ELKOMIKA: Jurnal Teknik Energi Elektrik, Teknik Telekomunikasi, & Teknik Elektronika

ISSN(p): 2338-8323 | ISSN(e): 2459-9638 | Vol. 9 | No. 3 | Halaman 678 - 689
DOI : http://dx.doi.org/10.26760/elkomika.v9i3.678 Juli 2021

Penerapan *DevOps* pada Sistem Tertanam dengan ESP8266 menggunakan Mekanisme *Over The Air*

AZIS WISNU WIDHI NUGRAHA, IMRON ROSYADI, FAHMI KHOERULLATIF

Teknik Elektro Universitas Jenderal Soedirman, Indonesia
Email: azis.wwn@unsoed.ac.id

Received 6 Desember 2020 **|** *Revised* 22 Desember 2020 **|** *Accepted* 17 April 2021

ABSTRAK

DevOps mendorong percepatan pengembangan sistem. Namun bukti nyata penerapannya pada sistem tertanam belum mencukupi. Salah satu penyebabnya adalah kesulitan proses deployment pada perangkat. Konsep IoT menghubungkan sistem tertanam dengan jaringan yang memungkinkan proses pembaharuan firmware menggunakan mekanisme Over The Air (OTA). Tulisan ini mengusulkan infrastruktur DevOps untuk pengembangan sistem tertanam. Perangkat keras yang digunakan adalah microcontroller ESP8266. Sedangkan lingkungan DevOps menggunakan perangkat lunak PlatformIO, GitHub dan Travis CI. Pengujian dilakukan dengan mengubah user requirement yang kemudian diterapkan pada perangkat keras. Tahapan DevOps (build and test, release hingga deploy) telah berhasil dilakukan secara otomatis. Sistem mampu mendeteksi kesalahan penulisan kode sumber. Rerata waktu keseluruhan proses adalah 77,21 detik. Proses build and test mendominasi waktu proses dengan rerata sebesar 77,21 detik dan waktu deploy memiliki rerata 1,41 detik.

***Kata kunci**: IoT, Sistem Tertanam, OTA, DevOps, ESP8266*

ABSTRACT

DevOps drives the acceleration of system development. However, the concrete evidence of its application in embedded systems is not sufficient. One of the causes is difficulty in the deployment process on the device. Firmware update using an Over-The-Air (OTA) mechanism is allowed by the IoT concept that connects embedded systems into a network. This paper is proposing a DevOps infrastructure for embedded system development. Proposed infrastructure using ESP8266 for the hardware and PlatformIO, GitHub, and Travis CI for the DevOps environment. Testing the proposed system is done by changing the user requirements that are applied to the hardware. The DevOps stages from building and test, release, and deployment have automatically been done. The system is also able to detect developer errors in writing source code. The average time of the whole process on trial was 77.21 seconds. The build and test process dominates the processing time with an average of 77.21 seconds and the deployment time is relatively short with an average of 1.41 seconds.

***Keywords**: IoT, Embedded System, OTA, DevOps, ESP8266*

1. PENDAHULUAN

Perkembangan *Internet of Things* (IoT) akhir-akhir ini cukup pesat dan telah memasuki berbagai macam bidang **(Rehman, dkk, 2017)**. Sebagai contoh adalah pengembangan sistem IoT pada bidang energi **(Santoso, dkk, 2018) (Tahtawi, dkk, 2019)** serta bidang keamanan rumah **(Kurniawan, dkk, 2018)**. Perkembangan ini semakin didorong oleh integrasi komputasi *cloud* sehingga memudahkan pengolahan data **(Botta, dkk, 2016)**. Integrasi tersebut memperkenalkan dimensi baru pada *big data* dan analisis data **(Bonomi, dkk, 2014)**. Penggabungan konsep *big data* dan IoT semakin memudahkan penggalian data serta mendorong pemanfaatan IoT secara masif **(Bonomi, dkk, 2014)**. Menurut **(Gubbi, dkk, 2013)**, IoT dapat didefinisikan sebagai interkoneksi antar piranti sensor maupun aktuator yang memiliki kemampuan untuk membagikan informasi pada lintas *platform* melalui *platform* terpadu guna membangun suatu gambaran operasi bersama yang memungkinkan aplikasi inovatif. Hal ini diperoleh melalui *seamless ubiquitous sensing*, analisis data, serta representasi informasi dengan komputasi *cloud* sebagai *framework* penyatu **(Gubbi, dkk, 2013)**. IoT juga menawarkan kemudahan untuk menanamkan kecerdasan pada perangkat yang pada akhirnya dapat berkomunikasi satu sama lain untuk bertukar informasi **(Chopra, dkk, 2019)**.

