DevOps Implementation with Enterprise On-Premise Infrastructure

Muhammad Alvin*, Rizal Fathoni Aji
Fakultas Ilmu Komputer, Magister Teknologi Informasi, Universitas Indonesia, DKI Jakarta, Indonesia
Email: *muhammad.alvin11@ui.ac.id, rizal@cs.ui.ac.id
Email Corresponding Author: mhdalvin@gmail.com

Abstract—Digital transformation is essential in today's VUCA era (volatile, uncertain, complex, ambiguous). As the primary driver of digital transformation, the software has widely adopted agile concepts with agile software development. Its short and iterative cycle makes it adaptable to change. Instead of producing significant changes simultaneously, the development team produces small but frequent changes. The operations team is overwhelmed with implementing these changes, and bottlenecks arise. DevOps comes to remove these bottlenecks and allow the development and operations teams to work together to release the software to users quickly. As part of the digital transformation, PT Logistik Pangan has started to implement DevOps with on-premise infrastructure, which is yet to be optimal. This qualitative research aims to understand the steps taken by the company for implementing DevOps with on-premise infrastructure using GitLab and offers suggestions on how to maximize its implementation. The results show that implementing DevOps with on-premises infrastructure requires additional work. The supporting infrastructures for DevOps. Implementation was done incrementally, by adopting DevOps practices one by one at a time. Version control (also known as source control or source code management) is implemented by using GitLab, and requires self-managed GitLab as supporting infrastructure. Continuous integration and continuous delivery are implemented using GitLab CI/CD, and requires GitLab Runner as supporting infrastructure. Besides the DevOps practices, the company also implement container technology by using Docker that is upgraded to Docker Swarm later, and requires local Docker Registry as supporting infrastructure. All the supporting infrastructures are installed on-premise in the company's data center. It includes servers, storage, and networking that must be managed separately. Some improvements are ensuring mindset and culture have been adjusted, implementing other principles alongside automation, and should continue beyond these stages: Code, Build, Test, Release, and Deploy. This research has limitation of using GitLab products only. Future research can use other DevOps tools or combine GitLab products with other products.

Keywords: Development; DevOPS; Infrastructure; Operations; On-Premise

1. INTRODUCTION

Digital transformation is a must in today's VUCA era (volatile, uncertain, complex, ambiguous) [1], which allows the enterprise to shift in a more agile direction. The highly competitive and fast-moving business environment requires quick reactions to changing market conditions, so agility is a promising option to meet this challenge [2]. Digital transformation is driven by a changing role for technology in the enterprise, which can now take on more than just support functions [3].

The main driving force behind digital transformation is a software [4], which has widely adopted agile concepts in software development [5]. Compared to the waterfall approach, the agile approach can deal with uncertainties and produce a working product in a relatively short time without sacrificing quality. The short and iterative development cycle of agile software makes this approach easy to adapt to changes [6]. The development team updates the software more and more frequently to introduce new features, but the operations team's capabilities are limited to implement these changes as often as possible. DevOps comes to enable development and operations teams to work together to deliver software quickly to users [7]. DevOps is recognized as the most effective way to improve collaboration between development and operations team [8].

PT Logistik Pangan is a state-owned enterprise (SOE) undergoing digital transformation. In parallel with this transformation, the company has recently implemented agile and DevOps with GitLab. However, the company's decision not to use cloud services requires that the implementation of DevOps be done with a fully self-managed on-premises infrastructure. Based on the data obtained, not all application software projects have implemented DevOps. The level of implementation of DevOps practices also varies. There are application projects that implement only basic practices like version control, but there are also application projects that have implemented more sophisticated practices like continuous integration and continuous delivery.

The company is aware of the problem and is trying to maximize the DevOps implementation it has started. Through these research questions: (1) How is the implementation of DevOps with a fully self-managed on-premises infrastructure? and (2) What can be done to maximize DevOps implementation?, this research aims to understand what the company has been doing in implementing DevOps and to provide suggestions so that the company can maximize the implementation of DevOps, which has been started with the limitations of a fully self-managed on-premise infrastructure in the company's data center.

