International Journal of Innovative Technology and Exploring Engineering (IJITEE)
ISSN: 2278-3075 (Online), Volume-9 Issue-5, March 2020

Round Robin Load Balancer for Node Swarm Clusters Running a Chatting Service on the Cloud

Tanmay Jain, Ayush Raina, Boominathan P

Abstract*: *In this paper, we have created a chat application which uses socket programming for communication and all of the messages are saved in mongoDB. We have taken Docker application and hosted it on a three-node swarm cluster. This cluster uses Docker swarm technology to create a private network through which each of the nodes can talk to each other along a specified RPC port. The application runs in each node as a service and all load coming to the application has been balanced across three IP addresses in the swarm. This creates a distributed system and each node can act as a manager or a worker in the system. This technique helps to decrease the execution time to run servers on the cloud and can help improve the feasibility of online servers provided by the IT companies.

Keywords: Docker, Virtualization, Socket Programming, Swarm Technology.

I. INTRODUCTION

These days, distributed computing is the business membership to outer administrations. Its guideline depends on the compensation for-use display that can influence diverse components, for example, the mentioned application, information stockpiling limit, memory handling and number of clients. The fundamental targets of this task are to manage and to setup: [1]

Receipt, in light of asset utilization so as to accomplish the business effectiveness of the organization;

Avoid squander in asset utilization and devour as it were what is vital and productive to applications to accomplish vitality effectiveness.

Facilitate for the client the appropriation of the compartment's innovation to build up a commercial centre. With regards to venture, we propose, in this paper, our underlying thoughts for another booking methodology executed in Docker Swarm, the principle innovation utilized in the venture.

This paper is for the most part committed to usage subtleties and general thoughts. More work has been done for the current year to broaden this work, see for example and a general union of our system is at present under looking into. The Discussion area will clarify, last on, some situating actualities of this work contrasted with progressing works. [2]

Revised Manuscript Received on March 30, 2020.

***** Correspondence Author

Tanmay Jain*, BTech Student in Computer Science and Engineering at Vellore Institute of Technology, Vellore, India.

Ayush Raina, BTech Student in Computer Science and Engineering at Vellore Institute of Technology, Vellore, India.

Dr. Boominathan P, Associate Professor, School of Computer Science and Engineering, Vellore Institute of Technology.

Docker Swarm is a bunching and booking apparatus for Docker holders. It chooses the principal demand that must be executed by utilizing the traditional FIFO system. At that point, Docker Swarm picks the suitable machine for example fulfilling the need of the client as far as CPU centres, by utilizing one procedure among the accompanying methodologies:

Spread technique which is embraced of course by the supervisor if no technique has been determined while making the Swarm administrator and it executes a compartment on the hub having the least number of compartmentsBinpack procedure, interestingly with spread, picks the hub with the most stuffed compartments on it Random system which picks a hub arbitrarily. In our specific circumstance, the last client isn't important a specialist in parallel processing and burden adjusting systems to know what number of CPU centres must be utilized to execute its application. In any case, our technique has two oddities. It chooses the main application that must be executed by the client monetary model. At that point, it processes powerfully for the chosen application the quantity of CPU centres, as indicated by the client monetary model and the heap of the parallel machines saved in the private framework. At long last, to execute the application/compartment, we propose to utilize one procedure among the prior procedures of Docker Swarm. [3]

Our proposed financial model relies upon three SLA classes to answer to the customers' needs. The showing with three classes is motivated by the mechanical endeavours and relies upon the discernment that encouraging courses of action don't allow makers or cloud providers to offer to their customers a sensible or exact receipt [2], for instance a precise receipt with reverence to the use of benefits by the referenced organization. Seen that the errand must respond to the going with use cases concerning the sent organizations, every organization addresses one SLA class:

Long administration it is a determined administration encountering pinnacles of burden (CPU, smash, plate, organize) like times of dormancy.

Short administration it is an administration causing an over-burden (CPU, slam, plate, arrange) or a sitting tight for a remote administration. Precedent: transformation to PDF.

