


Implementation of Continuous Integration and Continuous Deployment (CI/CD) to Speed up the Automation Process of Software Delivery In the Production Process Using Node.Js, Docker, and React.Js

Nurhayati

Computer Science Faculty, Pamulang University of Indonesia

| Article Info | ABSTRACT |
|--|--|
| <p>Keywords: ContinuousIntegration, Continuous Delivery, Developer, Software</p> | <p>With the increasing amount of software that has been developed by developers, this encourages experts in the software field to release products with quality and updates on a regular and scheduled basis. In the software development process, teamwork is crucial, where a developer must work together with his colleagues. Of the various software development processes that occur in the IT Cyber Community, challenges often occur at the code integration stage. This challenge arises when a lot of code from different developers has to be combined. Not only that, the time required is very fast to get new features. Continuous Integration/Continuous Delivery (CI/CD) can be a solution to overcome this problem. Through the implementation of CI/CD, the software integration and delivery process becomes more automated, allowing developers to integrate code more regularly and minimizing the risk of problems that may arise due to human error and also delays in deploying a new feature. The automation process in React.js and Node.js software development using Docker and Jenkins was successfully carried out by implementing Continuous Integration/Continuous Deployment in testing, code that was integrated more regularly and software delivery was faster. In the process, the time required to deploy a software can be minimized, which previously took a long time, can now be shortened, which previously took 8 minutes 48 seconds, now only takes 2 minutes 56 seconds.</p> |
| <p>This is an open access article under the CC BY-NC license</p>  | <p>Corresponding Author: Nurhayati Computer Science Faculty, Pamulang University of Indonesia dosen02378@unpam.ac.id</p> |

INTRODUCTION

Basically, technology is always developing every day and giving birth to many new applications whose uses can simplify and speed up human work. In the process of producing an application, there are several stages that must be carried out by a developer, such as the process of analysis, design, planning, testing and release. After the design and testing have been completed, usually the next step is to deploy the application or application. can be accessed and used by endpoints (users). However, in the deployment

Implementation of Continuous Integration and Continuous Deployment (CI/CD) to Speed up the Automation Process of Software Delivery In the Production Process Using Node.Js,

Docker, and React.Js— Nurhayati

process, manual systems are still used (Farid and Anugrah, 2021). The method that is often done is by entering an application into the server then running the application and if there is an update in the application then this step must be repeated continuously. This will create a less efficient method because a developer has to repeatedly deploy the application every time there is a change.

Previous research conducted by (Farid and Anugrah, 2021) the implementation of CI/CD carried out by the author was very It is also explained that the deployment stage is a very important stage in determining the final result of an application. There is a problem where the development team has created the features needed by the company but has to deploy them manually, which is less efficient. (Fitriyyah, Safriadi and Pratama, 2019) This research aims to apply a Continuous Integration practice to a Bengkalis Regency Karang Taruna profile web application, whether the Continuous Integration (CI) practice has been implemented successfully or not by carrying out simple tests including cloning, pulling, pushing profile web application code artifacts in Github Version Control System software.

(Shama and W. Chandra, 2021). This research aims at applications that will be developed by PT Emporia Digital Raya. there are no more vulnerabilities, and the deployment process is faster. In its processing, previous researchers adopted Jenkins as an automatic tool for building and deploying applications. In the process, it can be concluded that adopting Jenkins can speed up the automatic deployment process. (Jaeni, S. and Laksito, 2022). (Jaeni, S. and Laksito, 2022) The development process carried out traditionally on an application that has been designed, tested and deployed often encounters problems such as application delays and product quality. This problem occurs because the operational team has to wait for the developer to complete the testing process first before the application can be used by end users.

(Wahyu and Guna Noviantama, 2021) That ongoing application development in improving the features contained in the application and adapting changes to user needs really requires a concept that supports every change and can be implemented quickly. then a Continuous Integration/Continuous Deployment (CI/CD) concept was created which can focus on the application to be developed and every time a new feature is released it can be done quickly and through a series of processes tested on the Continuous Integration/Continuous Deployment (CI/CD) concept.). (Pane et al., 2021) With the Restfull Web Service Method and CI/CD Testing, Coverage and Grafana Protocol Simulation," it is explained that to get results that can help in carrying out a monitoring data analysis process in graphical form, the use of the CI/CD pipeline in carrying out implementation can provide comfort at this stage. develop and reduce bugs.