The scope of this research is limited to one case study company, PT Logistik Pangan, that implemented DevOps using GitLab. Although software development is not the core business of the company, the daily activities of the company are inseparable from applications. As part of good corporate governance (GCG), the company must also comply with various applicable regulations, one of which is not to place specific data outside of the infrastructure managed by the company.
Since it first appeared in 2011, researches related to DevOps has grown in popularity along with its broader implementation in the industrial world. In Scopus database, there are almost 1,800 documents about DevOps, and there are about 200-300 documents per year related DevOps in the last five years. Refining the search by combining DevOps with GitLab and infrastructure gives 3 documents. First document, a research by Sharif, Janto, and Lueckemeyer in 2020, proposes architecture using containerization that allow high performance computing (HPC) application to provide self-reliant, testable, and reproducible environment using GitLab Runner to handle complex integration process [9]. Second document, a research by Tripathi, Monroe, Hanby, and Robinson in 2020, discusses software-defined infrastructures (SDI) and highlight the importance to collaboration of open-source tooling and self-hosted community edition of GitLab [10]. Third document, a research by Stillwell and Coutinho in 2015, presents infrastructure being created to support the integration effort of a multi-partner research projects intended to bring the power of heterogeneous resources to the cloud [11]. Based on this fact, there are no researches about DevOps implementation with a fully self-managed on-premises infrastructure using GitLab, especially in SOEs.

In theory, this research contributes to the knowledge base related to DevOps and fills the research gaps of DevOps implementation in a fully self-managed on-premises infrastructure using GitLab. In practice, this research helps the company maximize its DevOps implementation. This research can also be a reference for other companies that share similar DevOps implementation characteristics, namely the use of on-premises infrastructure, where all layers (networking, storage, servers, virtualization, operating systems, middleware, runtime, data, and applications) are self-managed, in contrast to cloud infrastructure, where some layers are self-managed and others are managed by third party service providers (vendors) as shown in Figure 1.

![Figure 1. On-Premise vs Cloud (Infrastructure/Platform/Software as a Service)](image)

## 2. RESEARCH METHODOLOGY

### 2.1 Research Flow

This research consists of eight steps, as shown in Figure 2: problem identification, literature review, research design, instrument development, data collection, data analysis, suggestion for improvement, and conclusion.

![Figure 2. Research Flow](image)
Development Section is assisted by programmers, as well as Head of the Application Operations Section which is assisted by system administrators and database administrators. The selection of the two speakers was based on the fact that DevOps is a combination of the development team and the operations team, so data from the two team is needed. The section head as a structural officer is chosen because he has more accountable authority that responsible for the success and achievement of a job, in contrast to the programmers, system administrators, and database administrators who has more responsible authority that carries out and completes the work.

Table 1. Interview Questions as Research Instruments

<table>
<thead>
<tr>
<th>Category</th>
<th>Question</th>
</tr>
</thead>
<tbody>
<tr>
<td>DevOps adoption</td>
<td>1. What are the conditions before adopting DevOps?</td>
</tr>
<tr>
<td></td>
<td>2. What are the conditions after adopting DevOps?</td>
</tr>
<tr>
<td></td>
<td>3. What is the company’s journey in adopting DevOps?</td>
</tr>
<tr>
<td>Culture</td>
<td>4. What cultures have been implemented along with DevOps adoption?</td>
</tr>
<tr>
<td>Practices</td>
<td>5. What practices have been implemented along with DevOps adoption?</td>
</tr>
<tr>
<td>Tools</td>
<td>6. What technologies have been implemented along with DevOps adoption?</td>
</tr>
<tr>
<td>On-Premise</td>
<td>7. What are the specific barriers and challenges faced from using on-premises infrastructure when adopting DevOps?</td>
</tr>
</tbody>
</table>

The data that has been collected by the interview method is then analyzed and discussed to answer research questions. Suggestions for improvement are compiled by comparing what the company has done with the various theories contained in the literature. Finally, conclusions are explanations that answer research questions, drawn from the results of analysis and discussions that have been carried out previously.

