MSABot: A Chatbot Framework for Assisting in the Development and Operation of Microservice-Based Systems

Chun-Ting Lin
Department of Computer Science and Engineering
National Taiwan Ocean University
Keelung, Taiwan
00357014@email.ntou.edu.tw

Shang-Pin Ma
Department of Computer Science and Engineering
National Taiwan Ocean University
Keelung, Taiwan
albert@ntou.edu.tw

Yu-Wen Huang
Department of Computer Science and Engineering
National Taiwan Ocean University
Keelung, Taiwan
00557012@ntou.edu.tw

ABSTRACT

Microservice architecture (MSA) has become a popular architectural style. The main advantages of MSA include modularization and scalability. However, the development and maintenance of Microservice-based systems are more complex than traditional monolithic architecture. This research plans to develop a novel Chatbot system, referred to as MSABot (Microservice Architecture Bot), to assist in the development and operation of Microservice-based systems by using Chatbots. MSABot integrates a variety of tools to allow users to understand the current status of Microservice development and operation, and to push the information of system errors or risks to users. For the operators who take over the maintenance of Microservices, MSABot also allows them to quickly understand the overall service architecture and the operation status of each service. Besides, we invited multiple users who are familiar with the technology of Microservice or ChapOps to evaluate MSABot. The results of the survey show that more than 90% of the respondents believe that MSABot can adequately support the development and maintenance of Microservice-based systems.

CCS CONCEPTS

Software and its engineering → Software creation and management

KEYWORDS

Microservice, Microservice Architecture, Chatbot, ChatOps, Hubot, Rasa

ACM Reference format:

Chun-Ting Lin, Shang-Pin Ma, and Yu-Wen Huang. 2020. MSABot: A Chatbot Framework for Assisting in the Development and Operation of Microservice-Based Systems. In *Proceedings of 2nd International Workshop on Bots in Software Engineering (BotSE 2020). ACM, New York, NY, USA, 5 pages.* https://doi.org/10.1145/1234567890

ICSEW'20, May 23–29, 2020, Seoul, Republic of Korea

ACM ISBN 978-1-4503-7963-2/20/05…$15.00
https://doi.org/10.1145/3387940.3391501

1 Introduction

Nowadays, Microservice architecture (MSA) has become a popular architectural style [1, 2]. In MSA, Microservices operate as system components, which perform specific tasks and collaborate with each other via lightweight communication mechanisms. The main advantages of MSA include modularization and scalability. However, the development and maintenance of Microservice-based systems are not trivial and even complex. The development of Microservices could be divided into four phases: requirement analysis, service design, environment establishment, and implementation & testing. In addition to the analysis, design, implementation, and testing of service functionality, the Microservice team needs to set up the development environment that involves multiple tools and track the build status of Microservices in the development phase. The maintenance of Microservices could be divided into three phases: monitoring, detection, and repair. Maintenance engineers need to maintain the built services, monitor running services, and fix occurred failures in the operation phase.

In order to perform the above tasks well, the Microservice team need to be familiar with a lot of tools to obtain a variety of data [3], but can only collect scattered and unintegrated information. DevOps tools are often used to help the Microservice team to perform required tasks in a more automated way [4, 5]. Although the tools of CI (continuous integration) and CD (continuous deployment) can effectively reduce developers' effort, the selection, learning, use, and integration of DevOps tools is not trivial, and these tools still cannot help developers or maintenance operators to easily understand the important knowledge and information of current development environment, the service development status, the system architecture, and the service operation status.

The concept of ChatOps is to realize DevOps in a chat room by using Chatbots [6, 7]. Chatbots are computer programs designed to

chat with users via text or voice [8]. In ChatOps, Chatbots can passively or actively let developers and maintenance operators understand the current development and maintenance status [9]. Several ChatOps tools are available today to develop the Chatbots in DevOps [10-12]; however, these solutions are not devised for Microservice-based systems. Meanwhile, these solutions usually rely on regular expressions and are not able to understand the sentences with the same meaning [13]. Bots only accept queries with fixed instructions, which usually specify spaces, wording, and content structure strictly.

