Docker and Kubernetes Pipeline for DevOps Software Defect Prediction with MLOps Approach

Rizal Broer Bahaweres[1], Aldi Zulfikar[2], Irman Hermadi[3], Arif Imam Suroso[4], Yandra Arkeman[5]
[1,2]Department of Informatics, UIN Jakarta, Indonesia
[1,3]Computer Science Dept, IPB University, Bogor, Indonesia
[4]School of Business, IPB University, Bogor Indonesia
[5]Department of Agro-Industrial Tech., IPB University, Bogor, Indonesia
rizalbroer@computer.org, aldi.zulfikar19@mhs.uinjkt.ac.id, irmanhermadi@apps.ipb.ac.id, arifimamsuroso@apps.ipb.ac.id, yandra@apps.ipb.ac.id

2022 2nd International Seminar on Machine Learning, Optimization, and Data Science (ISMODE) | 978-1-6654-5564-0/22/$31.00 ©2022 IEEE | DOI: 10.1109/ISMODE56940.2022.10180973

***Abstract*—Software defects are common when it comes to software development. However, in reality, this is very detrimental for companies and organizations that are developing software. Prediction of software defects in the early stages of development can be a solution to this problem. Of course, the method used needs to be considered when developing a model for predicting software defects. The software continues to experience development, so the prediction model must always be updated so that it can adapt to existing conditions. This study proposes the MLOps approach, which combines development and operation processes to develop a software defect prediction model. We will create a prediction model and then create a Docker and Kubernetes pipeline to automate the entire software defect prediction process so that it can speed up the development process and have good performance. We are comparing the performance evaluation results of the proposed method with the traditional method, which is run manually by Docker. The results showed that the entire source dataset had a fairly good accuracy rate of 76%–83% and a good recall rate of 79%–94%. The precision and recall values were also very good. Apart from that, it also produces a good F1-score value of 84%–90%. And the development time until the model's release is shorter: the average time is 7:02 minutes. Performance monitoring on the built-in web server is also easy to do and shows very good results. The web server can receive up to 156.6/sec requests in all models based on the dataset used, with the highest error rate at 45.03%. The use of the Docker and Kubernetes pipelines with the MLOps approach has been proven to have good performance, and the development of software defect models can be sped up.**

Keywords—MLOps, DevOps, Software Defect Prediction, CI/CD Pipelines. Docker, Kubernetes

I. INTRODUCTION

The current digital era makes technology such as computers indispensable to support daily activities. A computer requires a specially designed program with a specific code and operating system to carry out its functions, which are packaged in software. Software is currently overgrowing in terms of size and complexity, which means developers must be able to create software with minimum defects. Research conducted by [1] explained that the cost of finding and repairing defects is one of the most expensive software development activities because the cost of defects will continue to increase as long as software development takes place. This makes software testing play an important role in the development process, especially in predicting defects that exist in the software. The right method of predicting software defects can help developers find faulty code modules automatically and save resources.

Software defect prediction aims to automatically predict defects and errors in software because it is expected to be able to identify modules that have defects in the early stages of project development, ensure software safety and quality, reduce manual testing costs, and shorten software development life cycles. The principle of software defect prediction is basically based on studying a model from a historical data set (training set) and applying it to the test data (test set) to carry out tests on the same project [2], [3].

Machine learning techniques can be applied to develop software defect prediction models such as Support Vector Machine (SVM), Naive Bayes (NB), Decision Tree (DT), Neural Network (NN), and others [4]. However, because software is currently developing very rapidly, making models for predicting software defects does not only stop at the deployment phase. Therefore, a continuously updated model is needed to better predict software defects.

The initial stage of creating a prediction model requires a lot of time and effort. In addition, the deployment and maintenance activities of the models that have been created can be a long process, and most models in the production phase experience a decrease in performance after some time. Thus, we need an appropriate method in the process of developing and deploying sustainable software defect prediction models.

DevOps is one of the methods used for software development that is very effective and efficient. Research conducted by [5] proves that the DevOps method can overcome problems when developing software, such as shortening development time, minimizing error rates, reducing defects during development and deployment, and minimizing resources used during software development.

MLOps refers to the DevOps concept applied in predictive modeling and aims to combine a series of practices based on tools, pipelines, and modeling processes so that it doesn't take up much time and costs [6], [7]. Each phase in the MLOps lifecycle runs in a CI/CD pipeline. Implementing a CI/CD pipeline can speed up the release process because every process in it runs automatically, so it can do 10–1000 software updates every day [8].