Dalam penerapan IoT, pengembangan sistem merupakan sesuatu yang pasti ada. Sering kali terjadi celah antara pengembangan dan pengoperasian. Pengembang sering kali akan melakukan perubahan kode yang membutuhkan penggabungan berkali-kali bahkan dalam jangka waktu satu hari **(Fowler, 2006)**. Untuk mengatasi hal tersebut maka dikembangkan konsep *DevOps* yang berarti penggabungan antara pengembangan dan operasional guna menyederhanakan proses *delivery* perangkat lunak, memperkuat proses pembelajaran dengan langsung mengarahkan umpan balik dari proses produksi kepada pengembang dan memperbaiki waktu siklus **(Hüttermann, 2012)**. Hüttermann juga menyatakan bahwa konflik yang terjadi pada pengembangan dan operasional dapat diselesaikan melalui konsep *DevOps* karena adanya peningkatan komunikasi di antara keduanya **(Hüttermann, 2012)**. Callanan dan Spillane melaporkan praktik baik pada *DevOps*. Melalui *DevOps* mereka berhasil memperbaiki waktu siklus rilis sehingga dapat meningkatkan laju rilis **(Callanan & Spillane, 2016)**.

Dengan perkembangan IoT yang cukup pesat, maka sistem IoT yang dikembangkan juga perlu untuk beradaptasi dengan cepat untuk memenuhi kebutuhan baru. Untuk menyikapi hal tersebut Guşeilă dkk menunjukkan berbagai macam perangkat lunak yang dapat digunakan untuk sistem IoT untuk menjamin *Continuous Integration* (CI) dan *Continuous Deployment* (CD) **(Guşeilă, dkk, 2019)**. Untuk menjamin proses *deployment* secara berkesinambungan pada perangkat IoT diperlukan strategi proses pembaharuan *firmware* **(Khemissa, 2018)**. Penerapan *DevOps* pada IoT dapat melibatkan *containerization* menggunakan *docker* untuk mempermudah *deployment* pada perangkat yang banyak **(Moore, dkk, 2016)**. Moore dkk menerapkan *docker* untuk menerapkan *DevOps* pada sistem informasi yang memiliki kemampuan untuk melakukan analisis waktu nyata penggunaan kendaraan listrik pada *smart city*. Beberapa pengembangan sistem IoT telah mengadopsi *DevOps* dengan memanfaatkan *docker* untuk proses *Continuous Deployment* pada perangkatnya **(Bae, dkk, 2016) (Prens, dkk, 2019) (Yigitoglu, dkk, 2017)**. Penggunaan *docker* ini mengharuskan penggunaan sistem operasi sebagai *host* bagi *docker* yang dijalankan. Untuk memenuhi kebutuhan tersebut banyak digunakan komputer SoC (*System on a Chip*) Raspberry Pi **(Bae, dkk, 2016) (Toro-Betancur, dkk, 2019) (Yigitoglu, dkk, 2017)**. Meskipun demikian, sistem-sistem yang dikembangkan di atas masih menggunakan sistem operasi yang kompleks, dan tidak dapat menggunakan perangkat *microcontroller* langsung. Hal ini juga didukung oleh kenyataan

bahwa hingga saat ini masih belum terdapat bukti empiris penerapan *DevOps* dalam pengembangan sistem tertanam berbasis *microcontroller* **(Lwakatare, dkk, 2016)**.

Sementara itu pada perangkat *microcontroller* yang tidak dapat menerapkan *docker*, sebenarnya proses pembaharuan *firmware* dapat dilakukan dengan menuliskan berkas biner *firmware* langsung ke *microcontroller*. Pembaharuan *firmaware* pada perangkat IoT yang terhubung ke jaringan menggunakan saluran nirkabel dikenal dengan istilah *Firmware Over The Air* (FOTA) **(Doddapaneni, dkk, 2017)**. *Microntroller* dapat dihubungkan dengan perangkat nirkabel untuk mendukung proses FOTA tersebut **(Chandra, dkk, 2016) (Hessar, dkk, 2020) (Jurković & Sruk, 2014) (Kerliu, dkk, 2019) (Nilsson, dkk, 2008) (Nilsson & Larson, 2008) (Thakur, dkk, 2019) (Zandberg, dkk, 2019)**. Selain itu terdapat juga sistem yang menggunakan SoC Raspberry Pi sebagai perantara untuk FOTA **(Chandra, dkk, 2016)**. ESP8266 adalah salah satu *microcontroller* yang telah memiliki konektivitas wifi di dalamnya. Keberadaan konektivitas wifi ini memungkinkan penerapan FOTA pada ESP8266 langsung melalui jaringan wifi tanpa menggunakan perantara **(Frisch, dkk, 2017) (Gore, dkk, 2017)**.

Tulisan kali ini berusaha untuk menyajikan kemungkinan penerapan metode *DevOps* pada pengembangan sistem tertanam. Pembaharuan *firmware* pada perangkat berbasis ESP8266 melalui mekanisme OTA akan digabungkan dengan konsep pengembangan berbasis *DevOps*. Infrastruktur pengembangan yang melibatkan lingkungan pengembangan dan lingkungan perangkat tertanam baik untuk unit tes maupun untuk unit operasional diusulkan pada tulisan ini. Infrastruktur yang diusulkan memanfaatkan perangkat lunak yang selama ini telah banyak dikenal dalam ekosistem *DevOps*. Simulasi kegagalan pembaharuan *firmware* yang disebabkan karena kesalahan dalam penulisan kode sumber juga dilakukan. Perangkat tertanam yang digunakan sebagai simulasi terdiri dari ESP8266, *push button* sebagai masukan sistem dan LED sebagai keluaran sistem. Perilaku *push button* dan LED akan divariasikan sesuai dengan perubahan *requirement*.

