p–ISSN: 2723 - 6609 e-ISSN: 2745-5254
Vol. 4, No. 9 September 2023 http://jist.publikasiindonesia.id/

CI/CD IMPLEMENTATION APPLICATION DEPLOYMENT PROCESS ACADEMIC INFORMATION SYSTEM (CASE STUDY OF PARAMADINA UNIVERSITY)

Rendy Indriyanto[1*], Diki Gita Purnama[2*]
Paramadina University Jakarta, Indonesia
Email: rendy.indriyanto@students.paramadina.ac.id[1*],
diki.purnama@paramadina.ac.id[2*]

*Correspondence

ARTICLE INFO

Accepted : 12-09-2023
Revised : 17-09-2023
Approved : 25-09-2023

Keywords: information system; DevOps; ci/cd; software deployment; academic application.

ABSTRACT

This research discusses the implementation of Continuous Integration/Continuous Delivery (CI/CD) in the deployment process of the Academic Information System application at Paramadina University, where the application development process still uses manual methods. CI/CD is an approach in software development that enables development teams to automatically integrate code, run tests, and deploy applications periodically and continuously. The purpose of this research is to implement CI/CD in the deployment process of Academic Information System applications so that the development process becomes more effective and the quality of the resulting application is better and more adaptable to changes. The method used in this research is the qualitative method by conducting direct observation to collect data. The results of the research show that by implementing the stages of CI/CD, the application deployment process becomes more efficient and can provide a solution to the problems that arise in the previously conventional or manual application deployment process. The results of this research can serve as a guide for development teams and operations who want to adopt CI/CD in the application deployment process, as well as provide insights for researchers and practitioners to optimize the use of CI/CD in software development.

Introduction

The development of technology in all aspects encourages universities to take strategic steps to remain superior in all fields (Aswati, Mulyani, Siagian, & Syah, 2015). To achieve this goal, one of the steps taken by higher education institutions is to build a qualified academic information system. The use of academic information systems can increase the efficiency and effectiveness of academic administration management, improve the quality of academic administrative services to stakeholders, especially students, and increase the competitiveness of universities (Yindrizal, 2021).

In conditions where the demand for a system to have high adaptation to changes that include changes in business processes, changes in infrastructure needs as well as the addition of features and other additions, it is important to consider how to determine the application deployment method that must be implemented (Sutarman, Fadli, & Aliim, 2023).

Past software development such as Waterfall methods, Prototyping, Agile, and RUP, was only discussed when software development was only, there was no topic to

discuss how to deploy and install software in the target infrastructure. Even though current conditions demand that application development must be complete up to the deployment and maintenance of software releases (Taryana, Fadli, & Nurshiami, 2020). The process of making software ready for use by end users, as well as by application testers and other related parties is referred to as deployment or delivery (Ghimire, 2020).

In the previous development of the Academic Information System, the IT team of Paramadina University was still carrying out the process of deploying applications conventionally and had not implemented an automation system. The entire process is done manually where the source code that has been completed is then uploaded to the production server (Arachchi & Perera, 2018).

The practice of deploying in this way is certainly simpler and easier to do, but problems will arise when updates or changes are made on the side of the running application. When the development team makes direct code changes to the application that is running on the production server, there will potentially be errors that cause the application to malfunction. Business process changes to the application will be difficult for the development team to do. Development in traditional ways often leads to problems such as application delays and product quality (Laksito, 2022).

Another problem that arises is the difference between the device and environment used on the development team's side with the server side so that there is a mismatch both from the operating system version, supporting software, libraries, and software package dependencies which results in the application cannot function correctly when it has been deployed to the server (Hadian, Hakim, & Fanani, 2023).

From the problems above, other deployment techniques are needed that can carry out the process automatically starting from the development, testing, and deployment stages. For this reason, the author conducted research on the implementation of Continuous Integration/Continuous Delivery (CI/CD) in the process of developing Academic Information System applications which are being updated at Paramadina University.

