International Journal of Recent Technology and Engineering (IJRTE)
ISSN: 2277-3878 (Online), Volume-9 Issue-2, July 2020

Load Balancing using Docker and Kubernetes: A Comparative Study

Prajval Mohan, Tejas Jambhale, Lakshya Sharma, Simran Koul, Simriti Koul

Abstract*: Still in its early years, containers are increasingly being used in production environments. Containers offer a streamlined approach, easy deployment, and secure method of implementing infrastructure requirements also provide a much-improved alternative to virtual machines. A load balancer is required to distribute traffic across clusters. And now, with multiple container environments becoming widespread, load balancers are becoming a necessity to distribute traffic and reduce server load. Different load balancing algorithms provide a solution to this with varying efficiency. This paper presents a study on the latest methods which are being implemented to perform effective load balancing on containers. Docker Swarm and Kubernetes are the most widely used systems for deploying and managing a cluster of containers in an environment. The paper further demonstrates how Docker Swarm and Kubernetes can be used to minimize load traffic through load balancing techniques. We have introduced load balancing and different algorithms. Also, we have shown the implementations of load balancing algorithms in Docker and Kubernetes and finally compared the results. The paper finally concludes why Kubernetes is often preferred over Docker Swarm for load balancing.

Keywords: Docker, Docker Swarm, Kubernetes, Ingress, Load balancing, NodePort, LoadBalancer, Nginx.

I. INTRODUCTION

Containers have changed the way we develop, distribute, and run the software. Developers can freely develop software, containerize them and distribute them as they wish – be it a co-worker, an entire department, or some random person on a network. Before running a container, the user will know exactly how it will proceed- the process of running a container is simple, formulaic, immutable and repeatable. Developers can spend more time developing the code rather than waste time setting up the environment. Earlier, the process of application deployment was manual, time-consuming and utilized a lot of company resources. With the emergence of containerization tools like Docker and Kubernetes, there has been a significant improvement in the deployment, management and scaling of the software applications. The entire process has now become more efficient, faster and systematized. Docker is an open-source tool developed to make the process of creation, deployment and running of applications in different platforms much more accessible. Docker uses standard containers to package an application along with all its dependencies and modules. These containers are gaining a massive stronghold in the market as it is simplifying and improving the efficiency of the entire development process. More often than not, micro-services and cloud web services require us to run multiple containers across numerous machines, but Docker containers do not provide a scalable solution. Kubernetes, which is developed and maintained by Google, is the main compartment coordination motor in the area of containerization. It makes use of pictures made by Docker. Compared with other virtual machines, Kubernetes provides more accessible, more efficient and faster to convey administrations and applications. We have to start the containers at the appropriate time, find out how the containers communicate with each other and store them appropriately. Kubernetes solves this problem efficiently. At the first look, Docker and Kubernetes may seem like similar technologies aiming to containerize applications, but on closer inspection, it becomes pellucid that both these tools function at different layers in an operating system stack and that they can be used together as well. In this growing era of cloud computing, modern cloud architectures often demand a specific understanding of Docker and Kubernetes applications. Some of the main reasons why Kubernetes is widely used rather than the Docker cluster are scalability, portability and self-healing. A website may be accessed by more than a thousand users at a time. Managing this load becomes an arduous task and puts immense pressure on the host server. This sometimes also results in a system crash. Load balancing is a process used in cloud computing to manage the traffic load by distributing resources and workload units among different servers, hard drives, etc., which results in more utilization and improved system response time. Containers are changing the way we develop and ship code. They have become an integral part of the development process. With the increasingly widespread use of these containers, multiple container environments have become common.

Manuscript received on May 25, 2020.
Revised Manuscript received on June 29, 2020.
Manuscript published on July 30, 2020.
***** Correspondence Author
Prajval Mohan*, School of Computer Science and Engineering, Vellore Institute of Technology, Vellore, Tamil Nadu, India.
Email: prajval.mohan23@gmail.com
Tejas Jambhale, School of Computer Science and Engineering, Vellore Institute of Technology, Vellore, Tamil Nadu, India.
Email: tejas.jambhale98@gmail.com
Lakshya Sharma, School of Computer **Science** and Engineering Vellore Institute of Technology, Vellore, Tamil Nadu, India.
Email: lakshya99sh@gmail.com
Simran Koul, School of Computer Science and **Engineering**, Vellore Institute of Technology, Vellore, Tamil Nadu, India.
Email: simran.koul@yahoo.com
Simriti Koul, School of Computer Science and Engineering, Vellore Institute of Technology, Vellore, Tamil Nadu, India.
Email: simriti.koul@yahoo.com

This leads to significant constraints on servers. Therefore, load balancing these containers can help improve not only the utilization of server resources but also enhance the efficiency of the entire development process.

The main components of a Kubernetes cluster include Pods, Flat Networking Space, Labels, Replication Controllers and Services.

Load Balancing in Docker

The extended Docker API provides effective methods to create and scale administrations, health checks, load balancing, traffic distribution, etc. Administrations are a collection of holders which are quite similar to Docker compose, however, with a lot more highlights.

Docker Swarm Load Balancing Topology

Docker Swarm is a Layer 4 TCP load balancer. For this paper, we have created three Swarm hubs, which include two specialist hubs and one ace hub. The swarm directions are being run in the ace hub. The Swarm takes care of load adjusting and distribution, scaling and DNS administration booking and revelation.

The Docker Swarm LB executes on all the hubs and can process equalization requests over either of the holder/hosts in the hub. In the absence of NGINX or NGINX Plus, the Docker Swarm Lb. takes care of the incoming customer demands in the swarm organization.

Advanced Load Balancing Using NGINX Plus

A scope of cutting edge includes in NGINX Plus make it a perfect load balancer before ranches of upstream servers:

- Load adjusting and session determination – Better burden adjusting crosswise over specialist procedures and session industriousness techniques to distinguish and respect application sessions
- HTTP wellbeing checking and server moderate beginning – Asynchronous' engineered exchanges to test the right activity of each upstream server, and agile 'moderate beginning' reintroduction of servers when they recuperate
- Live movement observing – Immediate report of action and execution
- Dynamically designed upstream server gatherings – Tool to encourage some regular upstream administration undertakings, for example, the protected and brief evacuation of a server

