Jurnal Teknologi Elektro, Universitas Mercu Buana
ISSN: 2086-9479

Rancang Bangun Prototipe Sistem Kendali Terdistribusi Instalasi Penerangan pada Gedung 3 Lantai berbasis *IoTaaS* (*Internet of Things as a Service*) menggunakan *Docker Container*

Zendi Iklima
Fakultas Teknik/Teknik Elektro
Universitas Mercu Buana
Jakarta, Indonesia
zendi.iklima@mercubuana.ac.id

***Abstrak* — Berkembangnya *Cloud Computing* dan *Internet of Things (IoT)* mengubah cara memahami suatu infomasi dan sistem komunikasi untuk saling berhubungan menggunakan suatu infrastruktur *cloud* jarak jauh. Lain dari itu, skalabilitas sistem sering difokuskan dalam *low-level architecture*. Sistem kendali terdistribusi dapat saling terhubung pada suatu platform *IoTaaS (Internet of Things as a Service)*. *IoTaaS* divirtualisasi dengan basis kontainer (*container-based*) guna meningkatkan layanan *IoT Cloud* atau *IoT Microservice*. *IoTaaS* dikembangkan pada *Docker Container* dimana telah layak diaplikasikan pada *nodes/gateway* misalnya *payment gateway, fog, microservices* dan lain-lain. Maka dari itu pengembangan *IoTaaS* telah berhasil diaplikasikan pada sistem instalasi penerangan untuk gedung perkantoran 3 lantai. Sehingga didapatkan nilai rata-rata *Network* I/O sebesar 14.02KB/4.59KB (*download/upload*) sehingga memungkinkan tiap *container* mengirimkan data sebesar 118.77 B dalam 7.02 *milisecond*.**

Kata Kunci — Cloud Computing, Docker Continer, Instalasi Penerangan, IoTaaS, Sistem Kendali Terdistribusi

I. PENDAHULUAN

Kombinasi antara *Cloud Computing* dan *Internet of Things (IoT)* memberikan peluang baru dalam meningkatkan layanan, dalam pendekatan strategis. IoT mendorong penyedia platform Cloud maupun *hybrid* untuk mengintegrasikan sistem dengan *embedded system* (perangkat tertanam / termasuk sensor dan aktuator) [1][2] untuk mengimplmentasikan hal tersebut maka dapat dilakukan dengan membangun Infrastruktur [3], Platform dan Perangkat Lunak untuk Layanan (IaaS, PaaS, SaaS) atau kombinasi yang baru, yang disebut IoT as a Service (IoTaaS) [4]. Sebagai akibatnya, tipe baru penyedia yang menggabungkan solusi *Cloud Computing* dengan IoT meningkat [5][6]. Berkembangnya *IoT Cloud* untuk menunjukkan jenis baru sistem terdistribusi yang terdiri dari satu set perangkat IoT yang saling berhubungan dengan infrastruktur *Cloud*, platform, atau system kendali jarak jauh melalui Internet yang mampu menyediakan IoTaaS.

Beberapa teknologi virtualisasi *Cloud Computing* telah berkembang salah satu teknologi paling populer adalah *hypervisor*. *Hypervisor* merupakan sebuah virtualisasi yang memerlukan modul perangkat lunak untuk memonitor mesin virtual / *Virtual Machine Monitor (VMM)* di atas *Host OS* yang menyediakan abstraksi penuh Mesin Virtual / *Virtual Machine (VM)*. Baru-baru ini, alternatif yang lebih ringan dari *hypervisor* adalah virtualisasi dikenal sebagai *container-based*, yang juga dikenal sebagai virtualisasi Level Sistem Operasi (OS). Virtualisasi semacam ini mem-partisi sumber daya mesin secara fisik, kemudian menciptakan banyak partisi contoh ruang pengguna [4][7].

Revolusi sistem *IoT Cloud* membawa keuntungan yang sama dengan virtualisasi *hypervisor* ke sistem *Cloud* tradisional dalam hal pengelolaan sumber daya. Bahkan, virtualisasi *Container* dapat memungkinkan *IoT Cloud* untuk memudahkan, memindahkan, dan mengoptimalkan kemampuan *Virtual IoT* dalam suatu teknologi yang dikembangkan yaitu IoTaaS. IoTaaS memliki skalabiliti, fleksibliliti yang baik secara khusus. Sehingga untuk menyediakan IoTaaS, Docker dapat

mevirtualiasasikan sebuah *platform container-based* untuk dapat mengaplikasikan sistem kontrol terdistribusi [8]. *Docker Container* memiliki layer yaitu *image (Read-only)* dan *container (Read-write)*. Sebuah *docker image* terdiri dari *docker image* didalamnya adalah Operating System dimana pada tahap *develop* akan dimanfaatkan sebagau *virtual server* atau *Cloud Service*. *Container* merupakan wadah dari image. Maka dari itu meskipun menggunakan environment yang sangat minim memungkinkan aplikasi dapat berjalan dengan baik [9].

Penelitian ini memanfaatkan IoTaaS sebagai Sistem Kendali Terdistribusi dengan menggunakan 4 container yang dibangun pada *hyperledger / docker container*. Sebagai implementasi yang sedehana system ini dapat melakukan kendali jarak jauh untuk bangunan 3 lantai dengan 4 NodeJS server dibangun pada container yang secara terdistribusi.

