High-Performance Serverless Request Generator: Capable of Generating a Hundred Thousand Requests Per Second

D. Mileski* and M. Gusev**
* Innovation Dooel, Skopje, North Macedonia
** Sts Cyril and Methodius University in Skopje, Faculty of Computer Science and Engineering, Skopje, North Macedonia
E-mail: dimitar.mileski@ieee.org, marjan.gushev@finki.ukim.mk

***Abstract*—Depending on the scope, performance testing systems are distributed and scalable systems that can use many distributed instances to generate workload. Server-based systems used in performance testing can not cope with flexible workloads and specifications that may generate high costs for an extensive number of generated requests. Our goal was to develop a Virtual Patient Generator for testing the electrocardiogram streaming system for monitoring thousands of patients. To deliver a fully functional workload generator of a hundred thousand requests per second, in this paper, we developed a High-Performance Serverless Request Generator as an implementation of a Distributed Serverless Workload Generator concept. The system architecture comprises Serverless services, Pub/Sub, and Cloud Storage capable to deliver a cheaper version of a flexible number of requests, being scalable to an even higher extend. We present limitations, performance, and advantages by comparing the presented serverless load testing system with other workload generators available as SaaS or distributed load testing systems based on servers.**

Keywords— high performance computing; performance testing; serverless; FaaS; distributed systems; cloud computing; ECG

I. Introduction

The performance of scalable and elastic systems is tested with workload generators. Performance testing [1] is the process of evaluating a system or application's performance characteristics, such as responsiveness, stability, and scalability, under a variety of workloads. The goal of performance testing is to identify bottlenecks and other issues that may affect the system's ability to meet its performance requirements and to ensure that the system can handle the expected levels of usage. This can include testing the system's response time, throughput, and resource utilization under different loads and conditions, as well as simulating real-world usage scenarios.

The generation of a considerable workload requires the deployment of scalable, elastic, and high-performance systems. Distributed systems may be necessary to accommodate the demands of such evaluation and testing procedures. The most common performance tests types are:

- *Load testing* [2] is a type of performance testing used to evaluate a system or application's behavior under a specific load or user demand.
- *Stress testing* [1], also known as endurance testing, is a type of performance testing that is used to evaluate a system or application's behavior under extreme loads or conditions. The goal of stress testing is to identify the upper limits of a system's capabilities and to determine how the system behaves when it is pushed beyond its normal operational limits.
- *Soak testing* [1], also known as endurance testing or reliability testing, is a type of performance testing that is used to evaluate a system or application's behavior over an extended period of time. The goal of soak testing is to identify any issues that may arise after prolonged usages, such as memory leaks or resource exhaustion, and to ensure that the system remains stable and responsive over an extended period of time.
- *Spike testing* [1] is a type of performance testing that is used to evaluate a system or application's behavior under sudden and significant changes in load or usage. The goal of spike testing is to identify how the system behaves under unexpected and extreme conditions to ensure that the system can handle sudden and significant increases in load or usage without impacting performance.

During our research, we have addressed embarrassingly parallel problems and evaluated the corresponding speedup performance [3] experimenting with virtual machine approaches versus parallel serverless threads in the CardioHPC use case. CardioHPC [4] project aims at realizing an experiment and simulating real-time processing and monitoring of electrocardiogram (ECG)s. To evaluate the performance of the ECG Streaming system for monitoring 10,000 patients for the CardioHPC project, we developed a High-Performance Serverless Request Generator (HPSRG) capable of generating hundred thousand requests per second. Evaluation of serverless ECG stream processing in federated cloud implementations was also addressed in our earlier research [5]. We have already evaluated the automatic scalability of serverless services focusing on FaaS serverless [6] that served as the basis for the development of HPSRG.