II. RELATED WORK

Docker and Kubernetes are open-source tools developed to make the process of creation, deployment and running of applications in different platforms much easier. They are increasingly being utilized for microservices and cloud-based services. They find their application in almost all the domains of information technology. One of the major purposes of using docker swarm and Kubernetes is utilizing their load balancing capabilities. Most of the research work in this domain has been focused on exploring and analyzing various scheduling strategies for container management and improving docker security. Some papers also talk about the different load balancing techniques and how they can be utilized for cloud services. Many recent research papers introduce improved methods for dynamic load balancing in cloud platforms. Most of the industries today are unaware of all the different load balancing functionalities provided by docker swarm and Kubernetes and the most efficient use cases for these different techniques. Our research paper provides a thorough analysis of different dynamic load balancing techniques afforded by Docker and Kubernetes. We also discuss the ideal scenarios for using each of these techniques and compare them with other methods

In-State machine replication in containers managed by Kubernetes [1], they have proposed the integration of coordination services Kubernetes (k8s), seeking to control the containers' size and to allow automatic state replication. For this purpose, they have presented a new protocol named DORADO (Dering Over Shared Memory) for the integration of coordination services in Kubernetes and to perform state machine replication in the containers. In the first three sections, they have explained the concepts about containers and Kubernetes, coordination and state replication protocol in Kubernetes. The fourth section of this research paper talks about the evaluation of DORADO on a number of preliminary tests. They have mentioned in detail about the execution environment and the experiments that they conducted to evaluate their protocol. Finally, they conclude the research paper by mentioning the challenges and future scope of their protocol. In a decentralized system for load balancing of containerized microservices in the Cloud [2], they have proposed a decentralized orchestration system for load balancing of containerized microservices and web services. They explained the internal working of virtualization containers and analyzed it, mentioning the shortcomings and limitations of containers for load balancing of microservices. They explain how a decentralized system for this purpose can yield increased throughputs, lower response time and better scalability of the services. Further, they introduce their swarm-like algorithm for container migration. This research paper also includes some preliminary experimental results of their proposed algorithm for decentralized systems. Finally, they conclude the paper with a brief summary and remarks.

Load Balancing and its Algorithms in Cloud Computing: A Survey [3] discusses the concept of load balancing and its ever-increasing importance in this era of cloud computing. They have prepared a literature survey on the different load balancing techniques available considering the following measurement parameters: fairness, throughput, fault tolerance, overhead, performance, and response time and resource utilization. They have analyzed two different categories of load balancing techniques (1) Static algorithms, which include Load Balancing Min-Min Algorithm, Load Balancing Min-Max Algorithm and Round Robin Load Balancing Algorithm. (2)

Dynamic algorithms, which include Throttled Load Balancing Algorithm, ESCE (Equally Spread Current Execution) Load Balancing Algorithm, Ant Colony Load Balancing Algorithm, Biased Random Sampling Load Balancing Algorithm, Modified Throttled Load Balancing Algorithm and Honeybee Foraging Behavior Load Balancing Algorithm. Finally, they conclude the paper with a brief summary and remarks. Using Docker Containers to Improve Reproducibility in Software and Web Engineering Research [4] discusses the importance and applications of Docker containers in this age of virtualization and how they can be utilized to aid the reproducibility of research artifacts in software and web engineering.

This research paper also throws some light on the challenges in the current web engineering researches and how the Docker containers can provide a promising solution to the same. They have provided a comprehensive tutorial on Docker containers, which also includes a discussion on the advantages, limitations and challenges of the containers. The tutorial also covers the Docker container basics, which walk through the basic commands, running Docker images, building custom images, deployment and production of web apps. They conclude the paper with a brief summary and remarks. In A New Docker Swarm Scheduling Strategy [5], they have presented a new economical scheduling strategy implementation for Docker swarm. Their strategy's novelty lies in the use of user's SLA classes (Long service, Short service and microservice) to schedule the containers and the dynamic allocation of CPU cores to execute the selected container. This model is loosely based on the observation that hosting solutions do not allow manufacturers or cloud providers to offer to their customers a fair or accurate invoice. They have explained their scheduling algorithm in great detail along with the code and compared it with the existing scheduling strategy in Docker swarm. Finally, they have included the results of the test conducted on their strategy by emulation and demonstrated the scope of their approach for further development. A Scheduling Strategy on Load Balancing of Virtual Machine Resources in Cloud Computing Environment [6] introduces a new scheduling strategy on load balancing of VM resources based on a genetic algorithm. This strategy computes beforehand the influence it will have on the system after the deployment of the needed VM resources and then selects the least effective solution, through which it achieves the best load balancing and reduces or avoids dynamic migration [6]. In the end, an analysis of the method is made and an experiment and summary are also conducted.

Research on Kubernetes' Resource Scheduling Scheme [7] presented a better scheduling algorithm than the original algorithm used in Kubernetes for resource scheduling. The proposed algorithm is a combination of an improved ant colony algorithm (ACA) and an adaptive particle swarm optimization algorithm (PSO). In the later sections of the paper, they have demonstrated each of the algorithms individually in great detail. Further, they presented their improved versions of ACA and PSO, followed by the combination of these algorithms. In the final section of the research paper, they have included the experimental results, which show that the proposed algorithm is suggestively better than the original Kube-scheduler model, which can effectively reduce the resource consumption cost and reduce the maximum load of the node. Finally, they have mentioned about the future scope of improvements in their algorithm to obtain even better use effect and lower usage cost. The need for virtualization has increased remarkably over the last few years. Container-based virtualization, like Docker, is one of the leading products used in this field. In the Analysis of Docker security [8], they have analyzed the security features of Docker containers. They have focused on two areas: (1) the internal security of Docker and (2) how does Docker works with the security features of the Linux kernel, such as SELinux and AppArmor, in order to secure the host system. Section 2 of this paper gives an introduction about the two types of main types of virtualization technology solutions in the market, i.e., Container-based virtualization and hypervisor-based virtualization. Section 3 discusses Docker and its underlying technologies. In section 4, they have presented their analysis of Docker security, and then finally, in Section 5, they discuss the security level of Docker and what could be done to increase its level of security. The paper winds up with a summary and brief remarks in Section 6.

Genomic pipelines include various pieces of third-party software and are prone to frequent changes and updates, which lead to a number of deployment and reproducibility issues. Docker containers are one of the most promising solutions for many of these problems as they allow the packaging of pipelines in a self-contained manner. But this might compromise with the performance of these pipelines. The impact of Docker containers on the performance of genomic pipelines [9], they have analyzed the effect of Docker containers on the genomic pipelines. In order to measure the impact of containers on the execution performance of bioinformatics tools, they have benchmarked three different genomic pipelines. The results show that Docker containerization has a negligible impact on the execution performance of common genomic pipelines, where tasks are generally very time-consuming. The paper winds up with a summary and brief remarks. In Slacker: Fast distribution with lazy Docker containers [10], they have introduced a new container benchmark, HelloBench, to measure the startup times of 57 different containerized applications. They have utilized HelloBench to analyses workloads in detail, studying the block I/O patterns produced during startup and the compressibility of container images. Their study shows that pulling packages accounts for 76% of the container start time, but only 6.4% of that data is read. They have used this and other results to design Slacker, a new Docker storage driver optimized for fast container startup. Slacker is based on centralized storage that is shared between all Docker workers and registries. In the later sections of the paper, they have demonstrated various benchmark tests used to evaluate Slacker's performance and also included the results of these tests, which revealed that Slacker speeds up the median container development cycle by 20 times and deployment cycle by five times. They have ended the paper with a summary and acknowledgments.