Based on the above analysis, this research proposes a novel Chatbot that assists in the development of Microservice architecture and supports NLP (natural language processing) capabilities, referred to be MSABot (Microservice Architecture Bot). MSABot combines a variety of Microservices development and maintenance tools to allow users to understand the current status of Microservices development and operation and to push the Microservice system's real-time error or risk information to users. For the operators who take over the maintenance of Microservices, MSABot also allows them to more quickly understand the overall service architecture and the operation status of each service.

The following paper is organized as follows: Section 2 presents system requirements. Section 3 describes system design. Section 4 discusses our evaluation survey and results. Conclusions are drawn in Section 5.

2 MSABot System Requirements

For a normal process of developing and maintaining a Microservice-based system, multiple tasks are involved. In the development stage, first, the Microservice development team (dev team shortly) specify the requirements, write or produce the OAS (OpenAPI Specification, or called Swagger) document of each Microservice, and implement and test services based on the OAS. The implementation task includes the integration of the API gateway and the registration to the service discovery system. Meanwhile, the team needs to connect the VCS (version control system) and CI/CD (continuous integration/continuous deployment) tools and set up the CI/CD pipeline. During the stage of development, dev team members need to check the version information from the VCS, the build and test results from the CI/CD tools, and the service status from the service discovery system.

In the operation stage, the operation team (i.e., the maintenance engineers) must first read the OAS and supplemental documents of each Microservice to understand the APIs. Similar to the dev team, the operation team must use the CI/CD tools to study the test cases and the test report to understand the functionality of Microservices, and check the service status from the service discovery system. Furthermore, the operation team must identify risks and detect errors from the log manually. It is obvious that if there is an assistant tool that can ease the above process in a more automated and integrative way, the effort and complexity of the development and operation of Microservice-based systems could be reduced.

Therefore, based on the above analysis, we identified nine main system requirements of the proposed MSABot:

1. Users can ask MSABot for detailed information about each service, such as version control status and person in charge.
2. Users can ask MSABot for the OAS (Swagger) document of each service.
3. Users can learn about the dependency among the Microservices from MSABot.
4. Users can ask MSABot for the service health information.
5. Users can ask MSABot for the usage status and statistics of a service.
6. Users can ask MSABot to browse the current system settings.
7. Users can use MSABot to receive the test and the deployment results.
8. Users can use MSABot to receive the error or risk notifications.
9. Users can send different sentences with the same meaning to MSABot, and MSABot can still recognize and respond to the correct results. In other words, the user may not fully follow the precise command formats.

3 MSABot System Design

To realize the system goals, MSABot integrates a lot of related tools, including Jenkins[1], Eureka[2], Zuul[3], Actuator[4], Swagger Browser, and VMAMV [14]. Jenkins is an open-sourced automation tool that enables the developers to build, test, and deploy their software systems automatically. Eureka, developed by Netflix, is a service registration and discovery tool with a built-in load balancer. Zuul provides an API gateway with multiple types of filters that intercept the traffic. Actuator is used to probe the status of services, such as health, metrics, or info. VMAMV (Version-based Microservice Analysis, Monitoring, and Visualization), which was developed by our lab, can discover service anomalies for all service versions in runtime and immediately notifies users of problems shortly after they occur. Besides, RabbitMQ[5] is also used to exchange messages between MSABot and other tools in a message-driven way.

Based on the identified requirements and integrated tools, we designed multiple scenarios (See Table 1) for the development and operation of Microservice-based systems. Notably, the scenarios 1~7 are the passive actions, which indicate that the Bot is passively waiting for queries given by the user, and the scenarios 8~10 are the active actions, which indicate that the MSABot actively pushes important messages to the users. Besides, the users of MSABot include developers, testers, maintenance staff and even customers and FAE (field application engineer). Note that not all users of

[1] https://jenkins.io/zh/
[2] https://github.com/topics/netflix-eureka
[3] https://github.com/Netflix/zuul
[4] https://github.com/spring-projects/spring-boot/tree/master/spring-boot-project/spring-boot-actuator
[5] https://www.rabbitmq.com/

MSABot are programmers and flexible conversational interface is more appropriate than the strict command interface.

