Scaling and Load-Balancing Applications with Kubernetes

Tyler Rowland
rowla070@morris.umn.edu
Division of Science and Mathematics
University of Minnesota, Morris
Morris, Minnesota, USA

Abstract

Containerized applications have become the best way to create application. Kubernetes is a tool used for automation of management and deployment of these applications. In this paper I will look at two studies on load-balancing and horizontal scaling. These two papers look at how each aspect increases the performance of applications.

Keywords: Kubernetes, containers, applications, Load-balancing, Scaling

1 Introduction

We use computer applications everyday, whether it is on our phones, laptops, or even televisions now. Applications have become an integral part of our lives.

Deploying and management of the application led to long nights and delayed releases. Also these applications could only be deployed on-premises, on a server in a data center. Engineers had to manage the application by looking at the number of user requests and network traffic to make sure the server wouldn't become overwhelmed. Engineers would then use load-balancing and scaling to make sure the servers could run efficiently. In a data center is a server becomes overwhelmed it could shut down, which causes downtime for applications. Cloud computing allows engineers to only have to manage the application being used and not have to worry about a server. Data centers are more expensive than cloud computing in the sense that you have to pay for the gear and the manpower to run them. Cloud computing, though cheaper, takes a lot of understanding of how to manage the money being spent. Changes to the application can all be done remotely, because there is no need to physically connect to anything in the data center. For example, an application called 'Shoe Guru' in the early 2000s ran their application in their data center in Silicon Valley. There were many developers that worked on this application, some worked at night to make sure everything was running fine in the data center. Whereas others worked the typical 9 to 5 work day looking at how to improve their application. Within the last 5 years developers of this application started looking at cloud computing and the advantages it brings. From there they moved 'Shoe Guru' to the cloud, which allows the developers only focus on improving of their application.

Cloud computing has brought on a new type of application. Containerized applications have become very popular since the boom of cloud computing. These applications run in containers. Containers are a standard unit of software that packages up code and allows applications to run faster and more consistent between environments. Containerized applications give us the option to run applications in the cloud, on-premises, or both. Also deploying these apps are much easier, which was what was so intriguing to software development teams such as Shoe Guru.

Kubernetes is used to manage these types of applications. Kubernetes was first released in 2015 by Google [5]. While Google was the first company to create and use a Kubernetes engine, software companies such as Red Hat, VMware, and Cloud computing competitors have their own versions as well. Big applications have been using Kubernetes because of the scalability or load-balancing and easy deployment. Kubernetes is a containerized application tool that automates the deployment and management process. Before, developers were very focused on the back-end of their application. They constantly made sure that it could handle an increased amount of traffic, and that everything was working correctly in their servers. Kubernetes is used so developers can focus on how an application looks and feels as well as integration of new features.

Kubernetes implements features to improve the deployment and management of applications. In this paper we are going to describe some of these features and how they improve the management of applications. Load-balancing and scaling help to improve not only managing applications, but also helps to increase the performance. These two features will be the main aspects of the paper.

In this paper we will first explore the necessary terminology to understand load-balancing and scaling in Kubernetes in section 2. Included will be the hierarchy of Kubernetes units, API server, and CPUs. Section 3 will look at scaling, more specifically CPU usage and response time of a single server versus multiple servers. This section also will look at the results of the study that we looked at. Lastly, Section 4 will dive into the load-balancing aspect of the paper. The study evaluated in this section explores distribution of the workload. The discussion will involve CPU utilization and

throughput of two clusters. Finally, the conclusion which will sum up this paper.

2 Background

In this section we are going to dive into the features that Kubernetes provides for big applications. Aspects such as load-balancing and scaling will be explored first, followed by an explanation of the Kubernetes hierarchy. Then we will look at the API server that Kubernetes uses. Finally is a section about other concepts needed to understand the paper.

