ANALISIS PENGGUNAAN *SYSTEM PROCESS* PADA MIGRASI APLIKASI DALAM LINUX CONTAINER(LXD) MENGGUNAKAN LXD API

ANALYSIS OF THE USE OF THE PROCESS SYSTEM IN APPLICATION MIGRATION IN LINUX CONTAINER(LXD) USING LXD API

Aditya Eka Saputra[1], Avon Budiono[2], Adityas Widjajarto[3]

[1,2,3]Prodi S1 Sistem Informasi, Fakultas Rekayasa Industri, Universitas Telkom
[1]adityaeka@student.telkomuniversity.ac.id, [2]avonbudi@telkomuniveristy.co.id,
[3]adtwjrt@telkomuniversity.ac.id

Abstrak

Linux Container adalah sekumpulan satu atau lebih suatu proses yang diisolasi dari seluruh sistem. Semua file yang diperlukan untuk menjalankannya disediakan dari gambar yang berbeda, yang berarti bahwa Linux Container *portable*. Dalam menggunakan Linux Container akan merujuk sumber daya virtualisasi dengan menggunakan Ubuntu sebagai sistem operasi. Studi ini membahas penggunaan proses sistem pada CPU terkait dengan proses migrasi *container* di mana ada beberapa aplikasi yang dilakukan dari berbagai kondisi. Seperti, mengukur proses sistem ketika *container* sebelum dimigrasi, saat migrasi *container* berlangsung, dan setelah migrasi *container* selesai. Setiap proses menggunakan spesifikasi *processor* dengan *core* yang berbeda-beda. Proses ini dilakukan untuk menentukan penggunaan CPU saat melakukan proses migrasi *container* menggunakan LXD API dari *platform* satu *platform* ke *platform* lainnya. Lalu, untuk mengetahui penggunaan CPU pada waktu sebelum migrasi, saat migrasi, dan setelah migrasi pada *container* yang nantinya menjadi acuan untuk mengetahui penggunaan dari suatu aplikasi yang ada dalam *container* yang bertujuan untuk tolak ukur pada *user* pada penggunaan aplikasi dalam *container*. Berdasarkan hasil di atas, dapat disimpulkan bahwa jumlah aplikasi dalam *container* memiliki pengaruh pada penggunaan CPU selama proses migrasi *container* dan jumlah *core* pada *processor* mempengaruhi kondisi responsif atau tidak responsifnya dalam menggunakan suatu layanan disaat mengakses banyak aplikasi layanan yang terdapat dalam *container*.

Kata Kunci : Linux Container, *Container*, migrasi, *processor*, CPU, LXD API

Abstract

Linux Container is a set of one or more processes that are isolated from the entire system. All files needed to run it are provided from different images, which means Linux Container is portable. In using Linux Container will refer to virtualization resources by using Ubuntu as an operating system. This study discusses the use of system processes on the CPU related to the container migration process where there are several applications that are carried out from various conditions. Like, measuring the system process when the container is before being migrated, when the container migration takes place, and after the container migration is complete. Each process uses processor specifications with different cores. This process is carried out to determine CPU usage when migrating containers using the LXD API from one platform to another platform. Then, to find out the CPU usage at the time before migration, during migration, and after migration to the container which later becomes a reference to find out the use of an application in the container that aims to measure the user on the use of the application in the container. Based on the above results, it can be concluded that the number of applications in the container has an influence on CPU usage during the container migration process and the number of cores on the processor affects the responsive or unresponsive conditions in using a service while accessing many service applications contained in the container.

***Key Word : Linux Container, Container, migration, processor,* CPU, LXD API.**

1. Pendahuluan.

Teknologi informasi ini merupakan kombinasi teknologi komputer yang terdiri dari perangkat keras dan lunak untuk mengolah dan menyimpan informasi dengan teknologi komunikasi untuk melakukan penyaluran informasi [1].

Saat ini sudah banyak perusahaan menggunakan layanan bisnis yang menunjang proses bisnis pada perusahaannya. Layanan yang dipakai seperti layanan *web* seperti odoo, mysql, SAP, PHP, dan Apache. Layanan layanan tersebut sangat penting untuk proses bisnis di perusahaan dikarenakan pada saat ini banyak perusahaan yang menggunakan layanan tersebut.

