Implementasi Continous Integration/Continous Delivery Menggunakan Process Manager 2 (Studi Kasus: SIAKAD Akademi Keperawatan Bina Insan)

Danur Wijayanto[1*], Arizona Firdonsyah[2], Faisal Dharma Adhinata[3]

[1,2]Program Studi Teknologi Informasi, Universitas `Aisyiyah Yogyakarta, Indonesia
[3]Program Studi Rekayasa Perangkat Lunak, Institut Teknologi Telkom Purwokerto, Indonesia
Email: [1*]danurwijayanto@unisayogya.ac.id, [2]arizona@unisayogya.ac.id, [3]faisal@ittelkom-pwt.ac.id

(Naskah masuk: 23 Agu 2021, direvisi: 17 Okt 2021, diterima: 27 Okt 2021)

Abstrak

Pada perkembangan perangkat lunak yang semakin beragam dan kompleks, diperlukan fleksibilitas dan adaptasi terhadap proses pengembangan perangkat lunak. Konsep *DevOps* muncul dari permasalahan yang muncul antara *developer* dan *operation*. CI/CD dapat mendukung *DevOps* dikarenakan dapat mempercepat proses integrasi dan *delivery* perangkat lunak kepada pengguna. Dalam menerapkan CI/CD diperlukan *tools* pendukung seperti *git* sebagai *source code control* dan *jenkins* untuk membantu proses *deployment*. Penelitian yang dilakukan penulis menggunakan *Process Manager* 2 (PM2) untuk implementasi CI/CD pada sistem Sistem Informasi Akademik (SIAKAD) Akademi Keperawatan Bina Insan. Diharapkan penelitian ini berkontribusi untuk memperluas wawasan mengenai *tools* dalam mengimplementasikan CI/CD. Hasil menunjukkan implementasi CI/CD menggunakan *GitHub Repository*, *Jenkins*, dan PM2 berhasil dilakukan dan berjalan dengan baik. PM2 menunjukkan performa yang lebih baik daripada *Docker* jika dilihat dari segi waktu *build* dan penggunaan RAM. PM2 memerlukan waktu *deployment* 185 detik, 46% lebih cepat daripada *Docker*. Sedangkan penggunaam RAM PM2 sebesar 1,9 GB, 45% lebih sedikit daripada *Docker*.

Kata Kunci: CI/CD, *Jenkins, Continous Deployment, Continous Integration, DevOps*.

Continous Integration/Continous Delivery Implementation using Process Manager 2 (Case Study: SIAKAD Akademi Keperawatan Bina Insan)

Abstract

Software development is increasingly diverse and complex, required flexibility and adaptation to the software development process. The concept of DevOps arises from problems that occur between developers and operations. CI/CD can support DevOps because it can speed up the process of software integration and delivery to the users. For CI/CD implementation is needed tools such as git for source code control, and jenkins to help deployment process. The research conducted by the author uses Process Manager 2 (PM2) in the implementation of CI/CD. The research was conducted on the Sistem Informasi Akademik (SIAKAD) Akademi Keperawatan Bina Insan. Hoped this research will contribute to broaden the knowledge of CI/CD implementation. The results show that the CI/CD implementation using the GitHub Repository, Jenkins, and PM2 is successful and running well. PM2 performs better than Docker in terms of build time and RAM usage. PM2 requires 185 seconds for deployment, 46% faster than Docker and PM2's RAM usage is 1.9 GB, which 45% lower than Docker.

Keywords: CI/CD, *Jenkins, Continous Deployment, Continous Deployment, DevOps.*

I. PENDAHULUAN

Pada realita pengembangan perangkat lunak yang semakin beragam dan kompleks, diperlukan fleksibilitas dan adaptasi. Diperlukannya koneksi yang erat antara proses *development* dan *execution* untuk mempersingkat waktu *delivery* yang ditunjukkan dengan munculnya *Continous Practices*, seperti *Continous Integration, Delivery*, dan *Deployment* [1]. Namun, menurut [2] pendekatan yang lebih umum lebih diperlukan daripada pendekatan yang hanya berfokus pada integrasi berkelanjutan dari perangkat lunak seperti *DevOps*.

Konsep *DevOps* muncul dari permasalahan antara *developer* dan *operation*. Menurut [3] *DevOps* memerlukan proses *automation* dimana proses *build*, *deployment*, dan *testing* suatu perangkat lunak dilakukan secara otomatis dengan tujuan mempercepat *delivery* dan mendapatkan umpan balik dari pengguna aplikasi serta memudahkan dalam pendistribusiannya. Hal ini tentunya didukung oleh teknologi komputasi awan yang mendukung *scalability*, *Continous Integration*, dan *Continous Delivery* (CI/CD) dengan minimal waktu *downtime* [4], [5]. Dalam praktiknya, CI/CD dapat mendukung *DevOps* dikarenakan dapat mempercepat proses integrasi dan *delivery* perangkat lunak kepada pengguna [6].

