RELIABLE PERFORMANCE METRICS FOR MICROSERVICE-BASED APPLICATION

This dissertation submitted in partial fulfilment of the requirements for the

Degree of MSc in Computer Science specialising in Cloud Computing

G.W.G.S.V Amaradasa

179302N

Department of Computer Science and Engineering

University of Moratuwa

Sri Lanka

March 2022

DECLARATION

I declare that this is my own work and this MSc Thesis Project Report does not incorporate without acknowledgement any material previously submitted for a Degree or Diploma in any other University or institute of higher learning, and to the best of my knowledge and belief, it does not contain any material previously published or written by another person except where the acknowledgement is made in the text.

Also, I hereby grant to the University of Moratuwa the non-exclusive right to reproduce and distribute my thesis, in whole or in part in print, electronic or other media. I retain the right to use this content in whole or part in future works.

UOM Verified Signature 31-03-2021

G.W.G.S.V. Amaradasa Date

The above candidate has carried out research for the Masters under my supervision

Name of the supervisor :

---------- ----------

Signature of the supervisor Date

Name of the supervisor : L C T Silva

UOM Verified Signature 31-03-2021

Signature of the supervisor (External) Date

ABSTRACT

Microservice is the most popular technology nowadays. But unfortunately, performance testing with microservice-based applications was comparatively more challenging than testing. As a result, many studies have focused on the performance testing bottleneck in the Microservice-based application. Furthermore, performance attributes were challenging to measure due to the unique features inherited from microservices, such as auto load balancing auto-provisioning elements.

The research aims to identify the best way to reduce the variation in performance test execution and propose more stable performance metrics that stand more reliable in microservice-based applications.
Performance tests execution uses different load patterns to identify the best way to reduce variations in a different implementation. Stress, endurance, and load tests were used primarily as the load patterns.

Key metrics were identified beforehand and evaluated to determine the most reliable metrics.
The thesis refers to the suggested future works of the existing research. In addition, refer the same analysis to the detailed studies on the performance characteristics of microservice applications.
The research's scope is to identify the possibility of measuring the microservice-based applications' performance and finding the best metrics with each load pattern.

The Benchmark application was TeaStore for the proposed evaluations. Further, the Sock Shop application illustrates the performance aspects comparatively against the TeaStore applications.
The proposed evaluation method used the TeaStore and Sock shop applications deployed on Google Kubernetes Engine and JMeter Scripts.

The analysis evaluates metrics collected, graphs, and other statistics in JMeter and Google Kubernetes dashboard.

ACKNOWLEDGEMENTS

I would like to express my great appreciation to my research supervisor Dr. Adeesha Wijayasiri for his guidance, enthusiastic encouragement, and valuable critiques on this research work. His valuable insight and expertise greatly assisted the research.

I am also grateful to Dr. Dilum Bandara, Dr. Gihan Dias, and Dr. Indika Perera, Dr Sanath Jayasena who initially guided me to start the proper research. Mrs. Chamadenri Silva and Mr. Suren Rodrigo supported me with all the aspects and necessary guidance. I express my sincere thanks to them for their help.

Last but not least, my sincere gratitude to my family for their continuous support and sacrifices. I would like to extend gratitude to my loving husband and mother- in -law, for their love toward my studies by caring for my little son, and my mother's and father's well wishes are with me in whatever I achieve.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

Abbreviation	Description
SOA	Service Oriented Architecture
AUT	Application Under Test
GKE	Google Kubernetes Engine
VM	Virtual Machine
JS	Java Scripts
DOM	Document Object Model
DB	Data Base
HIPAA	Health Insurance Portability and Accountability Act
YAML	A recursive acronym for "YAML Aren’t Markup Language."
PCIDSS	Payment Card Industry Data Security Standard
RAM	Random-Access Memory
WAR	Web Archive
API	Application Programming Interface
URL	A Uniform Resource Locator
IP	Internet Protocol
IOPS	Input/ Output Operations Per Second
HDD	Hard Disk Drive
SSD	Solid-State Drive
CSS	Cascading Style Sheets
CPU	Central Processing Unit

1. INTRODUCTION

Microservice-based application is a promising technology for large-scale software applications. Therefore, performance testing in a microservice-based application is crucial for a successful product launch in a competitive market. Various methods are available to measure the performance of a software application. The metrics are an essential key component in defining the application's performance in any manner. However, the technique used to collect performance metrics varies by application architecture.

Applications that are distributed on large scale use different software architectures such as Service Oriented Architecture (SOA) and Monolithic Architecture. Microservice-based architecture has become a trend for widely distributed architecture, in recent times.

Client-server interaction is the key focus area for performance testing tasks. In each architecture, the way the client-server interactions are conducted also differs. Carrying out a performance test in a cloud-based application is challenging. The microservice architecture is problematic when measuring performance-related issues since it has a lot of interaction inside the server. However, it is crucial to reduce resource usage costs.

Performance testing tools act as an agent to generate the load on the server and collect information related to performance metrics. Graphical representation and text-based metrics data reduce the effort and manual intervention of managing the performance test execution. As a result, those tools are handy for traditional client-server interactions. Besides, the statistics provided by the performance testing tools, application server-related metrics and database server-related metrics have to be collected to get a better insight into performance behaviours.

The approach exactly serves applications that are distributed in large scale although problems occur with microservice-based applications. Microservices [20, 21, 22] are the most promising architecture for developing systems distributed on a large scale[1].

The structure design principle helps to represent domain-based regional settings and loose coupling [19] paving way for the full use of containerization and self-recovering[17] technologies.

Microservices architecture complies with modern software engineering practices, including

continuous delivery, agile development methods, and DevOps.

1.1. Product Quality of Microservice-Based Application

Product quality is an essential part of the application. Functional and non -functional aspects are the main criteria that ensure product quality.
The application's performance plays a significant role in non-functional testing. Product quality is essential in any application to gain a competitive advantage. Regardless of the application architecture and the operative workflows, it is key to maintaining the product quality to achieve the application's competitive advantage in the market.
Therefore, performance testing plays a crucial role in maintaining product quality. Performance testing challenges in the microservice-based applications led to research different aspects to identify the bottlenecks. This thesis aims to identify the most reliable metrics and reduce variance in each execution.

1.2. The Problem Statement

Performance testing is one of the basic tests that is required by any software. Performance testing techniques assess the performance of the system. There are several methods to evaluate the performance of the system from various perspectives.
Generally, Testing AUT performance is considered a challenging task. The performance test results are dynamic and each test cycle can create a different outcome. Many facts are involved in the performance test results. They are application architecture, environment, technology, and network bandwidth.
Even with the same setup, there can be slight variations in each test execution cycle. Hence performance test results should have multiple samples to conclude the test in a traditional way itself.

Measuring performance metrics in microservice-based applications is more challenging than doing

the same in traditional systems. The technology and architecture used in microservice-based applications and how the application is organised are different from other systems.

Performance testers have to experience unavoidable and unchangeable inline pros and cons in microservice architectures. Some of them may complement the performance testing practices, while others make the testers' lives miserable.

This thesis refers to the related research paper [1] in which the researchers have concluded that there is potential research on identifying the more reliable performance metrics for the microservice-based architecture.

Researchers have proven results with a specific experimental setup. The same experimental setup is used in this thesis to identify the most suitable performance metrics for microservice-based architecture. The current research result will be the right candidate for the experiment's baseline values and it can be used to conclude the reliable metrics.

The experimental setup consists of the TeaStore application, a reference microservice application, used to demonstrate the test challenges, to identify the most reliable metrics, and to compare with other available metrics.

This paper evaluates the main research question and discusses the other issues outlined in the reference research.

The existing research has proven the followings,

01. Execution environments of the microservices are not reliable enough during the experiment; Researchers kept the same number of the total provisioned instances and did not observe any improvement

02. Although there is no major deviation in the CPU usage, the response significant degradation is shown in the response time. Thus, it is not easy to analyse the performance test results and the negative impact of the end-user experiences

03. Performance regressions are possible with microservices applications. However, we need to make sure that enough iterations are executed to identify the different services and the environmental behaviours.

In summary, conducting performance testing in the microservice-based application is more challenging than running it in typical systems. The application's performance is the most crucial aspect of maintaining the application quality.

The problem addressed is related to reduce the bottlenecks available in microservice-based application performance testing.

1.3. Objectives

The main questions that are going to be addressed in this thesis are as follows:

1. How to identify the most reliable performance testing setup.
2. The way to define the most suitable test execution strategy for the performance test execution.
3. How to identify the most relevant metrics to measure.

Here, we shall discuss the best way to address the additional issues inherited with the performance testing of the microservice-based applications.

1.4. Outline

Chapter 01

The introduction chapter consists of the research background and the problem statement addressed through the research. Finally, it describes the motivating factors and objectives of the study.

Chapter 02

The second chapter consists of details of analysis in similar researches, methodologies, and metrics.

Chapter 03

The third chapter presents the optimal approach selected at the end of the literature review. It also describes the steps that were followed during the process.

Chapter 04

The fourth chapter discusses the implementation steps with in-depth technical details applied during the evaluation. It consists of the deployment process, scripts and methods of gathering performance metrics, and performance statistics including analysis and graphs.

Chapter 05

The fifth chapter comprises the evaluation results gathered in relation to each performance testing pattern, sample and evaluation summary for test metrics to identify the most reliable metrics.

Chapter 06

The sixth chapter includes the research contribution, limitations, and future work.

2. LITERATURE REVIEW

2.1. The Experimental Setup

The test environment configurations help in conducting different variances of the test. As stated in the previous chapter, primary research [1] concluded bottlenecks related to microservice-based application performance testing. Therefore, it essentially focused on using a similar test environment configuration as stated in the preliminary experiments.

The experiment setup consists of two microservice-based applications deployed on the Google Kubernetes Engine cluster.

The first application is the TeaStore application which is a microservice reference application[2].

The performance test workflow of the TeaStore is,

- Load the Home page
- Users Signing in
- Click on sections and listed items

- Put the item into a shopping cart
- Signing out

The second application is the Sock Shop application[52]. API endpoint exists to generate the load. The endpoints used for the performance test are,

- index.html
- login
- Top bar
- navbar
- footer
- navigation
- catalogue
- cart
- category
- tags

The experiments consist of different performance testing types, mainly load tests, endurance tests, and stress tests. Each test has a unique objective. However, both applications generate a similar load regarding the number of hits/samples to observe the behaviour.

2.2. TeaStore Application

TeaStore is a microservices-based application used to evaluate and benchmark similar research. TeaStore comprises five unique services, each emphasising unique performance characteristics and issues[2]. Because of these diverse performance qualities and their dispersed landscape, TeaStore can be utilised as an application for testing and assessing software performance models and model extraction methods [2]. The application is intended to be scalable and supportive for both local and dispersed deployments.

These features are beneficial in terms of benchmarking performance metrics. TeaStore is a proven application to evaluate performance metrics. Therefore, it is an excellent candidate for this research and it is more reliable than the other in-house applications developed for testing purposes.

2.3. TeaStore Application Architecture

The TeaStore is an online shop for tea and tea-related utilities. The items are sorted into categories. The shop supports a synopsis of online shopping products, including preview pictures for each category with a configurable number of products per page. All pages of TeaStore exhibit an overview header bar, and it includesthe category menu and page footer. The main content indicates the selected category items, including abbreviated product information along with the preview image. Based on the number of products displayed per page, the consumer gets the option to cycle through multiple pages of this category view [2].

Figure 2-1: Shows the TeaStore architecture.

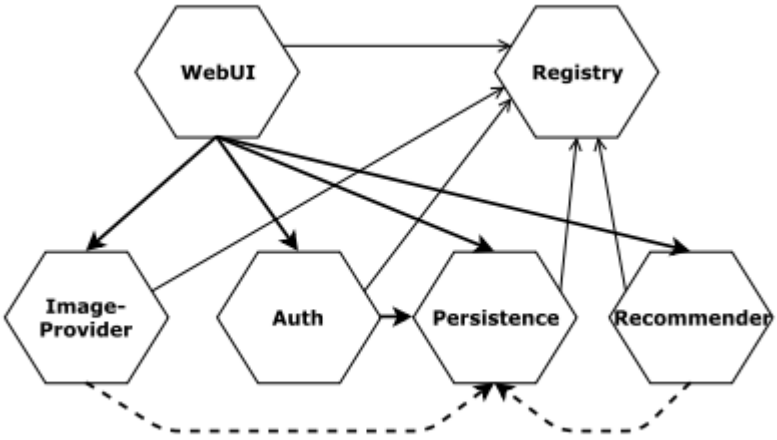

Figure 2-1: TeaStore Application Architecture, Source[2]

2.4 TeaStore Setup for Experimental Execution

As shown in Figure 2-1, TeaStore comprises five services and a service registry. The registry is not considered a part of the application under test. Therefore, it deploys to the cluster comprising the Kubernetes control plane. Each service uses a node in the cluster.
For the experimental implementation, TeaStore deploys single instances of each of the five services. Kubernetes utilises two parameters to control the resource utilisation of containers, the first one is resource requests, and the second one is resource limitations.

2.5 Sock Shop application

The Sock Shop is the front end of an online shop that sells socks. It is a microservice-based application intended to aid the behaviour of microservice-related cloud-native technologies. The technology used comprises Spring boot, Go Kit, and Node.js.

2.6 Sock Shop Architecture

The Sock Shop is a front-end online shop that sells socks. The design intends to provide as many Microservices as possible. The experiment uses five primary services, as shown in Figure 2-2. The services are Order, Payment, User, Catalogue, and Cart

.

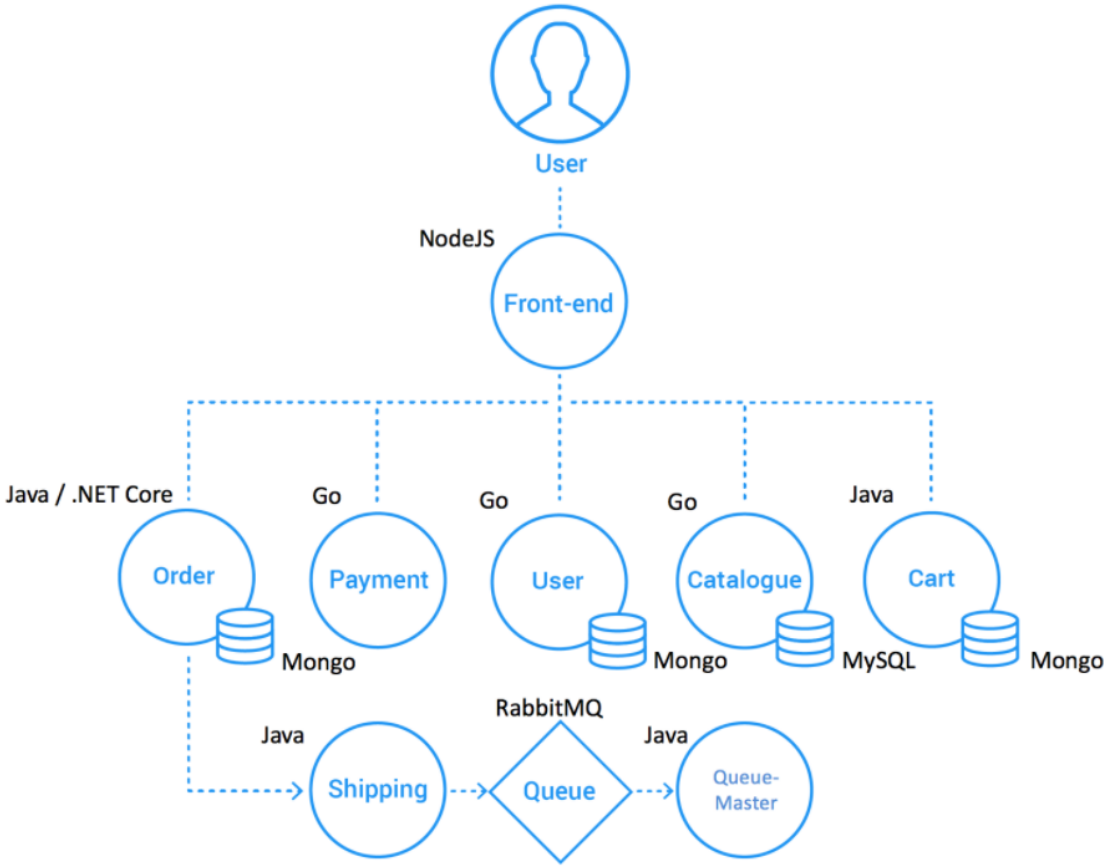

Figure 2-2 : Sock Shop Application Architecture, Source[52]

2.7 Sock Shop Setup for Experimental Execution

Figure 2-2 shows that the Sock Shop consists of five primary services. All services deploy to the cluster while the services use a unique Node. The Kubernetes dashboard is used to measure cluster-level performance as well as node-level performance.

2.8. Use of Google Kubernetes Engine Cluster for the Experimental Execution

The Google Kubernetes Engine cluster includes two node pools: the first node pool comprises two VMs (Virtual Machine), and the second includes the Kubernetes control plane services.

2.9. What is Kubernetes?

Kubernetes is a portable, flexible, open-source platform for handling containerized workloads and services that ease declarative setup and automation. It includes a large, fast-expanding ecosystem, Kubernetes services, tools, and support that are widely accessible.

2.10. What is Google Kubernetes Engine (GKE)?

Google Kubernetes Engine (GKE) provides a controlled environment for deploying, managing, and scaling your containerized applications using Google infrastructure. The GKE environment consists of multiple servers (namely, compute engine instances) grouped to create a cluster [7].

2.11. Google Kubernetes Engine[7]

GKE provides secure and manageable services with the most critical inbuilt features. Mainly, four-way auto-scaling and multi-cluster support features play a significant role in providing these services.

Four-way autoscaling is the ability to scale in four levels such as scale pods and services, make the right size of pods and the clusters while scaling the cluster.

Multi-cluster service makes geo-distribution for the service easier and also makes cross-cluster service available for the service.

2.12. Benefit of Google Kubernetes Engine

- Integrated Cloud Tracking with infrastructure, application and Kubernetes-specific perspectives.
- Safe by default, i.e, vulnerability scanning of container pictures and information encryption.
- Start fast with single-click clusters.

2.13. Key Feature of Google Kubernetes Engine [3]

- Supports for Security Standard

    - GKE complies with HIPAA and PCI DSS compliant
- Integrates logs and monitoring facilities
    - Configuration for enabling Cloud Logging and Cloud Monitoring is simple as a checkbox click and making it easy to gain insight into AUT and its execution details.
- Cluster options
    - The cluster is chosen on user requirements. The Main two options are available for cluster creation. Auto-cluster, a standard cluster which has more flexibility to edit creates clusters with predefined configuration
- Auto Scale
    - Automatically scales nodes of your application based on resource utilisation such as CPU and memory
- Auto repair
    - Nodes recovery mechanism is automatic; thus, in case of failure, the Node gets auto recover.
- Resource limits
    - Allows users to assign resources that they need for their application.
- Built-in dashboard
    - Cloud Console presents useful dashboards to your project's clusters and resources. It is possible to use these dashboards to view, inspect, handle, and delete resources on your clusters.
- Global load balancing
    - Helps users to distribute incoming requests across pools of instances across Global.
- Usage metering
    - Fine-grained visibility for the clusters. Watch your GKE clusters' resource utilisation broken down from namespaces and converted to meaningful entities.

