Jurnal Teknologi Informasi dan Ilmu Komputer (JTIIK) DOI: 10.25126/jtiik.202295750
Vol. 9, No. 3, Juni 2022, hlm. 647-654 p-ISSN: 2355-7699
Akreditasi KEMENRISTEKDIKTI, No. 36/E/KPT/2019 e-ISSN: 2528-6579

PENERAPAN MEKANISME *CONTINUOUS DEPLOYMENT* DALAM PENGEMBANGAN DAN PEMBARUAN PERANGKAT LUNAK SISTEM BENAM BERBASIS INTERNET OF THINGS

Yohanna Fransiska Aladina[1], Adhitya Bhawiyuga*[2], Reza Andria Siregar[3], Primantara Hari Trisnawan[4]

[1, 2, 3, 4] Universitas Brawijaya, Malang
Email: [1]yfransiskaas@gmail.com, [2]bhawiyuga@ub.ac.id, [3]reza.jalin@ub.ac.id, [4]prima@ub.ac.id
[*] Penulis Korespondensi

(Naskah masuk: 29 Oktober 2021, diterima untuk diterbitkan: 02 Juni 2022)

Abstrak

Sebuah sistem berbasis *Internet of Things* (IoT) umumnya terdiri atas perangkat sistem benam yang saling terhubung antara satu dengan lainnya melalui jaringan internet. Jumlah perangkat IoT selalu bertambah. Namun selama ini, proses pembaruan perangkat IoT memiliki permasalahan yang serius terkait biaya dan waktu, yaitu ketika hendak melakukan pembaruan *software*, manusia harus datang secara langsung ke lokasi di mana perangkat IoT tersebut berada. Dari permasalahan tersebut, maka dibuatlah penelitian ini mengenai pembaruan *software* pada perangkat IoT menggunakan *continuous deployment* yang dilakukan di *cloud*. *Continuous deployment* adalah proses penyebaran berkelanjutan pada *software* yang memanfaatkan proses otomatisasi yang dilakukan dari awal hingga akhir berjalan secara otomatis tanpa adanya campur tangan manusia. Proses ini memberikan dampak positif yang signifikan karena hanya dengan menjalankan *automated script*, lalu semuanya bisa berjalan dengan lancar sesuai apa yang diharapkan. Implementasinya dilakukan menggunakan *text editor* yaitu Visual Studio Code, *code repository* yaitu GitHub, layanan *cloud* bernama *Amazon Web Services (AWS),* tiga buah ESP32 sebagai perangkat IoT, dan Jenkins sebagai *tools* untuk *continuous deployment* yang menjadi penghubung pada pendistrubusian *code* dari GitHub ke layanan-layanan yang dipilih di AWS untuk mendukung proses *continuous deployment*. Hasil dari pengujiannya menunjukkan bahwa layanan-layanan pada AWS dapat saling terhubung serta dapat terintegrasi dengan Jenkins dan ESP32, sehingga dapat melakukan pembaruan *code* dari GitHub ke tiga buah ESP32 dengan proses otomatisasi sepenuhnya. Waktu rata-rata yang dibutuhkan untuk melakukan pembaruan hanyalah 63.5 detik. Berdasarkan hasil pengujian tersebut, solusi ini dapat menjadi jawaban dari permasalahan pembaruan *software* perangkat IoT yang selama ini masih dilakukan secara manual.

Kata kunci: *Cloud, Continuous Deployment,* IoT, Otomatisasi, *Software Update.*

CONTINUOUS DEPLOYMENT IMPLEMENTATION FOR AUTOMATION SOFTWARE UPDATES ON CLOUD-BASED IOT DEVICES

Abstract

An Internet of Things (IoT)-based system generally consists of embedded system devices connected through the internet network. The number of IoT devices is always growing. But so far, the process of updating IoT devices has serious problems related to cost and time, namely, when they want to update software, humans must come directly to the location where the IoT device is located. This research on software updates on IoT devices using continuous deployment is carried out in the cloud from these problems. Continuous deployment is a continuous deployment process on software that utilizes an automation process carried out from start to finish running automatically without any human intervention. This process has a significant positive impact because only by running the automated script, then can everything run smoothly as expected. The implementation is carried out using a text editor, namely Visual Studio Code, a code repository, namely GitHub, a cloud service called Amazon Web Services (AWS), three ESP32s as IoT devices, and Jenkins as tools for continuous deployment that serve as a liaison for code distribution from GitHub to other services. Services selected on AWS to support the continuous deployment process. The test results show that services on AWS can be interconnected and integrated with Jenkins and ESP32 so that they can update code from GitHub to three ESP32s with a fully automated process. The average time it takes to perform an update is only 63.5 seconds. Based on the results of these tests, this solution can answer the problem of updating IoT device software which is still done manually.

