ISSN: 2277-9655
Impact Factor: 4.116
CODEN: IJESS7

IJESRT
INTERNATIONAL JOURNAL OF ENGINEERING SCIENCES & RESEARCH TECHNOLOGY

DATA STORAGE & LOAD BALANCING IN CLOUD COMPUTING USING CONTAINER CLUSTERING

Trapti Gupta[*1] & Abhishek Dwivedi[2]
[*1]M-Tech Research scholar, Lakshmi Narain college of Technology and IT, Bhopal , RGPV University Bhopal
[2]Assistant Prof., Lakshmi Narain college of Technology and IT, Bhopal , RGPV University Bhopal

DOI: 10.5281/zenodo.996047

ABSTRACT

At the moment, cloud containers are a hot topic in the IT world in general, and security in particular. The world's top technology companies, including Microsoft, Google and Facebook, all use them. Although it's still early days, containers are seeing increasing use in production environments. Containers promise a streamlined, easy-to-deploy and secure method of implementing specific infrastructure requirements, and they also offer an alternative to virtual machines.

The key thing to recognize with cloud containers is that they are designed to virtualize a single application -- e.g., you have a MySQL container and that's all it does, provide a virtual instance of that application. Containers create an isolation boundary at the application level rather than at the server level. This isolation means that if anything goes wrong in that single container (e.g., excessive consumption of resources by a process) it only affects that individual container and not the whole VM or whole server. It also stops compatibility problems between applications that reside on the same operating system (OS).But despite their success, containers still present challenges. Container scalability, for example, remains somewhat of a mystery. Some organizations struggle when trying to scale Docker, one of the leading container technologies.There are couple of Open Source container cluster management tool. Each cluster management technology has something unique and different to offer. Kubernetes and Docker Swarm are probably two most commonly used tools to deploy containers inside a cluster. Both are created as helper tools that can be used to manage a cluster of containers and treat all servers as a single unit.

When trying to make a choice between Docker Swarm and Kubernetes, think in following terms. Do you want to depend on Docker itself solving problems related to clustering. If you do, choose Swarm. If something is not supported by Docker it will be unlikely that it will be supported by Swarm since it relies on Docker API. On the other hand, if you want a tool that works around Docker limitations, Kubernetes might be the right one for you. Kubernetes was not built around Docker but is based on Google's experience with containers. It is opinionated and tries to do things in its own way

I. INTRODUCTION

When we get shared resources like Infrastructure, storage platform and even software via network irrespective of its source,then we term it as Cloud Computing. The cloud consists of different models namely.
* Storage as a Service (StaaS)
* Container as a Service (CaaS)
* Platform as a Service (PaaS)
* Software as a Service (SaaS)
* Infrastructure as a Service (IaaS)

Cloud Computing has been there for a long time but people now days are getting a hold of this. Cloud at times is also known as group of "Pay as you use" services.

ISSN: 2277-9655
Impact Factor: 4.116
CODEN: IJESS7

Storage as a Service: This module of cloud computing deals with the storage of enormous amount of data that is being generated every day. Each day, a huge pile of data is being uploaded to the servers. SAAS incorporates with the technologies

of how the data is efficiently stored in those servers, ensuring stability, reliability, robustness, backup and security. The data is also geographically replicated from servers from one geographical location to a different geographical location for Disaster Management Practice.

Platform as a service: This module of cloud computing incorporates with the idea which allows the developers to only focus on development of their projects and apps, rather than focusing on how to setup the coding and working environment about the language they are willing to work upon. Most of the programmers find it tedious to create and maintaining their own cloud environment as from being a developer mindset, it is a waste of time for them. They rather want to invest their time in coding and development, which is more productive for them and the organizations.

Container as a service: As Docker being light weight isolated environment, they can be used for multiple tasks. The tasks can range from running a webserver fleet from running a pseudo terminal for your favorite Linux environment. PAAS can also be given by using containers.

