IMPLEMENTASI & ANALISIS PERFORMANSI LAYANAN WEB PADA PLATFORM BERBASIS DOCKER

IMPLEMENTATION & ANALYSIS OF WEB SERVICE PERFORMANCE BASED ON DOCKER PLATFORM

Muhammad Fihri[1], Ridha Muldina Negara[2], Danu Dwi Sanjoyo[3]
[1,2,3]Prodi S1 Teknik Telekomunikasi, Fakultas Teknik Elektro, Universitas Telkom
[1]**muhammadfihri @students.telkomuniversity.ac.id**, [2]**ridhanegara@telkomuniversity.co.id**,
[3]**danudwj@telkomuniversity.ac.id**

Abstrak

Docker merupakan platform yang berfungsi untuk mengembangkan, mengirim, serta menjalankan beragam aplikasi dengan lebih cepat. Seiring dengan perkembangan layanan *web*, ukuran aplikasi *web* yang dibangun atas arsitektur monolitik akan semakin membesar dan menyulitkan pengembangan berkelanjutan karena aplikasi terdiri menjadi 1 kesatuan aplikasi, dan perubahan sekecil apapun pada lini kode akan memengaruhi keseluruhan aplikasi. Lalu berkembang arsitektur *microservice* yang memecah aplikasi jadi beberapa komponen kecil yang beroperasi terhadap layanan yang spesifik. Platform docker memudahkan untuk mengembangkan sistem *microservice* karena membutuhkan integrasi sistem yang terdiri dari beragam pecahan komponen sistem dan memungkinkan untuk pengembangan sistem secara berkelanjutan. Pada penelitian ini, akan dikembangkan layanan *web E-Commerce* pada *platform* berbasis docker, selanjutnya akan dilakukan simulasi untuk melihat gambaran performa sistem yang telah dikembangkan berdasarkan parameter *throughput, response time, cpu utilization,* dan *memory utilization.* Hasil simulasi menunjukkan sistem yang dibangun mampu memenuhi standar dengan performa baik ketika menerima sejumlah *http load.*

Kata kunci : docker, load balancing, container, layanan web

Abstract

Docker is a platform to develop, send, and run various applications faster. Along with the development of web services, the size of web applications built on monolithic architecture will be increasingly large and make it difficult for continous delivery because the application form as a single unity, and the slightest change in the code line will affect the entire application. Then the microservice architecture was blossom and make it possible to split the application into several small components that operate on specific services. The docker platform makes it easy to develop a microservice system because it requires system integration consisting of a variety of fractional system components and allows for continuous system development. In this study, E-Commerce web services will be developed on a docker-based platform, then simulations will be conducted to see an overview of system performance that has been developed based on parameters throughput, response time, cpu utilization, and memory utilization. Simulation results show that the system built is able to meet the standard with good performance when receiving a number of http loads.

Keywords: docker, load balancing, container, web services

1. Pendahuluan

Dalam proses pengembangan layanan web dibutuhkan berbagai aplikasi yang saling terintegrasi, dimulai dari *front-end development* yang mencakup seluruh desain dan fitur pada *platform* web yang dapat diakses oleh user hingga *back-end development* yang mencakup database yang krusial[1]. Pada arsitektur monolitik , penyedia layanan membangun sebuah aplikasi yang seluruh komponennya terbuat menjadi sebuah kesatuan lalu kemudian disimpan dalam server. Seiring berjalannya waktu tentu layanan yang dijalankan akan tumbuh berkembang, baik dengan penambahan berbagai fitur baru pada *platform* tersebut maupun dari jumlah user yang berkunjung sehingga membutuhkan *scaling system* agar layanan web yang dijalankan mampu memproses seluruh trafik yang masuk karena jumlah trafik yang terus meningkat dan juga menampung seluruh data yang sangat besar dan terus bertambah. Dalam kebanyakan skenario, biasanya yang dibutuhkan hanya scaling pada aplikasi tertentu. Namun karena aplikasi tersebut terbentuk dalam satu paket jika penyedia layanan web ingin melakukan *scaling system* maka harus melakukan perombakan sistem secara menyeluruh karena penambahan *line coding* sekecil apapun akan memengaruhi keseluruhan aplikasi sehingga proses *development* sangat beresiko[2].