Docker and Kubernetes can be used in a CI/CD pipeline to automatically build applications into container images and manage these containers on a server cluster [9]. Based on the explanation above, the author will examine the use of Docker and Kubernetes pipelines for the release of software defect prediction models automatically through the MLOps approach.

The formulation of the problem that can be formulated in this study is as follows:

1. How about the classification performance results on the dataset used for the MLOps approach to making prediction models?
2. How can the MLOps approach to building software defect prediction models speed up the time to release?
3. How does the performance of the web server using the Docker and Kubernetes environments compare to that of the Docker and virtual machine environments?

The limitations of the problem in this study are as follows:

1. Docker and Kubernetes are used to deploy software defect prediction applications.
2. MLOps is used as a system development method.
3. GitHub Action is used to run the processing pipeline.

II. Literature Review

A. Related Research

TABLE I. LITERATURE STUDY

Ref.	Docker	Kubernetes	Application		Method
			SDP	Other	
[7]	-	-	-	✔	MLOps
[10]	-	-	-	✔	MLOps
[11]	-	-	-	✔	DevOps
[12]	-	-	-	✔	DevOps
[13]	✔	✔	-	✔	DevOps
[14]	✔	✔	-	✔	DevOps
[3]	-	-	✔	-	Cross-project
[15]	-	-	✔	-	Transfer Learning
[4]	-	-	✔	-	Deep Transfer Learning and Software Visualization
Author	✔	✔	✔	-	Docker and Kubernetes Pipeline with MlOps Approach

Table 1 describes several studies related to the research that the author did. [7], [10], machine learning applications using the MLOps method. [11], [12], developing web services using the DevOps method. [13], [14], using Docker and Kubernetes in the web service development process which is carried out using the DevOps method. [3], using cross-project defect prediction to make model prediction [15], create a software defect prediction model using the transfer learning method. [4], propose deep transfer learning and software visualization methods for the prediction of software defects.

From the research above, the DevOps and MLOps approaches were not explored in the problem of predicting software defects. We implemented Docker and Kubernetes with the MLOps approach for software flaw prediction.

B. DevOps

DevOps is an approach to carrying out a series of jobs by automating the process between development (Dev) and operation (Ops), which aims to create collaboration and interaction between divisions in developing software [12]. The DevOps life cycle consists of six phases that represent the processes, capabilities, and tools needed when carrying out development and operations. In each of these phases, the team will collaborate and communicate to create a good culture so that it can present a comprehensive application for users. Implementing DevOps can make the software release process faster with a high success rate.

C. MLOps

MLOps (Machine Learning Operations) are adaptations of DevOps practices such as continuous integration and continuous delivery that are implemented to manage the life cycle of machine learning [16]. The main goal of MLOps is to make the development and deployment model process run automatically so that it becomes faster with better quality, is easier to update, and has end-to-end tracking [6].

D. CI/CD Pipeline

A continuous integration/continuous delivery pipeline is a bridge between development and operations that automates the processes of building, testing, and deploying software collaboratively and in real-time. The goal of automation is to minimize human error and maintain consistency in how software is released [8]. The CI/CD process consists of three parts: continuous integration, continuous delivery, and continuous deployment.

1) Continuous Integrations: In this phase, integration between software development and operational processes will be carried out automatically. The details of the process in this phase are that every time the developer commits to the program code, the tools used will automatically execute a series of processes in the pipelines that have been made, such as code review, unit testing, and packaging [8].

2) Continuous Delivery: Continuous Delivery is an advanced process of continuous integration, where all changes to the source code, such as feature additions, bug fixes, and configuration changes, are automatically prepared before being deployed to the production environment or into the hands of the user [8].

3) Continuous Deployment: In the Continuous Deployment phase, program code that has gone through a series of processes in the previous phase will be deployed to the production server. Unlike continuous integration and continuous delivery, in continuous deployment, the process is done manually [8].

E. Docker and Kubernetes

Docker is an open-source container manager that can be used to develop, ship, and provide a fast and lightweight environment in which code can run efficiently. When an application is virtualized and executed in a container, Docker adds a layer of deployment machinery on top of it [9]. Docker also provides services to automate the deployment of applications into container images.

