Sinkron : Jurnal dan Penelitian Teknik Informatika
Volume 7, Number 4, October 2023
DOI : https://doi.org/10.33395/sinkron.v8i4.12624

e-ISSN : 2541-2019
p-ISSN : 2541-044X

Implementation of Continous Delivery using Jenkins And Kubernetes with Docker Local Images

Danur Wijayanto[1)*], Arizona Firdonsyah[2)]
[1,2)]Information Technology, Faculty of Science and Technology, Universitas 'Aisyiyah Yogyakarta, Indonesia
[1)]danurwijayanto@unisayogya.ac.id, [2)]arizona@unisayogya.ac.id

Submitted : Jul 2, 2023 | **Accepted** : Aug 31, 2023 | **Published** : Oct 1, 2023

Abstract: The need for software has increased and the development process has also become more complex as technology has developed which resulted application will take a long time to deployed, which can be completed in weeks or even months. One reason is the large number of teams involved in application development, especially the Development Team and Operations Team. These teams can cause the application delivery process to the user's side to be hampered if a conflict occurs, so the term DevOps appears. Support for DevOps continues to be improved, so there is CI/CD (Continuous Integration and Continuous Delivery/Deployment). Banyak penelitian mengenai CI/CD yang sudah dilakukan dan menggunakan tools Jenkins, Kubernetes, dan Docker. Namun penelitian yang sudah ada menggunakan repository DockerHub untuk menyimpan Docker Image This research focuses on the local implementation of the docker image which is then run with Kubernetes Orchestration so it can reduce the platform used by CI/CD. This implementation requires conversion from docker image to Kubernetes Image Cache. The results show that the Continous Delivery implementation using Kubernetes with the Local Docker image is successful and can run well. The results show that the average time required to create a docker image is 649 seconds (10 minutes 49 seconds) and image conversion process which takes an average time of 275 seconds (4 minutes 35 seconds). Research can be developed further by researching techniques to shorten build time, minimize resource utilitation and minimize time conversion from docker images to Kubernetes Image Cache.

Keywords: CI/CD, Continous Delivery, Continous Deployment, DevOps, Continous Integration, Jenkins

INTRODUCTION

The need for software has increased and the development process has also become more complex as technology has developed. This affects the organization in sending applications to production. The more complex an application will take a long time to update the application, which can be completed in weeks or even months. One reason is the large number of teams involved in application development, especially the Development Team and Operations Team. These teams can cause the application delivery process to users to be hampered when conflicts occur (Leite et al., 2020). This problem is one of the reasons for the emergence of the DevOps principle that collaborates the Development Team and IT Operations (Kim et al., 2016).

Leite et al (2020) states, DevOps is a collaborative and multidisciplinary effort within an organization to automate continuous delivery of new software versions, while guaranteeing their

*name of corresponding author

correctness and reliability. In DevOps, CI/CD Pipelines are known for facilitate the process of integration and delivery of software to users (Freeman, 2019). CI/CD stands for Continous Integration, Delivery, and Deployment (Shahin et al., 2017). By implementing CI/CD the process of building, deploying and testing a software is carried out automatically with the aim of speeding up delivery and getting feedback from application users as well as facilitating its distribution.

There are quite a lot of supporting tools for CI/CD implementation, such as git for source code, control for Continuous Integration, Jenkins as a CI/CD Pipeline, and docker to make it easier to package applications to be deployed. However, when building a CI/CD environment using Docker, users must build a container image before deployment. The container image itself usually has to be stored in a repository such as Docker Hub or Amazon ECR when using AWS. However, in this study, researchers will not save it to a repository such as Docker Hub or Amazon ECR but will store it on a local server, so that they can save costs for renting Docker Hub or Amazon ECR services.

Many studies on CI/CD have been carried out. Research conducted by Singh et al (2019) used CI/CD tools to deploy docker-based microservices on AWS by comparing Jenkin and Gitlab CI/CD. This research shows that both Gitlab CI/CD and Jenkins have their own advantages and need to be adjusted according to needs. Research on CI/CD tools such as Jenkins, Kubernetes, Docker was also conducted by Alpery & Ridha (2021), Fadil et al., (2020), Garg & Garg (2019), Luhana et al (2018), Mahboob & Coffman (2021), R & Mohana (2022), Singh et al (2019), dan Yin et al (2021). In some of these studies, it shows that CI/CD aims to facilitate automatic application deployment and minimize downtime. Kubernetes itself is a framework for managing containers such as docker on clusters with the help of local Daemons (the Kubelet) for lifecycle management. (Jeffery et al., 2021; Rodriguez & Buyya, 2019). The tools used are also as varied as the research conducted by (Mahboob & Coffman, 2021) not using Jenkins but using Tektons for CI/CD Pipelines. Apart from using Docker, research conducted by Mahboob & Coffman (2021) and Yin et al. (2021) uses Kubernetes for Docker Orchestration to facilitate container management. To be able to use Docker, it is necessary to carry out the build process to form an image which is then stored in a repository such as Docker Hub. The research conducted (Fadil et al., 2020) uses a private image repository which is distributed on several servers so that the research requires more than one server.