Systems for performance testing are often server-based distributed systems, and serverless architecture can provide automatic scalability as an infrastructure for the devel-

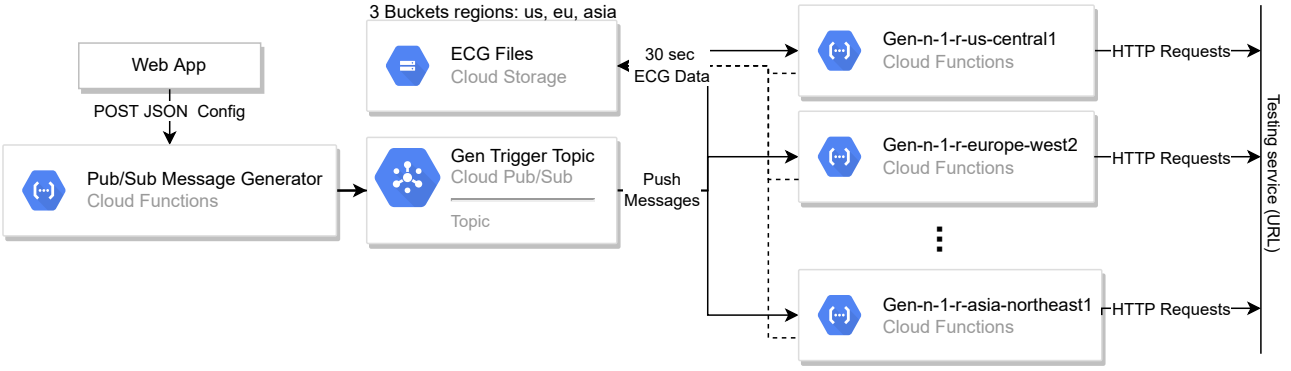


Fig. 1: System architecture: High-Performance Serverless Request Generator

opment of performance testing systems being able to generate such a large workload. Performance tests require flexible workloads and specifications, configured by a corresponding user interface.

Performance testing of CardioHPC and ECG Streaming system for monitoring of 10,000 patients uses the new HPSRG system that generates a workload in the form of JSON files containing ECG data. These are then streamed to a production system for monitoring of ECG streams [7]. HPSRG architecture can be implemented and deployed in any cloud that supports serverless services FaaS or serverless containers, object storage, and implementation of Pub/Sub - Publisher Subscriber model.

The paper follows the next structure. Related work and analysis of the state-of-the-art are presented in Section II. System architecture, Experiments, and Evaluation methodology are described in Section III illustrating the approach that will achieve generating a hundred thousand requests per second. Results are evaluated in Section IV. Finally, results are discussed in Section V and conclusions presented in Section VI along with future work directions.

II. State of the art

A variety of software and tools are used for testing the performance of different applications, most of which are server-based rather than serverless. A survey of performance testing software [8] focusing on well-known performance testing systems, such as WebLOAD [9] [10], LoadNinja [11] [12], HP Load Runner [13] [14], Apache J Meter [15] [14], Selenium [16] [14], NeoLoad [17] [10], WebLoad Professional [10], LoadUI [18] [10], WAPT [19] [20], Loadster [21] [10], and LoadImpact (Load Impact is now rebrended as k6) [22] [8], highlights the significance of performance testing, including scalability, crucial for accommodating increased numbers of users, and stability, ensuring the application works reliably at all times.

WebLOAD [9] [10], NeoLoad [17] [10], WebLoad Professional [10], and WAPT [19] [20] are load testing tools for performance testing of web applications, APIs, and mobile applications supporting a wide range of protocols, technologies, real-time monitoring and result analysis. These tools identify performance bottlenecks and scalability issues.

HP LoadRunner [13] [14] is a load testing tool developed by Hewlett Packard Enterprise (HPE) to simulate user traffic and monitor the performance of web applications and APIs, supporting multiple protocols and technologies, including various web, mobile, and cloud-based applications.

Apache J Meter [15] [14] is an open-source load testing tool that can simulate user traffic and monitor the web applications and API's performance. A wide range of supported protocols and technologies provide a highly extensible environment to create custom test cases and plug-ins.

Selenium [16] is an open-source testing framework that supports functional testing of web applications and APIs. It can also be used for load testing, but it is primarily used for functional testing.

SAAS load testing tools offer cloud-based performance and scalability testing for applications through simulated user traffic [23] [24]. Some well-known SAAS load testing tools are BlazeMeter [25] [26], LoadImpact [22] [8], LoadRunner Cloud [13] [26], Loader.io [27], Gatling FrontLine [28] [12], Neoload [17] [10], Flood.io [29], OctoPerf [30], LoadUI Pro Cloud [18] [8], and Sauce Labs [31] [32].

