# Analyzing QoS Performance in Kubernetes-Based High Scalability Clusters

1st Fathur Alfarisi Julana
*School of Electrical Engineering*
*Telkom University*
Bandung, Indonesia
fathuralfa@student.telkomuniversity.ac.id

2nd Istikmal
*School of Electrical Engineering*
*Telkom University*
Bandung, Indonesia
istikmal@telkomuniversity.ac.id

3rd Arif Indra Irawan
*School of Electrical Engineering*
*Telkom University*
Bandung, Indonesia
arifindrairawan@telkomuniversity.ac.id

**Abstract— This paper proposes a performance analysis of the auotscaling methods that can be utilized in Kubernetes Clusters in order to determine cost-efficiency for services. This paper aims to accommodate the specifications for an analysis by using Google Kubernetes Engine (GKE) built into Google Cloud Platform (GCP) as a platform to orchestrate clusters alongside deployed services. To fulfill the cluster model, customized cluster settings and NGINX-based services are developed to manage the website used for load testing in order to determine the behavior of the autoscaling methods. This paper proposes an experiment using real-time transmission of Virtual Users (VU) sent from an application, called k6, used to simulate a number of users accessing the website within a time frame through a public network. During the experiment, the quality of service of the website is measured and analyzed using metrics and events from GKE and the results made by k6 to determine whether the autoscaling system works within set parameters and determine cost efficiency in Kubernetes cluster. The proposed load test is able to ensure that the number of HTTP requests results in HTTP Status Code 200 and that 95% of the requests are completed in less than 2 seconds.**

## I. INTRODUCTION

Virtualization has recently emerged as a crucial cloud computing technique. In particular, container-based virtualization is a quick way to build a virtual environment that operates on the host machine's software level [1]. Due to its minimal resource utilization and excellent mobility, it has been expanding quickly with rotating virtual machines (VMs). Additionally, the methodologies of application design are being revolutionized by container-based microservices. The container-based micro-services enable an application to be made of several lightweight containers over a large number of nodes as opposed to the conventional single monolithic architecture wherein all components are merged into a single unit [2]. Multiple containers are scattered throughout the network in large-scale systems, necessitating the use of an orchestration tool to deploy the containers and manage resources. Microservices are small, self-contained modules of an application that are linked together to function as a whole. Along with the microservices design, enterprises must have stable application releases and shorten the time from development to deployment. This is enabled through the use of continuous deployment procedures [3]. Software containers and orchestration tools, such as Kubernetes, have made microservice deployment and maintenance easier. Kubernetes is one of many open-source platforms used in the managing of containerized services, including setup and deployment. The Kubernetes platform is also used for running highly available distributed systems. They make up what is known as Kubernetes Clusters, where clusters contain a master node and worker nodes. Each worker node hosts Pods that make up the workload of an application. And each worker in every pod is managed by the control plane, where commands are given by scripts to be distributed among the cluster for deployment. Because of the multitude in pods containing workers in clusters, this allows for a high availability service. To achieve high availability, Kubernetes runs clones of various containers, much like virtual machines. This allows for the service to avoid numerous failures should in case numerous apps are running in one single container. To maximize the service's scalability and availability, Kubernetes load balancing entails dividing network traffic among pod replicas in accordance with load balancing algorithms [4].

This paper proposes a performance analysis of the autoscaling methods used in Kubernetes Cluster to determine cost-efficiency by simulating Virtual Users (VUs) being load tested onto a website that is deployed inside the cluster using an application called k6 developed by Grafana. However, this paper is limited by several aspects, which are:

1. Clusters are deployed in a Kubernetes environment hostedusing Google Cloud Platform.
2. The QoS data is taken using k6 and the built-in metricsby Google Cloud Platform.
3. The machine used to host the cluster is limited to 4GBRAM and 2 vCPU.