Apart from using Docker, CI/CD can be done without using Docker, like research conducted by Danur Wijayanto et al. (2021). This research uses the help of the PM2 library to manage running applications so that it can shorten deployment time, but there are weaknesses, namely there is downtime during the deployment process. This downtime is caused because the deployment is done by directly replacing the running service. To overcome these problems, we can using Containers with Kubernetes Orchestrator so deployment can be managed properly. This research focused on implementation Kubernetes using local images because no one has discussed it. Previous research used Online Repositories such as Docker Hub and Harbor as a container registry so that to use these services it is necessary to configure and incur additional costs. With this research, we hope to add insight into CI/CD implementation especially for Continous Delivery and show that docker images can be stored on a local server so it can reduce the platform used by CI/CD.

METHOD

The method used in this study are shown in Figure 1: (1) Git Configuration, implementing Code Integration using Github, (2) Jenkins configuration, configuring CI/CD Pipelines using Jenkins to support the Continuous Delivery/Continous Deployment process, (3) Kubernetes Configuration, configuring Kubernetes in running the application, then finally (4) Testing, carrying out the testing process starting from Continous Delivery to Continous Integration/Continuous Deployment.

*name of corresponding author

Sinkron : Jurnal dan Penelitian Teknik Informatika
Volume 7, Number 4, October 2023
DOI : https://doi.org/10.33395/sinkron.v8i4.12624
e-ISSN : 2541-2019
p-ISSN : 2541-044X

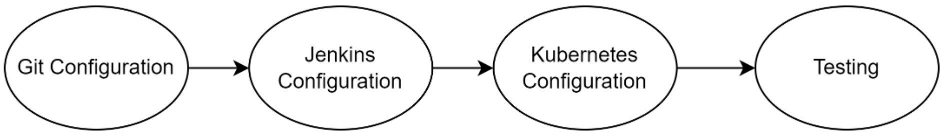


Fig 1. Research Method

This research uses a VPS from Contabo (Contabo, n.d.) with the specifications as in Table 1. The system built requires 1 VPS as a place to run applications and tools such as Jenkins, Kubernetes, and the Docker image repository. Meanwhile, the software requirements are shown in Table 2.

Table 1. Hardware Specification

Component	Spesification
CPU	4 vCPU
RAM	8 GB
SSD	200 GB SSD

Table 2. Software List

Tools	Description	Version
GitHub	Code Repository	-
Jenkins	CI/CD Pipeline	2.346.1
Docker	Container	20.10.12-0ubuntu4
Kubernetes	Container Orchestration	v1.24.13-2+cd9733de84ad4b

Git Configuration

This study uses Github as a code repository. In this study, we used 2 main branches, namely the master and staging branches as well as the feature branch taken from the master branch for the development process. The master branch is used to release and merge application code so that it can be used by developers for next software feature. While the staging branch is used to combine code from the developer module so that testing can be done in a staging environment. The relationship between Tools, Devices, Developers, and system users is shown in Figure 2.

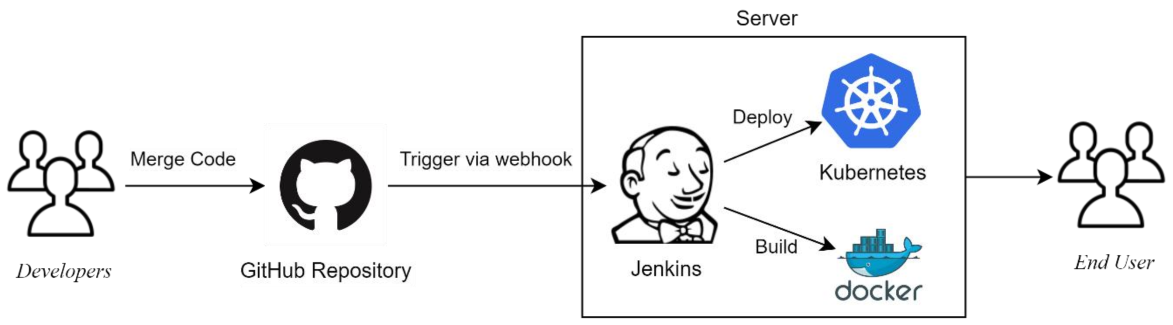


Fig 2. Relationship between Tools, Devices, Developers, and system

When the developer merges the code into the GitHub Repository, GitHub will access the webhook url that points to Jenkins Server so that it will trigger the pull process and do the docker image creation and deployment process on the Server. Jenkins will pull code based on the branch that was set during

*name of corresponding author

configuration, so that only certain branches can trigger Jenkins to work. The branches used to trigger Jenkins are the demo and master branches. This process can be seen in Figure 3.

