Performance Analysis of RESTful API and RabbitMQ for Microservice Web Application

Xian Jun Hong
School of Electronic Engineering
Soongsil University
Korea, Seoul
xianjun666@dcn.ssu.ac.kr

Hyun Sik Yang
School of Electronic Engineering
Soongsil University
Korea, Seoul
yangun@dcn.ssu.ac.kr

Young Han Kim
School of Electronic Engineering
Soongsil University
Korea, Seoul
yhkim@dcn.ssu.ac.kr

***Abstract*— In order to explore the communication methods of microservice web application, this paper uses RabbitMQ and REST API respectively as the message-oriented middleware of microservice web applications. We do experiments with both of the methods under various number of users to compare and evaluate their performance in different circumstances. The purpose is to provide understanding inside the two methods for microservice web applications so that service providers can select the appropriate method based on their need. Obtained experimental results show that when a large number of users send requests to the web application at the same time, it is more stable to use RabbitMQ as the Message-oriented middleware than the REST API communication method.**

Keywords—Microservice, RESTful API, AMQP

I. INTRODUCTION

REST (Representational State Transfer) is an architectural style in which a representational state transition is a set of architectural constraints and principles[1][2]. The RESTful API is an API that makes calling resources very convenient and intuitive, reducing the complexity of the service. In most cases, microservices use RESTful API to deliver messages. Considering that webpages or software may serve a large number of users sending requests at the same time, this paper analyzes the case and uses AMQP alternatively, which is relatively stable in information transmission, as a message middleware. AMQP (Advanced Message Queuing Protocol) is an application layer standard advanced message queuing protocol that provides unified messaging services. Message delivery can be more accurate and stable. When a large number of users send requests at the same time, even if they cannot be processed in time, the messages are stored in the message queue. The requests are then fetched as well processed when they can be processed. In this paper, AMQP-based RabbitMQ is used as the message middleware for a comparison. In order to analyze and evaluate the performance of RESTful API mode communication and AMQP mode in processing requests in the microservice web page architecture, a simple micro service is implemented and tested. Performance metrics are evaluated under a various number of users, and the performance of the two communication modes changes as the number of users increase.

II. RELATED WORK

A. Representational State Transfer ful Application Programming Interface

REST is the architecture of a distributed hypermedia system, first proposed by Roy Rleding in his noted paper in 2000. The key information abstraction in REST is a resource. Any information that can be named can be a resource: a document or image, a temporary service, a temporary service, a total of other resources, a non-virtual object (included in a task, etc.).

The RESTful API is a REST-based API that uses resource identifiers to represent specific resources designed for interaction between components. The state of a resource at any given time is called a resource representation. This representation consists of data, metadata describing the data, and hypermedia links that allow the user to switch state of the resource[3]. A key advantage of the RESTful API is that they provide a lot of flexibility. Data doesn't depend on resources or methods, so the RESTful API can handle multiple types of calls and return different data formats. Since the existing mature RESTful API is used in the HTTP environment, use GET, POST, PUT, DELETE and other actions to manipulate resources. Manipulating resources through these actions makes resource calls more flexible and convenient, and makes the code more concise. Although this flexible and concise communication method is very suitable for Microservice, it is only a method and cannot be applied to all scenarios. Because HTTP method will crash or lose when it faces the request amount exceeding its own computing power, those that cannot be processed Request, so you need to consider a Message-oriented middleware that can handle this scenario.

B. Advanced Message Queuing Protocol

AMQP is an advanced Message Queue protocol, which is an open standard application layer for protocol-oriented message-oriented middleware services[4]. AMQP enables applications to send and receive messages. In this regard, it works like instant messaging or email. AMQP is very different from other available solutions because it allows you to specify which messages can be received and received, and how to trade off security, reliability, and performance. Systems that integrate AMQP perform better than other solutions in unattended or "lights out" functionality. There are several reasons for choosing AMQP in the competition, including convenience, the possibility of connecting applications on different platforms, the possibility of connecting business partners with full-featured open standards, and the innovative position based on AMQP. As shown in Figure 1, in the service, the three main functional modules are linked into a processing chain to perform the intended function.

The messages sent by the publishing application are first received by "exchange" and are routed to the "message queue" according to certain rules. The "message queue" stores messages until they are completely processed by the consumer. "binding" defines the association between "exchange" and "message queue" and provides routing rules[5].

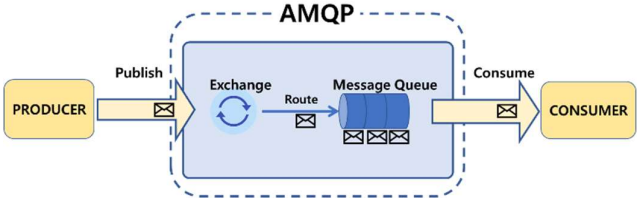

Fig. 1. AMQP Architecture.

In this paper, in order to perform performance testing on RESTful API and AMQP in Microservice web application architecture, AMQP-based RabbitMQ is used as Message-oriented middleware to provide communication between internal services.