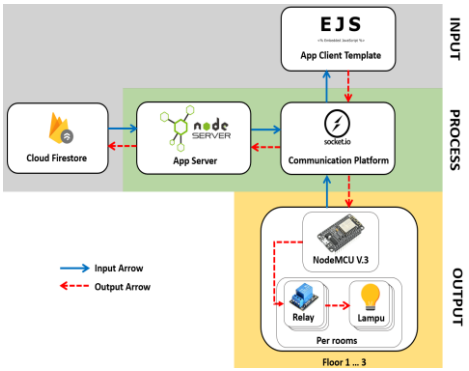
II. PENELITIAN TERKAIT

Sebagai contoh sederhana pengendalian perangkat rumah tangga / smart home telah banyak diaplikasian berdampingan dengan perkembangan teknologi IoT. Contohnya pengendalian perangkat rumah tangga (seperti lampu [10][11][15], AC [12], kipas angin [13], monitoring gas [14]) dengan menggunakan mobile phone. Metode pengiriman data yang digunakan pun beragam seperti menggunakan MIT APP Inventor, menggunakan web service atau menggunakan third party lainnya (BLINK, CAYANNE [16], dan lain-lain).

III. METODE PENELITIAN

A. *Diagram Blok Penelitian*

Interprestasi system meliputi input, proses dan output sangatlah sederhana. Diagram blok pada Gambar 1 merupakan diagram blok *per-container*. Disisi input nilai pada relay diset pada aplikasi dashboard yang ditampilkan halaman web dengan menggunakan *render engine EJS*. Selanjutnya akan diproses dan disimpan kedalam Cloud *Firestore*, kemudian melakukan perubahan pada status relay apakah *HIGH* atau *LOW*.

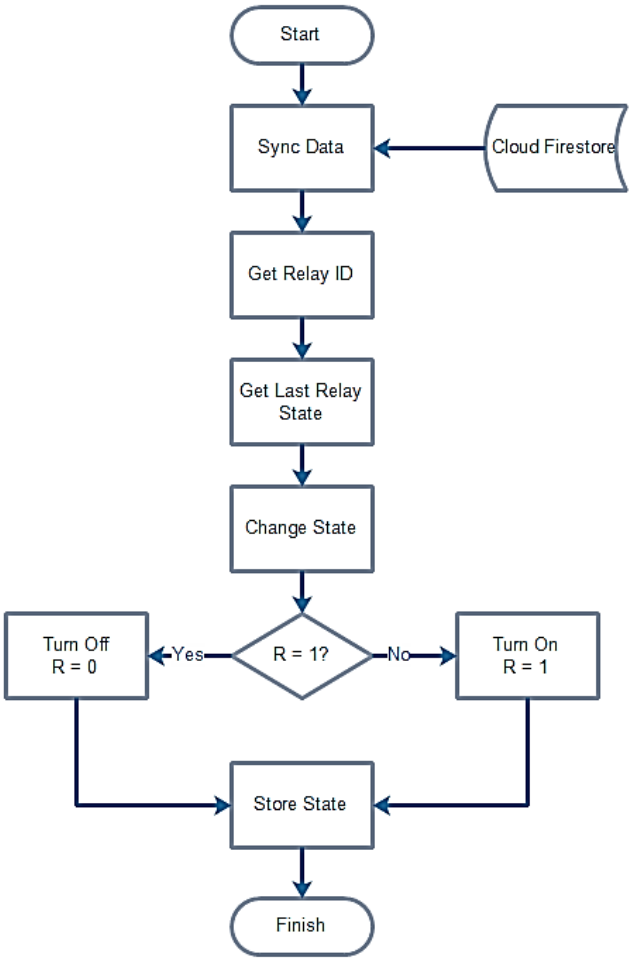


Gambar 1. Diagram Blok Sistem

Sebagai platform komunikasi data Socket.IO berjalan pada layer TCP mengatur *event* untuk menunjang sistem yang *real-time* agar semua *docker container* saling terdesentralisasi (*decentralized*).

B. *Flowchart*

Flowchart pada sistem kendali terdistribusi untuk pengendalian sistem instalasi penerangan pada Gedung 3 lantai sangatlah sederhana. Dimana seluruh *container* melakukan sinkronisasi pada data yang terdapat di *Cloud Firestore*, mendapatkan status terakhir kemudian melakukan perubahan status secara terdistribusi. Artinya data antar *container* terbuka secara menyeluruh yang memungkinkan tiap *container* mengetahui data apa saja yang ditransfer melalui Socket.IO. Data yang dimaksud berupa *floor_id, room_id, relay_id* dan status. Sehingga *event* pada *socket* mengatur transmisi data antar *container*. Perhatikan Gambar 2 dibawah ini bagaimana system dapat mengendalikan status relay HIGH atau LOW.

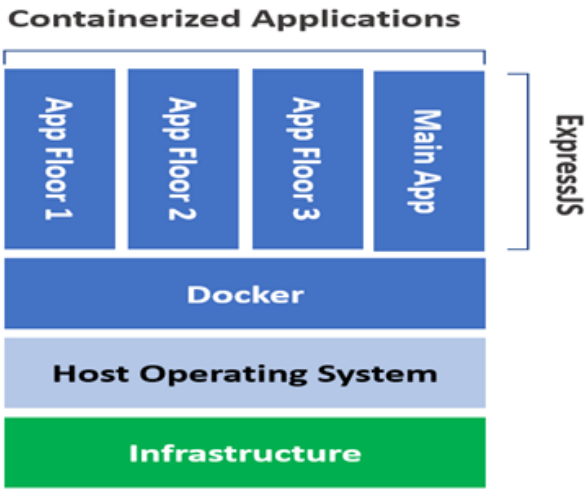


Gambar 2. Flowchart Sistem

Setelah status relay berhasil diubah, selanjutnya sistem akan menyimpan status terbaru pada *Cloud Firestore* sehingga proses akan diulang dengan melakukan sinkronsasi data.