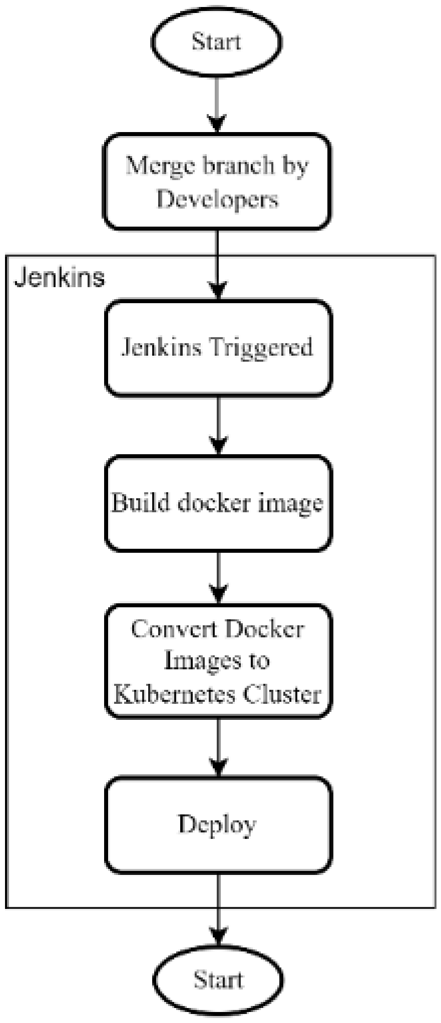

Start
Merge branch by Developers
Jenkins
Jenkins Triggered
Build docker image
Convert Docker Images to Kubernetes Cluster
Deploy
Start

Fig 3. Alur kerja jenkins

After Jenkins has finished pulling the code from the GitHub Repository, Jenkins will start the deployment process. The flow of the deployment process is: installing the packages needed to run the application, creating a Docker Image, converting the Docker Image - Open Container Initiative (OCI) into Kubernetes Image Cache and rollout Kubernetes deployment.

The deployment process can be monitored on the Jenkins interface. Developers can monitor whether the deployment process was successful or failed along with logs that can be used to find out where the error is when a deployment failure occurs. In this research, there is a conversion process from Docker Image to Kubernetes Image Cache. This is necessary because the Local Docker Daemon is not part of the Kubernetes Cluster and so Docker Images are not recognized (Canonical, n.d.).

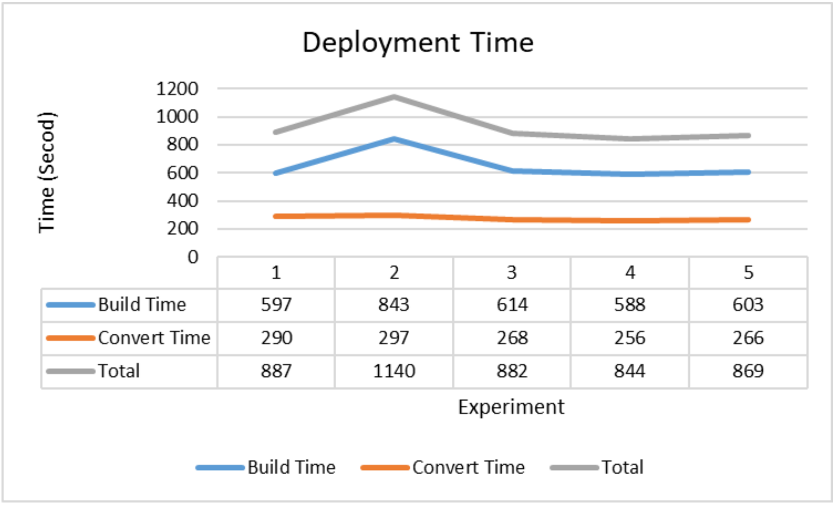
RESULT

In this section the author will explain the process and implementation results in accordance with the system implementation flow shown in Figure 1. The author conducted two tests, namely a performance test that test Docker's performance in terms of time to build the image and convert to the Kubernetes Image Cache which needed for deployment, and functional testing to find out whether CI/CD can run properly.

*name of corresponding author

Performance Testing

To measure deployment time, the authors conducted 5 experiments. Figure 3 shows a comparison of the build time required for Docker and conversion to Kubernetes Image Cache. Of the 5 trials conducted, the average deployment time was 924 seconds (15 minutes 25 seconds). Of that total time, creating a docker image takes more time than converting images. Build time takes an average of 649 seconds (10 minutes 49 seconds) while the image conversion process takes an average time of 275 seconds (4 minutes 35 seconds). This is because the build process installs application packages via the internet, so the more packages needed, the longer it takes and depends on internet speed.