Keywords: *Cloud, Continuous Deployment, IoT, Automation, Software Update.*

1. PENDAHULUAN

Jumlah perangkat *Internet of Things* (IoT) semakin banyak dan berkembang ke berbagai bidang sejak awal ide IoT ditemukan dan diimplementasikan. Seiring berkembangnya zaman, kompleksitas sistem dan penerapan *software* IoT perlu berurusan dengan pengelolaan versi *software* yang berbeda dan ketergantungan antara komponen *software* dengan suatu perangkat atau sistem (Hernandez-Ramos, Jose L et al., 2020). Terdapat salah satu pendekatan untuk melakukan pembaruan *software*, yaitu pendekatan *agile* yang merupakan pendekatan berulang dengan waktu terbatas untuk mengembangkan *software* secara bertahap dari awal hingga akhir (Niu, Nan et al., 2018). Ketika hal ini diimplementasikan pada perangkat IoT, maka kebutuhan pengembangannya bisa menjadi lebih cepat dan lebih efektif jika bisa dilakukan dengan memanfaatkan salah satu teknologi masa depan seperti *continuous deployment*, yang merupakan proses otomatisasi penerapakan aplikasi ke lingkungan produksi secara terus-menerus (Mysari, Sriniketan. & Bejgam, Vaibhav., 2020).

Terdapat berbagai macam penelitian-penelitian sebelumnya yang telah membahas tentang IoT *update.* Penelitian pertama adalah dengan menggunakan metode *Low-Power Wide Area Network* (LPWAN) sebagai teknologi jaringan IoT jarak jauh dan menggunakan *mobile edge cloud* untuk meningkatkan efisiensi komputasi dalam jaringan perangkat IoT dengan sumber daya yang tidak mencukupi. Hasil dari penelitian pertama yang dilakukan oleh Kim, Dae-Young et al., 2018 ini menyatakan bahwa perangkat LPWAN berhasil dengan mudah untuk melakukan *update* IoT. Penelitian kedua adalah penelitian yang dilakukan oleh Bui, Ngoc Hai et al., 2019 yang mengusulkan mekanisme *update* menggunakan algoritma untuk memperkirakan jadwal optimal dalam memperbaharui semua perangkat di seluruh jaringan IoT untuk meminimalkan konsumsi energi, sambil memenuhi batasan *deadline* untuk memperbaharui semua perangkat. Mereka memeriksa tiga algoritma topologi (*tree, patial mesh,* dan *full mesh*) di jaringan berbeda, hasilnya menggambarkan bahwa algoritma tersebut dapat memperoleh hasil yang optimum.

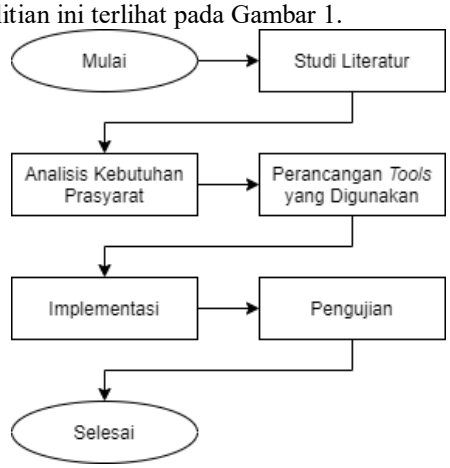
Penelitian ketiga adalah penelitian yang mengusulkan strategi baru untuk memberbarui kode yang digunakan untuk perangkat IoT pada *energy-harvesting* berdasarkan pembaruan kode *in-place* dan *code-trampolines* oleh Zhang, Chi et al., 2016. Penelitian ketiga ini secara efektif menghilangkan waktu henti sistem dan meminimalkan permintaan sumber daya untuk pembaruan sampai sebesar 99% pada memori *nonvolatile* dan 78% pada transmisi kode. Penelitian keempat yang menjadi rujukan atas dasar dari penelitian ini adalah yang dilakukan oleh Guseila, Ligia Georgeta et al., 2019 yang membahas tentang proses, metode, dan alat untuk melakukan *continuous integration, deployment,* dan *continuous testing* dalam proses pengembangan *software* berbasis *agile* pada perangkat IoT. Hasil dari penelitian keempat ini adalah prosesnya berhasil dilakukan secara otomatis, berkelanjutan, serta *software* yang disebarkan kepada *end-user* menghasilkan kualitas yang tinggi, stabil, dan *robust*.