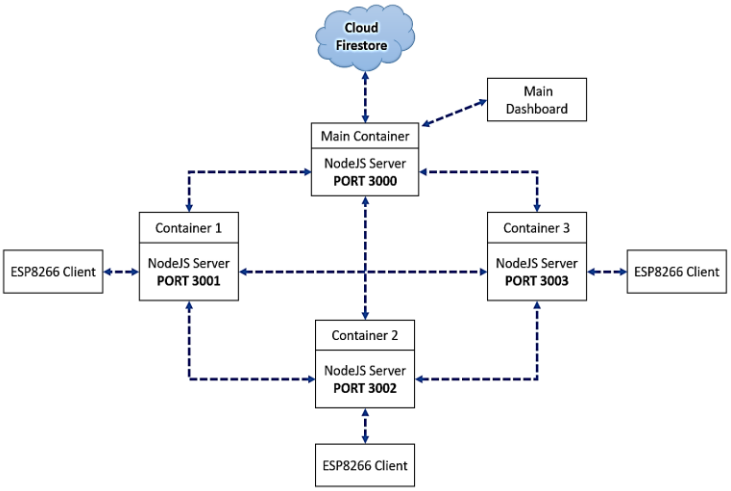
C. *Arsitektur Docker*

Container adalah unit standar perangkat lunak yang mengemas code dan semua dependensinya sehingga aplikasi berjalan dengan cepat dan andal dari satu lingkungan komputasi ke yang lain. *Container Docker* adalah paket perangkat lunak yang ringan, mandiri, dan dapat dieksekusi yang mencakup semua yang diperlukan untuk menjalankan aplikasi: kode, runtime, alat sistem, pustaka dan pengaturan sistem. *Docker Engine* dapat di install pada aplikasi berbasis *Linux* dan *Windows*. *Containerisation* merupakan suatu proses virtualisasi server local dengan alternatif yang lebih ringan yaitu menggunakan virtual server yang dibangun pada Docker. Dalam terminologi Docker, *Container* merupakan sebuah *sandbox* untuk dapat menjalankan perangkat lunak atau bahkan Sistem Operasi. Beberapa *container* dapat dibangun berdasarkan sistem operasi yang berbeda dan dapat digunakan pada satu *host machine / virtual machine*. Sebuah *container* dapat berjalan di *operating system* apapun yang mendukung Docker untuk membuat peranti lunak migrasi maupun membuat cadangan data. *Containerisation* mudah untuk dijalankan dan dipelihara pada sistem operasi ketika *virtual machine* digunakan. Maka dari itu, penggunaan *container* untuk membangun IoTaaS dapat dilakukan sebagai langkah untuk meng-inisialisasi, meng-optimasi, menjamin keandalan, skalabilitas, kualitas, kemanan pada *platform IoTaaS*. Perhatikan Gambar 3 dibawah ini bagaiman IoTaaS dibangun secara sederhana sebagai suatu langkah pengembangan IoT dan Cloud Computing.



Gambar 3. Containerisation Sistem Kendali Terdistribusi

Infastructur system memiliki kapasitas Processor Intel(R) Core(TM) i5-6300HQ CPU @2.30 GHz (4 CPUs), Memory 16GB DDR3L, GPU GeForce GTX 960M 4GB VRAM. Sehingga 4 buah aplikasi server mengekspresikan *microservice* secara spesifik Sokcet.IO menjadikan *node* pada tiap *container*. Maka dari itu tiap node saling terhubung membentuk suatu topologi *node* seperti Gambar 4 dibawah ini.

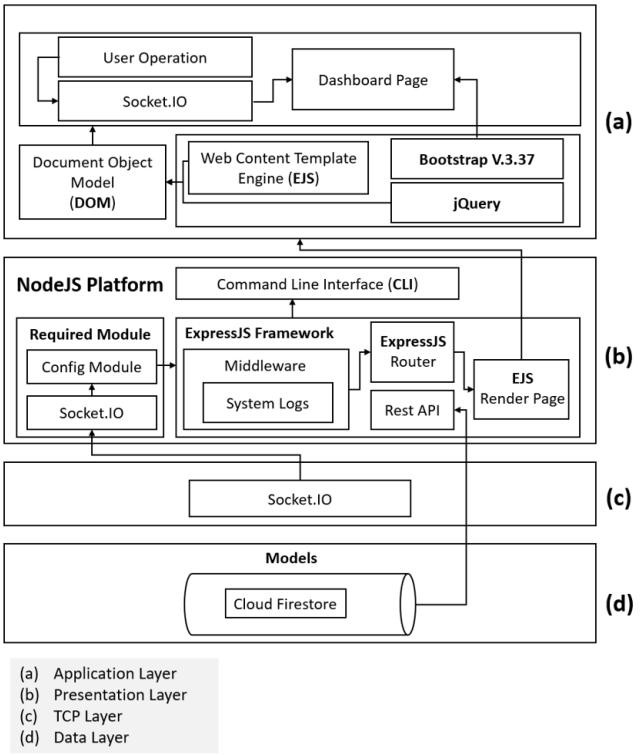


Gambar 4. Desentralisasi Container

Container pada masing-masing lantai memliki Mikrokontroler utama seperti pada Gambar 1 dimana nodeMCU akan berkomunikasi dengan *socker server* ditiap container. Maka dari itu system kendali utama untuk merubah status relay dibuat pada container yang terintegrasi secara langung dengan *Cloud Firebase*. Guna mempermudah design tiap *container*, maka penelitian ini membedakan *container* menggunakann *port* yang berbeda yaitu *port* 3000 (*Main Container*), *port* 3001 (*Container 1*), *port* 3002 (*Container 2*) dan *port* 3003 (*Container 3*).

