the apache-php base application container :

We can deploy and install the php-apache base web server application container using k8s deployment components which run on the respective worker node 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

Fig 6. Load balance traffic status

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 are equally get a balance among of the number of application pod which are automatically added behind the load balancer. When total number of web requests demand are increases , the application pod CPU and memory utilization start increasing, once it cross desiredscale replica threshold value , system automatically start deploying additional application pod 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!OK!
```

Fig 7. Generate the web traffic using 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 incresed

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

Fig 11. Total no of pod got auto deployed after load incresed

Fig 12. Load balance web traffic status

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

Total number of application pod status after web request demand decrease .

```
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

V. CONCLUSION

By adopting the cloud and cloud-based services benefit, Virtualization technology, and 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 sharing.

Considering current vertical and horizontal application scaling challenges for high traffic websites, in this project taking benefit of cloud technology, virtualization technology, container technology and kubernetes clustering features, build dynamic scaling large traffic platform and application which will be dynamically scaled up and down as per user request and traffic demand.

VI. 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

[12] Publication of ‘Kubernetes Fundamentals’ by The Linux Foundation Training, 2017

[13] Kubernetes Course on INE by David Corone