(Parama, Studiawan and Akbar, 2022) In the MyITS Single Sign On Application, it is explained that in the application development process they carry out, where the delivery and deployment process is carried out manually, the developer or developers who have developed the application push it to the code repository which will then be released to the server. In this problem, researchers implemented Continuous Integration/Continuous Deployment (CI/CD) to obtain the automation process expected by researchers. (Connelly

et al., 2022) Modeling Software with CI/CD and Jenkins" explained that researchers need to implement more effective software testing methods to increase the efficiency of team workflow and software. By modeling software with CI/CD, it is hoped that the risk of errors and errors in the software being developed will be reduced.

(Battina, 2021) explained that exploring migration challenges and strategies When adopting CI/CD during software development, usually the development and operational teams are sometimes at odds in providing products for consumers. So the solution that occurs in research is to implement CI/CD. (Achdian and Marwan, 2019) there is a conflict of interest between developers who want to write program code quickly, testers who want to test the program to find all bugs, and operational units. From the discussion, there is a need for a mechanism to resolve this conflict and harmonize the function of these elements so that the project can run smoothly.

CI/CD (Continuous Integration / Continuous Delivery) is an approach to software development that aims to automate and speed up the process of delivering software to a production environment. By using the application of CI/CD (Continuous Integration / Continuous Delivery) this can speed up the software integration and delivery process for a developer. Any integrations that have been verified by the automated build system, including testing to detect errors, are performed as soon as possible (Setiawan, 2021).

In the software development process that uses CI/CD (Continuous Integration / Continuous Delivery) for a website project that uses the Reactjs framework, tools such as Docker and Jenkins are needed so that the automation process can run according to predetermined procedures (Muhammad, 2022).

METHODS

Research Methods and Implementation

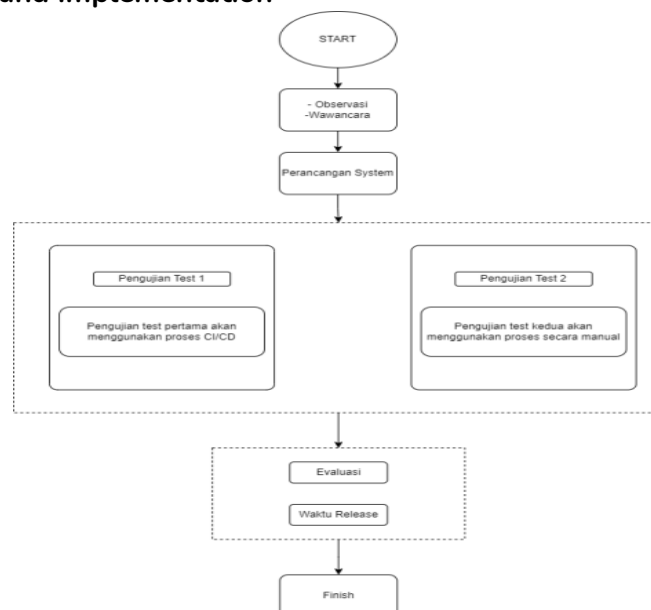


Figure 1. Research Methods and Implementation

Research Methods

The data collection process carried out in this research was by using literature studies, field observations, interviews, and bibliography. This aims to collect related research which can be used as a reference source in the research to be carried out .

The next stage that the researcher carried out was searching through journals and collecting them as references, then the author continued by looking for information that could be used as a theoretical basis, research methods, and everything else related to this research. The search for related references is based on learning several things as follows (Code fresh,):

1. Previous developments related to applications still used manual methods.
2. Previous research regarding Continuous Integration/Continuous Deployment (CI/CD) in various cases.
3. Previous research regarding Continuous Integration (CI) in various journal cases.
4. Previous researchers regarding the use of Docker in journal cases.
5. Previous researchers regarding the Jenkins journal case.
6. Previous researchers regarding Webhooks in journal cases.

Implementation Methods

The system to be built is divided into two important stages to find out the comparison between the two processes, namely, the first test which will use the implementation of Continuous Integration/Continuous Deployment on applications that have been built using react.js and node.js previously. The second test is carried out manually, where the application will be deployed manually. To carry out the implementation of this system, it is very necessary to prepare a server with technical specifications as in table 3.1.