The implementation of CI/CD is part of DevOps principles and practices. DevOps is a term for a process that focuses on improving collaboration, communication, and integration between software developers and IT operations (Chapman, 2014). DevOps reduces the gap between the development team, operations team, and application users allowing them to detect problems early (Tohirin, Utami, Widianto, & Al Mauludyansah, 2020).

By implementing CI/CD, it is useful to monitor a project in terms of the speed of code generation to deployment to a production environment (Achdian & Marwan, 2019). The application deployment process will be carried out automatically to the staging and production server bypassing the testing stage by the system. Coding errors will be avoided because when the development team makes a mistake, the deployment process will automatically stop and errors that occur will immediately be identified. Continuous Delivery (CD) provides many benefits for development teams including process

automation, increased developer productivity, improved coding quality, and faster distribution to customers.

Research Methods

This research was conducted using a qualitative approach, namely research that intends to understand the phenomenon of what is experienced by research subjects such as behavior, perception, motivation, action, etc., holistically, and using description in the form of words and language, in a special context that is natural and utilizes various natural methods (Hamid & Prasetyowati, 2021).

The type of research carried out is by observation, which is making direct observations to collect data about behavior or events in a particular location. The research was conducted at Paramadina University and began in the early days of the construction of a new academic information system application, namely in September 2021.

In this study, the author is directly involved and participates in the CI/CD implementation process, while the process carried out is as follows:

Data Collection.

Data collection was carried out by making observations at the research location, namely at Paramadina University, and also conducting literature studies from sources that provide information related to the research theme.

Tahapan Implementasi.

After the data is successfully collected, proceed with the implementation stage of the CI / CD system by adjusting the use of tools and technology referring to the data obtained. In general, the CI/CD workflow process has the following basic stages.

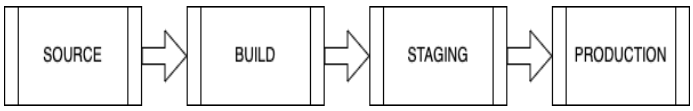


Figure 1. CI/CD Workflow

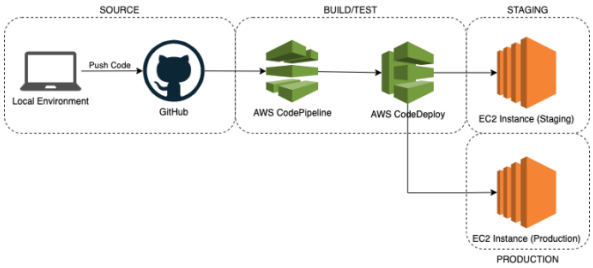
1) Source: At this stage, the development team begins to integrate the source code by committing and pushing in the local environment to a central repository that has been connected to the CI/CD pipeline.
2) Build: Pushing to the central repository automatically triggers the next build processed by the pipeline used. Another process that occurs at this stage is the testing stage of the code to be deployed.
3) Staging: At this stage, the application deployment process begins. The result of the process at this stage is that the application can already appear and can be tested on the functions of the application.
4) Production: This is the final stage of the CI/CD process. At this stage, the application can already be used by its users.

Tools

Because this research location uses AWS cloud computing services, most of the tools used in this study utilize many features and services from AWS. The following are the tools and technologies used:

a. Laravel Framework
b. VIM text editor
c. SSH remote client
d. Git Version Control System
e. GitHub (https://github.com)
f. AWS CodePipeline
g. AWS CodeDeploy
h. Amazon EC2 Instance

The relationship between the tools used can be illustrated in the diagram below.



Gambar 2. Tools dan komponen CI/CD yang digunakan

Results and Discussion

A. Source

At the initial stage, the author does setup and configuration starting from creating a repository, adjusting scripts for the auto-deploy application process, creating EC2 instances along with installing the required supporting software.