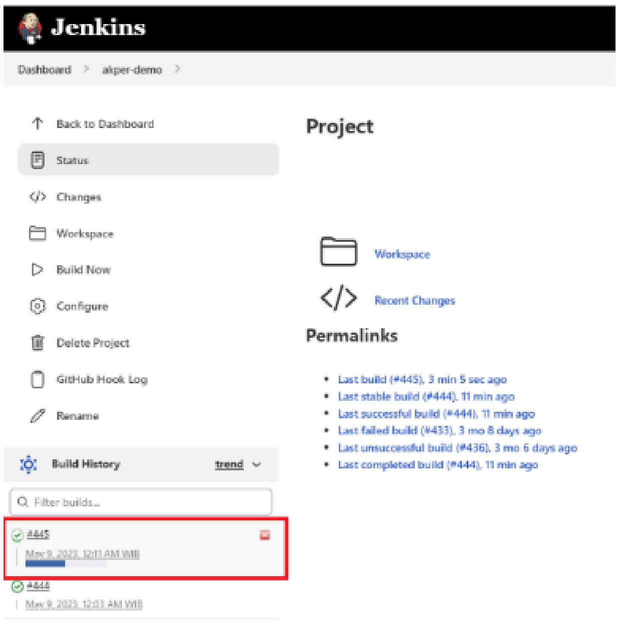


	1	2	3	4	5
Build Time	597	843	614	588	603
Convert Time	290	297	268	256	266
Total	887	1140	882	844	869

Fig 4. Deployment Time

Functional Testing

Functional testing is carried out by updating the code to the GitHub Repository until the Docker Image runs on the Kubernetes Cluster. After the developer has merged the code into the master or demo branches, Jenkins will be triggered from the previously configured GitHub Webhook to build by running the application deployment command as shown in Figure 5.



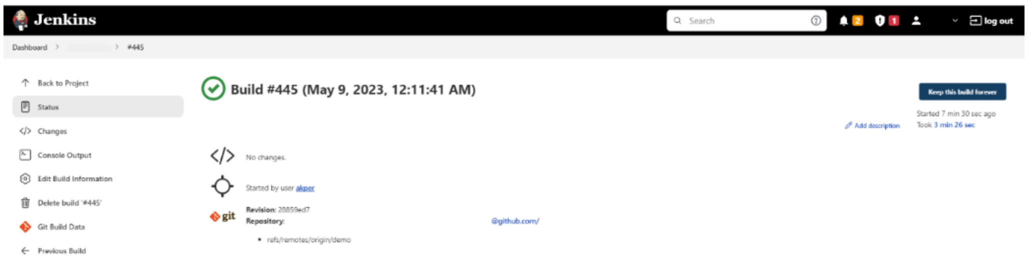
#445
May 9, 2023, 12:11 AM WIB
#444
May 9, 2023, 12:03 AM WIB

Permalinks

- Last build (#445), 3 min 5 sec ago
- Last stable build (#444), 11 min ago
- Last successful build (#444), 11 min ago
- Last failed build (#433), 3 mo 8 days ago
- Last unsuccessful build (#436), 3 mo 6 days ago
- Last completed build (#444), 11 min ago

Fig 5. Triggered Jenkins Build

*name of corresponding author



Build #445 (May 9, 2023, 12:11:41 AM)

Started 7 min 30 sec ago
Took 3 min 26 sec

No changes.