Dalam menerapkan CI/CD diperlukan *tools* pendukung seperti *git* untuk *source code control* dan *jenkins* untuk membantu proses *deployment*. Penelitian mengenai CI/CD sudah banyak dilakukan, diantaranya penelitian [7], [8]. Penelitian tersebut menggunakan *Docker* untuk menjalankan aplikasi dan menerapkan CI/CD. Berbeda dengan kedua penelitian tersebut, penelitian yang dilakukan penulis tidak menggunakan *Docker* dalam implementasi CI/CD, namun menggunakan *Process Manager* 2 (PM2). Penelitian dilakukan pada sistem Sistem Informasi Akademik (SIAKAD) Akademi Keperawatan Bina Insan. Diharapkan penelitian ini berkontribusi untuk memperluas wawasan mengenai *tools* dalam mengimplementasi CI/CD.

II. DASAR TEORI

Pada subbab ini akan dipaparkan mengenai dasar teori digunakan pada penelitian ini.

A. *DevOps*

DevOps berasal dari kata '*development'* dan '*operation'* [9], [10]. *DevOps* menurut [6] didefinisikan sebagai suatu budaya kolaborasi, kepemilikan, dan pembelajaran dengan tujuan untuk mempercepat pengembangan perangkat lunak dari ide sampai produksi, serta menghasilkan perangkat lunak dengan kualitas baik dan *reliable* [9]. *DevOps* menggabungkan antara pengembangan perangkat lunak dan operasi *deployment* [2], [6], [9].

DevOps memerlukan proses yang terotomatisasi seperti proses manajemen kode dan *deployment* untuk melakukan rilis aplikasi yang berulang secara cepat dan tetap menjamin kualitas perangkat lunak [9]–[12]. *DevOps* melakukan implementasi *Continous Deployment Pipeline* (CDP) [10], atau *Continous Delivery* [6], [9] yang bertujuan untuk meningkatkan kolaborasi antara *developer*s dan tim operasi.

B. CI/CD

CI/CD bersumber pada *agile methodologies* [6]. *Continous Integration* (CI) dan *Continous Delivery* (CD) merupakan aktivitas pengembangan perangkat lunak yang mengintegrasikan semua aktivitas dari tahap pengembangan sampai perangkat lunak tersebut sampai ke pengguna setiap saat [8], [13].

1. *Continous Integration*
   Continous Integration merupakan tahap dimana kode dari *developer* secara terus menerus digabungkan pada *Repository* yang terpusat, yang sekaligus berfungsi sebagai *source control system*. Dengan selalu melakukan integrasi kode di awal dan sesering mungkin, dapat mencegah tim *developer* mengalami masalah integrasi yang lebih besar apabila terdapat perubahan kode yang sedang dikerjakan bersama-sama [7].
2. *Continous Delivery*
   Continous Delivery bertujuan untuk menjamin aplikasi selalu pada kondisi siap pada *production-state* [1] dan merupakan kebijakan yang bertujuan untuk mengurangi waktu yang dibutuhkan untuk menghasilkan perangkat lunak [7]. *Continous Delivery* digunakan untuk melakukan tes pada lingkungan yang mirip dengan lingkungan *production* [9].
3. *Continous Deployment*
   Continous Deployment merupakan proses *deployment* setiap kode yang telah melewati tahap *Continous Delivery* ke lingkungan *production* supaya dapat digunakan oleh pengguna [1], [9].

Dalam membentuk lingkungan CI/CD, diperlukan banyak waktu karena memerlukan konfigurasi beberapa *tools*. Namun CI/CD menawarkan banyak keuntungan. Berikut keuntungan dari CI/CD menurut [6] :

1. Pengujian otomatis menyeluruh. Pengujian dijalankan saat *developers* melakukan *commit* kode di *branch* utama.
2. *Feedback* yang cepat. *Developers* menerima umpan balik yang cepat, seperti deteksi *error*, yang terjadi sebelum dilakukan *merge* kode. Dengan demikian *developers* dapat mempersingkat proses, mendeteksi *error*, dan melakukan *deployment* dengan lebih cepat.
3. Mengurangi konflik interpersonal. Proses yang otomatis akan mengurangi gesekan antar tim dan mendorong tim lebih efektif dan kolaboratif di lingkungan kerja.
4. Proses *deploy* yang handal. *Continous Integration* dapat menjamin kode telah lolos tes dan berjalan dengan baik pada lingkungan *testing* sebelum digunakan oleh pengguna.

C. *Git*

Menurut [4], *Git* merupakan salah satu alat yang disebarluaskan secara bebas dan *open source* [14], [15] yang digunakan secara luas oleh *developers* untuk memelihara versi dari kode, sehingga memungkinkan setiap *developers* menggunakan *branch* mereka sendiri. Saat ini terdapat perusahaan yang menawarkan *git Repository* yang memudahkan *developers* untuk berkolaborasi dan melakukan manajemen kode, seperti *Gitlab* dan *GitHub*.

GitHub merupakan *platform* penyimpanan *Repository* yang mengijinkan *developers* untuk melakukan pelacakan permasalah kode dan manajemen kode secara gratis [16]. Berbeda dengan *GitHub*, *GitLab* dibangun dengan tujuan sebagai alat kolaborasi daripada solusi penyimpanan *Repository*. *GitLab* bersifat *open source* dan berbasis *web* serta menyediakan fitur seperti CI/CD *pipelines* yang menjadi inti dari *GitLab* [4], [16].