The first feature we are going to explore is load-balancing. Load-balancing controls the network traffic of an application and distributes it to other replicated applications. This is so the application doesn't become too overwhelmed. This is helpful because in Kubernetes it is all automated. When the application uses too many resources and becomes slow, Kubernetes distributes this traffic to the different nodes and pods in the cluster. Load-balancing is the feature that controls this, which is done to avoid bottlenecks. A bottleneck happens when there is a workload imbalance in the worker nodes. Which, in turn, leads to decreased performance of the system [4].

Horizontal scaling is very similar to load-balancing in the sense that it helps to take the burden off of a machine. Where load-balancing distributes network traffic; horizontal scaling refers to changing the number of machines. When there is a large amount of requests or traffic, Kubernetes changes the number of machines to handle the load efficiently. If needed this can all be done automatically which is why so many big companies use Kubernetes.

2.1 Client-Server Architecture

Figure 1 shows the client server architecture in Kubernetes. In this case there are 3 nodes in the cluster, one master node and two worker nodes. User requests play a big part in the structure of this paper, as well as the usage of load-balancing and scaling. The user requests come from the users in the figure. These requests are sent over the internet to the Kubernetes cluster. The master node then distributes those requests to a worker node, where the requests are split up to each pod, container, and application. For example, in the Shoe Guru application, multiple users send requests to the application to order pairs of shoes. These requests are received and authenticated by the master node of Shoe Guru then distributed to the worker nodes for them to handle.

In the case of deployment, this is all set up by the users but done by Kubernetes. The user specifies in the master node the details of the deployment. The master node then deploys the application in the worker nodes. From there the worker nodes are in charge of managing the application that's deployed.

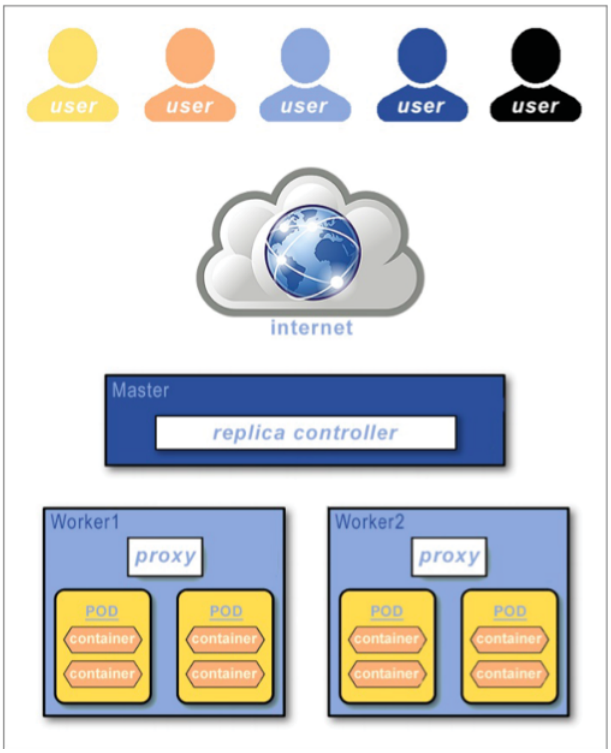

Figure 1. Client Server architecture [3]

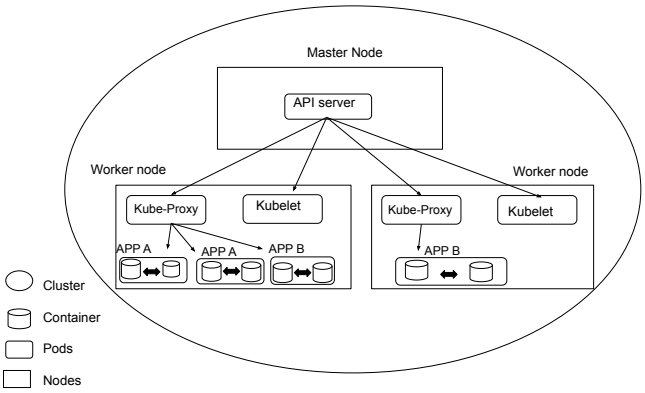

Figure 2. Kubernetes Hierarchy [4]