- Per-second billing
    - Pay for use. Further to this, users need to pay only for their computing time.

Multiple performance testing types have been planned to use for the experiments. The performance testing techniques and their purpose are defined in the next section. There are several techniques to assess performance.

However, Performance testing is considered hard in conventional systems and it is more difficult in microservice-based applications due to the structural, technological, and architectural changes. While some of these changes facilitate performance testing, others cause substantial challenges. Performance testing is categorised under non-functional testing and it is used to determine the application's performance under specified conditions.

Even though performance testing may start at the code level, overall performance testing is conducted at the system level. The most critical aspect of the performance measurements is the application of responsiveness scalability.

Performance testing serves to measure other quality attributes of the system, such as:

- Reliability
- Resource usage
- Bottlenecks in the network

Typical reasons for conducting performance test is,

- to assert whether the application is reliable enough to cater to the expected load
- to identify crucial performance bottlenecks.
- to engage customers together with better responsiveness
- to support market claims
- to enhance scalability
- to identify the most efficient architectural aspect

2.14. Performance Testing Levels

- Frontend

- Client-Server (Mainly used)
- Code Level
- DB Level

Figure 5-2 elaborates on testing levels.

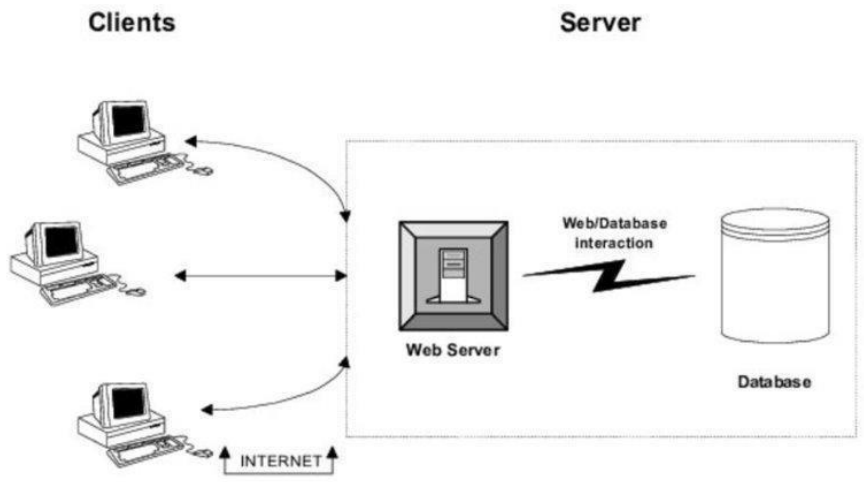

Figure 2-3:Performance Testing Levels, Source [8]

2.14.1 Frontend

The main focus is to measure the application's frontend performances, such as rendering time, DOM load time, lazy load, JS execution, and CSS loading time.

Tools used: Lighthouse, Web Page Test, Pingdom, Httperf, Page speed, Chrome dev tools (F12).

2.14.2 Client-Server (Mainly used)

The purpose of the test is to simulate the natural user behaviour by recreating client-server interaction.

Tools used:

- JMeter
- LoadRunner
- ApicaLoad
- WebLOAD

- LoadUI
- NeoLoad

2.14.3 Code Level

A code-level performance test may start at the beginning. However, at a point in the development life cycle, it measures code-level performance such as the functions and the method level.
Tools such as,

  - Hprof, Gprof, dotTrace, Jprofiler, NodeJS Profiler

Other Performance testing methods are also available to measure the performance in API and Database level.
However, this research focuses specifically on microservice-based architecture performance testing in which the testing practices are similar to the testing of clients/server performance testing to collect data and statistics.

2.12. Type of Performance Testing

- Load Test
- Endurance Test
- Stress Test
- Spike Test
- Breakpoint testing
- Volume testing

2.12.1 Load Test

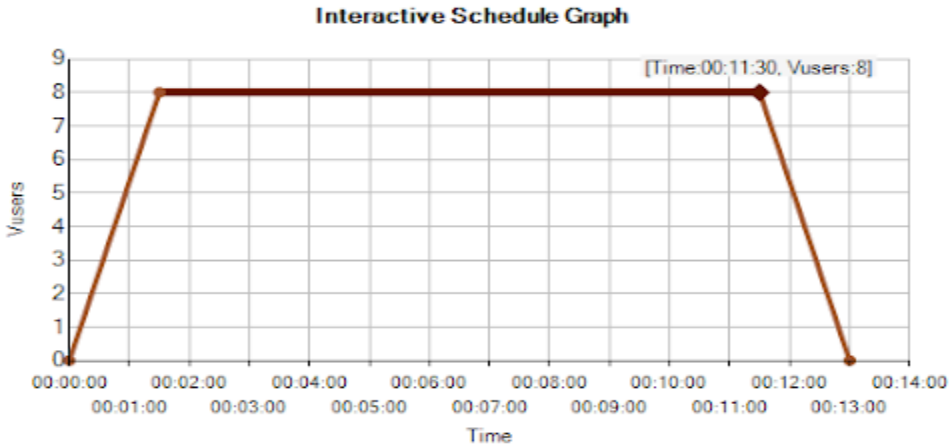

Figure 2-4 :Load Testing Pattern, Source[12]

- Refer Figure 2-4: Sustain the peak load for a small duration (ex: 20 min).
- Identify whether the system is stable enough to handle the peak load.
- If not, identify the root cause, in terms of:
    - Response time
    - Server resource
    - Network bandwidth
    - Database level

2.12.2 Endurance Test

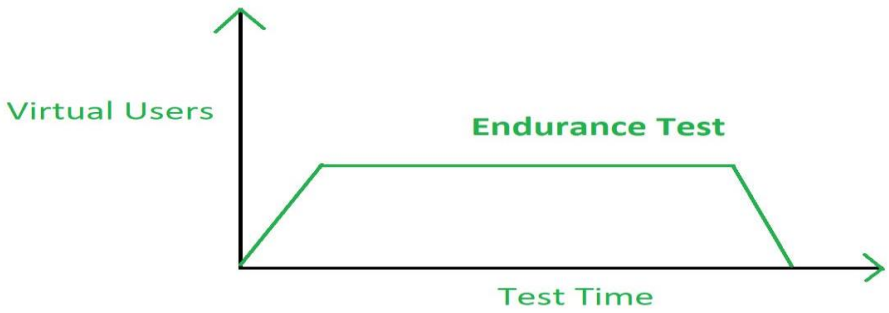

Figure 2-5:Endurance Test Pattern, Source[13]

- Refer Figure 2-5:
    - Sustain the peak load for an extended duration (ex: 3-5 hours).
    - Identify whether the system is stable enough to handle the peak load for a more extended period.
    - If not, identify the root cause, in terms of:
        - Memory leaks
        - Resource shortages
        - Response time spike
        - Network bandwidth issues

2.12.3 Stress Test

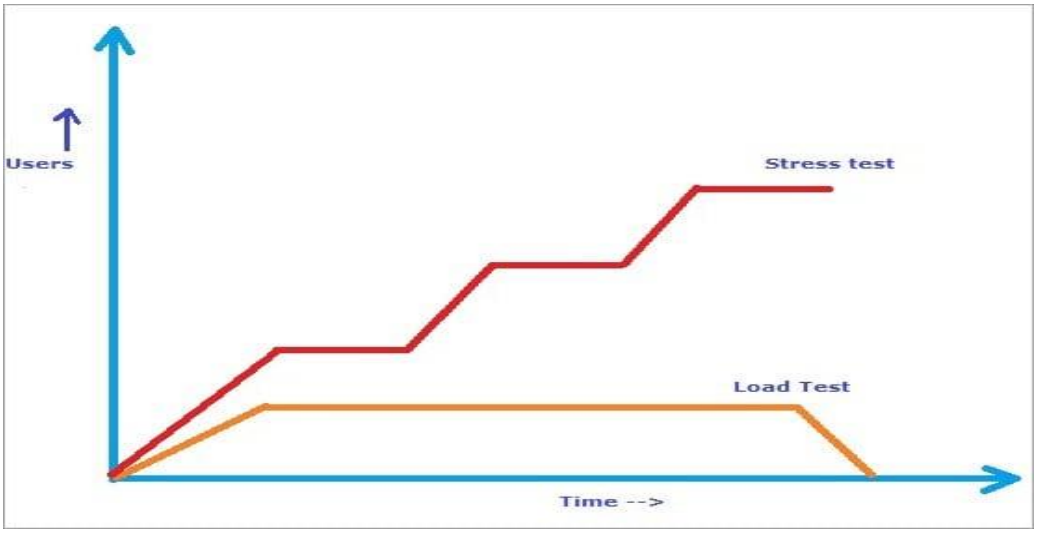

Figure 2-6:Stress Test Pattern, Source [14]

- Refer Figure 2-6:
    - Gradually increase the load until the application exhibits abnormal behaviour
    - Monitor system behaviour under increased load condition.
    - Identify potential issues in terms of:
        - Memory leaks

        - Synchronisation issues
        - Resource utilisation
        - Network bandwidth issues

2.12.4 Spike Test

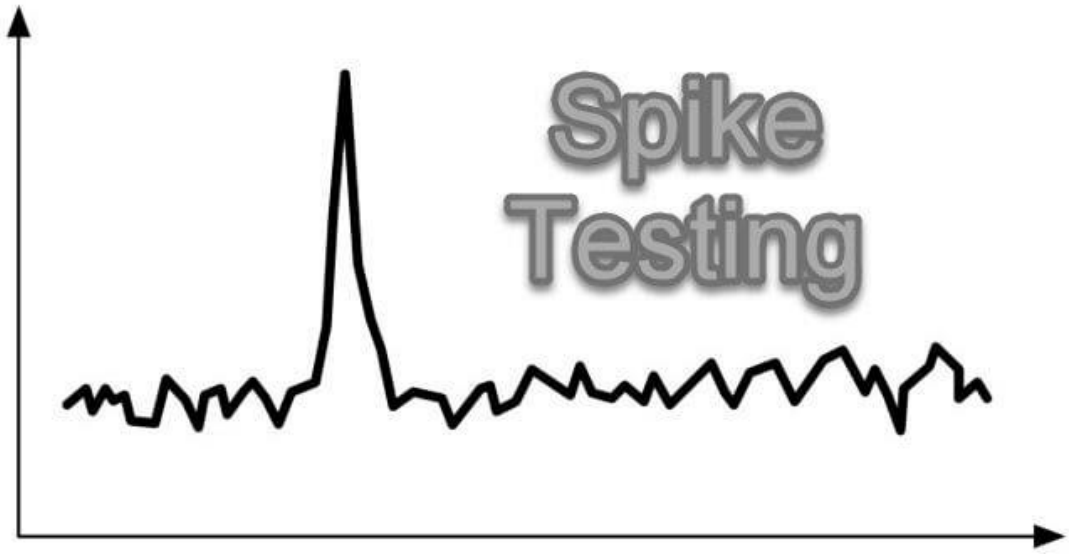

Figure 2-7:Spike Testing Patterns, Source [15]

- Refer Figure 2-7:
    - Apply extreme increments and decrements to the system in terms of load.
    - Monitor system behaviour under the given condition.
    - Identify potential issues in terms of:
        - Memory leaks
        - Synchronisation issues
        - Resource utilisation
        - Network bandwidth issues

2.12.5 Breakpoint Test

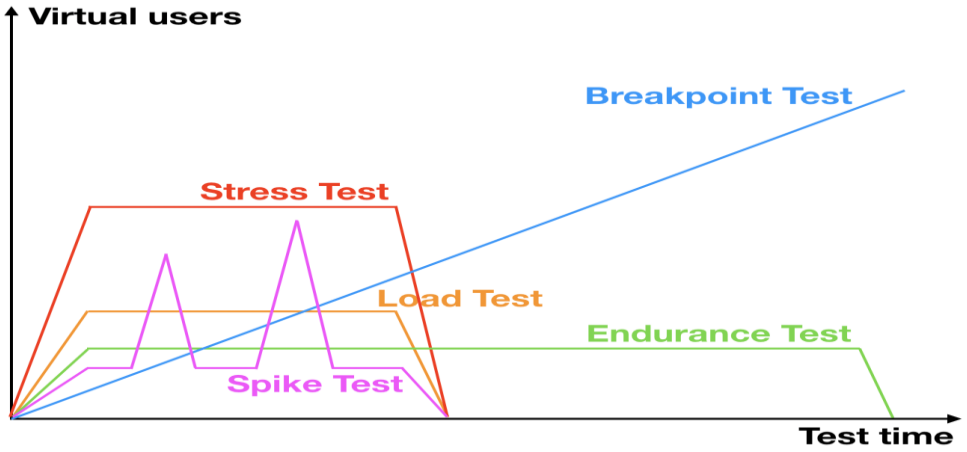

Figure 2-8:Breakpoint Testing Pattern, Source[16]

- Refer Figure 2-8:
    - Apply load until the system fails.
    - Analyse which component fails first and why
    - Determine the impact of the failure on the other systems.

First, three test types are used to evaluate microservices in this research and they are analysed to identify the best metrics for the microservice-based architecture.

2.13. Performance Measurements and Matrices

2.13.1 Measurements

Measurements are the data such as response time resource utilisation collected during the execution.

Typically, the following facts are measured during the performance test:

- Latency
- Throughput
- Disk read/write rate
- CPU usage
- Memory usage

- Network bandwidth utilisation
- Database profiling
- Power consumption

2.13.2 Metrics

The measurements derive metrics and the metrics are meaningful and easy to understand. Therefore Performance attributes are mainly determined by the Metrics.

Some of the metrics are shown below,

- Average response time (total response time/requests)
- Peak response time
- Error rate - (less than 3%)
- 90^{th} percentile response time
- Requests per Second
- Throughput

Other than the metrics, System performance is determined with the help of graphs. The "Response vs. data" graph and "hits per second" graph are examples of the graphs used for performance analysis.

The server counter collects measurements in the server. such as,

- Network bandwidth utilisation graphs -
- CPU/Memory/Disk input/output graphs

2.14. Performance Testing Process

Following steps should be conducted in the performance testing process ,

- Identify the objective of the test
- Create test plan and script
- Test execution
- Result analysis

The testing approaches is shown in Figure 2-9.

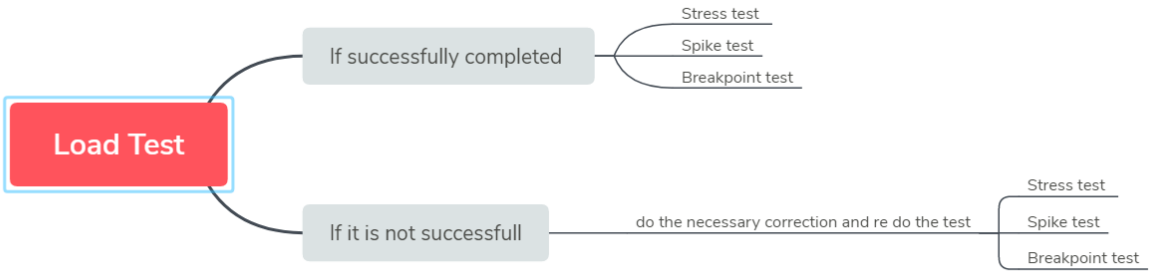

Figure 2-9:Testing Approaches

2.15. Microservice- based Application Architecture and Related Researches

Most organisations adopt microservice; this tends to expand their research on microservice-based application testing.

Few related researches are listed below.

- Functional tests [22]
- Performance test [23] , [24]
- Reference applications for microservices and benchmarking [18]
- Network overhead [25]
- Overview of the evolution of microservices[20]

2.16. Performance Testing in Microservices- Based Applications

Microservice has no proper definition. However, specific characteristics are typical in architecture [24, 21, 22]. The main features are,

(01) Self-containment:
Every microservice is self-contained, i.e., its setup unit is executed in its procedure context and it contains its own data storage.

(02) Loosely coupled:
The main characteristic of microservice is that a single service should not impact others. Thus, platform-independent interfaces mostly use microservices to maintain the loosely coupled behaviour.

(03) Independent development, build, and deployment:
Separate growth, installation and build services are facilitated with Microservices which are technically isolated from each other. Thus, it helps to construct and deploy independently in different microservice platforms. This behaviour suits the proper structure for DevOps [3].

(04) Containers and Container Orchestration:
The container image is used to configure the Container whereas the containers implement virtualized platforms such as Docker.
These images include the microservice itself, the essential runtime libraries, and the underlying operating system elements.
Container orchestration options like Kubernetes have emerged to effectively manage the possibly high number of container cases within a microservice structure.
Kubernetes automatically deploy containers onto many hosts, supply auto-scaling and load balancing functionalities and offer self-healing capabilities in terms of node failure..

(05) Cloud-native:
Many famous microservice installations (e.g., Netflix and Amazon) are in the cloud. As a result of loose coupling and separate setup, microservices permit extensive scalability, and the cloud provides the tools to operate the necessary cases.

2.16. Conclusions Based on Literature Review

Performance testing is a non - functional testing type that helps software systems to get comparative advantages in the market. Microservice-based applications are more distributed and they have more advanced technologies than monolithic applications. Thus, conducting performance tests with microservice-based applications is more challenging.

This Literature review aims to identify the proper way of experiment execution patterns and to discuss methods of determining more reliable metrics for performance testing in microservice-based architecture.

In this backdrop, the TeaStore and Sock Shop applications were selected to test. The deployment environment of both applications is the Google Kubernetes cluster, and also, the critical workflow is defined.

Experimental execution consists of selected testing types. Testing types are load tests, stress tests, and endurance tests. Additionally, TeaStore application performance is considered a benchmark, and Sock Shop application performance confirms the microservice behaviours.

Each test intention proves metrics reliability concerning responsiveness, availability, reliability, and resource usage limits.

3. METHODOLOGY

3.1 Chapter Overview

This section describes the deployment and maintenance of large-scale microservice-based applications and the experimental setup with Google Kubernetes Engine.
Then we discuss how to measure the application's performance metrics in TeaStore and Sock Shop applications.

3.2. Setup Application Under Test

Micro-service-based application environments are modern technology and they are open to many research areas. This section describes the application configurations used for the test execution. There are two applications used for the testing, TeaStore and Sock Shop.
The Tea Store is a micro-service-based web application mainly used to benchmark related research areas such as microservice environment performance. The Sock Shop is also a microservice demonstration application used for research purposes.
The TeaStore offers three ways to deploy and run the application as a benchmark application for performance tests,

- The recommended way is to deploy and run on the containers using Docker.
    - TeaStore facilitates configs for Kubernetes.
    - Deploy in one or several Java application servers(s) using Java WARs with any container supporting JAVA Servlet 3.1. it is tested primarily on Tomcat 8.5, the recommended container to use.

TeaStore was configured on Google Kubernetes Engine with the Kubernetes cluster for this study. Sock Shop facilitates different deployment methods, although this research uses the Kubernetes platform to deploy the Sock Shop.
Two methods can be used to deploy on the Kubernetes environment that is deployed with,

- Manifests directory

- Demo.yaml

YAML is a single file manifest made by linking all the manifests from the manifests directory to regenerate better results through changes made on the manifest. Sock Shop is also deployed on the Google Kubernetes cluster with Demo.yaml method.

