Development of Bot for Microservices Server Monitoring Using Life Cycle Approach to Network Design Method

Setyadhi Putra Deriyanto[1], Heru Agus Santoso[2] *

[1,2]*Department of Informatics, Fakulty of Computer Science, Dian Nuswantoro University, Semarang*

*corresponding author: heru.agus.santoso@dsn.dinus.ac.id

Abstract - This study was conducted on the emergency response system implemented in Demak, Kendal, and Batang districts, namely Tanggapin. This system provides services in the form of reporting and actions and information about the health sector, from the availability of hospital rooms to the availability of blood at PMI. Tanggapin uses a mobile-based and web-based platform, where the web is used as a medium for operators to input data and validates incoming reports. The complex system services with various supporting features require a microservices architecture to operate independently and distributed. Because of this dynamic system, real-time monitoring support for the server performance is needed via Telegram Bot. The monitoring system was built using the PPDIOO Lifecycle Approach to Network Design and Implementation method because it is reliable, and the flow of a process never breaks. The format used to exchange data uses the JSON (Javascript Object Notation) format. This study shows that the monitoring system runs well because every test carried out gives the expected results, namely, the admin receives a warning message via the Telegram Bot.

Keywords: Microservices, Server Monitoring, Bot

I. INTRODUCTION

Computing needs will continue to grow, and the development of services provided by an institution through information technology. In computer networks, more efficient techniques have been emerging in its development to provide services on a wide scale. Computer networks allow for centralized computing processes and distributed computing. Nowadays, the computing needs in service provision are increasing. This provision impacts the increasing volume of data traffic simultaneously, the demand for mobile services, and increasing flexibility. Increasing services through computer networks must be balanced with the increase in computing resources to trigger an overload of work on servers and their supporting devices [1].

Recently, the microservices paradigm that uses *container* technology, namely packages to run applications without virtual machines, is becoming increasingly popular. Microservices are an architecture based mostly on separate autonomous services that can be developed, deployed, and operated independently of one another [2]. Microservices have several challenges regarding team organization, development practices, and the infrastructure used [3]. In this regard, two critical challenges require support for the management and monitoring of microservices system servers. The independent nature of the microservices-based system and a highly dynamic distributed system urgently need the support of monitoring services and management of these services [4]. The first challenge concerns the relationship between software architecture and software teams. If the relationship between the software architecture and the software team is appropriate, it will be seen in the microservices system architecture. In the microservices system architecture, software teams can operate independently of each other. The second is the distributed nature and details of microservices. Microservices-based systems can consist of many independently developed and deployed services, which interact while they are running. Due to this independent nature, the interaction between the service and the overall system architecture becomes apparent as it is running.

This research is applied to an emergency response system that provides services in reporting and action, namely Tanggapin. Tanggapin is a mobile-based and web-based application that can be accessed according to the user's needs and conditions. Due to the complex system services with various supporting features, a microservices architecture is implemented independently and applied in a distributed manner. Therefore, keeping for monitoring and service management is crucial. According to Cinque et al. [5], Monitoring activities are essential in any software system. The increasing use of microservices architectures will complicate the monitoring process and create challenges due to the log data source used for the monitoring process. It is more comfortable for the team to manage and monitor the microservices system using a different machine from the microservices server.

Many studies on the use of bots to monitor servers have been conducted. Studies on the use of Telegram Bot as a monitoring tool for various applications have been widely carried out. Telegram Bot is used to integrate complaints of services available on university websites. With this application, a student does not need to visit the university website, just fill in a complaint via Telegram [6]. Telegram Bot is also used to guide non-expert users in monitoring radiofrequency links' feasibility, namely BotRf. BotRf is applied to smartphones and personal computers for bandwidth-limited environments [7].