BlazeMeter [25] [26], LoadNinja [11] [12], LoadUI [18] [8], Loadster [21] [10], LoadImpact [22] [8], Gatling FrontLine [28] [12], Neoload [17] [10], Flood.io [29], OctoPerf [30], LoadUI Pro Cloud [18] [8], and Sauce Labs [31] [32] are all cloud-based load testing tools that evaluate the performance of websites, mobile applications, and APIs by simulating large amounts of user traffic, providing real-time monitoring and results analysis with drag-and-drop interfaces to create and manage load tests. LoadRunner Cloud [13] [26] is a cloud-based performance testing solution offered by HPE. Loader.io [27] is a cloud-based load testing platform that offers an easy-to-use interface for load testing websites and APIs.

The HPSRG system offers a distinct advantage over the conventional distributed testing systems through its use of serverless technology, which eliminates the need for server management and significantly reduces deployment

time. The utilization of a serverless pay-as-you-go system offers a more precise payment model as users are solely charged for resources used per request, while in the case of virtual machines, the users are charged for the entire duration of usage, even during periods when performance tests are not executed.

III. Methods

In this section, we present the system architecture of a new High-Performance Serverless Request Generator, and then specify the experimental and evaluation methodology.

A. Solution Architecture

HPSRG is a solution based on serverless technology, and the specific implementation in this paper is based on Google Cloud [33]. In implementing the solution architecture presented in Fig 1, we have utilized Google Cloud Functions Generation 1 [34] Function-as-a-Service (FaaS) offering, in conjunction with Google Cloud Storage [35], an object storage solution, and Google Cloud Pub/Sub [36], which represents an implementation of the Publisher-Subscriber model within the Google Cloud ecosystem.

HPSRG generates requests to a Testing service (invoked by a URL) as presented in Fig 1. The Web App allows users to input configuration parameters for HPSRG through a dedicated client form. Upon submission of the form, the Web App sends an HTTP Post request containing JSON Config to start the HPSRG.

Configurations for HPSRG include Pub/Sub Message Generator (PMG) execution time, Pub/Sub topic name, and Google Cloud project ID. The configuration also includes parameters for individual Generators (Gen) (eg: Gen-1-r-us-central1, Gen-n-1-r-europe-west2, ... Gen-n-1-r-asia-northeast1) such as a number of requests per second, execution time in seconds, and Testing service (URL).

The web application makes an HTTP POST request to FaaS serverless function, which initiates the Pub/Sub Message Generator with Cloud Pub/Sub Topic. All Gen generators (Gen-n-1-r-us-central1, Gen-n-1-r-europe-west2 ...) have a subscription to a topic named "Gen Trigger Topic" being a Push or Pull request in Google Cloud Pub/Sub.

Table I presents the configuration of each FaaS serverless function. The automatic scalability of a FaaS serverless function is an important feature of the HPSRG architecture. Performance testing will reveal a sufficient number of active FaaS instances for each generator. According to Google Cloud Function Generation documentation [34], one FaaS can activate up to 3000 instances and each instance will generate requests to Testing service (URL), currently a high limit of all other serverless providers.

Note that the Testing service (URL) in Fig. 1 is a no-op function, which is a dummy placeholder in computer programming that performs no action, usually used in situations where a real function is to be implemented later or when a function must be specified but no operation is required. Although, the Testing service (URL) is a FaaS deployed in the Google Cloud europe-west-1 region but it can be deployed anywhere since the request generator developed and tested in this paper can send requests to any service.

TABLE I: FaaS Config

	PMG	Generator	Testing service (URL)
Memory	512MiB	256MiB	256MiB
vCPUs	8	1	1
Min number of instances	0	0	0
Max number of instances	1	3000	3000
FaaS Timeout	540s	540s	5s

