Evaluating the Performance of RESTful APIs Under Large HTTP Requests with K6

1st Fajra Risqulla
School of Elictrical Engineering
Telkom University
Bandung, Indonesia
fajrarisqulla@student.telkomuniversity.ac.id

2nd Casi Setianingsih
School of Elictrical Engineering
Telkom University
Bandung, Indonesia
setiacasie@telkomuniversity.ac.id

3rd Anggunmeka Luhur Prasasti
School of Elictrical Engineering
Telkom University
Bandung, Indonesia
anggunmeka@telkomuniversity.ac.id

***Abstract*— Application Programming Interfaces (APIs) are integral to contemporary software development, facilitating interoperability among various services without requiring knowledge of their internal implementations. Among API architectures, Representational State Transfer (REST) is widely adopted, leveraging HTTP methods such as GET, POST, PUT, and DELETE for client-server communication** [1]**. This paper focuses on evaluating the performances of RESTful API, specifically the dietary API, which employs image recognition to detect foods and provide nutritional data. Stress testing assesses the API's performance under high-volume HTTP requests to identify operational thresholds and improve reliability. Using the K6 tool, test scenarios simulate peak traffic conditions to measure critical metrics including response times, concurrency capacity, and requests per second. Findings highlight the impact of virtual user configurations and request parameters on API performance, offering insights crucial for reliability in real-world applications.**

Keywords—API, Stress Test, REST, RESTful API, K6, Virtual User

I. INTRODUCTION

API (Application Programming Interface) playing a crucial role in modern software development by allowing various services to integrated without needing to understand the internal workings of each component.

One of the most common architecture of APIs is the REST (Representational State Transfer). This architecture will run over HTTP protocol, so the communication between web services could accomplish by accessing HTTP methods, such as GET, POST, PUT, and DELETE [1], to interact with resources on a server. The API that implementing REST architecture will be called as "RESTful API". Furthermore, RESTful APIs could consume by various programming languages. These separating client and server components. To ensure the reliability of RESTful API under large HTTP requests, developer or QA must know the limitations of their applications, so that developer could improve the performance of their RESTful API. This paper will use a RESTful API that already exists, which is dietary [2].

Dietary is an application to assist users in managing their nutrition intake and helping them to choose healthier foods. It can detect foods from an image and provide the nutrition of foods.

This paper proposes a stress test to evaluate the performances of dietary API under large HTTP requests. By knowing the limitations of systems, it will help software developers to improve the RESTful API (dietary API). However, this paper is limited by several aspects, which are:

1. The RESTful API to testing or evaluating, which is dietary.
2. The machine used to host the RESTful API.
3. This stress test will cover only one endpoint, specifically the endpoint used to detecting foods "https://api.dietary.cloud/food/predict".
4. The number of images that used to detecting foods.

II. THEORITICAL REVIEW

A. *API (Application Programming Interface)*

API (Application Programming Interface) is an abstraction or interface that allows a software application (service) to communicate with other software applications without knowing the complexities of systems behind it [3]. Theses communications could exchange information between services. API has been used on web applications and operating systems [4].

B. *REST*

One of the most common architecture of APIs is the REST (Representational State Transfer). This architecture will run over HTTP protocol, so that HTTP client (Dietary Android Application) could access HTTP methods, such as GET, POST, PUT, and DELETE, to interact with resources on a server. JSON (JavaScript Object Notation) is used on REST architecture as standards message for communication between clients and servers. The API that implementing REST architecture will be called as "RESTful API" [2],[1].

C. *Stress Test*

Stress test is type of performance test to ensure the reliability of applications by testing applications under extreme conditions. This test will help developers to understand how their applications handle high traffic activities [5]. In the context of RESTful API, one of examples could be handling large HTTP requests. The objective of this test is to identify the limitations of systems [5].

D. *K6*

K6 is open-source testing tool, which is developed by Grafana Labs and designed to measure the performance and reliability of APIs, microservices, and websites [6]. It allows software developers and QA to run tests that simulate heavy loads on the applications by defining virtual users [7], [8].

K6 use JavaScript to define test scenarios and run the tests. Furthermore, this tool support stress testing. It provides metrics and reports to help software developer or QA to understand the application behaviors and its limitations as well as identify the areas for improvement.

III. RESEARCH METHODS

To ensure the reliability of RESTful APIs, developer must evaluate the performances. Stress test can be used effectively to find the limitations of RESTful APIs as well as the areas or parts that could improve the performances. There are several metrics that should be considered when evaluate the performances of RESTful APIs, which are:

1. The time taken for a HTTP request to get a response from RESTful APIs.
2. The number of users could be handled concurrently by RESTful APIs.
3. The number of requests that could be made in a second (Request Per Second).