3.3. Run the TeaStore on a Kubernetes Cluster

Two ways of configuring TeaStore on Kubernetes,

1. Configure Kubernetes to deploy the TeaStore pods.

    Way of communicating:

    Pods resolve each other and communicate directly using Ribbon (the client-side load balancer provided inside each TeaStore service instance).

    Benefits: Using Ribbon in Kubernetes emulates the communication and performance of a non-Kubernetes setup, enhancing comparability

YAML to deploy: kubectl create -f

https://raw.githubusercontent.com/DescartesResearch/TeaStore/master/examples/kubernetes/teastore-ribbon.yaml

2. Configure TeaStore to use Kubernetes Cluster IP for addressing services

    Way of communicating:

    All communications are routed through Kubernetes

    Benefits: Instrumentation of internal Kubernetes load balancing is useful for tests and research focussing on these aspects. TeaStore registers into thinking that only a single instance of each service is present.However, TeaStore's service status report page will not provide correct results in this case.

    YAML to deploy: kubectl create -f

    https://github.com/DescartesResearch/TeaStore/blob/master/examples/kubernetes/teastore-clusterip.yaml

Both variants expose the TeaStore WebUI using NodeIP on Port 30080,
Sample URL could be:Sample URL could be:
Example URL for deployments: http://K8S_NODE_IP:30080/tools.descartes.teastore.webui/

TeaStore is deployed with the "Configure Kubernetes to deploy the TeaStore pods" method. Each node uses the Ribbon in Kubernetes nodes emulating the communication and performance of a non-Kubernetes setup, enhancing comparability.

3.4. Run the Sock shop on a Kubernetes Cluster

First, create a cluster in the Google Kubernetes engine and install a kubectl to connect to the cluster.
The next step is to clone the microservice -demo repository. Then go to the Kubernetes/Deploy folder and execute the command below that is used to deploy the application successfully in the Google Kubernetes engine.
Command: kubectl apply -f complete-demo.yaml
Sample URL : https://8080-6de404d1-2c5a-4388-a48b-9b42bd971d69.cs-asia-southeast1-yelo.cloudshell.dev/catalogue/size?tags=

3.4.1.The GKE Cluster

The Cluster comprises a control plane and numerous nodes; it functions with the Kubernetes cluster orchestration system. [10] The Kubernetes cluster is the base of this Google Kubernetes Engine (GKE). Each program component can be found within the Kubernetes cluster and run along with the same. Overview of the GKE Cluster is shown in Figure 3-1.

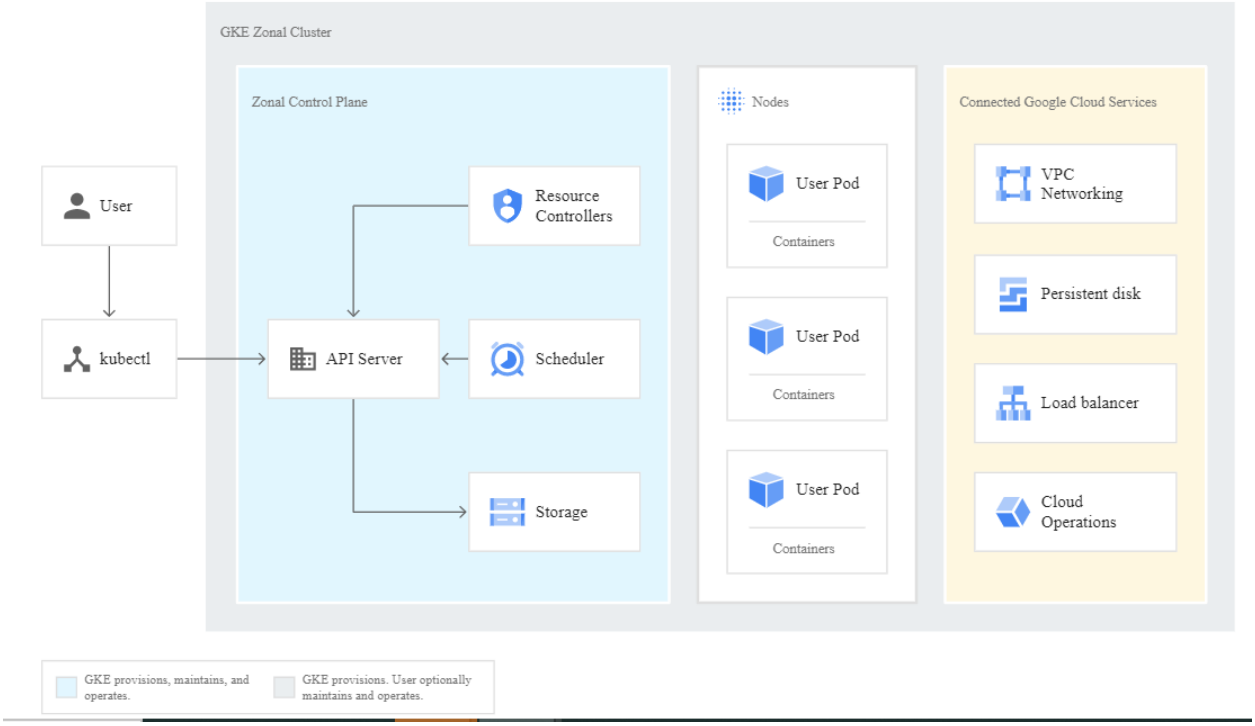

Figure 3-1:Overview of GKE Cluster, Source[18]

3.4.2.Modes of operations

Autopilot mode: Provides a fully provisioned and handled cluster configuration for clusters made using the Autopilot manner. Pre-defines the cluster configuration that is easy to use. Autopilot clusters are built together with an optimised cluster setup that is prepared for production workloads.

Standard Mode:

Cluster configurations for typical style production load are grouped under standard mode. Not like autopilot mode, the standard mode offers more advanced configuration flexibility. The experimental setup requires limiting the auto-scaling and auto-scalability features to understand the performance metrics behaviour of the microservice-based applications. Thus, TeaStore and Sock Shop are deployed with a standard mode setup. Overview of the cluster setup in standard mode is shown in Figure 3-2.

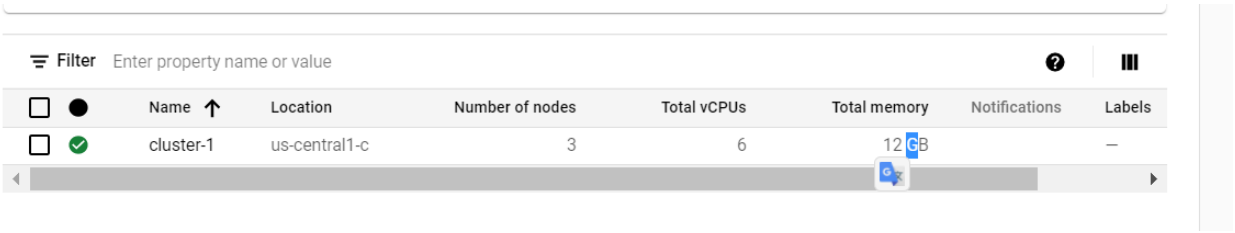

Figure 3-2 : Cluster Setup for the experiment

The cluster information is shown in Figure 3-3

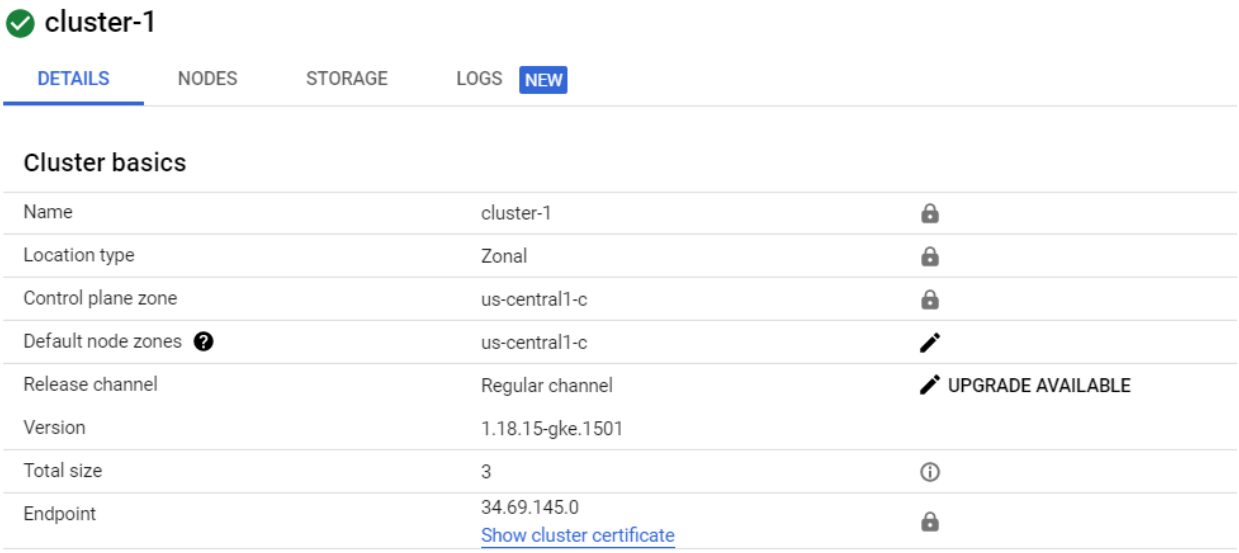

Figure 3-3 :Cluster configuration

Each service has unique nodes. Overview of the Node is shown in Figure 3-4.

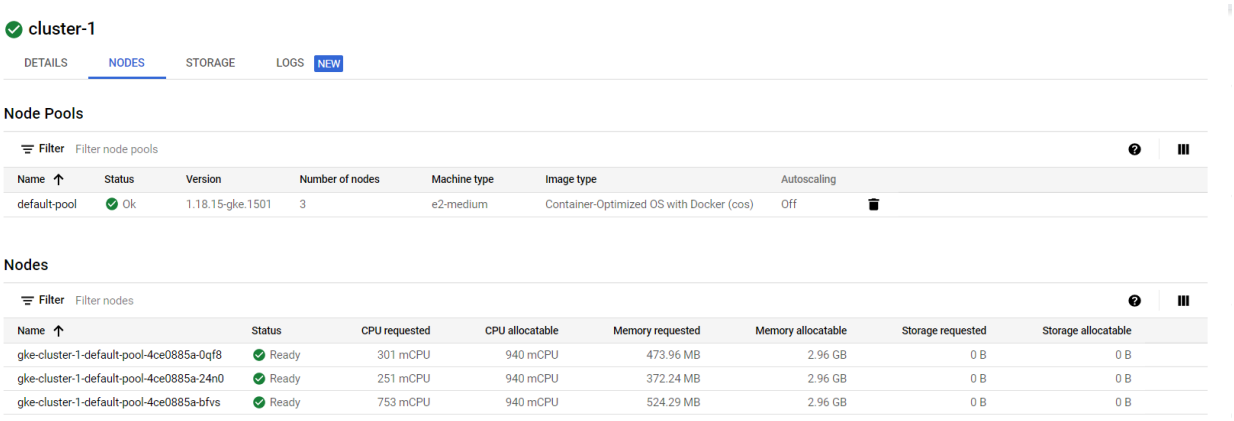

Figure 3-4 :Node Overview

3.5. Deployment Steps

The TeaStore and Sock Shop are configured to deploy in the microservice architecture. As discussed in Section [3.3], Google Kubernetes Engine (GKE) is used as an experimental environment. The YAML file with GKE configuration is executed inside the cluster terminal. By executing each YAML command relevant to each application, all features are deployed on the cluster.

Once successfully deployed, all features are listed under the services of work tab. The figure 3-5 shown the workload tab with services list.

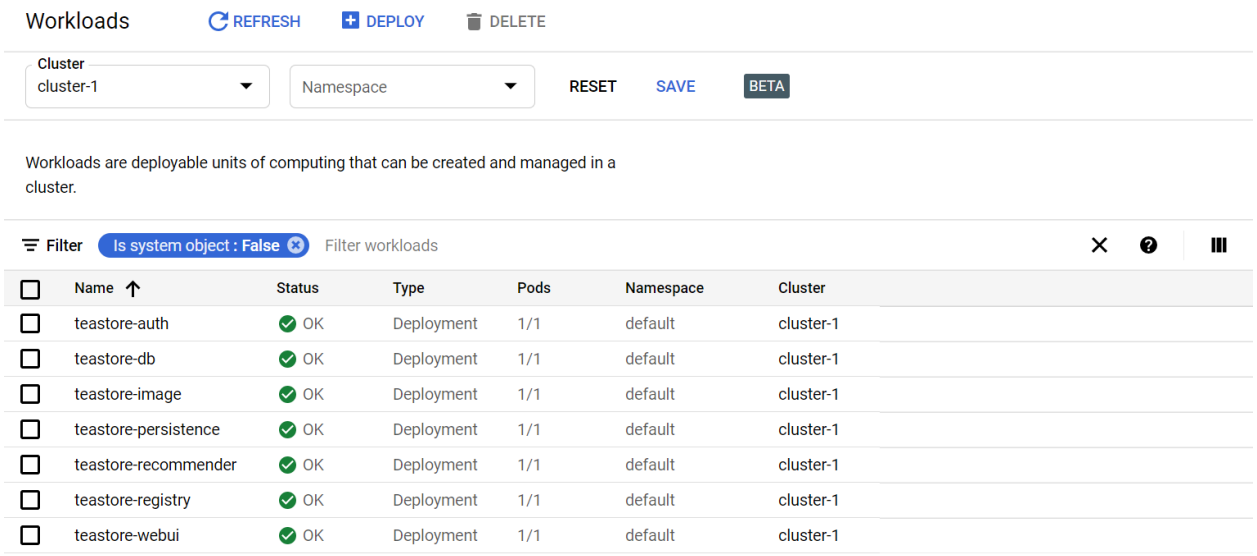

Figure 3-5 :Workload tab with TeaStore application

Figure 3-6 is shown WebUI deployment details including overview of cluster and pod specifications.

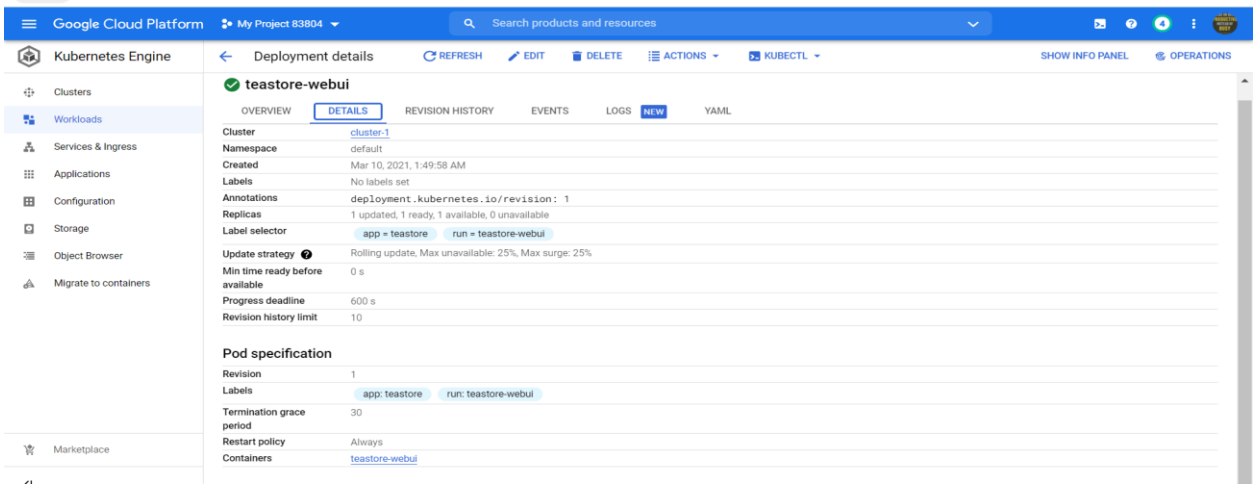

Figure 3-6 : Deployment details of the TeaStore UI

The Service & Ingress locate all the application services in the Kubernetes cluster. Both TeaStore and Sock Shop show their services under The Service & Ingress.

For example, figure 3-7 shows the way TeaStore services appear in the Kubernetes cluster.

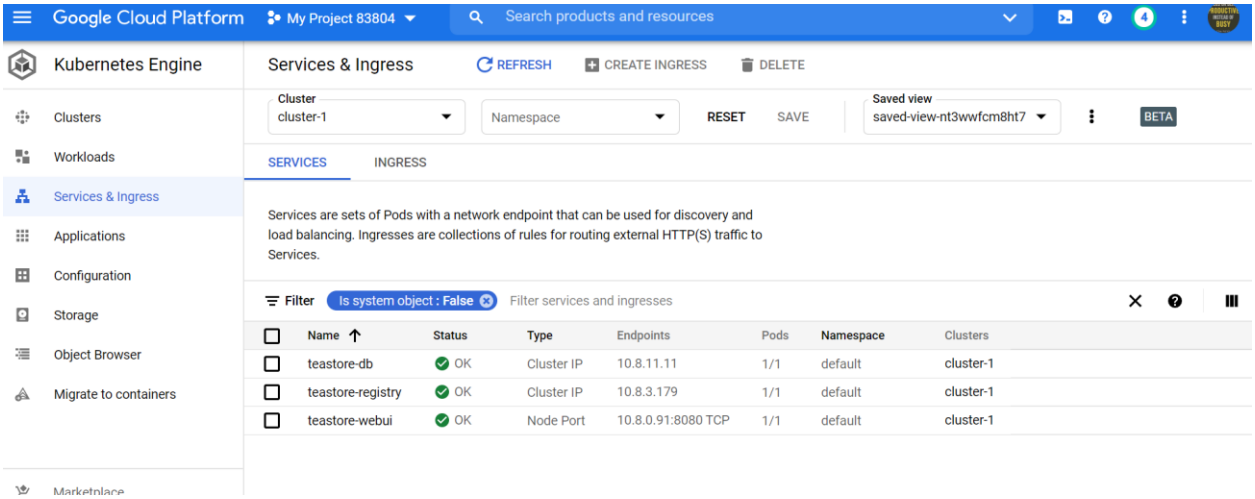

Figure 3-7 : Services of the TeaStore application

Port forwarding provides a URL to access the application, and through the URL, other services also become accessible. Figure 3-8 shows the node port that facilitates the port forwarding of TeaStore.