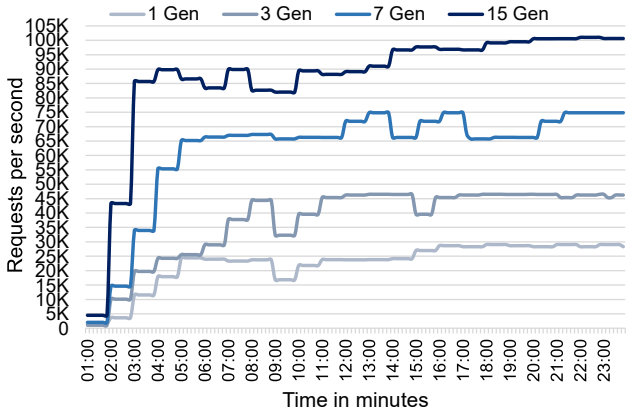
TABLE II: Cloud Regions

FaaS	Region
PMG	europe-west-1
Testing service (URL)	europe-west-1
1 Gen	europe-west-1
3 Gen	europe-west-1, asia-north-east-1, us-east-1
7 Gen	europe-west-1, asia-north-east-1, us-east-1 europe-central-2, asia-south1, us-west4, southamerica-east1
15 Gen	europe-west-1, asia-north-east-1, us-east-1, europe-central-2, asia-south1, us-west4, southamerica-east1, europe-west2, australia-southeast1, asia-southeast1, europe-west6, us-west1, us-central-1, asia-east2, northamerica-northeast1

B. CardioHPC Experiments

Our CardioHPC experiment aims at testing more FaaS serverless function generators: Gen-n-1-r-us-central1, Gen-n-1-r-europe-west2 ... Gen-n-1-r-asia-northeast1. In our use case, we use push messages, which means that when a message is published on that topic, everyone who has subscribed to that topic will receive the message following to the push principle. Generators (Gen) use ECG data prerecorded and stored in the Google Cloud Storage, as a type of Object Storage. Data in Google Cloud Storage is organized in buckets, and for the specific architecture we created three different buckets (ecg_files_eu, ecg_files_us, ecg_files_asia). The ECG data originates from the MIT-BIH [37] [38] ECG benchmark dataset, being resampled from 360 to 125 Hz and rescaled from 11 to 10-bit resolution. Each Generator selects a random ECG recording and sends a 30-second ECG signal (with 3750 ECG samples) to the specified Testing service (URL). The no-op function returns success or error based on the successful parsing of the JSON with ECG.

The HPSRG JSON Config parameters were experimentally determined to maximize the number of instances for Gen, then all 4 experiments were conducted with these parameters using a reasonable number of request generators, 1, 3, 7, and 15. Each experiment lasts 25 minutes, during which the number of requests stabilized after reaching a certain point. The deployment cloud regions of all Gen are listed in Table II.



Generators	req/s	req/min	req/h	req/day	req/month
1 Gen	30K	1.8M	108M	2.592B	77.76B
3 Gen	45K	2.7M	162M	3.888B	116.64B
7 Gen	75K	4.5M	270M	6.48B	194.4B
15 Gen	100K	6M	360M	8.64B	259.2B

Fig. 2: Number of Generators (Gen) and Requests per second hitting the Testing service (URL).

C. Evaluation methodology

To evaluate the performance of the service, various tests such as Load Tests, Stress Tests, Soak Tests, and Spike Tests require the HPSRG to generate a high volume of requests to the service. In our CardioHPC experiment we use the following evaluation metrics to test the generator:

- number of **Requests Per Second** sent to the Testing service (URL),
- number of **Active FaaS Instances** per Generator, which is used to assess if the maximum number of active instances specified by the cloud platform is being utilized,
- **cost** of running experiments on serverless services in the Google Cloud public cloud is the third evaluation metric.

The achieved results for these evaluation metrics are collected by the Google Cloud Monitoring service of Google Cloud Function Generation 1.

IV. Results

Fig. 2 presents the request rate per second directed toward the specified Testing service (URL) of the service being evaluated, as a function of varying numbers of Generators. The CardioHPC experiment was conducted for several distinct scenarios - test cases, and in this paper, for clarity of visual presentation, we present results only for a single generator ($n = 1$), $n = 3$, $n = 7$, and $n = 15$ generators.