III. Microservice Web Application

Representational State Transfer ful API Method

In the previous section, the RESTful API is briefly introduced. The RESTful API is flexible and convenient, and the data do not depend on resources or methods. In order to test the RESTful API for the Microservice architecture, a simple web application is designed as shown in Figure 2. All services and API gateways contain REST APIs that define actions in advance, are implemented based on the received METHOD and URL. Therefore, there is no need to understand the internal principles, as long as these actions are defined in advance. The services can communicate with each other to process the request. The API gateway aggregates the addresses of all services. Its main function is to forward requests to the corresponding services. When the API gateway receives the request, it first checks the method and the URL, and forwards the request to the corresponding service through the RESTful API defined in advance. When a user makes a request to the web application, the API gateway first receives the request and views the method and URL. The API gateway sends this request to the appropriate service via the defined REST API. When the service receives the request, it checks the method and URL again, processes the request through the defined REST API, and makes an action.

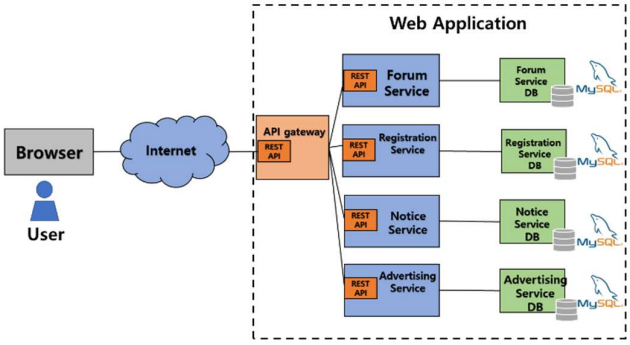

Fig. 2. Microservice Web Application with RESTful API

RabbitMQ Method

In order to test the Message Queue of the Microservice architecture, the Microservice architecture of Figure 2 is modified as shown in figure 3. The REST API is not used here and the API gateway is revoked. This architecture uses RabbitMQ as the information exchange tool and defines some actions in the Event APIs that are implemented according to the received message.

As shown in Figure 3, RabbitMQ has an Exchange function module inside. This module is used to route the release of the message and publishes the message to the specified queue. The Exchange is the core module of RabbitMQ and has four types. The four types are Direct Exchange, Fanout exchange, Topic exchange and Headers exchange. Since each type has a different CPU overhead. We select the appropriate type according to different needs [1]. In this architecture, a message needs to be sent to the specified queue and received by the specified service, so Direct Exchange is used. When a message is received by the Exchange module, it checks which queue the message is sent to and sends the message to the queue. The service module listening to the queue receives the information in the queue, checks the message, and executes the Event according to the content of the message. Actions are defined in advance in the API.

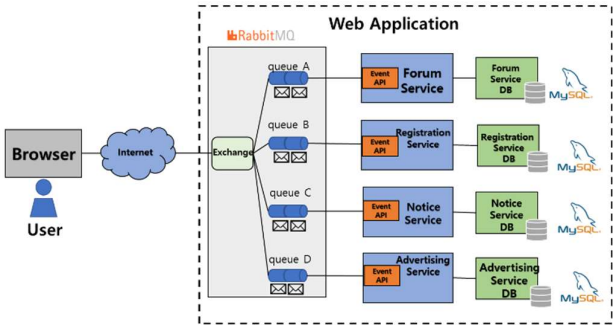

Fig. 3. Microservice Web Application with RabbitMQ

IV. Benchmark Results

In order to evaluate the above two architectures, we use the following hardware and tools for experiments: Computer model: Intel NUC NUC5I7RYH, Processor: Intel Core i7(Fifth)(4Mcache,3.1GHz,2cores, 4threads); Memory: 16GB DDR3L 1600MHz; Disk: SSD 256GB; Network: 10Gb/s interface, bandwidth 1GB; OS: ubuntu 16.04 (64-bit); RabbitMQ: 3.7.3; Apache: 2.4.29 (64-bit); performance test tool: apache jmeter 4.0.

In the experiments, in order to compare the two deployed Microservice web applications, this study conducts several experiments and deploys them. In order to ensure the accuracy of the test, we repeat tests and compute the results as average values. In order to test the performance impact of the number of users on the two deployment services, we vary the number of users, from 50, 100, 150, 200, 250, to 300 users who send requests for information to the service simultaneously within 15 minutes. The results are presented below. This article tests the speed at which users receive response messages after sending a request under different numbers of users. As shown in figure 4, when the number of simultaneous online users is 50, the difference in response speed between the two deployment modes is almost negligible. However, as the number of users increases, the response speed of the RESTful API method is gradually faster than that of the RabbitMQ mode. When the number of simultaneous online users is 200, the response speed of the RESTful API deployment mode is nearly twice that of the RabbitMQ deployment mode. When the number of users increases again, the speed of the RESTful

API method gradually decreases. When the number of simultaneous online users is 250, the response speed of the RESTful API method is lower than the corresponding speed of the RabbitMQ mode. When the number of simultaneous online users increases to 300, the RESTful API method has very poor performance and can only provide services with a response speed of 26.4/s, while the response speed of the RabbitMQ mode is relatively stable at 342.4/s.