Kubernetes is an open-source tool used to orchestrate containers. in terms of managing deployment, scaling and descaling, containerized applications, and load-balancing containers [17]. Kubectl and web apps can be used to run

commands on Kubernetes. Kubernetes can also be used to create ingress routes, run stateful services, and manage secrets and passwords.

F. Software Defect Prediction

Software defects are a phase where the software that is built experiences conditions that are not in line with the requests or needs of the user. In research conducted by [18], it is explained that the cost of finding and repairing defects is one of the most expensive software development activities because the cost of defects will continue to increase as long as software development takes place. Software defects can be prevented by predicting them. Software defect prediction is an activity that aims to automatically predict defects and errors that exist in software, and identify modules that have defects in the early stages of project development to ensure software safety and quality, reduce manual testing costs, and shorten the development cycle of software.

G. Transfer Learning

Transfer learning is a technique that can help when the training data for machine learning is insufficient. The working principle is to transfer information from the source domain to the target domain [4]. In other words, in transfer learning, other training model data is used to be applied to the training model that will be created.

H. Residual neural network (ResNet)

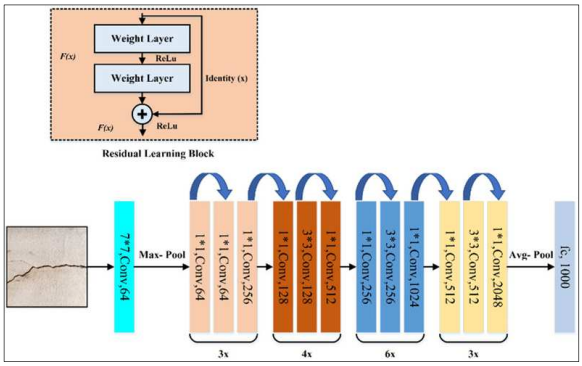

Fig. 1. Architecture ResNet50 [19]

Residual neural network (ResNet) is one of the artificial deep neural network (DNN) architectures. ResNet has several variants, such as ResNet-18, ResNet-50, ResNet-152, and ResNet, depending on the number of layers used in it. The use of ResNet aims to avoid missing the gradient problem. Decreasing the loss function to find the appropriate weight is the cause of the problem that occurs [20]. Repeated multiplication that occurs in deep learning causes the gradient value to get smaller and can even disappear as the number of layers increases. In addition, training a neural network model with multiple layers requires a significant learning rate. The architecture on ResNet is considered to be able to solve this problem because it uses an identity shortcut connection that is passed from several layers and uses the activation function of the previous layer as shown in figure 1.

I. Confusion Matrix

In determining whether a learning model is running well, measurement is needed. The confusion matrix is a measure that can be used when trying to calculate the performance of a classification model. Specifically, the confusion matrix contains several terms [3]. Such as:

- The prediction that a module is defective and has a true defect is called True Positive (TP).
- The prediction that a module is non-defective, but turns out to be defective is called False Positive (FP).
- The prediction that the module is non-defective and truly non-defective is called True Negative (TN).
- The prediction module is defective but non-defective, so it is called False negative (FN).

So, performance classifications such as Accuracy (A), precision (P), recall (R), and F1-scores can be defined in table II.

TABLE II. PERFORMANCE MEASURES

Performance Measure	Description	Formula
Accuracy	To percentage the number of correct predictions	$A = \frac{TN+TP}{(TN+FN+TP+FP)}$ (1)
Precision	To calculate the ratio of correct positive data predictions to all positive data prediction results	$P = \frac{TP}{(TP+FP)}$ (2)
Recall	To represent the amount of positive data that is predicted to be positive	$R = \frac{TP}{(TP+FN)}$ (3)
F1-Scores	To calculate the weighted comparison of the average precision and recall	$F1 - scores = \frac{(2\ x\ P\ x\ R)}{(P+R)}$ (4)