Fig. 3 presents the active instances of each service for the HP serverless request generator system architecture (Fig. 1). Specifically, the number of active instances is provided for the following services:

- Pub/Sub Message Generator,
- one of the Generators (such as the Gen-n-1-r-europe-west1), and

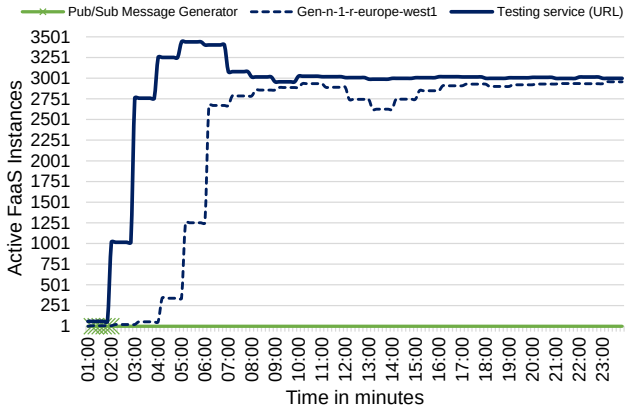


Fig. 3: Active FaaS Instances

- Testing service (URL)

V. Discussion

A. Performance analysis

The test cases within the CardioHPC experiment were conducted over a duration of 25 minutes. The initiation of the experiment (Fig. 3) for Gen-n-1-r-europe-west1 is specially marked. We observe an increase in the number of requests per second from zero to a stabilized value for all cases, upon activation of the instances from the Generators. The number of active instances increased from zero to the specified maximum of 3000, as outlined in the Google Cloud Function Generation 1 documentation.

Fig. 2 displays fluctuations of the number of requests per second until the value is stabilized, which can be attributed to the specified timeout of 540 seconds (9 minutes) for each of the Generators in Table I. This requires a new instance to be initiated upon termination, with a maximum running time of 540 seconds, as defined in the JSON Config.

These fluctuations can be mitigated through the utilization of Google Cloud Function Generation 2, which operates on the Google Cloud Run service with a maximum instance timeout of one hour. Additionally, oscillations may also occur due to network factors and the fact that the generator is a distributed system generating requests from various global locations (such as London, Las Vegas, Sao Paulo, Sydney, and Mumbai). The values are presented in terms of requests per second, but it may be more appropriate to consider them as requests per minute or hour, depending on the system being evaluated.

Fig. 2 presents the rate of requests per unit of time for different number of Generators within the HPSRG system. The values are indicated in the form of thousands, millions, and billions, with the respective units colour-coded in grey, orange, and red.

B. Limitations

The number of active instances of the Pub/Sub Message Generator (Fig. 2) shows only one active instance

which generates the messages to act as triggers for the FaaS Generators. The FaaS Generators, while generating requests for the Testing service (URL), exhibit a range of active instances from 2700 to 3000, with one instance falling below 2700. The Testing service (URL) represents the service under test, to which the HPSRG sends requests.

The stability of the number of active instances around 3000, which is the theoretical maximum according to the documentation, is noteworthy. Initially, the platform starts up to 3500 instances, which could be in response to sudden spikes in requests. However, the maximum number of instances in Google Cloud Function Generation 1 cannot be specified to be greater than 3000, despite the platform occasionally surpassing this number.

C. Scalability of HPSRG

Solutions to accommodate an increase in demand for higher rates of requests include:

- Increasing the number of Pub/Sub Message Generators and Gen Trigger Topic.
- Deploying new generators (ex: Gen-n-x-r-us-central1 ...) in the same or in other public or private clouds. Each generator uses the same codebase written in Python that can be deployed to other serverless services in other public or private clouds.

Potential limitations to scaling HPSRG, as suggested, could include restrictions from the public cloud provider in terms of resource consumption rates and quotas. However, it should be noted that such limitations are not inherent to the HPSRG architecture itself.

D. Advantages

The HPSRG system has an advantage over distributed testing systems by utilizing serverless technology, eliminating the need for server management, and reducing deployment time. It can handle any workload as HTTP Post requests and does not require the management of virtual machines to achieve high request throughput. However, HPSRG lacks result visualization tools and its Web App is only used for configuring test setups. Apache JMeter, on the other hand, is a more comprehensive system with a range of tools and plugins for visualizing performance testing results.

In the CardioHPC project, results from the HPSRG testing were obtained through the Google Cloud Monitoring [39] service and visualized using external tools specialized for ECG monitoring.