2.2 Concepts and the Kubernetes Hierarchy

Kubernetes uses containers, clusters, and pods to run its applications. A container is where the application is deployed. A container is a standard unit of software that packages up code and allows application to run fast and more consistent between environments [1]. Containers nest inside pods and one or more containers makes up a pod. A pod is the smallest execution unit, meaning it is the smallest

unit that is deployed by Kubernetes. The pods in Kubernetes share storage, which in the case of one of the pods getting shut down, the shared storage allows for back up for the pods. [4]. Applications in Kubernetes are typically replicated. This means there are multiple pods in each node that contain the same application. The containers in Figure 2 contain these applications. These replicated applications are used for load-balancing. Kubernetes distributes the workload to these applications to improve performance. User requests are also handled in these replicated application pods. Each pod in Kubernetes also contains it's own data storage container or data store. These data stores contain aspects of the application that the user can write data to or get data from. Usage of data stores is to ensure consistency throughout the Kubernetes cluster as well as the application.

Pods are run inside of nodes. Nodes in Kubernetes can either be a physical server or a virtual machine. Looking at the Shoe Guru app, at the beginning of their existence a node would have been a physical server in a data center. Now Shoe Guru uses virtual machines for their nodes. There are 2 types of nodes in Kubernetes: master nodes and worker nodes. In a cluster there can only be one master node, but there can be as many worker nodes as desired. The master node contains pods that are needed to keep the cluster running and available for requests. This node controls all of the management of the Shoe Guru application. Worker nodes contain pods, containers, and applications. They are used to handle all of the network traffic and user requests of the Shoe Guru application.

The master node controls the entire Kubernetes cluster. Inside the master node in Figure 2 there is the API-server. The API-server does two things in Kubernetes. First it is used to receive the user requests and services from the user. From there, the API distributes those request to the other aspects of the cluster. The API-server also can be used to change the number of nodes, pods, and containers used in the cluster [4].

Inside the worker nodes in Figure 2 there are two pods labelled kube-proxy and kubelet. Kube-proxy contains the network rules for communicating with the application pods. Kubelet manages all of the containers in the node while also talking with the master node about status of the worker node it is in. Typically there can be as many worker nodes as desired and are all managed by the one master node.

When we talk about a Kubernetes cluster we mean all of the nodes, pods, containers, and applications as well as what's inside of them. A Kubernetes cluster has it's own dedicated cluster IP address. A cluster IP is an IP address that that the cluster uses for things like management and exposure to the internet. Typically when pods are restarted there IP addresses change. This leads to inconsistent communication with the pods. A cluster IP gives developers consistent communication with every aspect of Kubernetes.

2.3 API Server

An API is an Application Programming Interface. In Kubernetes the API server handles all of the user requests and services. From there, the API server communicates to kubelet and kube-proxy in each worker node to determine the load-balancing and scaling needed to complete the request efficiently and effectively. Figure 2 shows this communication as the master node contains the API server.

For example, Shoe Guru on any given day has 3 application pods in the nodes. Black Friday is coming up soon and there is going to be a dramatic increase in the number of users. Without Kubernetes, the developer would have to manually scale the number pods or applications and distribute requests to them. With Kubernetes this process is all automated. The developer first specifies the maximum CPU usage a pod can have, say 80%. Once CPU usage on a pod hits that threshold the API server than reaches out to a controller which adds more pods. Then the new pods are given user requests to complete. This returns the pods' CPU usage back to desired levels. After Black Friday the number of users return to normal but there are still the extra pods from the scaling. This causes an extremely low CPU usage and a surplus of pods which all would have to be paid for. Kubernetes will then scale the number of pods back to 3. This results in a similar CPU usage as before Black Friday.