Table 1. Specifications of the Virtual Machine used

| | |
|-----------------|-------------------------|
| Provider | IdColud |
| Location | Southeast Asia(Jakarta) |
| vCPU | 2 Core |
| RAM | 4 GB |
| Oprating System | Ubuntu 20.04 |

In preparation for creating an optimal virtual environment, the author prepared specifications which can be seen in Table 3.3 as a reference in supporting CI/CD. The server that will be used as the basis for this virtual environment will be the container for the entire software infrastructure that strongly supports the CI/CD process. Provider as a service to rent a server so that the website that is built can be visited by internet users, the vCpu used is enough with 2 cores because it is still in the micro scope, the RAM used is also enough to use 4 GB as a temporary storage for the application that will be installed in it, and Operating The system used is Ubuntu 20.04.

Testing Test 1

By following the flow of the research process, the author then continued with creating a web server to operate the model, so that it could be accessed by users. This step is part of

a process that can be executed to support Continuous Integration/Continuous Deployment (CI/CD) requirements. The CI/CD process is carried out using technologies such as Docker, webhook, Jenkins, Docker, and Portainer.io. In this context, the author will explain in depth the steps taken during this research:

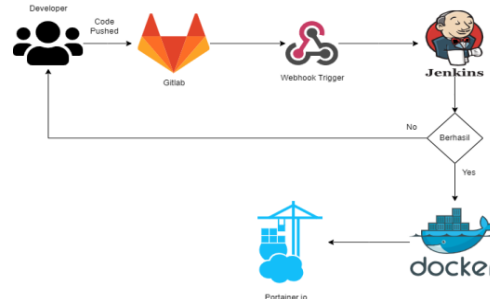


Figure 2. Continuous integration/ Continuous Deployment workflow

In the testing process, this first test will involve the process of implementing Continuous Integration/ Continuous Deployment, which can be seen in the image above, Figure 2. In the process, tools will be needed that support the Continuous Integration/ Continuous Deployment process (Red Hat, 2023), including:

- a. Gitlab
Gitlab here functions as a pipeline. In this pipeline, integrated automated steps will be created allowing the development team to store, manage and collaborate on one source which is carried out to bring code changes from the development stage to the production stage.
- b. Webhooks
Webhooks will be useful to help initiate Actions or trigger certain URLs or workflows automatically after an event or change occurs, reducing the need to monitor or manage the process manually. The webhook will be integrated into Gitlab to retrieve the address that is already in Jankins.
- c. Jankins
Jankins is used to automate the code integration process of the development team. Every time a code change is committed to the repository, Jankins can automatically start the build process.
- d. Docker
Docker akan memainkan peran penting sebagai solusi kontainerisasi untuk mengelola aplikasi yang sudah terdeploy di dalam server. Kontainerisasi menggunakan Docker memungkinkan isolasi yang kuat, memastikan bahwa setiap aplikasi berjalan dalam lingkungan yang terisolasi dan dapat diandalkan.
- e. Portainer.io
Portainer.io as a tool for monitoring the health and performance of containers, checking container logs to detect problems.

With the details of the equipment model described previously, the author will comprehensively outline the process of implementing Continuous Integration (CI) and

Continuous Deployment (CD) using two leading tools, namely GitLab and Jenkins. In-depth steps and configurations will be outlined to ensure efficient and reliable integration and deployment of code changes, optimizing the entire software development lifecycle:

Continuous Integration (CI) Implementation Process

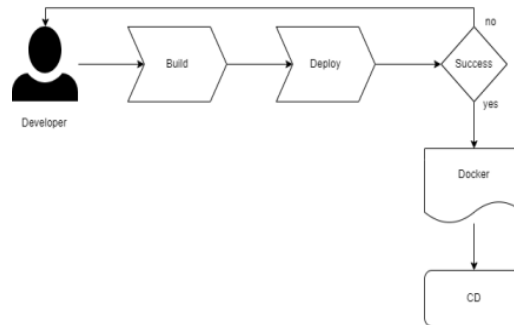


Figure 3. Picture Continuous Integration Workflow

In this process is the application of Continuous Integration (CI) where this process includes a series of processes called the build pipeline process which aims to provide instructions for the React.js and Node.js applications that will be built. This is done by getting the commit ID from the Repository and then running the source code. At this stage there is a series that needs to be passed, namely build to maintain the quality of the application by ensuring that every code change made by the developer is successfully built and tested quickly and automatically. If you have carried out a series of build pipelines, a deploy process will be carried out to create the Virtual Machine that is needed in containerization, so that applications can be isolated to maintain consistency at all stages of development.