1) Application repository: The implementation process is carried out by installing a version control system (Git client) on the developer's side, namely on each work equipment both laptop and PC used to code applications. After that, a repository is created using Git Hub so that all team members involved can collaborate. At this stage, the process of integrating the source code of the application can be done.
2) Script for automating the process of deploying source code: The main script in CI/CD implementations using AWS CodePipeline and Github is the 'appspec.yml' file. This script specifies the destination path of the source code files to be deployed, and will also execute other additional scripts needed in application deployments that use the Laravel framework. Other additional scripts that will be included in a special directory and executed during the deployment process include the following:

1. install_dependencies.sh

When this script is run, the system will adjust the web server configuration and install the PHP composer needed by the Laravel framework.

2. deploy_laravel.sh

This script when executed will create several directories including bootstrap/cache, storage/framework/sessions, storage/framework/view, storage/framework/cache, and public/cache. Then this script will execute the 'composer install' command in the root directory of the application so that all the required packages and dependencies are installed. Other commands performed in this script are database initiation, cache optimization, changing file permissions, and activating applications.

3. change_permission.sh

As in most web-based applications, files and directories need to be changed ownership and permissions, so this script will do it.

4. start_server.sh

This script will activate the web server service, in this case, Apache.

5. stop_server.sh

This script will be executed to disable the web server when needed.

All of the above scripts are then pushed to a central repository and merged into the root directory of the application project.

3) EC2 Instance Installation: For the host server in this study, the authors used an AWS-provided virtualization service, namely Elastic Compute Cloud (EC2), and used the Amazon Linux 2 operating system. Here are the manufacturing steps:

a. Log in with your AWS account to the management console and select EC2 services.
b. Select the launch instances button on the EC2 dashboard page.
c. A page will then appear to specify the name of the instance to be created along with options to specify the type of Amazon Machine Image (AMI) to use, instance type, security group, network settings, selection of storage media specifications, and other options tailored to your needs.
d. Once all options are adjusted press the launch instance button. In this phase, an instance is formed and automatically activated.

Until the above stage, EC2 instances can run and can be used as needed for application development, but to be integrated with CI / CD there are still steps to be done, namely by adding configuration and installation of code deploy agents.

Before the code deploy agent can be installed, an IAM role must be added to the EC2 instance with the following steps:

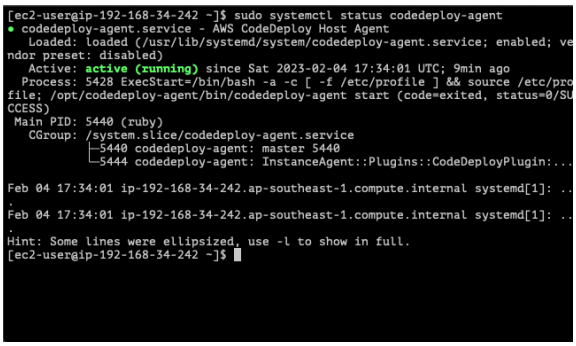
a. Sign in to the IAM console via https://console.aws.amazon.com/iam/
b. After entering the start page select 'policies' through the navigation panel.
c. Add a new policy by going through the 'create policy' button.
d. On the 'create policy' page select the JSON tab and enter the value as shown below.

Picture 3. Value JSON

1. Select 'next', and 'review policy', then give a name and description for the policy created.
2. Next, add roles by selecting the 'roles' menu on the navigation panel then select 'create role'.
3. On the 'create role' page, the AWS service to be used is EC2.
4. Make a selection on the policy that was created earlier and also add a policy with the name Amazon SSM Manage Instance Core.
5. Enter the name of the role that has been created on the review page, then select the 'create role' button.

After completing the addition of roles, the installation of the service code deploy agent on the EC2 instance can be carried out with the following steps:

1. Access the EC2 instance console via remote SSH.
2. Download the CodeDeploy Agent installer from your Amazon bucket by region.
3. Change the permissions file so that it can be executed, then run the installer.
4. After successfully performing without errors make sure the codedeploy agent service is running.



Gambar 4. Service codedeploy-agent aktif.

B. Build

At this stage, the pipeline setup is carried out to be used. This pipeline will later be connected to the application repository that was created in the previous stage.