2.2 DevOps

DevOps principles are often known as CALMS, which is an acronym for Culture, Automation, Lean, Measurement (also known as Monitoring, Metric), and Sharing. The principle of culture is related to implementing a culture of collaboration between the development team and the operations team. Automation principles are related to automating routine processes that are usually carried out. The lean principle related to optimizing the process becomes simpler. The measurement principle is related to transparency where both teams can see each other’s measurements that are usually carried out by each other. The principle of sharing is related to how personnel from both teams share their involvement in working on the same software [12]. The principles of culture and automation are universally considered the core of DevOps, measurement and sharing are considered important by most DevOps practitioners, and lean is considered part of DevOps by only a small percentage of DevOps practitioners [13].

![Figure 3. DevOps Life Cycle](image)

Despite the variations, the DevOps life cycle is generally represented in Figure 3, which consists of Plan, Code, Build, Test, Release, Deploy, Operate, and Monitor. The “plan” phase identifies and defines software requirements. The “code” phase develops the source code. The “build” phase converts the source code into the application. The “test” phase consists of a series of automated tests. The “release” phase prepares the application for users. The “deploy” phase puts the application in a production environment where users can use it. The “operate” phase ensures the availability and capacity of the application for use. The “monitor” phase monitors and provides feedback based on the monitoring results [14], [15].

The three phases of DevOps transformation are mindset and culture change, implementation of practices and tools, optimization, and adaptation. The first phase focuses on the development of individuals with an emphasis on teamwork and skill enhancement with appropriate training. The second phase focuses on implementing DevOps practices to increase the speed of conversion from idea to software, the accuracy of the software developed, and the quality of the software developed. The third phase is optimization and adaptation to the implemented DevOps [16].

2.3 Version Control (also known as Source Control or Source Code Management)

Version control is a practice that is essential for managing complex projects, even non-software-related projects [17]. In software projects, version control is also referred to source code or source code management. As one of
the most critical tools in software development in today’s modern era [18], version control can also provide helpful collaboration tools to share and consolidate file changes with other users [19].

Figure 4. Client-Server vs Distributed Version Control

In general, there are two types of version control, client-server and distributed, as shown in Figure 4. Client-server version control has only one repository on the server, e.g. Subversion. In contrast, distributed version control has many repositories but is usually designated as one shared repository, e.g. Git and other Git-based products like GitHub, GitLab, and BitBucket.

2.4 Continuous Integration

Continuous integration (CI) is a practice in software development that regularly integrates and tests source code to detect bugs in the early stages of development, which is usually beneficial for improving the quality of the software and the productivity of the development team [20]. CI is usually performed by sending (pushing) source code to a shared repository, followed by an automated build and test process [21].

2.5 Continuous Delivery

Continuous delivery (CD) is a practice in software development where developed software products can be used at any time [22]. As one of the DevOps practices, CDs enable organizations that implement them to release new features and products quickly [23].

3. RESULT AND DISCUSSION

As shown in research flow, there are eight steps in this research. The first four steps (problem identification, literature review, research design, and instrument development) has been discussed in previous sections about introduction and research methodology. Fifth step about data collection is performed by using interview, addressed to two representatives from development team and operations team, with questions shown in Table 1. Sixth step about data analysis is performed to the result of data collection, and will be discussed in this section. Seventh step about suggestion for improvement also be part of this section.

3.1 Condition Before DevOps Implementation

As shown in Figure 5, none of the apps use version control. The latest source code is located on the production server for applications that do not require a compilation process, such as web applications using PHP technology. For applications that require a compilation process, such as desktop-based applications with client-server using PowerBuilder technology, the most up-to-date source code resides in the programmer. The production environment is updated directly by programmers who has access to it. It is common for programmers to update the source code directly on the production server.

Since there are no version control, collaboration is a nightmare. If more than one development team members need to update the same file, they must communicate among team members so that one team member must wait for another team member completing his task. Forgetting to do so will results in changes made by one team member is overwritten by another team member.

Figure 5. Condition Before DevOps Implementation
At that time, the operations team was only focused on providing infrastructure in the form of servers, including operating systems, storage, and networking. The organizational structure also fits with this condition, where the operations team consists of two organizational units (called sections): user services section and infrastructure section. The development team focuses on managing the server's contents, from libraries, runtimes, dependencies, components, and services needed to run the application to the application itself.

3.2 Condition After Implementation DevOps

As shown in Figure 6, the company has already implemented version control. The initial phase uses Subversion, which then transitions to a self-managed version of GitLab installed on-premises in the company's data center.