Continuous Deployment (CD) implementation process

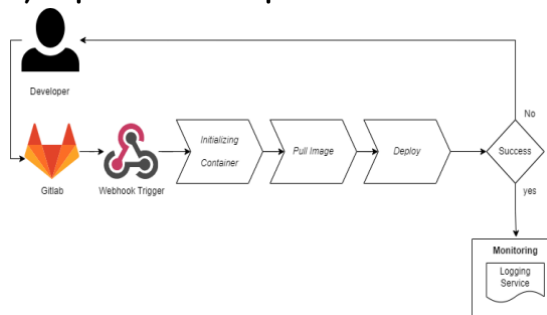


Figure 4. Continuous Deployment Workflow

In this process, Continuous Deployment (CD) is implemented, where this process includes a series of Deployment Pipelines which are intended as automation deployment instructions. Before the application reaches the end user, it goes through something called deploy to staging, which is the process when the application is pulled from the container to deploy to staging and then if it is appropriate it can go straight to the next process until it reaches production. Also, if there is a change in a containerized application, it can be changed automatically via a webhook by triggering it using the command prompt command, namely `curl<webhook_ip_address>`.

Monitoring Process

This process is carried out with the aim of monitoring and supervising every step in the CI/CD pipeline. All information regarding each stage will be recorded and stored in a log service (Logging Service) in the form of an entry log. This log entry can include information about the implementation time of the deployment and the final results of each stage. In addition, when errors or bugs occur, these problems can also be observed and studied through log entries, to make it easier to identify, solve and fix problems quickly (dinda hafid hafifah, 2023).

In this first test, continuous testing is required every time a change occurs in the application. Code that has been pushed into the master branch will automatically undergo changes without requiring a Docker reset when running the application.

Testing Test 2

In this manual model, the Manual development and operation model will be applied. This implementation is based on stages that are carried out manually and not through a pipeline process. This implementation cannot carry out monitoring easily in the form of logging to monitor each stage and notification when a failure occurs in the process. From here the author calls it the manual model.

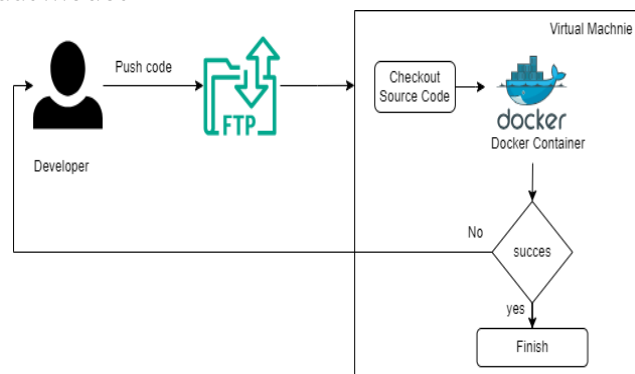


Figure 5. Manual Model Workflow

In the manual stage there are several stages where each stage that has been carried out if there is an error must be rolled back again,

Evaluation

This process is to show the evaluation results on the implementation of Continuous Integration / Continuous Deployment and implementation using a manual model which was previously carried out by the ITCyberCommunity organization. Based on implementation using Continuous Integration / Continuous Deployment and implementation using manual models, we will analyze the performance of these 2 implementation models. This analysis will include how quickly or how long it takes to update a feature in the React.js and Node.js applications from a process that has been created previously.

RESULTS AND DISCUSSION

Discussion

Application research implementation react.js and node.js by implementing a CI/CD *pipeline* with the help of *Docker* to isolate applications to remain unchanged or consistent and Jenkins as a tool so that applications can run automatically for each part of the software development process, such as building and deploying software. Jenkins can also be integrated continuously and *deployed* continuously as well. **Model Creation Process Application.**

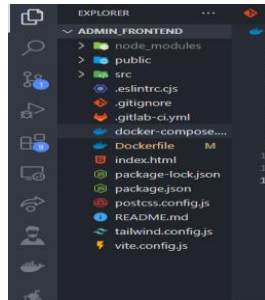


Figure 6. Application React.js

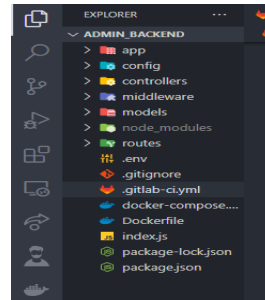


Figure 7. Application node.js

At this stage, an application from React.js and Node.js will be run *locally* using *docker* to see if the application will run properly or not. By doing it locally, you can also see whether this application can run or not.