2.4 CPUs and Millicores

Central Processing Unit or CPU measure the compute processing in a server or other type of machine. In Kubernetes they can be referred to as compute resources [2]. "Compute resources are measurable quantities that can be requested, allocated, and consumed." [2].

In Kubernetes CPU usage is measured in Cores, also known as Virtual CPUs (vCPU). Cores are similar to other units of measurement like inches or meters. The similarity is more in the sense of the prefix used to describe the amount. For example one thousandth of a meter is a millimeter, in the case of Kubernetes one thousandth of a core is a millicore. So if a pod is using 2000 millicores of CPU it is technically using 2 cores.

3 Horizontal Scaling

One of the most prominent features of Kubernetes is the ability to scale an application across multiple servers. This is done to take some of the workload off of our servers. Scaling in Kubernetes is called Kubernetes Horizontal Pod Autoscaling (KHPA). KHPA is used to increase or decrease the number of containers and pods based on the number of concurrent users. KHPA takes the target input as a percentage of CPU usage. The output is the target number of pods. What this means is KHPA takes in how much CPU usage a machine is using. It then gives the cluster the number of pods needed to easily run the workload.

Task	CPU usage Single Server (in millicores)	CPU usage Multiple Servers (in millicores)
1	576.00	209.00
2	526.00	369.00
3	498.00	333.00
Average	533.33	303.66

Table 1. CPU usage

Response Time(ms)	Single Server (no scalability)	Multiple Servers (with scalability)
Get data (first task)	00:00:44	00:01:07
Get data (second task)	00:00:40	00:01:06

Table 2. Response Time [3]

For example, in the Shoe Guru application on Black Friday. The developers specify that the input or the percent of CPU usage is 80 percent. The output of this algorithm is increasing the number of pods by 1 in order to handle the requests.

In a 2019 study, Dewi, et al [3] evaluates how Kubernetes' horizontal scaling increases the performance of servers. Dewi, et al focuses on increasing concurrent users that are accessing academic data. In accessing this data, users request 3 types of services: get data service, send data service, and delete data service. These services are used throughout the evaluations. There are also two types of scenarios that Dewi, et al evaluates in the study. The first is a single server, which does all of the request for it's scenario. Also there is a multiple server evaluation which includes three servers. One hosts the master node, where the other two host the worker nodes. This cluster uses Kubernetes horizontal scaling.

3.1 CPU usage

Table 1 compares CPU usage between a single server and multiple servers (using scalability). The CPU usage in this study is measured in millicores.

The scenario in this study uses a simulated behavior composed of 3 tasks. Each task includes 150 users making 11 requests each. The simulation included 1,650 user requests being handled by single server as well as multiple servers. The users continuously and simultaneously send 2 requests that get data, one request that deletes data, and seven requests that send data. Each task in this evaluation does these requests in a different order to ensure consistency.

3.2 Response Time

This scenario focuses on the response time to the user requests comparing a single server to multiple servers. The evaluation includes two tasks in which users only request get data services. Each task receives the same amount of get data requests. The study does multiple tasks to look for consistency and calculate an average. Table **??** compares the response time of 150 users making 11 requests each. This scenario compares these response times on a single server versus multiple servers using scalability. The scenario with multiple servers scales the containers when these requests come in. Scaling the containers comes with a delay which is included with the response time of the multiple servers cluster. As the number of request increase, the expectation is that this scaling delay begins to be less influential. This scaling is done once the CPU usage hits 80 percent. [3] Kubernetes does this so the containers do not become overwhelmed.