D. *Arsitektur Aplikasi*

Sistem kendali terdistribusi untuk pengendalian sistem instalasi penerangan pada bangunan 3 lantai dibangun dengan menggunakan *microservice* pada tiap *container*. Terdapat 4 layer utama agar *microservice* dapat terdesentralisasi meliputi *application layer, presentation layer, TCP layer* dan *data layer*. Semua data diintegrasikan oleh *container* utama, sehingga *container* ini yang memiliki data layer, sementara container lainya hanya mempunya 2 layer (b) dan (c), seperti pada Gambar 5 dibawah ini.



Gambar 5. Arsitektur Sistem

Socket.IO sebagai media transmisi data berperan penting agar system desentralisasi dapat saling terintegrasi mengingat hanya *container* utama yang memiliki koneksi pada *Cloud Firestore*. Selain itu container utama juga memiliki *dashboard* pada layer (a) sehingga system kendali terdistribusi dapat melakukan kendali untuk relay pada *container* yang berbeda. Semua container berjalan diatas *NodeJS Engine* sehingga penyetaraan performa telah dilakukan disisi *environment.*

E. Instalasi Penerangan

Tiap lantai disuplai oleh listrik PLN 5.5 KVa yang di distribusikan ke bangunan yang terdiri dari 3 lantai. Instalasi penerangan dibuat dalam sebuah denah ruangan 12m X 8m yang digunakan pada Gedung perkantoran. Kebutuhan jumlah relay dibagi berdasarkan kegunaan ruangan seperti lobby, ruang meeting, ruang *staff*, *pantry*, toilet, ruang penyimpanan, *receptionist*, ruang tunggu dan lainnya. Standar iluminasi perlu diperhatikan guna menentukan berapa jumlah titik lampu yang dipasang pada ruangan tertentu, maka [17]:

$$E = \frac{\emptyset \times CU \times LLF}{A} \quad (1)$$

Dimana:

E = Tingkat iluminasi rata-rata (Lux)

$\emptyset$ = Total luas cahaya pada bidang kerja (lm)

CU = Koefisien Penggunaan

LLF = Faktor Kehilangan Chaya

A = Luas Area (m^2)

Sehingga,

$$\emptyset = L \times N \quad (2)$$

Dengan L : Total lumen awal per luminer

N : Jumlah Luminer

Maka dari itu kebutuhan banyaknya *switch/saclar* yang diganti dengan relay seperti pada Tabel 1 dibawah ini.

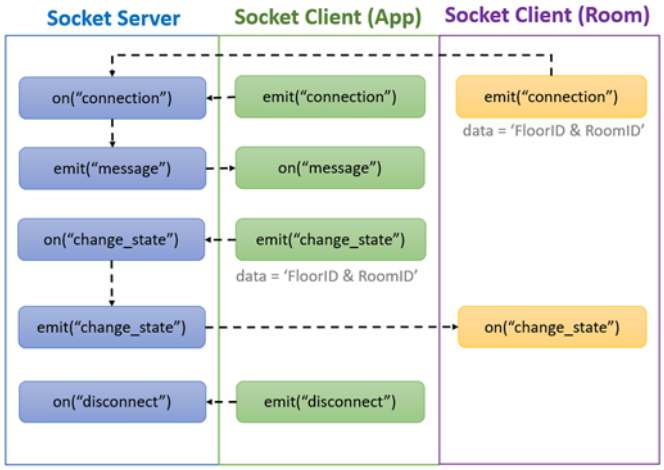
Tabel 1. Titik Relay

Lantai	Jumlah Titik Relay	Jumlah Luminer
1	8	16
2	7	17
3	6	17

Tabel 1 diatas menggambarkan jumlah titik relay yang digunakan sebagai *switch* pada tiap lantai. Sehingga pada Gambar 1 dibutuhkan 3 buah NodeMCU, 21 buah Relay dan 50 buah Lampu agar Instalasi Penerangan dapat terintegrasi dengan Sistem.

F. Socket.Io Event

Sistem terdistribusi antar *container* dibangun menggunakan platfom komunkasi Socket.IO. Terdapat 2 *event* didalam *Socket.IO* yaitu *onListenting* dan *onEmit*. *Event onListening* merupakan suatu *event* dimana menunggu data yang didistribusikan melalui *socket event*. *Event* ini akan menerima setiap aktivitas *event onEmit*. *Event onEmit* merupakan *event* dimana data dikirim melalui *socket event*. Maka dari itu pada penelitian ini *Socket.IO* diinstal pada mmasing-masing container dan 3 sisi yaitu sisi server, *client app* dan *client NodeMCU*. Dimana untuk dapat saling terintegrasi setiap *event* harus dibuat secara berpasangan *onListening* dan *onEmit*. Perhatikan Gambar 6 dibawah ini.