Berdasarkan hal yang telah disebutkan, dapat disimpulkan bahwa untuk sistem berskala besar arsitektur monolitik tidak cocok karena akan terdapat banyak hambatan dan permasalahan ketika tiba waktunya

dibutuhkan *scaling system* dan juga penggunaan *resource* yang tidak efisien. Untuk mengatasi permasalahan tersebut, saat ini mayoritas *developer* telah melakukan transisi ke arsitektur *microservice*. Pada arsitektur *microservice*, aplikasi dipecah menjadi beberapa komponen yang dikembangkan dan berjalan secara independen. Sehingga jika ingin melakukan *scaling system* tidak perlu merombak keseluruhan aplikasi karena telah terbagi menjadi komponen komponen kecil. Untuk mengembangkan *microservice application* terdapat sebuah *platform* yang mendukung yaitu docker. Docker merupakan sebuah *platform* yang berfungsi untuk *developing, shipping, and running application* sehingga prosesnya menjadi lebih mudah dan cepat. Dengan *container-based virtualization* yang merupakan inti dari teknologi docker membuatnya jauh lebih efisien untuk mengembangkan *microservice application* daripada aplikasi monolitik karena *microservice application* terdiri dari banyak komponen sehingga satu server bisa digunakan untuk menjalankan banyak aplikasi tanpa mengintervensi kinerja aplikasi lainnya.

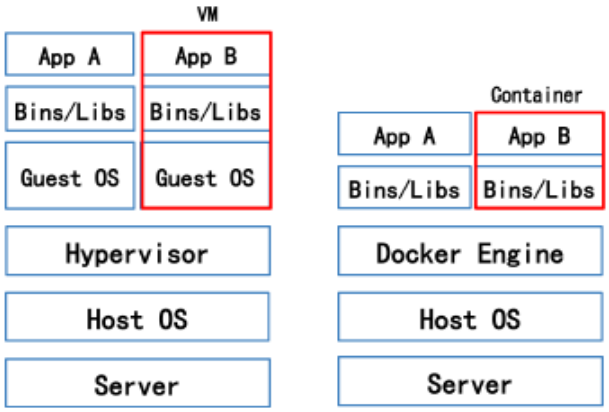
Pada Penelitian tugas akhir ini akan diimplementasikan dan dianalisa performansi layanan web E-Commerce yang diimplementasikan pada *platform* berbasis docker. Kemudian akan dilakukan analisa pada kinerja layanan web e-commerce yang dibuat pada *platform* berbasis docker. Kinerja sistem tersebut dianalisis dengan mengkaji parameter *response time*, *throughput, Memory Utilization* dan *CPU Utilization.* Penelitian ini akan dijelaskan dalam beberapa bagian. Bagian II menjelaskan tentang studi terkait penelitian yang dilakukan. Bagian III menjelaskan tentang desain sitem yang dilakukan. Bagian IV menjelaskan tentang analisis dari pengujian yang dilakukan. Dan kesimpulan yang dapat diambil dari penelitian pada Bagian V.

