JITE (Journal of Informatics and Telecommunication Engineering)

Available online <http://ojs.uma.ac.id/index.php/jite> DOI: 10.31289/jite.v4i2.4019

Received: 13 July 2020

Accepted: 13 January 2021

Published: 18 January 2021

Design and Implementation of Lightweight Virtualization Using Docker Container in Distributing Web Application with Experimental Methods

Stefanus Eko Prasetyo*

Prodi Teknologi Informasi, Fakultas Ilmu Komputer, Universitas Internasional Batam, Indonesia

*Corresponding Email: stefanus@uib.ac.id

Abstrak

Pesatnya perkembangan aplikasi berbasis *website* yang dapat diakses diberbagai *platform* komputer dan ponsel pintar, menyebabkan infrastruktur penyebaran aplikasi *web* terutama *server* sangat dibutuhkan. Hal ini menyebabkan terkadang organisasi perlu melakukan investasi berupa *server*. Dengan virtualisasi, perusahaan dapat membuat beberapa *server* pada mesin yang sama, dan setiap *server* berfungsi sebagai mesin individual. Virtualisasi berbasis *lightweight* atau dikenal dengan kontainer adalah pendekatan dimana sistem operasi *host* dibagikan dengan *server* virtual yang berarti *kernel* induk dibagikan kepada *server* virtual atau kontainer. Metode yang digunakan adalah metode eksperimental, penelitian ini mengukur pengaruh suatu perlakuan tertentu terhadap sebuah variabel dengan perlakuan yang berbeda. Variabel penelitian yang dilakukan pada penelitian sistem *lightweight* virtualization, diantaranya: *throughput*, *Average Latency*, *response time* dan *Dropped Request*. Perlakuan pada penelitian ini diuji dengan 100, 200, 250 dan 300 koneksi secara bersamaan dengan 1000, 3000, 5000, 7000 requests. Desain layanan server menggunakan virtualisasi berbasis kontainer untuk webserver dengan spesifikasi Prosesor Intel Core i7 2,8 Ghz (4 core), Memory 2 GB, Storage 80 GB M2 SSD, dan Gigabit Ethernet mampu menangani 250 permintaan secara bersamaan, tetapi tidak bisa diakses ketika menangani 300 permintaan secara bersamaan.

Kata Kunci: Virtualisasi, Container, Docker, Aplikasi Website.

Abstract

The rapid development of website-based applications that can be accessed on various computer platforms and smart phones, causing the infrastructure of web application deployment, especially servers, is needed. This causes sometimes the organization needs to invest in a server. With virtualization, companies can create multiple servers on the same machine, and each server functions as an individual machine. Lightweight based virtualization or known as containers is an approach where the host operating system is shared with a virtual server which means that the parent kernel is shared with virtual servers or containers. The method used is an experimental method, this study measures the effect of a particular treatment on a variable with different treatments. Research variables carried out in the lightweight virtualization system research, including: *throughput*, *Average Latency*, *response time* and *Dropped Request*. The treatments in this study were tested with 100, 200, 250 and 300 connections simultaneously with 1000, 3000, 5000, 7000 requests. The server service design uses container-based virtualization for the webserver with 2.8 Ghz (4 core) Intel Core i7 processor specifications, 2 GB memory, 80 GB M2 SSD storage, and Gigabit Ethernet capable of handling 250 requests simultaneously, but cannot be accessed when handling 300 requests simultaneously.

Keywords: Virtualization, Container, Docker, Website Applications.

How to Cite: Prasetyo, S. E. (2021). Desain dan Implementasi Lightweight Virtualization Menggunakan Docker Container dalam Pendistribusian Aplikasi Web Menggunakan Metode Eksperimental, JITE(Journal of Informatics and Telecommunication Engineering), 4 (2): 270-276.

I. PENDAHULUAN

Situs web maupun aplikasi berbasis web berkembang pesat dikarenakan peningkatan teknologi komputer, ponsel cerdas dan internet yang sangat cepat (Apridayanti, Isnawaty, & Saputra, 2018). Kemudahan proses akses terhadap aplikasi berbasis web menyebabkan infrastruktur pendukung aplikasi web terutama server harus dalam keadaan prima. Dengan layanan yang dijalankan semakin bertumbuh dan berkembang, baik dengan penambahan fitur baru pada aplikasi maupun dari jumlah kunjungan sehingga membutuhkan scaling system server agar aplikasi yang dijalankan mampu memproses seluruh kunjungan yang datang (Fihri, Negara, & Sanjoyo, 2019).