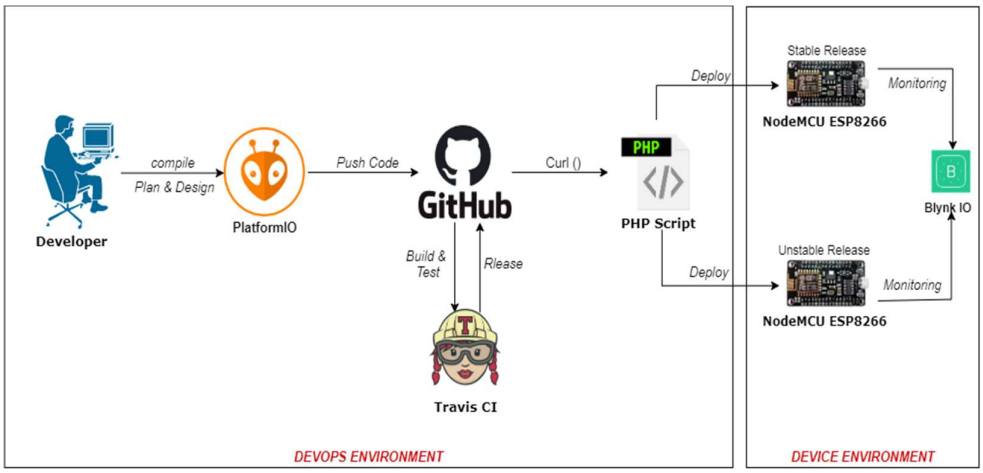
2. METODE

2.1 Rancangan Infrastruktur *DevOps* Sistem Tertanam dengan menggunakan ESP8266

Infrastruktur *DevOps* yang diusulkan dapat dilihat pada Gambar 1. Pada prinsipnya terdapat dua lingkungan, yaitu lingkungan *DevOps* dan lingkungan perangkat keras. Perbedaan dengan konsep *DevOps* perangkat lunak pada umumnya, adalah penggunaan sistem tertanam ESP8266. Pada Gambar 1 terlihat komponen penyusun infrastruktur *DevOps* untuk pengembangan sistem tertanam ESP8266 terdiri atas beberapa peranti dan perangkat sebagai berikut.

a. Lingkungan perangkat keras sistem tertanam
    a. Blynk sebagai *platform* pengembangan aplikasi pada perangkat bergerak di sistem IoT untuk memantau perubahan *firmware* pada perangkat keras.
    b. NodeMCU *development kit* sebagai piranti dengan inti *microcontroller* ESP8266 yang digunakan sebagai perangkat tes untuk rilis *unstable* dan perangkat operasional untuk rilis *stable*.
b. Lingkungan *DevOps*
    a. *PlatformIO* sebagai lingkungan pengembangan IoT yang bersifat *Open Source*.
    b. GitHub sebagai *repository* git yang dapat digunakan untuk melakukan kolaborasi dalam pengembangan sistem.

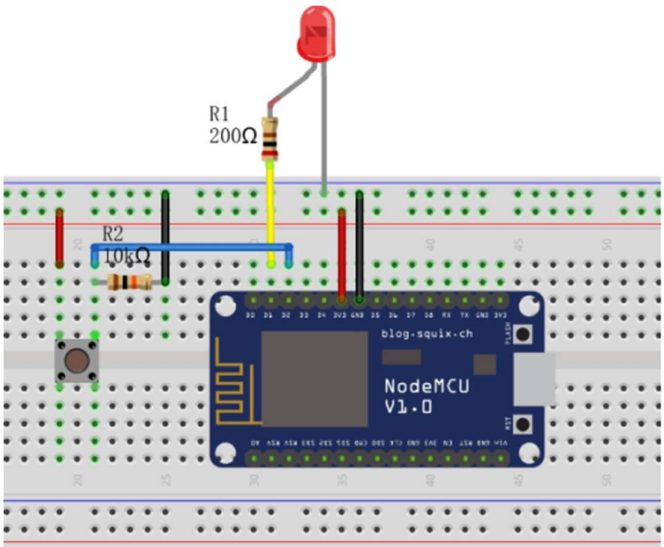
c. Travis CI sebagai platform *Continuous Integration* (CI) yang mendukung otomatisasi proses pengembangan sistem, pengujian perubahan kode serta umpan balik terhadap hasil perubahan kode maupun automasi *deployment* dan notifikasi.
d. Sebuah *server* dengan skrip php di dalamnya sebagai perantara sistem tertanam dengan *repository* git untuk keperluan pengunduhan berkas *binary firmware*.



Gambar 1. Rancangan Infrastruktur *DevOps* untuk Sistem Tertanam ESP8266.