In Applying computational intelligence for enhancing the dependability of multi-cloud systems using Docker swarm [11], they have introduced a Computer Intelligence based solution for enhancing the dependability of a multi-cloud system using Docker swarm. At the present Docker swarm makes use of RAFT consensus algorithm which has few major design problems. The proposed method in this paper, which is based on a fuzzy interference system, provides a promising solution to this. Section 2 of this paper discusses Docker Containers, Docker Swarm, Docker Hub, fuzzy logic, and coding in R software. Section 3 explains the architecture and theoretical background of a multi-cloud system using Docker Swarm. Section 4 of this paper includes a simulation of a multi-cloud system using Docker Swarm.

Section 5 presents an experimental evaluation of dependability. Section 6 talks about the proposed Computational Intelligence based strategy for improving the dependability of a multi-cloud system using Docker Swarm. Lastly, the paper winds up with a brief summary and the future scope of improvement.

MPI is a widely used technology used in the field of the high-performance computing environment. However, setting up an MPI cluster can be challenging and time-consuming. Distributed MPI cluster with Docker swarm mode [12] provides a solution for this issue by using modern containerization technology like Docker and Docker Swarm to automate the MPI cluster setup and deployment. In Section 2 they have discussed the background technologies used in the paper. Section 3 involves the project overview that includes software specifications. In Section 4 and 5, they have discussed the reason for system design and showed how to use the proposed solution to develop the MPI program and to deploy a fully connected MPI cluster as Docker containers operating in Docker Swarm mode that runs on multiple machines. Lastly, the paper winds up with a brief summary and the future scope of improvement.

In the Evaluation of Docker as an edge computing platform [13], they have assessed Docker as a platform for Edge Computing. They evaluated Docker based on four parameters: deployment and termination, resource & service management, fault tolerance, and caching. Based on the results of their evaluation and experiment, it showed that Docker provides rapid and efficient deployment, low overhead and good performance, which makes it one of the best technologies for edge computing platform.

Load Balancing in cloud computing [14] introduces the vast field of cloud computing and given a brief introduction on the load balancing implementation in cloud platforms. They discuss the concept of load balancing, its needs and goals, types and comparison between traditional computing environment and cloud computing environment. A list of policies for implementation is given to help in the analysis process. They list out the advantages and disadvantages of various algorithms and give metrics for them. They have concluded by mentioning the current status of load balancing in cloud computing and future prospects. THE efficient VM load balancing algorithm for a cloud computing environment [15] introduces an efficient mechanism to implement load balancing in cloud environments. They concentrate on cloud computing as Iaas. They talk about how load balancing is important to utilize full resources of parallel and distributed systems. The first modeling of VM allocation is done to provide and configure hardware resources. They call it VM policy allocation and VM scheduling. Here they use CloudSim to provision this. They go on mentioning different algorithms and then introduce their own algorithms Weighted Active Monitoring Load Balancer. They have used this novel algorithm to get results and compared them with traditional algorithms. The VM assigns a varying amount of the available processing power to the individual application services. They have optimized different parameters and showed why this new method outperforms others.

In the Dynamic load balancing strategy for grid computing [16], they have introduced a load balancing technique specifically for grid computing. They mention how traditional algorithms, though, are widely used and efficient enough. They are not suitable for grid computing environments as they must address main new issues, namely: heterogeneity, scalability and adaptability. They propose a layered algorithm that achieves dynamic load balancing in grid computing, which is totally independent of the architecture of the grid. They compare static and dynamic algorithms and show how suitable dynamic algorithms is suitable for the current scenario. The proposed algorithms follow a tree-based balancing model specific for grid computing. The generic model is a non-cyclic connected graph with four levels. They list out three policies for the new proposed algorithm, and the results of the new algorithms are measured. They conclude by saying that the proposed algorithms perform well enough for the application of grid computing. But this new model raises new challenges for future researchers and admits that though results are good for particular environments, it needs testing in different grid environments. In Data storage and load balancing in cloud computing using container clustering [17], they talk about the importance of containers in real-world applications and how their performance can be improved using load balancing techniques. They aim to compare Docker swarm and Kubernetes load balancing techniques and show how Kubernetes can be used to overcome Docker limitations. They look to explain Kubernetes and how it can be implemented for load balancing. They explain theoretically without giving practical proof of various Kubernetes techniques and compare results. They write on how Kubernetes has the capability of improving load balancing over Docker swarm. In A dynamic load balancing strategy for cloud computing platforms based on exponential smoothing forecast [18], they write about the importance of why load balancing is essential. They say as cloud computing increases, the need for load balancing techniques will increase. They introduce a method to calculate the current load. They propose an exponential smoothing forecast method, which is a type of dynamic balancing method. They choose a physical server for deployment for Paas and server clusters for Iaas. They use graphical analysis to show how their algorithm performs over time.

Exponential Smoothing Forecast-Based on Weighted Least-Connection (ESBWLC) optimizes the number of connections to actual load service capability and shows real-life applications. In the improvement of container scheduling for Docker using ant colony optimization [19], they propose a method to improve Docker performability by introducing a new algorithm called ant colony optimization. They look to bring a new algorithm to the Docker swarm kit scheduler to improve performance. The main contribution is an ACO-based algorithm that distributes application containers over Docker hosts to offer better balance in resource usages and leads to the performance improvements of applications as compared to the current greedy scheduler.

They say that their proposed algorithms perform better than the current greedy approach by 15%. They have initially discussed the limitations of current schedulers and then shown how their algorithm overcomes them. They explained the architecture behind the proposed algorithm and proposed a formula to compute resource utilization at each node. They show how their algorithm performs better with an improvement of nearly 15%.

In research and implementation of Docker performance service in distributed platform [20], they write about the current status of Docker. They write about how Docker cluster environment can be improved. They mention Docker module design and architecture behind it. They conduct experimental tests for cluster building and cluster performance monitoring. They create a log file to maintain their results and make different performance tests. They look for improvement in Docker deployment.