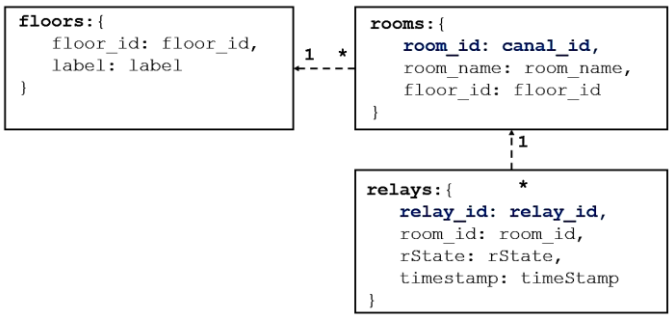


Gambar 6. Socket.IO Event

Data perubahan status relay dikirim pada *event* yang diberi label *change_state*. Sehingga event ini terkonfigurasi pada seluruh *container* untuk menunjang pengembangan teknologi pada IoTaaS.

G. Cloud Firestore Collection

Relasi pada *Cloud Firestore* direalisasikan dengan menggambarkan database menggunakan *collection*. Sehingga kebutuhan pada *Data Layer* pada Gambar 5 (d) dapat diimplementasikan dengan menggambarkan relasi antar *collection* seperti Gambar 7 dibawah ini.



Gambar 7. Cloud Firestore Collection

IV. HASIL DAN PEMBAHASAN

A. Pengujian Docker Container

Pengujian IoTaaS dilakukan per-container, mengingat terdapat 4 container seperti pada Gambar 3, sehingga didapat spesifikasi pada docker image seperti Tabel 2 dibawah ini.

Tabel 2. Pengujian Docker Container

Image	Container	Size	Ports
cmainlamp	4c877de03ade cranky_mayer	168MB	TCP/3000
clamp1	436b295fbab3 recursing_mclaren	168MB	TCP/3001
clamp2	3e34bfa5ef19 musing_keller	168MB	TCP/3002
clamp3	3878da65beff silly_golick	168MB	TCP/3003

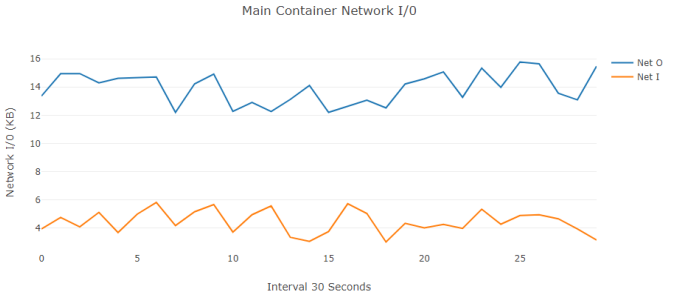
Tiap *container* diberi label seperti *Main App* pada *image* cmainlamp, *App Floor* 1 pada *image* clamp1, *App Floor* 2 pada *image* clamp2 dan *App Floor* 3 pada *image* clamp3.
Selanjutnya dilakukan pengujian *worker* tiap container meliputi pengujian performa CPU, *Network* I/O serta *Block* I/O. Sehingga hasil pengujian didapat seperti pada Tabel 3 dibawah ini.

Tabel 3. Pengujian Worker Container

Image	CPU (%)	Worker (MiB)	Net I/O	Block I/O
cmainlamp	0.02%	239	14kB / 3.63kB	42kB / 10.3MB
clamp1	0.01%	237.2	15kB / 5.25kB	
clamp2	0.01%	237.1	13.1kB / 5.25kB	
clamp3	0.01%	237.2	12.6kB / 5.97kB	

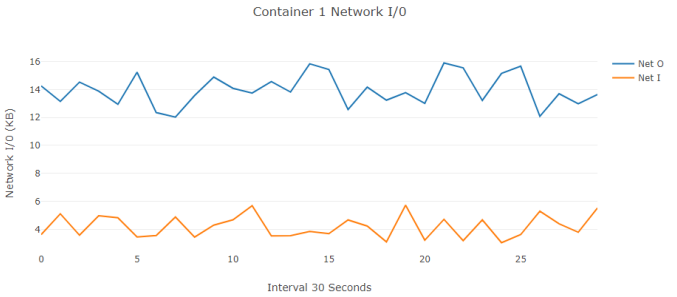
Kinerja CPU sangat rendah dikarenakan tidak ada aktivitas pada HTTP *request/response*, hingga Socker.IO berjalan pada TCP layer sehingga dapat diketahui nilai *traffic* seperti pada Tabel 3 diatas.

Pengujian Network I/O pada tiap *container* juga dilakukan sehingga didapatkan rata-rata *Netwotk I/O* untuk *main container* sebesar 13.75KB/4.16KB seperti pada Gambar 8 dibawah ini.



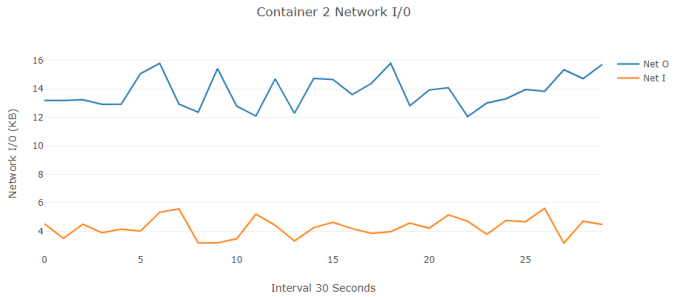
Gambar 8. Pengujian *Network I/O* pada *Main Container*

Pengujian pada *container* yang pada bagunan lantai 1 (*container* 1) didapatkan rata-rata *Netwotk I/O* untuk *main container* sebesar 14.24KB/4.75KB seperti pada Gambar 9 dibawah ini.