Dengan bantuan virtualisasi, pengguna dapat membuat beberapa instance pada mesin yang sama dan setiap instance berfungsi sebagai mesin individu (Bansal & Kaur, 2015). Virtualisasi berbasis hypervisor terdiri dari dua tipe yaitu Tipe-1 dan Tipe-2. Virtualisasi Tipe-1 adalah virtualisasi di mana hypervisor program perangkat lunak langsung dimuat pada perangkat keras mesin dan pengguna dapat membuat sejumlah mesin virtual menggunakan jenis ini, hal ini disebut hypervisor asli atau baremetal. Sedangkan, Tipe-2 adalah virtualisasi di mana hypervisor dimuat pada sistem operasi host (Popek & Goldberg, 1973). Virtualisasi berbasis sistem operasi adalah pendekatan di mana sistem operasi host dibagikan dengan instance virtual. Ini berarti kernel host dibagikan dengan instance virtual dan instance virtual tersebut dikenal sebagai wadah, itulah sebabnya kadang-kadang dikenal sebagai kontainerisasi atau lightweight Virtualization (Bansal & Kaur, 2015). Contoh aplikasi yang digunakan dalam lightweight Virtualization, adalah OpenVZ (Openvz.org, n.d.), Linux Container (LXC) (Linuxcontainer.org, n.d.), Docker (Lingayat, Badre, & Gupta, 2018; Pande, Bansal, & Hivarekar, 2016; Zhang et al., 2018), dan Kubernetes (Joy, 2015).

Perbedaan yang terlihat antara hypervisor dan kontainer adalah ukuran file pada kontainer lebih kecil dibandingkan dengan hypervisor karena kontainer tidak menyiapkan sistem operasi secara keseluruhan. Dengan menggunakan kontainer, sebuah aplikasi akan di tempatkan pada wadah khusus beserta sumber pustakanya, konfigurasi, pengaturan dan semua hal yang diperlukan oleh aplikasi tersebut (Apridayanti et al., 2018). Docker mengotomasi pengerjaan sebuah aplikasi di dalam kontainer atau wadah khusus tersebut. Docker memanfaatkan fitur isolasi sumber daya kernel linux yaitu cgroups dan namespaces untuk menjaga kontainer beroperasi secara independent (Bairagi, Chourasiya, Silakari, Dixit, & Sharma, 2020). Docker memiliki kelebihan dari teknologi virtual tradisional yang menggunakan virtual mesin dalam pendistribusian dan eksekusi sebuah aplikasi (Turnbull, 2014). Docker adalah teknologi virtualisasi kontainer atau wadah. Jadi, ini seperti mesin virtual yang sangat ringan. Selain membuat wadah, docker menyediakan apa yang disebut alur kerja pengembang, yang sebenarnya tentang membantu orang membuat wadah dan aplikasi di dalam wadah dan kemudian membagikannya di antara wadah (Anderson, 2015).

Rad, Bhatti, & Ahmadi, (2017) pada penelitiannya yang berjudul "An Introduction to Docker and Analysis of its Performance", docker menyediakan beberapa fasilitas, yang berguna untuk pengembang dan administrator. Ini adalah platform terbuka yang dapat digunakan untuk membangun, mendistribusikan, dan menjalankan aplikasi secara portabel dan ringan. Biaya dapat dikurangi dengan mengganti mesin virtual tradisional dengan Docker Container.

Menurut Bansal & Kaur, (2015) pada penelitiannya yang berjudul "An Implementation of Servers using Lightweight Virtualization / Containerization", Server dapat di implementasikan pada mesin virtual menggunakan pendekatan hypervisor dari virtualisasi, tetapi karena kekurangan hypervisor, lebih baik menggunakan virtualisasi berbasis Sistem Operasi. peneliti menggunakan alat virtualisasi ringan bernama Docker. Dengan docker, peneliti menerapkan dua server web apache pada satu host dengan overhead prosesor minimum. Dapat disimpulkan dari hasil bahwa server bekerja sama seperti ketika ini diterapkan pada masing-masing mesin. Dengan pendekatan ini, administrator jaringan dan administrator sistem dapat dengan mudah mengelola beberapa server mereka pada satu host sehingga mengurangi biaya infrastruktur.