Infrastructure as a service: This service is the mother of all services in cloud computing. This service allows institutions or individuals to create their own virtual environment with no maintenance cost. This service incorporates with virtual servers, which can be up, and running in no time by anyone willing to.

This service provides storage resources like hard disks or ephemeral disks, computing resources like RAM, CPU processing power in a place, which may or may not be physically accessible by the user but is accessible for working through various protocols for working. This service ensures the portability of our computer, as it is not running at our local area. It is running in cloud and every instance in cloud has a public IP address and hence it can be accessed from any corner of the world. Virtual servers come with a lot of variety based on the ram size and data transfer rate. Some of the providers are Amazon Web Services, Rackspace, IBM Soft watch, Google Compute Engine, Microsoft Azure, Openstack etc.

Kubernetes: Kubernetes is based on Google's experience of many years working with Linux containers. It is, in a way, a replica of what Google has been doing for a long time but, this time, adapted to Docker. That approach is great in many ways, most important being that they used their experience from the start. If you started using Kubernetes around Docker version 1.0 (or earlier), the experience with Kubernetes was great. It solved many of the problems that Docker itself had. We could mount persistent volumes that would allow us to move containers without loosing data, it used flannel to create networking between containers, it has load balancer integrated, it uses etcd for service discovery, and so on. However, Kubernetes comes at a cost. It uses a different CLI, different API and different YAML definitions. In other words, you cannot use Docker CLI nor you can use Docker Compose to define containers. Everything needs to be done from scratch exclusively for Kubernetes. It's as if the tool was not written for Docker (which is partly true). Kubernetes brought clustering to a new level but at the expense of usability and steep learning curve.

Kubernete Architecture: *Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications*." Kubernetes was built by Google based on their experience running containers in production over the last decade. See below for a Kubernetes architecture diagram and the following explanation.

ISSN: 2277-9655
Impact Factor: 4.116
CODEN: IJESS7

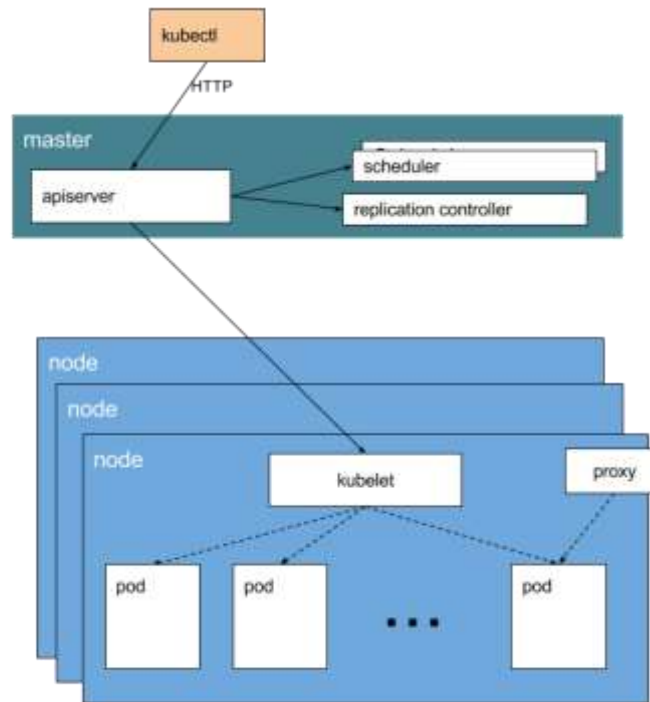

Fig 1.1 Kubernetes Architecture

The major components in a Kubernetes cluster are:

- **Pods** – Kubernetes deploys and schedules containers in groups called pods. A pod will typically include 1 to 5 containers that collaborate to provide a service
- **Flat Networking Space** – The default network model in Kubernetes is flat and permits all pods to talk to each other. Containers in the same pod share an IP and can communicate using ports on the localhost address.
- **Labels** – Labels are key-value pairs attached to objects and can be used to search and update multiple objects as a single set.
- **Services** – Services are endpoints that can be addressed by name and can be connected to pods using label selectors. The service will automatically round-robin requests between the pods. Kubernetes will set up a DNS server for the cluster that watches for new services and allows them to be addressed by name.
- **Replication Controllers** – Replication controllers are the way to instantiate pods in Kubernetes. They control and monitor the number of running pods for a service, improving fault tolerance