Miniaturized scale administration: administration life is not exactly the recurrence of measurements' accumulation. Precedent: measurements' gathering happens each moment and the lifetime of the administration is a couple of milliseconds.

II. PROPOSED SYSTEM

The first task is creating a chat application for which we will be using a bootstrap template for the frontend.

For the backend we will be doing socket programming in node is and we will be catching the emitted events in the frontend while saving the chats in mongo DB for history. The second task is creating three nodes on the cloud. For this we will be using digital ocean hosting service. Then we will install Docker on all three nodes and initialize the Docker swarm. Then we will generate a join token which the other nodes can use to join the swarm as managers.

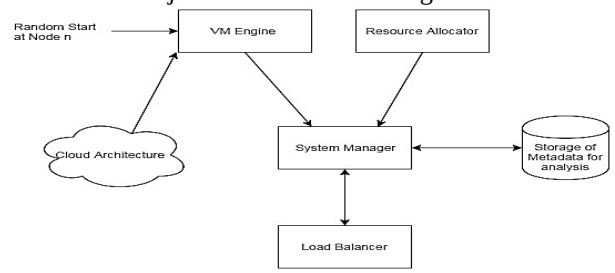

Figure 1: System Model

A. VM Engine

Its job is to maintain a sub-level between the system/server, so as to provide seamless dataflow in the given model. It maintains all the virtual nodes and keeps them at a separate level-1. It provides the start for the virtual machine and also maintains the cloud architecture in-order ton ensure security protocols between server and n-node user.

B. Resource Allocator

This part manages resource allocation in the system, as many virtual servers try to access the system it will create different allocation times based on priority. It is a very important component of the model as it will help manage the resources that might try to enter the critical section at once.

C. System Manager

It will overlook the complete model. It can be considered as the master component of the model because everything has to pass through the system manger. It will keep in check if each and every module is working properly because if it isn't it could create large security breaches which will make it difficult for the system administrator to manage the virtual network.

D. Load Balancer

Its job will be to avoid deadlock after the resources have been allocated. It will help decrease the starvation through context switching which will help in the prevention of deadlock as it will not allow a virtual node-n to access the Critical section for a long period of time.

The Final task is creating a service in one of our manager nodes. For this we will pull the Docker image of the chat application from Docker-hub and running it on port 80. The entire model will execute and will provide seamless access to the application which has a well-developed front-end.

Algorithm

Begin - : T(Queue), C(Container)
while (true) do
if R is submitted **then**
Update the priority of all containers saved in T
Add C in T
Order all the containers saved in T according to their priority
end if
if T is not empty **then**
C∗= container with the highest priority in T
Compute the number of CPU associated to C∗
if R∗ can be executed **then**
Remove C∗ from T
Execute C∗
end if
end if
end while
Stop

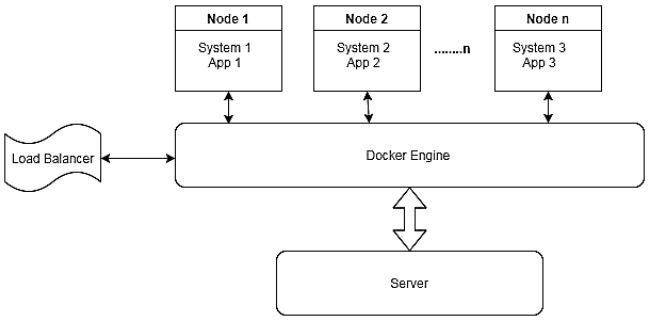

Figure 2: Network Architecture

This figure represents the network for the given model. The Docker engine will be taking all the nodes head-on and creating safe and secure resource allocation with the help of the Load Balancer which will help prevent any unnecessary problems in the system. The Docker Engine will be in direct connection to the server managing each and every requirements of the server. This will create an ease of network with faster execution time and efficiency. The applications that will be running on the virtual network will be running with almost zero lag. This will also ensure safety of the server and its resources. This type of combination for the model and the network will help in improving the already existing systems.

