MATICS : Jurnal Ilmu Komputer dan Teknologi Informasi (*Journal of Computer Science and Information Technology)*
ISSN : 1978-161X(p); 2477-2550(e)
Volume 15, No. 2(2023), pp 90-95
Received : October 9th 2023 ; **Accepted** : October 18th 2023 ; **Avalaible Online** : October 23th 2023

Performance Analysis of Devops Practice Implementation Of CI/CD Using Jenkins

Rismanda Tyas Kusumadewi, Ronald Adrian

Abstract— **Continuous Integration and Continuous Delivery (CI/CD) are methods used in agile development to automate and speed up the process of building, testing, and validating services. To support and simplify all development and deployment processes, several methods such as containerized and CI/CD automation are needed. In this research, a DevOps Practice is carried out which includes process integration, deployment, and testing automatically using a tool called Jenkins. These tools are open source automation servers to help the Continuous Integration and Continuous Deployment process. Jenkins is equipped with various open source plugins that can be used to simplify and assist CI/CD automation and testing processes. The implementation of CI/CD in performance testing makes the testing process integrated, automated, and can be run on a regular basis.**

Index Terms— **Automation, CI/CD, DevOps, Jenkins.**

I. Introduction

CI/CD is a method of delivering applications on a regular basis to customers by bringing automation into the application development phase. However, organizations often face obstacles such as inefficiency, implementation delays, sluggish behavior, and lack of automation when practicing CI/CD. These constraints can cause confusion between product delivery paths and take up system resource capacity. Thus, emphasizing the impact of human factors, CI/CD performance has become a popular field in software development research. In its application, CI/CD requires supporting tools to facilitate the process. Such as Git which is used as source code management and Jenkins as a tool to support the automation process [1]

Often developers work and collaborate with teams to complete their work assignments. Because the challenges of working with a team require continuous communication, and require good task resource management, developers need supporting tools that are used to perform resource management and interact with each other. That way jobs that have a high load can be handled easily using these supporting tools. In addition, because of the need for developers who work in teams, in the process of working on a task until the task is completed, developers need to review each other so that the work they are doing is sustainable and in line with expectations. Therefore, source code management is a supporting tool needed in the application system development process.

Process automation is often used in the industrial world to make work easier. With the automation process, developers will find it easy to integrate on a regular basis. This automation process is carried out with the help of supporting tools that are integrated with source code management. This automation process is the most important part of the CI/CD process. With the automation process, work that is done manually can be done and completed quickly by the automation process.

In general, developers do testing in application development before the application is sent to the production environment to ensure whether the application is feasible to launch and will not throw errors when the application is running. One of these tests is performance testing where this test has an important role in the application development process. This performance test includes stress point testing with additional load methods, scalability testing, and stability testing.

In this study, a performance analysis of the CI/CD implementation will be carried out on a simple web application. Analysis activities include performance testing by utilizing an application called BlazeMeter. In addition, to get data in real time, this research provides a development by sending notifications through the Slack application [2]. It is hoped that the automation process for testing CI/CD performance in this study can be done automatically and reduce the role of humans in testing. In addition, this research analysis is expected to reduce the problems of inefficiency, implementation delays, and sluggish behavior when practicing CI/CD [3].

II. Research Method

The research method carried out by the author can be seen in the flow chart shown in Figure 1.

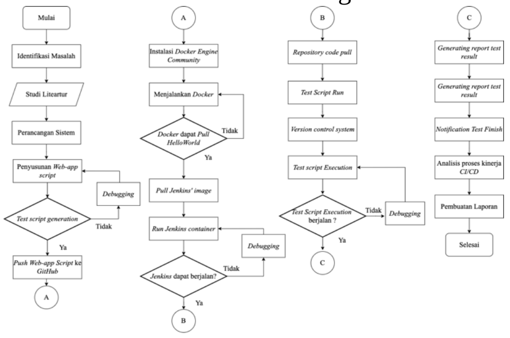


Fig. 1. Research Flowchart

Manuscript received October 9, 2023. This work was supported in part by Universitas Gadjah Mada.

Rismanda Kusumadewi is with the Departemen Teknik Elektro dan Informatika, Universitas Gadjah Mada, Yogyakarta, Indonesia (email rismandatyas@mail.ugm.ac.id)

Ronald Adrian was with Departemen Teknik Elektro dan Informatika, Universitas Gadjah Mada, Yogyakarta, Indonesia (e-mail: ronald.adr@ugm.ac.id).