1) AWS CodeDeploy

a. Login to the AWS console at https://console.aws.amazon.com/codesuite/codedeploy/home address
b. As a first step, an application and group deployment are first created which will later be assigned to the pipeline to be created. Select Deploy > Applications on the panel menu, then select the 'Create Application' button.
c. Enter the application name on the application name form and select the platform option used which is EC2/On-premises.
d. After selecting the 'Create Application' button it will then be directed to a page to create a deployment group of the created application.
e. Select the 'create deployment group' button and enter a name. Then select the service role that is AWS CodeDeploy, For the deployment setting option select CodeDeployDefaultAllAtOnce. The load balancer did not need to be selected in this study.

2) AWS CodePipeline

a. The next step is to create a pipeline, namely by selecting the Pipeline menu on the navigation pane, and then selecting the Create Pipeline menu, or it can also be through the wizard provided on the Getting Started page.
b. Name the pipeline to use, choose a name that is relevant to the name of the application to be built so that it will be easy to distinguish from other pipelines. In this study, the author named the pipeline "pipeline-dev-and". In this process, it will automatically establish new service roles needed in the pipeline creation process.
c. In the next step is to determine the source provider used, in this case, it is the central repository that has previously been created, using the GitHub repository. Connect the GitHub account you want to use via the Connect to GitHub button.
d. After the account is connected, select the name of the connection that has been created and navigate to the name of the application repository and the name of the branch that will be integrated into the staging server. For other options, there is no need to change it and leave it by default.
e. After clicking the next button on the previous page, the next page will appear to select the build provider tools to be used. Because this is optional, the author skipped this stage because for this research applications with the Laravel framework do not require a build process using a third-party provider.
f. The next process is the deployment stage. That is, choosing the Deploy provider to use in this case is AWS CodeDeploy. Select a region and then navigate to the name of the application and deployment group that asŽQOe created before the pipeline creation stage.
g. After all settings are correct, pipeline creation can be processed by selecting the Create pipeline button. With the formation of the pipeline, it will automatically execute the build and test process against the source code connected to the central repository. The results of the execution will be seen through the event log which informs that the script and source code deployed through the pipeline is successful

or failed so that tracing the source of the deployment process failure can be carried out and then corrections and improvements can be made.

From the series of processes above, will then produce the source code of the application that has been built and tested through the pipeline. The source code of this application will then be deployed to staging and production servers.

C. Staging

As a continuation of the build/test stage in the CI/CD workflow, the staging stage can already be seen as the results of the application source code that has been successfully deployed in the root directory on the preconfigured web server. However, when access is made through the public IP server using an internet browser, the application still cannot be displayed correctly or error, It happens because the source code deployed has not been set to the .env file which is the main configuration file needed by Laravel framework-based applications.

For this reason, the author will configure the required .env files and at the same time demonstrate how adding and changing files in the local repository can automatically execute the deployment process to the staging server. Here are the steps taken:

1. When you first initiate a Laravel framework-based application project in the local repository, by default it will generate a.env. example file in the root directory of the application. This file will be copied into a separate file named env. Before copying the author first make sure that the .env file name is removed from the. git ignore file because otherwise by default the .env file will be ignored by the Git system.
2. Change the contents of the .env file, and adjust the endpoint of the database to be used.
3. After customization, the .env file is saved and then experimented with accessing the application through a browser. As a result, the application can be displayed as shown below.

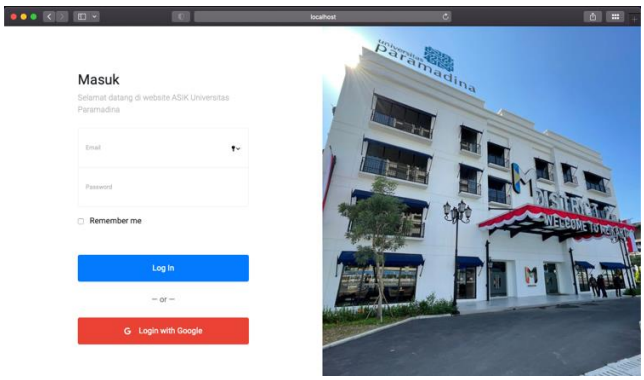


Figure 5. App View in an on-premises environment

4. After the application can run in the local environment, all changes that have been made are then committed and pushed to the central repository. In the commit process,

the author gives the message "setup.env" as a message of changes or additions that have been made before pushing.