II. LITERATURE SURVEY
III.

The Research Tool: Kubernetes

Kubernetes is an open-source platform for automating deployment, scaling, and operations of application containers across clusters of hosts, providing container-centric infrastructure. It groups containers that make up an application into logical units for easy management and discovery. Kubernetes builds upon 15 years of experience of running production workloads at Google, combined with best-of-breed ideas and practices from the community.

With Kubernetes, you are able to quickly and efficiently respond to customer demand:

- Deploy your applications quickly and predictably.
- Scale your applications on the fly.
- Seamlessly roll out new features.
- Optimize use of your hardware by using only the resources you need.

ISSN: 2277-9655
Impact Factor: 4.116
CODEN: IJESS7

Our goal is to foster an ecosystem of components and tools that relieve the burden of running applications in public and private clouds.

Kubernetes is:

- **portable**: public, private, hybrid, multi-cloud
- **extensible**: modular, pluggable, hookable, composable
- **self-healing**: auto-placement, auto-restart, auto-replication, auto-scaling

The Kubernetes project was started by Google in 2014. Kubernetes builds upon a decade and a half of experience that Google has with running production workloads at scale, combined with best-of-breed ideas and practices from the community.

Kubernetes has three main quality that make it different from other container clustering tool:

- **Planet Scale:** Designed on the same principles that allows Google to run billions of containers a week, Kubernetes can scale without increasing your ops team.
- **Never Outgrow:** Whether testing locally or running a global enterprise, Kubernetes flexibility grows with you to deliver your applications consistently and easily no matter how complex your need is.
- **Run Anywhere:** Kubernetes is open source giving you the freedom to take advantage of on-premise, hybrid, or public cloud infrastructure, letting you effortlessly move workloads to where it matters to you.

Why do I need kubernete and what can it do:
At a minimum, Kubernetes can schedule and run application containers on clusters of physical or virtual machines. However, Kubernetes also allows developers to ‘cut the cord’ to physical and virtual machines, moving from a **host-centric** infrastructure to a **container-centric** infrastructure, which provides the full advantages and benefits inherent to containers. Kubernetes provides the infrastructure to build a truly **container-centric** development environment.

Kubernetes satisfies a number of common needs of applications running in production, such as:

- co-locating helper processes, facilitating composite applications and preserving the one-application-per-container model,
- mounting storage systems,
- distributing secrets,
- application health checking,
- replicating application instances,
- horizontal auto-scaling,
- naming and discovery,
- load balancing,
- rolling updates,
- resource monitoring,
- log access and ingestion,
- support for introspection and debugging, and
- identity and authorization.

This provides the simplicity of Platform as a Service (PaaS) with the flexibility of Infrastructure as a Service (IaaS), and facilitates portability across infrastructure providers.

Even though Kubernetes provides a lot of functionality, there are always new scenarios that would benefit from new features. Application-specific workflows can be streamlined to accelerate developer velocity. Ad hoc orchestration that is acceptable initially often requires robust automation at scale. This is why Kubernetes was also designed to serve as a platform for building an ecosystem of components and tools to make it easier to deploy, scale, and manage applications.

Labels empower users to organize their resources however they please. Annotations enable users to decorate resources with custom information to facilitate their workflows and provide an easy way for management tools to checkpoint state.

Additionally, the Kubernetes control plane is built upon the same APIs that are available to developers and users. Users can write their own controllers, schedulers, etc., if they choose, with their own APIs that can be targeted by a general-purpose command-line tool.
This design has enabled a number of other systems to build atop Kubernetes.

Key Features of Kubernetes:

Automatic binpacking
Automatically places containers based on their resource requirements and other constraints, while not sacrificing availability. Mix critical and best-effort workloads in order to drive up utilization and save even more resources.