III. RESEARCH METHODOLOGY

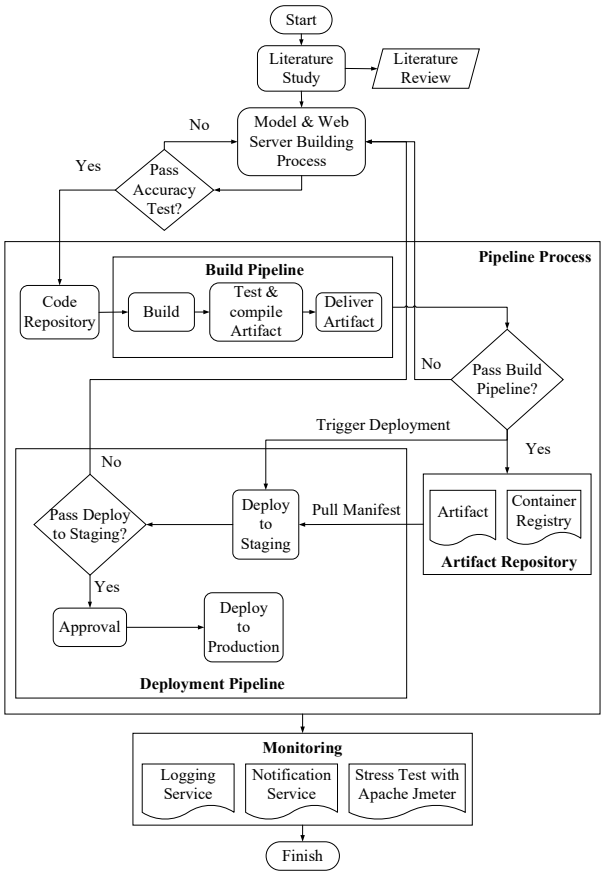

Fig. 2. Research Methodology

The steps that will be used in this study are shown in Figure 2 as follows: (1) literature review; (2) model and web server building process, (3) testing the accuracy of the model that has been made; (4) pushing source code and the model into the code repository; (5) build pipeline; (6) artifacts repository; (7) deployment pipeline; (8) monitoring; (9) conclusion.

A. Model and Web Server Building Process

The model-building process aims to create a model that will be used in predicting defects in software. The architecture used in this modeling activity is residual network 50 (ResNet50). The choice of this architecture is based on several previous studies that have proven that ResNet50 has good accuracy and effectiveness.

The image datasets used are ant-1.6, poi-3.0, and log4j-1.0, which will be used as source datasets, and log4j-1.1, which will be used as target datasets. The source dataset will go through a data augmentation process and is ready to be used as input in the prediction model-building process. This process consists of the first two activities, namely adaptation, which aims to modify the ResNet50 model classifier and adapt it to the dataset that the author has. The second is fine-tuning, which aims to retrain a model that has been adjusted during the adaptation.

TABLE III. DETAILS OF DATASETS

Class	Source Dataset			Target Dataset
	ant-1.6	***poi-3.0***	***log4j-1.0***	***log4J-1.1***
Buggy	91	276	34	37
Clean	258	157	83	65
Total	**349**	**433**	**117**	**102**

Making a web server is done to operate the model that has been trained so that it can be used by users to make predictions. The Python programming language and the Flask framework will be used to create this web server.

B. Accuracy Test

This process aims to evaluate the performance of the trained model. This evaluation process is carried out by mapping the prediction results from the model into a confusion matrix table, whose results are then used to calculate classification performance such as accuracy, precision, recall, and the F-1 score.

C. Code Repository

Source code management tools such as Git and GitHub will be used in this activity as a repository for storing source code to manage the creation process of a web server, for example, storing, managing, and developing it.

D. Build Pipeline

The build pipeline process aims to create instructions for the upcoming web server. This is done by taking the commit ID from the repository and using the source code to run it. The series of stages in this process are building, testing, and compiling software artifacts; delivering artifacts; and optionally triggering the deployment process. The build spec file is used to define the flow and instructions of the build process.

E. Artifact Repository

The purpose of this process is to store and group the built artifacts into an artifact repository. These artifacts are a collection of text files, binaries, and deployment manifests that will be delivered to the target deployment environment. The naming of artifacts in the repository is based on the *path: version.*

F. Deployment Pipeline

The deployment pipeline process is used to deliver and deploy a set of artifact to the target environment. The stages of a software release in this process can be arranged serially or in parallel.

G. Monitoring

The monitoring process aims to observe all stages of the pipeline. All information about all stages will be stored in a logging service in the form of log entries. It can contain the deployment runtime and its final output. When errors or bugs occur, constraints can also be observed through log entries. The notification service is also used in monitoring, such as to send information on the latest status of the project being executed and prompts to take the necessary action. Aside from that, Apache Jmeter is used to run stress tests on web servers deployed in the Docker and Kubernetes environments.

IV. RESULT AND DISCUSSION

A. Experiment Result

The experiment we conducted consisted of 3 main stages, namely the process of building a model and a web server for predicting software defects, the pipeline process, and monitoring. Where all stages run automatically, we also compare the performance of all the main stages when doing it manually.