In An introduction to Docker and analysis of its performance [21], they talk about the impact Docker has had on the market. They review the technology and analyses its performance. They analyze Docker client and servers, images and registries. They undertake a comprehensive comparison of Docker and KVM and explain why Docker is the future. They then use different performance parameters for reviewing like speed, portability, scalability, rapid delivery, density. They talk about their disadvantages and their competitor. They compare boot time, CPU calculation, time to compute 1 and 1000 SQL queries. Finally, they compare with VM and mention how Docker is the future.

In Resilience enhancement of container-based cloud load balancing service [22], they talk about web traffic is unpredictable and sometimes lead to high load on servers. Load balancers play an important role in reducing this and mitigating the effect of high web traffic. They chose Nginx to show the effect load balancers have and look to make servers and containers services more resilient to handle server load. NGINX plus allows the user to configure dynamic weight-based on certain metrics. They propose a flexible, pluggable, cloud-agnostic, and metric-agnostic dynamic algorithm for any cloud load balancer services. They model the resources in a server as a multidimensional vector, based on which they convert the relative resource availability of all backend servers to the weights assigned to them dynamically. They use an agent-based modeling service as its architecture that can collect various types of metric data of the backend servers. With CPU load-based load balancer policy, the average RTT is 0.0072 seconds. Without any policy, the average RTT is 0.0658 seconds. They conclude by saying their algorithm can easily be integrated with Docker, Kubernetes and AWS.

In the Value-Based Allocation of Docker Containers [23], the main objective was to figure out the main objective of docker containers. A rapid increase in the number of public cloud vendors has led to the addition of containers as a Service (CaaS) to their portfolio. This is the reason, the popularity of Docker, a software that allows Linux containers to run independently on the host of an isolated environment. Depending on the software, the orchestration and allocation approaches must vary. The key objective of this paper was to see how this execution varies with time. Here, two dynamic allocation algorithms were deployed and compared with the default docker algorithm. The efficiencies of these algorithms are based on the weight of the workload and scales with the growing number of nodes in the Cloud.

According to a Portable Load Balancer for Kubernetes Cluster [24], Linux containers have gained popularity due to their lightweight and portable nature. Nowadays, many web services are being deployed as clusters of containers. Here, in this paper, the authors have concentrated on Kubernetes Clusters. But Kubernetes relies on load balancing supplied by cloud providers. The authors proposed a portable load balancer that was usable in any environment, and hence facilitated web services migration. This was implemented using the Linux kernel's Internet Protocol Virtual Server (IPVS). The product resulted in an improved portable web service without compromising performance.

Distributed computing gives clients close to moment access to apparently boundless assets, and gives specialist organizations the chance to send complex data innovation framework, as an administration, to their clients. Suppliers' profit by economies of scale and multiplexing increases managed by sharing of assets through virtualization of the basic physical foundation. In any case, the scale and exceptionally dynamic nature of cloud stages force huge new difficulties to cloud specialist co-ops. Specifically, acknowledging refined cloud administrations requires a cloud control structure that can coordinate cloud asset provisioning, design, use and decommissioning over an appropriated set of physical assets. In Cloud Resource Orchestration: A Data-Centric Approach [25], they advocate an information-driven way to deal with the cloud organization. Following this methodology, cloud assets are demonstrated as organized information that can be questioned by an explanatory language and refreshed with well-characterized value-based semantics. They look at the possibility, advantages and difficulties of the methodology, furthermore, present our plan and model execution of the Information-Driven Management Framework (DMF) as an answer, with information models, question dialects and semantics that are explicitly intended for cloud asset arrangement. In Cloud Computing Networking: Challenges and Opportunities for Innovations [26], distributed computing appears the vision of utility figuring.

Inhabitants can profit by on-request provisioning of processing, stockpiling, and organizing assets as indicated by compensation for each utilization plan of action. Inhabitants have just constrained permeability and power over system assets. The proprietors of distributed computing offices are likewise confronting difficulties in different parts of giving what's more, productively overseeing IaaS offices. In this work, they present the systems administration issues in IaaS. What's more, league difficulties that are as of now tended to with existing innovations. They moreover present creative programming characterized organizing proposition, which is connected to a portion of the challenges and could be utilized in future organizations as productive arrangements.

Distributed cloud computing has conveyed uncommon processability to NASA missions at moderate rates. Missions like the Mars Investigation Rovers (MER) and Mars Science Lab (MSL) are getting a charge out of the versatility that empowers them to use hundreds, if not thousands, or machines for brief spans without making any equipment obtainments. In Polyphony: A Workflow Orchestration Framework for Cloud Computing [27], they depict Polyphony, a flexible, adaptable, and measured structure that proficiently uses an enormous arrangement of processing assets to perform parallel calculations. Polyphony can utilize assets on the Cloud, overabundance limit on nearby machines, just as extra assets on the supercomputing focus, and it empowers these assets to work in show to achieve a shared objective. Polyphony is flexible to hub disappointments, regardless of whether they happen in an exchange. They will close with an assessment of a generation prepared application manufactured over Polyphony to perform picture handling activities of pictures from around the nearby planetary group, including Mars, Saturn, and Titan. SDN orchestration architectures and their integration with Cloud Computing application [28] explain that developing cloud-based applications, running in geologically disseminated Data Centers (DCs), produces new unique traffic designs that guarantee for an increasingly effective administration of the traffic streams. Topographically appropriated DCs interconnection requires programmed and progressively unique provisioning and cancellation of end to end (E2E) network administrations, through heterogeneous system areas. Each system space may utilize various information transport innovation yet, in addition, an alternate control/the board framework. The quick advancement of Software Defined Networking (SDN) and the interworking with current control plane innovations, for example, Generalized Multi-convention Label Switching (GMPLS), request coordination over the heterogeneous control examples to give consistent E2E network administrations to outer applications

In an in-depth analysis and study of Load balancing techniques in the cloud computing environment [29], distributed cloud computing worldview, load adjusting is one of the difficulties, With Tremendous increment in the clients and their request of various administrations on the distributed computing stage, productive or proficient use of assets in the cloud condition turned into a basic concern. Burden adjusting is assuming a crucial job in keeping up the beat of Cloud registering. The exhibition measurements of burden adjusting calculations in the Cloud are reaction time and holding uptime. In this paper, they fundamentally center around two burden adjusting calculations in cloud, Min-Min and Max-Min algorithm.

In Dynamic Balance Strategy of High Concurrent Web Cluster Based on Docker Container [30], they propose improvements to the existing Round-Robin and Weighted Round-Robin algorithms. They look to make load balancing a dynamic technique depending on the traffic. They first compare Docker and Kubernetes architecture and then explain the drawbacks of the existing NIGEX strategy of load balancing. It does not consider the dynamic change of server performance during t system running processes and the number of backend servers can't be adjusted according to the requested amount. [31] The strategy defines performance quotas of the quantized Pod service and the weight of relative performance quotas. They then calculate real-time performance weight ratios of the cluster by the weight sum algorithm.