Horizontal scaling
Scale your application up and down with a simple command, with a UI, or automatically based on CPU usage

Storage orchestration
Automatically mount the storage system of your choice, whether from local storage, a public cloud provider such as GCP or AWS, or a network storage system such as NFS, iSCSI, Gluster, Ceph, Cinder, or Flocker.

Self-healing
Restarts containers that fail, replaces and reschedules containers when nodes die, kills containers that don't respond to your user-defined health check, and doesn't advertise them to clients until they are ready to serve.

Service discovery and load balancing
No need to modify your application to use an unfamiliar service discovery mechanism. Kubernetes gives containers their own IP addresses and a single DNS name for a set of containers, and can load-balance across them

Secret and configuration management
Deploy and update secrets and application configuration without rebuilding your image and without exposing secrets in your stack configuration.

Batch execution
In addition to services, Kubernetes can manage your batch and CI workloads, replacing containers that fail, if desired.

Compare Kubernetes v/s Docker Swarm:

	Kubernetes	**Docker Swarm**
Types of Workloads	Cloud Native applications	Cloud native application
Application Definition	A combination of Pods, Replication Controllers, Replica Sets, Services and Deployments. As explained in the overview above, a pod is a group of co-located containers; the atomic unit of deployment. Pods do not express dependencies among individual containers within them.	Apps defined in Docker Compose can be deployed on a Swarm cluster The built-in scheduler has several filters such as node tags, affinity and strategies that will assign containers to the underlying nodes so as to maximize performance and resource utilization.

ISSN: 2277-9655
Impact Factor: 4.116
CODEN: IJESS7

	Kubernetes	Docker Swarm
	Containers in a single Pod are guaranteed to run on a single Kubernetes node.	
Application Scalability constructs	Each application tier is defined as a pod and can be scaled when managed by a Deployment or Replication Controller. The scaling can be manual or automated.	Possible to scale individual containers defined in the Compose file.
High Availability	Pods are distributed among Worker Nodes. Services also HA by detecting unhealthy pods and removing them.	Containers are distributed among Swarm Nodes. The Swarm manager is responsible for the entire cluster and manages the resources of multiple Docker hosts at scale. To ensure the Swarm manager is highly available, a single primary manager instance and multiple replica instances must be created. Requests issued on a replica are automatically proxied to the primary manager. If a primary manager fails, tools like Consul, ZooKeeper or etcd will pick a replica as the new manager.
Load Balancing	Pods are exposed through a Service, which can be a load balancer.	Load balancer is typically just another service defined in a Compose file. The swarm manager uses ingress load balancing to expose the services you want to make available externally to the swarm. Internally, the swarm lets you specify how to distribute service containers b/w nodes
Auto-scaling for the Application	Auto-scaling using a simple number-of-pods target is defined declaratively with the API exposed by Replication Controllers. CPU-utilization-per-pod target is available as of v1.1 in the Scale subresource. Other targets are on the roadmap.	Not directly available. For each service, you can declare the number of tasks you want to run. When you scale up or down, the swarm manager automatically adapts by adding or removing tasks to maintain the desired state.
Rolling Application Upgrades and Rollback	"Deployment" model supports strategies, but one similar to Mesos is planned for the future Health checks test for liveness i.e. is app responsive	At rollout time, you can apply service updates to nodes incrementally. The swarm manager lets you control the delay between service deployment to different sets of nodes. If anything goes wrong, you can roll-back a task to a previous version of the service.

Load Balancing In Kubernetes:

Kubernetes is excellent for running (web) applications in a clustered way. This blog will go into making applications deployed on Kubernetes available on an external, load balanced, IP address. Before diving into HTTP load balancers there are two Kubernetes concepts to understand: Pods and Replication Controllers.