5. From the push process above, trigger the CI / CD pipeline that has been formed to automatically carry out the build/test process and then continue with the deployment process to the staging environment. Here's what CodePipeline's output looks like a few moments after the push indicates that the additions or changes made have been successful or successful.

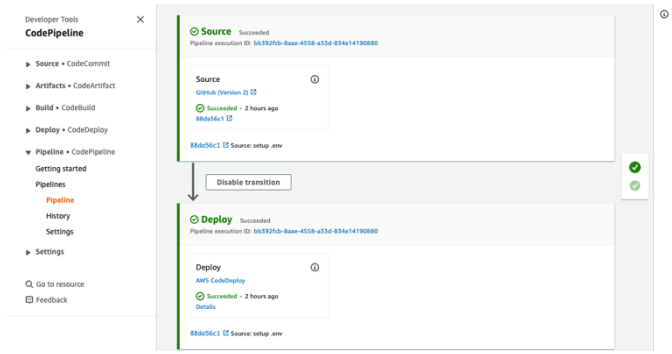


Figure 6. Pipeline Staging View

6. Results in the staging environment can be directly viewed through an internet browser with the IP address of the staging server. The image below is a view of the application that has been successfully deployed and opened through an internet browser

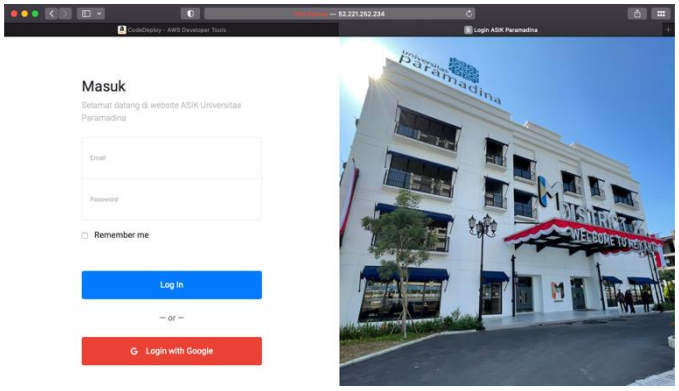


Figure 7. Application View on Staging Server

From the demonstration above, the application deployment process to the staging environment carried out through the CI / CD pipeline has been successful and then can be tested functionally by Quality Assurance (QA) to ensure that the application has functioned correctly and there are no bugs so that it can then be launched in a production environment.

D. Production

At this stage, all changes and additions to the application that have been tested in the previous stage are integrated into the product that is ready to use. Then the product is deployed to a production server and ready for end-user use.

For the implementation of the production stage, a separate pipeline is needed and it is also necessary to create branches or branches in the application's source code repository. In addition, a server is also needed as a host for the application production environment.

1) Production Server: As with server staging, for production servers setup and configuration are carried out starting from the web server, PHP, and its extensions, as well as libraries and dependencies needed. All use software versions that are identical to server staging.
2) Git Branch: Git branch is a feature in Git that allows developers to create branches or versions of existing source code that can be changed and developed separately from the master branch. With this branch feature, it can be created more than two versions of the main source code.

Thus, for applications in the production environment, branches will be created and sourced from the main source code that has previously been integrated into the staging environment. The following are the steps taken in creating a new branch:

a. Run the git branch command and the branch name that will be used in the root directory of the application source code. In this case, the new branch is named "production".