Docker

In this phase the author will also be able to see *what images* will be needed in running the application so that it can be adjusted to support the *Continuous Integration / Continuous Deployment* process. Because the app is created using react.js and node.js the author takes the node:14-alpine Image that has been provided by *Dockerhub*. By using the same Image so that the application that runs can still run consistently, not only can it be an efficiency so there is no need to build an *Image* from scratch every time you want to start a project. The project will be run in a remote manner to avoid conflict between the *backend* and *frontend*.

```

Dockerfile > ...
1 FROM node:14-alpine
2
3
4 WORKDIR /app
5
6 COPY ./package*.json ./
7 RUN npm install
8
9 COPY . .
10
11 EXPOSE 5000
12
13 CMD ["node", "index.js"]
  
```

Figure 8. Docker file to run backend

```

Dockerfile
1 FROM node:18-alpine
2
3 WORKDIR /react-vite-app
4
5 EXPOSE 3000
6
7 COPY package.json package-lock.json ./
8
9 RUN npm install --silent
10
11 COPY . ./
12
13 CMD ["npm", "run", "dev"]
14
  
```

Figure 9. Docker file to run frontend

From the Figure 8, above it can be described as a command where a *docker image* will be created running on *port 5000* in the *local* to run the *backend* application *locally*

with the appropriate results. From the Figure 9, above it can be described as a command where a *docker image* will be created that runs on *port 3000* in the *local* to run the *backend application locally* with the appropriate results.

Web Server Creation Process

In this process the server will install Jenkins and then a plugin will be setup to support automation needs such as *Git* and *Gitlab* Versioning, *Pipeline Plugin*, and *Generic Webhook Trigger*.

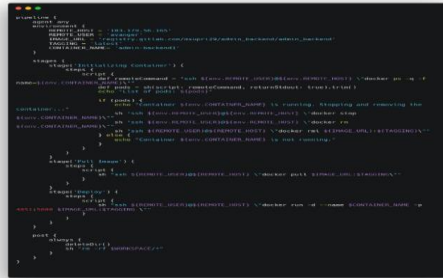


Figure 10. Groovy script on Jenkins

The picture above shows the creation of a script in which there are stages to support the needs of automation in the *pipeline process*. The process of making a *docker* which in making this *docker* occurs on a server, namely a *cloud server*.

Proses Source Code Repository

After the model creation process and web server creation are created, the author saves the program code in the *Gitlab* repository. The one in *Gitlab* also creates a program code for the integration process between *Jenkins* and *Gitlab*. *Gitlab* itself provides tools to provide *Continuous Integration / Continuous Deployment* processes called *pipelines*. This *pipeline* will run if in the environment code there is an extension that is *yml* or *yaml* this extension will run all the commands in it such as *build* and *deploy*.

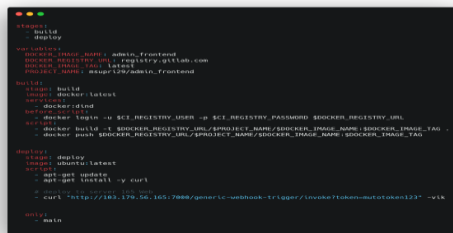


Figure 11. Extension script .yml or .yaml

In the picture above explained how the *Pipeline* will work, if in it there is a command that can be seen in the first line, namely *Stages* which is useful as a command that will be run in the *Pipeline*, in it there is an array containing *build* and *deploy* where the *build* is instructed to build a *docker* to get the *docker registry* which is *docker*. This registry will be useful to fetch in *Jenkins* later and *deploy* is useful for running the script on the server with the help of *webhooks* to trigger the running of a command.