Table 1: Conversation scenario and actions

No	Scenarios	Used Tools
1	Users want to browse a service's information.	Zuul
2	Users want to track the usage of a service.	Actuator on Zuul
3	Users want to read the API documentation for a service.	Swagger Browser on Zuul
4	Users want to see the environmental setting.	Eureka
5	Users want to check the service health status.	Eureka
6	Users want to see the service dependency graph [15] for all Microservices in a project.	VMAMV
7	Users want to set and view the system parameters for a Microservice project.	All tools
8	When the build or deployment on Jenkins is completed, the users are notified.	Jenkins and RabbitMQ
9	When a service error occurs on the Eureka Server, the users are notified.	Eureka and RabbitMQ
10	When an error or warning message is produced by VMAMV, the users are notified.	VMAMV and RabbitMQ

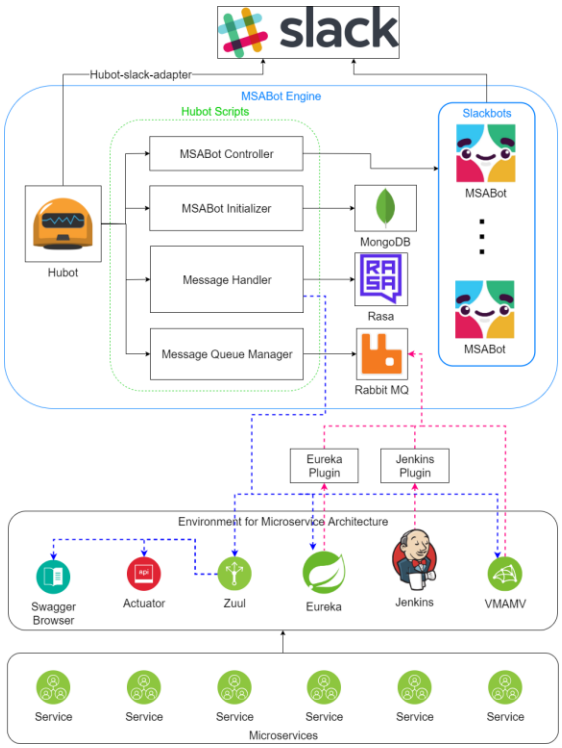

Figure 1: MSABot system architecture

For the use of MSABot, users can install MSABot in the chat room, and use the chat room to interact with MSABot to get the required information after setting the system-related parameters. This research is developed based on the Hubot[6] robot framework, Spring Cloud[7] ecosystem, and the Slack[8] IM (instant messaging) system. Note that new Microservice platform and chat room system can be integrated into MSABot by building required adapters and plugins.

The MSABot system (shown in Figure 1) is divided into the MSABot Engine and plugins connected to Jenkins and Eureka. There are five main parts in the MSABot Engine: (1) Each MSABot inherits Slackbot and records the chat room's information. (2) The MSABot Controller is used to manage MSABot instances. (3) The MSABot Initializer is used to initialize and reset MSABot instances. (4) The Message Handler performs semantic analysis and corresponding actions through Rasa[9] (discussed later). (5) The Message Queue Manager is used to monitor the messages in RabbitMQ, which sent by the plugins on the user's server. We use MongoDB to manage the user authentication information.

As mentioned, there are many kinds of users in MSABot. Scripts developed by Hubot only accept queries with fixed instructions, which may specify spaces, wording, and content structure strictly. In fact, it is not friendly to users, especially non-program developers. Therefore, we use Rasa for natural language processing in MSABot. Rasa is an open-source machine learning framework that can be used to create conversational AI Chatbots. The Rasa framework has two important parts: (1) Rasa NLU is used for NLP and machine learning, and (2) Rasa core SDK is responsible for keyword extraction. Currently, we design more than ten intents in the Rasa framework, such as bot_join, service_info, service_health, service_usage_info, service_api_list, service_env, last_build_fail, connect_error, and service_dependency_graph, and also design corresponding actions. Besides, Rasa provides multiple training methods, such as MITIE, Supervised method, spaCy, and Tensorflow. The comparison of these training methods is shown in Table 2. Based on the analysis, we chose and applied the spaCy training method by using the pretrained_embeddings_spacy pipeline and pre-trained language models in this research to allow users input semantically-equivalent sentences.

Table 2: Alternative training methods in Rasa

Method	Pros and cons
MITIE	● Need to set up the pipeline manually. ● Need to train the model manually. ● The Rasa community does not recommend to use MITIE.
Supervised method	● The Rasa community suggests using this method if you have 1000 or more labeled utterances.
spaCy	● Pre-processing of words in the same classification. ● The Rasa community suggests using this method.
Tensorflow	● Fetching (keywords) Entity is not supported. ● This method was abandoned after version 0.15.