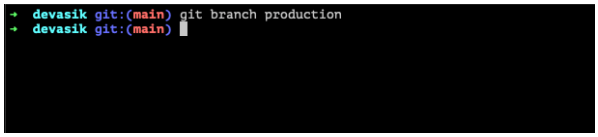


Figure 8 Create Branch Production

b. To check the branch that has just been created, run the git branch command so that it produces output as shown below, where there are now two branches, namely main and production.

Figure 9. Git Branch result view

c. After the new branch is successfully created in the local repository, it will also be created in the central repository. To do so, first, perform "git checkout" to the new branch and then run the command "git push origin production".

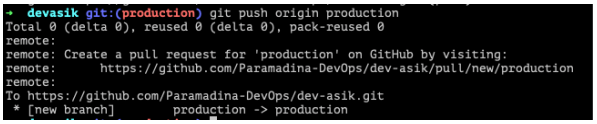


Figure 10. Push view to the production branch

d. With this command, a new branch with the name production will be formed on the side of the central repository. This branch is then connected to the new pipeline which is then integrated with the production environment.

3) Production Pipeline: The steps are the same as in creating a pipeline for the build stage, only the pipeline naming is differentiated. In addition, it is necessary to create additional deployment groups for the integration process into the staging environment.

a. Re-create the deployment group named "Prod-Asik-Group" in the DevAsik application. Details of the configuration as shown in the image below.

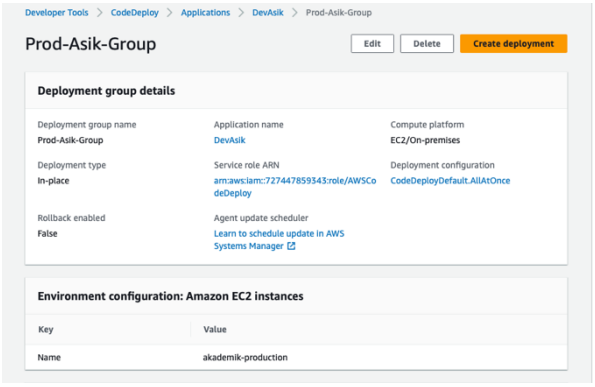


Figure 11. Production Group Deployment View

Noteworthy is that the environment configuration section specified for the EC2 Instance is a server used to integrate source code in production environments.

b. Then the next step creates a new pipeline with the name "pipeline-prod-and". For detailed configuration, the author displays in the form of screenshots as below.

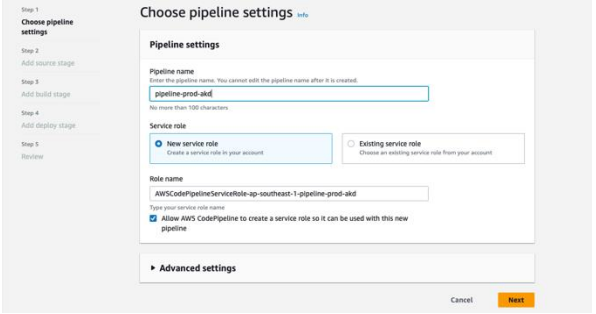


Figure 12. Production Pipeline View

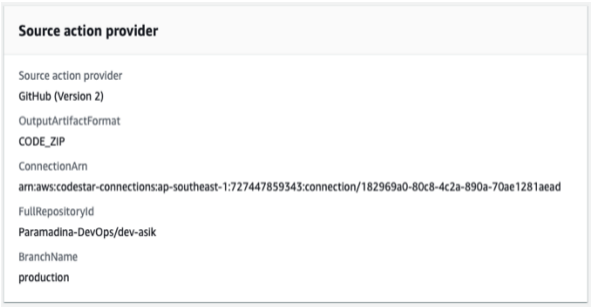


Figure 13. Source Provider Production Display

The source provider uses the same repository as staging, but the branch uses the name of the production branch.

c. After the pipeline is formed, the deployment process to the production environment can be immediately seen through the active pipeline page as well as in the staging environment.

Furthermore, the mechanism of integrating applications from the staging environment to the production environment can be done simply by merging from the main branch to the production branch using the branch merging feature available in the Git tool.