Continuous Integration/ Continuous Deployment Continuous Integration

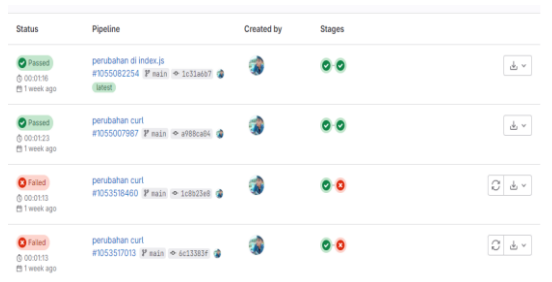


Figure 12. Pipeline Process

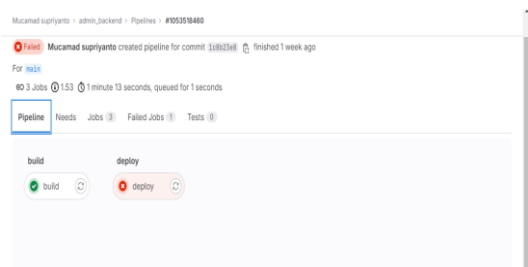


Figure 13. Process Failed Deploy

In this *pipeline process*, it will be checked whether the build process was successfully run and whether the *deployment process* was also successfully run. In figure 13, you can see the successful process and the failed process, the successful process is the build, and the failed process is the *deploy* process. To see what failures happened we can see also in the Failed Jobs section.

Continuous Deployment

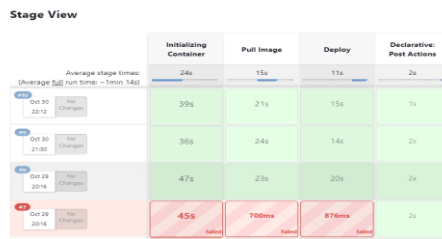


Figure 14. proses stage view

This process is where we can monitor or visualize each stage that has been created in the *plugin pipeline*. This stage view also helps to understand and monitor the software development workflow better.

Initializing Container & Pull Image

```

stages {
  stage('Initializing Container') {
    steps {
      script {
        def remoteCommand = "ssh ${env.REMOTE_USER}@${env.REMOTE_HOST} 'docker ps -q -f name=${env.CONTAINER_NAME}'"
        def pods = sh(script: remoteCommand, returnStdout: true).trim()
        echo "List of pods: ${pods}"

        if (pods) {
          echo "Container ${env.CONTAINER_NAME} is running. Stopping and removing the container..."
          sh "ssh ${env.REMOTE_USER}@${env.REMOTE_HOST} 'docker stop ${env.CONTAINER_NAME}'"
          sh "ssh ${env.REMOTE_USER}@${env.REMOTE_HOST} 'docker rm ${env.CONTAINER_NAME}'"
          sh "ssh ${env.REMOTE_USER}@${env.REMOTE_HOST} 'docker rmi ${IMAGE_URL}:${TAGGING}'"
        } else {
          echo "Container ${env.CONTAINER_NAME} is not running."
        }
      }
    }
  }
}
    
```

Figure 15 Script Pipeline Initializing Container

```

stage('Pull Image') {
  steps {
    script {
      sh "ssh ${REMOTE_USER}@${REMOTE_HOST} 'docker pull ${IMAGE_URL}:${TAGGING}'"
    }
  }
}
    
```

Figure 16. Script Pipeline Pull Image

Initializing Container in figure 15, is a process of preparation and initial configuration required for services in the container. In this section figure 16, will be remote a command

Implementation of Continuous Integration and Continuous Deployment (CI/CD) to Speed up the Automation Process of Software Delivery In the Production Process Using Node.js,

Docker, and React.js— Nurhayati

to pull an *Image* in the *Image* url contained in Gitlab that was created earlier during the Integration process.

Deploy & Monitoring

```

stage('Deploy') {
  steps {
    script {
      sh "ssh $(REMOTE_USER)@$(REMOTE_HOST) 'docker run -d --name $CONTAINER_NAME -p 4051:5000 $IMAGE_URL-$STAGING'"
    }
  }
}
    
```

Figure 17. Script Pipeline Deploy

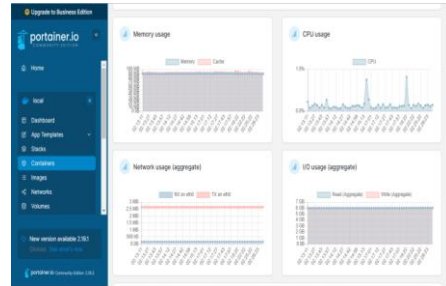


Figure 18. Monitoring Docker

Figure 17, In this section, we will build a *dockerfile* using a remote connection so that *docker* can run on the virtual server that was created earlier. Figure 18, In this *monitoring* process, you can explore the use of memory, CPU, Network, and when the application turns off and on. This monitoring will be very useful if one day the use of the CPU is too large or the memory is too large it will be overcome by giving enough CPU or memory in a software.