Based on the research flow chart, there are several stages carried out in this study. The stages of the research are as follows:

2.1 Design Stage

In this stage, a topology design is carried out which will be used as a reference for performance analysis of Web-app deployments on Jenkins. Figure 2. shows how the topology is used for this study. The topology describes the CI/CD pipeline process flow using Jenkins. The CI/CD process starts from the developer user building a python source code. Then by using Git, the python source code is pushed to the GitHub repository that has been created. Then the role of Jenkins begins to perform code integration by checkout python source code management and build configuration. At the build stage, Jenkins integrates the code that has been obtained from checkout to build an image which, when executed, will become a container containing the application to be run. All these processes will be recorded by a Jenkins plugin which integrates the build process with the real-time notification system using the. This process from building source code to running container is called CI/CD.

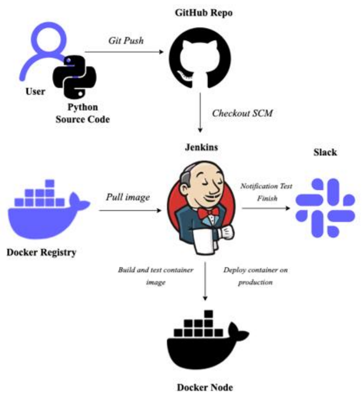


Fig. 2. Design Research Topology

2.2 Installation and Configuration Stage

At this stage, the installation and configuration of the tools that will be used in this research are carried out. It can be seen that the installation and configuration steps required are as follows:

2.2.1Installing Docker Engine Community (Docker Desktop)

The Docker Engine Community (Docker Desktop) installation is used to unify applications with the containerized method. An indication that the Docker Desktop installation was successful is that you can pull "helloworld". The pull will display a message that the Docker installation went smoothly.

2.2.2Installation and Configuration Environment

The installation environment for Jenkins uses a Jenkins image that has been pulled from the Docker Image Registry. To be able to access Jenkins via localhost, it must be ensured that the host has allowed the associated port to be accessed from outside. The port used for Jenkins environment is port 8181.

2.2.3Jenkins Configuration

To run the pipeline CI/CD process. The first CI/CD. Process is to pull the source code from the previously created GitHub Repository. Then after making sure all the configurations and source code are correct, then the next step is to carry out the build process. The indication of the build process was successful if the test script execution did not experience an error. At each stage of the build will create a trigger notification in Slack that notifies the status of the build process.

2.2.4 Test Report Configuration

Configure the generating test report where the results of the build process will be generated using a performance test tool called BlazeMeter. Furthermore, a summary report will be obtained for functional and performance tests which are then analysed and documented in a written report.

2.2.5 Configure Slack Notifications

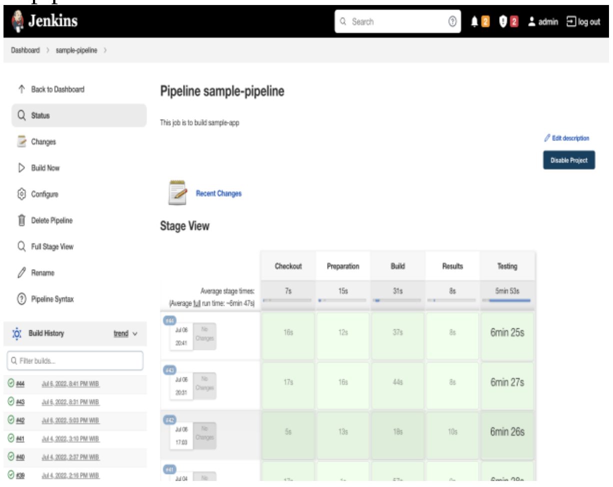
Slack's notification configuration aims to generate notifications and find out the initial status and whether the final status of the job pipeline is successful or failed.

III. Result and Discussion

This study has a scope of testing to compare performance tests between web-app deployments using Jenkins services and conventional web-app deployments via local machine terminals. The web-app integration outputs a simple dynamic web application display that displays the client's IP address when the web-app is accessed.

3.1 Web-App Display Integration

The web-app integration results from the build on the job pipeline. The job of the configured job pipeline is to run the web-app integration via the build job. Figure 3 is a sequential web-app integration process with job pipelines.