Ketika telah menemukan sebuah solusi berupa *continuous deployment,* ternyata masih terdapat permasalahan lain, yaitu tentang bagaimana caranya agar proses pembaruan bisa dilakukan di mana saja (tidak terbatas pada lokasi geografis). Sehingga muncullah solusi yang dinamakan *cloud computing*, merupakan model untuk memungkinkan akss jaringan dimana-mana dan dapat dirilis dengan cepat (NIST., 2011). Sehingga berdasarkan apa yang telah dijelaskan, penulis melakukan penelitian tentang implementasi pembaruan *software* pada perangkat IoT menggunakan *continuous deployment* yang diterapkan di *cloud.*

Untuk melakukan implementasinya, maka diperlukan adanya perancangan dan implementasi layanan-layanan yang dapat terintegrasi dari proses awal *code* dibuat pada perangkat pengguna, hingga *code* diterima secara otomatis oleh perangkat IoT. Layanan-layanan tersebut tidak hanya yang berada pada layanan *cloud* saja, namun juga *tools* maupun servis di luar *cloud* karena layanan *cloud* tidak memiliki *code repository* sendiri. Tujuannya adalah agar ketika terdapat pembaruan *code* untuk perangkat IoT yang ingin dikembangkan dan dikirimkan ke pengguna akhir melalui *cloud,* maka proses pembaruan dapat dilakukan otomatis secara terus-menerus dan terintegrasi sepenuhnya tanpa campur tangan manusia, menjadi lebih cepat, bekerja lebih maksimal, dan dapat menurunkan tingkat kegagalan yang disebabkan oleh *human error*.

2. METODE PENELITIAN

Tahapan penelitian yanng dilakukan pada penelitian ini terlihat pada Gambar 1.



Gambar 1 Diagram alir metode penelitian

2.1 Studi Literatur

Studi literatur merupakan tahapan yang dibutuhkan oleh penulis agar dapat memperdalam pengetahuan tentang konsep-konsep dan cara pengimplementasian ide yang akan dibuat. Pengetahuan tersebut diperoleh dari jurnal ilmiah dan sumber informasi yang berasal dari internet. Terdapat tujuh dasar teori yang menjadi acuan, yaitu IoT, perangkat IoT, otomatisasi, *software*, *software update,* serta pendekatan pembaruan *software* (yaitu *continuous integration, continuous delivery, dan continuous deployment*) yang berguna agar penulis memiliki landasan yang kuat untuk melakukan penelitian.

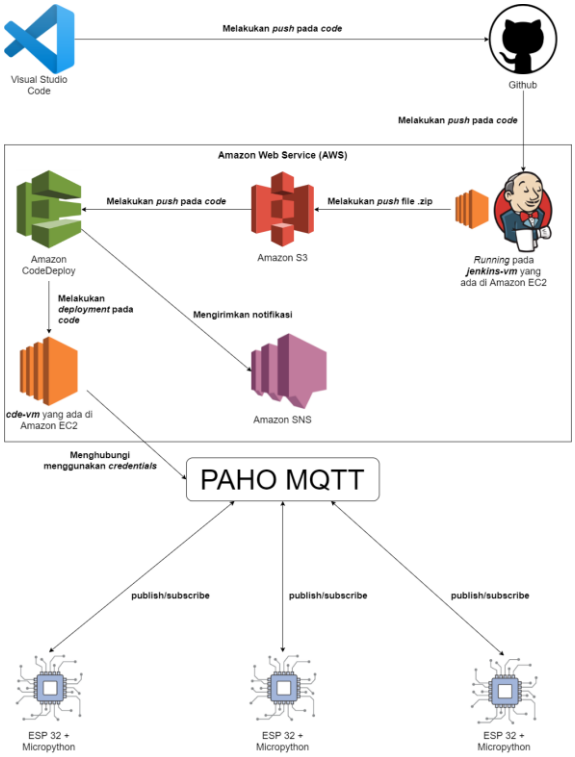
2.2 Analisis Kebutuhan Prasyarat

Tahapan analisis kebutuhan prasyarat digunakan untuk menganalisis kebutuhan-kebutuhan apa saja yang diperlukan dan harus sudah disediakan sebelum melakukan implementasi. Kebutuhan prasyarat tersebut adalah:

1. *Code editor,* menggunakan Visual Studio Code sebagai tempat untuk membuat maupun mengedit *code*.
2. *Code repository*, menggunakan GitHub yang berguna untuk melakukan penyimpanan dan pendistribusian *code* yang berasal dari Visual Studio Code ke layanan *cloud* (Vasilescu, Bogdan et al., 2015),
3. Layanan *cloud,* menggunakan *Amazon Web Services* (AWS) yang berguna untuk melakukan proses pengolahan daya komputasi yang bisa diakses kapanpun dan dimanapun (Jackson, Swedha, K. & Dubey, Tanuja., 2018 dan Kotas, Charlotte et al., 2018)
4. Broker Paho-MQTT, merupakan salah satu *library* Python yang bertindak sebagai broker untuk melakukan *publish* dan *subscribe* yang mengaplikasikan protokol MQTT pada layanan *cloud* AWS dan ESP32 (Grgic, Kresimir et al., 2016).
5. Broker Mosquitto, merupakan layanan *open-source* broker untuk melakukan pengamatan tentang pengiriman data yang dilakukan menggunakan protokol MQTT.
6. Python, merupakan bahasa pemrograman yang digunakan pada *publisher* yaitu GitHub (Mehta, Amardeep et al., 2017).
7. Micropython, merupakan implementasi ulang dari bahasa pemrograman Python versi 3 yang dikhususkan untuk dijalankan pada perangkat mikrokontroler dan *embedded* yang digunakan sebagai pahasa pemrograman pada *subscriber* yaitu tiga buah ESP32 (Laroche, Thomas et al., 2018).

Berdasarkan apa yang telah dijelaskan pada poin 2.2, maka langkah selanjutnya adalah melakukan rancangan *tools* untuk bisa menjalankan alur dari kebutuhan-kebutuhan yang telah ditetapkan agar saling terintegrasi sehingga dapat menjalankan *continuous deployment.*

2.3 Perancangan *Tools* yang Digunakan



Gambar 2 Perancangan *tools* yang digunakan

Tools yang digunakan adalah Visual Studio Code untuk melakukan pembuatan atau *editing code* hingga kemudian dilakukan *commit*, GitHub sebagai *code repository*, Jenkins sebagai *automation tools* untuk menghubungkan GitHub dengan AWS, AWS beserta layanan-layanan di dalamnya (Amazon S3 sebagai tempat penyimpanan *code* yang berasal dari GitHub, Amazon CodeDeploy sebagai penghubung antara *file* yang berada pada Amazon S3 dan dihubungkan dengan Amazon EC2 menggunakan proses otomatisasi, Amazon EC2 sebagai layanan komputasi berupa *instance* untuk menyimpan *code* serta melakukan *publishing* secara otomatis, dan Amazon SNS untuk mengirim pesan notifikasi ke email *user* maupun grup yang memegang ESP32 menggunakan mekanisme *push*), dan Paho MQTT. Perancangannya dapat dilihat pada Gambar 2.

2.4 Implementasi

Implementasi merupakan tahapan yang dilakukan berdasarkan perancangan *tools* yang telah dibuat sebelumnya. Pada AWS, sebelum melakukan konfigurasi pada beberapa layanan yang akan digunakan, perlu diatur beberapa konfigurasi berupa hal-hal penting. Yaitu membuka *Identity and Access Management* (IAM) untuk membuat *user* dan *roles.* Tujuan dibuatnya *user* adalah agar nantinya proses Amazon CodeDeploy dapat dijalankan sesuai

dengan *user* yang dibuat (disebut sebagai IAM *user*) dan bukan *user root.* Ketika pembuatan *user* telah berhasil, maka perlu dilakukan penyimpanan *secret access key* untuk dapat melakukan pengaturan di Amazon EC2 yang akan digunakan.

Implementasi pada Amazon S3 dilakukan dengan memberikan nama *bucket,* memilih *region,* dan mengedit *bucket policy* agar *bucket* pada Amazon S3 hanya boleh diakses oleh *user* yang berhak. Pada Amazon CodeDeploy, dilakukan pemilihan EC2/*On-premises* sebagai *compute platform* karena *code* akan di *deploy* pada Amazon EC2, membuat *deployment groups,* menambahkan IAM *role,* memilih Amazon EC2, melakukan spesifikasi *tag group,* menambahkan *triggers* berupa Amazon SNS *topics* agar dapat mengirimkan notifikasi dari hasil *deployment.*

Terdapat dua buah *instances* pada Amazon EC2 yang perlu dibuat, yaitu *cde-vm* sebagai tempat untuk melakukan *code automation* yang berasal dari GitHub ke ESP32 dan *jenkins-vm* untuk melakukan instalasi *tools* bernama Jenkins. Setelah keduanya dibuat, maka perlu dilakukan konfigurasi agar kedua *instances* dapat saling terhubung. Pada *jenkins-vm* juga perlu dilakukan konfigurasi pada *tools* Jenkins agar dapat terhubung ke GitHub. Sedangkan pada *cde-vm* dilakukan implementasi *code* untuk proses Amazon CodeDeploy dan *code* yang berasal dari GitHub sebagai *publisher*.