Model Manual

In the manual model this is very crucial because the steps will affect the arrival of the production environment or end user. The initial stage starts from the code that is pushed into the machine virtual environment Through FTP (*File Transfer Protocol*) the *file* will be isolated in a container and then created an *image* to run *docker* build so that it can run on the *server side*.

Evaluation

After completing the *Continuous Integration / Continuous Deployment process* and also the manual model, it will be evaluated by the author starting from the application of *Continuous Integration / Continuous Deployment* and also the manual model. In the application of *Continuous Integration / Continuous Deployment*, the calculation of time until it can be accessed requires a fairly short time while to use the manual calculation using a stopwatch requires a long time.

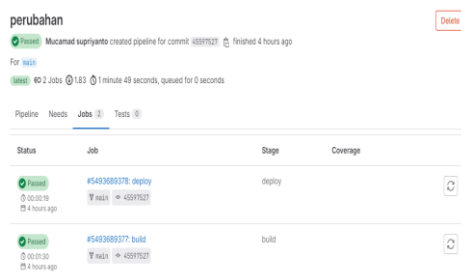


Figure 19. Required CI/CD

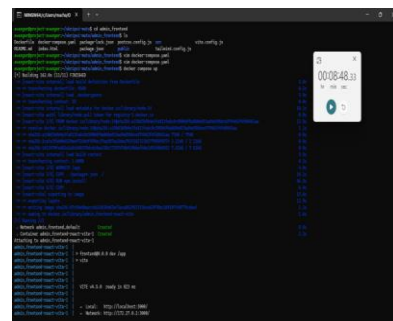


Figure 20. Time required Model Manual

Implementation of Continuous Integration and Continuous Deployment (CI/CD) to Speed up the Automation Process of Software Delivery In the Production Process Using Node.js,

Docker, and React.js— Nurhayati

From the time shown in the table above, it can be concluded that the time using the *Continuous Integration/Continuous Deployment* process takes about 00:02:56, while the process using the manual model gets around 00:08:48. The result of that comparison is about 00:05:92. The *Continuous Integration / Continuous Deployment* process is also a long-term process where every new new feature will be easily integrated and *deployed* easily.

Result

Continuous Integration/ Continuous Deployment Evaluation Results

Table 2. Continuous Integration/ Continuous Deployment Duration Time

| Evaluasi | Waktu yang dibutuhkan |
|------------------------|-----------------------|
| Continuous Integration | 00:01:49 |
| Continuous Deployment | 00:01:07 |
| Hasil | 00:02:56 |

Table 2. shows that the time or duration required for integration only takes 00:01:49 seconds, while for the Deployment process it takes 00:01:07 seconds. From the table above it can be concluded that it only takes 00:02:56 seconds to complete. easy to deploy an application.

Manual Model Evaluation Results

Table 3. Manual Model Duration Time

| Evaluasi | Waktu yang dibutuhkan |
|------------------|-----------------------|
| Pull code | 00:02:40 |
| Docker Container | 00:06:08 |
| Hasil | 00:08:48 |

Table 3 shows the results of the time or duration required for the code to be pulled into the Virtual Machine. After the code is copied or cloned, the code will be created in a container to be built and run in Docker so that it can be accessed on the server.

CONCLUSION

The automation process of React.js and Node.js software development using *Docker* and *Jenkins* was successfully carried out by implementing *Continuous Integration / Continuous Deployment* in testing, code that has been integrated more regularly and software delivery faster. In the process, the time needed to deploy a software can be minimized, which previously took a long time, can now be shortened, which previously took 8 minutes 48 seconds, now only takes 2 minutes 56 seconds. The positive impact provided is in the form of minimizing time and errors due to human error, if there is a change there is no need to repeat *the deployment* manually because everything is automated. Based on the research that has been carried out, according to the conclusions obtained there are still several shortcomings. Therefore, it provides several suggestions that can be implemented so that in the future, if there is research that is somewhat similar, it can be carried out better. Suggestions that can be given include: Conduct regular evaluations of *CI/CD*

implementation, and update procedures as needed. Encourage better collaboration between development team members by leveraging collaborative tools. This can increase efficiency and strengthen relationships within the team. As test results show significant time savings, consider developing additional features that can add value to the software development process