Pipeline sample-pipeline

This job is to build sample-app

Stage View

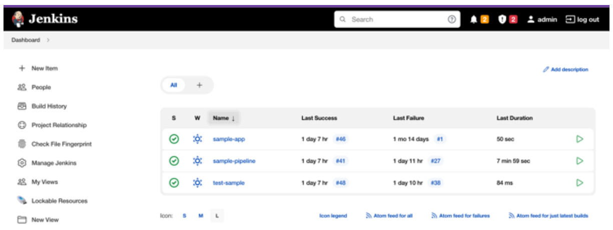
	Checkout	Preparation	Build	Results	Testing
Average stage times: (Average full run time: ~6min 47s)	7s	15s	31s	8s	5min 53s
#44 Jul 06 20:41 No Changes	16s	12s	37s	8s	6min 25s
#43 Jul 06 20:31 No Changes	17s	16s	44s	8s	6min 27s
#42 Jul 06 17:03 No Changes	5s	13s	18s	10s	6min 26s
#41 Jul 04 No	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

Build History

Filter builds...

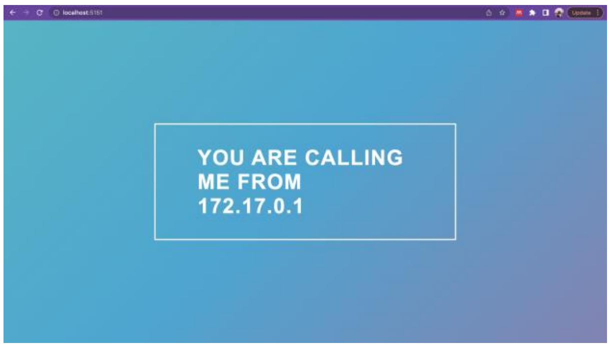
#44	Jul 6, 2022, 8:41 PM WIB
#43	Jul 6, 2022, 8:31 PM WIB
#42	Jul 6, 2022, 5:03 PM WIB
#41	Jul 4, 2022, 3:10 PM WIB
#40	Jul 4, 2022, 2:37 PM WIB
#39	Jul 4, 2022, 2:16 PM WIB

Fig. 3. Pipeline Sequencial History



S	W	Name ↓	Last Success	Last Failure	Last Duration
✓	☼	sample-app	1 day 7 hr #46	1 mo 14 days #1	50 sec
✓	☼	sample-pipeline	1 day 7 hr #41	1 day 11 hr #27	7 min 59 sec
✓	☼	test-sample	1 day 7 hr #48	1 day 10 hr #38	84 ms

Fig. 4. Job List



YOU ARE CALLING ME FROM 172.17.0.1

Fig. 5. Web-app View of Jenkins Integration

In the pipeline there are several stages that represent the sub-tasks for the integration process. The green pipeline indicates that the stages have a successful status. Within each stage is displayed a time called a timestamp which is the time span of the process from which the stages took place. The jobs run by the pipeline are shown in Figure 4. The build jobs and test jobs can be seen on the Jenkins dashboard together with the job pipeline. To view the detailed contents and configuration of each job, click on the job name. The build job (sample-app) is tasked with integrating Docker containers derived from Docker web-app files so that web-apps can be run. Meanwhile, the test job will verify whether the web-app has been running successfully. Each job has a console output that is used to view the job process when it is run. The display of the results of the integration and deployment of the web-app is shown in Figure 5. The appearance of the web-app is generated by the css configuration that has been created and integrated with the html script. In the middle of the web-app display, it can be seen that the accessing ip is 172.17.0.1 which is the ip of the client who is accessing the web-app.

3.2 Results of Integration Performance Testing Using Jenkins

Table 1. Integration Performance Testing with Jenkins

Virtual User	Time	Throughput (hit/s)	Response Time (ms)	Latency
1	18:54:40	2.3	39.09	15
2	18:54:50	68.6	20.69	8.74
3	18:54:55	78.9	29.8	8.86
4	18:55:00	84.7	37.7	16.76
5	18:55:05	83.2	46.13	23.06
6	18:55:10	82.8	58.05	26.62
7	18:55:20	96.3	67.65	32.77
8	18:55:25	107.8	65.7	31.56
9	18:55:30	128.9	63.24	30.54
10	18:59:20	202.4	48.9	23.81