2. Studi Penelitian

2.1 *Virtualization* dan *Containerization*

Virtualisasi dapat definisikan sebagai abstraksi dari empat sumber daya komputasi *(storage, processing power, memory, and network or I/O)* [3] yang memungkinkan satu perangkat keras untuk dialokasikan menjadi beberapa perangkat virtual yang dapat beroperasi secara independen agar dapat menjalankan berbagai macam jenis aplikasi yang dibutuhkan di waktu yang bersamaan untuk menyediakan sebuah layanan. Namun pada sistem berskala besar yang terus berkembang sistem yang dibangun dengan fondasi arsitektur *microservice* teknologi virtualisasi kurang efisien dalam penggunaan *resource* karena pada arsitektur *microservice* sebuah sistem terdiri dari banyak pecahan komponen aplikasi dibandingkan dengan sistem yang berbentuk satu kesatuan aplikasi yang dibangun dengan arsitektur monolitik, sehingga jika kita ingin melakukan proses isolasi pada setiap aplikasi yang dibangun pada *virtual machine* akan menghabiskan resource dalam jumlah yang besar, dengan rata-rata minimal virtual machine memakan kapasitas per gigabyte hanya untuk menjalankan sebuah aplikasi walaupun ukuran aplikasi tersebut relatif jauh lebih kecil. Hal ini dikarenakan *virtual machine* dibangun atas keseluruhan sistem yang dimulai dari *operating system (OS), Virtualization Layer* berupa hypervisor yang menjembatani komunikasi dan kinerja diantara *host operating system* dan *virtual machine / guest operating system*, dan program-program lainnya yang perlu diinstalasi dan dijalankan untuk mendukung kinerja *virtual machine* [4].

Teknologi *container* dikembangkan untuk menyederhanakan segala persoalan yang telah dijabarkan. *Container* adalah *lightweight operating system* yang dapat bekerja secara langsung didalam *host operating system. Container* menjalankan segala proses instruksi secara langsung ke *core CPU*. *Container* mampu memberikan penghematan penggunaan resource tanpa *overhead* dari *virtualization* serta menjamin kinerja aplikasi yang terisolasi [5]. Besaran *container* juga jauh lebih kecil jika dibandingkan dengan *virtualization,* biasanya ukuran container hanya berskala megabyte karena container hanya berisi paket aplikasi yang dibutuhkan tanpa *operating system*. Dan tanpa adanya hypervisor performansi aplikasi yang dijalankan jauh lebih baik, terutama jika aplikasi membutuhkan proses interaksi dengan *I/O devices* [4].



Gambar 2.1. Perbedaan Arsitektur *Virtualization* dan *Containerization* [4]

2. 2 Docker

Docker adalah sebuah aplikasi open source yang berfungsi untuk mengembangkan, mendistribusi, serta menjalankan *software application.* Desain arsitektur docker memudahkan untuk mendistribusi serta mengembangkan aplikasi secara lebih cepat karena docker memiliki sifat *lightweight containerization* yang dilengkapi dengan beragam komponen serta fitur sehingga mampu memudahkan para *developer* untuk mengembangkan dan memantau kinerja aplikasi yang diciptakan [5].

2.2.1 Arsitektur Docker

Arsitektur docker dibangun dengan fondasi *Linux container* (LXC) dengan fitur-fitur seperti *cgroups* dan *namespaces* untuk mengatur sumber daya komputasi dan proses isolasi yang kuat. Dengan hal ini docker mampu memproses kinerja sistem serupa dengan *kernel based virtual machine (*KVM) dengan *resource* yang lebih sedikit [7].

Linux container (LXC) adalah *OS level virtualization* yang mampu menjalankan beberapa proses *linux system* sekaligus secara terisolasi dalam sebuah linux *control host* dan menjadi penjembatan dengan kinerja fitur-fitur sistem linux kernel. *Linux container* (LXC) dibangun dengan basis sistem *chroot* yang termuat atas *binaries, libraries,* dan file konfigurasi yang disebut *chroot jail* sehingga mampu membuat lingkungan pengembangan yang terisolasi yang berjalan diatas kernel untuk aplikasi [8].