Server web adalah aplikasi yang bertugas dalam menyalurkan halaman web ke pengguna, halaman web yang dikirimkan akan disesuaikan dengan permintaan dari pengguna. Cara kerja web server adalah pada saat peramban web meminta halaman web ke server maka permintaan data peramban web tersebut dienkapsulasi dalam Transmission Control Protocol (TCP) dan dikirim ke protokol berikutnya yaitu HTTP. Permintaan data yang dikirimkan dari peramban web ke web server disebut sebagai permintaan HTTP dan kemudian web server akan mencari data HTML yang dibutuhkan dan dienkapsulasi dalam protokol TCP kemudian dikirim kembali ke peramban (Novianta & Setyaningsih, 2015).

II. METODE PENELITIAN

Metode yang digunakan dalam penelitian ini adalah metode eksperimental, penelitian ini mengukur pengaruh suatu perlakuan tertentu terhadap sebuah variabel dengan perlakuan yang berbeda. Variabel

penelitian yang dilakukan pada penelitian sistem *lightweight virtualization*, diantaranya: *throughput*, *Average Latency*, *response time* dan *Dropped Request*.

1. *Throughput* adalah *bandwidth* aktual yang terukur pada ukuran waktu tertentu di jaringan. Pengujian pada *throughput* merupakan parameter variabel dari *Quality of Service* yang ditujukan untuk melihat performansi jaringan dari segi kecepatan pengiriman paket yang dilakukan dengan mengirim dan memanfaatkan *bandwidth* yang ada.
2. Pengujian *Request per Second* untuk menunjukkan seberapa banyak pengunjung yang dilayani dalam satu detik dalam satu permintaan masal.
3. *Response Time* Pengujian *response time* ditujukan guna mengukur seberapa cepat suatu *lightweight virtualization* dapat menerima request dari client.
4. Pengujian *Dropped Request* dilakukan untuk meninjau paket yang dikirimkan terbuang karena sumberdaya menampung lebih dari kapasitas yang bisa di tamping. Proses pembuangan paket ini disebut dengan *dropped request*.

Perlakuan pada penelitian ini diuji dengan 100, 200, 250 dan 300 koneksi secara bersamaan dengan 1000, 3000, 5000, 7000 requests. Sistem yang dirancang merupakan layanan web sederhana yang dibangun menggunakan docker. Seluruh proses sistem yang berjalan dibangun menggunakan teknologi kontainer untuk membantu proses isolasi sistem layanan yang kompleks dan memungkinkan untuk melakukan penambahan dan pengurangan kapasitas sumber daya sesuai kebutuhan dan dengan spesifikasi server.

Rancangan ini diujicoba melalui Docker Desktop sebagai media penghubung dan pengatur dari beberapa kontainer doker yang bekerja saling terkoneksi dengan docker image *docker-nginx-php-mysql*. Layanan server yang digunakan dalam ujicoba ini adalah Nginx, PHP-FPM, dan database MySQL seperti pada Gambar 1. Penelitian ini dilakukan pada jaringan local dan web page yang digunakan dalam penelitian ini hanya berupa website cms wordpress 5 dengan pengaturan *default*. Pengiriman koneksi dan permintaan ke server menggunakan aplikasi Apache Benchmark dan dianalisa menggunakan Wireshark untuk mengukur *throughput*, *response time*, *dropped package* dan *request per second*, pengujian dilakukan untuk melihat kemampuan server *lightweight virtualization*.

Gambar 1. Desain Sistem

Beberapa perangkat keras yang dibutuhkan untuk melakukan penelitian ini yaitu:

1 Buah laptop sebagai docker desktop yang dikembangkan, dengan spesifikasi sebagai berikut:

Prosesor : Intel Core i7 2,8 Ghz (4 core)
Memory : 2 GB
Storage : 80 GB M2 SSD
NIC : Gigabit Ethernet

1 Buah laptop sebagai client simulasi http load testing untuk mengukur performa rancangan yang dikembangkan, dengan spesifikasi sebagai berikut:

Prosesor : Intel Core i7 2,8 Ghz (8 core)
Memory : 8 GB
Storage : 500 GB M2 SSD
NIC : Gigabit Ethernet

III. HASIL DAN PEMBAHASAN

Skenario pengujian dilakukan pada layanan aplikasi yang ada dengan mengukur *throughput*, *response time*, *dropped package* dan *request per second*, pengujian dilakukan untuk melihat kemampuan *server lightweight virtualization*. Dilakukan permintaan pada server untuk menguji kemampuan *web server*, *request* yang diuji sebesar 1000, 3000, 5000 dan 7000 menggunakan *apache benchmark* dan *wireshark*. Adapun skenarionya sebagai berikut:

1. Pengujian dengan 100 koneksi secara bersamaan dengan 1000, 3000, 5000, 7000 *requests*.
2. Pengujian dengan 200 koneksi secara bersamaan dengan 1000, 3000, 5000, 7000 *requests*.
3. Pengujian dengan 250 koneksi secara bersamaan dengan 1000, 3000, 5000, 7000 *requests*.
4. Pengujian dengan 300 koneksi secara bersamaan dengan 1000, 3000, 5000, 7000 *requests*.