D. *Jenkins*

Jenkins merupakan *open source automation server* yang menyediakan *plugins* untuk mendukung *development*, *deploying*, dan otomatisasi proyek [17] serta digunakan untuk automatisasi CI [2]. *Jenkins* memperoleh perhatian tinggi daripada *tools* CI yang lain seperti *Bamboo*, *Gitlab*, *Subversion*, dan yang lainnya [1], [4].

Jenkins melalukan *deployment* setelah terjadi pergantian kode dilakukan di *Git Repository*. *Jenkins* kemudian akan melakukan *pull* kode dari *branch* yang bersangkutan yang kemudian akan dilakukan proses *deployment* sesuai instruksi yang diberikan.

E. PM2

Process Manager 2 (PM2) merupakan sebuah *library* yang berfungsi sebagai *Process Manager* untuk aplikasi *Node.js* [18]. Pada penelitian ini, penulis akan menguji apakah PM2 dapat menggantikan *Docker* dalam menjalankan aplikasi. PM2 membuat aplikasi dapat berjalan sebagai *daemon* dan membantu dalam mengelola dan menjaga aplikasi tetap berjalan. Seperti yang dijelaskan pada dokumentasinya [19], PM2 membutuhkan konfigurasi sebelum dapat dioperasikan seperti nama aplikasi, *script* yang dijalankan, *environment variabel* dan yang lainnya.

III. METODOLOGI PENELITIAN

Sistem yang dibangun membutuhkan 1 VPS sebagai tempat untuk menjalankan aplikasi dan *tools* seperti *jenkins*, dan PM2. Sedangkah untuk *hosting repository* menggunakan *GitHub*. Alur dalam implementasi sistem ditunjukkan pada Gambar 1 yang dimulai dengan membuat *GitHub Repository*, kemudian membuat *branch* yang akan digunakan. Kemudian melakukan konfigurasi *jenkins*, PM2, *apache*, dan *database* pada *Server*, dan yang terakhir dilakukan pengujian sistem.

Terdapat 2 pengujian, yaitu pengujian performa yang membahas mengenai performa dari PM2 dan *Docker* dari segi waktu dan penggunaan RAM (*Random Access Memory)* yang dibutuhkan untuk melakukan *deployment*, dan pengujian fungsional untuk mengetahui apakah CI/CD dan sistem dapat berjalan dengan baik.

Gambar 1. Alur Implementasi Sistem

Dalam penelitian ini, kami tidak menggunakan *Docker container* sebagai wadah aplikasi, namun aplikasi dijalankan langsung menggunakan PM2. Dengan menggunakan PM2, kita tidak memerlukan r*epository* atau tempat penyimpanan tambahan untuk menyimpan *image Docker*. Tabel 1 menunjukkan *tools* dan perangkat yang digunakan dalam implementasi sistem. Sedangkan Tabel 2 menunjukkan spesifikasi dari *server* yang digunakan.

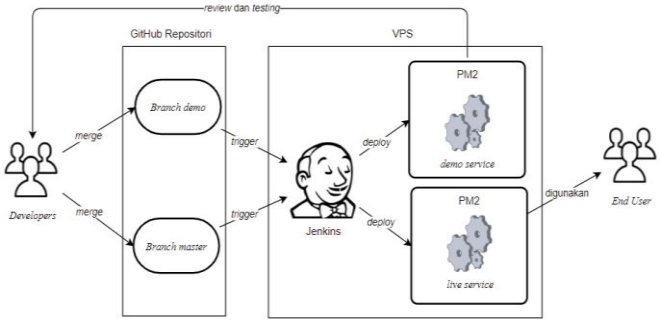
Tabel 1. *Tools* dan Perangkat

***Tools* dan Perangkat**	**Fungsi**
Virtul Private *Server* (VPS)	Tempat untuk menjalankan aplikasi dan *tools*
GitHub	*Hosting Repository*
Jenkins	*Tools deployment*
PM2	*Process Manager*
Apache	*Web Server*
Postgres	*Database*

Tabel 2. Spesifikasi *Server*

Komponen	**Spesifikasi**
CPU	4 vCPU
RAM	8 GB
SSD	50 GB NVMe

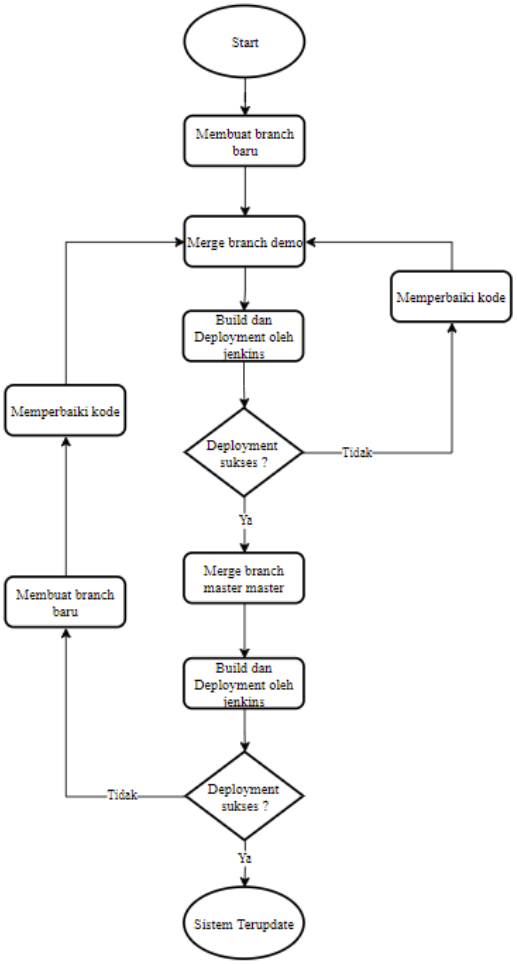
Dalam *GitHub Repository*, kami menggunakan 2 *branch* utama yaitu *master* dan *demo* serta *branch* modul yang diambil dari *branch master* untuk proses pengembangan. *Branch master* digunakan untuk merilis dan menggabungkan kode aplikasi supaya dapat digunakan oleh pengguna. Sedangkan *branch demo* digunakan untuk menggabungkan kode dari modul *developer* sehingga dapat dilakukan *testing* di lingkungan *demo*. Hubungan antara *Tools*, Perangkat, *Developers*, dan pengguna sistem ditunjukkan pada Gambar 2.