3.3 Results

The scenario evaluating CPU usage shows that on average, CPU usage with a single server is almost double the CPU usage with multiple servers meaning each server in the cluster is using half the amount of CPU compared to the single server. This is because the workload is being scaled across multiple different servers. As we see in Table 1 this takes a lot of the burden off of the servers. In the first task the difference of CPU usage is 367 millicores. [3] This is the largest difference of usage. In comparison, on average the difference is 230 millicores. [3]

Table 2 shows us that single server has faster response times when responding to 1,650 user requests. [3] With multiple servers there is a delay though. The delay is due to the master node scaling the containers of the worker node. As the number of requests increases the scaling time becomes less of a factor. [3]

These two aspects of scaling are being studied because it helps to make machines more efficient and effective. Response time and CPU usage are two issues that Kubernetes is trying to be made better. Scaling ties into this because it is supposed to make these two aspects more efficient and more effective. While the response time does not show how efficient Kubernetes is, the CPU usage shows the effectiveness of Kubernetes. The CPU usage for this single server is at 80% [3]. When adding more requests the scaling delay should become less influential. This is because the multiple servers cluster is doing much less work than the single server. With that being said, as the number of requests increase the single server will become more overwhelmed. Therefore it will take more time for the single server to respond to those requests.

4 Load-balancing

For load-balancing Kubernetes uses algorithms to decide the leader of the cluster. The leaders and followers in Kubernetes are an application pod within the different nodes. In each node there are multiple replicated applications as talked about previously. For load-balancing purposes, each pod that the replicated applications are in have 2 more containers that are used for this purpose. One container is used for

the leader election, meaning the sole responsibility of this container is to be involved in the leader election with other replica pods. The second container is used to handle all of the client requests being sent over the network. This container is crucial to understand what type of requests a pod can receive. If there is a read request both leaders and followers can serve it. Whereas a write request can only be handled by the leader pod. In the case of a write request, followers will direct all of the requests to the leaders of the nodes. Each of these pods save and retrieve the data requested by the user from the data store. For example the write requests are saved in only the leader pods data store because every write request is directed to the leader pods. Whereas read requests are retrieved from all of the pods data stores because all pods can handle read requests.

Nguyen and Kim [4] talk about 3 different algorithms that could be used in leader selection. While these three algorithms have different election processes, there are many similarities. Leaders send "heart-beats" to its followers continuously. This is to ensure each follower knows who the leader is. The election process typically begins when a follower doesn't receive the heart beat from the leader for a specific period of time. From there the pods try to become the leader. Each algorithm is different in the race to become the leader. At the end of the election process the pod that is now called the leader begins to send the heart beats out to it's follower.

4.1 CPU utilization

Nguyen and Kim compare the CPU utilization between a cluster with concentrated leaders and another cluster with balanced leaders [4]. They evaluate distributing 5 leader pods across 3 worker nodes. The concentrated leaders cluster in this study have all five leader pods in node 1, and nodes 2 and 3 in this cluster don't have any leaders. Whereas the balanced leaders cluster distributes the leader pods across the worker nodes, with two leaders in nodes 1 and 2 and one leader is in node three [4].

The scenario used here is constructed with 4 clients sending write requests for 150 seconds simultaneously to each cluster, requests are then distributed to the different worker nodes in each cluster [4]. The write requests used require the leaders of the worker nodes to do the majority of the work. The followers though, still have to direct those requests to the leaders of the nodes, which is why CPU utilization is not zero. In the concentrated leaders cluster the majority of the traffic is directed to node 1 because that is where all of the leaders are. In the balanced leaders cluster the traffic is distributed fairly evenly because the leaders are distributed to all 3 nodes.