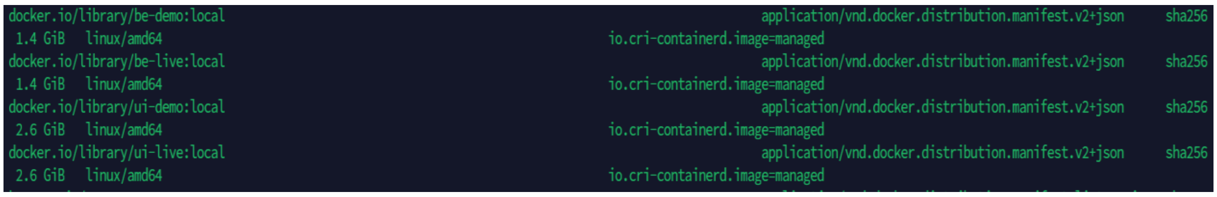
Started by user akper

git Revision: 28859ed7
Repository: [redacted]@github.com/

- refs/remotes/origin/demo

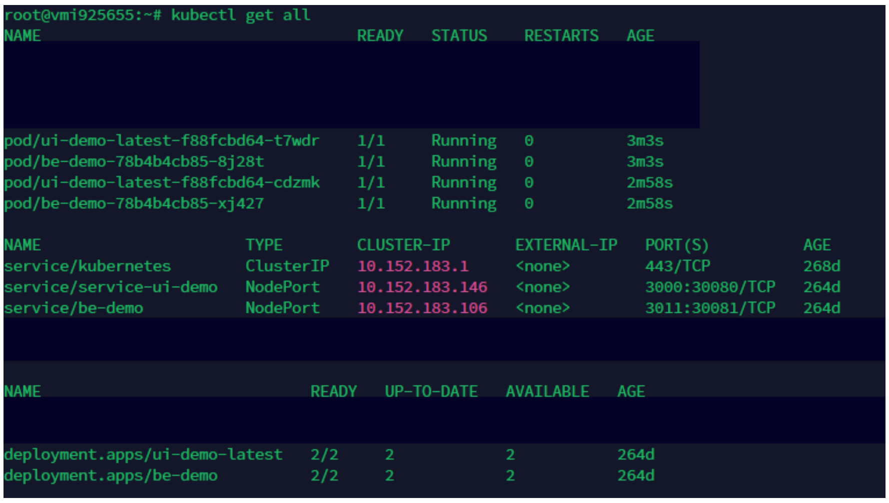
Fig 6. Build information that jenkins has done successfully

If Jenkins builds and deploys the application successfully, a description will appear as shown in Figure 6. To see whether the images for Kubernetes were created successfully, you can check with the microk8s crt images ls command as shown in Figure 7. In this image there are "be-demo" images which in the Jenkins build with the command Figure 5. Then to make sure whether the pods have run successfully, you can see the status of the pods in the RUNNING status. Figure 8 shows the status of pods and deployment in kubernetes. The figure shows 2 deployments with each running 2 pods that have a Running status and a service that exposes port 30080 30081.



```
docker.io/library/be-demo:local                                          application/vnd.docker.distribution.manifest.v2+json    sha256
 1.4 GiB   linux/amd64                                      io.cri-containerd.image=managed
docker.io/library/be-live:local                                          application/vnd.docker.distribution.manifest.v2+json    sha256
 1.4 GiB   linux/amd64                                      io.cri-containerd.image=managed
docker.io/library/ui-demo:local                                          application/vnd.docker.distribution.manifest.v2+json    sha256
 2.6 GiB   linux/amd64                                      io.cri-containerd.image=managed
docker.io/library/ui-live:local                                          application/vnd.docker.distribution.manifest.v2+json    sha256
 2.6 GiB   linux/amd64                                      io.cri-containerd.image=managed
```

Fig 7. List of Kubernetes Images



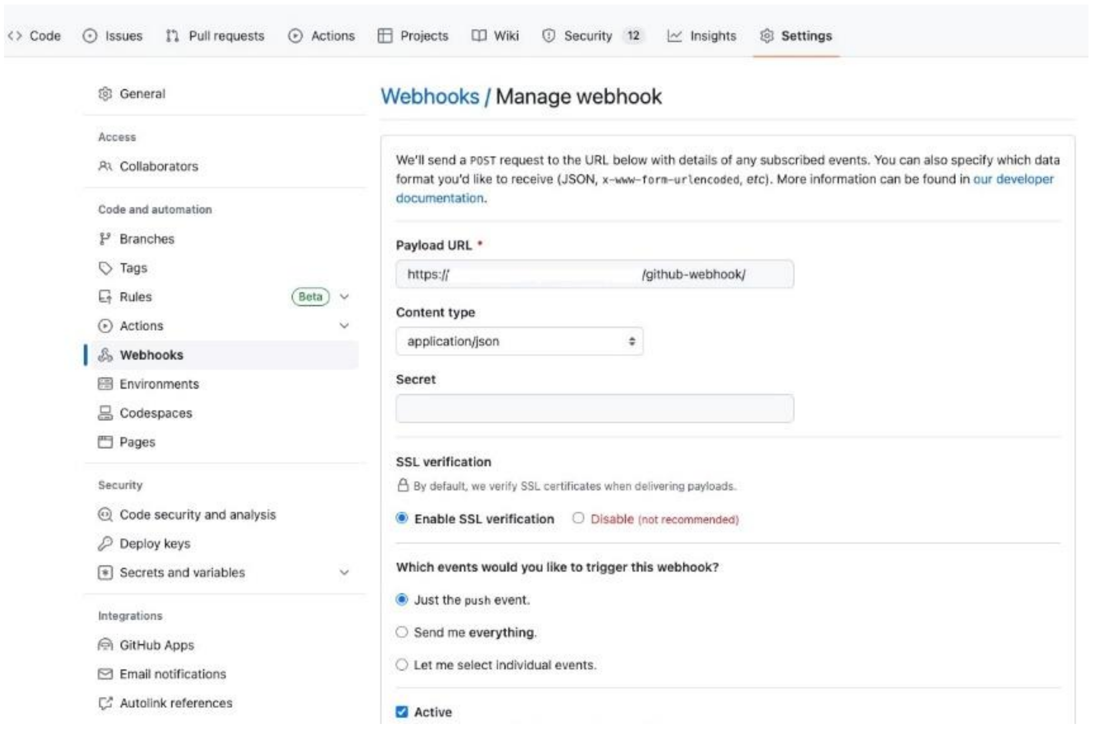
```
root@vmi925655:~# kubectl get all
NAME                                   READY   STATUS    RESTARTS   AGE
[redacted]
pod/ui-demo-latest-f88fcbd64-t7wdr     1/1     Running   0          3m3s
pod/be-demo-78b4b4cb85-8j28t           1/1     Running   0          3m3s
pod/ui-demo-latest-f88fcbd64-cdzmk     1/1     Running   0          2m58s
pod/be-demo-78b4b4cb85-xj427           1/1     Running   0          2m58s

NAME                        TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
service/kubernetes          ClusterIP   10.152.183.1     <none>        443/TCP          268d
service/service-ui-demo     NodePort    10.152.183.146   <none>        3000:30080/TCP   264d
service/be-demo             NodePort    10.152.183.106   <none>        3011:30081/TCP   264d
[redacted]

NAME                              READY   UP-TO-DATE   AVAILABLE   AGE
[redacted]
deployment.apps/ui-demo-latest    2/2     2            2           264d
deployment.apps/be-demo           2/2     2            2           264d
```