## II. THEORITICAL REVIEW

### A. Kubernetes

Early on, applications would be run on a physical server causing resource allocation issues. One of these issues is where one application would take up most of the resources and as a result, the other applications would either underperform or not function. A solution for this would be to run each application on a different physical server. But this did not scale as resources were underutilized, and it was expensive for

organizations to maintain many physical servers. A different solution was introduced and that is virtualization. It allows the user to run multiple VMs on a single physical server's CPU. Virtualization allows applications to be isolated between VMs and provides a level of security as the information of one application cannot be freely accessed by another application. Virtualization allows better utilization of resources in a physical server and allows better scalability because an application can be added or updated easily, reduces hardware costs, and much more. With virtualization you can present a set of physical resources as a cluster of disposable virtual machines.

Containers are similar to VMs, but they have relaxed isolation properties to share the OS among the applications. Containers have become more popular due to their extra benefits such as:

1. Continuous development, integration, and deployment: provides for reliable and frequent container image build and deployment with quick and easy rollbacks.
2. Environmental consistency across development, testing, and production.
3. Resource isolation: predictable application performance.
4. Resource utilization: high efficiency and density.

Containers also lower management costs. Because they share the same operating system, only one operating system requires attention and feeding for bug fixes, patches, and so on. This notion is similar to that of hypervisor hosts: fewer administration points but a slightly larger fault domain. In short, containers are lighter and more portable than virtual machines. Conclusion The fundamental distinction between virtual machines and containers is that containers provide a means to virtualize an operating system so that numerous workloads can execute on a single OS instance. The hardware is virtualized with VMs in order to run multiple OS instances. Containers, with their speed, agility, and portability, are another tool for helping to expedite software development [5].

Multiple applications can now share the same underlying operating system. This feature makes containers much more efficient than full-blown VMs. They are portable across clouds, different devices, and almost any OS distribution.

### *B. QoS Parameters*

QoS is defined as a process or approach for determining whether a service or system in a network can achieve its goal based on its service characteristics. To meet the QoS requirements for the Kubernetes Cluster platform, load tests, measurements, and events must be observed in real-time. Several evaluations, such as network QoS and resource use, could be utilized to determine whether the Kubernetes Cluster meets the described QoS.

*1. Network QoS Parameters:* The network QoS parameters consists of delay, jitter, throughput, and packet loss that is measured throughout the test. By use of k6, the parameters that are going to be measured is the number of HTTP requests that respond with a code 200, interrupted iterations and the size of the load that was sent. HTTP Code 200 is a status response indicating that the request has been successfully fetched and transmitted. When the test duration is achieved or when scaling VUs with the stages option, k6 would interrupt any ongoing iterations leading to some errors in the metrics. [6]

*2. Resource Utilization Parameters:* The resource utilization analysis uses CPU usage and memory usage data to determine whether an application is using the available computing and memory resources efficiently. The measure of resource utilization is the percentage of time that a resource is being used compared to the time the resource is available. When a resource becomes overused or highly used compared to other resources, that resource becomes critical. A critical resource indicates an inefficient design of the system or the system is having a peak load.

Various processes use the CPU to perform calculations and execute certain instructions. The more calculations and instructions that the CPU executes in a time period, the higher the throughput of the CPU. Thus, high CPU utilization does not always indicate a performance problem when the CPU throughput is also high. However, when the CPU utilization is high but the throughput is low, there is a process that uses the CPU inefficiently.

Memory usage is determined by the activity and usage of small components that make up the memory called pages. Processes use these pages by locating unused pages and store them with temporary data. A page scan is a process where a CPU cycle is used to select unused pages for other processes. A high rate of page scans indicates that poor memory utilization is becoming a problem.

## III. METHOD

According to [7], the framework for a proper Kubernetes cluster, it consists of three parts:

1. A Kubernetes cluster is made up of a collection of machines known as nodes that run containerized apps. Each cluster contains at least one node with a pod.
2. The nodes host resources needed by the Pods which uses the app that make up the application workload.
3. The control plane manages the cluster's nodes and Pods.