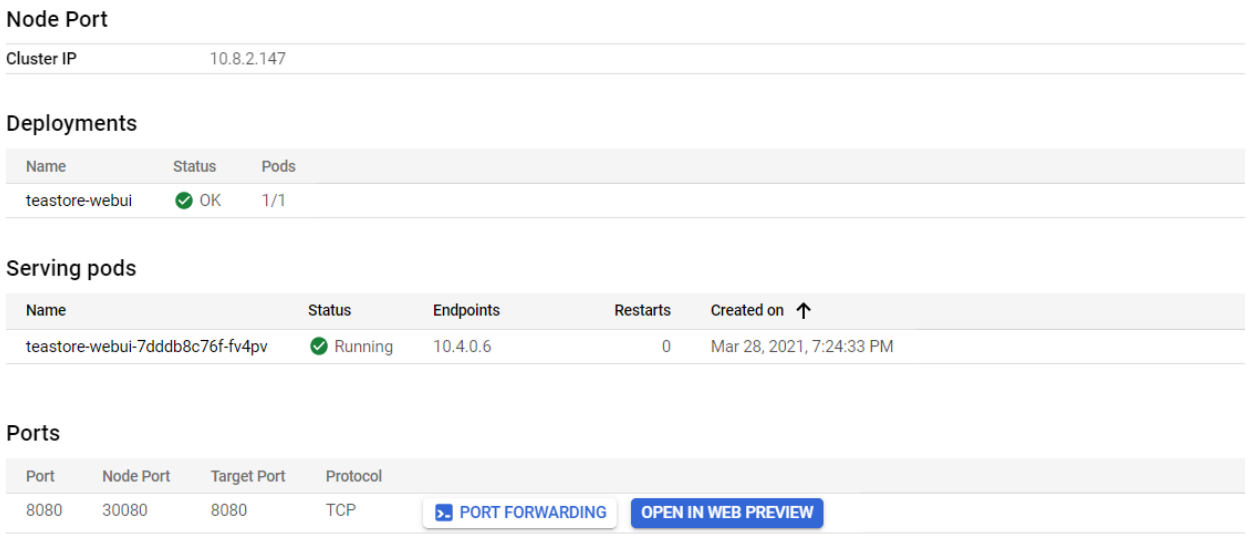

Figure 3-8:TeaStore: WëbUI Node Port

The default port for WebUI of the TeaStore is 30080, and it is redirected to 8080 to access as URL. Sock Shop uses the same method to expose the URL.

Actual URL of the experimental setup :

TeaStore : https://8080-cs-454127571836-default.cs-asia-southeast1-vjqr.cloudshell.dev/tools.descartes.teastore.webui/

Sock Shop :

https://8080-6de404d1-2c5a-4388-a48b-9b42bd971d69.cs-asia-southeast1-yelo.cloudshell.dev/catalogue/size?tags=

3.6. TeaStore Application UI

The TeaStore [2] is an online shop for tea and tea-related utilities. Products are sorted into categories, with unique information such as trailer images for each category. The primary content reveals the chosen category products, such as shortened product information and the preview image. The main UI of the TeaStore is shown in the Figure 3-9.

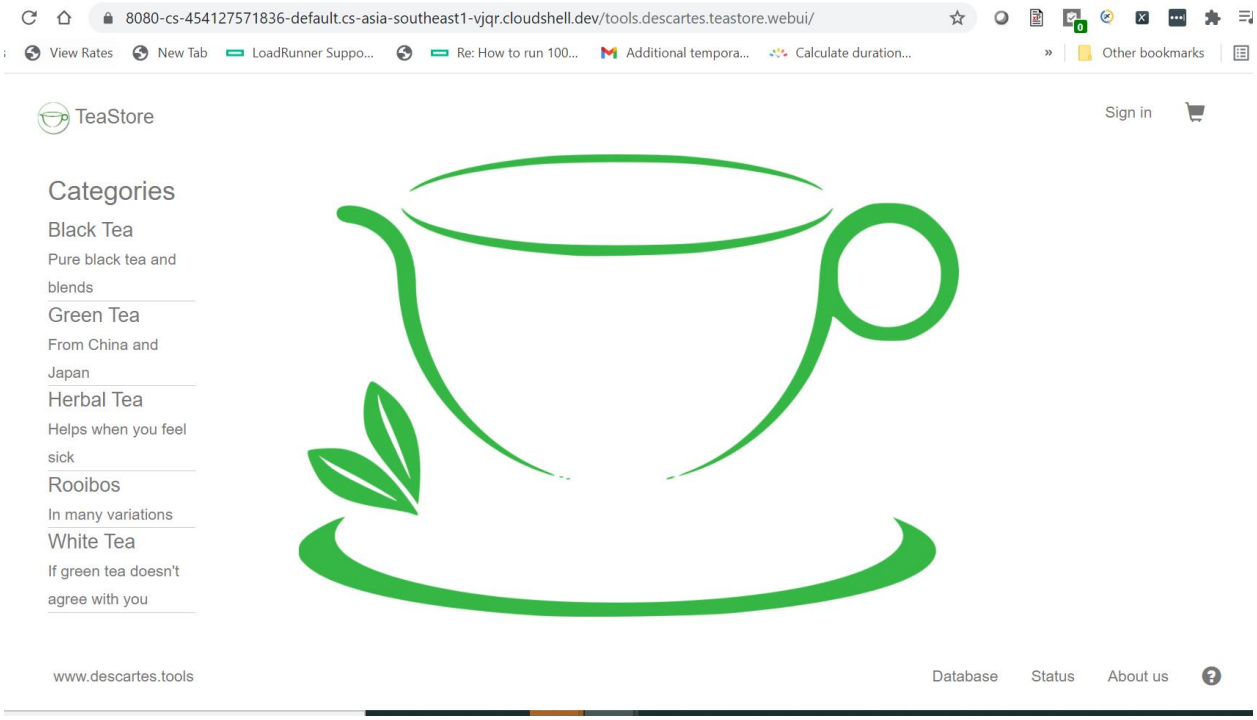

Figure 3-9 :TeaStore UI

3.6. Sock Shop Application UI

The workflow covers most of the services deployed in the GKE.

Some of them are,

- index.html
- login
- topbar
- navbar
- footer
- navigation
- cart

Each transaction accesses the services of the Sock Shop application. The main UI of the Sock Shop application is shown in the Figure 3-10.

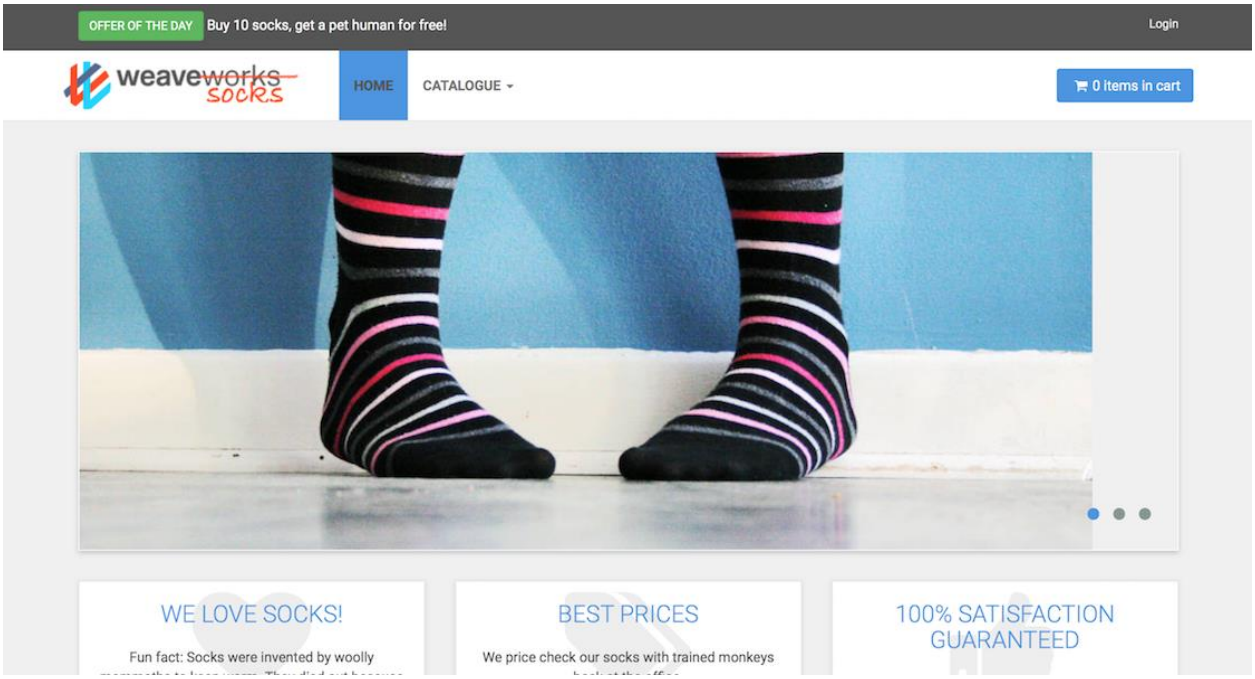

Figure 3-10 :Sock Shop UI

4. IMPLEMENTATION

4.1. Chapter Overview

The research objective is to determine the execution variance to identify the most reliable metrics for microservice-based architecture.

Applications under test are tested with various load patterns used for performance testing. The behaviour of the microservice-based application and the necessary performance metrics were collected and analysed comparatively to identify the correlation between each metric used for the testing.

This section discusses the load patterns, the scripts, the environment setup, and the necessary information to execute the performance test.

4.2. The Workload Modelling and Execution

The evaluation consists of the three load generation patterns with various load levels. The base load was the peak load which is ten threads

Each load pattern has a unique objective whereas each test helps to identify microservice-application behaviours and bottlenecks related to various loads, ultimately helping to conclude reliable performance metrics.

Pattern 1: Load Testing:

Figure 4.1 illustrates the load testing pattern. The objective of the load test is to identify the performance testing behaviour under a peak load workload. Typically, The peak load is the maximum load that could cope with the application. To make it in line with the rule of thumb, nearly the same workload was used as the benchmark test that was done in the related research [1]; Thus, the load used in the benchmark test was around 700 requests.

Ten threads is used in this experimental setup to simulate the number of requests which are nearly similar to the benchmark test.

Load test simulation has three main steps,

Step 1: Increase the load gradually up to the expected load

Step 2: Maintain the threads for a duration of 20 to 30 minutes

Steps 3: Gradually decrease the load

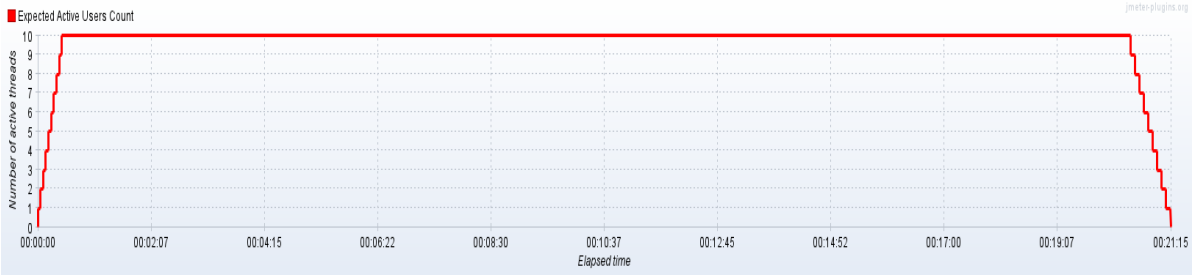

Figure 4-1:Load Testing Setup

Pattern 2: Endurance Testing:

An endurance test determines the availability of the application. For example, if the peak load is "x," the load for the endurance test is defined as 0.5x or higher. In this test, the duration was around three hours. Therefore, scalability, Durability, and availability were the quality attributes verified by the endurance test.

Endurance test simulation has three main steps.

Step 1: Increase the load gradually up to the expected load

Step 2: Maintain for a duration of at least 3 to 4 hours

Steps 3: Gradually decrease the load

Figure 4.2, Illustrate the endurance test pattern

Figure 4-2 :Endurance Testing Setup

Pattern 3: Stress Test:

This test determines the robustness of the application by testing beyond the expected load. If the Peak load was "X," then the load simulates in the pattern of 1x, 1.5x, 2x, 2.5x, and simulates for 15 minutes in each load level. The application's stability in each load level is the crucial area verified with this test.

Stress test simulation:

Step 1: Increase the load gradually up to the expected load

Step 2: Maintain for a 10 - 15minitue duration

Steps 3: Increase the load to 1.5X and maintain for a 10 - 15 minute duration

Steps 4: Repeat step 3 for 1.5x,2x,2.5x; in this test expected load is ten users and increased up to 200 users

Steps 5: Gradually decrease the load

Figure 4.3 Illustrates the stress testing pattern.

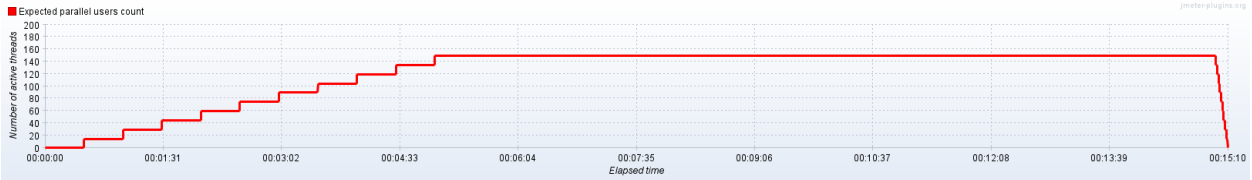

Figure 4-3: Stress Testing Setup

By analysing the test results and metrics collected during all the above tests, identify the pros and cons of each metric. Most importantly, these metrics analyse the application's performance deployed in the microservice infrastructure. As per the related research conducted [1], finding out the most reliable metrics in the microservice-based architecture is challenging when compared to the other deployment environments.

4.3. Performance Metrics for the Evaluation

This section describes the metrics used for the experimental evaluation and critically evaluate the pros and cons of the stated metrics

Following metrics are taken from the client perspective

Response time/ Latency: In the client-server architecture, the response time is the time taken to send the client's response, which means how long the request takes to receive the server's response on which the application is deployed.

Following measurements were considered during the experimental setup,

- Average: Accumulation of response time of total samples/ number of samples
- Max: maximum time taken to respond
- Min: Minimum time taken to respond

This section describes the metrics used for the experimental evaluation, critically evaluating the pros and cons of the selected metrics. Some of the metrics are shown below,

Response time/ Latency : In the client-server architecture, Response time means the time taken to receive a response for a particular request that is sent to the application.

Thus, it indicates how long the request takes to receive the application server's response.

Response time describes different perspectives, such as,

- Average: Accumulation of response time of total samples/ number of samples
- Max: maximum time to respond
- Min: Minimum time to respond

Throughput: Throughput is defined as the amount of data/ requests transferred during the test.

Some of the features in Throughput are,

- Receive KB/sec
- Sent KB/sec
- Average bytes

Hits per Second: Number of HTTP/HTTPS requests sent by the user(s) to the application in a second.

Also, the collected 'connect times over time' and 'Active threads over time.' metrics.

Other than Performance Test Tool-based metrics, the application's server-side metrics are considered. Generally, it is called resource utilisation metrics. The measurements are,

- Overall cluster resource utilisation
- The node level resource utilisation

4.4. Test Scripts and Execution

JMeter scripts used for the test,

Figure 4-4 shows the transactions and listeners used for the TeaStore test.

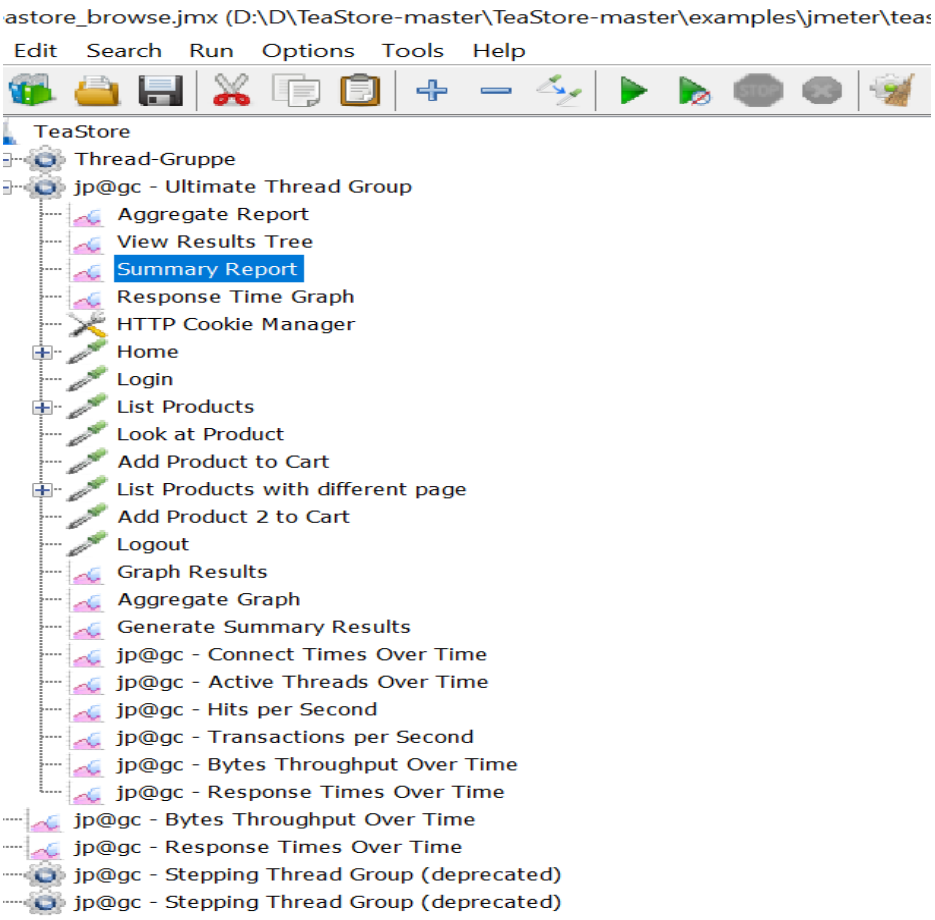

Figure 4-4 :JMeter script overview

5. EVALUATION

5.1. Chapter Overview

This section describes the methods used to evaluate the performance metrics. The evaluation consists of two steps while the assessment of the TeaStore application is a preliminary task. In this test, a Series of load generation patterns were used to identify the best performance metrics in TeaStore. As stated in the previous session, load patterns are load test, endurance, and stress test.
The evaluation of the Sock Shop application is the secondary evaluation. Again, the same set of test patterns were used for the secondary assessment.
The critical contribution of the experiments is the analysis of data collected through the series of tests conducted on each application under test. Following the step of the evaluation, the below-mentioned primary actions were taken.

- Conduct a test and collect the necessary test results
- Analysis of the test result to identify the best metrics

The evaluation section describes the initial observations and the test results.
Finally, the analysis identifies the best performance metrics for the microservice-based architecture.

5.2 .Evaluation of Load Test Results

The Load test plays a significant role in analysing reliable metrics for microservice-based applications.
The load test duration was 20 minutes during the execution with the pre-defined metrics. Response time is a crucial metric because the different aspects of response time are monitored.
They are;

- The standard deviation of latency
- Average response time
- Response time graphs

Along with response time, throughput and resource utilisation of application servers are monitored.

5.2.1 Response Time for TeaStore - Load Test

TeaStore deploys in Google Kubernetes Engine. Response times were monitored and collected with the JMeter console with the help of a few different graphs as well.

JMeter generates the load via a virtual machine which helps to reduce the network latency that impacts the overall result. The same test is executed twice to check the consistency of the test results.

The total number of samples used to test the TeaStore was 54,689. Key response time metrics were Average response time and standard deviation.

As illustrated in table 5-1, the response time has not shown a significant deviation.