1) Model and Web Server Building Process for Software Defect Prediction

At this stage, we create predictive models and web servers with the best accuracy. Table IV are the results of a comparison of the classification performance of the datasets used in the prediction models that have been made. log4j-1.1 is used as the target dataset in this study.

TABLE IV. CLASSIFICATION PERFORMANCE REPORT

Source Datasets	Accuracy	Precision	Recall	F1-Score
ant-1.6	0.83	0.86	0.94	0.90
poi-3.0	0.76	0.91	0.79	0.84
log4j-1.0	0.83	0.93	0.85	0.89

2) Pipelines Process

The pipeline process that we create consists of two cases namely release time and recovery time. Tables V and VI show the duration of time required for each phase to perform its duties based on the dataset used.

TABLE V. DURATION TIMES REPORT WITH PIPELINE PROCESS

	Source Datasets (Times in Minutes)			AVG
	Ant-1.6	Poi-3.0	Log4j-1.0	
Release Time	07:14	06:24	07:28	07:02

	Source Datasets (Times in Minutes)			AVG
	Ant-1.6	Poi-3.0	Log4j-1.0	
Recovery Time	01:54	04:03	01:35	08:21

TABLE VI. DURATION TIMES REPORT WITH MANUAL PROCESS

	Source Datasets (Times in Minutes)			AVG
	Ant-1.6	Poi-3.0	Log4j-1.0	
Release Time	36:05	31:30	32:04	33:13
Recovery Time	14:04	12:37	13:08	13:16

3) Monitoring

At this monitoring stage, the performance of each web server in the entire dataset will be monitored. Table VII is the result of web service performance testing that has been successfully deployed in Docker and Kubernetes environments, and table VIII is the result of web service performance testing that has been successfully deployed in the Docker and instance compute (Virtual Machine) environments. Both were performed on Oracle Cloud Infrastructure and measured using Apache Jmeter with 10 repetitions for 10 minutes based on the dataset used.

TABLE VII. WEB PERFORMANCE WITH DOCKER AND KUBERNETES ENVIRONMENT

Source Dataset	Threads	Sample	Avg	Std. Deviation	Error	Throughput
ant-1.6	5,000	48,137	13,932	55,332.04	25,29%	80,2/sec
	10,000	93,984	17,051	67,762.18	45,03%	156,6/sec
poi-3.0	5,000	47,671	14,785	59,183.72	19,27%	79,5/sec
	10,000	90,283	19,876	78,277.41	42,79%	150,2/sec
log4j-1.0	5,000	46,993	15,582	62,583.78	18,44%	78,2/sec
	10,000	88,755	22,840	85,326.67	42,91%	147,6/sec

TABLE VIII. WEB PERFORMANCE WITHOUT DOCKER AND KUBERNETES ENVIRONMENT

Source Dataset	Threads	Sample	Avg	Std. Deviation	Error	Throughput
ant-1.6	5,000	31,575	64,677	162,782.60	26,73%	51,9/sec
	10,000	64,066	63,128	155,196.17	38,47%	106,3/sec
poi-3.0	5,000	41,006	30,312	104,590.60	15,03%	68,2/sec
	10,000	73,156	51,304	133,141.84	30,27%	121,2/sec
log4j-1.0	5,000	38,125	37,189	120,357.95	15,70%	63,5/sec
	10,000	95,338	29,836	60,689.61	58,89%	158,6/sec

B. Result

From the experimental results of automating the process of releasing software defect prediction models using the MLOps approach. We started the process of automating the creation of this model and web server with the first stage, namely the process of building the model and web server for software defect prediction. The results of the first stage in table IV show that the prediction model created with the ant-1.6 source dataset has an accuracy value of 83% with a precision value of 86%, a recall value of 94%, and an F1-score value of 90%. The poi-3.0 source datasets have an accuracy value of 76%, a precision value of 91%, a recall value of 79%, and an F1-score value of 84%. And for the log4j-1.0 source datasets, we have an accuracy value of 83% with a precision value of 93%, a recall value of 85%, and an F1-score value of 89%.

Then proceed to the second stage, namely the pipeline process. The results of this stage in tables V and VI show testing based on the dataset model used. From each model based on the dataset, the duration for the release time using the pipeline process produces a faster average time of 7 minutes 2 seconds when compared to using the manual process, which is 33 minutes 13 seconds. Meanwhile, the recovery time required when using the pipeline process is 8 minutes 21 seconds and when using the manual process, it is 13 minutes 16 seconds.