III. DESIGN AND IMPLEMENTATION OF LOAD BALANCING

A. Load Balancing using Docker

1. Architecture and Design

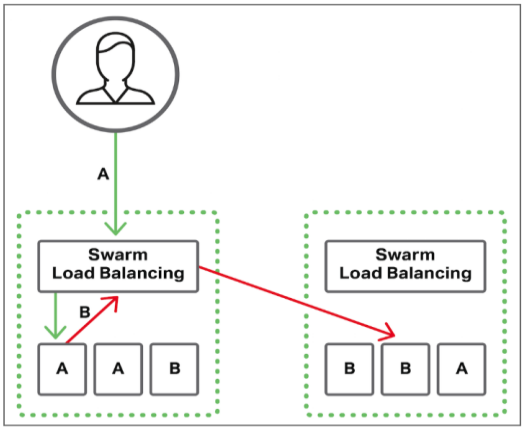

Figure 1. Load balancing of the client and service-to-service requests in a Swarm cluster without NGINX or NGINX Plus

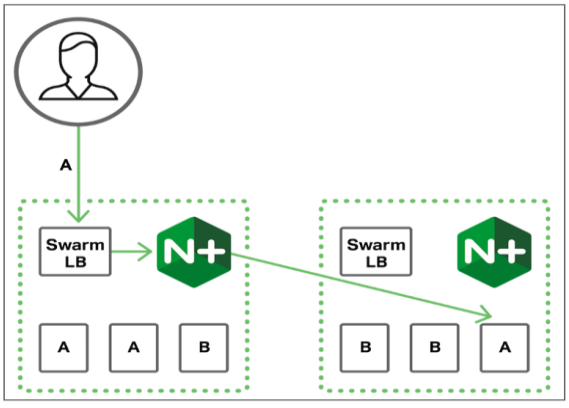

Figure 2. The Docker Swarm load balancer forwards client requests to NGINX Plus for load balancing among service instances

Nginx

Nginx, a web server used as a reverse proxy, HTTP cache and load balancer. It is built to high concurrency and low memory using an asynchronous approach in which requests executed by single thread, rather than creation of a new process for each web request. With Nginx, multiple worker processes are controlled by one master process. The master maintains worker processes, while workers do the actual processing of the server. Nginx is asynchronous, meaning each request received can be executed by the worker concurrently without blocking all other requests. Two load balancers are Open source NGINX and NGINX Plus that provide application critical features that are missing from the native Swarm load balancer. (Figure. 1. and Figure. 2.)

2. Experimental Setup

The configuration file for load balancing using Nginx is as follows:

```
http {
    upstream myapp {
        server srv1.sample.com;
        server srv2.sample.com;
        server srv3.sample.com;
    }
    server {
        listen 80;
        location / {
            proxy_pass http://myapp1;
        }
    }
}
```

We created 3 instances of an application running on server1- server 3. The default load balancing method used here is round-robin. All requests are provided as proxy to the server pool myapp1, and Nginx enforces HTTP load balancing to decimates the requests.

Reverse proxy implementation in Nginx consists of load balancing for HTTP, HTTPS, FastCGI, USWGI, SCGI and GRPC.

Least connected load balancing

A least-connected load balancing algorithm is used when some of the requests take too much time to complete. In such situations, the least connected ensures that the load on the application is controlled more fairly.

With the least-connected load balancing, Nginx will avoid overloading an already occupied application server with additional requests. Instead, it will distribute the new requests to the server, which is less busy.

To activate Least-connected load balancing in Nginx we have used least_conn directive as part of the server group configuration:

```
upstream myapp {
    least_conn;
    server srv1.sample.com;
    server srv2.sample.com;
    server srv3.sample.com;
}
```

Session persistence

In round-robin or least-connected load balancing, all clients' request is distributed to a different server. There is no assurance that the same client will be directed to the same server.

If a client needs to be tied to the same server, which means that if we have to make a client's session persistent in terms of always selecting the same server, we use the IP-hash load balancing mechanism.

In the IP-hash algorithm, the client's IP address is used as a hashing key to deciding which server in a server pool should be selected for the client's requests. This algorithm makes sure that the requests from the same client will always be focused on an identical server except when this server is inaccessible.

In the configuration file for IP-hash load balancing, we add the IP-hash directive to the server group configuration:

```
upstream myapp {
    ip_hash;
    server srv1.sample.com;
    server srv2.sample.com;
    server srv3.sample.com;
}
```

Weighted load balancing

We have another possibility to influence Nginx load balancing algorithms even further by using server weights. In the previous two methods for load balancing, we have not configured the server weights. This means that for a specific load balancing system, all specified servers are treated as equally eligible. When the server-weight parameter is defined for a specific server, the weight is considered as part of the load-balancing decision.

```
upstream myapp {
    server srv1.sample.com weight=3;
    server srv2.sample.com;
    server srv3.sample.com;
}
```

With this conformation, every five new requests will be distributed across the application instances in the following manner: Three requests will be directed to server1, one request will go to server2, and the other request will be directed to server 3.

In the recently updated versions of Nginx, it is also possible to use weights with the least-connected and IP-hash load balancing.

Health checks

In Nginx, the implementation of a reverse proxy algorithm involves in-band health checks of the server. Nginx will flag a server as being 'failed' if the response that it received from that particular server fails with an error.

The max_fails command sets the quantity of continuous failed attempts to interact with the server that ought to occur during fail_timeout. max_fails is set to 1 as a default value. At the point when it is set to 0, health checks are deactivated for this server. The fail_timeout parameter likewise characterizes to what extent the server will be set apart as failed. After fail_timeout interim after the server failure, Nginx will begin to effortlessly test the server with the live customer's solicitations. In the event that the tests have been fruitful, the server is set apart as a live one.

3. Implementation

Steps to create a Simple Load Balancer using Nginx

1) Creating Our Node.js Application

First, we make a basic Node.js application, which will fill in as a static HTML document. After making this node.js document, we containerize it and run it twice. Toward the end, we will arrange a dockerized NGINX case to send requests to the two instances of our application.

After this, we will have the option to arrive at http://loacalhost:8080 on our machine, which will get the outcomes from some occurrence. It will use the round-robin approach to choose which instance will recognize for every new request.

To make this node.js application, we initially make a directory for this application, which incorporates an index.js document that will react to HTTP demands.

2) Dockerizing Our Node.js Application

We will first make a file called 'Dockerfile' in our main directory so as to dockerize our Node.js application.