In a nutshell, TeaStore application performance shows,

- The Average response time is around 332 - 345 (milliseconds)
- The standard deviation is approximately 48.79 - 76.02

	Test Run – 01			Test Run - 02		
Workflow	**Sample**	**Average RT(milliseconds)**	**st.Dev**	**Sample**	**Average RT(milliseconds)**	**st.Dev**
Home	6840	332	76.02	6482	345	48.06
Login	6840	327	67.32	6484	346	49.69
List Products	6840	327	67.12	6491	349	57.28
Look at Product	6838	328	67.89	6486	346	49.86
Add Product to Cart	6836	328	66.28	6483	345	44.63
List Products with different page	6834	328	68.16	6489	345	48.79
Add Product 2 to Cart	6831	327	65.87	6481	345	46.38
Logout	6830	328	68.97	6485	344	40.91

Table 5-1: Response time summary in two execution cycles

Other than the text-based response time, graphical data gives more insight regarding metrics. The response time behaviour is shown in Figure 5-1. Neither outbreak nor spike is shown in the graphs over time.

Figure 5-1: Response time over the period of time is shown in the graph.

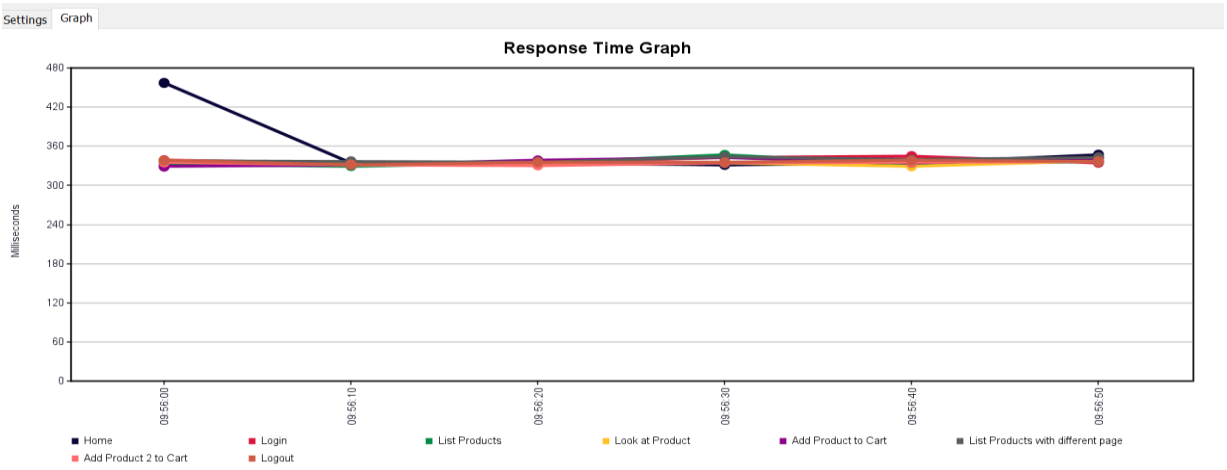

Figure 5-1 :Response time over time graph

Figure 5-2 shows the comparative view of all the response time metrics and Throughput against the data.

No significant deviation is identified in Figures 5-2.

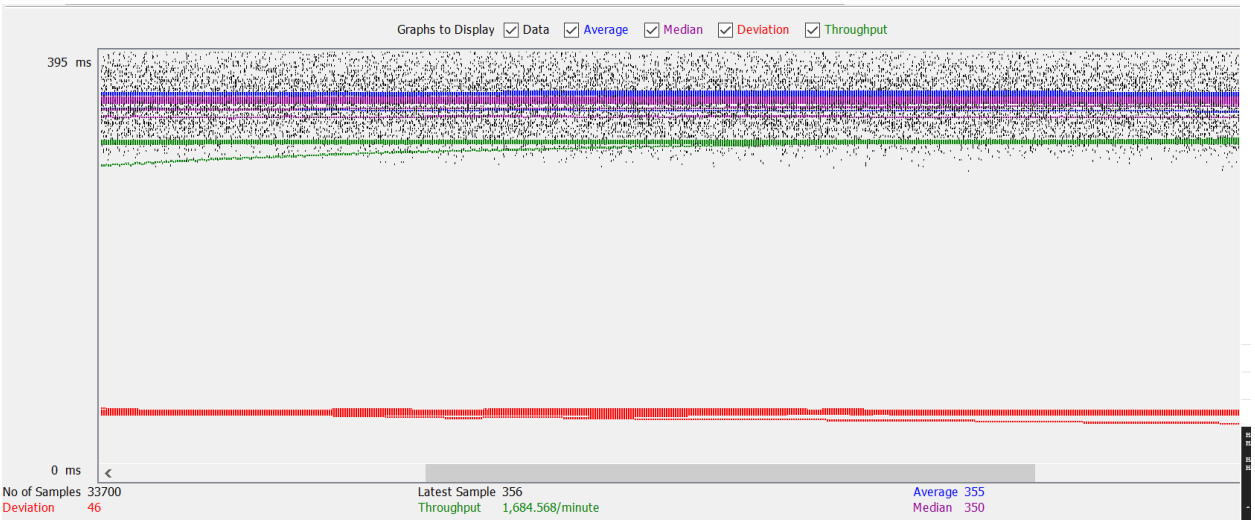

Figure 5-2 : All key metrics , vs data

Figure 5-3 shows the response time behaviour vis-a-vis the load.

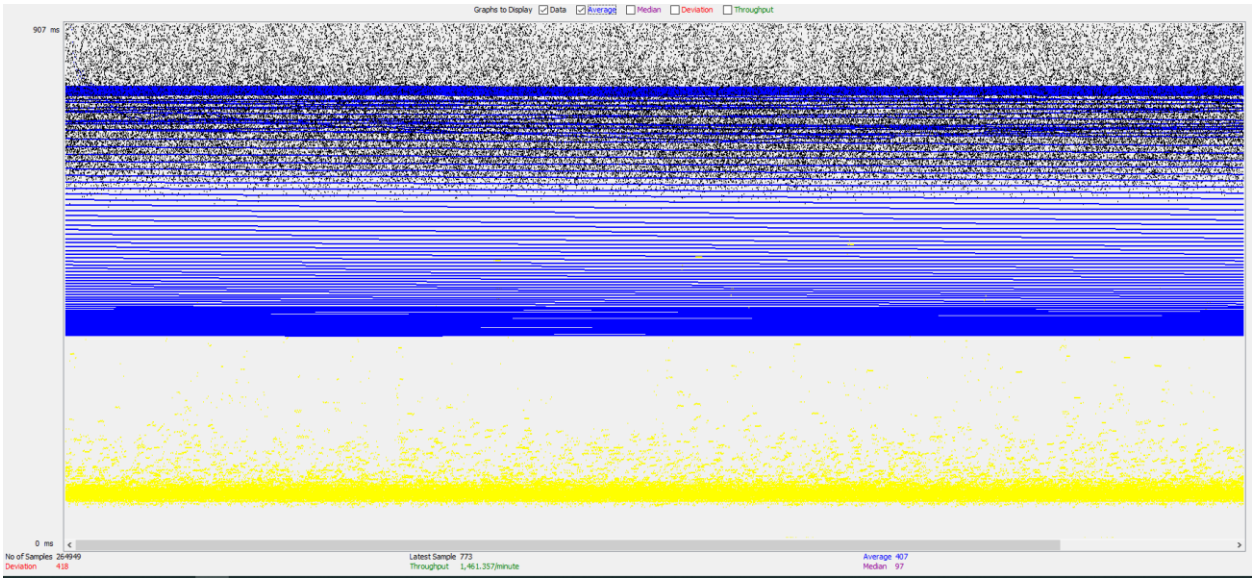

Figure 5-3 : the response time behaviour vs with the load

5.2.2 Response time for Sock Shop - load Test

The Sock Shop deploys in the GKE, and the exact load test steps are followed. JMeter is the load generation tool that generates load via the virtual machine. The total number of samples used for the test was 52,532.

Key response time metrics are the Average response time and the standard deviation.

In a nutshell, the Sock Shop application performance shows that,

- The Average response time is around 317- 940 (milliseconds)
- The standard deviation is approximately 36.23 - 90.62

As illustrated in table 5-2, the response time has not shown a significant deviation.

Workflow	Test Run -01			Test Run -02		
	Sample	Average	st.Dev	Sample	Average	st.Dev
index.html	3279	940	63.76	3291	939	55.53
Login	3279	940	63.77	3291	939	55.53
top bar	3276	320	48.32	3289	322	51.77

Navbar	3276	317	41.50	3288	320	44.15
Footer	3276	317	36.23	3287	317	36.86
Navigation	3276	954	90.62	3287	960	92.27
catalogue?size=	3273	317	41.17	3287	318	39.69
catalogue/size?tags=	3272	317	40.04	3285	319	42.71
Catalogue	3272	634	66.64	3285	638	66.47
Cart	6544	318	42.86	6564	318	38.00
category.html	3264	318	42.29	3274	319	40.70
Category	3272	317	45.07	3282	318	43.59
Tags	6526	318	44.30	6548	322	56.00

Table 5-2 : Sock Shop Load Test Execution summary

Figure 5- 4: Response time behaviour of Sock Shop is shown in the graph.

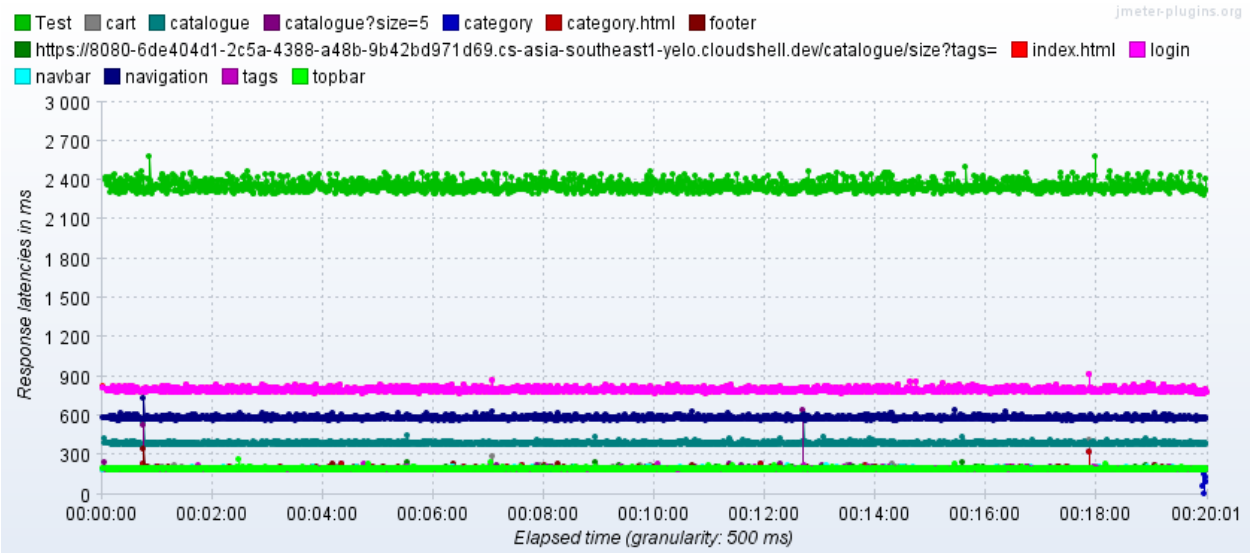

Figure 5-4 Sock Shop Response Time behaviour

5.2.3 Throughput for TeaStore - Load Test

The throughput indicates the data transfer through the network. The Average throughput is 3.8/Sec for each transaction. No deviation in the graphs results.

Throughput summary of TeaStore is shown in table 5-3.

Workflow	Test Run - 01	Test Run - 02
	Throughput	Throughput
Home	3.80	3.61
Login	3.80	3.61
List Products	3.80	3.61
Look at Product	3.80	3.61
Add Product to Cart	3.80	3.61
List Products with different page	3.80	3.61
Add Product 2 to Cart	3.80	3.61
Logout	3.80	3.61

Table 5-3: Sock shop Throughput summary

Figure 5-5, shows the Throughput behaviour during the load test

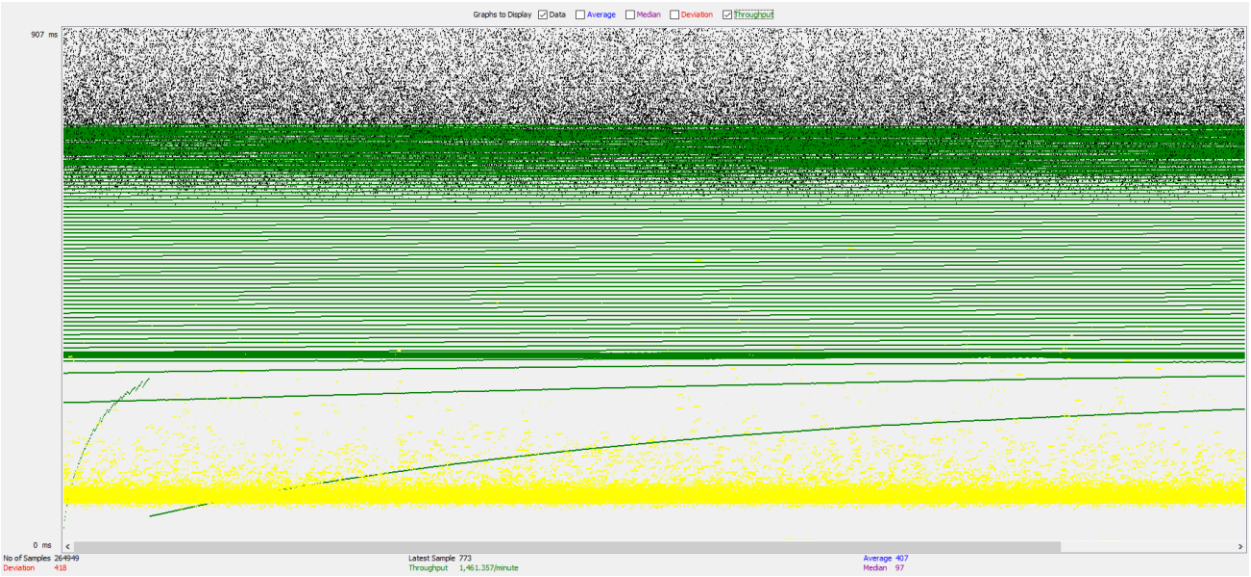

Figure 5-5:the Throughput behaviour during the load test

5.2.4 Throughput for Sock Shop - Load Test

The sock shop shows an average throughput of 2.7% for each transaction. No deviation in the throughput.

Throughput summary of Sock Shop is shown in table 5-4.

Workflow/URL	Test Run -01 Throughput	Test Run – 02 Throughput
index.html	2.73	2.74
Login	2.73	2.74
top bar	2.73	2.74
Navbar	2.73	2.74
Footer	2.73	2.74
Navigation	2.73	2.74
catalogue?size=5	2.73	2.74
catalogue/size?tags=	2.73	2.74
Catalogue	2.73	2.74
Cart	5.47	5.48
category.html	2.73	2.74
Category	2.73	2.74
Tags	5.46	5.48
Test	2.72	2.73

Table 5-4:Average Throughput in Load test of Sock shop

Figure 5-6 : shows the Throughput behaviour during the load test - Sock Shop

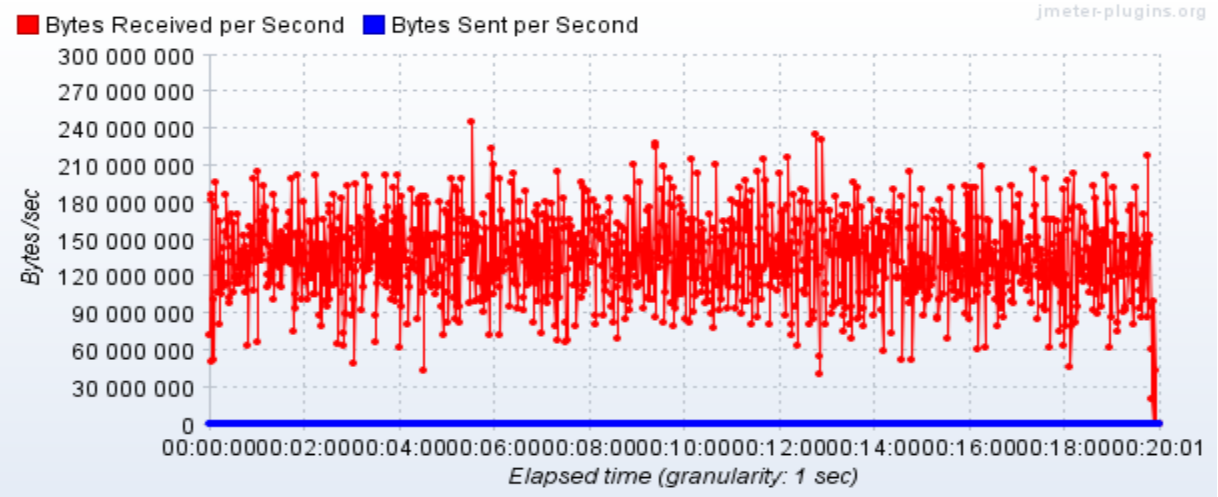

Figure 5-6 :the Throughput behaviour during the load test - Sock Shop

5.3. Resource Utilisation - Load test

Resource utilisation is another critical metric. Application resources are evaluated during the test execution.

The resources are CPU utilisation, Memory utilisation, and disk I/O utilisation. Graphical representation of resource utilisation shows the best way to identify the bottleneck. Resource utilisation is measured as percentage. The percentage calculated out of the total resources is available in the application server. This section discusses resource utilisation results and highlights the key observations.

5.3.1 Cluster CPU utilisation:

The cluster is the highest level of the microservice architecture. The resource utilisation of the cluster did not show any significant deviation. However, 20% of CPU utilisation was shown at the cluster level in terms of percentage.

Figure 5-7 : Shows the cluster CPU utilisation.

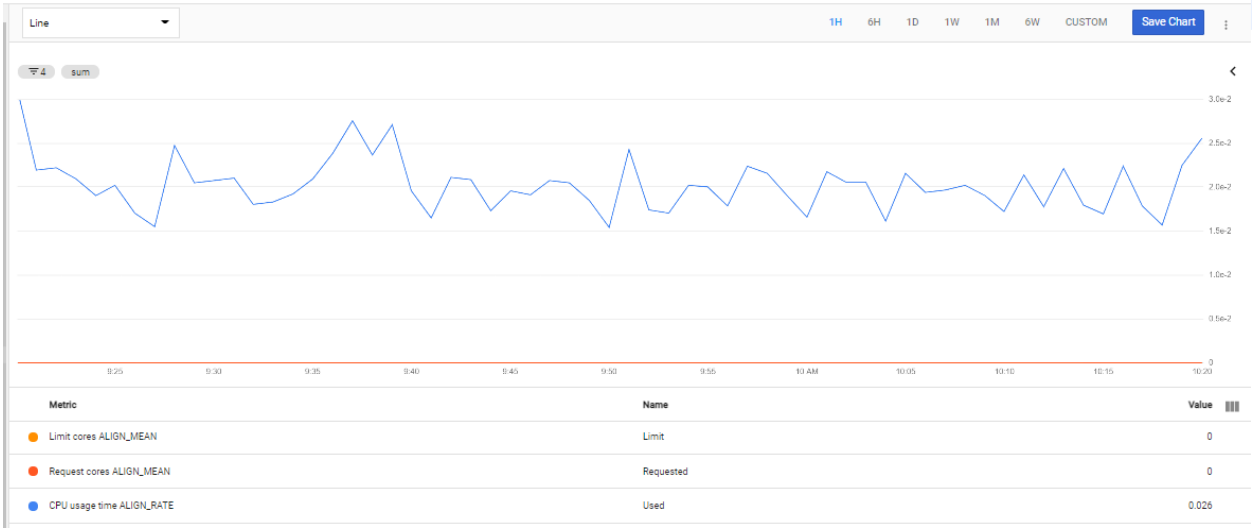

Figure 5-7:Cluster CPU utilisation, Load Test

The Cluster consists of Nodes for each service. Node level shows the resource utilisation of each service.