Gambar 2. Relasi Antara *Tools*, Perangkat, *Developers*, dan Pengguna Sistem

Setelah *developer* melakukan *merge* kode dari modul ke *branch demo* atau *merge* kode dari *branch demo* ke *master*, maka *jenkins* akan terpicu untuk melakukan proses *pull* dan melakukan *deployment* yang dilakukan di VPS. *Jenkins* akan melakukan *pull* kode berdasarkan *branch* yang sudah diatur saat konfigurasi, sehingga hanya *branch* tertentu saja yang

bisa memicu *jenkins* untuk bekerja. *Branch* yang digunakan untuk memicu *jenkins* yaitu *branch demo* dan *master*, sedangkan *branch* modul tidak diikut sertakan dikarenakan *branch* modul digunakan saat masih proses *development*. Proses ini dapat dilihat pada Gambar 3.



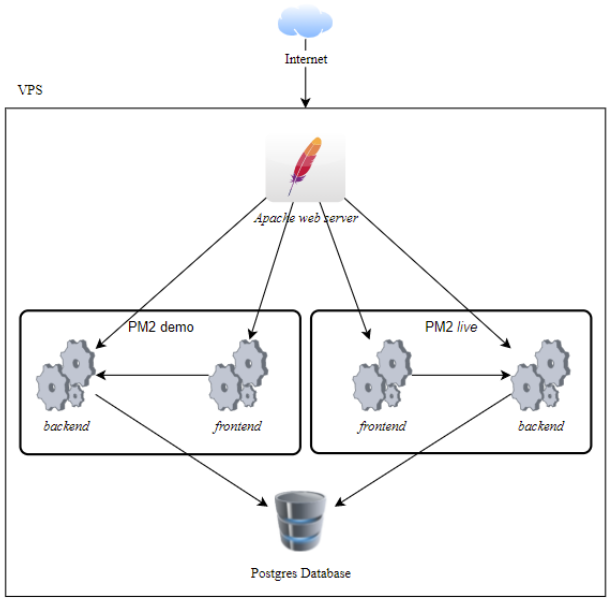
Gambar 3. Alur Pembuatan *Branch* dan *Merge*

Setelah *jenkins* selesai melakukan *pull* kode dari *GitHub Repository*, *jenkins* akan menjalankan proses *deployment*. Alur proses *deployment* yaitu:

1. Melakukan *instalasi package* yang diperlukan untuk menjalankan aplikasi.
2. Menjalankan aplikasi dalam bentuk *daemon* dengan bantuan PM2.

Proses *deployment* dapat dipantau pada interface *Jenkins*. *Developer* dapat memantau apakah proses *deployment* sukses dilakukan atau gagal beserta *log* yang dapat digunakan untuk mengetahui letak *error* saat terjadi kegagalan *deployment*.

Detail dari *service* PM2 dan topologi *server* ditunjukkan pada Gambar 4. Jumlah total *service* aplikasi yang dijalankan oleh PM2 adalah 4 yang masing-masing terbagi menjadi 2 *service* (*backend* dan *frontend*) pada *demo* dan *live*. Untuk dapat menjalankan *service* yang bersamaan, diperlukan konfigurasi *port* yang berbeda antar *service*. Masing-masing *service frontend* dan *backend* berhubungan dengan *apache web server* supaya *service* tersebut dapat diakses dari internet menggunakan *domain*. Selain itu *service backend* juga akan berhubungan dengan *Postgres database* untuk melakukan pengolahan data. Tabel 3 menunjukkan detail domain dan *port* yang digunakan pada setiap *service*.



Gambar 4. Topologi *Server*

Tabel 3. Pembagian *Domain* dan *Port*

Service	*Port*	*Domain*
demo frontend	3000	*demo-*siakad.akperbinainsan.ac.id
demo backend	3011	*service-*demo.akperbinainsan.ac.id
live frontend	3001	siakad.akperbinainsan.ac.id
live backend	3012	demo.akperbinainsan.ac.id

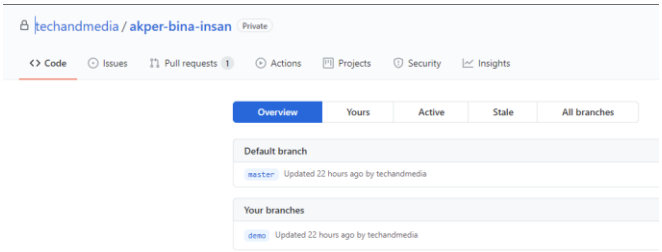
IV. HASIL DAN PEMBAHASAN

A. Hasil

Pada bagian ini penulis akan memaparkan mengenai proses dan hasil implementasi sesuai dengan alur implementasi sistem yang ditunjukkan pada Gambar 1.