A Replication Controller describes a deployment of a container. One of the aspects to configure on a Replication Controller is the number of replicas. Each replica will run on a different Kubernetes node. This way an application or application component can be replicated on the cluster, which offers failover and load balancing

ISSN: 2277-9655
Impact Factor: 4.116
CODEN: IJESS7

over multiple machines. Each instance of a replica is called a Pod. Kubernetes will monitor Pods and will try to keep the number of Pods equal to the configured number of replicas.

IV. PROPOSED METHODOLOGY

In this section, we define how to perform load balancing in kubernetes and the types of load balancing or we can say that, When we have multiple Pods running we still need a way to make the application available to the outside world, listening to a dns name like "http://myapp.io". Each Pod gets it's own virtual IP address (this works great combined with Weave), but these IP addresses aren't useful to the outside world.

Kubernetes has another concept, Services, that look useful in this context at first hand. A service is a proxy on top of Pods. This makes the service available on a more or less fixed IP address within Kubernetes, and it's clients don't know by which Pod a request is handled. Sounds like load balancing! Unfortunatly, the mechanism was designed to be used for components *within* the Kubernetes cluster (think Micro Services) to communicate. It is possible to assign your own IP address to a service as well, which makes it possible to point your DNS to it, but we still miss features like support for SSL offloading, HTTP cache settings etc. Services are a very useful concept, but simply not designed for external HTTP load balancing. We need something much better.

Types of load balancing

There are two different types of load balancing in Kubernetes. I'm going to label them internal and external.

Internal – aka "service" is load balancing across containers of the same type using a label. These services generally expose an internal cluster ip and port(s) that can be referenced internally as an environment variable to each pod.

Ex. 3 of the same application are running across multiple nodes in a cluster. A service can load balance between these containers with a single endpoint. Allowing for container failures and even node failures within the cluster while preserving accessibility of the application.

External –Services can also act as external load balancers if you wish through a NodePort or LoadBalancer type.

NodePort will expose a high level port externally on every node in the cluster. By default somewhere between 30000-32767. When scaling this up to 100 or more nodes, it becomes less than stellar. Its also not great because who hits an application over high level ports like this? So now you need another external load balancer to do the port translation for you. Not optimal.

LoadBalancer helps with this somewhat by creating an external load balancer for you if running Kubernetes in GCE, AWS or another supported cloud provider. The pods get exposed on a high range external port and the load balancer routes directly to the pods. This bypasses the concept of a service in Kubernetes, still requires high range ports to be exposed, allows for no segregation of duties, requires all nodes in the cluster to be externally routable (at minimum) and will end up causing real issues if you have more than X number of applications to expose where X is the range created for this task.

Because services were not the long-term answer for external routing, some contributors came out with **Ingress** and **Ingress Controllers**. This in my mind is the future of external load balancing in Kubernetes. It removes most, if not all, the issues with NodePort and Loadbalancer, is quite scalable and utilizes some technologies we already know and love like HAproxy, Nginx or Vulcan. So lets take a high level look at what this thing does.

HTTP load balancing

Google Container Engine offers integrated support for network load balancing with the type: LoadBalancerfield in your service configuration file.

If you require more advanced load balancing features, such as HTTPS balancing, content-based load balancing, or cross-region load balancing, you can integrate your Container Engine service with Compute Engine'sHTTP/HTTPS Load Balancing.

ISSN: 2277-9655
Impact Factor: 4.116
CODEN: IJESS7

Google Cloud Platform (GCP) HTTP(S) load balancing provides global load balancing for HTTP(S) requests destined for your instances. You can configure URL rules that route some URLs to one set of instances and route other URLs to other instances. Requests are always routed to the instance group that is closest to the user, provided that group has enough capacity and is appropriate for the request. If the closest group does not have enough capacity, the request is sent to the closest group that does have capacity.

HTTP requests can be load balanced based on port 80 or port 8080. HTTPS requests can be load balanced on port 443.

The load balancer acts as an HTTP/2 to HTTP/1.1 translation layer, which means that the web servers always see and respond to HTTP/1.1 requests, but that requests from the browser can be HTTP/1.0, HTTP/1.1, or HTTP/2.