Node level has not shown a resource utilisation deviation. The Node level Resource utilisation illustrate in Figure 5-8, Figure 5-9, Figure 5-10.

Figure 5-8, GKE - Node 01 - Resource Utilisation is shown below.

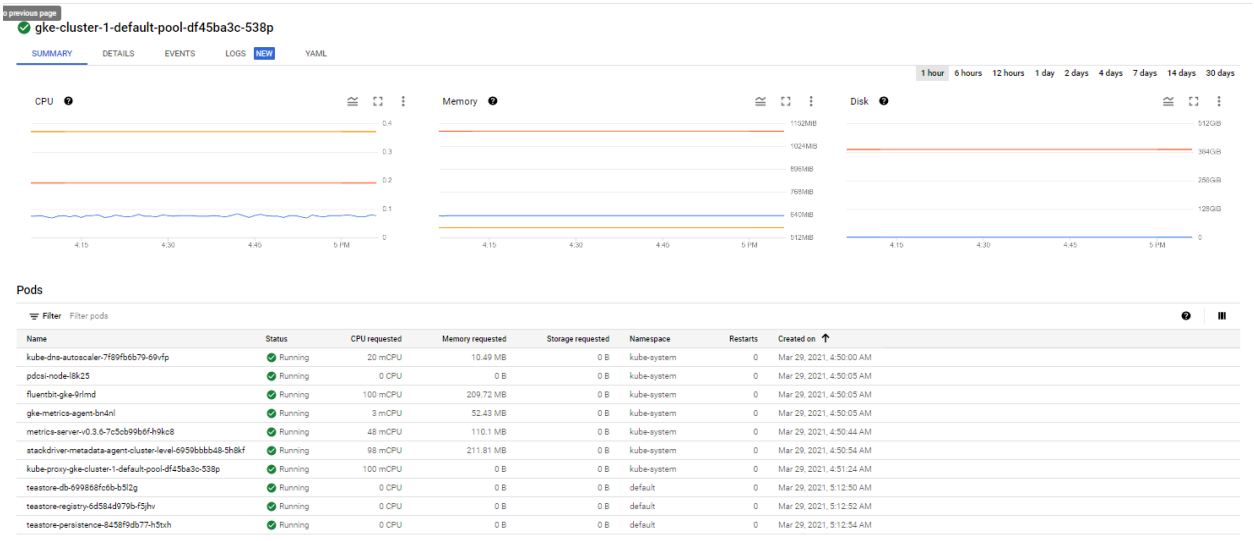

Figure 5-8 : Node level Resource utilisation: Node 1

Figure 5-9, GKE - Node 02 - Resource Utilisation is shown below.

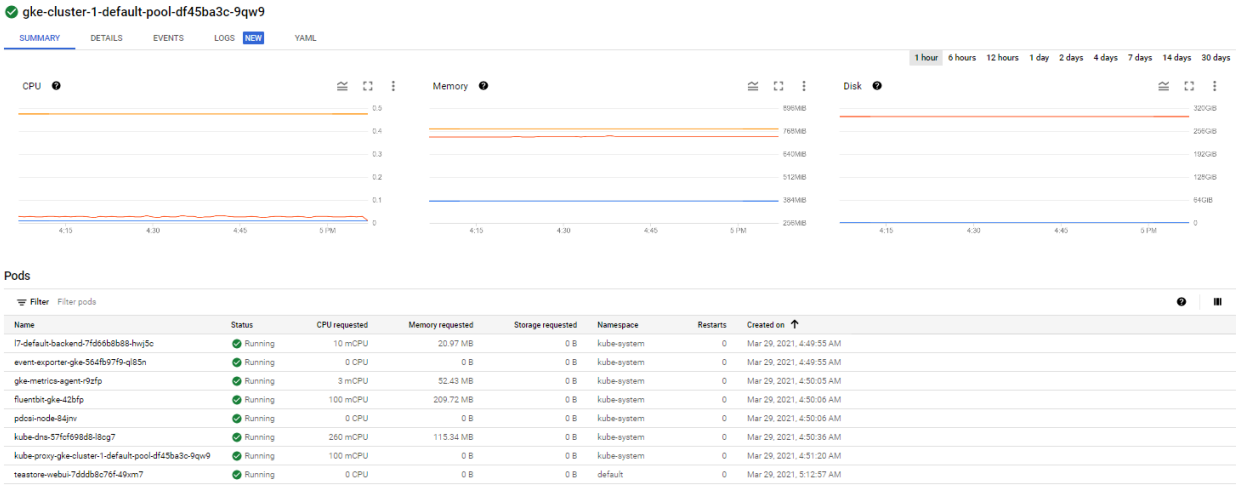

Figure 5-9 :Node Level Resource utilisation : Node 2

Figure 5-10, GKE - Node 03 - Resource Utilisation is shown below.

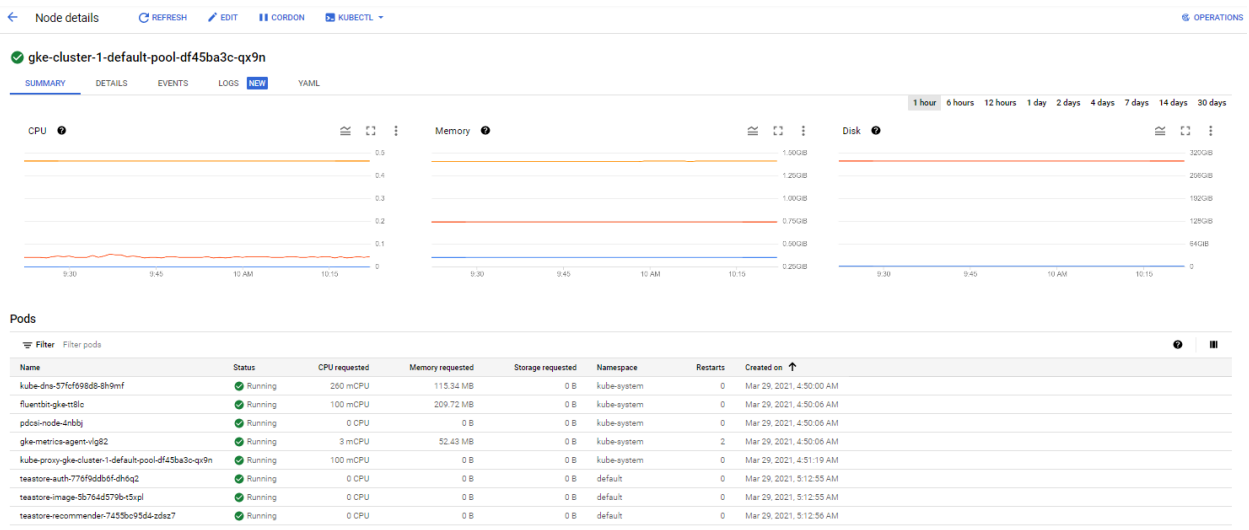

Figure 5-10 : Node Level Resource Utilisation :Node 03

Service level resource utilisation are shown in Figure 5-11: TeaStore registry and Figure 5-12 - TeaStore Database.

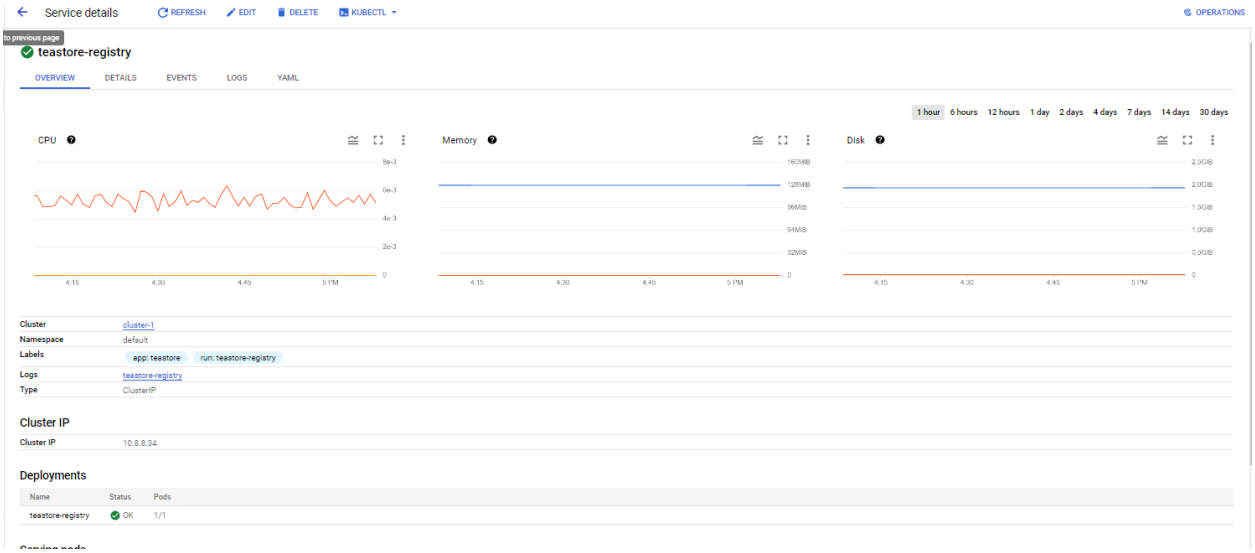

Figure 5-11:Resource Utilisation of Registry Services: Load Test

Database service resource utilisation is shown in the Figure 5-12.

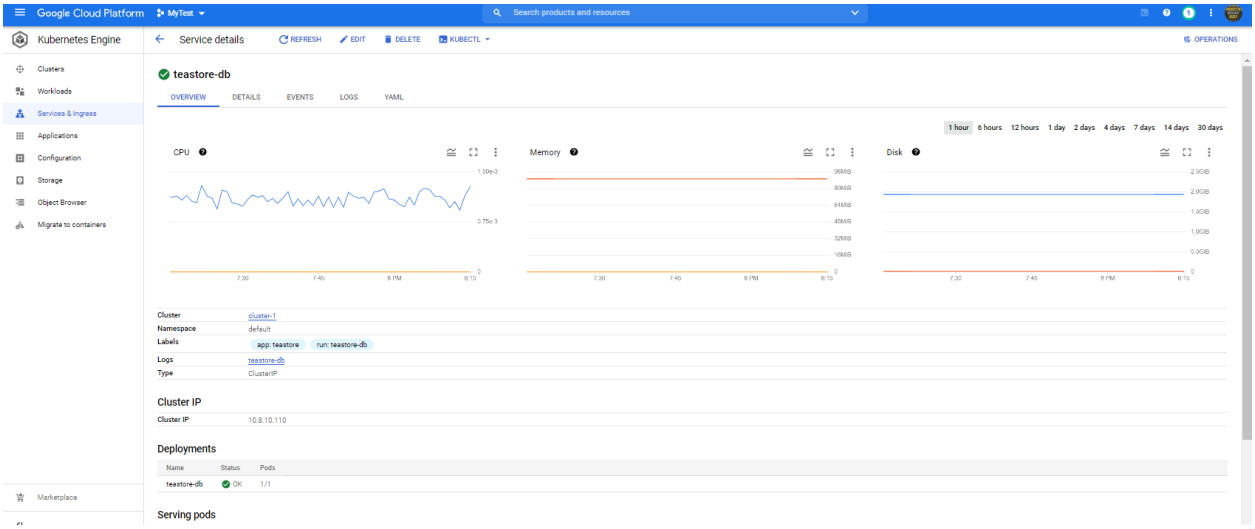

Figure 5-12 :Resource Utilisation of DB Services: Load Test

5.2. Evaluation of Endurance Test Results

The purpose of the endurance test is to get a better idea of the availability of the application round the clock. Two endurance tests on TeaStore and the Sock Shop applications understand the behaviour of microservice-based applications.

The duration of the test was around three and a half hours and during it, the server resources and the metrics were monitored.

The main observation was that the standard deviation was comparatively higher than the load test.

5.2.1 Response time for TeaStore - Endurance Test

Other observations were,

- Standard deviations around 90
- The average response time is around: 325 milliseconds

The table below shows the number of samples, average response time, and the standard deviation of the TeaStore.

Workflow	Sample	Average(milliseconds)	st.Dev
Add Product to Cart	41573.00	324.00	85.29
Home	41581.00	326.00	91.14
List Products	41574.00	324.00	84.77
List Products with different page	41573.00	323.00	85.27
Login	41579.00	324.00	84.97
Logout	41572.00	323.00	84.12
Look at Product	41574.00	323.00	84.53

Table 5-5:Endurance Test Summary - TeaStore

The figure shows the response time behaviour graphically.

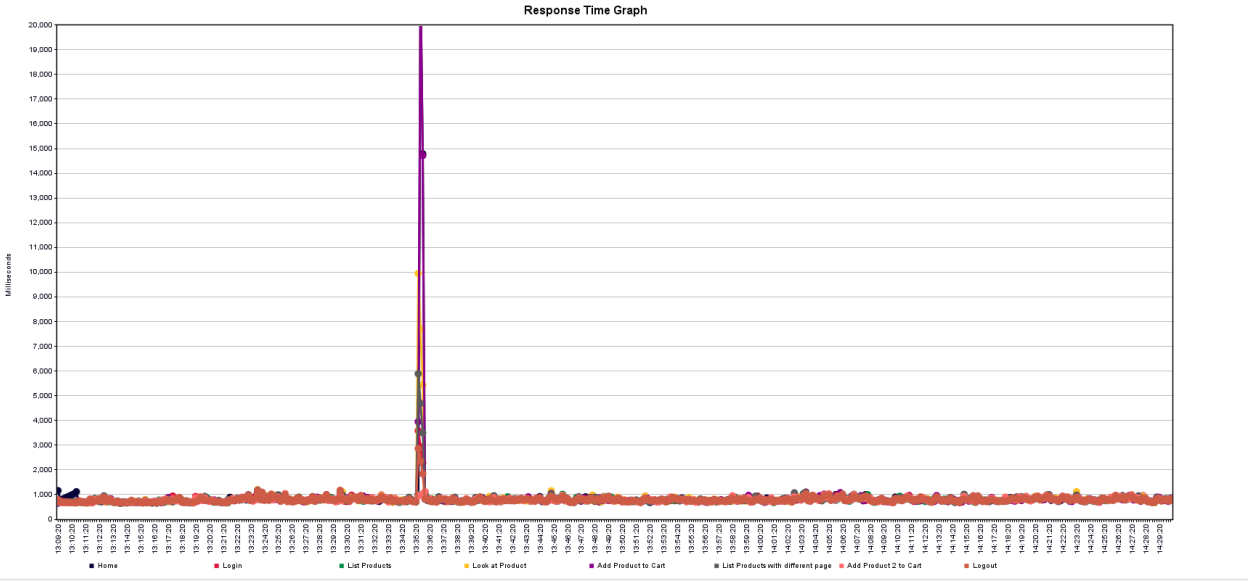

Figure 5-13 Response Time Behaviour of Endurance Test

A considerable spike is seen in the response time graph. According to the error information, service unavailability was the primary root cause for the spike.

Figure 5-13 shows the response time graph with the spike. Such Service unavailability is widespread in the performance test execution with Microservice-based applications, and it is the most crucial aspect to consider to get to a conclusion on the reliable metrics.

Figure 5-14 , shows an error details during the spike.

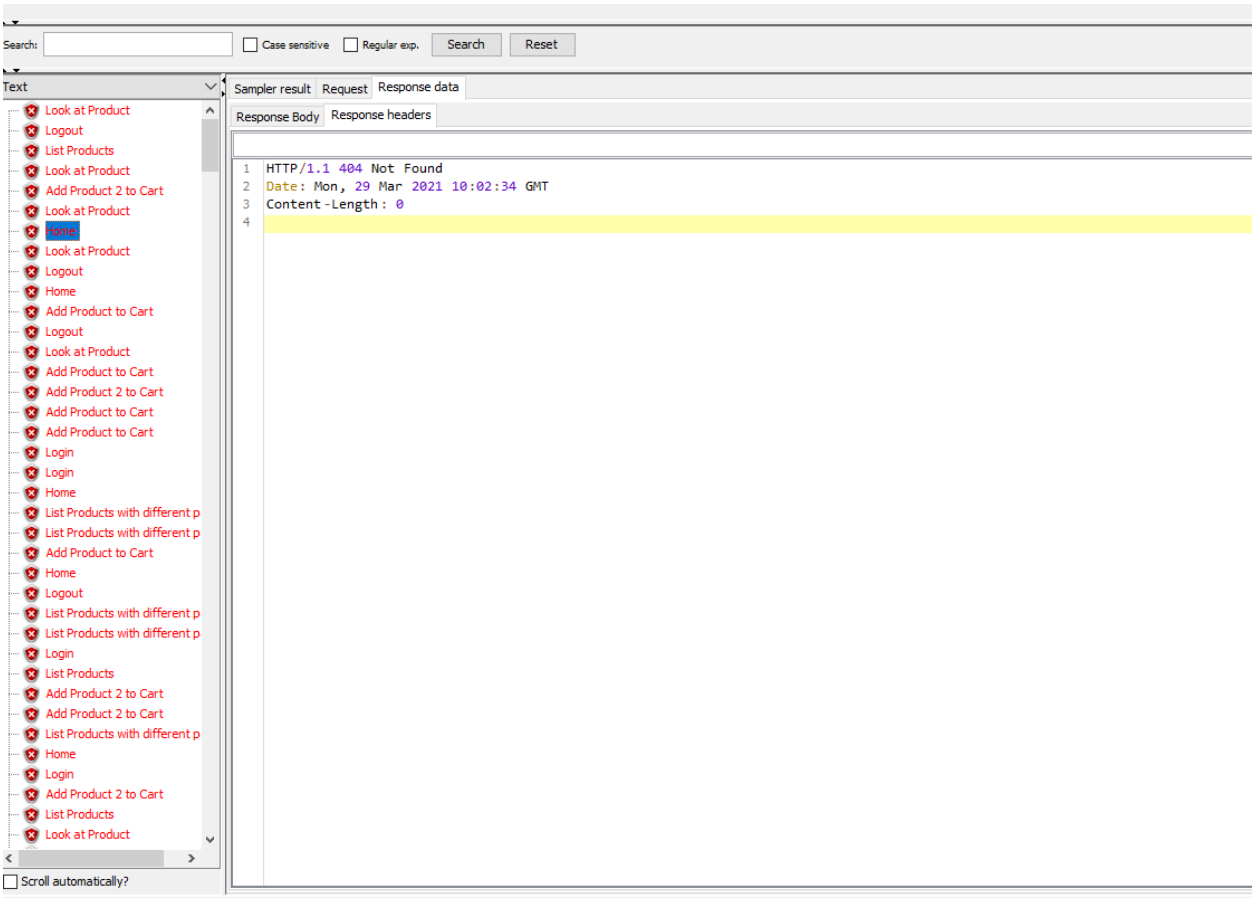

Figure 5-14 : Endurance Test - Error Log

5.2.2 Response Time for Sock Shop- Endurance Test

Table 5-6, shows the response time related to the observations of the Sock Shop endurance test. It consists of standard deviation, average response time and the number of samples.