VI. Conclusion and Future Work

In this paper, we present the High-Performance Serverless Request Generator as a new solution for service performance testing. The HPSRG can generate a hundred thousand requests per second, millions of requests per minute, and billions of requests per day. This demonstrates the potential of the serverless model and automatic scalability in cloud-based performance testing. The HPSRG is exploited to evaluate the serverless ECG Streaming system for monitoring 10,000 patients in the CardioHPC experiment for real-time processing and monitoring of ECG streams for more than 10.000 patients simultaneously.

As a potential future work, we aim at offering the HPSRG as a software-as-a-service (SaaS) publicly available solution and support other configurable performance testing systems for a lower cost and scalable flexible increased throughput. It is essential to examine the operational cost of such a generator and determine if a private cloud that supports serverless services would be more advantageous.

Acknowledgement

The experiment "CardioHPC - Improving DL-based Arrhythmia Classification Algorithm and Simulation of Real- Time Heart Monitoring of Thousands of Patients" has received funding from the European High-Performance Computing Joint Undertaking (JU) through the FF4EuroHPC project under grant agreement No 951745. The JU receives support from the European Union's Horizon 2020 research and innovation program and Germany, Italy, Slovenia, France, and Spain.

References

[1] I. Molyneaux, *The art of application performance testing: from strategy to tools*. " O'Reilly Media, Inc.", 2014.

[2] S. Pradeep and Y. K. Sharma, "A pragmatic evaluation of stress and performance testing technologies for web based applications," in *2019 Amity International Conference on Artificial Intelligence (AICAI)*. IEEE, 2019, pp. 399–403.

[3] D. Mileski and M. Gusev, "Serverless implementations of real-time embarrassingly parallel problems," in *2022 30th Telecommunications Forum (TELFOR)*, 2022, pp. 1–4.

[4] M. Gusev, S. Ristov, A. Amza, A. Hohenegger, R. Prodan, D. Mileski, P. Gushev, and G. Temelkov, "Cardiohpc: Serverless approaches for real-time heart monitoring of thousands of patients," in *2022 IEEE/ACM Workshop on Workflows in Support of Large-Scale Science (WORKS)*, 2022, pp. 76–83.

[5] S. Ristov, M. Gusev, A. Hohenegger, R. Prodan, D. Mileski, P. Gushev, and G. Temelkov, "Serverless ecg stream processing in federated clouds with lambda architecture," *IEEE Computer*, vol. to be published, 2023.

[6] D. Mileski and M. Gusev, "Serverless faas scalability evaluation: An ecg signal processing use case," in *2022 45th Jubilee International Convention on Information, Communication and Electronic Technology (MIPRO)*, 2022, pp. 853–858.

[7] S. Nastic, T. Rausch, O. Scekic, S. Dustdar, M. Gusev, B. Koteska, M. Kostoska, B. Jakimovski, S. Ristov, and R. Prodan, "A serverless real-time data analytics platform for edge computing," *IEEE Internet Computing*, vol. 21, no. 4, pp. 64–71, 2017.

[8] N. Srivastava, U. Kumar, and P. Singh, "Software and performance testing tools," *Journal of Informatics Electrical and Electronics Engineering (JIEEE)*, vol. 2, no. 1, pp. 1–12, 2021.

[9] RadView, "Webload overview," Apr 2022, last accessed 06 February 2023. [Online]. Available: https://www.radview.com/webload-overview/

[10] M. Sharma, S. Vaishnavi, S. Sugandhi, and S. Abhinandhan, "A comparative study on load testing tools," *International Journal of Innovative Research in Computer and Communication Engineering*, vol. 4, no. 2, pp. 1906–1912, 2016.

[11] SmartBear, "No learning curve. no correlations. just real-world accuracy." last accessed 06 February 2023. [Online]. Available: https://loadninja.com/

[12] A. Kołtun and B. Pańczyk, "Comparative analysis of web application performance testing tools," *Journal of Computer Sciences Institute*, vol. 17, pp. 351–357, 2020.