Figure 3a shows the CPU utilization within the concentrated leaders cluster. Figure 3b shows again the CPU utilization within the balanced leaders cluster. Within the concentrated leaders cluster all of the leader pods are inside

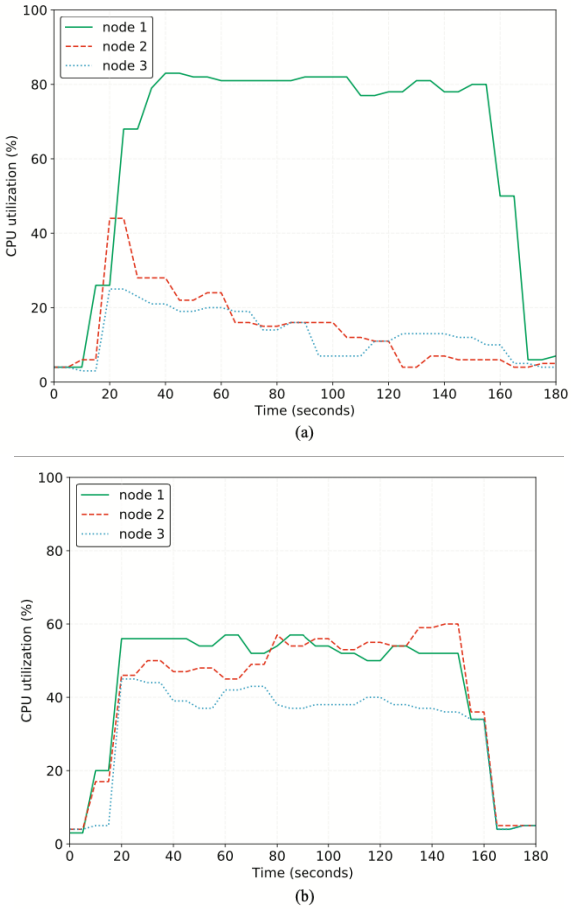

Figure 3. CPU utilization (a) Concentrated vs. (b) Balanced leaders [4]

of node 1. Because all of the requests are being handled by the leader applications, the CPU utilization of node 1 is 60 percent higher than the other two nodes [4]. This causes a bottleneck in the cluster. Node 1 is consuming much more resources than nodes 2 and 3 so it becomes overwhelmed.

On the flip side, Figure 3b shows the CPU utilization for the balanced leaders cluster. The figure shows how balanced the CPU utilization is across the nodes. For nodes 1 and 2 the CPU utilization is approximately 60 percent meaning the nodes are fairly balanced [4]. Node 3 only had one leader application so the utilization is less. With distributed leaders the evaluation shows how sharing the CPU usage with Kubernetes is effective. This helps increase the throughput of the systems, which is shown in the next section.

4.2 Throughput

The second evaluation from Nguyen and Kim focuses on their study [4] which is represented in Figure 4. This evaluation looks at the throughput (requests processed per second) comparing concentrated and balanced leaders clusters. For

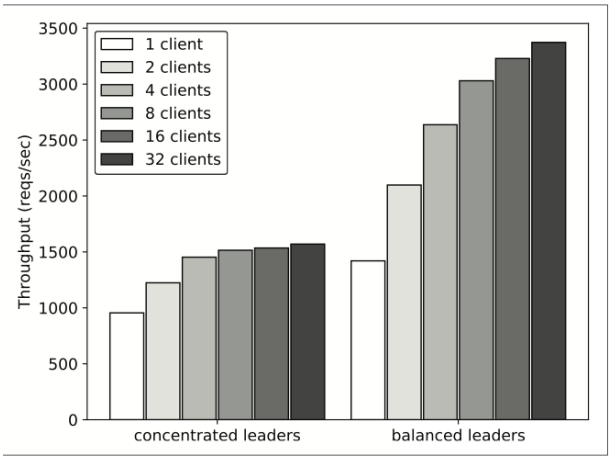

Figure 4. Throughput Concentrated vs. Balanced leaders [4]

this evaluation clients would simultaneously send write requests to the leaders. The number of clients increased from 1 to 32, where each client sent 2,000 requests to each leader [4]. Nguyen and Kim run this evaluation on the same clusters as the CPU utilization evaluation.

In the evaluation of the concentrated leaders cluster, once the number of clients reach 4, the number of requests processed per second begins to stay relatively the same. This is due to a bottleneck once the number of clients is greater than 4. The requests in this scenario all go to one node because all of the leaders are in that one node. Throughput then relies solely on the capacity of the single node.