Tabel 1. Hasil Pengujian

Paket	Througput (kbps)	Response Time (s)	Dropped (%)	Request per second
100 Koneksi Secara Bersamaan				
1000	5051	0,37	0	2745
3000	4633	1,208	0	2628
5000	4613	2,021	0	2571
7000	4726	2,761	0	2540
200 Koneksi Secara Bersamaan				
1000	5244	0,355	0	2838
3000	5070	1,106	0	2738
5000	5115	1,819	0	2758
7000	5049	2,585	0	2714
250 Koneksi Secara Bersamaan				
1000	6369	0,295	0	2598
3000	5099	1,097	0	3450
5000	4593	2,043	0	3190
7000	4860	2,688	0	2526
300 Koneksi Secara Bersamaan				
1000	Server down	Server down	Server down	Server down

Pada Tabel 1 dapat dilihat bahwa hasil pengujian cukup bervariasi, terjadi peningkatan pada response time server dan penurunan jumlah throughput serta request per second.

Dari pengujian yang telah dilakukan, pada Tabel 2 dan Gambar 2 didapatkan data throughput berdasarkan jumlah koneksi secara bersamaan pada layanan, berdasarkan data dan grafik di atas dapat dilihat bahwa dengan penggunaan docker kontainer dengan spesifikasi server yang minimal memberikan throughput sebesar 5049 KBps pada jumlah 7000 koneksi dengan akses 200 koneksi secara bersamaan.

Tabel 2. Hasil Pengujian Throughput

Throughput (kbps)				
	1000	3000	5000	7000
100	5051	4633	4613	4726
200	5244	5070	5115	5049
250	6369	5099	4593	4860

Gambar 2. Hasil Pengujian Throughput

Dari pengujian yang telah dilakukan, pada Tabel 3 dan Gambar 3 didapatkan data response time berdasarkan jumlah koneksi secara bersamaan pada layanan, berdasarkan data dan grafik di atas dapat dilihat bahwa dengan penggunaan docker kontainer dengan spesifikasi server yang minimal memberikan response time yang cukup cepat pada jumlah 7000 koneksi dengan rata-rata selama 2 detik waktu tunggu.

Tabel 3. Hasil Pengujian Response Time

Response Time (s)				
	1000	3000	5000	7000
100	0,37	1,208	2,021	2,761
200	0,355	1,106	1,819	2,585
250	0,295	1,097	2,043	2,688

Gambar 3. Hasil Pengujian Response Time

Dari pengujian yang telah dilakukan, pada Tabel 4 dan Gambar 4 didapatkan data request per second berdasarkan jumlah koneksi secara bersamaan pada layanan, berdasarkan data dan grafik di atas dapat dilihat bahwa rata – rata server layanan mampu melayani paling sedikit 2526 koneksi dan paling banyak 3450 koneksi.

Tabel 4. Hasil Pengujian Request per Second

Request per second				
	1000	3000	5000	7000
100	2745	2628	2571	2540
200	2838	2738	2758	2714
250	2598	3450	3190	2526

Gambar 4. Hasil Pengujian Request per Second

Pengujian dilakukan dengan koneksi sebanyak 300 secara bersamaan dan 1000 permintaan menghasilkan error terlalu banyak koneksi sehingga tidak mampu di tangani oleh server seperti pada Gambar 5.

```

This is ApacheBench, Version 2.3 <$Revision: 1843412 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking localhost (be patient)
socket: Too many open files (24)

```

Gambar 5. Hasil pengujian 300 koneksi secara bersamaan

IV. SIMPULAN

Desain layanan server menggunakan virtualisasi berbasis kontainer untuk webserver dengan spesifikasi Prosesor Intel Core i7 2,8 Ghz (4 core), Memory 2 GB, Storage 80 GB M2 SSD, dan Gigabit Ethernet mampu menangani 250 permintaan secara bersamaan secara jaringan lokal, tetapi tidak bisa diakses ketika menangani 300 permintaan secara bersamaan. Pemanfaatan kontainer bagi layanan server sangat ringan dan hanya mengambil service service yang diperlukan sehingga resource induk tidak terbebani.