2.2 Rancangan Perangkat Keras Sistem Tertanam

Perangkat sistem tertanam yang digunakan untuk percobaan cukup sederhana. Terdiri dari sebuah NodeMCU, sebuah *push button* dan sebuah LED. Perangkat keras ini dibuat dua set, satu untuk perangkat keras tes (untuk rilis *firmware unstable*) dan satu untuk perangkat keras operasional (untuk rilis *firmware stable*).

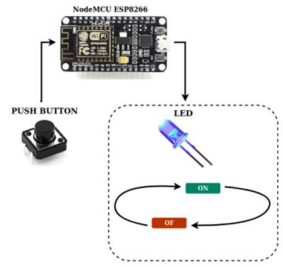
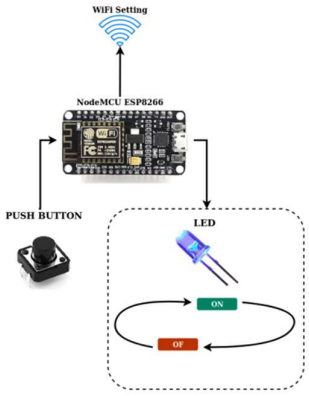
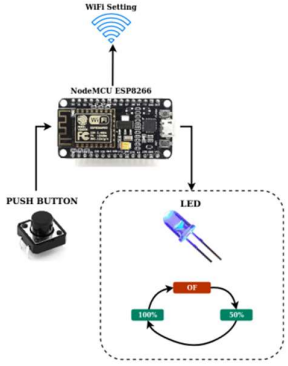
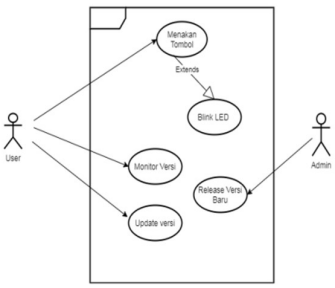
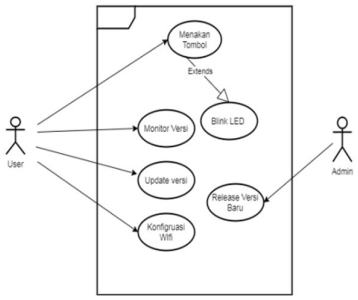
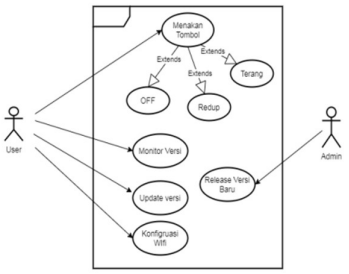


Gambar 2. *Wiring Diagram* Perangkat Sistem Tertanam.

Adapun *wiring* diagram dari perangkat keras sistem tertanam yang digunakan dapat dilihat pada Gambar 2. Sebuah *push button* terhubung dengan *port* GPIO 04 ESP8266 atau D2 NodeMCU devkit melalui sebuah resistor *pulldown*. Dengan konfigurasi ini, pada saat tombol ditekan maka logika pada GPIO04 akan tinggi. Sebuah LED terhubung ke GPIO 05 atau D1 NodeMCU devkit dengan sebuah resistor pembatas arus menuju GND. Dengan konfigurasi ini

maka pada saat GPIO 05 berlogika tinggi maka LED akan menyala. Tombol *push button* dan LED akan menjadi masukan dan keluaran sistem dengan perilaku yang disesuaikan dengan permintaan setiap siklus DevOps.

Tabel 1. *User Requirement* Tiga Fase Simulasi Pengembangan Perangkat Tertanam

	Fase **1**	Fase **2**	Fase **3**
User Requirements	1. LED dapat menyala dan mati (*blinking*) ketika *push button* ditekan sebanyak dua kali. 2. Sistem dapat memonitor versi melalui Blynk Apps 3. Sistem dapat melakukan *update* melalui Blynk Apps	1. LED dapat menyala dan mati (*blinking*) ketika *push button* ditekan sebanyak dua kali. 2. Sistem dapat memonitor versi melalui Blynk Apps 3. Sistem dapat melakukan *update* melalui Blynk Apps 4. Koneksi WiFi dapat dikonfigurasi melalui peramban	2. Sistem dapat memonitor versi melalui Blynk Apps 3. Sistem dapat melakukan *update* melalui Blynk Apps 4. Koneksi *WiFi* dapat dikonfigurasi melalui peramban 5. Lampu LED dapat menyala dengan kondisi mati, redup (50%), dan terang (100%).
Design			
Use Case Diagram			

2.3 Penerapan *DevOps* pada Pengembangan Perangkat Tertanam

Pengembangan sistem dengan menggunakan *DevOps* pada umumnya diterapkan pada piranti tehubung jaringan. Proses *DevOps* sendiri paling tidak terdiri atas tahapan *plan*, *code*, *build*, *test*, *release*, *deploy*, serta *feedback*. Pada simulasi ini akan direncanakan tiga fase pengembangan. Rangkuman simulasi perubahan *user requirement* pada tahap *plan* ketiga fase simulasi dapat dilihat pada Tabel 1.