The content of Dockerfile looks as follows:

FROM node
RUN mkdir -p /user/scr/app1
COPY index.js /user/scr/app1
EXPOSE 8080
CMD ["node", "/user/scr/app1/index"]

After that, we have to make an image, from this Dockerfile, which should be possible through the command given below:

Docker build -t load-balanced-app1

Then we run both instances of the application with the following instructions:

Docker run -e "MESSAGE=Instance one" -p 8081:8080 -d load-balanced-app

Docker run -e "MESSAGE=Instance two" -p 8081:8080 -d load-balanced-app

Subsequent to running the two commands, we will have the option to open the two instances on the browser by going to http://localhost:8081 and http://localhost:8082. The main URL will show a message saying, "First case," the subsequent URL will show a message saying, "The second example."

3) Load Balancing using a Dockerized NGINX Instance

The two instances of our application running on various Docker containers and on different ports on our host machine, we configure an instance of NGINX to load balance demands between them. First, we will begin by making another directory called Nginx-docker.

In this directory, we have created a configuration file called nginx.conf with the following code:

Upstream my-app1{
server 172,17.0.1;8081 wieght=1;
server 172.17.0.1:8082 weigth=2;
}
Server{
Location/ {
Proxy_pass http://my-app1 }
}

This will be utilized to configure NGINX. On it we create an upstream collection of servers containing the two URLs that react for the instances of our application. By not characterizing a specific algorithm to load balance requests, we are utilizing round robin approach, which is the default on NGINX.

From that point onward, we configure a server property that allows NGINX to pass HTTP solicitations to http://my-application, which is dealt with by the upstream created previously.

After this, we will make the Dockerfile that will be utilized to dockerize NGINX with this setup. This document will contain the accompanying code:

FROM nginx
RUN rm/etc/nginx/conf.d/default.conf
COPY nginx.conf/etc/nginx/conf.d/default.conf

After successfully creating both the files, we will now build and then run NGINX container on Docker. To do that we run the following command:

docker build –t load-balance-nginx
docker run –p 8080:80 –d load balance-nginx

After the above configurations, we can simply open our web browser and access http://localhost:8080. In the case of everything went well, we will see a website page with one of the two messages: 'First instance' or 'Second instance.' In the event that we hit reload on our internet browser a couple of times, we will have understood that every now and then, the message showed switches between 'First instance' and 'Second instance.' Here the round-robin algorithm is being used in real-time.

B. Load Balancing using Kubernetes

1. Architecture and Design

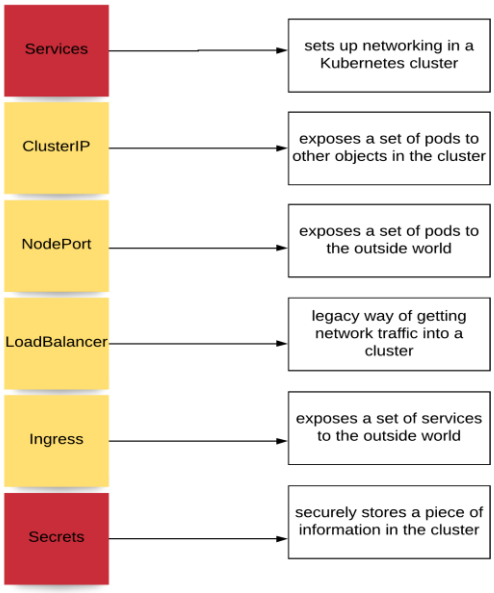

Figure 3. Flow chart of load balancing implementations using Kubernetes

Kubernetes permits two kinds of load balancing, i.e., Internal and External. Aside from these two, we have another technique, Ingress; it sits before various administrations and serves as a router into your cluster. (Figure. 3.)

Internal – otherwise known as "service" is load balancing crosswise over compartments of a similar kind utilizing a label. These administrations, by and large, reveal an internal cluster IP and port(s) that can be referenced inside as an environment variable to each unit. A service can load offset between these holders with a solitary endpoint. It takes into account container failures and even node failures inside the cluster while preserving the availability of the application.

External – Services can likewise go about as outer load balancers whenever wanted that is through a NodePort or LoadBalancer or Ingress type.

NodePort

NodePort opens a significant level port remotely on each node in the cluster. Naturally, somewhere close to 30000-32767. When scaling this up to at least 100 hubs, it turns out to be a little faulty. (Figure 4.)

There are numerous drawbacks to this strategy:

1. Has just one service for each port.
2. Only 30000–32767 can be utilized.
3. If your Node/VM IP addresses changes, appropriate changes will have to be made.

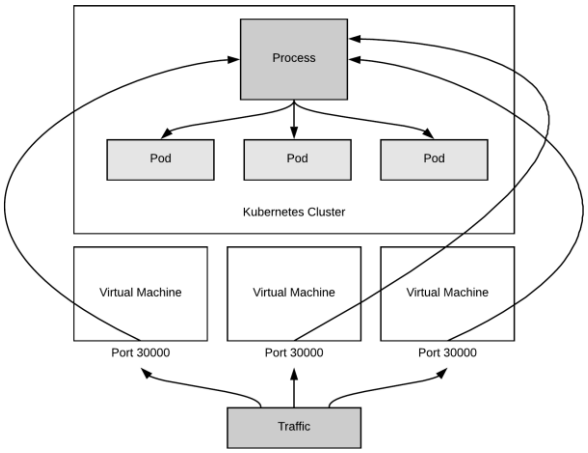

Figure 4. Traffic Control of NodePort

LoadBalancer

LoadBalancer helps by creating an external load balancer for you if your running Kubernetes in GCE, AWS, or another supported cloud provider. The pods get exposed to a high range external port and the load balancer routes directly to the pods. This bypasses the concept of service in Kubernetes, still requires high range ports to be exposed, allows for no-no segregation of duties, mandates all nodes in the cluster to be externally routable (at minimum) and will result in triggering real issues if you have more than X number of applications to expose where X is the range created for this task. The downside is that each service that is exposed to the LoadBalancer will get its own IP address, and one will have to pay for a LoadBalancer per exposed service. (Figure 5.)