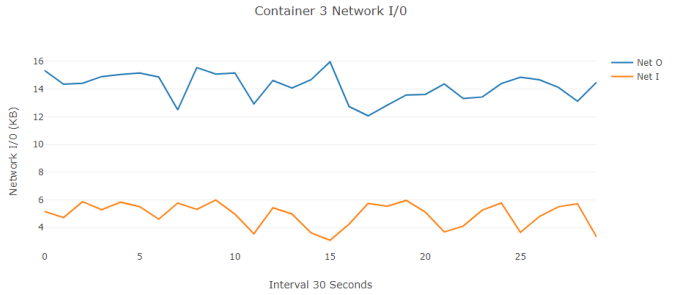
Gambar 9. Pengujian *Network I/O* pada *Container* 1

Pengujian pada *container* yang pada bagunan lantai 2 (*container* 2) didapatkan rata-rata *Network* I/O untuk *container* 1 sebesar 14.12KB/4.75KB seperti pada Gambar 10 dibawah ini.



Gambar 10. Pengujian *Network* I/O pada *Container* 2

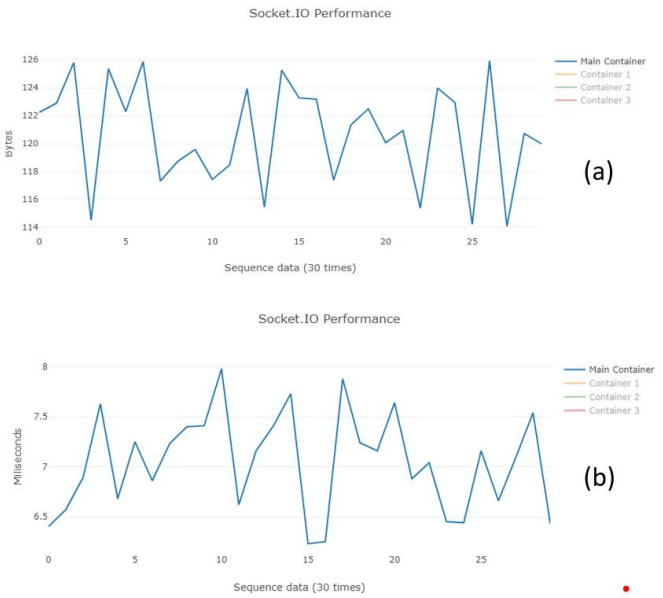
Pengujian pada container yang pada bagunan lantai 2 (*container* 2) didapatkan rata-rata *Network* I/O untuk *container 2* sebesar 13.97KB/4.68KB seperti pada Gambar 11 dibawah ini.



Gambar 11. Pengujian Network I/O pada Container 3

B. Pengujian Socket.IO

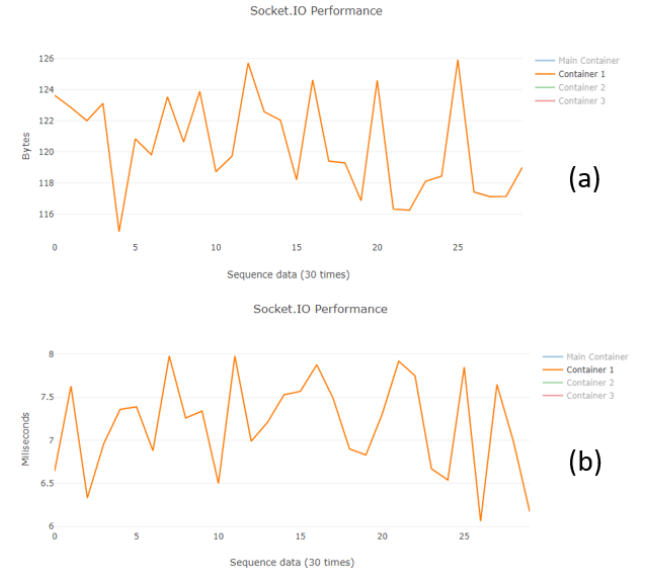
Pengujian pada *TCP layer* dilakukan dengan melihat 2 variabel penting yaitu besarnya data yang dikirim pada *Socket event*. Kedua, berapa respons dari *Socket client* pengirim ke *Socket Server* hingga didistribusikan ke *Socket client*, sehingga peroleh seperti Gambar 12 dibawah ini.



Gambar 12. (a) Kapasitas data pada Socket.IO, (b) Waktu respon *Socket.IO* pada *Main Container*

Pengujian Socket.IO pada *main container* memiliki data dengan kapasitas 118.75 Byte dalam 7.05 ms.

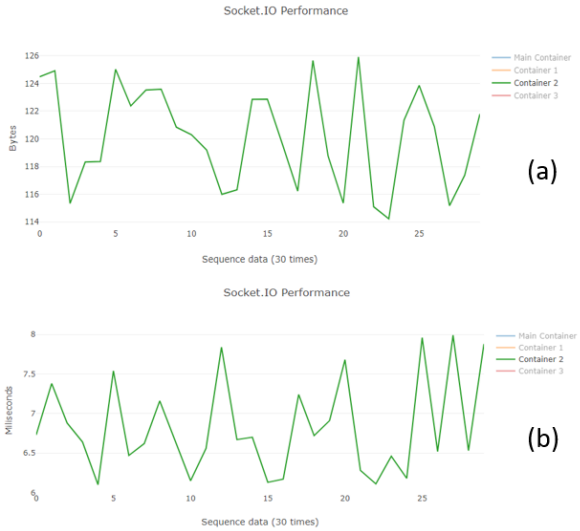
Pengujian selanjurnya dilakukan pada *container* lainnya yaitu *container* 1 yang mengatur perubahan status relay pada lantai 1. Sehinnga didapatkan grafik respons Socket.IO pada container 1 seperti Gambar 13 dibawah ini.