Dari Tabel 1 dapat dilihat bahwa *requirement* baru untuk setiap fasenya ditunjukkan dengan huruf miring dan latar belakang abu-abu. Pada fase pertama LED akan berkedip ketika tombol *push button* ditekan. Sementara fitur pengesetan koneksi wifi melalui peramban dimunculkan pada fase kedua. Pada fase ketiga perilaku LED berubah dari berkedip menjadi berubah keadaannya bergantian dari mati, menyala redup dan menyala terang ketika tombol ditekan. Berdasarkan *user requirement* tersebut, selanjutnya dibuat *use case diagram* seperti yang terlihat pada Tabel 1 dan *class diagram*. Tahap selanjutnya adalah tahap pembuatan kode. Pembuatan kode dilakukan menggunakan *PlatformIO* dengan menggunakan *platform* Espressif8266 dan *framework* Arduino. Penggunaan *PlatformIO* sebagai lingkungan pengembangan karena kemudahan integrasinya dengan GitHub sebagai sistem *versioning* yang memungkinkan kolaborasi dengan mudah dan Travis CI untuk pengujian dan *deplyoment* otomatis.

2.4 Penerapan Mekanisme *Firmware Update Over The Air*

Mekanisme pembaharuan *firmware over the air* menggunakan *library* ESP8266 *httpUpdate*. Pada prinsipnya perangkat tertanam akan melakukan pengecekan dan pengunduhan *firmware* rilis terbaru dari *repository* GitHub melalui skrip PHP pada *server*. Informasi keberadaan *firmware* versi terbaru dapat dilihat oleh pengguna melalui aplikasi Blynk pada perangkat *smartphone*. Pada aplikasi Blynk terdapat tombol untuk menjalankan pembaharuan *firmware* ketika terdapat versi *firmware* yang baru. Ketika pengguna menekan tombol tersebut, maka perangkat tertanam akan menjalankan mekanisme *firmware update over the air* dengan memanggil skrip php pada *server*. Pada prinsipnya perangkat tertanam akan menggunakan *server* dengan skrip PHP tersebut sebagai perantara.

3. HASIL DAN PEMBAHASAN

Pada uji coba, pengembang akan membuat kode sumber yang akan dorong ke *repository* GitHub. Setelah kode sumber diterima oleh GitHub, secara otomatis akan ada pemicu Travis CI untuk menjalankan proses *build and test*, *release* dan *deploy*. Kode sumber yang belum stabil akan dirilis dengan status *unstable* dan dipasang pada perangkat tes. Fitur-fitur yang diminta pada tahapan *user requirement* dicoba seluruhnya. Ketika versi kode sumber tersebut sudah dinyatakan lolos tahap pengujian pada perangkat keras, maka kode tersebut akan dirilis dengan status *stable* dan dipasang pada perangkat operasional. Ketiga fase pengembangan yang terlihat pada Tabel 1 berhasil diterapkan seluruhnya secara bertahap pada perangkat keras.

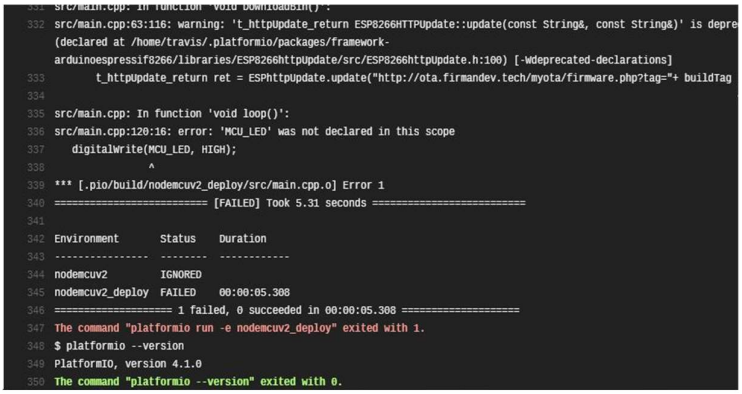
3.1 *Automatic Test* pada Pengembangan

Automatic test adalah pengujian yang dilakukan untuk menguji keberhasilan automasi tahapan proses sistem. Tahapan proses yang diuji dimulai dari tahap pengembang membuat kode sumber yang dilanjutkan dengan mendorong kode baru tersebut ke GitHub hingga tahap *deploymet* ke perangkat ESP8266. Uji ini dilakukan sebanyak tujuh kali dengan melakukan modifikasi terhadap kode sumber.