HTTP(S) load balancing does not support WebSocket. You can use WebSocket traffic with Network load balancing.

Before you begin: HTTP(S) load balancing uses instance groups to organize instances. Make sure you are familiar with instance groups before you use load balancing.

Example configuration: If you want to jump right in and build a working load balancer for testing, the following guides demonstrate two different scenarios using the HTTP(S) load balancing service. These scenarios provide a practical context for HTTP(S) load balancing and demonstrate how you might set up load balancing for your specific needs.

How load balancer are constructed & how they work

Creating a cross-region load balancer

You can use a global IP address that can intelligently route users based on proximity. For example, if you set up instances in North America, Europe, and Asia, users around the world will be automatically sent to the backends closest to them, assuming those instances have enough capacity. If the closest instances do not have enough capacity, cross-region load balancing automatically forwards users to the next closest region.

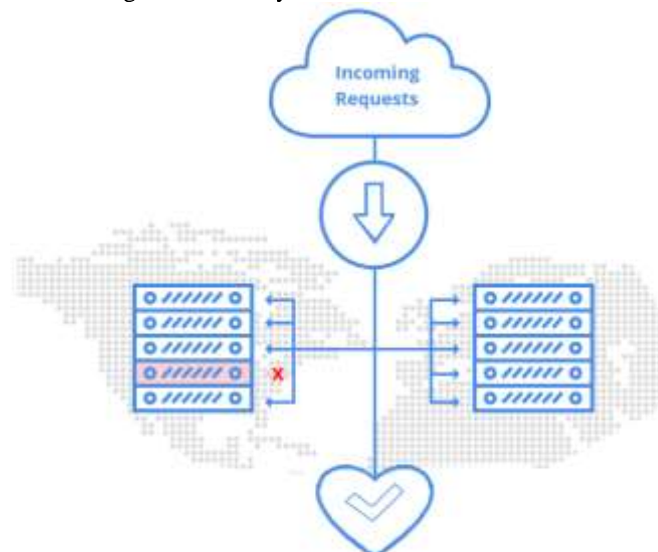

Fig 3.1 Cross region load balancer

Creating content-based load balancer

Content-based or content-aware load balancing uses HTTP(S) load balancing to distribute traffic to different instances based on the incoming HTTP(S) URL. For example, you can set up some instances to handle your video content and another set to handle everything else. You can configure your load balancer to direct traffic for example.com/video to the video servers and example.com/ to the default servers.

ISSN: 2277-9655
Impact Factor: 4.116
CODEN: IJESS7

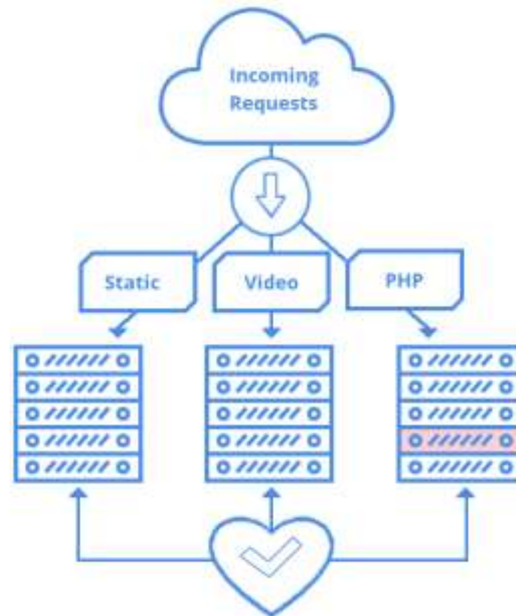

Fig 3.2 Content based load balancer

Content-based and cross-region load-balancing can work together by using multiple backend services and multiple regions. You can build on top of the scenarios above to configure your own load balancing configuration that meets your needs.