Konfigurasi pada Amazon SNS adalah membuat *topic, subscriptions,* dan memilih *email* serta memasukkan alamat *email* yang akan digunakan untuk mengirim notifikasi. Implementasi pada GitHub dilakukan dengan membuat *pribate repository* agar *repository* tidak bisa diakses oleh semua orang, membuat *webhooks* agar *code* dapat dikirimkan ke Jenkins yang terdapat pada *jenkins-vm,* dan menambahkan SSH-*Key* yang sudah di *generate* sebelumnya pada *jenkins-vm* agar Jenkins dapat mengambil *code* yang ada secara otomatis.

Implementasi pada Mosquitto yang bertindak sebagai *publisher* menggunakan bahasa pemrograman Python pada *cde-vm* adalah dengan melakukan instalasi Python, pip, dan *library* Mosquitto untuk pengiriman *code* dari *cloud* ke ESP32. Implementasi instalasi Micropython *firmware* perlu dilakukan pada ESP32 dan perangkat pengguna yang digunakan untuk menjalankan ESP32, tujuannya adalah agar bisa menjalankan *script* Micropython dan dapat melihat status *deployment*. Ketika Micropython *firmware* telah berhasil dilakukan instalasi, maka dilakukan implementasi *code* pada *subscriber* (tiga buah ESP32) agar dapat menerima dan menjalankan pembaruan *code* secara otomatis.

2.5 Pengujian

Bagian ini membahas tentang lingkungan pengujian, kerangka pengujian, dan analisis hasil pengujian. Seperti yang ada pada Gambar 2, pengujian dimulai dari proses awal hingga akhir. Yaitu dari Visual Studio Code yang ada di perangkat pengguna, GitHub, Jenkins, layanan-layanan di AWS, sampai dengan tiga buah ESP32 yang ditempatkan di lingkungan pengujian. Protokol komunikasi yang digunakan pada pengujian adalah HTTPS, HTTP, dan MQTT. Protokol HTTPS digunakan oleh Visual Studio Code, Github, layanan-layanan di AWS, HTTP dipakai pada Jenkins, dan MQTT diterapkan pada tiga buah ESP32 ketika melakukan *publish* dan *subscribe*. Terdapat dua kerangka pengujian, yaitu:

1. Pengujian waktu

Dilakukan untuk mengetahui berapa total waktu yang diperlukan dari awal hingga akhir untuk melakukan pembaruan *code* menggunakan proses *continuous deployment.* Pengujian dilakukan pada sebuah ESP32 dengan melakukan 6 kali uji di waktu yang berbeda dalam 2 hari. Enam kali uji tersebut adalah hari pertama waktu pertama (D1T1), hari pertama waktu kedua (D1T2), hari pertama waktu ketiga (D1T3), hari kedua waktu pertama (D2T1), hari kedua waktu kedua (D2T2), dan hari kedua waktu ketiga (D2T3). Ketika melakukan pengujian waktu, pembagian waktunya dibedakan menjadi tujuh bagian. Yaitu:

a) Dari Visual Studio Code ke GitHub.
b) Dari GitHub ke Jenkins.
c) Dari Jenkins ke Amazon S3.
d) Dari Amazon S3 ke Amazon CodeDeploy.
e) Dari Amazon CodeDeploy ke Amazon EC2 yang bernama *cde-vm*.
f) Dari Amazon CodeDeploy ke Amazon SNS.
g) Dari Amazon EC2 ke ESP32 yang diamati melalui Paho-MQTT dan *shell script* pada Thonny yang berfungsi sebagai *Integrated Development Environment* (IDE) Micropython.

2. Pengujian *overhead*

Digunakan untuk mengetahui berapa penggunaan memori (RAM) dan CPU yang dihabiskan oleh masing-masing ESP32 setelah mendapatkan pembaruan *code* dengan menggunakan proses *continuous deployment*. Proses pengujian ini dilakukan pada tiga buah ESP32 untuk menguji bagaimana hasil *overhead* ketika proses pembaruan *code* dilakukan per satu kali *publish*. Kemudian juga dilakukan pembaruan *code* pada sebuah *topic* sebanyak 10 kali dalam rentang waktu 30 menit.