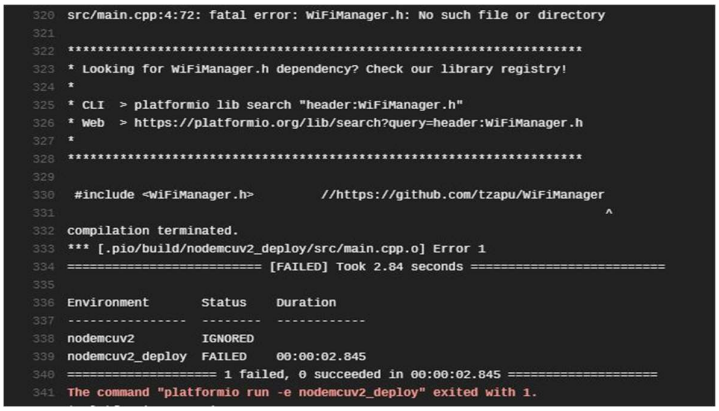
Dari ketujuh uji tersebut, terdapat dua kali simulasi kegagalan dengan melakukan kesalahan pada penulisan kode sumber. Tabel 2 menunjukkan hasil dari ketujuh uji yang dilakukan. Baris pertama, kedua, keempat, kelima dan keenam menunjukkan keberhasilan proses, sedangkan baris ketiga dan baris ketujuh menunjukkan kegagalan proses. Keberhasilan proses diperoleh dari kode sumber yang benar. Dengan menggunakan kode sumber yang benar, sistem telah berhasil melakukan proses *build and test*, *release* serta *deploy* secara otomatis melalui Travis CI. Hasil kompilasi biner *firmware* akan disimpan secara otomatis pada *repository* GitHub.

Tabel 2. Hasil *Automatic Test*

No	*Commits Number*	*Build &Test*	*Release*	*Deploy*
1	7560887	*Auto*	*Auto*	*Auto*
2	1079c36	*Auto*	*Auto*	*Auto*
3	181842c	*Failed*	*Failed*	*Failed*
4	77998ac	*Auto*	*Auto*	*Auto*
5	0809c7a	*Auto*	*Auto*	*Auto*
6	a8d19d6	*Auto*	*Auto*	*Auto*
7	bc900c7	*Failed*	*Failed*	*Failed*



Gambar 3. Kegagalan yang Disebabkan karena Penggunaan Variabel yang Belum Dideklarasikan.



Gambar 4. Kegagalan yang Disebabkan karena Kesalahan Penulisan *Library* yang Digunakan.

Kegagalan proses pada baris ketiga dan ketujuh diperoleh dari simulasi kesalahan penulisan kode sumber. Kegagalan pada baris ketiga disebabkan penggunaan variabel yang belum dideklarasikan sebagaimana terlihat pada Gambar 3. Sedangkan kesalahan pada baris ketujuh disebabkan karena kesalahan dalam penulisan *library* yang mengakibatkan sistem tidak dapat menemukan *library* yang digunakan. Simulasi kegagalan ini ditunjukkan pada Gambar 4. Berdasarkan laporan di atas, kesalahan terjadi karena tidak telitinya pengembang dalam menuliskan kode program.

Tabel 3. Hasil *Lead Time Test*

Tahapan Proses	*Lead Time (s)*				
	Commit-7560887	***Commit-1079c36***	***Commit-a8d19d6***	***Commit-77998ac***	***Commit-0809c7a***
Build & Test	71.14	69.82	***71.53***	69.64	***69.53***
Release	***5.86***	***5.18***	5.47	5.36	5.47
Deploy	1.12	1.56	***1.86***	***1.09***	1.42
Total (s)	78.12	76.56	***78.86***	***76.09***	76.42

3.2 *Lead Time Test*

Dari ketujuh pengujian yang dilakukan dengan data yang terlihat pada Tabel 2, lima pengujian menunjukkan keberhasilan *automatic test*. Kelima pengujian yang berhasil tersebut diukur waktu yang dibutuhkan oleh setiap tahapan proses untuk melakukan *lead time test*. Pengujian ini hanya dapat dilakukan pada proses yang berhasil dari Tabel 2, karena proses yang gagal tidak akan diproses lebih lanjut oleh sistem. Waktu masing-masing tahapan tersebut dapat dilihat pada Tabel 3. Terlihat untuk uji coba yang dilakukan total waktu yang dibutuhkan berkisar antara 76,09 detik hingga 78,86 detik dengan rerata waktu total 77,21 detik. Secara keseluruhan waktu terbanyak digunakan untuk melakukan proses *build and test* (berkisar antara 69,53 detik hingga 71,53 detik). Lama waktu keseluruhan tahapan tersebut tentunya bergantung pada ukuran kode sumber yang digunakan serta ketersediaan sumber daya peladen pada Travis CI. Waktu yang dibutuhkan untuk *deployment* ke perangkat keras berkisar antara 1,09 detik hingga 1,86 detik dengan rerata waktu *deployment* sebesar 1,41 detik. Waktu *deployment* tersebut juga dipengaruhi oleh kondisi jaringan.