Based on the test results obtained on the web-app integration test using Jenkins, the data results obtained represent the value of web-app integration performance on increasing the number of concurrent users with a test time of 6 minutes. It can be seen in table 1 regarding the results of data throughput, response time, and latency, these three parameters are related and correlated with each other. When the throughput graph increases, the response time and latency graphs will decrease. Changes in the values of these three parameters can be influenced by several factors, such as concurrent users, machine tools used, traffic density, and the type of connection or network used. Therefore, the values for these three parameters have changed to the concurrent users. It can be seen in table 1. throughput testing, the results of the test values obtained mean that the average throughput value on integration using Jenkins has a fairly good ability or speed. This can be seen in the average number of throughput generated by 93.59 hits/s or it can be interpreted that the average HTTP requests transferred are 93 requests per second. This means that the level of ability to handle additional loads is quite good. In addition, with the number of requests and concurrent users accessing the web-app simultaneously, the response time and latency values that can be seen in table 1 are relatively close to 0, namely the average response time is 0.047 s and the average delay value is 0.021 s. That way the response time and latency conditions meet the ideal value which means that the response time of request processing and delay in the Jenkins integration process is very good.

It can be seen in table 2, the parameters used to test engine health are network I/O, memory usage, CPU usage, and connection. The data is retrieved based on the test time range defined in the yaml file. This parameter is part of determining the availability and ability of the test engine to run tests. Machine health testing is also needed to determine how many users the machine can support. It can be seen in table 2 that there was an increase after the engine test was run. This increase in network I/O was caused by the number of connections and the addition of users that occurred in the testing process [4]. The resulting network I/O value represents the amount of data that flows or is transferred in the I/O network on the engine test used in this study. Based on research [5], the supporting factor for network I/O depends on the redirector on the network protocol used. In addition, network I/O depends on how many I/O operations are performed on the network and the speed of the network connection.

Table 2. Engine Health Test

Time	*Network i/o (KB/s)*	*Memory (%)*	*Cpu (%)*	*Connection*
18:54:40	1.38	54.2	91.8	3
18:54:50	98.61	54.8	76.5	3
18:54:55	179.86	55.8	73.3	5
18:55:00	159.54	56.2	83.2	6
18:55:05	193.4	56.3	84.2	8
18:55:10	238.88	56.4	76.8	8
18:55:20	242.76	56.4	68.2	9

18:55:25	183.03	56.5	78.7	9
18:55:30	277.52	57	65.5	12
18:55:56	239.41	59.2	95.7	10
18:56:14	179.24	60.2	94.7	10
18:56:34	369.33	61	65	11
18:56:45	477.21	61	57.1	10
18:56:59	460.32	61	60.8	10
18:57:06	362.57	61	69.4	13
18:57:22	369.76	61	73.5	9
18:58:22	341.31	61.1	64.5	13
18:58:49	498.3	61.1	61.9	5
18:59:20	443.62	61.1	62.1	15
18:59:36	222.37	61.1	76.9	13
19:00:48	211.11	52.4	38.4	2

Memory usage and CPU usage are also very important to note. CPU and memory usage in the test engine, it is not recommended to exceed the normal threshold. To maintain the health of the machine, CPU usage is not recommended to exceed 80% and memory usage is not recommended to exceed 70% [6]. Memory usage data in engine health testing on integration using Jenkins is relatively constant with an average memory usage of 56.7%. While the data from the CPU usage in this test resulted in a relatively high data value at the initial conditions of the machine performing the integration. CPU usage in the integration process has relatively increased with CPU usage by 80% and after concurrent user conditions have been achieved, CPU usage becomes relatively constant at 68.6%. It can be concluded that the memory and CPU usage in testing engine health on integration using Jenkins has a fairly good value because after being in a stable condition, memory and CPU usage does not exceed the normal threshold.

3.3 Conventional Integration Performance Testing Results

Table 3. Conventional Integration Performance Testing with Jenkins

Virtual User	Time	Throughput (hit/s)	Response Time (ms)	Latency
1	4:40:40	5.1	19.65	10.1
2	4:40:45	49.3	17.2	8.2
3	4:40:50	130.2	11.17	4.56
4	4:41:00	176.8	18.01	7.93
5	4:41:05	180.1	24.23	10.2
6	4:41:10	184.4	26.2	12.12
7	4:41:15	173.5	30.7	16.72
8	4:41:20	165.5	39.6	18.47
9	4:41:30	119.7	70.79	34.85
10	4:44:00	190	51.98	25.1

The second performance test in this study was carried out conventionally by using the terminal as a command processing tool to perform web-app integration. This test is carried out to determine the performance test parameters which test parameters are material for identification and evaluation of the performance of the conventional integration process. Based on previous research by [7]. Performance testing is conventionally a test that generally runs on the basis of resources on hardware or personal devices. In this research, integration and deployment are conventionally done using personal devices via terminals. All files are put together in a folder and to call the python script, you must first enter the folder. This conventional integration process requires repeated configurations and is done manually so that the process is quite time consuming.