Before doing the git merge process into the production environment, it must be ensured that the application in the staging environment has no problems or bugs. After the git branch process to the production branch is carried out, the CI / CD process will automatically run and the results will be identical to the results in the staging environment.

Conclusion

The process of deploying the source code of academic information system applications to the production environment is successfully carried out by implementing the appropriate stages in the CI / CD workflow using available tools and technologies. Thus, the process of deploying application source code that was previously entirely done manually can be replaced with a CI / CD system. Although the process of integrating application source code can be done automatically both in staging environments and in production environments, there are still processes that require human intervention, namely setup, and configuration of "env" files in the Laravel framework that still have to be done manually. CI/CD implementation speeds up the application deployment process, minimizes the possibility of bugs, and the development team becomes easier to make updates and changes at any time when needed.

Bibliography

Achdian, Asfin, & Marwan, M. Akbar. (2019). Analysis of CI/CD Application Based on Cloud Computing Services on Fintech Company. *CD Application Based on Cloud Computing Services on Fintech Company*, *4*(3), 112–114.

Arachchi, SAIBS, & Perera, Indika. (2018). Continuous integration and continuous delivery pipeline automation for agile software project management. *2018 Moratuwa Engineering Research Conference (MERCon)*, 156–161. IEEE.

Aswati, Safrian, Mulyani, Neni, Siagian, Yessica, & Syah, Arridha Zikra. (2015). Peranan sistem informasi dalam perguruan tinggi. *Jurteksi Royal Edisi2*.

Chapman, David. (2014). Introduction to DevOps on AWS. *Amazon Web Services*.

Ghimire, Ramesh. (2020). *Deploying Software in the Cloud with CICD Pipelines*.

Hadian, Nur, Hakim, Mujibul, & Fanani, M. Rudi. (2023). Implementasi Model Service-Oriented Architecture (SOA) dalam Perancangan Sistem Informasi UMKM. *Jurnal Teknologi Dan Sistem Informasi Bisnis*, *5*(3), 311–318.

Hamid, Abdul, & Prasetyowati, M. S. Dr Riris Aishah. (2021). *Metodologi Penelitian Kualitatif, Kuantitatif, Dan Eksperimen*. CV Literasi Nusantara Abadi.

Laksito, Arif Dwi. (2022). IMPLEMENTASI CONTINUOUS INTEGRATION/CONTINUOUS DELIVERY (CI/CD) PADA PERFORMANCE TESTING DEVOPS. *Journal of Information System Management (JOISM)*, *4*(1), 62–66. https://doi.org/10.24076/joism.2022v4i1.887

Sutarman, Muhammad Daniswara, Fadli, Ari, & Aliim, Muhammad Syaiful. (2023). Perancangan Infrastruktur Devops Sistem Informasi Sentra HKI Lppm Unsoed. *TESLA: Jurnal Teknik Elektro*, *25*(1), 59–71.

Taryana, Acep, Fadli, Ari, & Nurshiami, Siti Rahmah. (2020). Merancang Perangkat Lunak Sistem Penjaminan Mutu Internal (SPMI) Perguruan Tinggi yang Memiliki Daya Adaptasi Terhadap Perubahan Kebutuhan Pengguna secara Cepat dan Sering. *Jurnal Al-Azhar Indonesia Seri Sains Dan Teknologi*, *5*(3), 121.

Tohirin, Tohirin, Utami, Sri Farida, Widianto, Septian Rheno, & Al Mauludyansah, Widhy. (2020). Implementasi DevOps Pada Pengembangan Aplikasi e-Skrining Covid-19. *JURNAL MULTINETICS*, *6*(1).

Yindrizal, Yindrizal. (2021). Dampak Penggunaan Sistem Informasi Akademik Untuk Meningkatkan Kualitas Pelayanan Akademik Mahasiswa Universitas Andalas Padang. *Jurnal Manajemen Publik Dan Kebijakan Publik (JMPKP)*, *3*(1).