In the proposed model of the Kubernetes Cluster, this paper uses a Google Cloud Platform virtual machine (VM) provided by Google to act as the public cloud server to make Kubernetes management more efficient using Google Kubernetes Engine to host the cluster using the machine from a selection that is also provided.

As shown in Figure 2, the connection between the client, the NGINX website, and the GCP is available through the internet. The client accesses the website through the services provided by the deployment that the admin has setup for the NGINX application which directly connects to the web server. The web server is contained in replica pods located inside the nodes to improve efficiency in the cost of CPU and memory usage that is allocatable.

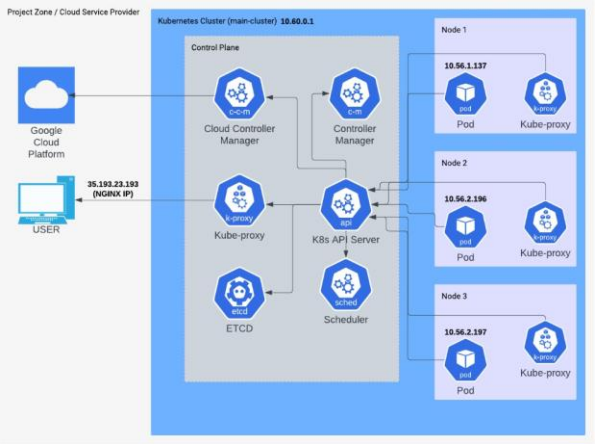


FIG. 1
THE PROPOSED SYSTEM DESIGN OF THE KUBERNETES CLUSTER DESIGN.

## IV. SYSTEM SPECIFICATIONS

This paper uses Google Cloud Platform cloud services to provide a virtual private server (VPS) designated as the server where the proposed IoT platform is installed. The VPS uses a default Linux node image version 1.23.8-gke.1900 as the kernel of the virtual machine named e2 medium with 4 GB of RAM, 100 GB of storage of a standard persistent disk.

## V. EXPERIMENT SETUP

To fulfill the performance analysis for Kubernetes Cluster by load testing the NGINX website, the transmitted VUs by k6 is set in real-time. As such, the analyzed network parameters from the simulation are the total number of HTTP requests and the events related to the autoscaling methods that is taken into effect once the load testing begins. Finally, the analyzed hardware parameters from the experiment are the CPU and memory usage from the cluster of each test.

The load testing experiment is done four times to notice the effect of the autoscaling methods to the cluster made by the packets received by the Kubernetes cluster. Each trial has different simulated VUs and time frames to simulate different scenarios. After all trials are done, the measured parameters are analyzed and compared with each other according to the autoscaling method. The analysis results are then concluded to finish this paper.

## VI. EXPERIMENT ANALYSIS

### A. Network QoS Analysis

This paper uses the methods described in Chapter 3 to determine whether the proposed Kubernetes Cluster can be of use to a web service. Fig. 2 shows the number of VUs that is simulated within a time frame of 13 minutes, included with the thresholds to define the passing criteria for the load testing. Although the capability of the simulated cluster is not ideal due to machine limitations, the situation may serve as an approximation for a real working condition when the machines that the Kubernetes Cluster is hosted on and the autoscaling

```
import http from 'k6/http';
import { check, sleep } from 'k6';
import { Trend } from 'k6/metrics';

const myTrend = new Trend('waiting_time');

export const options = {
  stages: [
    { duration: '2m', target: 450 }, // simulate ramp-up of traffic from 1 to 450 users over 2 minutes
    { duration: '5m', target: 700 }, // ramp-up and stay at 700 users for 5 minutes
    { duration: '5m', target: 1000 }, // ramp-up to 1000 users for 5 minutes
    { duration: '2m', target: 600 }, // ramp-down to 600 users for 2 minutes
    { duration: '1m', target: 100 }, //ramp-down to 100 users before shutting down
  ],
  thresholds: {
    http_req_failed: ['rate<0.02'], // http errors should be less than 2%
    http_req_duration: ['p(95)<2000'], // 95% requests should be below 2s
  },
};

export default function () {
  const res = http.get('http://35.193.23.193/index.html');
  check(res, { 'status was 200': (r) => r.status == 200 });
  sleep(1);
}
```

FIG. 2
K6 SCRIPT CODE FOR VPA TEST 2.