Gambar 13. (a) Kapasitas data pada Socket.IO, (b) Waktu respon Socket.IO pada *Container* 1

Pengujian Socket.IO pada container 1 memiliki data dengan kapasitas 119.42 Byte dalam 6.91 ms.

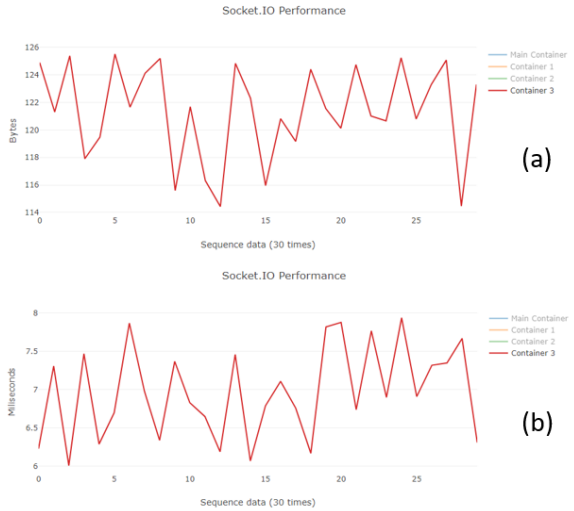
Kemudian dilakukan pada *container* lainnya yaitu *container* 2 yang mengatur perubahan status relay pada lantai 2. Sehinnga didapatkan grafik response Socket.IO pada *container* 2 seperti Gambar 14 dibawah ini.



Gambar 14. (a) Kapasitas data pada Socket.IO, (b) Waktu respon Socket.IO pada *Container* 2

Pengujian Socket.IO pada *container* 2 memiliki data dengan kapasitas 117.97 Byte dalam 6.85 ms.

Terakhir, dilakukan pada *container* lainnya yaitu container 3 yang mengatur perubahan status relay pada lantai 3. Sehinnga didapatkan grafik response Socket.IO pada *container* 3 seperti Gambar 15 dibawah ini.



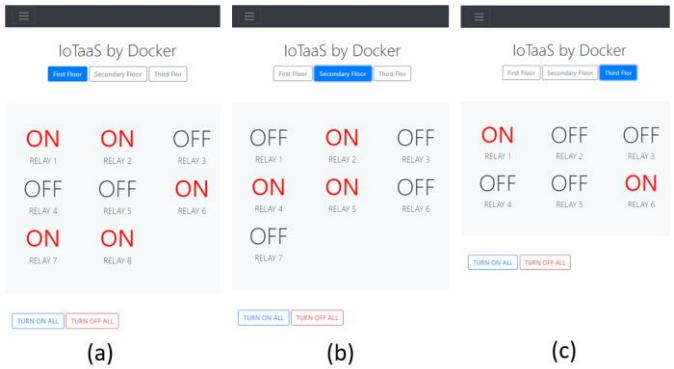
Gambar 15. (a) Kapasitas data pada Socket.IO, (b) Waktu respon Socket.IO pada *Container* 3

Pengujian Socket.IO pada *container* 3 memiliki data dengan kapasitas 118.92 Byte dalam 7.27 ms.

C. *Pengujian Aplikasi Dashboard*

Pengujian aplikasi dashboard dilakukan oleh NodeJS server di *main container*. Sehingga koneksi database hanya terjadi pada server ini. *Container* yang lainnya melakukan *Socket event* Listenting or Emit sehingga data terdistribusi kesemua container. Gambar 16 (a) merupakan gambar dashboard dimana terdapat status tiap relay di lantai 1. Berdasarkan Tabel 1 diketahui jumlah titik relay pada lantai 1 adalah 8 sehingga pada *dashboard* ditampilkan 8 buah relay beserta actionnya untuk set HIGH atau LOW. Gambar 16 (b) merupakan dashboard yang digunakan untuk melakukan pengendalian tau melihat status relay pada lantai 2 sehingga jumlah relay yang dibutuhkan pada lantai ini adalah 7. Kemudian, Gambar 16 (c) merupakan dashboard yang digunakan untuk melakukan pengendalian tau melihat status relay pada lantai 3 sehingga jumlah relay yang dibutuhkan pada lantai ini adalah 6.

Lain dari pada hal tersebut, disediakan pula sebuah *service* untuk melakukan perubahan status semua relay pada sebuah *event* baik melakukan perubahan pada status HIGH ataupun LOW. Maka, perhatikanlah Gambar 16 dibawah ini.



Gambar 16. Pengujian Aplikasi *Dashboard* Lantai 1 sampai dengan Lantai 3

V. KESIMPULAN

IoTaaS telah berhasil diimplementasikan untuk pengendalian instalasi penerangan pada Gedung 3 lantai. Masing-masing lantai diatur oleh sebuah *container*. *Container* dibuat sebagai suatu model pengaplikasian IoTaaS menggunakan Docker. Sehingga secara menyeluruh pengujia dilakukan pada *Network* I/O dengan rata-rata 14.02KB/4.59KB (*download/upload*). Selanjutnya dilakukan juda suatu skema komunikasi menggunkanan koneksi pada layer TCP yaitu menggunakan Socket.IO *event*. Sehingga memungkinkan semua *container* dapat saling bertukar informasi dalam waktu

singkat. Adapun hasil pengujian *Socket event* untuk melakukan perubahan status pada tiap relay didapatkan nilai rata-rata sebesar 118.77 B dalam respon 7.02 *milisecond*.