3. ANALISIS HASIL PENGUJIAN

Berdasarkan kerangka pengujian yang telah dijelaskan pada poin 2.5, maka analisis hasil pengujiannya adalah:

1. Pengujian waktu

Hasil dari pengujian waktu berhasil dilakukan pada sebuah ESP32 dengan menggunakan tujuh bagian yang telah dijelaskan pada nomor 1 di poin 2.5, yang nantinya dijadikan satu untuk menghitung

total waktu yang diperlukan. Hasilnya diuraikan pada Tabel 1 yang menunjukkan bahwa rata-rata waktu yang diperlukan untuk melakukan *continuous deployment* dari awal hingga akhir adalah 63.5 detik.

Tabel 1 Hasil pengujian waktu pada sebuah ESP32

Aksi	D1 T1	D1 T2	D1 T3	D2 T1	D2 T2	D2 T3
Visual Studio Code ke GitHub	10 detik	9 detik	7 detik	6 detik	9 detik	4 detik
GitHub ke Jenkins	3 detik	3 detik	3 detik	4 detik	3 detik	4 detik
Jenkins ke Amazon S3	15 detik	14 detik	13 detik	14 detik	11 detik	15 detik
Amazon S3 ke Amazon Code Deploy	11 detik	10 detik	11 detik	12 detik	11 detik	11 detik
Amazon Code Deploy ke *cde-vm*	6 detik	6 detik	6 detik	6 detik	5 detik	6 detik
Cde-vm ke ESP32	21 detik	21 detik	23 detik	24 detik	21 detik	23 detik
Total	66 detik	63 detik	63 detik	66 detik	60 detik	63 detik
Rata-rata total = 381 detik / 6 = 63.5 detik						

2. Pengujian *overhead*

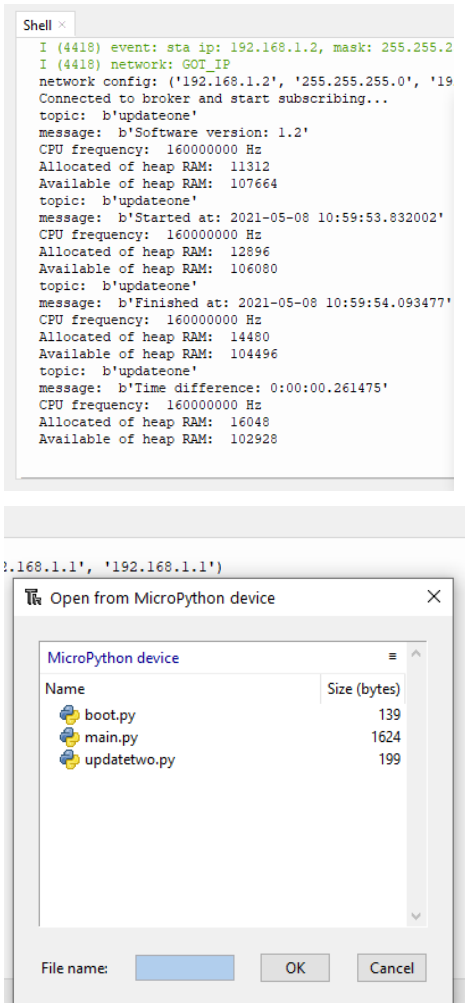
Hasil dari pengujian *overhead* dilakukan pada tiga *file* berukuran beda, yaitu 168 *bytes* pada ESP32 pertama (Gambar 3), 199 *bytes* pada ESP32 kedua (Gambar 4), dan 145 *bytes* pada ESP32 ketiga (Gambar 5) menunjukkan bahwa setiap kali ESP32 mendapatkan satu kali *publish* dari *publisher* (yaitu *cde-vm* pada AWS), maka ukuran memori RAM pada ESP32 semakin bertambah. Sedangkan tidak terdapat perbedaan pada CPU *frequency* seiring bertambahnya pesan yang di *publish* oleh *publisher*.

Kemudian hasil dari *publishing* pembaruan *code* pada sebuah *topic* sebanyak 10 kali di ESP32 dalam rentang waktu 30 menit dapat dilihat pada Gambar 6. Hasilnya menunjukkan bahwa setiap *message* yang di *publish* ke ESP32 akan menaikkan tingkat *overhead* yang ada pada ESP32.