At the monitoring stage, a performance test is carried out on the released web server. Tables VII and VIII show the results of the performance tests performed. It can be seen in the two tests that there is a significant difference. For samples generated from 5,000 and 10,000 threads with 10 iterations over 10 minutes, when using Docker and Kubernetes environments on all model-based datasets. The number of samples reached above 93% with the lowest value of 46,993 and the highest error rate of 25.29% at 5,000 threads, and above 88% with the lowest value of 88,755 and the highest error rate of 45.03% at 10,000 threads. As for the samples generated without using the Docker and Kubernetes environments, they only reached above 63% with the lowest value of 31,575 and the highest error rate of 26.73% on 5,000 threads, and above 64% with the lowest value of 64,066 and the highest error rate of 58.89% on 10,000 threads. Then, for the standard deviation values generated in both tests, it can be seen that the data distribution varies. This can be proven by the resulting standard deviation value, which is greater than the average. However, the resulting throughput does not have a significant difference, that is, when using the Docker and Kubernetes environments, the maximum is 80.2/sec when using 5,000 threads and 156.6/sec when using 10,000 threads. Meanwhile, when not using the Docker and Kubernetes environments, the biggest is 68.2/sec when using 5,000 threads and 158.6/sec when using 10,000 threads.

It can be seen that the model release automation process becomes faster by using the approach from MLOps. The development and deployment processes can run faster than manual ones, and it's easy to monitor for errors, which makes the release process more effective. The use of Docker and Kubernetes is also very effective and has good performance to be used as a container and orchestrator to deploy a web server based on the prediction model that has been made.

V. CONCLUSION AND FUTURE WORK

In our experiment, we apply the MLOps approach to release software defect prediction models. Our model used the ResNet50 architecture. Of all the source datasets used (log4j-1.0, ant-1.6, and poi-3.0), and use the target dataset log4j-1.1. It produces an accuracy rate of 76%-83% and has a fairly good precision and recall value of 86%-93% for precision values and 79%-94% for recall values. Besides that, it also produces a good F1-Score value of 84%–90%. From the results of research using the MLOps approach, it was found that the prediction model release time was faster, with an average of

07:02 minutes and an average recovery time of 08:21 minutes for all source datasets. Thus, the model that has been built will be more easily developed and updated to suit future conditions. Overall, monitoring the performance testing of web servers built in the Docker and Kubernetes environments shows very good results for each model of each dataset used compared to not using the Docker and Kubernetes environments.

Future studies are expected to pay attention to class imbalances and misclassifications in making predictive models so that they can obtain better accuracy values. Then we can also apply the infrastructure as code so that the infrastructure creation process can also run automatically, further speeding up the release time for the prediction model.

ACKNOWLEDGMENTS

We would like to thank Oracle Academy Indonesia, the Asia Pacific Region, the Digital Talent Scholarship Program, and the Ministry of Communication and Information Republic Indonesia for supporting this work.

REFERENCES

[1] S. Ramadhina, R. B. Bahawares, I. Hermadi, A. I. Suroso, A. Rodoni, and Y. Arkeman, "Software Defect Prediction Using Process Metrics Systematic Literature Review: Dataset and Granularity Level," in *2021 9th International Conference on Cyber and IT Service Management, CITSM 2021*, 2021. doi: 10.1109/CITSM52892.2021.9587932.

[2] Q. Yu, S. Jiang, and Y. Zhang, "A feature matching and transfer approach for cross-company defect prediction," *Journal of Systems and Software*, vol. 132, pp. 366–378, Oct. 2017, doi: 10.1016/J.JSS.2017.06.070.

[3] P. A. Habibi, V. Amrizal, and R. B. Bahaweres, "Cross-Project Defect Prediction for Web Application Using Naive Bayes (Case Study: Petstore Web Application)," in *2018 International Workshop on Big Data and Information Security, IWBIS 2018*, Sep. 2018, pp. 13–18. doi: 10.1109/IWBIS.2018.8471701.

[4] J. Chen *et al.*, "Software visualization and deep transfer learning for effective software defect prediction," in *Proceedings - International Conference on Software Engineering*, Jun. 2020, pp. 578–589. doi: 10.1145/3377811.3380389.