III. IMPLEMENTATION

The below implementation requires both hardware and software components.

A. Backend System

This part shows the execution of Node-1 and how many nodes in the swarm is it reachable to, this provides the controls to Node-1 which is the node manager and it will further add reachable nodes as node manager creating a chatting application on a virtual environment.

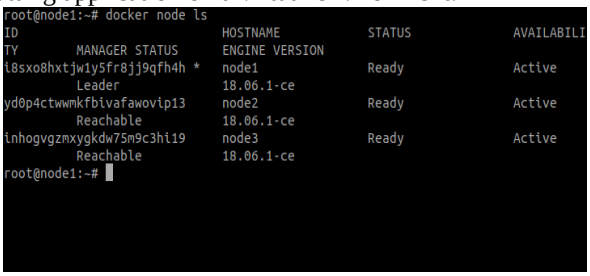

Figure 3: List of Reachable nodes through Node-1

After Execution of Node-1, it makes many other Nodes as the manager of the swarm, in this output we can see.

Node 2 has become a new manager and the number of reachable nodes are increased. This explains the reason for faster execution, it can add more number of participants by increasing number of managers.

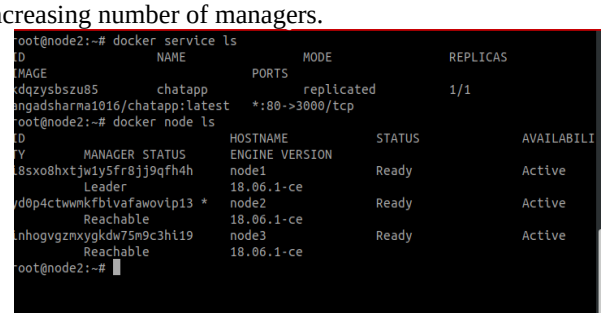

Figure 4: Node 2 showing its reachable nodes

We can see that in few seconds from Node-1 we have generated many nodes with whom we can chat on a virtual environment. Due to the fact that everything is being controlled virtually the system always thinks he is in control and in-return provides memory and efficiency.

B. Frontend System

This is the face of our virtual network, it is easy to understand and requires no in-depth knowledge of the subject, this helps to give a way to check system capability.

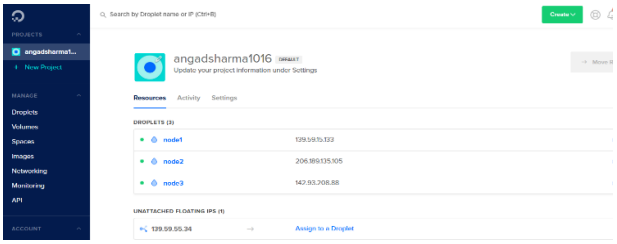

Figure 5: Three Nodes Hosted on DigitalOcean

The image shows that we there are three nodes present in the swarm at that particular instant, through these nodes we will run a chat-application on a virtual environment. These three photos show how our chat app is running on all three IP addresses and they have intercommunication. The implementation part included two sets of our project backend and frontend.

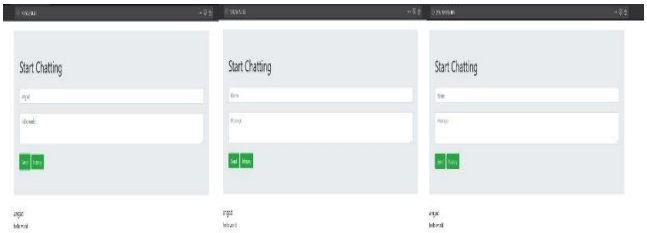

Figure 6: Depicts all three Nodes chatting with each other

The Backend portion was responsible for introducing new nodes, creating resources, making nodes manager so they can start chatting on a platform in a secure manner. It was also responsible for directly interacting with the server which gave Docker engine the power to create a complete virtual environment in the project. This implementation resulted into some improvements of this project on the existing ones running on the same domain. The next part is the Frontend which controlled the face of the project. It was responsible for taking key user inputs and sending it to backend. It gave the chat application a look and usability that each and every one of us could use the chat application without knowing in-depth about the subject.