Overview

An HTTP(S) load balancer is composed of several components. The following diagram illustrates the architecture of a complete HTTP(S) load balancer:

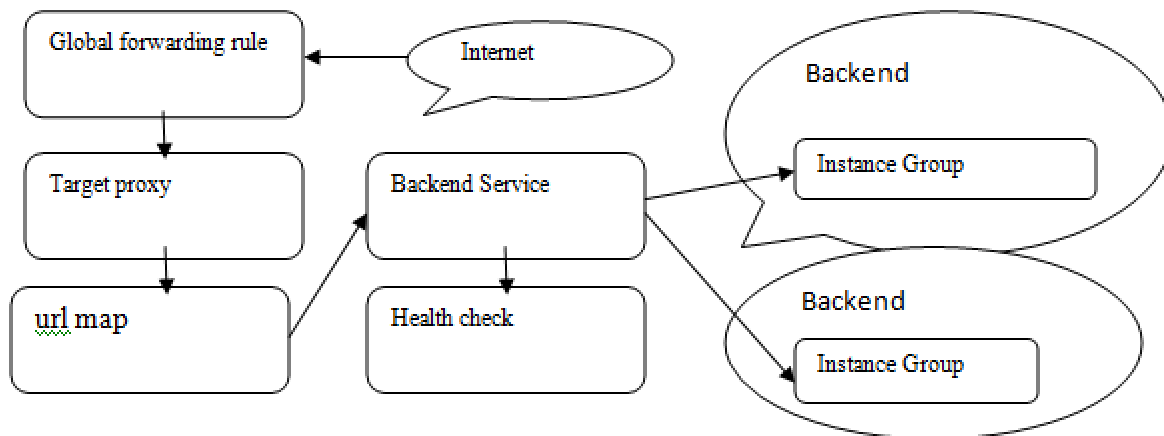

Fig 3.3 HTTP load balancer

The following sections describe how each component works together to make up each type of load balancer. For a detailed description of each component, see Components below.

HTTP load balancing

A complete HTTP load balancer is structured as follows:

1. A global forwarding rule directs incoming requests to a target HTTP proxy.
2. The target HTTP proxy checks each request against a URL map to determine the appropriate backend service for the request.
3. The backend service directs each request to an appropriate backend based on serving capacity, zone, and instance health of its attached backends. The health of each backend instance is verified using

ISSN: 2277-9655
Impact Factor: 4.116
CODEN: IJESS7

either an HTTP health check or an HTTPS health check. If the backend service is configured to use the latter, the request will be encrypted on its way to the backend instance.

In addition, you must create a firewall rule that allows traffic from 130.211.0.0/22 to reach your instances. The rule should enable traffic on the port your global forwarding rule has been configured to use (either 80 or 8080).
HTTPS load balancing
An HTTPS load balancer shares the same basic structure as an HTTP load balancer (described above), but differs in the following ways:

- Uses a target HTTPS proxy instead of a target HTTP proxy
- Requires a signed SSL certificate for the load balancer
- Requires a firewall rule that enables traffic from 130.211.0.0/22 on port 443 to reach your instances
- The client SSL session terminates at the load balancer. Sessions between the load balancer and the instance can either be HTTPS (recommended) or HTTP. If HTTPS, each instance must have a certificate.

Nginx & Ingress
Nginx is a web server, which can also be used as a reverse proxy, load balancer and HTTP cache. Nginx can be deployed to serve dynamic HTTP content on the network using FastCGI, SCGI handlers for scripts, WSGI application servers or Phusion Passenger modules, and it can serve as a software load balancer.

Step 1: Create an nginx server
Create a pod with a single nginx server:
```
$ kubectl run nginx --image=nginx --port=80
```

This runs an instance of the nginx image serving on port 80.