Based on the test results obtained on conventional web-app integration testing, the data results obtained represent the value of web-app integration performance on increasing the number of concurrent users with a test time of 6 minutes. It can be seen in the flow chart of throughput, response time, and latency, in table 3, these three parameters are related and correlated with each other. When the throughput graph increases, the response time and latency graphs will decrease. Changes in the values of these three parameters can be influenced by several factors, such as concurrent users, machine tools used, traffic density, and the type of connection or network used. Therefore, the values for these three parameters have changed to the concurrent users. It can be seen in table 3 throughput testing, the results of the test values obtained mean that the average throughput value in conventional integration has good capability or speed. This can be seen in the average number of throughputs generated by 137.59 hits/s or it can be interpreted that the average HTTP request transferred is 137 requests per second. This means that the level of ability to handle additional loads is quite good. In addition, with the number of requests and concurrent users accessing the web-app simultaneously, the response time and latency values that can be seen in table 3 are relatively close to 0, namely the average response time is 0.0309 s and the average delay value is 0.014 s. That way the response time and latency conditions meet the ideal value which means that the response time of request processing and delay in the conventional integration process is very good.

From the results of the engine health test, it was found that the resource utilization carried out in conventional web-app integration testing was not good. This is evidenced by the data obtained from the memory usage on the engine test, namely personal devices with IDs that have relatively variable values due to the high level of interference with the use of personal machines. Based on research [5]. The supporting factors for network I/O depend on the redirector on the network protocol used. In addition, network I/O depends on how many I/O operations are performed on the network and the speed of the network connection.

Table 3. Conventional Engine Health Test

Time	*Network i/o (KB/s)*	*Memory (%)*	*Cpu (%)*	*Connection*
4:40:40	10.23	69.1	72.7	36
4:40:45	2.03	69.1	66.9	36
4:40:50	300.29	69.7	80.1	37
4:41:00	625.75	65.2	83.1	48
4:41:05	888.68	67.1	87.7	49
4:41:10	966.34	68.8	86.3	47
4:41:15	905.94	66.9	85.4	51
4:43:30	843.13	66.2	68.7	51
4:43:41	782.79	66.9	70.3	53
4:44:00	872.67	66.1	69.8	63
4:44:10	983.32	66.1	70.9	62

4:44:36	980.44	64.6	70.1	51
4:44:42	844.33	65.1	72.3	60
4:45:23	824.54	65.2	67.3	58
4:45:35	851.55	66.1	67.1	48
4:46:01	769.4	68.6	70.3	58
4:46:21	1025.18	67.8	72.1	47
4:46:45	766.83	65.2	67.3	47

Memory usage and CPU usage are also very important to note. CPU and memory usage on the test engine, it is not recommended to exceed the normal threshold. To maintain the health of the machine, CPU usage is not recommended to exceed 80% and memory usage is not recommended to exceed 70% [6]. In table 4 the data on the results of memory usage in engine health testing on conventional integration are relatively variable with an average memory usage of 67.6%. While the data from the CPU usage in this test results in a high data value at the initial conditions of the machine performing the integration. CPU usage in the integration process has relatively increased with CPU usage by 80% and after concurrent user conditions have been achieved, CPU usage becomes relatively constant at 77.8%. Thus, it can be concluded that the use of memory and CPU in testing engine health on conventional integration has a fairly good value because after being in a stable condition, memory and CPU usage does not exceed the normal threshold. CPU usage in the engine test has a value that changes quite a bit at the beginning of the web-app integration because the test engine used requires booting and warm booting (start up section) as well as interference with the use of other applications. so it requires more CPU resources.

3.4 Comparative Analysis of Integration Performance Using Jenkins and Conventional Integration Performance

Based on the analysis of performance test results between integration using Jenkins and conventional integration. The throughput value in conventional testing has a higher average value than the average throughput value on integration using Jenkins. This condition is caused by differences in network media and devices used. The throughput value on the device is relatively higher because the network media used directly leads to the web-app integration process, while the web-app integration uses Jenkins, the network media used is divided by the use of Jenkins and Docker containers. That way the throughput value will affect the response time and latency values in the testing process. The higher the throughput value, the lower the response time and latency values.