IV. RESULTS AND DISCUSSION

In this part we will discuss how our project was an improvement from the last implemented projects of the same ideas. We will provide a complete analytical comparison between the two and determine the best one which can be used in real life scenarios.

A. Analysis of KVM vs Docker on Execution-time

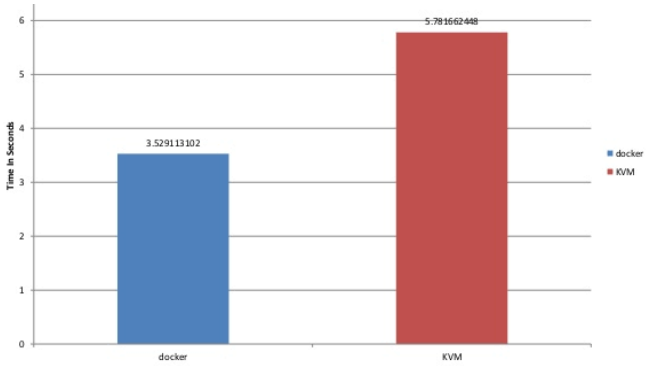

Figure 7: KVM vs Docker

The Execution-time of Docker is way better than the Execution time KVM (Existing Technology). This shows that in terms of speed and efficiency Docker is way ahead of the already present technologies. It provides a key infrastructure of virtualization with great implementation at an eased-out rate. We also determined through Brute-force that Docker provides more resistant to attackers on the server. In conclusion Docker is Reliable, safe, secure, efficient and fast.

B. lculation of CPU Performance

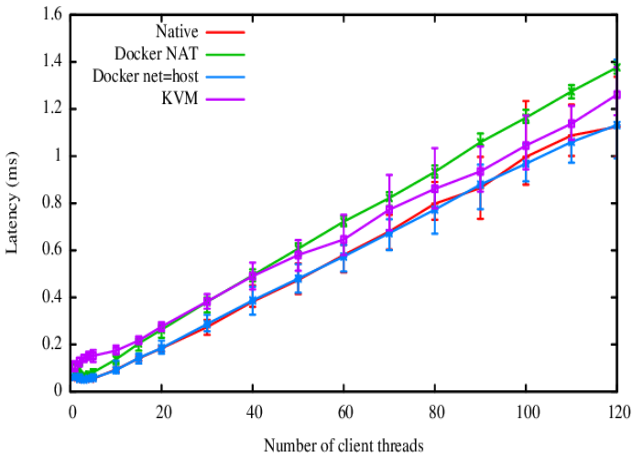

Figure 8: CPU comparison between VM and Docker

We can clearly see that the CPU performance does not deteriorate using Docker as a VM alternative whereas VM takes up a lot of CPU space creating possibilities for a deadlock or glitches in the system.

C. Random I/O Read/Write Analysis

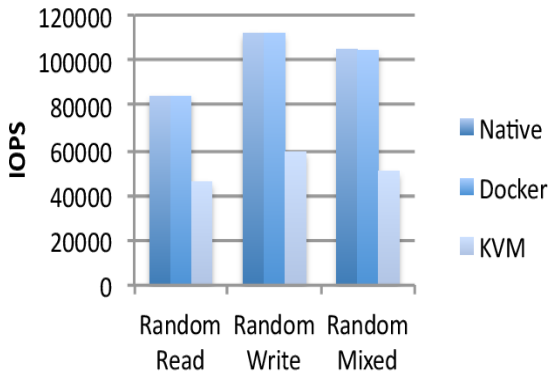

Figure 9: Random Read/Write

This draws another point in favour of Docker as it has a very fast Read/Write for Every Instruction/second. Overall Docker provides faster Read/Write capabilities than the native system. It is however slower than KVM due to context switching in Docker.