After installing and setting up MSABot, users can ask MSABot about services information, such as its GIT repository, current version, service URL, or API documentation (See Figure 2). Note that the user may input imprecise sentences to obtain the required information, since the machine learning for intents was executed. Furthermore, users can obtain the service status and be notified about the failure status in the chat room (See Figure 3).

[6] https://hubot.github.com/
[7] https://spring.io/projects/spring-cloud
[8] https://slack.com/intl/en-tw/
[9] https://rasa.com/

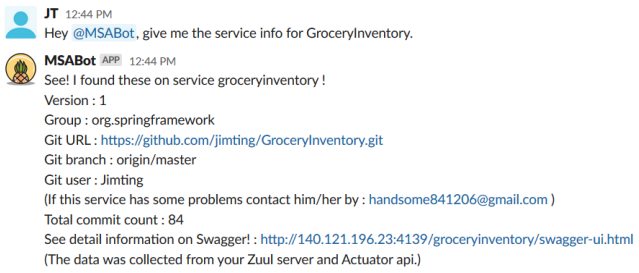

Figure 2: Use case example: query for service information

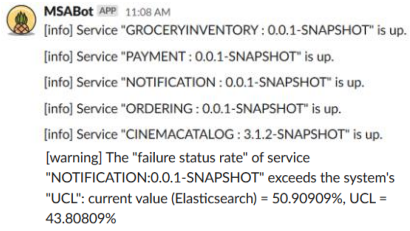

Figure 3: Use case example: active risk notification

4 Evaluations

To evaluate the proposed approach, we designed a survey questionnaire and invited 11 participants (including software engineers, software architects, and computer science students) who are familiar with Microservices technology or ChatOps technology to fill in the questionnaire and provide feedback. The questions need to be filled in include:

Q1. Background information (such as age and work experiences, and job title).

Q21. It is not convenient for developers to manually check the specifications and status of each service.

Q22. Integrating various development tools, such as version control, CI/CD, and notifications is quite complicated.

Q23. Maintenance engineers have difficulty for taking over system maintenance.

Q24. It is difficult for developers and maintenance engineers to understand the entire Microservice system architecture.

Q31. MSABot is helpful to acquire information and knowledge of a Microservice project.

Q32. MSABot can assist in Microservice development.

Q33. MSABot can facilitate the Microservice maintenance.

Q34. Via MSABot, Microservice team can more easily know the basic information and status of each service.

Q35. MSABot allows non-programmers or new operators to understand the system quickly.

Q36. In general, MSABot is beneficial for the development and maintenance of auxiliary Microservice systems.

Q4. Suggestions to MSABot.

The results of the survey (Q21~Q36) are shown in Figure 4. Widely-used Likert scaling was applied. SA indicates “strongly agree”; A indicates “agree”; N indicates “no comment”; D indicates "disagree"; "SD" indicates " strongly disagree". In general, respondents agree that the development and operation of Microservice-based applications are complex, and MSABot is able to assist developers and maintenance engineers in building and operating Microservice systems by providing real-time information passively and alert notifications actively.

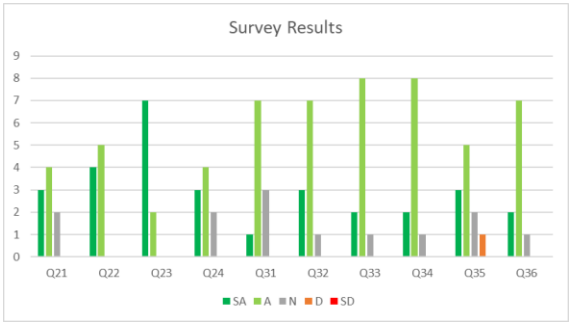

Figure 4 Results of the survey

Two of the respondents provided additional feedbacks for Q4, including two kinds of suggestions:

1. Developer issue: Programmers may prefer the strict command interface and have no confidence in bots.
   Our responses: as mentioned, strict commands are not friendly for users, especially non-programming stakeholders. With the conversational interface with NLP support, both programmers and non-programmers can issue queries more easily and confidently.
2. Context issue: One of the important features of a Chatbot is to record the context. It would be better if the Chatbot could understand the current situation and may give different responses for different contexts.
   Our responses: At present, MSABot can record the Microservice name mentioned in the previous conversation Users do not need to repeatedly enter the service name. In the future, we plan to improve the NLP capability to further understand the context and provide adequate information that the users may be interested in, for example, version change log or related test cases and reports.