On the other hand, Telegram Bot is also used to facilitate information dissemination on campus. Having the ability to handle multiple requests simultaneously, this bot can provide information according to user requests [8]. Meanwhile, Botanicum is a Telegram Bot application for classifying trees based on leaf images. This bot can classify approximately twenty different tree species in Russia. Interaction user with bots is carried out using simple conversation and brief instructions on how to take photos. The resulting classification accuracy reached 97.8% [9]. In this study, Telegram Bot is used as a notification sender to the user's telegram application when a microservices system experiences problems. The bot also provides easy access to time and place as a medium connected to the mobile microservices system via a smartphone device. Using the Telegram Bot in this study is a medium to provide easy access and notification from the microservice system. The use of Telegram Bot service requires its device to run smoothly. For this reason, the server is used as a place for installing Telegram Bot to make it more efficient. Because the server is a separate device from the microservices system, the use of Telegram Bot will not burden the processes, memory, and services of other devices.

II. METHOD

This study uses the PPDIOO *Lifecycle Approach to Network Design and Implementation* method [10]. This method is widely used in the development and management of a computer network. PPDIO is implemented in six phases as an uninterrupted process flow. In this method phase, there is no beginning and end of the development of a computer network, which is described in six stages as follows:

1) Preparation: The preparation stage in our study involves the current requirements for computer network development strategies. It is also the technology preparation that is most suitable for the existing architecture. One of them is the use of Docker to perform testing as it can isolate all processes into a specific server.

2) Planning: Our study's planning stage involves identifying network requirements based on user needs, objectives, facilities, and other planning aspects. Specifically, our study assesses the existing network's characteristics and conducts a gap analysis of whether the existing infrastructure can support the proposed system's development. This part of the planning stage is also useful so that the network monitoring development process is aligned with the parameters of resource requirements.

3) Design: Our study's design stage involves the design and specification aspects of the network as a comprehensive and detailed design to meet the current engineering aspects. Our study aims to provide a design system that improves the availability, scalability, performance, and reliability of network services to be provided in the design stage. The availability and scalability improvement will be carried out by adding more components to a complete system design. On the other hand, the performance and reliability improvement will be implemented by design specifications that have been carefully compiled use as the basis for implementing computer network development.

4) Implementation: The implementation stage in our study produces a computer network with a reliable monitoring system or the addition of new components that are integrated with the existing network infrastructure based on design specifications and objectives. The main principle is to maintain the stability of the current network without creating new vulnerabilities.

5) Operation: The operational stage is considered as the final test of the conformity of network development with the design. In our study, this phase involves maintaining the network through day-to-day operations. Deviation detection, corrections, and performance monitoring are carried out to provide data for the optimization phase.

6) Optimization: It involves proactive and responsive network management. In our study, a proactive and responsive management strategy is used to identify and resolve previous problems that affect the organization and users' needs. Proactive and responsive error detection and correction are required for optimal network performance. In-network development with the PPDIOO method, the optimization phase can require network re-planning if performance does not meet the goals and expectations.

The PPDIOO method requires that all stages be carried out in a structured and sequential manner, as illustrated in Fig. 1.

To implement this method, this research requires hardware and software support as follows::

1) *Hardware*
   a. Server Intel® Xeon® E5-2650 v2 @ 2.6 GHz, DDR4 16 GB.
   b. Local network devices and the internet. Network devices with high speed are essential to maintain the quality and stability of communication between servers.
   c. Smartphones are used as the messenger application's installation device, which receives notification messages sent via the Telegram Bot.
2) *Software*
   a. Ubuntu Operating System 18.04 LTS is used as the operating system installed on all servers.
   b. Docker is used as the primary tool installed on all computer servers.
   c. Prometheus is software that functions to collect metrics from targets that running on docker and server.
   d. Nodeexporter is Prometheus client libraries that are installed on Docker, and also installed on each computer server.
   e. Alertmanager can handle alert messages sent from Prometheus and routed to recipients.
   f. Telegram Messenger is a messenger application installed on the user's Smartphone, which is used as a device for receiving warning messages.

Fig. 1 PPDIOO development method

III. RESULT AND DISCUSSION

Tanggapin is an integrated emergency response system implemented in three districts in Central Java, namely Kabupaten Batang, Kabupaten Kendal, and Kabupaten Demak. Regarding this study, which is conducted on Tanggapin using the method explained above, the implementation of the method is described as follows:

1) The preparation and planning stage identifies the requirements that apply to Tanggapin. It is used to provide Tanggapin with computer network development strategy and propose the most appropriate technology for the current architecture. The use of Docker is to perform testing as it can isolate all processes into a specific server.