Sering terjadi kasus yang terjadi saat ini terkait dengan *down* sebuah layanan. Hal tersebut disebabkan karena berbagai faktor, diantaranya yaitu kegagalan sistem, *human error*, bencana alam, dan maintenance. Matinya sebuah layanan dapat mengganggu proses bisnis perusahaan dan berdampak kerugian yang disebabkan diantaranya matinya *server* dan kehilangan data. Untuk mengatasinya, perusahaan tersebut dapat melakukan perpindahan layanan dari suatu *server* ke *server* lainnya agar ketika pada saat *server* mati, layanan masih bisa dijalankan.

Virtualisasi Server adalah penggunaan perangkat lunak yang memungkinkan satu perangkat keras untuk menjalankan beberapa sistem operasi dan layanan secara bersamaan. Sebagai solusi dari permasalahan ini dapat dilakukan dengan cara migration dengan menggunakan *Virtual Machine* (VM) dan *container*. Tetapi pada VM, sering terjadi kegagalan saat proses migrasi dikarenakan sumber daya komputasi seperti CPU atau tidak kompatibel dengan *platform* yang digunakan [2].

Solusi lainnya dapat menggunakan *Linux Container*. *Linux Container* adalah sebuah *tools* untuk melakukan virtualisasi *server* yang didalamnya bisa di *install* berbagai sistem operasi Linux seperti, Ubuntu, Debian, dan CentOS untuk menjadi sebuah *server*. Dengan menggunakan *Linux Container*, dapat melakukan virtualisasi dilevel sistem operasi dan aplikasi-aplikasi *web server* [4].

Proses adalah program yang sedang dieksekusi/running. Proses merupakan bagian unit kerja terkecil dalam sistem operasi. ketika sebuah proses dibuat maka proses tersebut dapat memperoleh sumber daya seperti waktu CPU, memori, dan perangkat. Pada sistem operasi linux, *system process* dapat dilihat dengan menggunakan tools Htop.

LXD adalah API berbasis *platform* yang memungkinkan akses dan pengelolaan *container* dengan menggunakan HTTP. LXD merupakan sebuah *container*(wadah) yang digunakan untuk membangun sebuah sistem, situs *web*, dan lainnya. LXD API adalah suatu proses memindahkan *container* ke *container* lainnya agar saat suatu *container* berpindah ke *server* lain, *server* tersebut dan dapat digunakan langsung tanpa perlu *install* aplikasi yang ada di *server* sebelumnya.[7]

2. Dasar Teori

2.1 Jaringan

Menurut Jafar Noor Yudianto (2007) menyatakan bahwa jaringan komputer ialah suatu sistem yang terdiri atas sebuah komputer-komputer yang didesain untuk bisa berbagi sumber daya (printer, CPU), berkomunikasi (surel, pesan instan), dan bisa mengakses informasi (peramban *web*) [5].

2.2 *Container-Based Virtualization*

Virtualisasi adalah teknik dalam membuat suatu versi *virtual* dari suatu sumber daya. Dengan melakukan virtualisasi, dapat menjalankan dan menyimpan beberapa sumber daya *virtual* dalam satu sumber daya fisik. *Container-Based Virtualization* adalah salah satu dari tiga jenis virtualisasi yang ada selain *hardware virtualization* dan *paravirtualization* [3].

2.3 Linux Container

Linux Container adalah sekumpulan satu atau lebih suatu proses yang diisolasi dari seluruh sistem. Semua file yang diperlukan untuk menjalankannya disediakan dari gambar yang berbeda, yang berarti bahwa Linux Container *portable* dan konsisten ketika mereka berpindah dari pengembangan menuju ke pengujian, dan akhirnya ke produksi. Linux Container merupakan generasi dari pengaturan sistem *container* selanjutnya yang menawarkan *user experience* yang mirip dengan *virtual machine* akan tetapi menggunakan Linux Container sebagai gantinya [4].

2.4 Processor

Processor merupakan salah satu komponen penting komputer yang berfungsi untuk memproses data dan mengontrol sistem yang ada pada komputer. *Processor* juga bisa disebut sebagai otak dari komputer. Secara singkat, processor bekerja untuk melakukan perhitungan serta menjalankan perintah-perintah yang diperintahkan oleh pengguna komputer itu sendiri. *Processor* merupakan sebuah chip yang disebut "*Microprocessor*" yang berfungsi untuk memproses data yang diterima dari perangkat masukan (*input*) kemudian memprosesnya dan menghasilkan keluaran (*output*) [8].