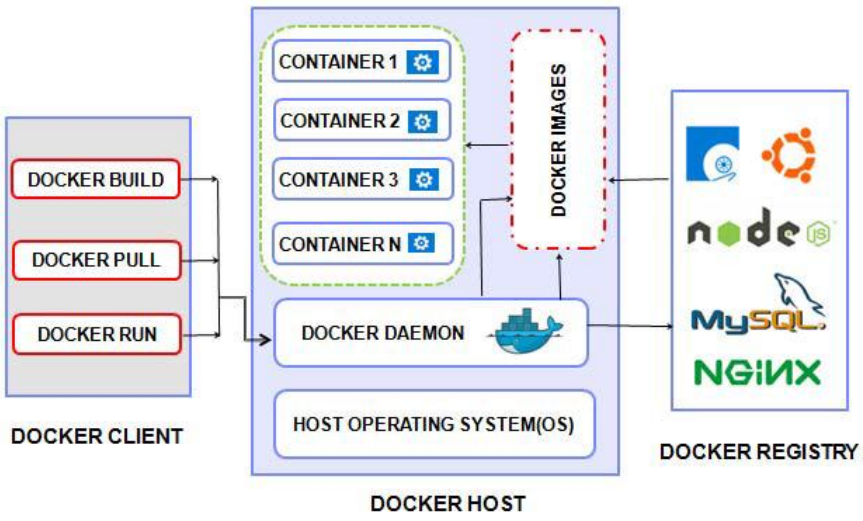
Cgroups adalah kumpulan *resource* yang dapat dibuat pada *Linux kernel level. Cgroups* bertugas untuk membatasi penggunaan *resource* sehingga setiap *virtual machine/container* hanya menyerap *resource* sesuai kebutuhan, hal ini menjadikan teknologi *containerization* memiliki lingkungan pengembangan yang efisien.

Docker mempunyai dua komponen utama, yaitu *open source containerization platform* yang disebut docker dan docker hub yang merupakan *Software-as-a-Service (*SaaS) *platform* untuk berbagi dan mengatur docker container, docker menggunakan model arsitektur client-server, dimana docker *client* dapat terhubung ke docker daemon yang membuat, menjalankan, serta mendistribusikan docker container.

Docker *client* dan docker daemon dapat berjalan diatas sistem yang sama ataupun docker *client* dapat melakukan komunikasi melewati sockets atau RESTful API ke docker daemon secara *remote* [9].

Docker daemon berjalan di *host machine* dan pengguna terhubung dengan docker daemon melalui docker *client.* Fungsi dari docker *client* adalah menerima serangkaian perintah dari pengguna lalu menghubungkan perintah tersebut agar dapat diproses oleh docker daemon.

Docker *images* berbentuk *read-only templates* dan docker registries menampung docker *images* tersebut. Docker *container* terbuat dari docker *image* yang berisi segala paket yang dibutuhkan untuk menjalankan suatu aplikasi secara terisolasi [10].

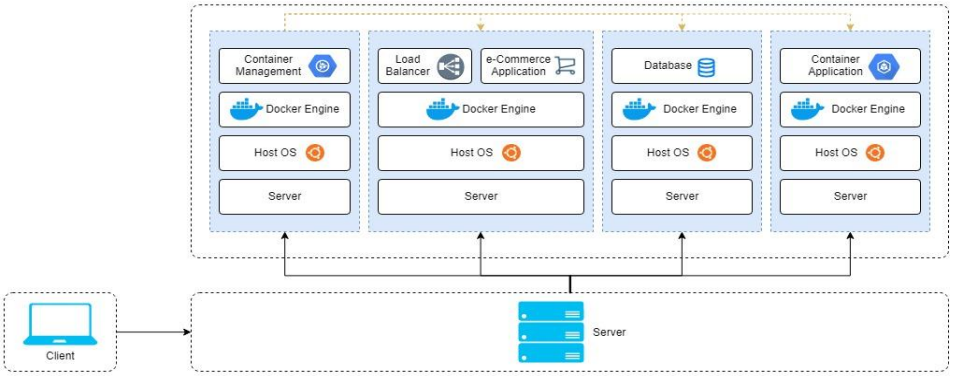


Gambar 2.3.1. Arsitektur Docker [10]