(2) Design stage. The design stage involves the design aspects and network specifications as a comprehensive design to meet the engineering aspects Tanggapin. The system uses a computer server that functions as a monitoring server as well as a place to install Docker and Telegram Bot running in Docker. This design process is focused on analyzing hardware requirements, analyzing software and operating system requirements, as well as network architecture and topology. The network topology used in the monitoring system is depicted in Fig. 2. The computer server that functions as a monitoring server is used to record all activity and data traffic on the Docker network and also to monitor the microservices system [11]. Meanwhile, the Telegram on computer server functions as an API service provider that will be used in the development—the client functions as a recipient of notification messages sent from the Telegram Bot.

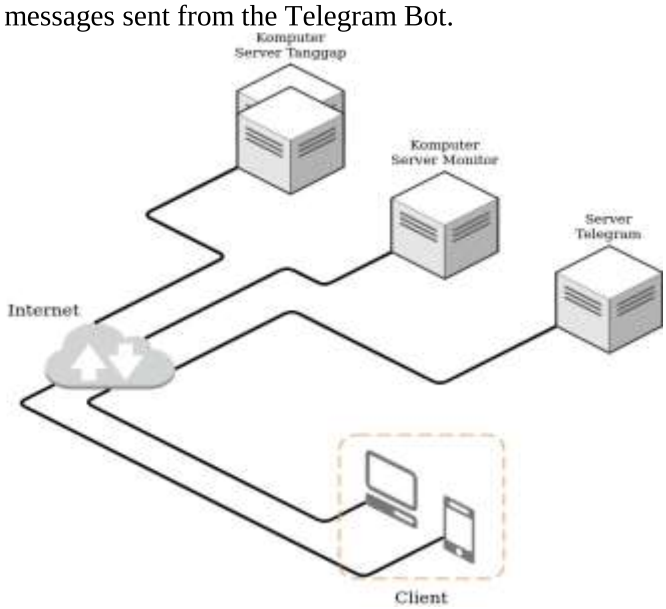

Fig. 2 Star network topology used in this study

(3) Implementation Phase. As explained above about the software requirements, the use of Docker, which is to help to make the testing of the system easier. Because it can isolate all processes that are running on one computer server [12]. Hence, the test will run optimally. In a sequential manner, the software installation on the server starts from the installation of (a) the Ubuntu Server 18.04 LTS operating system, (b) the monitoring server configuration, to communicate with each other between the monitoring server and the microservices server, (c) Installing the Docker engine, (d) Docker compose installation, (e) installation and configuration of Traefik used to balance microservices server workloads, (f) Prometheus installation and configuration, (g) Alertmanager installation and configuration, (h) Nodeexporter installation and configuration, (i) Prometheus integration and Alertmanager, (j) Integration of Nodeexporter and Prometheus, (k) Integration of Traffic and Prometheus, (l) up-down monitoring of server computers, (m) up-down monitoring of microservice services, (n) grouping of computer server metrics data and (o) monitoring the server status.

In accordance with the design that has been provided, then testing of Nodeexporter is carried out so that it can collect metrics data from the computer server and then send it to Prometheus, which will later be processed and executed by Prometheus. Fig. 3 depicts the alert rule monitoring server. It is used to test the warning rule configuration on the Alertmanager to determine if a service has a problem or is down. Testing on the Telegram Bot API is carried out, but with the condition that only certain Telegram accounts can receive notification messages via the Telegram messenger application. After simulating and getting the desired results, the system will be ready to be implemented.