Table 1: Tabular comparison between the two Systems

SN	EXISTING SYSTEM	PROPOSED SYSTEM
1	Existing framework takes a put away informational collection on a specific subject into thought.	Proposed framework will give you the opportunity to pick the information of any subject.
2	It neglects to decide the effect the outcomes may or will have in the individual field.	Here, it gives you the effect the outcomes and measurements will have on the separate field.
3	Existing framework does not permit the recovery of information dependent on the inquiry entered by client.	Proposed framework permits recovery of information dependent on the inquiry entered by the client.
4	Existing framework does not give precise component determination.	Proposed framework will give exact element choice.
5	UI product services not provided	UI product services provided

Through all the depictions we can make a Final conclusion that the Proposed System has the best implementation with better efficiency than the Existing System. The proposed system outweighs the existing system in each and every department, it has a much better execution time with reliability and it provides secure and safe communication between nodes.

V. CONCLUSION

Using technologies like Docker and Docker swarm we were able to deploy a standard chat application to a three-node swarm cluster where each node can be either a swarm manager or a swarm worker and communicating among each other via an open RPC port. Thus, we created a load balancer for our chat application. The implementation part included two sets of our project backend and frontend. The Backend portion was responsible for introducing new nodes, creating resources, making nodes manager so they can start chatting on a platform in a secure manner. It was also responsible for directly interacting with the server which gave Docker engine the power to create a complete virtual environment in the project. This implementation resulted into some improvements of this project on the existing ones running on the same domain. The next part is the Frontend which controlled the face of the project. It was responsible for taking key user inputs and sending it to backend. It gave the chat application a look and usability that each and every one of us could use the chat application without knowing in-depth about the subject.

The future work that underlies within our scope are listed as follows:

A. Including Machine Learning

We will provide ML algorithm in our system to develop an understanding of nodes using Naïve Bayes Algorithm or Natural Language Processing. This will provide a much better and accurate model and help achieve 100% accuracy in our model.

B. Global Implementation

We our using the model to only help election process in India. We will be implementing this model for other countries and according to their country we can train different models which will suit that particular country.

C. Providing more than one application for the same

This time we could only work on one application, in the future we will be implementing much more

D. Providing a much in-depth analysis on KVM vs Docker

We will provide a much more in-depth analysis for the upcoming future.

REFERENCES

1. G. Rastogi and D. R. Sushil, "Analytical Literature Survey on Existing Load Balancing Schemes in Cloud Computing," International Conference on Green Computing and Internet of Things , 2015.
2. B. B. Rad, H. J. Bhatti and M. Ahmadi, "An Introduction to Docker and Analysis of its Performance," International Journal of Computer Science and Network Security, vol. 17, no. 3, pp. 228-235, 2017.
3. C. Cerin, "A New Docker Swarm Scheduling Strategy," Universite de Paris.
4. K. S. Akshay, K. J. Anish, K. Amit, K. T. Vivek and H. Praveen, "RESOURCE MONITORING OF DOCKER CONTAINERS," International Journal of Advance Engineering and Research Development.
5. S. S. Hegde and D. P. Jayarekha, "Basic Analysis of Docker Networking," International Journal of Advanced Research in Computer Science, vol. 8, no. 5, pp. 740-743, 2017.
6. P. J. Dilip and P. K. Nangana, "Docker, An Optimized Virtualization Using Container," International Journal for Research in Applied Science & Engineering Technology.
7. P. Dziurzanski and L. S. Indrusiak, Value-Based Allocation of Docker Containers.

AUTHORS PROFILE

Tanmay Jain, BTech student in Computer Science and Engineering at Vellore Institute of Technology, Vellore, India. He has previously worked on image restoration and object detection. His current research interests also include data mining and human computer interaction.

Ayush Raina, BTech student in Computer Science and Engineering at Vellore Institute of Technology, Vellore, India. He has previously worked on image recognition. His current research interests include data mining and machine learning.

Dr. Boominathan P, Associate Professor, School of Computer Science and Engineering, Vellore Institute of Technology.