Workflow	Sample	Average	st.Dev
Login	29539	1330	68911.94
Top bar	29534	313	68.15
Navbar	29532	311	62.75
Footer	29532	310	53.50
Navigation	29532	935	151.02

catalogue? Size=5	29532	310	54.43
Catalogue/size? Tags=	29530	310	59.16
Catalogue	29530	621	98.56
Cart	59058	310	65.01
category.html	29523	311	54.60
Category	29529	311	54.78
Tags	59046	313	67.67
Test	29523	3421	372.15

Table 5-5 : Response Time summary for Sock Shop

5.2.2 Throughput for TeaStore - Endurance Test

The TeaStore shows an average throughput of 3.85/sec for each transaction. No significant deviation was identified in terms of Throughput, a comparatively less value than the load test.

The Throughput for each transaction is shown in figure 5-7.

Workflow	Throughput
Add Product 2 to Cart	3.85
Add Product to Cart	3.85
Home	3.85
List Products	3.85
List Products with different page	3.85
Login	3.85
Logout	3.85
Look at Product	3.85

Table 5-6 :Throughput Summary - Endurance Test -TeaStore

5.2.3 Throughput for Sock Shop - Endurance Test

The Sock shop shows an average throughput of 2.79/sec for each transaction. Throughput has not shown a significant deviation throughout the test. Comparatively less value than the load test.

Workflow	Throughput
index.html	2.79
Login	2.79
Topbar	2.79
Navbar	2.79
Footer	2.79
Navigation	2.79
catalogue?size=5	2.79
catalogue/size?tags=	2.79
Catalogue	2.79
Cart	5.59
category.html	2.79
Category	2.79
Tags	5.59
Test	2.79
TOTAL	44.70

Table 5-7 :Throughput Summary - Endurance Test - Sock shop

5.5. Comparisons - Load test statistics vs endurance test statistics

Below tables show the comparison of metrics with the different load patterns, mainly the response time and the throughput.

Table 5-8 illustrates the comparison of the load test and the endurance test metrics for TeaStore.

Workflow	Load Test		Endurance test	
	Average response time	Std Deviation	Average response time	Std Deviation
Home	357	52.17	406	400.51
Login	354	39.34	407	419.74
List of Products	356	47.73	405	382.26
Look at product	355	51.23	408	493.51
Add product to cart	354	41.53	407	428.83
List product with different page	354	41.9	408	426.3
Add products to cart	354	52.45	407	386.66
Logout	354	42.8	407	401.64
	Throughput		**Throughput**	
Home	3.5		3	
Login	3.5		3	
List of Products	3.5		3	
Look at product	3.5		3	
Add product to cart	3.5		3	
List product with different page	3.5		3	
Add products to cart	3.5		3	
Logout	3.5		3	

Table 5-8:Comparisons - Load test statistics vs Endurance test statistics -TeaStore

Table 5-9 illustrates the comparison of the load test and the endurance test metrics for Sock Shop.

Workflow	Load Test		Endurance test	
	Average Response Time	St.Dev	Average Response Time	St.Dev
index.html	940	60	920	79
Login	940	60	920	79
Topbar	321	50	309	64
Navbar	319	43	308	62
Footer	317	37	307	56
Navigation	957	91	926	152
catalogue?size=	318	40	307	58

catalogue/size?tags=	318	41	307	56
Catalogue	636	67	615	103
Cart	318	40	308	58
category.html	319	41	308	59
Category	318	44	308	59
Tags	320	50	309	61
	Throughput		**Throughput**	
index.html	2.73		2.79	
Login	2.73		2.79	
Topbar	2.73		2.79	
Navbar	2.73		2.79	
Footer	2.73		2.79	
Navigation	2.73		2.79	
catalogue?size=5	2.73		2.79	
catalogue/size?tags=	2.73		2.79	
Catalogue	2.73		2.79	
Cart	5.47		5.59	
category.html	2.73		2.79	
Category	2.73		2.79	
Tags	5.46		5.59	

Table 5-9 : Illustrate the comparison of load test and the endurance test metrics for Sock Shop

5.6. Stress Test Result and Metrics Evaluation

The purpose of a stress test is to identify abnormal behaviour under a gradually increasing load. The test starts with the initial load and a peak load (1x) expected to have with the application and then tested for 1.5x and 2x by gradually increasing the load. The number of parallel threads expected to reach is 200.

5.6.1 Stress Test - TeaStore Observations :

The stress test was started with ten threads and increased up to 200 users. The key metrics were response time, throughput, and resource utilisation.

Key observations were,

- Standard deviations around 504 to 740 range
- The average response time is around: 466 to 1067 seconds
- The error rate is approximately 0.1%

5.6.1.1 Stress test Response time - TeaStore

Figure 5-10 shows the response time details observed during the stress test.

Workflow	Sample	Average
Home	193377	1067
Login	193331	466
List Products	193307	487
Look at Product	193286	483
Add Product to Cart	193259	483
List Products with different page	193238	484
Add Product 2 to Cart	193215	483
Logout	193193	485
TOTAL	1546206	555

Table 5-10 :Response Time Summary – TeaStore

5.6.1.2 Stress Test - Sock Shop - Observations :

The test started with ten threads as TeaStore and gradually increased to 200 threads. The critical metrics were response time, throughput, and resource utilisation on application servers.
Key observations made on the TeaStore stress test were,

- Standard deviations around 389 to 1045 range
- The average response time is around: 389 to 1045 seconds
- The error rate is approximately 0.1%
- Throughput: 9.9/sec - Comparatively less value than the load test

5.6.1.3 Stress Test Response Time - Sock Shop

The table 5-11, shows the response time details observed during the stress test.

Workflow	Sample	Average
index.html	173307	1023
Login	173286	1200
top bar	173259	400
navbar	173238	389
footer	173215	401
navigation	173193	1045
catalogue?size=5	170175	389
/catalogue/size?tags=	173377	389
catalogue	173331	789
Cart	340350	387
category.html	173193	345
category	163190	367
Tags	340350	390

Test	340350	4590

Table 5-11 : Stress Test Response Time -Sock Shop

The response time's graphical representation is easier to understand than the text-based values. Figure 5-15 shows the response time behaviour during the test. The interval of the sample is 10,000 milliseconds.

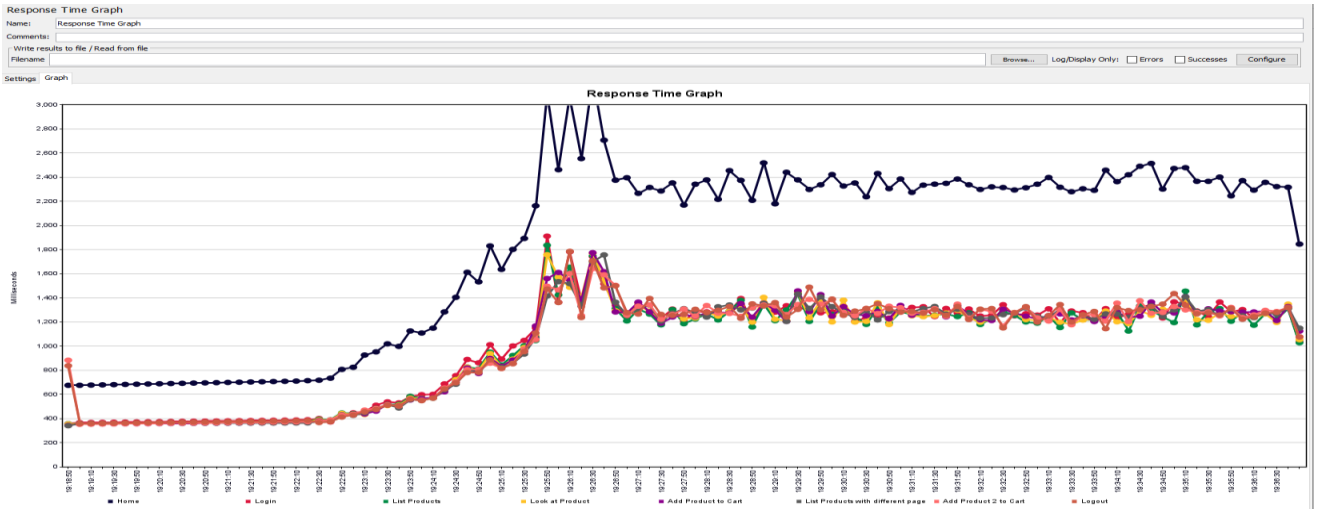

Figure 5-15:Overview of Response time : Stress test

The second graph shown in Figure 5-16 displays response time over time, at a more granular level. However, both graphs do not indicate a significant behavioural change in response to time

patterns.

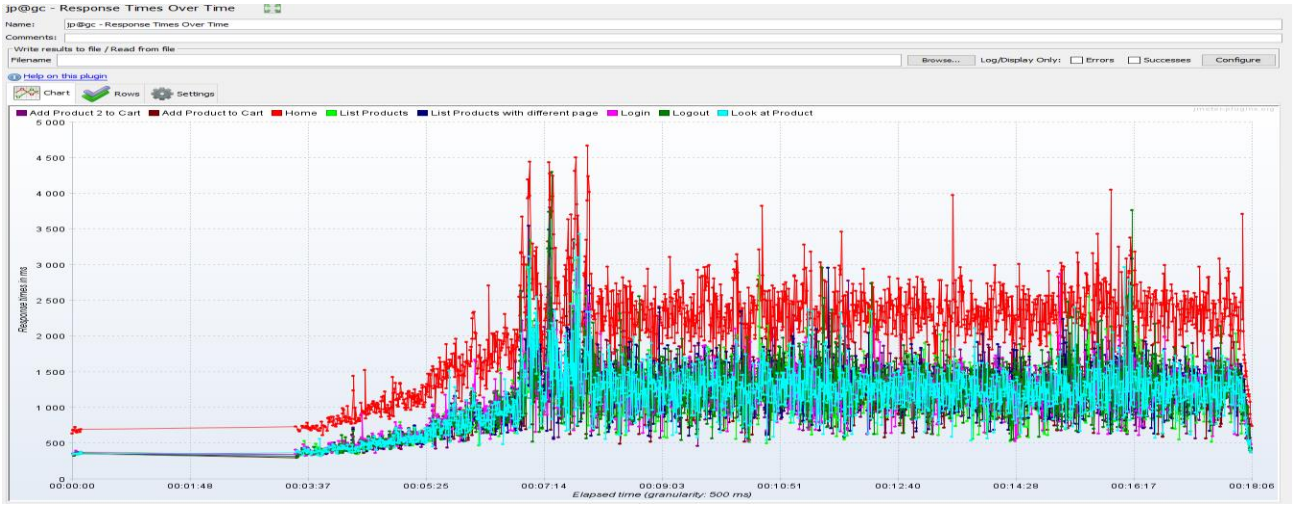

Figure 5-16 :Response Time Over Time

Refer to figure 5-17 for the active thread graphs which show the thread count behaviour during the test. Each Thread implies a user and an active thread for a given moment shows the load generation on the application in terms of users.

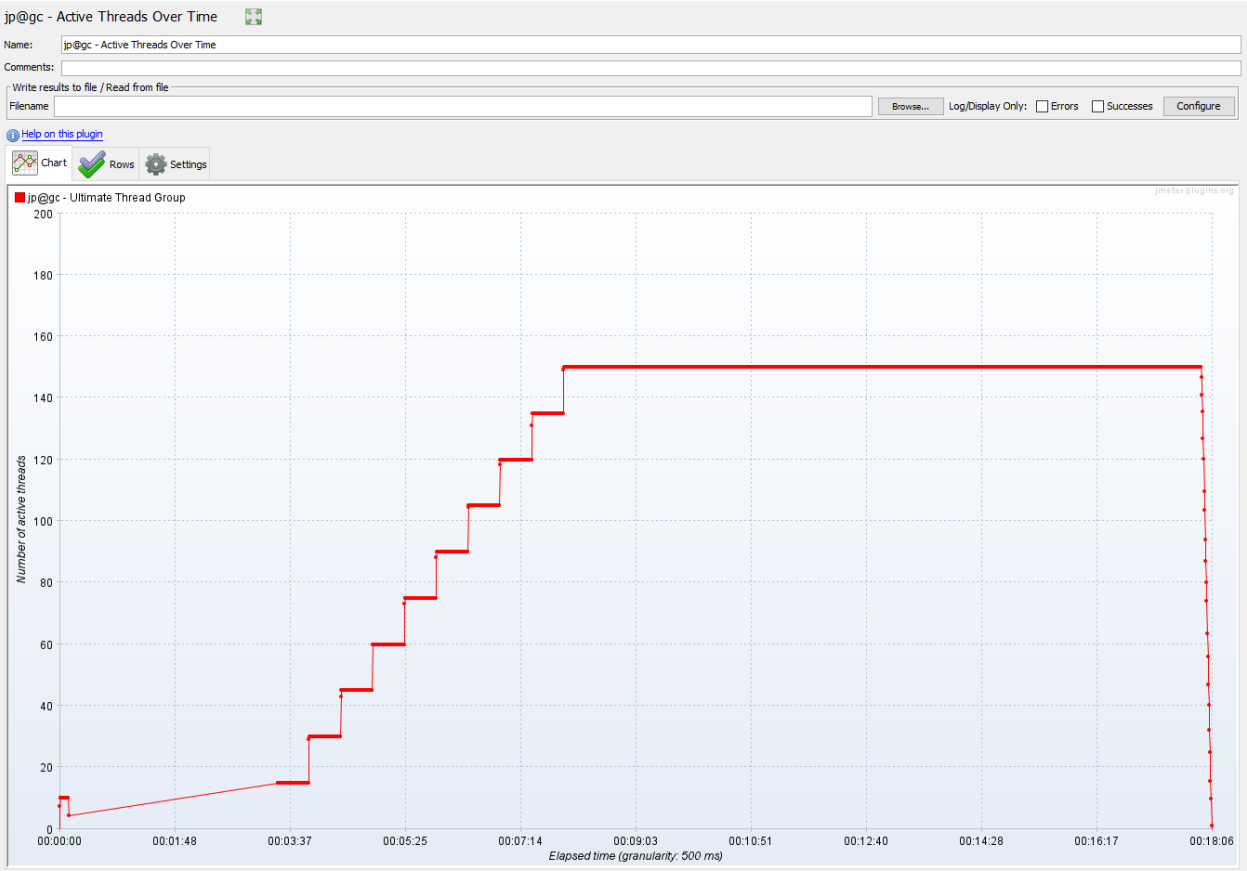

Figure 5-17: The active threads graphs

The test was kept running even after most users ramped down from the performance test plan. The graph in Figure 5-18 shows a load execution, and the test was executed for 20 minutes. Therefore, the force stop option had to be used to stop the test, and this is an abnormal behaviour that occurred with the difficulty of recovering the services.

Figure 5-18 :Response Times over Time :After the Test : Stress Test

In addition to the above graph, a high response time was observed during the test. The high response behaviour times is shown in figure 5-19.

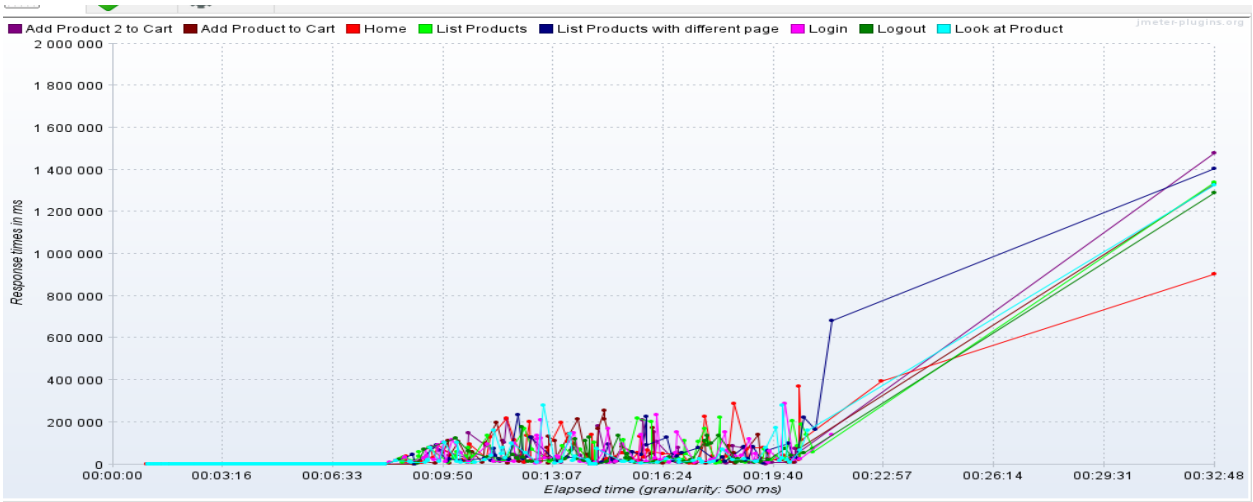

Figure 5-19 : Response Times over Time 2 : After the Test : Stress Test

All the errors are related to the GKE exception and service unavailability. Figure 5-20 is shown the error details.

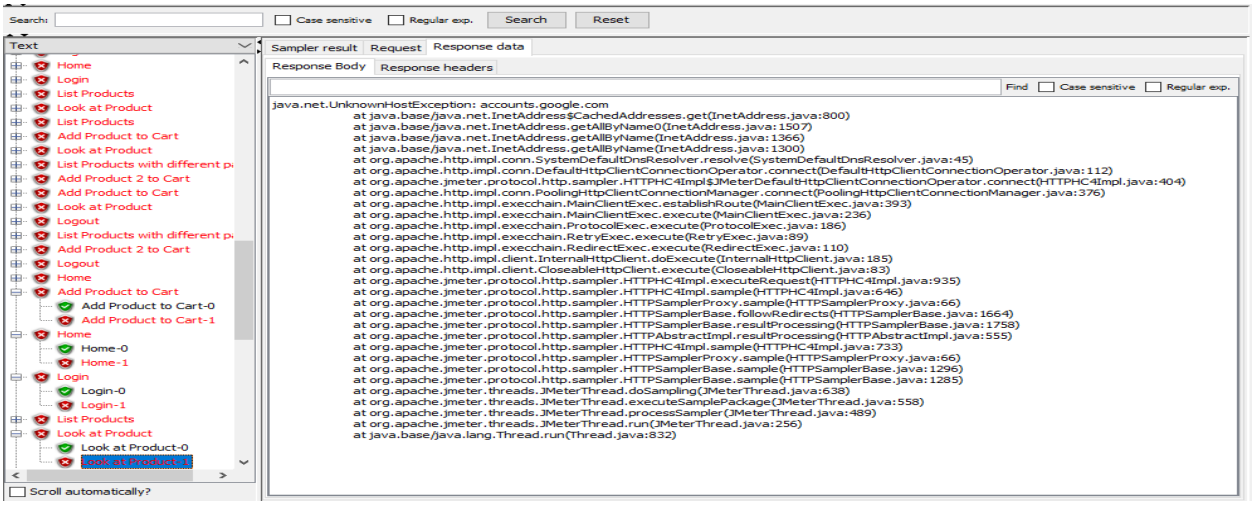

Figure 5-20 : Error details : Stress Test

Figure 5-21 is shown the Throughput over time.