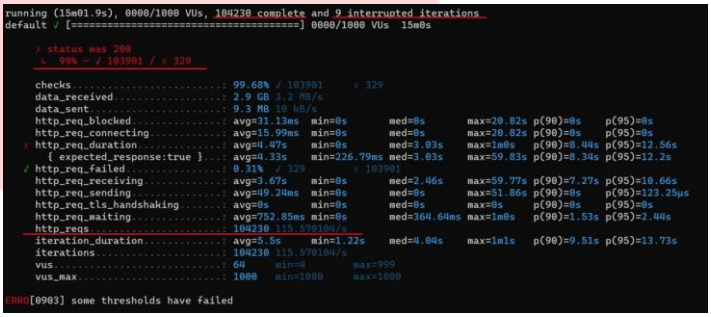


FIG. 3
THE RESULTS FROM A TRIAL MADE TOWARDS THE KUBERNETES CLUSTER DENOTING SOME IMPORTANT PARAMETERS TO LOOK FOR.

methods are implemented correctly. The QoS is calculated by analyzing the results of HTTP packets made by k6 and by reviewing the autoscaling events made by GKE.

A total of 4 trials, 2 CA trials and 2 VPA trials, are conducted to account for different scenarios to maintain fairness. Each trial sends a stream of GET requests in the span of varying time frame with 2 threshold where one of the thresholds specifies that 95% of the requests happen within 2000 milliseconds and that the other threshold evaluates the rate of HTTP errors. Each trial is shown to have a different effect on the cluster metrics. As shown in Fig. 3, the checkmark signifies a successful threshold while the cross is for a failed response.

### B. Resource Utilization Analysis

The resource utilization analysis describes how effective are the CPU and memory usage of the VM when the IoT platform receives streams of images. The CPU usage is measured during the image transmission experiment with the result as shown in Fig. 4. All trials are conducted while the CPU usage of the system was initially at around 0.2% and the CPU usage increases until 1.75% on average. After 120 seconds, the CPU usage drastically decreases to the initial value of just above 0.2% across all trials. The CPU usage is not affected significantly because the database only creates new documents for each image to be stored in. Since the writing process does not require scanning the whole collection of documents, the

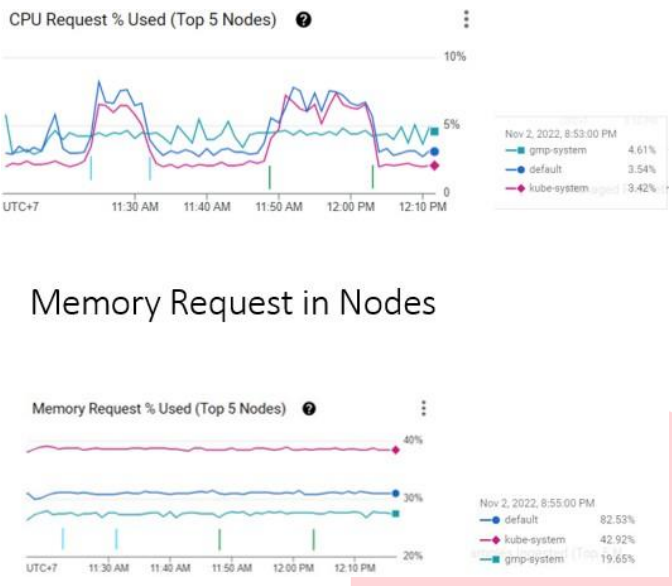


FIG. 4
THE CPU AND MEMORY USAGE DURING K6 LOAD TESTING FOR THE CA TRIALS.