Fig 8. Pods dan Deployment Status of Kubernetes

DISCUSSIONS

In this chapter, we will discuss how implementation is configured. As explained in Figure 1, implementation begins with creating a GitHub Repository followed by creating a branch. Then a webhook configuration is required to integrate into Jenkins (GitHub, n.d.). The webhook configuration on the GitHub Repository can be seen in Figure 9 which is on the Webhooks menu then fill in the Jenkins url in the Payload URL Form.

*name of corresponding author



Webhooks / Manage webhook

We'll send a POST request to the URL below with details of any subscribed events. You can also specify which data format you'd like to receive (JSON, x-www-form-urlencoded, etc). More information can be found in our developer documentation.

Payload URL *
https:// /github-webhook/

Content type
application/json

Secret

SSL verification
By default, we verify SSL certificates when delivering payloads.
[✓ Enable SSL verification] Disable (not recommended)

Which events would you like to trigger this webhook?
[✓ Just the push event.]
Send me everything.
Let me select individual events.

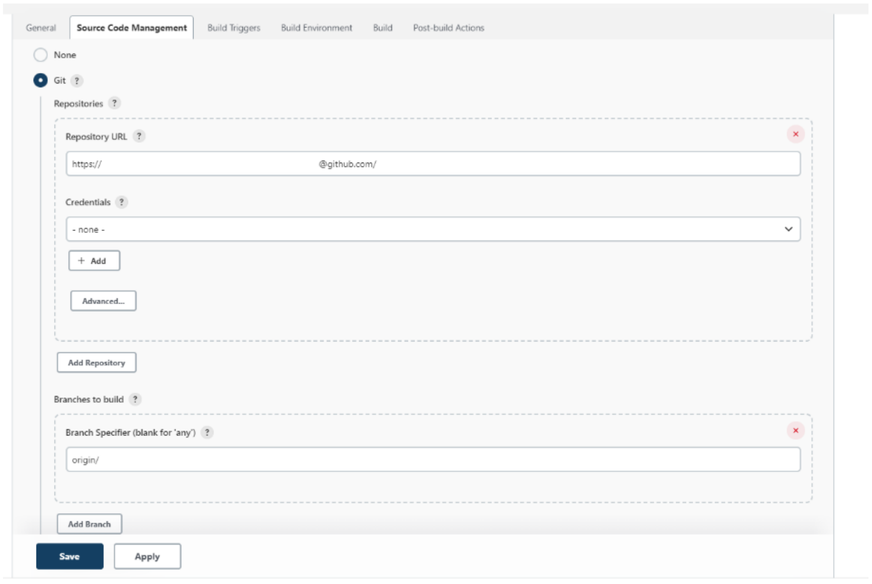
[✓ Active]

Figure 9. GitHub Repository Configuration

Jenkins Configuration

Configuring Jenkins is done so that it can be integrated with GitHub and carry out the deployment process. What needs to be configured here is the configuration of source code management and build which contains internal commands for deployment. The Source Code Management configuration is shown in Figure 10. In the Source Code Management configuration, specify the GitHub Repository URL, and which branch will be integrated. Meanwhile, Figure 10 shows the Build configuration in which there are several commands for deploying.

Jenkins use 2 branches master and staging branches for deployment. The development branch is not used because the branch is specifically for local development. so when you want to deploy, the programmer has to merge the code into the master and staging branches.