REFERENCE

- Achdian, A. and Marwan, M.A. (2019) 'Analysis of CI/CD Application Based on Cloud Computing Services on Fintech Company', *International Research Journal of Advanced Engineering and Science Asfin Achdian*, 4(3), pp. 112–114. Available at: <https://puppet.com/resources/whitepaper/state-of-devops-report>.
- Battina, D.S. (2021) 'Improving La Redoute ' s CI / CD Pipeline and DevOps Processes by Applying Machine Learning Techniques', 8(10), pp. 8–11.
- Code fresh (no date) 'CI/CD Pipeline Stages and Phases'. Available at: <https://codefresh.io/learn/ci-cd-pipelines/ci-cd-process-flow-stages-and-critical-best-practices/>.
- Connelly, L.T. et al. (2022) 'Automated Unit Testing of Hydrologic Modeling Software with CI/CD and Jenkins', *Proceedings of the International Conference on Software Engineering and Knowledge Engineering, SEKE*, pp. 225–230. Available at: <https://doi.org/10.18293/SEKE2022-074>.
- dinda hafid hafifah (2023) 'perancangan proses'. Available at: <https://www.liputan6.com/hot/read/5358749/monitoring-adalah-proses-menganalisa-informasi-secara-sistematis-ketahui-dampak-dan-bentuknya>.
- Farid, A. and Anugrah, I.G. (2021) 'Implementasi CI/CD Pipeline Pada Framework Androbase Dengan Menggunakan Jenkins (Studi Kasus: PT. Andromedia)', *Jurnal Nasional Komputasi dan Teknologi Informasi (JNKTI)*, 4(6), pp. 522–527. Available at: <https://doi.org/10.32672/jnkti.v4i6.3703>.
- Fitriyyah, S.N.J., Safriadi, N. and Pratama, E.E. (2019) 'Analisis Sentimen Calon Presiden Indonesia 2019 dari Media Sosial Twitter Menggunakan Metode Naive Bayes', *Jurnal Edukasi dan Penelitian Informatika (JEPIN)*, 5(3), p. 279. Available at: <https://doi.org/10.26418/jp.v5i3.34368>.
- Jaeni, J., S., N.A. and Laksito, A.D. (2022) 'Implementasi Continuous Integration/Continuous Delivery (Ci/Cd) Pada Performance Testing Devops', *Journal of Information System Management (JOISM)*, 4(1), pp. 62–66. Available at: <https://doi.org/10.24076/joism.2022v4i1.887>.
- Muhammad, A. (2022) 'Pengertian, Manfaat CI CD'. Available at: <https://www.niagahoster.co.id/blog/ci-cd-adalah/>.
- Pane, S.F. et al. (2021) 'Implementasi Middleware Pada Evomo Dengan Metode Web Service Restfull Dan Pengujian CI/CD, Coverage Serta Simulasi Protokol Grafana', *Jurnal Tekno Insentif*, 15(2), pp. 110–121. Available at: <https://doi.org/10.36787/jti.v15i2.507>.

- Parama, R.A., Studiawan, H. and Akbar, R.J. (2022) 'Implementasi Continuous Integration dan Continuous Delivery Pada Aplikasi myITS Single Sign On', *Jurnal Teknik ITS*, 11(3). Available at: <https://doi.org/10.12962/j23373539.v11i3.99436>.
- Red Hat (2023) 'What is continuous deployment?'
- Setiawan, R. (2021) *No Title*. Available at: <https://www.dicoding.com/blog/apa-itu-ci-cd/>.
- Shama, A.M. and W. Chandra, D. (2021) 'Implementasi Static Application Security Testing Menggunakan Jenkins Ci/Cd Berbasis Docker Container Pada Pt. Emporia Digital Raya', *Jurnal Ilmiah Informatika*, 9(02), pp. 95–99. Available at: <https://doi.org/10.33884/jif.v9i02.3769>.
- Wahyu, A.P. and Guna Noviantama, I. (2021) 'Implementasi Contionous Integration Dan Continous Deployment Pada Aplikasi Learning Management System Di Pt. Millennia Solusi Informatika', *Jurnal Ilmiah Teknologi Infomasi Terapan*, 8(1), pp. 183–186. Available at: <https://doi.org/10.33197/jitter.vol8.iss1.2021.744>.