![Figure 6. Condition After DevOps Implementation](image)

The production environment, which used to be updated directly by the development team, became updated by the operations team manually, and later by continuous delivery automatically. This also corresponds to the changing organizational structure, where the operations team now consists of three organizational units: user services section, infrastructure section, and (new) application operations section. The development team no longer manages the production environment but only prepares applications ready for release to the production environment. The operations team performs the deployment process to the production environment.

What is also new after the DevOps implementation is that several applications started to use container technology. However, not all applications migrate to container technology, especially legacy applications that are more effective when deployed directly to virtual machines than containers.

There are three phases of DevOps transformation, but the data obtained does not show how far the changes in mindset and culture have been made. In order to maximize DevOps implementation, the company must ensure that the required mindset and culture have been developed, particularly the culture of collaboration. Competency improvement with appropriate training is also needed so the company's employees can perform well in the next phase, namely implementing practices and tools.

There are five DevOps principles, abbreviated as CALMS (Culture, Automation, Lean, Measurement, Sharing), whereby the data obtained shows that the company is still focused on applying the automation principle. To maximize the benefits of DevOps, the company must also apply other principles, namely culture, lean, measurement, and sharing. An important principle is the culture of collaboration, which is also mentioned in the first phase of the DevOps transformation.

There are eight phases in the DevOps cycle, but the data obtained shows that the company is still implementing until the deploy phase. The company should continue beyond this stage and at the next stage, such as operate and monitor, to maximize the benefits of DevOps. Applicable activities are continuous monitoring, which monitors the performance, capacity, availability, and security of running applications, and continuous feedback, which provides feedback to the development team and operations team based on the results of the monitoring performed.

3.3 DevOps Implementation Journey

The company's journey in implementing DevOps is grouped based on the step-by-step implementation of DevOps practices with the chosen tools, as shown in Table 2. Gradually, the company implemented the practice of version control, continuous integration, and continuous delivery.

<table>
<thead>
<tr>
<th>DevOps Practices</th>
<th>DevOps Tools</th>
<th>On-Premise Infrastructure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version Control</td>
<td>Subversion</td>
<td>Visual SVN Server</td>
</tr>
<tr>
<td></td>
<td>GitLab</td>
<td>GitLab (self-managed)</td>
</tr>
<tr>
<td>Continuous Integration</td>
<td>GitLab CI/CD</td>
<td>GitLab Runner</td>
</tr>
</tbody>
</table>
DevOps Practices | DevOps Tools | On-Premise Infrastructure
---|---|---
Continuous Delivery | GitLab CI/CD | GitLab Runner

The company also began implementing container technology using Docker, which was later upgraded to a Docker cluster using Docker Swarm. One of the goals of the upgrade is to break the barriers of the on-premises infrastructure is being used.

### 3.3.1 Version Control with Subversion

Although DevOps has been implemented for about the last 2-3 years (2019-2020), the journey started about 4-5 years ago (2017-2018). The first step is to ensure that the application’s source code is managed in version control. Development team are instructed to store the latest source code to version control repository.

As shown in Figure 7, the company uses Subversion (SVN), a client-server version control. After reviewing some products, the company decides to use Visual SVN Server for server component and Tortoise SVN for client component. On-premise infrastructure needed is a Windows server to be installed with Visual SVN Server that provides SVN repository. For network configuration, that server must be reachable by development team and any deployment servers.

![Visual SVN Server, Installed On-Premise in Company’s Data Center](image)

**Figure 7.** Visual SVN Server, Installed On-Premise in Company’s Data Center

**Challenge 1:** The process of storing latest source code to SVN repository (using commit feature) is not effective. Development team who still have access to the production environment prefer to update the production environment using the source code on their machine. As the result, the source code in the SVN repository is not updated regularly.

To address the challenge, the company was doing organizational restructuring. The operations team added a new organizational unit, namely application operations section. Accordingly, updates to the production environment are assigned to the operations team by fetching the source code from the SVN repository. This method indirectly forces the development team to store the source code to the SVN repository; otherwise, the changes cannot be deployed to the production environment.

**Challenge 2:** Although the SVN repository already contains the latest source code, sometimes the source code is not ready to be released. No control or validation of the source code submitted to the SVN repository is the source code that works properly.