Step 2: Expose nginx as a service
Next, create a Container Engine service which exposes this nginx Pod on each node in your cluster:
```
$ kubectl expose deployment nginx --target-port=80 --type=NodePort
```

Step 3: Create an ingress object
Ingress is an extension to the Kubernetes API that encapsulates a collection of rules for routing external traffic to Kubernetes endpoints. Ingresses are currently a Kubernetes beta feature, but they are supported on Google Container Engine via a kube-system addon that creates and manages HTTP Load Balancers.
The following config file defines an ingress that will direct traffic to your nginx server:
```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: basic-ingress
spec:
  backend:
    serviceName: nginx
    servicePort: 80
```

This basic ingress defines a default backend that directs all traffic to the nginx service on port 80. You can also define rules that direct traffic by host/path to muliple Kubernetes services:

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: sample-ingress
spec:
  backend:
    serviceName: default-handler
    servicePort: 80
```

```
rules:
- host: my.app.com
  http:
    paths:
    - path: /foo
      backend:
        serviceName: foo-service
        servicePort: 8080
    - path: /bar
      backend:
        serviceName: bar-service
        servicePort: 80
```

Note, when using ingresses, verify that you have sufficient resource quota in the project. Ingresses require oneGoogle Compute Engine Backend Service for each Kubernetes service..

V. REFERENCES

[1] Con Wang, Qian Wang, Kui Ren Wenjing, "Ensuring Data Storage Security in Cloud Computing",Quality of Service,2009.IWQoS. 17th International Workshop,DOI: 10.1109/IWQoS.2009.5201385
[2] P.G. Dorey, A. Leite, "Commentary: Cloud computing A security problem or solution?",Journal Information Security tech.Report[archive],August 2011
[3] Rohit Bhadauria , Sugata Sanyal,"Survey on Security Issues in Cloud Computing and Associated Mitigation Techniques",International Journal of Computer Applications,
[4] Rohit Bhadauria,Rituparna,Chaki,Nabendu Chaki,Sugata Sanyal,"A Survey on Security Issues in Cloud Computing ",IEEE Communications Surveys and
[5] Tutorials, 1-15.2011
[6] S. Subashini , V. Kavitha,"A survey on security issues in service delivery models of cloud computing"Journal of Network and Computer Applications[archive],January 2011
[7] Mohamed Al Morsy, John Grundy, Ingo MÃ¼ller,"An Analysis of The Cloud Computing Security Problem.",2010 Asia Pacific Cloud Workshop,January 2010
[8] Wang Shao-hui,Chang Su-qin , Chen Dan-wei , Wang Zhi-wei ,"Public Auditing for Ensuring Cloud Data Storage Security With Zero Knowledge Privacy",International Journal of Computer Trends and Technology,2015
[9] Keiko Hashizume, David G Rosado Eduardo,Fernandez- Medina and Eduardo B Fernandez,"An analysis of security issues for cloud computing",Journal of Internet
[10] http://www.infoworld.com/article/3118345/cloud-computing/why-kubernetes-is-winning-the-container-war.html
[11] http://www.bigswitch.com/blog/2016/06/20/accelerating-network-transformation-with-big-cloud-fabric-36-and-big-monitoring
[12] Divya Muntimadugu, Anjana Suparna Sriram, "Red Hat Storage 2.0 Administration Guide".
[13] Arun gupta "Clustering Using Docker Swarm 0.2.0", April 23,2015,http://blog.arungupta.me/clusteringdocker- swarm-techtip85
[14] Dr. Yong Yu, Prof. Atsuko Miyaji, Dr. Man Ho Au, Prof. Willy Susilo, "Special Issue on Cloud Computing Security and Privacy: Standards and Regulations"
[15] "NGINX vs. Apache (Pro/Con Review, Uses, & Hosting for Each) - HostingAdvice.com". HostingAdvice.com. 5 April 2016. Retrieved 28 December 201
[16] http://www.devoperandi.com/load-balancing-in-kubernetes/

CITE AN ARTICLE

Gupta, T., & Dwivedi, A. (2017). DATA STORAGE & LOAD BALANCING IN CLOUD COMPUTING USING CONTAINER CLUSTERING. *INTERNATIONAL JOURNAL OF ENGINEERING SCIENCES & RESEARCH TECHNOLOGY, 6*(9), 656-666.