This paper utilizes K6 as a tool to create and run stress test scenarios. K6 has various built-in metrics that can be used to evaluate the performances, which are [9]:

TABLE 1.
K6 METRICS

Metric	Description
http_reqs	Counts the total number of HTTP requests made during the test.
http_req_duration	Measures the time taken for an HTTP request to complete, including connection time, DNS resolution, and data transfer.
http_req_blocked	Time spent in waiting for an available TCP connection, including connection retries.
http_req_connecting	Time spent in establishing a TCP connection to the server.
http_req_tls_handshaking	Time spent performing TLS handshake.
http_req_sending	Time taken to send the HTTP request to the server.
http_req_waiting	Time spent waiting for the first byte of the response.
http_req_receiving	Time taken to receive the response data from the server.
http_req_failed	Counts the number of failed HTTP requests.
checks	Number of checks that passed during the test.
vus	Number of active virtual users.
vus_max	Maximum number of active virtual users during the test.
iterations	Counts the total number of requests made during test for each virtual user.

IV. STRESS TEST SCENARIOS AND SYSTEM SPECIFICATIONS

This section discusses the scenarios of stress tests and system specifications.

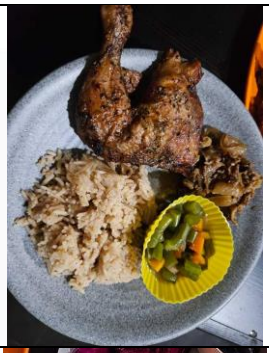
A. *System Specifications*

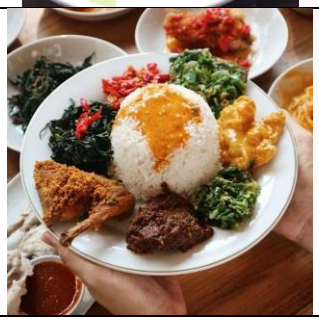
The RESTful API are deployed in Google Cloud Platform and the machine that host RESTful API has Intel(R) Xeon(R) @ 2.20GHz as CPU with 2GB of RAM. The machine uses GNU/Linux Debian Bookworm as operating system.

B. *Stress Test Scenarios*

This paper uses several data in stress testing. Furthermore, the data are images, it will use to detecting foods.

TABLE. 2.
STRESS TEST DATA

No.	Data
1	
2	

No.	Data
3	
4	
5	
6	
7	

No.	Data
8	
9	

Each data will be used randomly in the stress test scenarios. The stress test scenarios are as follows:

TABLE. 3.
Stress Test Scenarios

No.	Virtual User (vus)	Request Per Virtual User (iterations)	Total Request	RESTful API Endpoint
1	10	100	1000	https://api.dietary.cloud/food/predict
2	15	50	750	https://api.dietary.cloud/food/predict

The testing results are expected to have the number of failed HTTP requests, which are less than 10% and the average time taken for all HTTP requests to complete must be less than or equal to 1 seconds.

V. STRESS TEST SCRIPTS AND RESULTS

To fulfill the stress test scenarios, first developer must write scenarios in JavaScript then run it with K6 CLI (Command Line Interface).

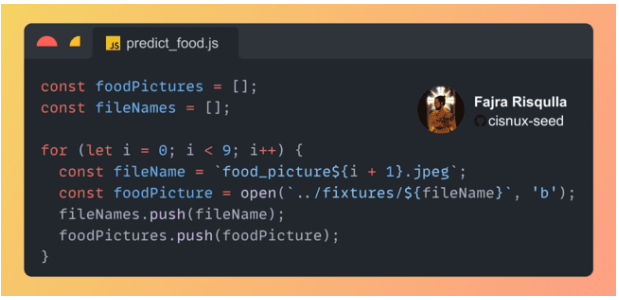


FIG. 1.
Load All The Data Before Run Testing

FIG. 2.
Define Virtual User And Iterations (Request Per Virtual User)

FIG. 3.
Define Checks During Testing

Now, the scripts could run it by K6 CLI. The results of these stress tests are as follows:

TABLE. 4.
Overview Of Stress Test Results

No.	Average Request Duration (http_req_ duration)	Number of Failed Requests (http_req_failed)	Request Per Second (http_reqs)
1	1s	0%	7.26 rps (request per second)
2	1s	27%	8.96 rps (request per second)