4. KESIMPULAN

Dari hasil uji coba terlihat bahwa infrastruktur *DevOps* yang diusulkan untuk melakukan pengembangan sistem tertanam berbasis ESP8266 telah dapat mendeteksi kesalahan dan berhasil melakukan proses hingga *deployment* ke perangkat. *Lead time test* menunjukkan waktu terpanjang terjadi pada proses *build and test* (rata-rata 77,21 detik). Sedangkan rerata waktu yang dibutuhkan untuk proses *deployment* ke perangkat sebesar 1,41 detik. Hal ini menunjukkan bahwa proses *upgrade firmware* ke perangkat cukup singkat. Hasil ini menunjukkan bahwa penerapan *DevOps* pada sistem tertanam berbasis *microcontroller* terutama ESP8266 dapat dilakukan. Dengan menggunakan infrastruktur *DevOps* yang diusulkan, pengembangan sistem tertanam dapat dilakukan secara cepat. Perubahan *user requirement* dapat dengan cepat ditanggapi oleh pengembang serta dapat segera diterapkan pada sistem tertanam. Penggunaan ESP8266 pada infrastruktur yang diusulkan cukup cocok digunakan untuk perangkat-perangkat IoT yang terhubung dengan jaringan melalui wifi dengan beban komputasi yang tidak terlalu besar, seperti saklar, lampu, *power meter* maupun pemantauan parameter lingkungan (seperti suhu dan kelembaban). Penggunaan pada *microcontroller* jenis lain masih perlu diuji lebih lanjut.

UCAPAN TERIMA KASIH

Terima kasih kami ucapkan kepada LPPM Unsoed, FT Unsoed, TE Unsoed dan Laboratorium Komputer FT Unsoed serta Laboratorium Elektronika FT Unsoed yang telah memungkinkan terwujudnya penelitian ini. Penelitian ini juga didukung oleh skim Penelitian Peningkatan Kompetensi BLU Unsoed 2020.

DAFTAR RUJUKAN

Bae, J., Kim, C., & Kim, J. (2016). Automated deployment of SmartX IoT-cloud services based on continuous integration. *2016 International Conference on Information and Communication Technology Convergence (ICTC)*, (pp. 1076–1081). https://doi.org/10.1109/ICTC.2016.7763372

Bonomi, F., Milito, R., Natarajan, P., & Zhu, J. (2014). Fog Computing: A Platform for Internet of Things and Analytics. Dalam N. Bessis & C. Dobre (Ed.), *Big Data and Internet of Things: A Roadmap for Smart Environments, 546*, 169–186. Springer, Cham. https://doi.org/10.1007/978-3-319-05029-4_7

Botta, A., Donato, W. de, Persico, V., & Pescapé, A. (2016). Integration of Cloud computing and Internet of Things: A survey. *Future Generation Computer Systems*, *56*, 684–700. https://doi.org/10.1016/j.future.2015.09.021

Callanan, M., & Spillane, A. (2016). DevOps: Making It Easy to Do the Right Thing. *IEEE Software*, *33*(3), 53–59. https://doi.org/10.1109/MS.2016.66

Chandra, H., Anggadjaja, E., Wijaya, P. S., & Gunawan, E. (2016). Internet of Things: Over-the-Air (OTA) firmware update in Lightweight mesh network protocol for smart urban development. *2016 22nd Asia-Pacific Conference on Communications (APCC)*, (pp. 115–118). https://doi.org/10.1109/APCC.2016.7581459

Chopra, K., Gupta, K., & Lambora, A. (2019). Future Internet: The Internet of Things-A Literature Review. *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)*, (pp. 135–139). https://doi.org/10.1109/COMITCon.2019.8862269

Doddapaneni, K., Lakkundi, R., Rao, S., Kulkarni, S. G., & Bhat, B. (2017). Secure FoTA Object for IoT. *2017 IEEE 42nd Conference on Local Computer Networks Workshops (LCN Workshops)*, (pp. 154–159). https://doi.org/10.1109/LCN.Workshops.2017.78

Fowler, M. (2006, Mei 1). Continuous Integration. *Https://Martinfowler.Com/*. https://martinfowler.com/articles/continuousIntegration.html

Frisch, D., Reißmann, S., & Pape, C. (2017). An over the air update mechanism for ESP8266 microcontrollers. *Proceedings of the ICSNC, the Twelfth International Conference on Systems and Networks Communications, Athens, Greece*, (pp. 8–12).

Gore, S., Kadam, S., Mallayanmath, S., & Jadhav, S. (2017). Review on Programming ESP8266 with Over the Air Programming Capability. *International Journal of Engineering Science and Computing (IJESC)*, *7*(4), 6684–6686.

Gubbi, J., Buyya, R., Marusic, S., & Palaniswami, M. (2013). Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, *29*(7), 1645–1660. https://doi.org/10.1016/j.future.2013.01.010

Guşeilă, L. G., Bratu, D., & Moraru, S. (2019). Continuous Testing in the Development of IoT Applications. *2019 International Conference on Sensing and Instrumentation in IoT Era (ISSI)*, (pp. 1–6). https://doi.org/10.1109/ISSI47111.2019.9043692

Hessar, M., Najafi, A., Iyer, V., & Gollakota, S. (2020). TinySDR: Low-Power SDR Platform for Over-the-Air Programmable IoT Testbeds. *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, (pp. 1031–1046). https://www.usenix.org/conference/nsdi20/presentation/hessar

Hüttermann, M. (2012). Beginning DevOps for Developers. Dalam *DevOps for Developers* (pp. 3–13). Apress, Berkeley, CA. https://link.springer.com/chapter/10.1007/978-1-4302-4570-4_1