3. Desain Sistem

Sistem yang akan dikembangkan merupakan layanan web E-Commerce yang dibangun menggunakan docker. Sistem terdiri dari beberapa aplikasi yang saling terkoneksi. Seluruh proses sistem yang bekerja terdapat pada Cluster Virtual Server yang dibangun menggunakan teknologi virtualisasi server serta *containerization* untuk membantu proses isolasi sistem aplikasi yang kompleks dan memungkinkan untuk *scaling system* sesuai perancangan dan kebutuhan hingga kapasitas tertentu sesuai dengan spesifikasi server. Sistem dapat dibangun dan dikembangkan melalui rancherOS yang merupakan sebuah *container management tools* sebagai medium penghubung dan pengatur dari beberapa docker *container* yang bekerja saling terkoneksi melalui medium web browser. Desain sistem web E-Commerce yang dibangun terdiri dari variasi Reaction Commerce sebagai *front-*

end system dan mongoDB sebagai *back-end system* yang bekerja saling terintegrasi untuk menciptakan layanan dengan performa terbaik. Sistem Reaction Commerce akan di *scale* yang dapat ditampung di *host* yang tersedia untuk meningkatkan performa server dan aplikasi. Kedua sistem tersebut terhubung berkat tools dan konfigurasi dari rancherOS, terdapat konfigurasi automasi *load balancer* untuk membagi beban kerja server sehingga mampu menyajikan layanan *high performance.*



Gambar 3. Desain Sistem

3.1 Perangkat Keras (Hardware)

Beberapa perangkat keras yang dibutuhkan untuk menunjang implementasi sistem pada tugas akhir ini yaitu:

a. 4 Buah PC sebagai server aplikasi yang dikembangkan, dengan masing-masing server memiliki spesifikasi sebagai berikut:
   - Prosesor : Intel® Core™ i5
   - Memory : 2 GB
   - Harddisk Drive : 20 GB
   - NIC : Fast Ethernet

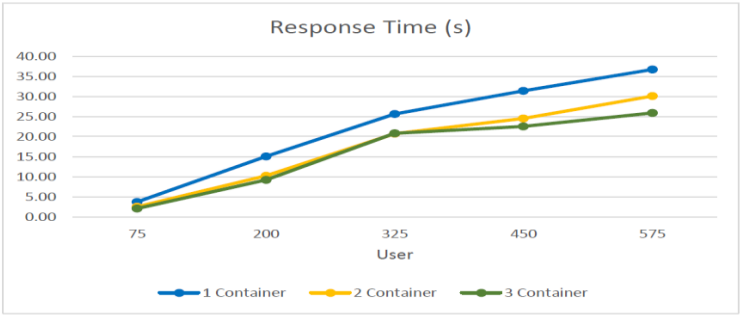
b. 1 Buah PC sebagai client server untuk simulasi *http load testing* untuk mengukur performa aplikasi yang dikembangkan, dengan masing-masing server memiliki spesifikasi sebagai berikut:
   - Prosesor : Intel® Core™ i5
   - Memory : 2 GB
   - Harddisk Drive : 20 GB
   - NIC : Fast Ethernet

4. Hasil Pengujian dan Analisa

4.1 Performa Response Time Sistem

Response Time (s)

	Response Time (s)				
	75	200	325	450	575
1 Container	3.71	15.05	25.66	31.35	36.73
2 Container	2.54	10.21	20.75	24.51	30.08
3 Container	2.06	9.19	20.83	22.57	25.86



Gambar 4.1 Hasil Reponse time (s)

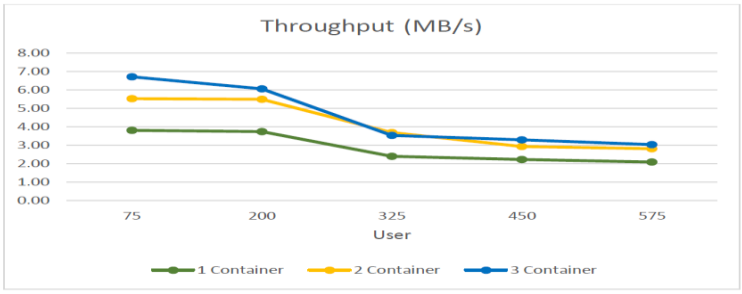
Dari seluruh pengujian sistem yang telah dilakukan telah didapat data response time sistem berdasarkan jumlah *container front-end application* yang telah dikembangkan dan simulasi sejumlah user yang memberikan

http load pada layanan, berdasarkan data dan grafik di atas dapat diketahui bahwa dengan cara melakukan *scaling container front-end application* serta penambahan konfigurasi load balancer dapat meningkatkan kinerja sistem dengan cara membagi beban trafik sehingga sistem dapat bekerja lebih cepat untuk merespon *request* user.

