CHAPTER I
INTRODUCTION

1.1. Background

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 creation and maintenance easier. To manage the horizontal elasticity of containers, Kubernetes employs a decentralized threshold-based policy that necessitates setting thresholds on system-oriented metrics (i.e., CPU utilization). This may not be suitable for scaling latency sensitive applications that must express needs in terms of reaction time [4].

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 [5].

To maximize the service's scalability and availability, Kubernetes load balancing entails dividing network traffic among pod replicas in accordance with load balancing algorithms [6]. In other words, one of the replicas responds to user queries, and users are not required to be aware of the internal procedure. A stateful application must provide consistency across these distributed data stores because each replica in Kubernetes can have its own data store. Several distributed systems use a leader-based consistency maintenance approach, where an elected leader is entirely in charge of updating data and subsequently replicating it to the followers, to address this consistency challenge. The container automation process, using Kubernetes technology, can be implemented based on the number of concurrent users. With a few additional options, the scalability mechanism of this container may be applied to Kubernetes. Scalability is supposed to improve performance and server response time to users without diminishing server utility capabilities [7].

Since problems can occur when the webserver fails while being accessed for data and services hosted by servers, steps are taken to ensure server is running in a healthy manner. An example would be of a university website being accessed by a large number of users applying or submitting a form at the same time is an example of this; if the server cannot accommodate a large number of users, the service may degrade leading to a bottleneck and users may experience some latency when viewing the website. To compensate, Kubernetes can be used to scale the clusters hosting the microservices in order to obtain the optimal mix of performance results. This affects either the pods used by the container in a vertical fashion that affects the pod or the entire cluster by influencing the nodes. When there is a tremendous load, as previously described by the large number of users accessing, Kubernetes will scale up to match the load by raising the amount of CPU and memory

constraints, and then scale back down after the load has lessened to save costs and efficiency.

Kubernetes can manage cloud applications and microservices' resource allocation and traffic. Kubernetes excels when the applications consist of multiple services running in different containers. This may be more than sufficient for a monolithic program with a static user base. Because of containers, developers are divided into teams that must focus on one service at a time. This assured that these teams become specialists in their respective fields. When there is a problem, they can fix it quickly and without disrupting other service areas. The speed, quality, and manageability of new development are immediately improved [8].

1.2. Problem Formulation

Setting up a cluster with policies for autoscaling nodes and pods as well as finding parameters that deems the tested result to be positive.

- The server not being able to handle a huge number of users resulting in a bottleneck.
- Ensuring Kubernetes cost-efficiency across many potential clusters.

1.3. Objectives and Benefits

The objective and benefit of this thesis is to determine and analyze the performance of a Kubernetes cluster. By conducting this simulation, network and resource optimization can be achieved to determine if Kubernetes is cost effective for use in a multitude of services that can be deployed and managed using Kubernetes.

1.4. Scope and Limitations

This thesis sets some scope and limitations:

1. Clusters are deployed in a Kubernetes environment hosted by an E2 machine, which has a 2vCPU, 4GB memory, and a 100 GB persistent disk provided by Google Cloud Platform.
2. A predetermined number of virtual users access the test website hosted by NGINX using a load testing program within a time frame to test the data packet type to be utilized during simulation trials.
3. The cluster will be tested with different autoscaling methods which are Vertical Pod Autoscaling and Cluster Autoscaling.

1.5. Research Methodology

This thesis considers the following methods:

1. Researching into Kubernetes for designing a type of clusters that can be used by a high number of users for services.
2. Research regarding Google Kubernetes Engine and methods of implementation inside a cluster network to understand Kubernetes.
3. Set virtual users parameters and demands in GKE Clusters.
4. Simulate different cluster parameters to deduce cost efficiency.