Layanan web yang menggunakan server Lightweight Virtualization berbasis Docker Container secara sederhana dengan kapasitas ram 4GB dan prosesor 4 core mampu melayani permintaan sebanyak 7000 request layanan dengan maksimal 250 koneksi secara langsung. Diperlukan penambahan jumlah kontainer yang berisikan load balancing untuk menangani koneksi sebanyak 300 atau lebih.

DAFTAR PUSTAKA

- Anderson, C. (2015). Docker. *IEEE Software*, 32(3), 102–105. <https://doi.org/10.1109/MS.2015.62>
- Apridayanti, S., Isnawaty, & Saputra, R. A. (2018). Desain dan implementasi virtualisasi berbasis Docker untuk Deployment Aplikasi Web. *Semantik*, 4(2), 37–46.
- Bairagi, H., Chourasiya, U., Silakari, S., Dixit, P., & Sharma, S. (2020). Tools And Techniques In Cloud Environment. *International Journal of Scientific & Technology Research*, 9(01), 1425–1430.
- Bansal, A. K., & Kaur, H. (2015). An Implementation of Servers using Lightweight Virtualization/Container. *International Journal of Innovative Research in Computer and Communication Engineering*, 3(12), 12427–12431. <https://doi.org/10.15680/IJIRCCE.2015>
- Fihri, M., Negara, R. M., & Sanjoyo, D. D. (2019). Implementasi & Analisis Performansi Layanan Web Pada Platform Berbasis Docker Implementation & Analysis of Web Service Performance Based on Docker Platform. *E-Proceeding of Engineering*, 6(2), 3996–4001. Retrieved from <https://librarye-proceeding.telkomuniversity.ac.id/index.php/engineering/article/viewFile/10367/10222>
- Joy, A. M. (2015). Performance comparison between Linux containers and virtual machines. *Conference Proceeding - 2015 International Conference on Advances in Computer Engineering and Applications, ICACEA 2015*, (Lxc), 342–346. <https://doi.org/10.1109/ICACEA.2015.7164727>
- Lingayat, A., Badre, R. R., & Gupta, A. K. (2018). Performance Evaluation for Deploying Docker Containers on Baremetal and Virtual Machine. *Proceedings of the 3rd International Conference on Communication and Electronics Systems, ICCES 2018*, (Icces), 1019–1023. <https://doi.org/10.1109/CESYS.2018.8723998>
- Linuxcontainer.org. (n.d.). Infrastructure for container projects. Retrieved December 15, 2019, from [Linuxcontainers.org](http://linuxcontainers.org)
- Novianta, M. A., & Setyaningsih, E. (2015). Sistem Informasi Monitoring Kereta Api Berbasis Web Server Menggunakan layanan GPRS. *Jurnal Momentum*, 17(2), 58–67.
- Openvz.org. (n.d.). Open source container-based virtualization for Linux. Retrieved December 15, 2019, from <https://openvz.org/>
- Pande, S., Bansal, P., & Hivarekar, P. (2016). DOCKER Application Management Web Portal. *Imperial Journal of Interdisciplinary Research*, 2(7), 2454–1362. Retrieved from <http://www.onlinejournal.in>
- Popek, G. J., & Goldberg, R. P. (1973). Formal requirements for virtualizable third generation architectures. *ACM SIGOPS Operating Systems Review*, 7(4), 121. <https://doi.org/10.1145/957195.808061>
- Rad, B. B., Bhatti, H. J., & Ahmadi, M. (2017). An Introduction to Docker and Analysis of its Performance. *IJCSNS International Journal of Computer Science and Network Security*, 17(3), 228–235.
- Turnbull, J. (2014). *The Docker Book : Containerization is A the New Virtualization*. Melbourne: Turnbull Press.
- Zhang, Q., Liu, L., Pu, C., Dou, Q., Wu, L., & Zhou, W. (2018). A Comparative Study of Containers and Virtual Machines in Big Data Environment. *IEEE International Conference on Cloud Computing, CLOUD, 2018-July*, 178–185. <https://doi.org/10.1109/CLOUD.2018.00030>