FIG. 5
THE CPU AND MEMORY USAGE PERCENTAGE DURING K6 LOAD TESTING FORTHE VPA TRIALS.

requests to make new documents are handled without using the CPU too much.

Meanwhile, Fig. 4 shows that the CPU is affected more significantly than memory usage during the trial for Cluster Autoscaling. Each trial has an average memory usage increase by. The increases in CPU usage between trials are due to additional CPU used by the pods for processing the requests towards the web server.

### *C. K6 Performance Analysis*

This paper uses k6 developed by Grafana to simulate and record load testing performances. Load testing performances are indicated by the rate of HTTP requests and the number of packets dropped due to thresholds made for the script. The Fig. 3 shows that there are HTTP requests that meet the criteria set in Fig. 2for as thresholds for errors and duration of a request. By having a large number of VUs, the performance of a web server will be affected drastically which requires the Kubernetes Cluster to scale according to the workload and available allocatable resources a which enables the Kubernetes Cluster to lessen the impact of heavy traffic.

## VII. CONCLUSION

The working process of this paper is concluded in this section.

A. This paper proposed a Kubernetes cluster as a platform for a web service. The role of the web server is handled by NGINX. This paper rented a trial version of Google Cloud Platform service to host the Kubernetes cluster while also allowing for the cluster to be able to be accessed publicly. The QoS parameters as well as the metrics performance from GKE show that the with enough resources, the cluster can meet the requirements for managing applications and scaling them accordingto need.

B. The Vertical Pod Autoscaler set resource requests automatically based on usage while CA is triggered by pending pods inside the node. In order to maintain stability, higher cost is put in place in terms of a better machine, higher memory and CPU capacity to avoidany downtime but to ensure efficiency, there is a risk of downtime due to the allocation of resources being lower to save costs.

C. Should a cluster or application request more than needed, slack increases, adding extra costs to the service.

D. The cluster is intended to scale up to account for the extra workload due to a higher number of VUs in the given time frame on CA Test 2. However, because the system is limited in memory and CPU due to the machine that was used from GCP, scaling up and establishing new nodes was not viable. As a result, there were more failures and interrupted iterations in CA Test 2 than in CA Test 1.

## VIII. SUGGESTION

These suggestions are given to help the readers acknowledge what this paper lacks so improvements can be made for future research.

***A.*** Due to trial use of Google Cloud Platform, only a limited amount of CPU and memory was allowed. Upgrading machines that allows for more RAM and CPU allows for more room in scalability and stability.

***B.*** Since CA is triggered by pending pods, and sends a request to scale up the cluster, the time it takes to create a node could take several minutes causing delay to the application performance thus degrading any services.

## REFERENCES

[1] Kubernetes. (2022-11-24) Virtualization. [Online]. Available: https: //kubernetes.io/docs/concepts/overview/

[2] IBM. Containerization. [Online]. Available: https://www.ibm.com/cloud/ learn/containerization

[3] H. Rajavaram, V. Rajula, and B. Thangaraju, *Automation of Microservices Application Deployment Made Easy By Rundeck and Kubernetes*, 2019- 07-26.

[4] Y. Pribadi, A. B. PN, and M. A. Irwansyah, "Analysis of the use of the failover clustering method to achieve high availability on a web server (case study: Informatics department building)," 2020-04-02.
[5] D. Jones. (2018-03-16) Containers vs. virtual machines (vms): What's the difference? [Online]. Available: https://www.netapp.com/ blog/containers-vs-vms/
[6] k6. Metrics. [Online]. Available: https://k6.io/docs/using-k6/metrics/
[7] Kubernetes. (2022-10-24) Kubernetes components. [Online]. Available: https://kubernetes.io/docs/concepts/overview/components/