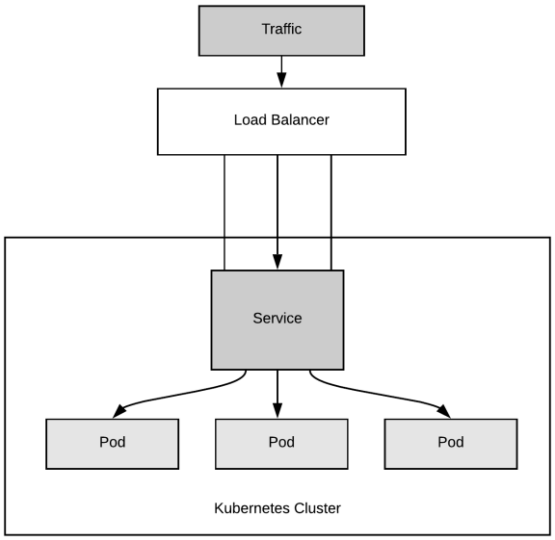

Figure 5. Traffic control of LoadBalancer

Ingress

Ingress serves as a kind of a smart router for your cluster that sits in front of multiple services. The inbuilt GKE ingress controller will spin up an HTTP(S) Load Balancer for the user. Ingress is perhaps the most powerful way to expose services, but can also be rather complicated. There are different types of Ingress controllers, which include Google Cloud Load Balancer (GCLB), Nginx, Contour, etc. (Figure 6.).

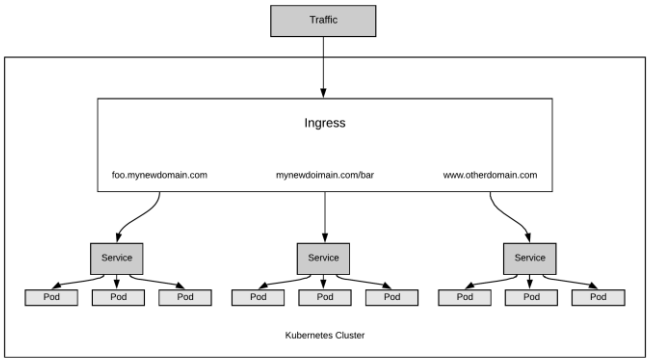

Figure 6. Traffic control of Ingress

2. Experimental Setup

1) Setting up the Dockerfile for worker node

```
FROM node:alpine
WORKDIR "/app"
COPY ./package.json ./
RUN npm install
COPY . .
CMD ["npm", "run", "start"]
```

2) Dockerfile for server node

```
FROM node:alpine
WORKDIR "/app"
COPY ./package.json ./
RUN npm install
COPY . .
CMD ["npm", "run", "start"]
```

3) Dockerfile for client node

```
FROM node:alpine as builder
WORKDIR '/app'
COPY ./package.json ./
RUN npm install
COPY . .
RUN npm run build
FROM nginx
EXPOSE 3000
COPY./nginx/default.conf/etc/nginx/conf.d/default.conf
COPY--from=builder/app/build /usr/share/nginx/html
```

4) Configuration file for Ingress

```
server {
 listen 3000;
 location / {
   root /usr/share/ingress/html;
    index index.html index.htm;
  }
}
```

3. Implementation

1) client-cluster-ip-service.yaml file

```
apiVersion: v1
kind: Service
metadata:
   name: client-cluster-ip-service
spec:
   type: ClusterIP
   selector:
      component: web
ports:
      -port: 3000
          targetPort: 3000
```

2) client-deployment.yaml file

```
apiVersion : apps/v1
kind: Deployment
metadata:
   name: client-deployment
spec:
   replicas: 3
selector:
   matchLabels:
      component: web
template:
   metadata:
label:
    component: web
   spec:
      containers:
         -name: client
            Image: Virtualization/multi-client
            ports:
               -containerPort: 3000
```

IV. CONCLUSION

Before container technologies, deploying an application normally took a long time. Deployment was done manually, which cost the company time and resources. When container technologies became popular with Docker and Kubernetes, the entire process became more streamlined and standardized. Currently, in the field, load balancing for containerized applications exists in multiple forms for different use cases, but they remain unexplored with each having their own advantages and disadvantages

The capacity to scale and finds benefits in Docker is currently simpler than at any other time. With the administration revelation and burden adjusting highlights incorporated with Docker, designers can invest less energy making these sorts of supporting capacities all alone and additional time concentrating on their applications. Rather than making API calls to set DNS for administration disclosure, Docker consequently handles it for you. In the event that an application should be scaled, Docker deals with adding it to the heap balancer pool. By utilizing these highlights, associations can convey exceptionally accessible and flexible applications in a shorter measure of time.

The problems with existing methods are that they are not efficient enough, and the new algorithms being designed are not scalable to be widely used. With the increasing use of containers, load balancing will become a necessity, and further research is needed on efficient implementations.

Research Challenges

One observation we can see through this research on the different load balancing techniques is that there is no clear understanding of why specific algorithms are better and how each can be improved. Researchers have tried implementing their own algorithms or enhance the current algorithms but have not been able to completely change load balancing in container environments. Containers are still a developing field, and much research remains to be done, and as we move forward, load balancing becomes increasingly essential. Load balancing will improve system performance and reduce carbon emissions. We look to show which algorithms can be used for different purposes and seek to implement the techniques comparing their performance.

Currently, there is a universal acceptance that with respect to container orchestration, Kubernetes performs much better than Docker Swarm. One of the reasons Kubernetes is widely used instead of the native Docker cluster, Docker Swarm, is its scalability, portability and self-healing attributes. Kubernetes has been around longer than Docker Swarm and therefore has much more documentation. We look to study why Kubernetes is more famous for implementing load balancing.

REFERENCES

1. Netto, Hylson V., et al. "State machine replication in containers managed by Kubernetes." Journal of Systems Architecture 73 (2017): 53-59.
2. Rusek, Marian, Grzegorz Dwornicki, and Arkadiusz Orłowski. "A decentralized system for load balancing of containerized microservices in the cloud." International Conference on Systems Science. Springer, Cham, 2016.
3. Sajjan, Rajani. (2017). Load Balancing and its Algorithms in Cloud Computing: A Survey.
4. Cito, Jürgen & Ferme, Vincenzo & C. Gall, Harald. (2016). Using Docker Containers to Improve Reproducibility in Software and Web Engineering Research. 609-612. 10.1007/978-3-319-38791-8_58.
5. C. Cérin, T. Menouer, W. Saad and W. B. Abdallah, "A New Docker Swarm Scheduling Strategy," 2017 IEEE 7th International Symposium on Cloud and Service Computing (SC2), Kanazawa, 2017, pp. 112-117. doi: 10.1109/SC2.2017.24
6. A Scheduling Strategy on Load Balancing of Virtual Machine Resources in Cloud Computing Environment.
7. Wei-guo, Zhang & Xi-lin, Ma & Jin-zhong, Zhang. (2018). Research on Kubernetes' Resource Scheduling Scheme. 144-148. 10.1145/3290480.3290507.