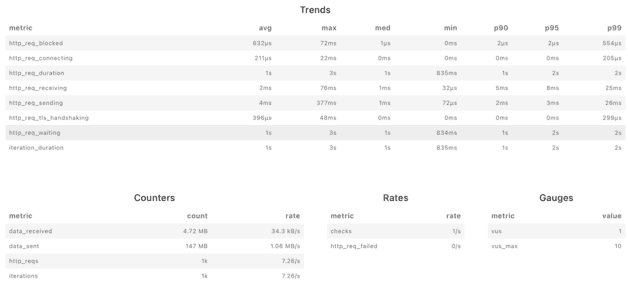


FIG. 4.
The K6 Result For First Scenario

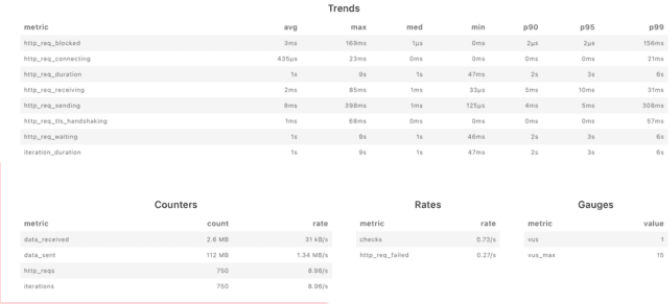


FIG. 5.
The K6 Result For Second Scenario

VI. STRESS TEST SCENARIOS AND SYSTEM SPECIFICATIONS

Based on the stress test results, the following analysis can be drawn:

A. Request Duration (http_req_duration)

The average request duration for first scenarios is 1 second with a maximum request duration of 3 seconds, whereas for second scenario, the average request duration is still same but with a maximum request duration of 9 seconds, indicating greater variation in response times under heavier loads.

B. Number of Failed Requests (http_req_failed)

The first scenario is more reliable, with zero failed requests compared to the 27% (0.27) failed requests per second in the second scenario.

C. Request Per Second (http_reqs)

First scenario has 7.26 requests per second, while second scenario has fewer total requests, but the requests per second increase to 8.96. The increase in requests per second correlates directly with the rise in virtual user count.

All these analyses highlight the importance of adjusting virtual users and request parameters to simulate real-world usage scenarios accurately. As the results, the first scenario successfully satisfied the expected output, but the second one is failed. These stress test results are affected by the programming language, the tools and libraries, which are used to build the RESTful API, and the AI model that chose to detecting foods is also affecting the performance. By analyzing response times, error rates, and overall metrics across these scenarios, developers can effectively optimize the RESTful API's performance.

REFERENCES

[1] B. Xu, K. Mou, Institute of Electrical and Electronics Engineers. Beijing Section, and Institute of Electrical and Electronics Engineers, *The Design of Embedded Web System based on REST Architecture*. 2019.

[2] A. Ehsan, M. A. M. E. Abuhaliqa, C. Catal, and D. Mishra, "RESTful API Testing Methodologies: Rationale, Challenges, and Solution Directions," *Applied Sciences (Switzerland)*, vol. 12, no. 9. MDPI, May 01, 2022. doi: 10.3390/app12094369.

[3] I. Rauf, E. Troubitsyna, and I. Porres, "Systematic mapping study of API usability evaluation methods," *Computer Science Review*, vol. 33. Elsevier Ireland Ltd, pp. 49–68, 2019. doi: 10.1016/j.cosrev.2019.05.001.

[4] E. Mosqueira-Rey, D. Alonso-Ríos, V. Moret-Bonillo, I. Fernández-Varela, and D. Álvarez-Estévez, "A systematic approach to API usability: Taxonomy-derived criteria and a case study," *Inf Softw Technol*, vol. 97, pp. 46–63, May 2018, doi: 10.1016/j.infsof.2017.12.010.

[5] M. Hendayun, A. Ginanjar, and Y. Ihsan, "ANALYSIS OF APPLICATION PERFORMANCE TESTING USING LOAD TESTING AND STRESS TESTING METHODS IN API SERVICE," *JURNAL SISFOTEK GLOBAL*, vol. 13, no. 1, p. 28, Mar. 2023, doi: 10.38101/sisfotek.v13i1.2656.

[6] "Garafana K6," Grafana Labs. Accessed: Jul. 10, 2024. [Online]. Available: https://grafana.com/docs/k6/latest/

[7] F. A. Julana, "Analyzing QoS Performance in Kubernetes-Based High Scalability Clusters," 2023.

[8] C. Konkel Bachelor's Thesis, "Benchmarking Scalability of Load Generator Tools," 2023.

[9] I. Vals, "Understanding K6 Results." Accessed: Jul. 10, 2024. [Online]. Available: https://github.com/grafana/k6-learn/blob/main/Modules/II-k6-Foundations/03-Understanding-k6-results.md