1. Membuat *GitHub Repository* dan *Branch*

Seperti yang sudah dijelaskan pada Gambar 1, implementasi diawali dengan membuat *GitHub Repository* yang dilanjutkan dengan membuat *branch master* dan *demo*. Tampilan *GitHub Repository* bisa dilihat pada Gambar 5.



Gambar 5. *GitHub Repository*

2. Konfigurasi PM2

Supaya *server* dapat menjalankan *service* PM2, perlu dilakukan instalasi PM2 di *server* dan membuat *file* konfigurasi untuk menjalankan aplikasi. Instalasi PM2 dapat dilakukan dengan perintah "*npm install pm2@latest*" [19]. Sedangkan untuk *file* konfigurasi, *file* ditempatkan di *Repository* aplikasi supaya dapat diperbaharui kapan saja oleh *developer*. Konfigurasi PM2 dapat dilihat pada Gambar 6.

```
1  module.exports = {
2    apps: [
3      {
4        name: "be-akper-demo",
5        cwd: "be",
6        script: "dist/main.js",
7        args: "cross-env node",
8        watch: true,
9        env: {
10         NODE_ENV: "production",
11       },
12     },
13     {
14       name: "ui-akper-demo",
15       cwd: "ui",
16       script: "npm",
17       args: "start",
18       env: {
19         NODE_ENV: "production",
20       },
21     },
22   ],
23 };
```

Gambar 6. Potongan Konfigurasi PM2

Format konfigurasi PM2 menggunakan *.js* (*javascript file*). Konfigurasi yang ditunjukkan pada Gambar 6 menunjukkan konfigurasi PM2 akan menjalankan 2 *service* yang bernama be-akper-demo sebagai *backend* dan ui-akper-demo sebagai *frontent* yang ditunjukkan pada baris ke 4 dan ke 14. Baris ke 5 dan ke 15 menunjukkan *directory* dari kode yang akan dijalankan, sedangkan baris selanjutnya menunjukkan jenis *script*, argumen dan jenis *environment* yang akan digunakan pada aplikasi. Konfigurasi dapat dijalankan oleh PM2 dengan perintah: *pm2 start <file-konfigurasi>*.

3. Konfigurasi *Web Server*

Dalam melakukan konfigurasi *Web Server*, perlu menerapkan *reverse proxy*. *Reverse proxy* digunakan untuk mengarahkan *request* dari *client* ke *backend server* [20] . Pada penelitian ini, *backend server* berupa *service* aplikasi yang dijalankan pada beberapa *port* yang berbeda yang ditunjukkan pada Tabel 2. Dengan menggunakan *reverse proxy*, aplikasi yang berjalan di *port* selain 80 (HTTP) dan 443 (HTTPS) dapat diarahkan ke *port* tersebut.

```
1  <VirtualHost *:80>
2      ServerName
3  siakad.akperbinainsan.ac.id
4      ProxyPass / http://127.0.0.1:3001/
5  ProxyPassReverse                    /
6  http://127.0.0.1:3001/
7  RewriteEngine on
8  RewriteCond              %{SERVER_NAME}
9  =siakad.akperbinainsan.ac.id
10 RewriteRule                         ^
11 https://%{SERVER_NAME}%{REQUEST_URI}
12 [END,NE,R=permanent]
13 </VirtualHost>
```

Gambar 7. Konfigurasi *Web Server*

Konfigurasi *reverse proxy* dapat dilihat pada Gambar 7 pada baris 3 sampai 6. Konfigurasi tersebut akan mengarahkan pengguna yang mengakses *domain* siakad.akperbinainsan.ac.id pada *port* 80 ke alamat *localhost* dengan *port* 3001.

4. Konfigurasi *Jenkins*

Jenkins merupakan aplikasi berbasis *Java*, sehingga *server* perlu diinstall *Open Java Development Kit* (*OpenJDK*). *Jenkins* berfungsi sebagai alat untuk melakukan *deployment*. *Jenkins* bekerja dengan mendeteksi apakah terjadi proses *push* pada *GitHub Repository*. Apabila *jenkins* mendeteksi hal tersebut, maka *jenkins* akan melakukan proses *pull* kode ke *server* dan *deployment*.

```
1  cd be && cp -R config-deployment/demo/*
2  config/
3  yarn && yarn build && cd ..
4  cd ui && cp -R config-deployment/demo/* .
5  yarn && yarn build && npx next telemetry
6  disable && cd .. && sudo pm2 restart
7  ecosystem-demo.config.js
```

Gambar 8. Perintah Dalam *Deployment* Aplikasi

Perintah yang terdapat pada Gambar 8 secara otomatis akan melakukan *deployment* aplikasi *backend* dan *frontend* sekaligus. Secara garis besar, langkah-langkah dalam proses *deployment* sebagai berikut :