5 Conclusion

This paper proposes a Chatbot framework that assists in the development and maintenance of the Microservices architecture, called MSABot (Microservice Architecture Bot). In summary, MSABot provides a flexible conversational interface to extract the service information, service API documentation, building and testing results, the health status, service usage analysis, and service dependency graphs by connecting multiple tools. Users can issue queries to MSABot using normal sentences, not strict commands. Users can also receive real-time deploying notifications and service error or risk messages by MSABot.

Acknowledgment

This research was sponsored by Ministry of Science and Technology in Taiwan under the grant MOST 108-2221-E-019-026-MY3.

REFERENCES

[1] Xabier Larrucea, Izaskun Santamaria, Ricardo Colomo-Palacios, and Christof Ebert, "Microservices," *IEEE Software*, vol. 35, no. 3, pp. 96-100, 2018.

[2] Shang-Pin Ma, Chen-Yuan Fan, Yen Chuang, I. Hsiu Liu, and Ci-Wei Lan, "Graph-based and scenario-driven microservice analysis, retrieval, and testing," *Future Generation Computer Systems*, vol. 100, pp. 724-735, 2019/11/01/ 2019, doi: https://doi.org/10.1016/j.future.2019.05.048.

[3] Rory V. O'Connor, Peter Elger, and Paul M. Clarke, "Continuous software engineering-A microservices architecture perspective," *Journal of Software: Evolution and Process*, vol. 29, no. 11, 2017, doi: 10.1002/smr.1866.

[4] Lianping Chen, "Microservices: Architecting for Continuous Delivery and DevOps," presented at the 2018 IEEE International Conference on Software Architecture (ICSA), 2018.

[5] Leonardo Leite, Carla Rocha, Fabio Kon, Dejan Milojicic, and Paulo Meirelles, "A Survey of DevOps Concepts and Challenges," *ACM Computing Surveys*, vol. 52, no. 6, pp. 1-35, 2019, doi: 10.1145/3359981.

[6] Rory V O'Connor, Peter Elger, and Paul M Clarke, "Continuous software engineering—A microservices architecture perspective," *Journal of Software: Evolution and Process*, vol. 29, no. 11, p. e1866, 2017.

[7] Jason Hand, *ChatOps: Managing Operations from Group Chat*. O'Reilly Media, 2016.

[8] Shang-Pin Ma and Ching-Ting Ho, "Modularized and Flow-Based Approach to Chatbot Design and Deployment," *Journal of Information Science & Engineering*, vol. 34, no. 5, 2018.

[9] Philipp Hukal, Nicholas Berente, Matt Germonprez, and Aaron Schecter, "Bots Coordinating Work in Open Source Software Projects," *Computer*, vol. 52, no. 9, pp. 52-60, 2019, doi: 10.1109/mc.2018.2885970.

[10] Markus Juopperi, "Deployment automation with ChatOps and Ansible," 2017.

[11] Junji Kinoshita, Yoji Ozawa, Ken Akune, and Nazim Sebih, "Cloud Service Based on OSS," *Technology Innovation for Accelerating Digital Transformation*, pp. p78-82, 2017.

[12] Kavyashree S, "Nia–Semi-Automated Intent Based Enterprise Chatbot," *International Journal of Innovative Science and Research Technology*, vol. 3, no. 3, 2018.

[13] Amir Shevat, *Designing Bots: Creating Conversational Experiences*. 2017. "O'Reilly Media, Inc.", 2017.

[14] Shang-Pin Ma, I-Hsiu Liu, Chun-Yu Chen, Jiun-Ting Lin, and Nien-Lin Hsueh, "Version-Based Microservice Analysis, Monitoring, and Visualization," in *2019 26th Asia-Pacific Software Engineering Conference (APSEC)*, 2019: IEEE, pp. 165-172.

[15] Shang-Pin Ma, Chen-Yuan Fan, Yen Chuang, Wen-Tin Lee, Shin-Jie Lee, and Nien-Lin Hsueh, "Using service dependency graph to analyze and test microservices," in *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, 2018, vol. 2: IEEE, pp. 81-86.