None [✓ Git]

Repositories

Repository URL
https:// @github.com/

Credentials
- none -

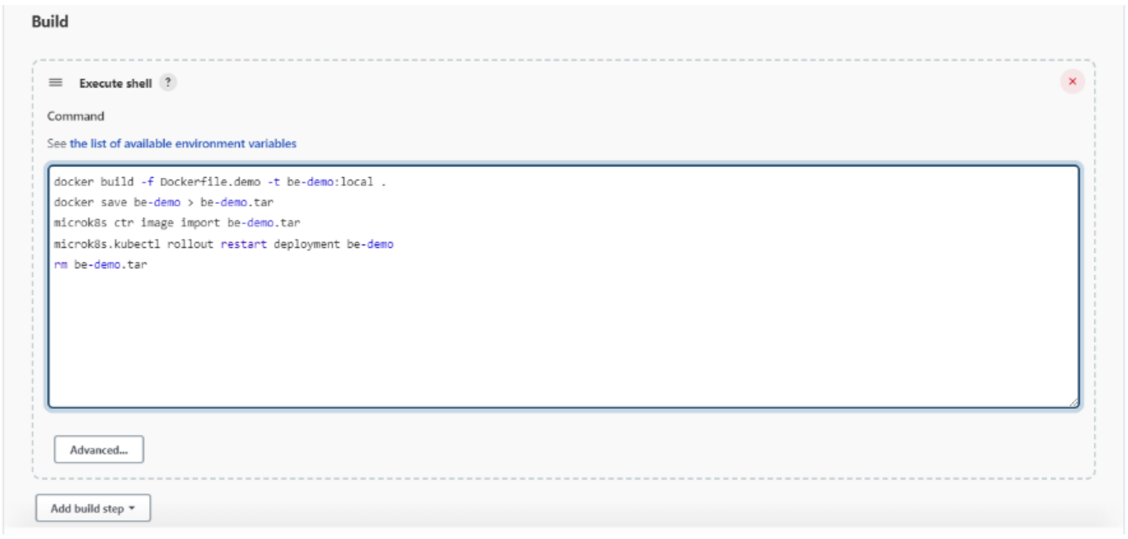
Branches to build

Branch Specifier (blank for 'any')
origin/

Fig 9. Source Code Management Configuration

*name of corresponding author

e-ISSN : 2541-2019
p-ISSN : 2541-044X

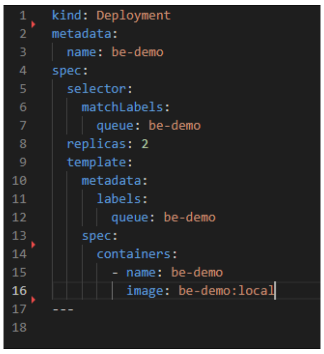


```
docker build -f Dockerfile.demo -t be-demo:local .
docker save be-demo > be-demo.tar
microk8s ctr image import be-demo.tar
microk8s.kubectl rollout restart deployment be-demo
rm be-demo.tar
```

Fig 10. Build Command Configuration

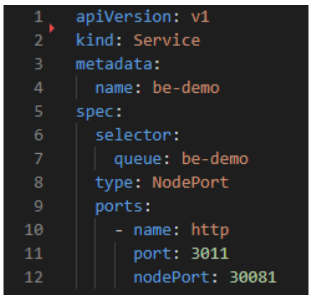
Kubernetes Konfiguration

Kubernetes is used to run pods, a container that contains applications that were previously built using Docker. In this study, we use 2 pods to maintain application stability, especially during deployment so as to reduce downtime by using the rollout deployment method. (Kubernetes, n.d.). There are 2 configurations performed, namely Deployments and Service configurations. Deployments configuration is used to declare how updates are performed for pods while Services are used to expose network applications so that other services can access them. The Deployment configuration is shown in Figure 11.



```
kind: Deployment
metadata:
  name: be-demo
spec:
  selector:
    matchLabels:
      queue: be-demo
  replicas: 2
  template:
    metadata:
      labels:
        queue: be-demo
    spec:
      containers:
        - name: be-demo
          image: be-demo:local
---
```

Fig 11. Kubernetes Deployment Configuration



```
1   apiVersion: v1
2   kind: Service
3   metadata:
4     name: be-demo
5   spec:
6     selector:
7       queue: be-demo
8     type: NodePort
9     ports:
10      - name: http
11        port: 3011
12        nodePort: 30081
```

Fig 12. Kubernetes Service Configuration

The port configuration that pods use to communicate with other services is shown in Figure 12 lines 11 and 12. This configuration will open port 3011 in the application to address 30081.

*name of corresponding author

CONCLUSION

From the test results it can be concluded that the Continous Delivery implementation using Kubernetes with the Local Docker image is successful and can run well. With this type of implementation we do not need additional cost for using Online Repositories such as Docker Hub as a container registry. This is proven from the test results where the docker image creation process, the conversion of the docker image to the kubernetes image was successfully carried out. However, this implementation has drawbacks regarding deployment time. The average time required to create a docker image is 649 seconds (10 minutes 49 seconds) and image conversion process which takes an average time of 275 seconds (4 minutes 35 seconds). Build process need more time to installs application packages via the internet, so that the more packages needed, the longer it takes and depends on internet speed and specification of computer hardware. If at the same time doing deployment, server may will have high load and result in a crash. Research can be developed further by researching techniques to shorten build time, minimize resource utilitation and minimize time conversion from docker images to Kubernetes Image Cache.