[13] MicroFocus, "Load testing software: Loadrunner professional," last accessed 06 February 2023. [Online]. Available: https://www.microfocus.com/en-us/products/loadrunner-professional/overview

[14] R. Abbas, Z. Sultan, and S. N. Bhatti, "Comparative analysis of automated load testing tools: Apache jmeter, microsoft visual studio (tfs), loadrunner, siege," in *2017 international conference on communication technologies (comtech)*. IEEE, 2017, pp. 39–44.
[15] Apache, "Apache jmeter," last accessed 06 February 2023. [Online]. Available: https://jmeter.apache.org/
[16] ThoughtWorks, "The selenium browser automation project," last accessed 06 February 2023. [Online]. Available: https://www.selenium.dev/documentation/
[17] Tricentis, "Tricentis neoload for enterprise performance testing," Feb 2023, last accessed 06 February 2023. [Online]. Available: https://www.tricentis.com/products/performance-testing-neoload
[18] SmartBear, "Ensure high performance apis in less time," last accessed 06 February 2023. [Online]. Available: https://www.soapui.org/tools/readyapi/api-performance-testing/
[19] SoftLogica, "Test the performance of web applications under load," last accessed 06 February 2023. [Online]. Available: https://www.loadtestingtool.com/
[20] S. Kundu, "Web testing: tool, challenges and methods," *IJCSI International Journal of Computer Science Issues*, vol. 9, no. 2, pp. 1694–0814, 2012.
[21] I. Loadster, "Load stress testing for high-performance websites," last accessed 06 February 2023. [Online]. Available: https://loadster.app/
[22] G. Labs, "Open-source load testing tool for developers: K6 oss," last accessed 06 February 2023. [Online]. Available: https://k6.io/open-source/
[23] Q. Gao, W. Wang, G. Wu, X. Li, J. Wei, and H. Zhong, "Migrating load testing to the cloud: a case study," in *2013 IEEE Seventh International Symposium on Service-Oriented System Engineering*. IEEE, 2013, pp. 429–434.
[24] M. Yan, H. Sun, X. Liu, T. Deng, and X. Wang, "Delivering web service load testing as a service with a global cloud," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 3, pp. 526–545, 2015.
[25] I. Perforce Software, "The complete continuous testing platform," last accessed 06 February 2023. [Online]. Available: https://www.blazemeter.com/
[26] P. Memon, T. Hafiz, S. Bhatti, and S. S. Qureshi, "Comparative study of testing tools blazemeter and apache jmeter," *Sukkur IBA Journal of Computing and Mathematical Sciences*, vol. 2, no. 1, pp. 70–76, 2018.
[27] "Loader.io." [Online]. Available: http://docs.loader.io/api/intro.html
[28] G. Corp, "Gatling enterprise - load testing tool for business," Jan 2023, last accessed 06 February 2023. [Online]. Available: https://gatling.io/enterprise/
[29] "Flood io." [Online]. Available: https://guides.flood.io/overview-of-flood/what-is-flood
[30] [Online]. Available: https://doc.octoperf.com/
[31] S. L. Inc, "Cross browser testing, selenium testing, mobile testing," last accessed 06 February 2023. [Online]. Available: https://saucelabs.com/
[32] N. Kilinc, L. Sezer, and A. Mishra, "Cloud-based test tools: A brief comparative view," *Cybernetics and Information Technologies*, vol. 18, no. 4, pp. 3–14, 2018.
[33] "Google cloud documentation," last accessed 25 April 2023. [Online]. Available: https://cloud.google.com/docs
[34] "Google cloud functions generation 1 documentation," last accessed 25 April 2023. [Online]. Available: https://cloud.google.com/functions/docs
[35] "Google cloud storage documentation," last accessed 25 April 2023. [Online]. Available: https://cloud.google.com/storage/docs
[36] "Google cloud pub/sub documentation," last accessed 25 April 2023. [Online]. Available: https://cloud.google.com/pubsub/docs
[37] G. B. Moody and R. G. Mark, "The impact of the mit-bih arrhythmia database," *IEEE engineering in medicine and biology magazine*, vol. 20, no. 3, pp. 45–50, 2001.
[38] A. L. Goldberger, L. A. Amaral, L. Glass, J. M. Hausdorff, P. C. Ivanov, R. G. Mark, J. E. Mietus, G. B. Moody, C.-K. Peng, and H. E. Stanley, "Physiobank, physiotoolkit, and physionet: components of a new research resource for complex physiologic signals," *circulation*, vol. 101, no. 23, pp. e215–e220, 2000.
[39] Last accessed 25 April 2023. [Online]. Available: https://cloud.google.com/monitoring/docs