To address the challenge, the source code is forked in the SVN repository. There is a separation between the latest source code (latest development) and the release source code. Periodically, the release source code is updated from the latest source code using the merge feature in SVN; after ensuring that the latest source code is working properly.

**Challenge 3:** The merging process performed by development team has no control mechanisms such as merge requests, verification of merge requests, and merge approvals. SVN can track and manage source code changes through commits or merges, but the previously mentioned control mechanism cannot be performed with this tool.

### 3.3.2 Version Control with GitLab

The previous challenge can’t be solved by using existing SVN, so the company needs to use another tools. After evaluating several products, the company chose GitLab to replace SVN. GitLab is similar to other products like GitHub and Bitbucket, that based on Git version control and enriched with additional features for easy collaboration. New applications are directed to using GitLab. Existing applications that already operational, some applications are migrated to GitLab, and others remain in SVN.

As shown in Figure 8, the company uses GitLab. There are two version of GitLab available, software as-a-service (SaaS) version that is accessible from https://gitlab.com and self-managed version that must be installed on company’s infrastructure. The company choose self-managed version as it is not ready to adopt cloud services. On-premise infrastructure needed is a Linux (Ubuntu) server to be installed with GitLab that provides Git
repository. For network configuration, the server must be reachable by development team and any deployment servers.

![Diagram of a network setup with a GitLab repository, a load balancer, and servers connected to it.]

**Figure 8.** Self-Managed GitLab, Installed On-Premise in Company’s Data Center

The previous challenge is now can be solved by GitLab. Previously in SVN, the merge process is done directly in source code. Now in GitLab, development team submit merge request through GitLab web interface. Any reviewers then review and verify that merge request, and decide whether it is approved to be merged or need some changes before merged. If it is approved, GitLab automatically merge the source code.

Apart from the new GitLab tool, the operations team work pattern has stayed the same. The deployment process from SVN repository and GitLab repository is no different; both take the latest source code from a specific branch. While SVN uses the update feature, GitLab uses the pull feature. After receiving the latest source code, the deployment process needs to continue with several other processes performed manually by the operations team.

But there are changes in the way development team work. The process of storing source code to repository, previously one step with SVN (commit), is now two steps with GitLab (commit and push). This is because SVN is client-server while GitLab (and any other Git based version control) is distributed. SVN's commit feature sends the changes directly to a central repository, while Git's commit feature only sends the changes to local repository in development team’s working machine. A push feature will sends the changes in local repository to a central repository. During the early transition period, development team always follow commit process with push process. This is inherited from the habit of using SVN, which commit process means that the work products are sent to the central repository. For Git, the commit process can be repeated multiple times as needed, and the push process is needed when the work products will be shared to others.

### 3.3.3 Container with Docker

So far, applications with a large number of users are only client-server desktop-based information systems for managing logistics transactions (called SI Logistik) and accounting transactions (called SI Akuntansi). Both applications are used in all branch offices, with an estimated of 1,000 users. The company has not started to use a load balancer, so the application is deployed across multiple servers and ports, divided by region. For example, a user from the North Sumatera region connects to an application on server A port 1000, a user from the South Sumatera region connects to an application on server A port 2000, and a user from the Java region connects to an application on server B port 1000.

Then comes the web-based information system for managing employee performance (called SI Kinerja), which will be accessed by all employees of more than 4,500 people. The project failed because the resulting application didn’t meet the required non-functional requirements, especially performance. After adding resources, especially processors and memory, the application still didn’t meet the expected availability and capacity goals. The application is so slow that is difficult to use. Many offices work overtime just to use the application. The application was evaluated and decided not to be use anymore, but a replacement application must be provided.

**Challenge 4:** Various discussions concluded that a server with huge resources is not enough. Sharing the load across multiple servers and ports like before is problematic because this application is a web application where users can come from anywhere, unlike the previous application which was a client-server where the client can be configured to connect to specific pre-determined servers and ports. Load balancing with a load balancer by deploying multiple virtual servers running applications is also impossible as the company still needs to own and implement load balancer technology.