In the balanced leaders cluster the evaluation shows that as the number of clients increase, throughput increases more consistently. In this scenario the requests are distributed across all nodes because there are leaders in all three nodes, so the throughput relies on capacity of three nodes instead of just one. Distributed workload allows for higher productivity because there is a lower chance of a bottleneck [4]. Meaning the chance of a bottle neck happens when the load gets even bigger than 32 clients, instead of after 4 clients. The study shows that throughput is increased in the scenario of balanced leader, illustrating the importance of distributing leaders throughout nodes instead of having one node with all of the leaders.

4.3 Results

This study by Nguyen and Kim demonstrate a direct correlation between CPU utilization and throughput. On one hand, the lopsided CPU utilization creates a bottleneck because of the workload imbalance, which in turn leads to decreased performance shown in the throughput evaluation. In this evaluation once the number of clients gets to 4 the request per second stays fairly constant. With all of the requests going to only one node, the cluster cannot keep taking more requests because the node is almost at its maximum CPU utilization.

In the case of the balanced leaders, the distribution of the workload creates a much better effect. Because the leaders are distributed fairly evenly between the three nodes, the CPU utilization of the nodes are similar. The lower and more even CPU utilization allows the cluster to have improved performance, which is demonstrated in throughput evaluation. As the number of clients increase the throughput increases at fairly large, positive rate. The throughput after receiving requests from one client is 48.7 percent higher with balanced leaders compared to with concentrated leaders [4]. This shows that without increasing the number of clients, the throughput is still higher in the balanced leaders cluster.

These results demonstrate the importance of load-balancing. When only one node is doing all of the work, the cluster cannot reach improved performance. Load-balancing allows nodes and cluster to have the best performance. With the different leader selection algorithms and the configuration of distribution, Kubernetes helps clusters improve performance through load-balancing.

5 Conclusion

Kubernetes is becoming a widely used containerized application tool. It is being used for the automation of many aspects of management and deployment. These are two aspects that engineers have spent a lot of time on, and Kubernetes helps do that automatically. This paper focuses more on the management aspects of Kubernetes, including features such as horizontal scaling and load-balancing which are used to increase performance. Scaling pods horizontally focuses more on elasticity, or how much workload can an application handle before becoming overwhelmed. Horizontal scaling comes from Kubernetes horizontal pod auto scaling (KHPA). This algorithm automates the horizontal scaling process and helps to create elasticity. Whereas load-balancing focuses on distributing network traffic to the different replica application pods created. This comes from leader selection algorithms which elects a leader to control the traffic. Distributing this traffic leads to increased throughput and decreased CPU utilization.

Acknowledgments

I would like to thank Nic McPhee and Elena Machkasova for the support and feedback throughout this process. As well as my classmates for the feedback they have provided.

References

[1] Docker Contributors. [n.d.]. *Use containers to Build, Share and Run your applications.* https://www.docker.com/resources/what-container

[2] Kubernetes Contributors. [n.d.]. *Managing Resources for Containers.* https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/

[3] Lily Puspa Dewi, Agustinus Noertjahyana, Henry Novianus Palit, and Kezia Yedutun. 2019. Server Scalability Using Kubernetes. In *2019 4th Technology Innovation Management and Engineering Science International Conference (TIMES-iCON)*. 1–4. https://doi.org/10.1109/TIMES-iCON47539.2019.9024501

[4] Nguyen Nguyen and Taehong Kim. 2020. Toward Highly Scalable Load Balancing in Kubernetes Clusters. *IEEE Communications Magazine* 58, 7 (2020), 78–83. https://doi.org/10.1109/MCOM.001.1900660

[5] Wikipedia contributors. 2021. Kubernetes — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Kubernetes&oldid=1049396434. [Online; accessed 12-October-2021].