8. Bui, Thanh. "Analysis of Docker security." arXiv preprint arXiv:1501.02967 (2015).
9. Di Tommaso, Paolo, et al. "The impact of Docker containers on the performance of genomic pipelines." PeerJ 3 (2015): e1273.
10. Harter, Tyler, et al. "Slacker: Fast distribution with lazy Docker containers." 14th {USENIX} Conference on File and Storage Technologies ({FAST} 16). 2016.
11. Naik, Nitin. "Applying computational intelligence for enhancing the dependability of multi-cloud systems using Docker swarm." 2016 IEEE Symposium Series on Computational Intelligence (SSCI). IEEE, 2016.
12. Nguyen, Nikyle, and Doina Bein. "Distributed mpi cluster with Docker swarm mode." 2017 ieee 7th annual computing and communication workshop and conference (ccwc). IEEE, 2017
13. Ismail, Bukhary Ikhwan, et al. "Evaluation of Docker as edge computing platform." 2015 IEEE Conference on Open Systems (ICOS). IEEE, 2015.
14. Kherani, Foram F. "Prof. Jignesh Vania, "Load Balancing in cloud computing"." International Journal of Engineering Development and Research 2.1 (2014).
15. James, Jasmin, and Bhupendra Verma. "Efficient VM load balancing algorithm for a cloud computing environment." International Journal on Computer Science and Engineering 4.9 (2012): 1658.
16. Yagoubi, Belabbas, and Yahya Slimani. "Dynamic load balancing strategy for grid computing." Transactions on Engineering, Computing and Technology 13.2006 (2006): 260-265.
17. Data storage and load Balancing in cloud computing using container
18. clustering Trapti Gupta & Abhishek Dwivedi
19. Ren, Xiaona, Rongheng Lin, and Hua Zou. "A dynamic load balancing strategy for cloud computing platform based on exponential smoothing forecast." 2011 IEEE International Conference on Cloud Computing and Intelligence Systems. IEEE, 2011.
20. Kaewkasi, Chanwit, and Kornrathak Chuenmuneewong. "Improvement of container scheduling for Docker using ant colony optimization." 2017 9th international conference on knowledge and smart technology (KST). IEEE, 2017.
21. Research and implementation of Docker performance service in distributed platform Liu Lijuan
22. Rad, Babak Bashari, Harrison John Bhatti, and Mohammad Ahmadi. "An introduction to Docker and analysis of its performance." International Journal of Computer Science and Network Security (IJCSNS) 17.3 (2017): 228.
23. Zhang, Dongsheng. Resilience enhancement of container-based cloud load balancing service. No. e26875v1. PeerJ Preprints, 2018
24. Value-Based Allocation of Docker Containers by Piotr Dziurzanski, and Leandro Soares Indrusiak
25. Takahashi, K., Aida, K., Tanjo, T., & Sun, J. (2018). A Portable Load Balancer for Kubernetes Cluster. Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region - HPC Asia 2018. doi:10.1145/3149457.3149473
26. Cloud Resource Orchestration: A Data-Centric Approach By Changbin Liu, Yun Mao, Jacobus E. Van der Merwe, Mary F. Fernández
27. Prajval Mohan, Adiksha Sood, Lakshya Sharma, Simran Koul, Simriti Koul. 'PC-SWT: A Hybrid Image Fusion Algorithm of Stationary Wavelet Transform and Principal Component Analysis.' International Journal of Engineering and Advanced Technology (IJEAT). ISSN: 2249-8958 (Online), Volume-9 Issue-5, June 2020, Page No.700-705.
28. Polyphony: A Workflow Orchestration Framework for Cloud Computing Khawaja S Shams, Dr. Mark W. Powell., Tom M. Crockett, Dr. Jeffrey S. Norris, Ryan Rossi, Tom Soderstrom
29. SDN orchestration architectures and their integration with Cloud Computing applications By Arturo Mayoral, Ricard Vilalta, Raul Muñoz, Ramon Casellas, Ricardo Martínez
30. An in-depth analysis and study of Load balancing techniques in the cloud computing environment. By Geethu Gopinath P P, Shriram K Vasudevan
31. Dynamic Balance Strategy of High Concurrent Web Cluster Based on Docker Container. Weizheng Ren et al 2018 IOP Conf. Ser.: Mater. Sci. Eng. 466 012011
32. Prajval Mohan, Pranav Narayan, Lakshya Sharma, Tejas Jambhale, Simran Koul, "Iterative SARSA: The Modified SARSA Algorithm for Finding the Optimal Path". International Journal of Recent Technology and Engineering (IJRTE). ISSN: 2277-3878, Volume-8 Issue-6, March 2020

AUTHORS PROFILE

Prajval Mohan was born in Hyderabad, India on 23rd October, 1998. He completed his Senior High School from FIITJEE Junior College, Hyderabad, graduated with 96.1 percent and received the Honorary Certificate of Merit in the year 2016. Prajval is currently pursuing his B. Tech in Computer Science and Engineering from Vellore Institute of Technology, Vellore, India. His areas of interest include Robotics, Machine Learning, Artificial Intelligence and Cloud Computing. He has advanced working knowledge of Robotics and Database handling which were strengthened by completing various projects and internships in the respective fields. He also has ongoing research in the field of Deep Learning and Parallel Distributed Computing.

Tejas Jambhale was born in Maharashtra, India on 17th November, 1998. He is a student of Vellore Institute Technology, Vellore currently pursuing Computer Science Engineering. He has skills in domains including machine learning, web development and deep learning. He has completed research in cyber security and deep learning and likes building different projects to explore his skills.

Lakshya Sharma was born in Jaipur, India on 25th January, 1999. He was raised in Delhi, India and completed his Senior high school from D.A.V Public School. He is currently pursuing B. Tech in Computer Science Engineering from Vellore Institute of Technology, Vellore. His research interests include Deep learning, artificial intelligence, autonomous object avoiding and path planning robots. He has worked on several machine learning, deep learning projects and has an ongoing research in offline signature recognition using Siamese networks.

Simran Koul was born in Jammu, India on 10th November, 1998. She completed her senior high school in Indian School Ahmadi, Kuwait. She is currently pursuing her Bachelor's Degree in Computer Science Engineering in VIT, Vellore, India. She is currently working on projects which involve concepts of Robotics and Artificial intelligence and Natural Language Processing.

Simriti Koul was born in Jammu, India, on 10th November 1998. She completed her senior high school in FAIPS DPS, Ahmadi, Kuwait. She is currently pursuing her Bachelor's Degree in Computer Science Engineering at Vellore Institute of Technology, Vellore, India. She is currently working on projects which involve concepts of Artificial intelligence, Data Analytics and Natural Language Processing.