To address the challenge, the company decided to implement container technology with Docker. By using Docker, the application is packaged as Docker image that can be run easily later. Application that will serve high number of users, the focus is no longer increasing the server resources (called scale-up or vertical scaling) but increasing the number of servers (called scale-out or horizontal scaling). Using container technology is very easy to increase or decrease the number of application servers.

**Challenge 5:** Manual work is increasing, especially in development team. Previously, it is sufficient to store the source code to the GitLab repository, now it must involve building a Docker image and sending it to the operations team.
3.3.4 Continuous Integration with GitLab CI/CD

Docker image creation is now automatic by using the CI/CD feature in GitLab. Changes to the source code that are pushed to the repository automatically trigger the creation of this image.

As shown in Figure 9, the company use GitLab CI/CD. It requires GitLab Runner, that will execute CI/CD jobs, i.e. job build_apps to convert the source code into an application, job build_image to convert the application into a Docker pack image, job release to push the image to docker registry. Jobs definition is placed in CI/CD script in the form of plain text file, called .gitlab-ci.yml, that part of source code and pushed to repository. On-premise infrastructure needed is one or more Linux servers to be installed with GitLab Runner. Besides GitLab Runner, sometimes it is needed to install additional software, depends on the configuration of GitLab Runner is being used, i.e. GitLab Runner with Docker executor needs Docker to be installed [24]. A complete guide can be found in the official documentation. For network configuration, the servers must be able to contact self-managed GitLab that has been installed previously. The server must be connected to internet or another resource needed for continuous integration purpose. The server must be able to contact deployment server for continuous delivery purpose.

Continuous integration is not only applicable to applications that already use container technology. Other applications, especially legacy applications that do not use container technology, can also implement continuous integration. The most important thing is that the source code stored in GitLab repository has CI/CD script with the appropriate jobs.

The results of job execution in continuous integration are typically artifacts, which is an archive containing files and/or folders of CI/CD job execution results [25]. Applications that do not use containers will deploy these artifacts during continuous delivery. However, these artifacts must first be packaged into a Docker image for applications that use containers, that will be sent to the Docker registry and deployed by continuous deployment.

![Figure 9. GitLab Runner, Executing Continuous Integration Job](image)

3.3.5 Continuous Delivery with GitLab CI/CD

Continuous delivery aims to deploy applications in the appropriate environment. Applications can be artifacts or Docker images resulting from the continuous integration process. The environment can be production, staging, testing, or other environments as needed.

As shown in Figure 10, no additional supporting infrastructures are needed. The existing GitLab Runner will be used for continuous integration and continuous delivery. Continuous delivery is the process of deploying artifacts to deployment environment. There are at least three deployment environments: development, testing, and production.

![Figure 10. GitLab Runner, Executing Continuous Delivery Job](image)

As shown in Table 3, the development environment uses source code from master branch, which now changes its name to the main branch [25]. The testing environment uses source code from testing branch, which results from a merge of master/main branch. The production environment uses source code from production branch, which results from a merge of testing branch.
The branch separation scheme offers several advantages. Application development activities continue as usual, although there are another running activities such as testing or release to production. Source code changes generated from development activities are sent to the master/main branch. Applications testing in the test environment will not be disturbed because they are in a different branch, namely testing. Likewise, the live application in the production environment is not disturbed either since it is located in a different branch, namely production.

<table>
<thead>
<tr>
<th>Table 3. Deployment Environment</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Deployment Environment</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Development</td>
</tr>
<tr>
<td>Testing</td>
</tr>
<tr>
<td>Production</td>
</tr>
</tbody>
</table>

Branch production is the result of merging from the testing branch, and branch testing is the result of merging from the master/main branch. This provided an advantage in the form of applications to be deployed for production has been approved as tested applications. The master/main branch should not be merged directly into the production branch without passing through the testing branch.

Application updates in the testing and production environments are performed with a control mechanism. The process starts with submitting a merge request to the appropriate branch and then proceeds with approving the merge request. The programmers (development team) submits merge requests to the testing branch after the completion of development activities and is approved by quality assurance (development team). Merge requests to the production branch are performed by quality assurance (development team) after the completion of testing activities and approved by the system administrator (operations team). This offers the advantage that the application update process is transparent in every environment, and it is clear when and by whom the update is submitted and approved.