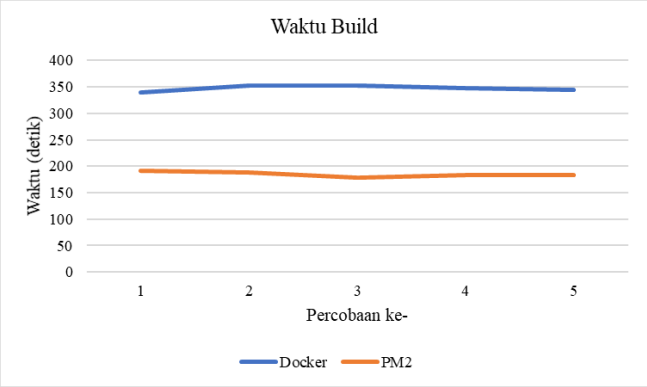
1. Menyalin *file* konfigurasi aplikasi yang ditunjukkan pada baris 1,2 dan 4.
2. Melakukan instalasi *package* yang dibutuhkan dalam menjalankan aplikasi yang ditunjukkan pada baris 3 dan 5.
3. Menjalankan *service* aplikasi menggunakan PM2 yang ditunjukkan pada baris 7.

B. Pembahasan

Pada subbab ini akan membahas mengenai pengujian dari implementasi sistem. Penulis melakukan dua pengujian, yaitu pengujian performa yang membahas mengenai performa dari PM2 dan *Docker* dari segi waktu dan penggunaan RAM (*Random Access Memory)* yang dibutuhkan untuk melakukan *deployment*, dan pengujian fungsional untuk mengetahui apakah CI/CD dan sistem dapat berjalan dengan baik.

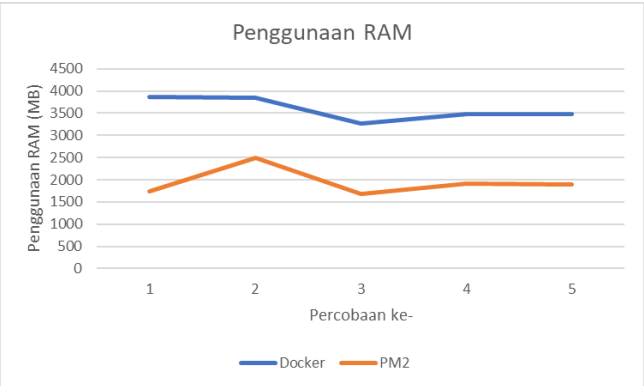
1. Pengujian Performa

Untuk mengukur waktu *deployment* dan penggunaan RAM, penulis melakukan 5 kali percobaan. Gambar 9 menunjukkan perbandingan waktu *build* antara *Docker* dan PM2. Dari 5 percobaan yang dilakukan, waktu *build* antara *Docker* dan PM2 cenderung konstan. Dengan menggunakan PM2, dapat mengurangi waktu *build* hampir sekitar 46% dengan rerata waktu *build* PM2 memerlukan waktu 185 detik, sedangkan *Docker* memerlukan waktu 347 detik.



Gambar 9. Perbandingan Waktu *Build* Antara *Docker* dan PM2

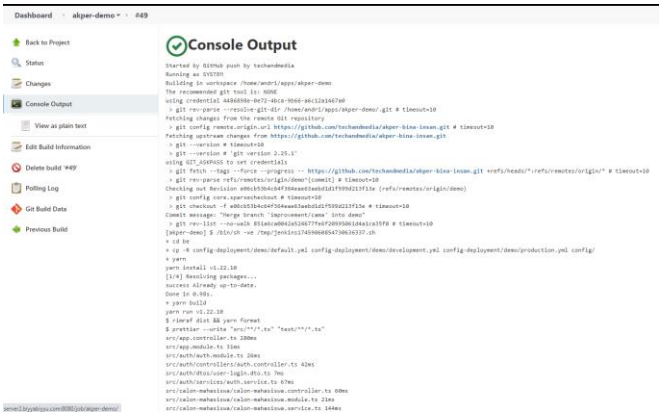
Sedangkan hasil dari pengujian RAM tidak jauh berbeda dengan pengujian waktu *deployment*. PM2 juga menunjukkan hasil yang lebih baik daripada *Docker* karena menggunakan RAM yang lebih sedikit. Dengan menggunakan PM2, dapat mengurangi penggunaan RAM sekitar 45% dengan rerata penggunaan RAM PM2 sebesar 1,9 GB, sedangkan *Docker* memerlukan RAM sebesar 3,5 GB.



Gambar 10. Perbandingan Pengunaan RAM Antara *Docker* dan PM2

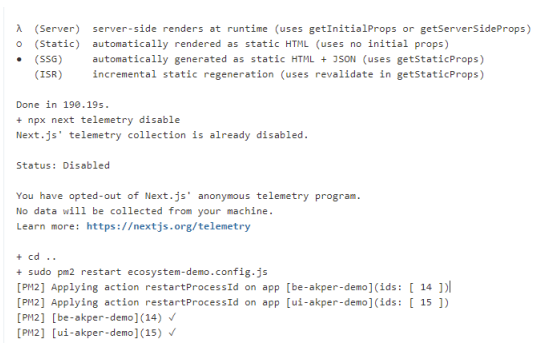
2. Pengujian Fungsional

Pengujian fungsional dilakukan dengan memulai pembuatan *branch* baru sampai aplikasi berhasil ter*update* dan berjalan di *server*. Untuk meyakinkan apakan kode di *server* sudah sesuai dengan *update* kode yang terbaru, perlu dicocokkan *commit id* dari *GiHub Repository* dan *server* menggunakan perintah *git log*. Setelah *developer* melakukan *merge* kode ke *branch master* dan *demo*, maka *jenkins* akan terpicu seperti yang ditunjukkan pada Gambar 11.