(4) Operational Phase. The operational phase is considered the final test of the network development's suitability to the design [13]. A script using the extension ".rules" is created to perform up/down monitoring of the computer. It is created using the command ~ *$ touch alert-server.rule*, and the file is used to determine alert conditions based on metrics data collected by Prometheus and using Prometheus expressions. The Server Microservices monitoring architecture with Telegram Bot was developed on Tanggapin, depicted in Fig. 4.

```
groups:
- name: tanggap_server_frontend
  rules:
  - alert: tanggap_server_frontend_down
    expr: up{instance="node.tanggap.in:80"} == 0
    for: 30s
    labels:
      severity: critical
    annotations:
      summary: "Monitor server tanggap down"
      description: "Server tanggap is down. \
                    Reported by instance {{ $labels.instance }} of job {{ $labels.job }}."

- name: tanggap_server_backend
  rules:
  - alert: tanggap_server_backend_down
    expr: up{instance="node.gcp.tanggap.in:80"} == 0
    for: 30s
    labels:
      severity: critical
    annotations:
      summary: "Monitor server tanggap down"
      description: "Server tanggap is down.
                    Reported by instance {{ $labels.instance }} of job {{ $labels.job }}."
```

Fig. 3 Script alert rule monitoring server

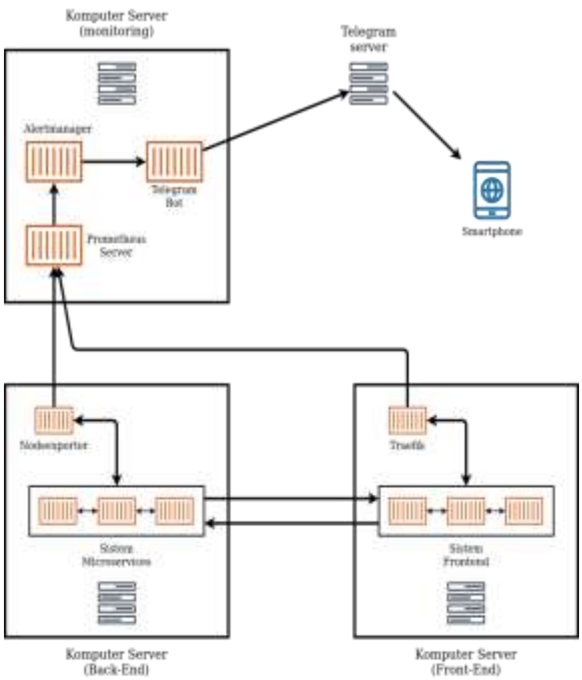

Fig. 4 The architecture of monitoring system

In Fig. 4, the computer server is used to run supporting software such as docker, and in Docker there are several supporting services and tools running. Computers servers are connected to the internet and functioning as internet network providers and network management for Docker. On the Smartphone, the Telegram messenger application will be installed, which is used to connect with the Telegram Bot installed on Docker.

(5) Optimation Phase. The optimization phase is carried out after considering that the condition through the information obtained from monitoring has been built both on the hardware and software sides. Based on the operational phase that has been carried out, it is assessed whether the system is running well according to the design that has been made or not. Optimization is done after checking to ensure whether the system is running according to the configuratoin setting that has been set or not. Management and maintenance are carried out so that the network system runs according to quality standards and services.

In this study, testing was carried out using several types of problem conditions in one of the microservices system's services.

A. Testing of the monitoring system

Testing the monitoring system is carried out to determine whether the monitoring system is running as expected or not; hence, testing is carried out several times. From the tests that have been carried out repeatedly, it can be concluded that the warning message *down* on one of the service containers from the microservice system can be received via the Telegram Bot three times at different times. Table I below describes the monitoring system testing scenario.

Based on Table I, server monitoring testing is carried out with six server conditions, where each condition is conducted using a different test method. In each test method, the test was performed three times with the results shown in Fig. 5. The results explained that the average time of notification appearance after six types of testing is 2,4 minutes.

TABLE I
THE TESTING SCENARIO OF THE MONITORING SYSTEM

No	Testing aspects	Ways of testing
I	Down of computer server	Shut down the computer server and turn it back on. Done in several tests.
II	CPU overload of the computer server	Simulate the use of CPU on the computer server using *"stress-ng"* tools. It is carried out for several tests.
III	Memory overload of the computer server	Simulates memory usage on the computer server using the "memtester" tool. It was done several tests.
IV	Microservices system down	Turn down one of the responsive microservices system services and turn it back on
V	CPU overload of Microservices system	Simulates CPU usage on one of the responsive microservices system services using the "stress-ng" tool. It was done in several tests.
VI	Memory overload of Microservices System	Simulates memory usage on one of the responsive microservices system services using the "memtester" tool. It was done in several tests.