Jurković, G., & Sruk, V. (2014). *Remote firmware update for constrained embedded systems*. 1019–1023. https://doi.org/10.1109/MIPRO.2014.6859718

Kerliu, K., Ross, A., Tao, G., Yun, Z., Shi, Z., Han, S., & Zhou, S. (2019). Secure Over-The-Air Firmware Updates for Sensor Networks. *2019 IEEE 16th International Conference on Mobile Ad Hoc and Sensor Systems Workshops (MASSW)*, (pp. 97–100). https://doi.org/10.1109/MASSW.2019.00026

Khemissa, S. (2018). *Recommendations for IoT Firmware Update Processes*. Cloud Security Alliance. https://downloads.cloudsecurityalliance.org/assets/research/internet-of-things/recommendations-for-iot-firmware-update-processes.pdf

Kurniawan, M. I., Sunarya, U., & Tulloh, R. (2018). Internet of Things: Sistem Keamanan Rumah berbasis Raspberry Pi dan Telegram Messenger. *ELKOMIKA: Jurnal Teknik Energi Elektrik, Teknik Telekomunikasi, & Teknik Elektronika*, *6*(1), 1–15. https://doi.org/10.26760/elkomika.v6i1.1

Lwakatare, L. E., Karvonen, T., Sauvola, T., Kuvaja, P., Olsson, H. H., Bosch, J., & Oivo, M. (2016). Towards DevOps in the Embedded Systems Domain: Why is It So Hard? *2016 49th Hawaii International Conference on System Sciences (HICSS)*, (pp. 5437–5446). https://doi.org/10.1109/HICSS.2016.671

Moore, J., Kortuem, G., Smith, A., Chowdhury, N., Cavero, J., & Gooch, D. (2016). DevOps for the Urban IoT. *Proceedings of the Second International Conference on IoT in Urban Space*, (pp. 78–81). https://doi.org/10.1145/2962735.2962747

Nilsson, D. K., & Larson, U. E. (2008). Secure Firmware Updates over the Air in Intelligent Vehicles. *ICC Workshops - 2008 IEEE International Conference on Communications Workshops*, (pp. 380–384). https://doi.org/10.1109/ICCW.2008.78

Nilsson, D. K., Sun, L., & Nakajima, T. (2008). A Framework for Self-Verification of Firmware Updates over the Air in Vehicle ECUs. *2008 IEEE Globecom Workshops*, (pp. 1–5). https://doi.org/10.1109/GLOCOMW.2008.ECP.56

Prens, D., Alfonso, I., Garcés, K., & Guerra-Gomez, J. (2019). Continuous Delivery of Software on IoT Devices. *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, (pp. 734–735). https://doi.org/10.1109/MODELS-C.2019.00112

Rehman, H. U., Asif, M., & Ahmad, M. (2017). Future applications and research challenges of IOT. *2017 International Conference on Information and Communication Technologies (ICICT)*, (pp. 68–74). https://doi.org/10.1109/ICICT.2017.8320166

Santoso, H. B., Prasojo, S., & Mursid, S. P. (2018). Pengembangan Sistem Pemantauan Konsumsi Energi Rumah Tangga Berbasis Internet of Things (IoT). *ELKOMIKA: Jurnal Teknik Energi Elektrik, Teknik Telekomunikasi, & Teknik Elektronika*, *6*(3), 357. https://doi.org/10.26760/elkomika.v6i3.357

Tahtawi, A. R. A., Hendrawati, T. D., Abdurrahim, A., & Andika, E. (2019). Perancangan dan Analisis Kinerja Sistem Kontrol dan Penjadwalan Lampu Berbasis IoT. *ELKOMIKA: Jurnal Teknik Energi Elektrik, Teknik Telekomunikasi, & Teknik Elektronika*, *7*(3), 533. https://doi.org/10.26760/elkomika.v7i3.533

Thakur, P., Bodade, V., Achary, A., Addagatla, M., Kumar, N., & Pingle, Y. (2019). Universal Firmware Upgrade Over-The-Air for IoT Devices with Security. *2019 6th International Conference on Computing for Sustainable Global Development (INDIACom)*, (pp. 27–30).

Toro-Betancur, V., Zamora, J. V., Antikainen, M., & Di Francesco, M. (2019). A Scalable Software Update Service for IoT Devices in Urban Scenarios. *Proceedings of the 9th International Conference on the Internet of Things*. https://doi.org/10.1145/3365871.3365880

Yigitoglu, E., Mohamed, M., Liu, L., & Ludwig, H. (2017). Foggy: A Framework for Continuous Automated IoT Application Deployment in Fog Computing. *2017 IEEE International Conference on AI Mobile Services (AIMS)*, (pp. 38–45). https://doi.org/10.1109/AIMS.2017.14

Zandberg, K., Schleiser, K., Acosta, F., Tschofenig, H., & Baccelli, E. (2019). Secure Firmware Updates for Constrained IoT Devices Using Open Standards: A Reality Check. *IEEE Access*, *7*, 71907–71920. https://doi.org/10.1109/ACCESS.2019.2919760