Gambar 11. *Log Jenkins*

Gambar 12 menunjukkan *Jenkins* melakukan *pull code* sesuai dengan *branch* yang ter-*update*. Kode tersebut disimpan pada *directory workspace jenkins* dan kemudian *jenkins* mulai menjalankan list perintah *deployment* aplikasi yang ditunjukkan pada Gambar 8.



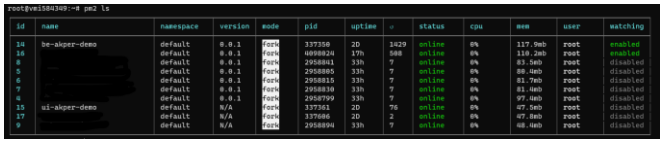
Gambar 12. *Jenkins* selesai melakukan *deployment*

Gambar 13. Informasi *build* yang dilakukan *jenkins*

Apabila *build* dan *deployment* aplikasi sukses dilakukan oleh *Jenkins*, maka akan terdapat keterangan seperti Gambar 12 dan 13. Dalam Gambar 14, menunjukkan aplikasi *frontend* dan *backend* berhasil dijalankan. Seperti pada Gambar 6, aplikasi *frontend* mempunyai nama ui-akper-demo sedangkan

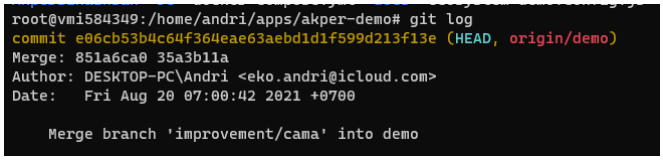
aplikasi *backend* mempunyai nama be-akper-demo. Proses *deployment* memerlukan waktu 4 menit 24 detik dikarenakan diperlukan proses intalasi beberapa *package* yang dibutuhkan untuk menjalankan aplikasi.

Untuk memastikan apakah aplikasi berjalan, bisa menggunakan perintan *pm2 ls*. Perintah tersebut digunakan untuk melihat list *service* yang dijalankan oleh PM2. Gambar 14 menunjukkan kedua *service* tersebut mempunyai status online, menandakan bahwa aplikasi berhasil dijalankan.

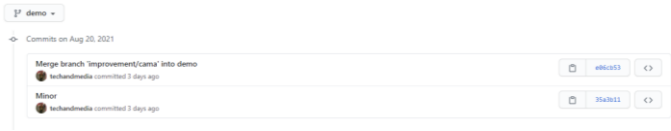


Gambar 14. Status Service Yang Dijalankan PM2

Setelah mengetahui aplikasi berhasil dijalankan oleh PM2, langkah selanjutnya yaitu melakukan pencocokkan *commit id* dari *GiHub Repository* dan *server*. Gambar 15 menunjukkan hasil dari perintah *git log* yang menampilkan riwayat dari *commit* yang dilihat dari *server*. Sedangan Gambar 16 menunjukkan riwayat dari *commit* yang dilihat dari *GitHub Repository*.



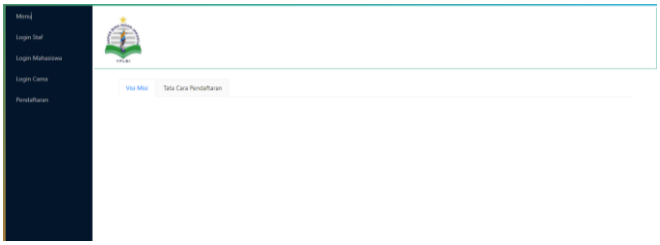
Gambar 15. Hasil Perintah *Git Log*



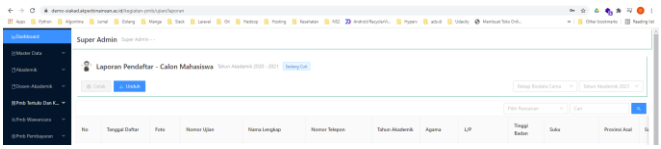
Gambar 16. Riwayat *Commit GitHub*

Dari Gambar 15 dan 16, dapat disimpulkan bahwa *commit id* antara aplikasi yang berada di *server* dan *GiHub Repository* telah selaras karena mempunyai commit id yang sama yaitu e06cb53. Hal tersebut menandakan bahwa versi aplikasi sudah berhasil diperbaharui. Pengujian terakhir dilakukan dengan mengakses alamat *url* dari masing-masing *service*. Pengujian ini dilakukan untuk mengetahui apakah kedua *service* tersebut berhasil berkomunikasi dengan baik dan dapat digunakan oleh pengguna.