![Simulation of Branch Usage Activities](image)

**Figure 11. Simulation of Branch Usage Activities**

Figure 11 shows a simulation of branch usage activities in application development until release to the production environment. Programmer develops features A, B, and C and sends (push) source code to master/main branch (activities 1, 2, 3). The team decides to release features A, B, and C, so programmer submits a merge request to the testing branch (activity 4). When quality assurance approves the merge request, the testing environment is updated with features A, B, C, and quality assurance can start testing all three features.

Besides the testing performed by the quality assurance, the programmer continues the development of D, E feature and sends (push) the source code to the master/main as usual (activities 5, 6). Quality assurance is done with testing and decide that these three features can be released to production, so quality assurance submits merge request to production branch (activity 7).

When system administrator approves the merge request, the production environment is updated with feature A, B, and C that have passed previous test. Users can uses the new features. Along with the release to production activities, the programmer continues the development of feature F and sends (push) the source code to the master/main branch as usual (activity 8). Then the team decides to release feature D, E, and F, so programmer submits a merge request to the testing branch (activity 9). Besides the testing performed by quality assurance for feature D, E, and F, programmer continues the development of feature G and sends (push) the source code to the master/main branch (activity 10).

In those three environments (development, testing, and production), there are two types of technologies: virtual machines and Docker Swarm. Virtual machines are used for applications that do not yet use container technology, while Docker Swarm is used for applications that already use container technology.

### 3.4 Implementation Obstacles with On-Premise Infrastructure

DevOps implementations with on-premises infrastructure require additional work to setup and manage supporting infrastructure for DevOps, namely version control tools in the form of GitLab and CI/CD tools in the form of GitLab Runner. In addition to this additional work, there are some obstacles to implementation. One of the obstacles is that on-premises infrastructure is less flexible than cloud infrastructure, so scalability and elasticity are not maximized.
The process of scaling by adding resources to an existing server (referred to as scaling up or scaling up) must first shut down the server for downtime to occur. The process of scaling by adding a new server (referred to as scaling out or scaling out) requires the operations team to deploy and configure a new server before it can be used. Nor does the existing infrastructure support elasticity, where resources automatically adjust to meet demand.

Another obstacle is that it is not self-service. Unlike cloud infrastructure, which can be managed by the development team, on-premises infrastructure is not, as the operations team manages it. Some requests sometimes cannot be processed immediately due to workload and limited human resources.

4. CONCLUSION

PT Logistik Pangan, as the state-owned enterprise, implements DevOps incrementally, adopting DevOps practices one by one, starting with version control (also known as source control or source code management), then continuous integration, and finally continuous delivery. DevOps can be implemented with on-premises infrastructure but requires additional work to setup and manage supporting infrastructure for DevOps, in the form of version control tools (e.g. GitLab) and CI/CD tools (e.g. GitLab Runner). There are also some obstacles, such as on-premises infrastructure that is less flexible than cloud infrastructure and is not self-service. The company tries to overcome this by implementing container technology with Docker. Additional DevOps supporting infrastructure to be prepared and managed in the form of a Docker registry to store images produced from continuous integration and used by continuous delivery.

In order to maximize DevOps implementation, the company must ensure that the required mindset and culture have been developed, particularly the culture of collaboration. Competency improvement with appropriate training is also needed so the company’s employees can perform well in the next phase, namely implementing practices and tools. The company must also implements principles other than automation, namely culture, lean, measurement, and sharing. An important principle is the culture of collaboration, which is also mentioned in the first phase of the DevOps transformation. The company should continues to the next stage, such as operate and monitor, to maximize the benefits of DevOps. Applicable activities are continuous monitoring, which monitors the performance, capacity, and availability, and security of running applications, and continuous feedback, which provides feedback to the development team and operations team based on the results of the monitoring performed.

One of the limitations of this study is that the company still need to adopt cloud infrastructure, either the core infrastructure that runs applications or the supporting infrastructure of DevOps. Future research could use a combination of infrastructures, some infrastructures use cloud and others use on-premises. Another limitation is that the tools used are all GitLab products, namely self-managed GitLab as version control tool and GitLab Runner as CI/CD tools. Future research can use other DevOps tools or combine GitLab products with other products.

REFERENCES