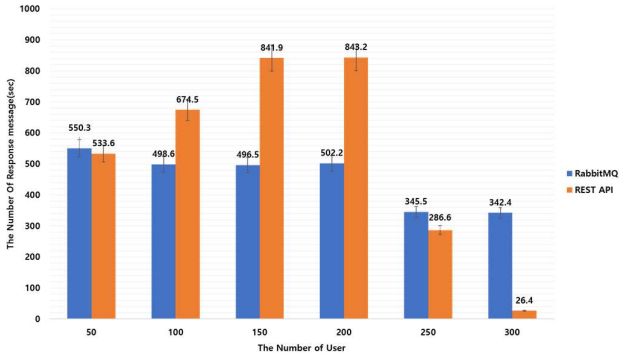

Fig. 4. RabbitMQ and RESTful API response speed comparison

It can be observed from figure 5 that at the same time, a great number of online users also result in some error messages in the RESTful API mode. When the number of online users increases to 300, the error information rate increases from 0.12% to 28.41%. Under normal circumstances, the service cannot be provided normally, which seriously affects the quality of service.

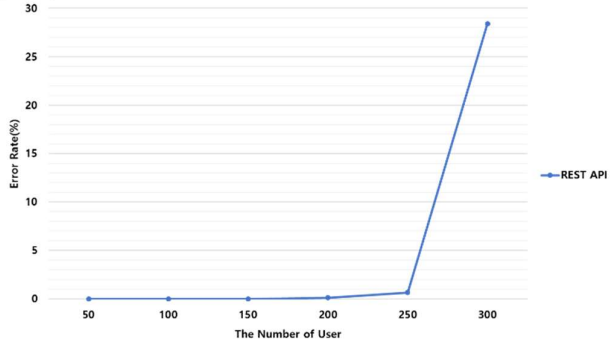

Fig. 5. The error rate of the RESTful API

Through experiments, it can be concluded that in the RESTful API mode, when the number of online users is small, the response request speed is faster than the RabbitMQ method, but when the number of requests exceeds the load, the performance is gradually degraded and errors occur, and even the service cannot be provided. Because the number of request information exceeds the system processing load and cannot respond to these requests, errors occur. When the number of requests exceeds a certain level, the error is too much and the system cannot be serviced normally. In the RabbitMQ mode, when the number of users is small, the response request speed is not as good as the RESTful API method, but the increasing in the number of online users does not have much impact on performance, and the performance is relatively stable. Because all requests are sent to the Message-oriented middleware Message Queue and stored, they can be extracted and processed from the Message Queue when they can be processed, instead of directly requesting and responding like HTTP. As a result, when the request volume is relatively small, the response speed compared with the RESTful API method, it is relatively slow. However, when the number of requests increases, there is no error or cannot respond to the request. The performance is maintained relatively stable.

V. Conclusion

In order to test the two communication methods of the Microservice architecture, this paper performs some simulation tests on the Microservice web application using the RESTful API method and the RabbitMQ method. The final result is that the Microservice web application uses the RESTful API, and the response request performance is relatively good when the number of simultaneous requests is small. However, when the number of requests exceeds the load, the performance is seriously affected or even the service cannot be provided normally. The Microservice web application using the RabbitMQ method, although the response request performance is not as good as the RESTful API when there are few online users at the same time, the number of requests does not have much impact on performance and is relatively stable. Since the testing environment does not take into account many practical factors, as a future plan, adding a variety of actual situations and factors that may occur makes the test more accurate and reliable.

VI. Acknowledgment

This work was supported by Institute for Information & Communications Technology Promotion (IITP) grants funded by the Korea government (MSIT) (No. 2015-0-00575, Global SDN/NFV Open-Source Software Core Module/Function Development) and MSIT(Ministry of Science and ICT), Korea, under the ITRC(Information Technology Research Center) support program(IITP-2018-2017-0-01633) supervised by the IITP(Institute for Information & communications Technology Promotion).

References

[1] W. Hasselbring, and G. Steinacker, "Microservice architectures for scalability, agility and reliability in e-commerce," IEEE International Conference on, vol. 1, pp. 243-246, April 2017.

[2] L. Li, and W. Chou, "Design and describe REST API without violating REST: A Petri net based approach," IEEE International Conference on Web Services, pp. 508-515, July 2011.

[3] V. Mario, G. Oscar, C. Harold, V. Mauricio, S. Lorena, C. Rubby, and G. Santiago. "Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud," In Computing Colombian Conference, 2015 10th, pp. 583-590, November 2015.

[4] J.L. Fernandes, I.C. Lopes, J.J. Rodrigues, and S. Ullah. "Performance evaluation of RESTful web services and AMQP protocol," In Ubiquitous and Future Networks, 2013 Fifth International Conference on, pp. 810-815, July 2013.

[5] V.M. Ionescu, "The analysis of the performance of RabbitMQ and ActiveMQ," In RoEduNet International Conference-Networking in Education and Research, 2015 14th, pp. 132-137, October 2015.