[5] T. Tohirin, S. F. Utami, S. R. Widianto, and W. al Mauludyansah, "Implementasi DevOps Pada Pengembangan Aplikasi e-Skrining Covid-19," *MULTINETICS*, vol. 6, no. 1, pp. 15–20, May 2020, doi: 10.32722/multinetics.v6i1.2764.

[6] R. Subramanya, S. Sierla, and V. Vyatkin, "From DevOps to MLOps: Overview and Application to Electricity Market Forecasting," *Applied Sciences (Switzerland)*, vol. 12, no. 19, Oct. 2022, doi: 10.3390/app12199851.

[7] Y. Liu, Z. Ling, B. Huo, B. Wang, T. Chen, and E. Mouine, "Building A Platform for Machine Learning Operations from Open Source Frameworks," in *IFAC-PapersOnLine*, 2020, vol. 53, no. 5, pp. 704–709. doi: 10.1016/j.ifacol.2021.04.161.

[8] S. A. I. B. S. Arachchi and I. Perera, "Continuous Integration and Continuous Delivery Pipeline Automation for Agile Software Project Management," pp. 156–161, 2018, doi: 10.1109/MERCon.2018.8421965.

[9] B. B. Rad, H. J. Bhatti, and M. Ahmadi, "An Introduction to Docker and Analysis of its Performance," 2017. [Online]. Available: https://www.researchgate.net/publication/318816158

[10] F. Lanubile, L. Quaranta, and F. Calefato, "A Preliminary Investigation of MLOps Practices in GitHub," *Association for Computing Machinery*, pp. 283–288, 2022, doi: 10.48550/arXiv.2209.11453.

[11] M. S. Arefeen and M. Schiller, "Continuous Integration Using Gitlab," *Undergraduate Research in Natural and Clinical Science and Technology (URNCST) Journal*, vol. 3, no. 8, pp. 1–6, Sep. 2019, doi: 10.26685/urncst.152.

[12] V. Arulkumar and R. Lathamanju, "Start to Finish Automation Achieve on Cloud with Build Channel: By DevOps Method," in *Procedia Computer Science*, 2019, vol. 165, pp. 399–405. doi: 10.1016/j.procs.2020.01.032.

[13] I. C. Donca, O. P. Stan, M. Misaros, D. Gota, and L. Miclea, "Method for Continuous Integration and Deployment Using a Pipeline Generator for Agile Software Projects," *Sensors*, vol. 22, no. 12, Jun. 2022, doi: 10.3390/s22124637.

[14] M. K. Abhishek, D. R. Rao, and K. Subrahmanyam, "Framework to Deploy Containers using Kubernetes and CI/CD Pipeline," 2022. doi: 10.14569/IJACSA.2022.0130460.

[15] S. Tang, S. Huang, C. Zheng, E. Liu, C. Zong, and Y. Ding, "A Novel Cross-Project Software Defect Prediction Algorithm Based on Transfer Learning," 2022. doi: 10.26599/TST.2020.9010040.

[16] G. Symeonidis, E. Nerantzis, A. Kazakis, and G. A. Papakostas, "MLOps -- Definitions, Tools and Challenges," Jan. 2022, doi: 10.48550/arXiv.2201.00162.

[17] S. Chakrabarti *et al.*, *Building Modern Clouds: Using Docker, Kubernetes & Google Cloud Platform*. 2019. doi: 10.1109/CCWC.2019.8666479.

[18] R. B. Bahaweres, K. Zawawi, D. Khairani, and N. Hakiem, "Analysis of statement branch and loop coverage in software testing with genetic algorithm," in *International Conference on Electrical Engineering, Computer Science and Informatics (EECSI)*, Dec. 2017, vol. 2017-December. doi: 10.1109/EECSI.2017.8239088.

[19] L. Ali, F. Alnajjar, H. al Jassmi, M. Gochoo, W. Khan, and M. A. Serhani, "Performance evaluation of deep CNN-based crack detection and localization techniques for concrete structures," *Sensors*, vol. 21, no. 5, pp. 1–22, Mar. 2021, doi: 10.3390/s21051688.

[20] B. Falahkhi, E. F. Achmal, M. Rizaldi, R. Rizki, and N. Yudistira, "Comparison of AlexNet and ResNet Models in Flower Image Classification Utilizing Transfer Learning," *Jurnal Ilmu Komputer dan Agri-Informatika*, vol. 9, pp. 70–78, 2018, doi: 10.29244/jika.9.1.70-78.