Gambar 17 menunjukkan halaman *login frontend* yang beralamat di demo-siakad.akperbinainsan.ac.id. Untuk mengetahui apakah aplikasi *frontend* dan *backend* dapat berkomunikasi dengan baik, maka perlu dilakukan *login* ke aplikasi. Apabila berhasil *login*, pengguna akan melihat *dashboard* aplikasi seperti pada Gambar 18. Hal tersebut menandakan antara *frontend*, *backend*, dan *database* dapat saling berkomunikasi dengan baik.



Gambar 17. Antarmuka Aplikasi



Gambar 18. *Dashboard* Aplikasi

V. KESIMPULAN

Dari hasil pengujian dapat disimpulkan bahwa implementasi CI/CD menggunakan *GitHub Repository*, *Jenkins*, dan PM2 berhasil dan dapat berjalan dengan baik. Terlihat dari hasil pengujian, PM2 menunjukkan performa yang lebih baik daripada *Docker* dan akan sangat berguna jika PM2 diterapkan pada lingkungan komputer yang mempunyai sumber daya terbatas. Jika dilihat dari segi waktu *build* dan penggunaan RAM, PM2 memerlukan waktu *deployment* 185 detik, 46% lebih cepat daripada *Docker*. Sedangkan penggunaam RAM PM2 sebesar 1,9 GB, 45% lebih sedikit daripada *Docker*.

Penelitian dapat dikembangkan lebih lanjut dengan meneliti mengenai teknik untuk mempersingkat waktu *deployment* atau *load balancing* dan *failover* menggunakan PM2.

UCAPAN TERIMAKASIH

Terima kasih disampaikan kepada Eko Andri Subarnanto, S.Kom. selaku *Project Manager* dan pihak terkait dalam pengerjaan proyek SIAKAD Akademi Keperawatan Bina Insan yang telah mengijinkan penulis untuk menyelesaikan penelitian ini.

REFERENSI

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

[2] Fitzgerald, B. & Stol, K.J. (2017). Continuous Software Engineering: a Roadmap and Agenda. *Journal of Systems and Software*, vol. 123, pp. 176–189, doi: 10.1016/j.jss.2015.06.063.

[3] Humble, J. (2010). *Continuous Delivery vs Continuous Deployment*. Diakses dari:

https://continuous*delivery*.com/2010/08/continuous-*delivery*-vs-continuous-*deployment*/ pada tanggal 13 Agustus 2021.

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

[5] Wu, J. & Wang, T. (2014). Research and Application of SOA and Cloud Computing Model. *Proceeding of 2014 Enterprise Systems Conference*, pp. 294–299. doi: 10.1109/ES.2014.58.

[6] Freeman, E. (2019). *DevOps for dummies*, 1st ed. Indianapolis: John Wiley and Sons.

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

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

[9] Prince, S. (2016). The Product Managers' Guide to Continuous *Delivery* and DevOps. Diakses dari: https://www.mindtheproduct.com/what-the-hell-are-ci-cd-and-*DevOps*-a-cheatsheet-for-the-rest-of-us/ pada tanggal 14 Agustus 2021.

[10] Weber, I., Nepal, S. & Zhu, L. (2016). Developing Dependable and Secure Cloud Applications. *IEEE Internet Computing*, Vol. 20(3), pp. 74–79, doi: 10.1109/MIC.2016.67.

[11] Rodríguez, P. et al. (2017). Continuous Deployment of Software Intensive Products and Services: A Systematic Mapping Study. *Journal of Systems and Software*, Vol. 123, pp. 263–291, doi: 10.1016/j.jss.2015.12.015.

[12] Luhana, K.K., Schindler, C. & Slany, W. (2018). Streamlining Mobile App Deployment with Jenkins and Fastlane in the Case of Catrobat's Pocket Code. *Proceeding of 2018 IEEE International Conference on Innovative Research and Development (ICIRD)*, pp. 1–6. doi: 10.1109/ICIRD.2018.8376296.

[13] Laukkanen, E., Itkonen, J. & Lassenius, C. (2017). Problems, Causes and Solutions When Adopting Continuous Delivery - A Systematic Literature Review. *Information and Software Technology*, Vol. 82, pp. 55–79, doi: 10.1016/j.infsof.2016.10.001.

[14] Git. (2021). *About*. Diakses dari https://*git*-scm.com/ pada tanggal 16 Agustus 2021.

[15] Atlassian. (2021). *What is Git*. Diakses dari: https://www.atlassian.com/*git*/tutorials/what-is-*git* pada tanggal 16 Agustus 2021.

[16] Inara, A. (2021). *GitLab vs GitHub: Which is Right for You*. Diakses dari https://spectralops.io/blog/*GitHub*-vs-*git*lab/ pada tanggal 16 Agustus 2021.

[17] Jerkins. (2021). *Jenkins*. Diakses dari https://www.*jenkins*.io/ pada tanggal 16 Agustus 2021.

[18] PM2. (2021). *Process Manager* 2. Diakses dari https://*GitHub*.com/Unitech/pm2 pada tanggal 17 Agustus 2021.

[19] Keymetrics (2021) *PM2 Process Management Quick Start*. Diakses dari: https://pm2.keymetrics.io/docs/usage/quick-start/ pada tanggal 17 Agustus 2021.

[20] Nginx. (2021). *What Is a Reverse Proxy Server?* Diakses dari https://www.nginx.com/res*ources*/glossary/reverse-proxy-*server*/ pada tanggal 19 Agustus 2021.