4.2 Performa Throughput Sistem

Throughput (MB/s)

	Throughput (MB/s)				
	75	200	325	450	575
1 Container	3.80	3.74	2.39	2.23	2.08
2 Container	5.52	5.49	3.69	2.93	2.80
3 Container	6.70	6.05	3.54	3.29	3.03



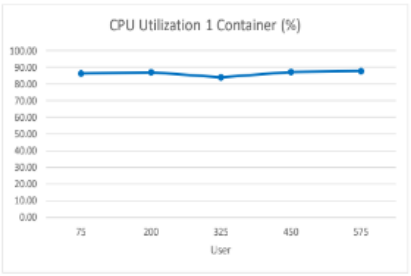
Gambar 4.2 Hasil throughput (MB/s)

Dari seluruh pengujian sistem yang telah dilakukan telah didapat data **throughput** sistem berdasarkan jumlah *container front-end application* yang telah dikembangkan dan simulasi sejumlah user yang memberikan *http load* pada layanan, berdasarkan data dan grafik di atas dapat diketahui bahwa dengan cara melakukan *scaling container front-end application* serta penambahan konfigurasi load balancer dapat meningkatkan kinerja sistem dengan cara meningkatkan kapasitas maksimal server karena server terdiri dari beberapa replika *front-end application container* yang saling terintegrasi oleh konfigurasi load balancer sehingga server mempunyai kapasitas lebih besar untuk memberikan layanan dan transfer informasi data, kapasitas rata-rata informasi *byte* yang dapat diberikan server kepada user meningkat ketika tekah dilakukan *scaling system.*

4.3 Performa CPU Utilization

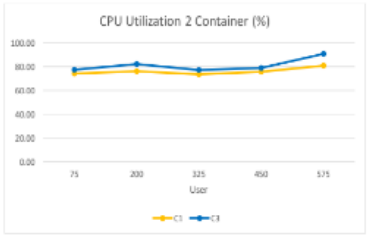
CPU Utilization 1 Container

	CPU Utilization (%)				
	75	200	325	450	575
C1	86.34	86.94	83.96	87.26	87.80



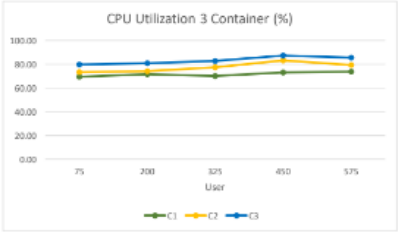
CPU Utilization 2 Container

	CPU Utilization (%)				
	75	200	325	450	575
C1	74.14	76.06	73.51	75.73	80.93
C3	77.31	82.06	77.15	78.97	90.65



CPU Utilization 3 Container

	CPU Utilization (%)				
	75	200	325	450	575
C1	69.44	71.58	70.04	73.21	73.83
C2	73.47	74.17	77.37	83.17	79.28
C3	79.90	80.80	82.90	87.28	85.72



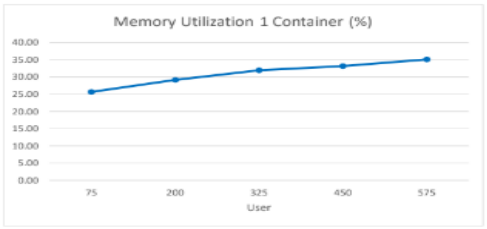
Gambar 4.3 Data performa CPU Utilization dengan *n* container

Data kinerja performa *CPU utilization* menunjukkan konsumsi resource CPU total per *virtual machine* yang dikembangkan dengan kapasitas memory 2 GB dan 2 *core CPU* sekitar 80% dari kapasitas total CPU, sistem yang dikembangkan dapat dioperasikan dengan optimal baik dengan sistem *single server* ataupun sistem *load balancing.* Tetapi penggunaan *load balancing* dapat memberikan kualitas layanan yang lebih baik karena jika terjadi *server failure* terdapat *server* lain yang mampu melayani *request* dari *client* sampai *server* yang mengalami kegagalan sistem dapat berjalan kembali.