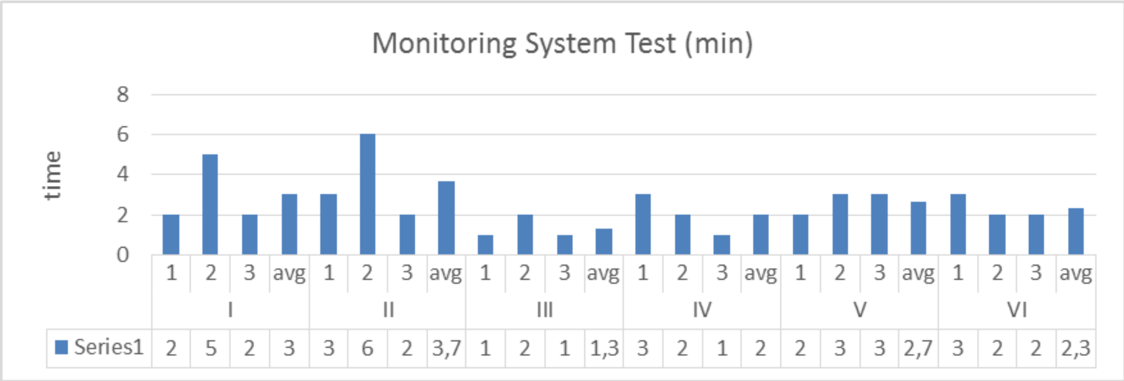

Fig. 5 Testing of monitoring system

B. Telegram Bot testing

A bot is a computer program that handles tasks in an automated manner [14]. To provide a better bot performance, bot testing is crucial [15]. The test is carried out to determine whether the Telegram Bot responds to commands entered and runs as expected. The Telegram Bot test scenario is described in Table II. The scenario of bot testing is carried out with three testing conditions.

The result of the testing scenario based on three commands, i.e., alerts, silences, and status, is presented in Fig. 6. The server feedbacks are received in real-time when the server experienced a sudden downtime, overload, or warning. The notifications in the monitoring system appear 2.4 minutes after the incident, whereas the response time is at an average of 1.33 seconds.

TABLE II
TELEGRAM BOT SCENARIO TESTING

No	Testing
I	By entering the command "*/alerts*."
II	By entering the command "*/silences*".
III	By entering the command "*/status*".

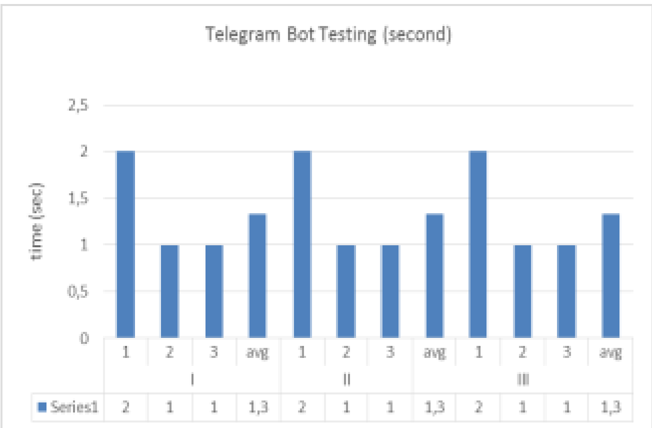

Fig. 6 Test of telegram bot

IV. Conclusion

This study produces a microservices server monitoring system with Telegram Bot. From the results of the tests that have been carried out, it can be concluded that the monitoring system can run well because every test carried out gives the expected results; namely, the admin can receive a warning message via the Telegram Bot. The author concluded that the messages are received in real-time when the server computer/microservices system experienced a sudden downtime or overload and warning. The appearance of automatic notification in the monitoring system is around 2.4 minutes after the incident. Meanwhile, by entering the test command/alerts, / silences, and / the average on the Telegram Bot test, its response at an average of 1.33 seconds. With this monitoring system, when the server computer/microservices system is down or overloaded, the damage information can be conveyed to the admin in real-time. It will be easier to get this information and be quickly respond to repair when damage occurs.