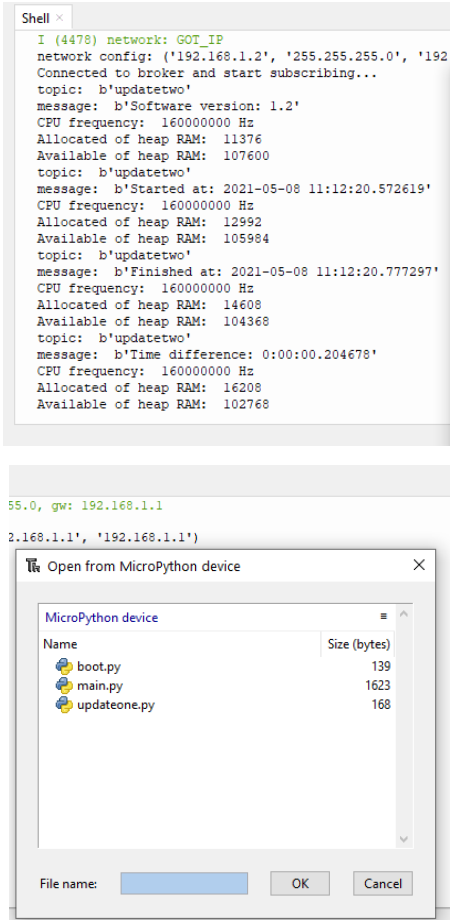
4. KESIMPULAN

Kesimpulan yang didapatkan pada penelitian ini adalah:

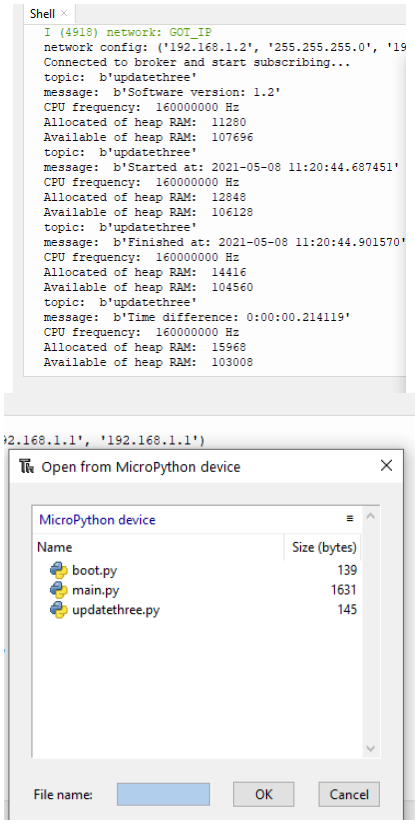
1. Implementasi *continuous deployment* untuk melakukan otomatisasi *software update* pada perangkat IoT (tiga buah ESP32) berhasil dilakukan dengan proses otomatisasi sepenuhnya. Caranya adalah dengan melakukan pemilihan *tools* yang digunakan sebagai *text editor* (Visual Studio Code dan Thonny IDE), *code repository* (GitHub) sebagai tempat untuk membuat, mengedit, menyimpan, serta membagikan *code* untuk terhubung ke penyedia *cloud*, dan pemilihan penyedia *cloud* (AWS) serta layanan-layanan di dalam *cloud* tersebut (Amazon S3, Amazon CodeDeploy, dan Amazon SNS) yang digunakan untuk dapat saling terintegrasi sehingga bisa melakukan *continuous deployment.*
2. Pengaruh implementasi *continuous deployment* pada otomatisasi *software update* pada perangkat IoT berbasis *cloud* adalah pihak yang mempunyai hak akses untuk melakukan pembaharuan hanya perlu menjalankan sebuah *code* yang berasal dari *code repository* dan ESP32 langsung menerima serta menjalankan pembaruan *code* secara otomatis dan langsung. Hal ini juga berpengaruh pada rata-rata waktu yang dibutuhkan dan tingkat *overhead*. Rata-rata waktu yang dibutuhkan oleh ESP32 untuk melakukan *continuous deployment* adalah 63.5 detik. Sedangkan tingkat *overhead* menghasilkan kesimpulan bahwa semakin banyak atau sering ESP32 menerima *message* dari *publisher* (*cde-vm* pada AWS) untuk melakukan pembaruan, maka semakin besar tingkat *overhead* memori RAM pada ESP32 yang dibutuhkan. Hal ini membuat ketersediaan memori RAM pada ESP32 semakin berkurang seiring bertambahnya proses pembaruan *code.*