4.4 Performa Memory Utilization

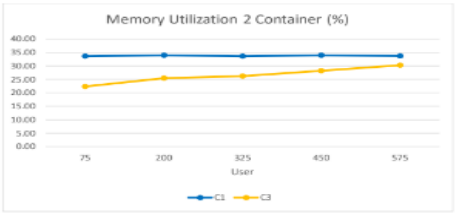
Memory Utilization 1 Container

	Memory Utilization (%)				
	75	200	325	450	575
C1	25.64	29.18	31.94	33.18	35.11



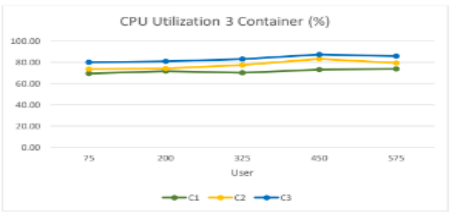
Memory Utilization 2 Container

	Memory Utilization (%)				
	75	200	325	450	575
C1	33.66	33.97	33.63	33.92	33.70
C3	22.39	25.45	26.23	28.22	30.25



Memory Utilization 3 Container

	Memory Utilization (%)				
	75	200	325	450	575
C1	33.63	33.79	33.53	33.73	33.85
C2	19.90	22.07	23.74	25.53	24.60
C3	27.53	27.71	27.12	27.28	29.70



Gambar 4.4 Data performa Memory Utilization dengan *n* container

Data kinerja performa *memory utilization* menunjukkan konsumsi *resource* dari *memory* total per *virtual machine* yang dikembangkan dengan kapasitas *memory* 2 GB dan 2 *core CPU* sekitar 30% dari kapasitas total *memory*. Seluruh *stack* aplikasi *container* yang berjalan dalam suatu *server* dapat dioperasikan dengan optimal baik dengan sistem *single server* ataupun sistem *load balancing*, tetapi pada sistem *load balancing* salah satu *server* biasanya bekerja lebih berat karena pada *server* tersebut terdapat lebih banyak *stack* aplikasi *container* serta *konfigurasi load balancer* dan *networking* sehingga beban kinerjanya lebih berat daripada *server* lain.

5. Kesimpulan

Layanan *web E-Commerce* yang dikembangkan atas platform docker mampu memenuhi standar response time, hal ini dapat dicapai melalui teknik *load balancing* yang mampu mengurai *trafik* pada sistem yang dikembangkan. Penambahan jumlah *container* mampu meningkatkan kapasitas informasi *byte* yang mampu ditransfer *server* terhadap *client*. Penggunaan *resource* CPU & *Memory* juga menunjukkan performa *server* dalam keadaan optimal.

Daftar Pustaka:

[1] S. E. Ullah, T. Alauddin and H. U. Zaman, "Developing an E-commerce website," *2016 International Conference on Microelectronics, Computing and Communications (MicroCom)*, Durgapur, 2016, pp. 1-4.

[2] A. Koschel, I. Astrova and J. Dötterl, "Making the move to microservice architecture," *2017 International Conference on Information Society (i-Society)*, Dublin, 2017, pp. 74-79.

[3] Rajkumar Buyya, James Broberg, Andrzej Goscinski. (2011). *CLOUD COMPUTING : Principles and Paradigms.* New Jersey. John Wiley & Sons, Inc. Retrieved from http://libgen.io.

[4] Zhang, Qi & Liu, Ling & Pu, Calton & Dou, Qiwei & Wu, Liren & Zhou, Wei. (2018). A Comparative Study of Containers and Virtual Machines in Big Data Environment.

[5] "About Docker" Docker documentation. [Online]. Available: https://docs.docker.com/engine/misc/