ACKNOWLEDGMENT (*optional*)

Thank you to Eko Andri Subarnanto, S.Kom. as the Project Manager and related parties in working on the application project which has allowed the author to complete this research.

REFERENCES

Alpery, A., & Ridha, M. A. F. (2021). Implementasi CI/CD dalam Pengembangan Aplikasi Web Menggunakan Docker dan Jenkins. *9th Applied Business and Engineering Conference*, *9*, 287–296.

Canonical, L. (n.d.). *How to use a local registry*. How to Use a Local Registry. Retrieved May 8, 2023, from https://microk8s.io/docs/registry-images

Contabo. (n.d.). *Contabo*. Contabo Cloud VPS & Dedicated Servers for a Price You'll Love. Retrieved April 27, 2023, from https://contabo.com/en

Danur Wijayanto, Arizona Firdonsyah, & Faisal Dharma Adhinata. (2021). Implementasi Continous Integration/Continous Delivery Menggunakan Process Manager 2 (Studi Kasus: SIAKAD Akademi Keperawatan Bina Insan). *Teknika*, *10*(3), 181–188. https://doi.org/10.34148/teknika.v10i3.400

Fadil, I., Saeppani, A., Guntara, A., & Mahardika, F. (2020). Distributing Parallel Virtual Image Application using Continuous Integrity/Continuous Delivery Based on Cloud Infrastructure. *2020 8th International Conference on Cyber and IT Service Management (CITSM)*, 1–4. https://doi.org/10.1109/CITSM50537.2020.9268860

Freeman, E. (2019). *Devops for dummies* (1st ed). John Wiley and Sons.

Garg, S., & Garg, S. (2019). Automated Cloud Infrastructure, Continuous Integration and Continuous Delivery using Docker with Robust Container Security. *2019 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR)*, 467–470. https://doi.org/10.1109/MIPR.2019.00094

GitHub, I. (n.d.). *About webhooks*. About Webhooks: Learn the Basics of How Webhooks Work to Help You Build and Set up Integrations.

Jeffery, A., Howard, H., & Mortier, R. (2021). Rearchitecting Kubernetes for the Edge. *Proceedings of the 4th International Workshop on Edge Systems, Analytics and Networking*, 7–12. https://doi.org/10.1145/3434770.3459730

Kim, G., Debois, P., Willis, J., Humble, J., & Allspaw, J. (2016). *The DevOps handbook: How to create world-class agility, reliability, & security in technology organizations* (First edition). IT Revolution Press, LLC.

Kubernetes, I. (n.d.). *Kubernetes Deployment*. Kubernetes Deployment.

Leite, L., Rocha, C., Kon, F., Milojicic, D., & Meirelles, P. (2020). A Survey of DevOps Concepts and Challenges. *ACM Computing Surveys*, *52*(6), 1–35. https://doi.org/10.1145/3359981

*name of corresponding author

Luhana, K. K., Schindler, C., & Slany, W. (2018). Streamlining mobile app deployment with Jenkins and Fastlane in the case of Catrobat's pocket code. *2018 IEEE International Conference on Innovative Research and Development (ICIRD)*, 1–6. https://doi.org/10.1109/ICIRD.2018.8376296

Mahboob, J., & Coffman, J. (2021). A Kubernetes CI/CD Pipeline with Asylo as a Trusted Execution Environment Abstraction Framework. *2021 IEEE 11th Annual Computing and Communication Workshop and Conference (CCWC)*, 0529–0535. https://doi.org/10.1109/CCWC51732.2021.9376148

R, N. D. & Mohana. (2022). Jenkins Pipelines: A Novel Approach to Machine Learning Operations (MLOps). *2022 International Conference on Edge Computing and Applications (ICECAA)*, 1292–1297. https://doi.org/10.1109/ICECAA55415.2022.9936252

Rodriguez, M. A., & Buyya, R. (2019). Container-based cluster orchestration systems: A taxonomy and future directions. *Software: Practice and Experience*, *49*(5), 698–719. https://doi.org/10.1002/spe.2660

Shahin, M., Ali Babar, M., & Zhu, L. (2017). Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices. *IEEE Access*, *5*, 3909–3943. https://doi.org/10.1109/ACCESS.2017.2685629

Singh, C., Gaba, N. S., Kaur, M., & Kaur, B. (2019). Comparison of Different CI/CD Tools Integrated with Cloud Platform. *2019 9th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*, 7–12. https://doi.org/10.1109/CONFLUENCE.2019.8776985

Yin, Z., Liu, J., Chen, B., & Chen, C. (2021). A Delivery Robot Cloud Platform Based on Microservice. *Journal of Robotics*, *2021*, 1–10. https://doi.org/10.1155/2021/6656912

*name of corresponding author