Perkembangan selanjutnya diharapkan IoTaaS berkembang untuk memecahkan permasalahan yang lebih dinamis bahkan dapat pula dipaduan dengan algoritma *Artificial Intellegence* seperti Machine Learning atau *Deep Learning*.

UCAPAN TERIMA KASIH

Ucapan terima kasih kami sampaikan untuk tim riset yang telah memberikan waktu dan tenaganya hingga riset ini selesai. Terima kasih juga kami sampaikan kepada Prodi Teknik Elektro dan P4 Universitas Mercu Buana yang telah memberikan support pada riset ini, hingga terbit di Jurnal Teknologi Elektro.

DAFTAR PUSTAKA

[1] Y. Gunardi, A. Adriansyah and T. Anindhito, "Small Smart Community: An Application Of Internet Of Things", ARPN Journal of Engineering and Applied Sciences, Vol. 10, No. 15, 2015.

[2] A. Celesti, D. Mulfari, M. Fazio, M. Villari and A. Puliafito, "Exploring Container Virtualization in IoT Clouds", IEEE International Conference on Smart Computing, 2016.
DOI: 10.1109/SMARTCOMP.2016.7501691

[3] J. Rufino, M. Alam, J. Ferreira,A. Rehman and K. F. Tsang, "Orchestration of containerized microservices for IIoT using Docker", IEEE International Conference on Industrial Technology (ICIT), 2017.
DOI: 10.1109/ICIT.2017.7915594

[4] J. Moore, G. Kortuem, A. Smith, N. Chowdhury, J. Cavero and D. Gooch, "DevOps for the Urban IoT". Proceedings of the Second International Conference on IoT in Urban Space, 2016.
DOI: 10.1145/2962735.2962747

[5] P. Patel, M. I. Ali and A. Sheth, "On Using the Intelligent Edge for IoT Analytics", IEEE Intelligent Systems, Vol. 32(5), pp.64–69, 2017.
DOI: 10.1109/MIS.2017.3711653

[6] S. Qanbari, S. Pezeshki, R. Raisi, S. Mahdizadeh, R. Rahimzadeh, N. Behinaein, N., … S. Dustdar, "IoT Design Patterns: Computational Constructs to Design, Build and Engineer Edge Applications", IEEE First International Conference on Internet-of-Things Design and Implementation (IoTDI), 2016.
DOI: 10.1109/IoTDI.2015.18

[7] M. Alam, J., Rufino, J. Ferreira, S. H. Ahmed, N. Shah, and Y. Chen, "Orchestration of Microservices for IoT Using Docker and Edge Computing", IEEE Communications Magazine, 56(9), 118–123, 2018.
DOI: 10.1109/MCOM.2018.1701233

[8] P. Bellavista, and A. Zanni, "Feasibility of Fog Computing Deployment based on Docker Containerization over RaspberryPi", Proceedings of the 18th International Conference on Distributed Computing and Networking, 2018.
DOI: 10.1145/3007748.3007777

[9] B. Cheng, A. Papageorgiou, F. Cirillo, and E. Kovacs, "GeeLytics: Geo-distributed edge analytics for large scale IoT systems based on dynamic topology", IEEE 2nd World Forum on Internet of Things (WF-IoT), 2015.
DOI:10.1109/WF-IoT.2015.7389116

[10] R. Muzawi, Y. Efendi dan N. Sahrun, "Prototype Pengendalian Lampu Jarak Jauh Dengan Jaringan Internet Berbasis Internet of Things (IoT) Menggunakan Rasberry Pi 3", Jurnal Inform, Vol. 3, No.1, 2018.
DOI: 10.25139/ojsinf.v3i1.642

[11] Y. Efendi. "Internet of Things (IoT) "Sistem Pengendalian Lampu Menggunakan Raspberry Pi Berbasis Mobile", Jurnal Ilmiah Ilmu Komputer, Vol. 4, No. 1, pp. 19-26, 2018

[12] F. Z. Rachman, "Smart Home Berbasis IoT", Seminar Nasional Inovasi Teknologi Terapan (SNITT), Politeknik Negeri Balikpapan, 2017.

[13] I. T. Baskoro, Darjat dan Sudjadi, "Perancangan Pengontrolan Nyala Lampu Dan Kipas Angin Pada Sebuah Ruangan Menggunakan Raspberry Pi Model B Dengan Web Gui", Jurnal Transient, Vol. 3, No. 4, pp. 567-571, 2014.

[14] A. Setiawan, I. W. Mustika dan T. B. Adji. "Perancangan Context-Aware Smart Home Dengan Menggunakan Internet O f Things", Seminar Nasional Teknologi Informasi dan KOmunikasi (Sentika), pp. 455-459, 2016.

[15] D. Prihtamoko. "Pemanfaatan Raspberry Pi Sebagai Server Web Untuk Penjadwalan Kontrol Lampu Jarak Jauh", Jurnal Infotel, Vol. 9, No. 1, pp. 84-91, 2017.

[16] B. Artono dan Fredy Susanto, "Led Control System with Cayenne Framework for The Internet of Things (IoT)", Journal of Electrical Electronic Control and Automotive Engineering (JEECAE),VOl. 2, No. 1, pp. 95-100, 2017.

[17] P. C. Devi, A. R. Matondang dan D. Wahyuni. " Usulan Perbaikan Sistem Pencahayaan Di Unit Percetakan Perusahaan XXX Sumatera Utara", Jurnal Teknik Industri, Sumatera Utara, Vol. 5, No. 1, pp. 7-12, 2014.