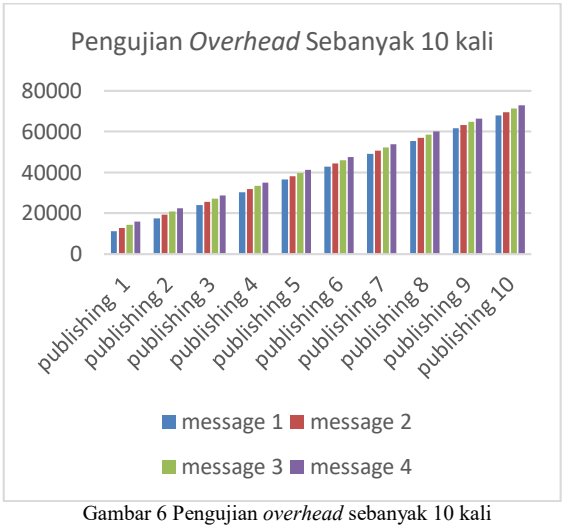
Gambar 3 Pengujian *overhead* di ESP32 pertama



Gambar 4 Pengujian *overhead* di ESP32 kedua



Gambar 5 Pengujian *overhead* di ESP32 ketiga



Gambar 6 Pengujian *overhead* sebanyak 10 kali

DAFTAR PUSTAKA

BUI, NGOC HAI., PHAM, CHUAN., NGUYEN, KIM KHOA. & CHERIET, MOHAMED., 2019. Energy Efficient Scheduling for Networked IoT Device Software Update. Canada: IEEE.

GRGIC, KRESIMIR., SPEH, IVAN. & HEDI, IVAN., 2016. A Web-based IoT Solution for Monitoring Data Using MQTT Protocol. Osijek: IEEE.

GUSEILA, LIGIA GEORGETA., BRATU, DRAGOS-VASILE. & MORARU, SORIN-AUREL., 2019. Continuous Testing in the Development of IoT Applications. Lisbon: IEEE.

HERNANDEZ-RAMOS, JOSE L., BALDINI GIANMARCO, MATHEU., SARA N. & SKARMETA, ANTONIO., 2020. Updating IoT devices: challenges and potential approaches. Dublin: IEEE.

KIM, DAE-YOUNG., KIM, SEOKHOON. & PARK, JONG HYUK., 2018. Remote Software Update in Trusted Connection of Long Range IoT Networking Integrated with Mobile Edge Cloud. South Korea: IEEE.

KOTAS, CHARLOTTE., NAUGHTON, THOMAS. & IMAM, NEENA., 2018. A Comparison of Amazon Web Services and Microsoft Azure Cloud Platforms for High Performance Computing. Las Vegas: IEEE.

LAROCHE, THOMAS., DENIS, PIERRE., PARISIS, PAUL., GEORGE, DAMIEN., DE LA LIANA, DAVID SANCHEZ. & TSIODRAS, THANASSIS., 2018. MicroPython Virtual Machine for On Board Control Procedures. Belgium: Micropython.

MEHTA, AMARDEEP., BADDOUR, RAMI., SVENSSON, FREDRIK., GUSTAFSSON, HARALD. & ELMROTH, ERIK., 2017 Calvin Constrained – A Framework for IoT Applications in Heterogeneous Environments. Atlanta: IEEE.

MYSARI, SRINIKETAN. & BEJGAM, VAIBHAV., 2020. Continuous Integration and Continuous Deployment Pipeline Automation Using Jenkins Ansible. India: IEEE.

NIST., 2011. Final Version of NIST Cloud Computing Definition Published. [Online] tersedia di <https://www.nist.gov/news-events/news/2011/10/final-version-nist-cloud-computing-definition-published> [Diakses 1 September 2020].

NIU, NAN., BRINKKEMPER, SJAAK., FRANCH, XAVIER., PARTANEN, JARI. & SAVOLAINEN, JUHA., 2018. Requirements Engineering and Continuous Deployment. Ohio: IEEE.

SHAHIN, MOJTABA., BABAR, MUHAMMAD ALI. & ZHU, LIMING., 2017. Continuous Integration, Delivery, and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices. Australia: IEEE.

SWEDHA, K. & DUBEY, TANUJA., 2018. Analysis of Web Authentication Methods Using Amazon Web Services. Bengaluru: IEEE.

VASILESCU, BOGDAN., YU, YUE., WANG, HUAIMIN., DEVANBU, PREMKUMAR. & FILKOV, VLADIMIR., 2015. Quality and Productivity Outcomes Relating to Continuous Integration in GitHub. Davis: ACM.

ZHANG, CHI., AHN, WONSUN., ZHANG, YOUTAO. & CHILDERS, BRUCE R., 2016. Live Code Update for IoT Devices in Energy Harvesting Environments. South Korea: IEEE.

Halaman ini sengaja dikosongkan