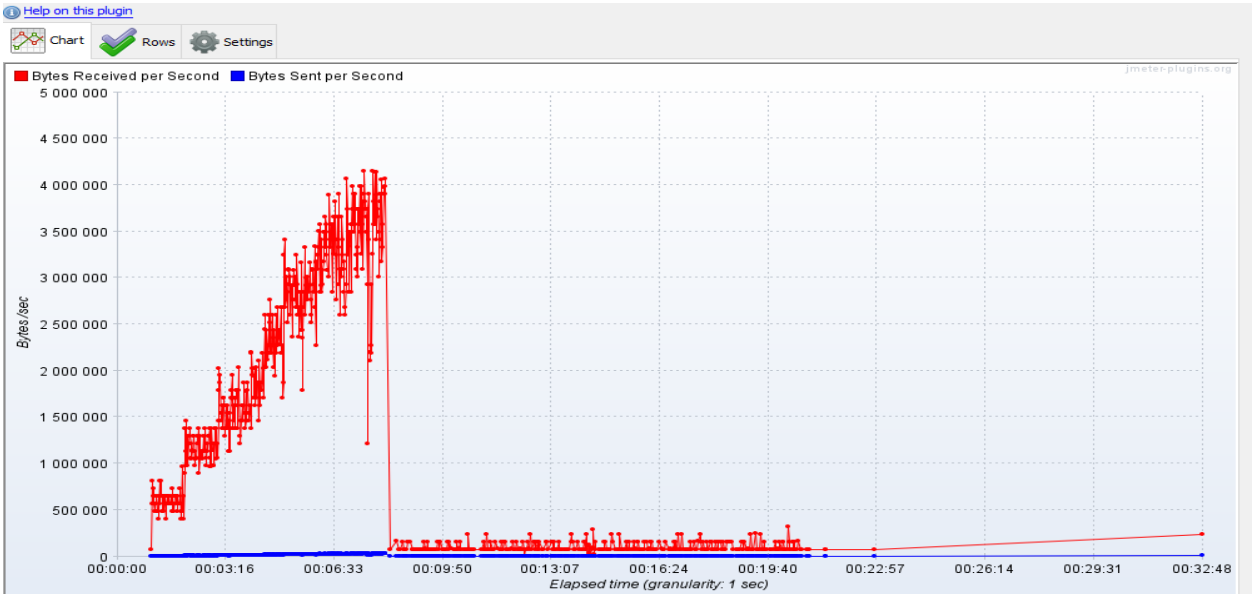

Figure 5-21:Throughput over time

5.7. Overall Resource utilisation

The order of tests was sequential, and no issue was observed in terms of resource utilisation patterns.

Overall node CPU utilisation is shown in Figure 5-22, and each node has a colour code.

Node 1: Red: 17.508KB/s (Throughput)

Node 2: Green: 3.776 KB/s (Throughput)

Node 3: Blue: 4.120 KB/s (Throughput)

The most used node is the first node. The overall throughput of the node is 17.50KB/s, which is comparatively higher than the other two nodes.

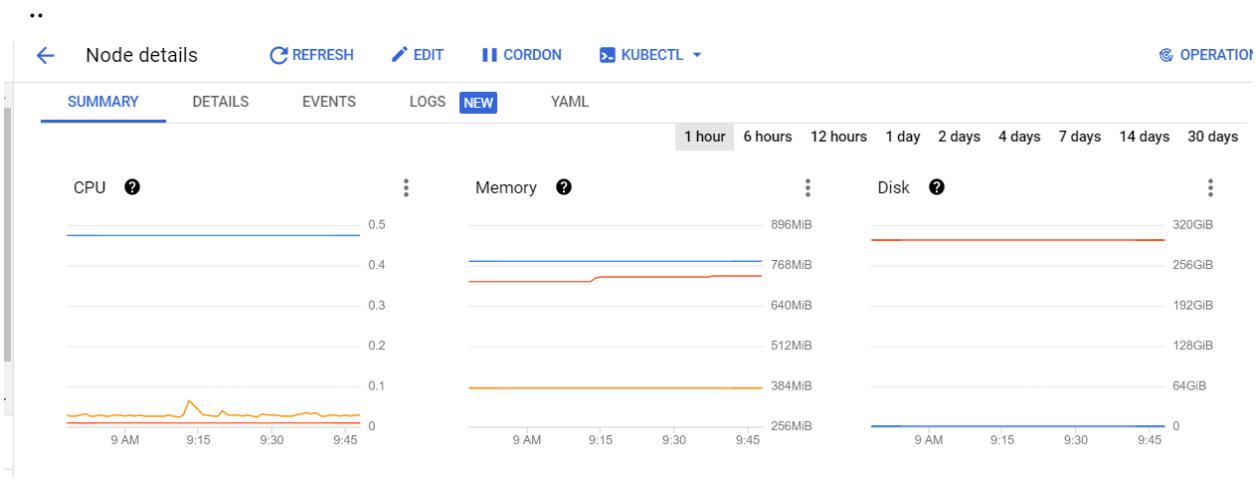

Figure 5-22 : Overall usage : Node Level

6. CONCLUSION

6.1. Chapter Overview

Due to the advanced benefits of microservices technologies, microservices-based software development has become a promising paradigm for building scalable, large-scale software systems. The microservice-based systems have inbuilt features to improve the application's performance and scalability. For example, The manual overhead of maintaining load balancing is less in microservice-based applications.

Conducting performance testing in microservice-based applications is more challenging than accomplishing testing tasks.

Performance testers have researched to rectify issues related to performance testing with microservice-based applications. This research is an extension of the current research work. The research aims to identify the best way to reduce the variation in performance test execution and it proposes more stable performance metrics that are more reliable in microservice-based applications.

TeaStore and Sock Shop were used for the experimental executions. Google Kubernetes cluster is the deployment environment for both applications. Both applications used a similar load generation pattern.

The most important findings of our paper are outlined as follows:

- Regardless of any factor, the best way to evaluate performance behaviours is to analyse the patterns. Graphical representation makes it easy to identify the fluctuation and deviation caused by errors.
- The response time was the primary metric monitored during the test. Hypothetically, the Average needs to be calculated only with successful threads. However, average response time was not a reliable metric to measure response time. The reasons are as follows:
    - Accumulated time taken to errors. Such time could be higher than the average response time.

    - If the initial step of the workflow fails, then the number of threads for the initial step becomes higher than the threads for other steps.

- The 90% percentile is the most suitable response time metric as it removes the time taken for extreme case scenarios, which mainly occur due to service unavailability.
- The distributed nature of the microservices-based application causes higher network interaction, and network latency significantly impacts performance more results than in a monolithic application.
- Comparatively, throughput was a fair metric to identify the performance behaviour with the distributed behaviour.
- The load, endurance, and stress tests were the most suitable load patterns for the experimental setup. In conclusion, all three tests are ideal for performance testing of microservice-based applications.
- Identifying the Peak load for the application and setting up the required server resource is the crucial precaution required to reduce the variance in each test execution.
- Keeping auto-provisioning off is the best testing setup.
    - Microservice-based load balancing is a reliable service in the microservices platform.
    - Keeping the auto-provisioning ‘on’ might reduce the possibility of identifying the application-level resource optimization issues.
- Load testing, endurance testing, and Stress tests were the load patterns. The test result shows identical results with both applications and with each scenario. Also, variants in each cycle were minor as we kept the auto-provisioning disabled and requested resources for peak load, and generated identical load for both applications.
- Resource monitoring is a crucial factor. The most suitable way to monitor the CPU and memory was tracking at the node level and cluster-level resource utilisation that is accumulated for all services.
- Tracking error logs also could give a lot of indication of performance bottlenecks. Logs can be application servers as well as JMeters error logs.

- Errors are more likely to occur.
- Service unavailability is the situation.

6.2. Future Work

Future studies should investigate the execution within the application as well as in different technology platforms. The variance in performance test execution should be reduced to identify the stable metrics. Even though the effort put in here contributes to identifying the most suitable matrices, it can be further extended with different technologies and different platforms.
Also, we have executed a few AI algorithms to identify the relationship between the results of the performance test. There is scope to extend further research with AI-based evaluation of results to come up with tangible outcomes for the research.

7. REFERENCES

[1] S. Eismann, C.-P. Bezemer, W. Shang, D. Okanović, and A. van Hoorn, “ CXA in Proceedings of the ACM/SPEC International Conference on Performance Engineering, 2020.

[2] J. von Kistowski, S. Eismann, N. Schmitt, A. Bauer, J. Grohmann, and S. Kounev, “TeaStore: A microservice reference application for benchmarking, modeling and resource management research,” in 2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), 2018.

[3] “Kubernetes - Google Kubernetes Engine (GKE),” Google.com. [Online]. Available: https://cloud.google.com/kubernetes-engine. [Accessed: 06-Apr-2021].

[4] “Cluster autoscaler,” Google.com. [Online]. Available: https://cloud.google.com/kubernetes-engine/docs/concepts/cluster-autoscaler. [Accessed: 06-Apr-2021].

[5] “Top 10 cloud computing research topics in 2020 - GeeksforGeeks,” Geeksforgeeks.org, 26-Sep-2020. [Online]. Available: https://www.geeksforgeeks.org/top-10-cloud-computing-research-topics-in-2020/. [Accessed: 06-Apr-2021].

[6] “Configuring a Google Kubernetes Engine cluster for AI Platform Pipelines,” Google.com. [Online]. Available: https://cloud.google.com/ai-platform/pipelines/docs/configure-gke-cluster. [Accessed: 06-Apr-2021].

[7] “GKE overview,” Google.com. [Online]. Available: https://cloud.google.com/kubernetes-engine/docs/concepts/kubernetes-engine-overview. [Accessed: 06-Apr-2021].

[8] Wikipedia contributors, “Client–server model,” Wikipedia, The Free Encyclopedia, 03-Apr-2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Client%E2%80%93server_model&oldid=1015755070. [Accessed: 06-Apr-2021].

[9] Researchgate.net. [Online]. Available: https://www.researchgate.net/publication/288835218_The_Design_and_Execution_of_Performance_Testing_Strategy_for_Cloud-based_System. [Accessed: 06-Apr-2021].

[10] “Standard cluster architecture,” Google.com. [Online]. Available: https://cloud.google.com/kubernetes-engine/docs/concepts/cluster-architecture. [Accessed: 06-Apr-2021].
[11] “Standard cluster upgrades,” Google.com. [Online]. Available: https://cloud.google.com/kubernetes-engine/docs/concepts/cluster-upgrades. [Accessed: 06-Apr-2021].
[12] Software Testing Genius, “Software Testing Genius,” 2003.
[13] “Software Testing,” Geeksforgeeks.org, 10-May-2019. [Online]. Available: https://www.geeksforgeeks.org/software-testing-endurance-testing/. [Accessed: 06-Apr-2021].
[14] “Stress testing guide for beginners,” Softwaretestinghelp.com, 14-Feb-2019. [Online]. Available: https://www.softwaretestinghelp.com/stress-testing/. [Accessed: 06-Apr-2021].
[15] K. Rungta, “What is Spike Testing? Learn With Example,” Guru99.com, 01-Jan-2020. [Online]. Available: https://www.guru99.com/spike-testing.html. [Accessed: 06-Apr-2021].
[16] Team Merlin, “Web Performance Testing — DCube’s Practices,” Government Digital Services, Singapore, 22-Nov-2019. [Online]. Available: https://blog.gds-gov.tech/web-performance-testing-dcubes-practices-fbbc20606000. [Accessed: 06-Apr-2021].
[17] A. Stringfellow, “A complete guide to performance testing types: Steps, best practices, metrics, and more,” Dzone.com, 29-Apr-2017. [Online]. Available: https://dzone.com/articles/a-complete-guide-to-performance-testing-types-test. [Accessed: 06-Apr-2021].
[18] C. M. Aderaldo, N. C. Mendonca, C. Pahl, and P. Jamshidi, “Benchmark requirements for microservices architecture research,” in 2017 IEEE/ACM 1st International Workshop on Establishing the Community-Wide Infrastructure for Architecture-Based Software Engineering (ECASE), 2017.
[19] F. Rademacher, J. Sorgalla, and S. Sachweh, “Challenges of domain-driven microservice design: A model-driven perspective,” IEEE Softw., vol. 35, no. 3, pp. 36–43, 2018.
[20] P. Jamshidi, C. Pahl, N. C. Mendonca, J. Lewis, and S. Tilkov, “Microservices: The journey so far and challenges ahead,” IEEE Softw., vol. 35, no. 3, pp. 24–35, 2018.
[21] A. Schwartz, “Microservices: Mehr als nur ein Hype?,” Inform.-Spektrum, vol. 40, no. 6, pp. 590–594, 2017.

[22] S. Newman, Building Microservices, 1st ed. Sebastopol, CA: O'Reilly Media, 2015.
[23] R. Heinrich et al., "Performance Engineering for Microservices: Research Challenges and Directions," in Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion, 2017.
[24] H. Knoche, "Sustaining runtime performance while incrementally modernising transactional monolithic software towards microservices," in Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering, 2016.
[25] N. Dragoni et al., "Microservices: Yesterday, today, and tomorrow," in Present and Ulterior Software Engineering, Cham: Springer International Publishing, 2017, pp. 195–216.
[26] T. M. Ahmed, C.-P. Bezemer, T.-H. Chen, A. E. Hassan, and W. Shang, "Studying the effectiveness of application performance management (APM) tools for detecting performance regressions for web applications: An experience report," in Proceedings of the 13th International Conference on Mining Software Repositories, 2016.
[27] M. M. Arif, W. Shang, and E. Shihab, "Empirical study on the discrepancy between performance testing results from virtual and physical environments," in Proceedings of the 40th International Conference on Software Engineering - ICSE '18, 2018.
[28] S. M. Blackburn et al., "The DaCapo benchmarks: Java benchmarking development and analysis," SIGPLAN not., vol. 41, no. 10, pp. 169–190, 2006.
[29] M. C. Calzarossa, L. Massari, and D. Tessera, "Workload characterisation: A survey revisited," ACM Comput. Surv., vol. 48, no. 3, pp. 1–43, 2016.
[30] E. Cecchet, J. Marguerite, and W. Zwaenepoel, "Performance and scalability of EJB applications," SIGPLAN not., vol. 37, no. 11, pp. 246–261, 2002.
[31] T.-H. Chen et al., "Analytics-driven load testing: An industrial experience report on load testing of large-scale systems," in 2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP), 2017.
[32] D. E. Damasceno Costa, C.-P. Bezemer, P. Leitner, and A. Andrzejak, "What's wrong with my benchmark results? Studying bad practices in JMH benchmarks," IEEE trans. softw. eng., pp. 1–1, 2019.

[33] T. F. Dullmann, R. Heinrich, A. Van Hoorn, T. Pitakrat, J. Walter, and F. Willnecker, "CASPA: A platform for comparability of architecture-based software performance engineering approaches," in 2017 IEEE International Conference on Software Architecture Workshops (ICSAW), 2017.

[34] C. Esposito, A. Castiglione, and K.-K. R. Choo, "Challenges in delivering software in the cloud as microservices," IEEE Cloud Comput., vol. 3, no. 5, pp. 10–14, 2016.

[35] K. C. Foo, Z. M. Jiang, B. Adams, A. E. Hassan, Y. Zou, and P. Flora, "An industrial case study on the automated detection of performance regressions in heterogeneous environments," in 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, 2015.

[36] R. Gao, Z. M. Jiang, C. Barna, and M. Litoiu, "A framework to evaluate the effectiveness of different load testing analysis techniques," in 2016 IEEE International Conference on Software Testing, Verification and Validation (ICST), 2016.

[37] S. He, G. Manns, J. Saunders, W. Wang, L. Pollock, and M. L. Soffa, "A statistics-based performance testing methodology for cloud applications," in Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2019.

[38] K. Li, "Quantitative modeling and analytical calculation of elasticity in cloud computing," IEEE trans. cloud comput., vol. 8, no. 4, pp. 1135–1148, 2020.

[39] Z. M. Jiang and A. E. Hassan, "A survey on load testing of large-scale software systems," IEEE trans. softw. eng., vol. 41, no. 11, pp. 1091–1118, 2015.

[40] C. Laaber and P. Leitner, "An evaluation of open-source software microbenchmark suites for continuous performance assessment," in Proceedings of the 15th International Conference on Mining Software Repositories, 2018.

[41] C. Laaber, J. Scheuner, and P. Leitner, "Software microbenchmarking in the cloud. How bad is it really?," Empir. Softw. Eng., vol. 24, no. 4, pp. 2469–2508, 2019.

[42] P. Leitner and C.-P. Bezemer, "An exploratory study of the state of practice of performance testing in java-based open source projects," in Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering, 2017.

[43] J. Scheuner and P. Leitner, “A cloud benchmark suite combining micro and applications benchmarks,” in Companion of the 2018 ACM/SPEC International Conference on Performance Engineering, 2018.
[44] W. Shang, A. E. Hassan, M. Nasser, and P. Flora, “Automated detection of performance regressions using regression models on clustered performance counters,” in Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering, 2015.
[45] P. Stefan, V. Horky, L. Bulej, and P. Tuma, “Unit testing performance in java projects: Are we there yet?,” in Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering, 2017.
[46] A. Uta and H. Obaseki, “A performance study of big data workloads in cloud datacenters with network variability,” in Companion of the 2018 ACM/SPEC International Conference on Performance Engineering, 2018.
[47] J. von Kistowski, M. Deffner, and S. Kounev, “Run-time prediction of power consumption for component deployments,” in 2018 IEEE International Conference on Autonomic Computing (ICAC), 2018.
[48] P. Xiong, C. Pu, X. Zhu, and R. Griffith, “vPerfGuard: An automated model-driven framework for application performance diagnosis in consolidated cloud environments,” in Proceedings of the ACM/SPEC international conference on International conference on performance engineering - ICPE ’13, 2013.
[49] L. Bass, I. Weber, and L. Zhu, DevOps: A Software Architect’s Perspective. Boston, MA: Addison-Wesley Educational, 2015.
[50] M. Waseem, P. Liang, and M. Shahin, “A Systematic Mapping Study on Microservices Architecture in DevOps,” arXiv [cs.SE], 2020.
[51] J. Humble and D. Farley, Continuous delivery: Reliable software releases through build, test, and deployment automation. Addison-Wesley Professional, 2011.
[52] “GitHub - microservices-demo/microservices-demo: Deployment scripts & config for Sock Shop", GitHub, 2022. [Online]. Available: https://github.com/microservices-demo/microservices-demo. [Accessed: 07- Mar- 2022]