In the engine health test, the test data on network I/O and connections on conventional integration are higher than those with integration using Jenkins. This condition is caused by differences in the network media used and the test engine used. Personal devices used for conventional integration have higher specifications than the Docker container which is used as a test engine for the integration process using Jenkins so that the network I/O and connection data obtained will be affected. Whereas in resource testing, namely CPU and memory, conventional integration produces higher data due to interference with resource use combined with the use of other applications running simultaneously. Continuously high levels of CPU and memory usage will make the device quickly experience system errors and damage.

Based on the analysis data from the previous results, Jenkins has continuous functionality and the integration process using Jenkins can be done automatically and only needs to be manually configured when configuring the desired job for the first time. Whereas conventional integration requires repeated configuration and is done manually. So it can be said that the integration process using Jenkins is more efficient than conventional integration [8]. This is what makes CI/CD practices widely used in industry, because CI/CD practices are more efficient, effective, and cost-effective.

IV. Conclusion

CI/CD is a series of activities in DevOps practice that simplify the application integration process and enable applications to continue to integrate and deploy on an ongoing basis. The integration process using Jenkins makes it easy and time efficient for developers to do application development because it uses an automation system from the CI/CD process. Performance testing is carried out to find out and ascertain whether a system will not produce errors at run time and has a value worth launching. In this study, the performance of web-app integration using Jenkins has an interval of difference that is not too far from conventional integration with an average response time of 47.7 ms for integration using Jenkins and an average response time of 30.9 ms for conventional integration. However, the performance of the engine test using a Docker container gives healthier results than using a personal device with an average memory usage value of 56.7% and cpu usage of 68.6%.

Bandwidth in Jenkins integration provides a constant value this is because the bandwidth value depends on the bandwidth management system settings in Jenkins. The value of throughput, response time, and latency obtained is influenced by the number of concurrent users. The higher the concurrent user value, the higher the throughput value and the smaller the response time and latency.

Acknowledgment

With the completion of this research, the author would like to thank:

1. Mr. Dr. Ronald Adrian, S.T., M.Eng., as the supervisor of the final project and the head of the Internet Engineering Technology study program who has been pleased to guide and support the author in the work of the final project.
2. Various parties who cannot be mentioned one by

one for all the help and support in completing this final report.

REFERENCES

[1] S. Mysari and V. Bejgam, "Continuous Integration and Continuous Deployment Pipeline Automation Using Jenkins Ansible," Feb. 2020. doi: 10.1109/ic-ETITE47903.2020.239.

[2] J. Benjamin and J. Mathew, "Enhancing the efficiency of continuous integration environment in DevOps," *IOP Conference Series: Materials Science and Engineering*, vol. 1085, no. 1, p. 012025, Feb. 2021, doi: 10.1088/1757-899x/1085/1/012025.

[3] J. Mahboob and J. Coffman, "A Kubernetes CI/CD Pipeline with Asylo as a Trusted Execution Environment Abstraction Framework," *2021 IEEE 11th Annual Computing and Communication Workshop and Conference, CCWC 2021*, pp. 529–535, Jan. 2021, doi: 10.1109/CCWC51732.2021.9376148.

[4] G. Lettieri, V. Maffione, and L. Rizzo, "A Study of I/O Performance of Virtual Machines," *The Computer Journal*, vol. 61, no. 6, pp. 808–831, Jun. 2018, doi: 10.1093/COMJNL/BXX092.

[5] Y. Luo, "Network I/O Virtualization for Cloud Computing," *IT Professional*, vol. 12, no. 05, pp. 36–41, Sep. 2010, doi: 10.1109/MITP.2010.99.

[6] B. Igli Tafa *et al.*, "The Evaluation of Network Performance and CPU Utilization during Transfer between Virtual Machines The Evaluation of Network Performance and CPU Utilization during Transfer between Virtual Machines The Evaluation of Network Performance and CPU Utilization during Transfer between Virtual Machines," *Type: Double Blind Peer Reviewed International Research Journal Publisher: Global Journals Inc*, vol. 11, 2011.

[7] S. Jia, G. Juell-Skielse, and E. Uppström, "Integrating Conventional ERP System with Cloud Services From the Perspective of Cloud Service Type INTEGRATING CONVENTIONAL ERP SYSTEM WITH CLOUD SERVICES: FROM THE PERSPECTIVE OF CLOUD SERVICE TYPE".

[8] J. Shah, D. Dubaria, and J. Widhalm, "A Survey of DevOps tools for Networking," in *2018 9th IEEE Annual Ubiquitous Computing, Electronics and Mobile Communication Conference, UEMCON 2018*, Nov. 2018, pp. 185–188. doi: 10.1109/UEMCON.2018.8796814.