References

[1] D. Richter, M. Konrad, K. Utecht, and A. Polze, "Highly-Available Applications on Unreliable Infrastructure: Microservice Architectures in Practice," in *2017 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, Jul. 2017, pp. 130–137, doi: 10.1109/QRS-C.2017.28.

[2] "Microservices: Flexible Software Architecture [Book]." https://www.oreilly.com/library/view/microservices-flexible-software/9780134650449/ (accessed Aug. 27, 2020).

[3] P. Jamshidi, C. Pahl, N. C. Mendonça, J. Lewis, and S. Tilkov, "Microservices: The Journey So Far and Challenges Ahead," *IEEE Softw.*, vol. 35, no. 3, pp. 24–35, May 2018, doi: 10.1109/MS.2018.2141039.

[4] B. Mayer and R. Weinreich, "A Dashboard for Microservice Monitoring and Management," in *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*, Apr. 2017, pp. 66–69, doi: 10.1109/ICSAW.2017.44.

[5] M. Cinque, R. Della Corte, and A. Pecchia, "Microservices Monitoring with Event Logs and Black Box Execution Tracing," *IEEE Trans. Serv. Comput.*, pp. 1–1, 2019, doi: 10.1109/TSC.2019.2940009.

[6] M. A. Rosid, A. Rachmadany, M. T. Multazam, A. B. D. Nandiyanto, A. G. Abdullah, and I. Widiaty, "Integration Telegram Bot on E-Complaint Applications in College," *IOP Conf. Ser. Mater. Sci. Eng.*, vol. 288, p. 012159, Jan. 2018, doi: 10.1088/1757-899X/288/1/012159.

[7] M. Zennaro, M. Rainone, and E. Pietrosemoli, "Radio Link Planning Made Easy with a Telegram Bot," in *Smart Objects and Technologies for Social Good*, Cham, 2017, pp. 295–304, doi: 10.1007/978-3-319-61949-1_31.

[8] H. Setiaji and I. V. Paputungan, "Design of Telegram Bots for Campus Information Sharing," *IOP Conf. Ser. Mater. Sci. Eng.*, vol. 325, p. 012005, Mar. 2018, doi: 10.1088/1757-899X/325/1/012005.

[9] D. Korotaeva, M. Khlopotov, A. Makarenko, E. Chikshova, N. Startseva, and A. Chemysheva, "Botanicum: a Telegram Bot for Tree Classification," in *2018 22nd Conference of Open Innovations Association (FRUCT)*, May 2018, pp. 88–93, doi: 10.23919/FRUCT.2018.8468278.

[10] "PPDIOO Stages > Cisco's PPDIOO Network Cycle | Cisco Press." https://www.ciscopress.com/articles/article.asp?p=1697888&seqNum=2 (accessed Aug. 27, 2020).

[11] D. Jaramillo, D. V. Nguyen, and R. Smart, "Leveraging microservices architecture by using Docker technology," in *SoutheastCon 2016*, Mar. 2016, pp. 1–5, doi: 10.1109/SECON.2016.7506647.

[12] C. Kaewkasi, *Docker for Serverless Applications: Containerize and orchestrate functions using OpenFaas, OpenWhisk, and Fn*. Packt Publishing Ltd, 2018.

[13] J. Mohorko, F. Matjaž, and K. Saša, "Advanced Modelling and Simulation Methods for Communication Networks," p. 6, 2008.

[14] S. Basso, A. Servetti, and J. C. De Martin, "The network neutrality bot architecture: A preliminary approach for self-monitoring of Internet access QoS," in *2011 IEEE Symposium on Computers and Communications (ISCC)*, Jun. 2011, pp. 1131–1136, doi: 10.1109/ISCC.2011.5983857.

[15] R. van Tonder and C. Le Goues, "Towards s/engineer/bot: Principles for Program Repair Bots," in *2019 IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE)*, May 2019, pp. 43–47, doi: 10.1109/BotSE.2019.00019.