Bagian terpenting dari prosesor adalah:

a. *Aritcmatics Logical Unit* (ALU) : Melakukan semua perhitungan aritmatika (matematika) yang terjadi sesuai dengan intruksi program.
b. *Control Unit* (CU) : Pengatur lalu lintas data seperti *input*, dan *output*.
c. *Memory Unit* (MU) : Alat penyimpanan kecil yang mempunyai kecepatan akses cukup tinggi.

2.5 *Core Processor*

Core Processor merupakan *processing unit* yang membaca semua instruksi untuk melakukan tindakan tertentu. Suatu instruksi ditampung ketika dijalankan secara *realtime*. Sebuah *core* biasanya hanya bisa menjalankan satu program, selain itu *core* juga mempertahankan *cache core* dengan salinan yang seing digunakan di sebagian memori.

Meskipun *processor* berbeda arsitekturnya, tetapi *processor* melalui empat langkah utama yang sama setiap kali memproses *task*, yaitu [9].

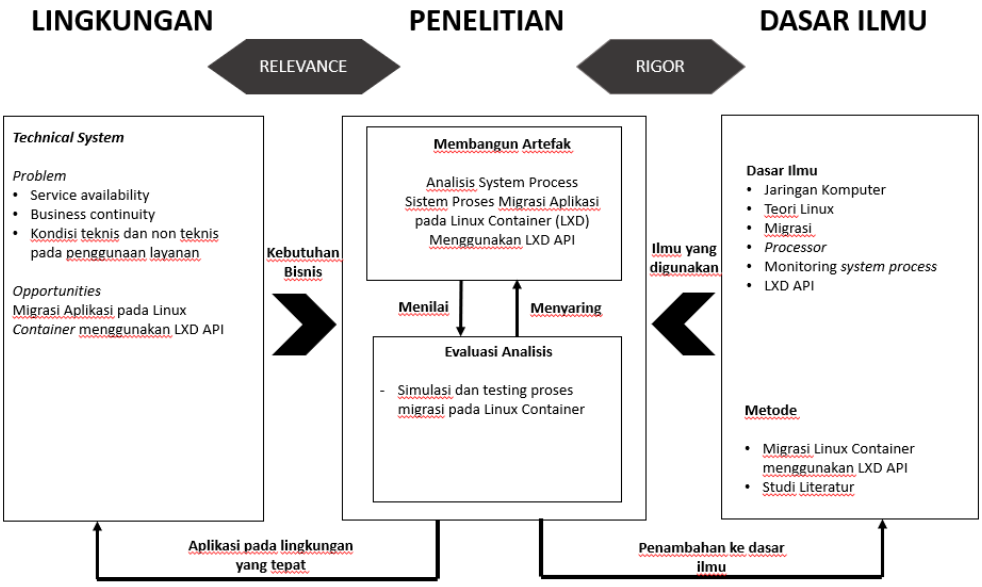
a. *Fetch*: Langkah dimana *user* ingin melakukan sesuatu dan *core processor* menginstruksikan dan menunggu untuk melakukan proses, biasanya diambil dari beberapa bagian memori, termasuk dari RAM
b. *Decode*: Suatu instruksi yang sering melibatkan beberapa c*ore processor* seperti aritmatika dan *core processor*. Setiap bagian memeiliki sesuatu yang disebut opcode yang mana *decode* memberitahukan ke *core processor* apa yang harus dilakukan terhadap infomasi yang diberikan.
c. *Execute*: Suatu langkah pada saat *processor* mengetahui apa yang harus dilakukan. Yang terjadi disini biasanya bervariasi, tergantung dari informasi yang diberikan oleh *core processor*.
d. *Writeback*: Yaitu hasil yang telah dilakukan akan dikirim ke memori untuk menampilkan *output* tergantung dari kebutuan aplikasi akan bekerja. Pada saat berada dalam *register processor* untuk mendapatkan akses yang lebih cepat untuk instruksi selanjutnya. Lalu akan disampaikan pada *output* yang perlu di proses lagi yang nantinya bisa dikirim ke RAM

3. Metode Penelitian

3.1. Model Konseptual

Menurut Jan Joker, dkk (2011) konseptual model yaitu kerangka kerja yang menggambarkan hubungan kasual antara faktor faktor yang saling berkaitan. Fungsi model konseptual antara lain [6]:

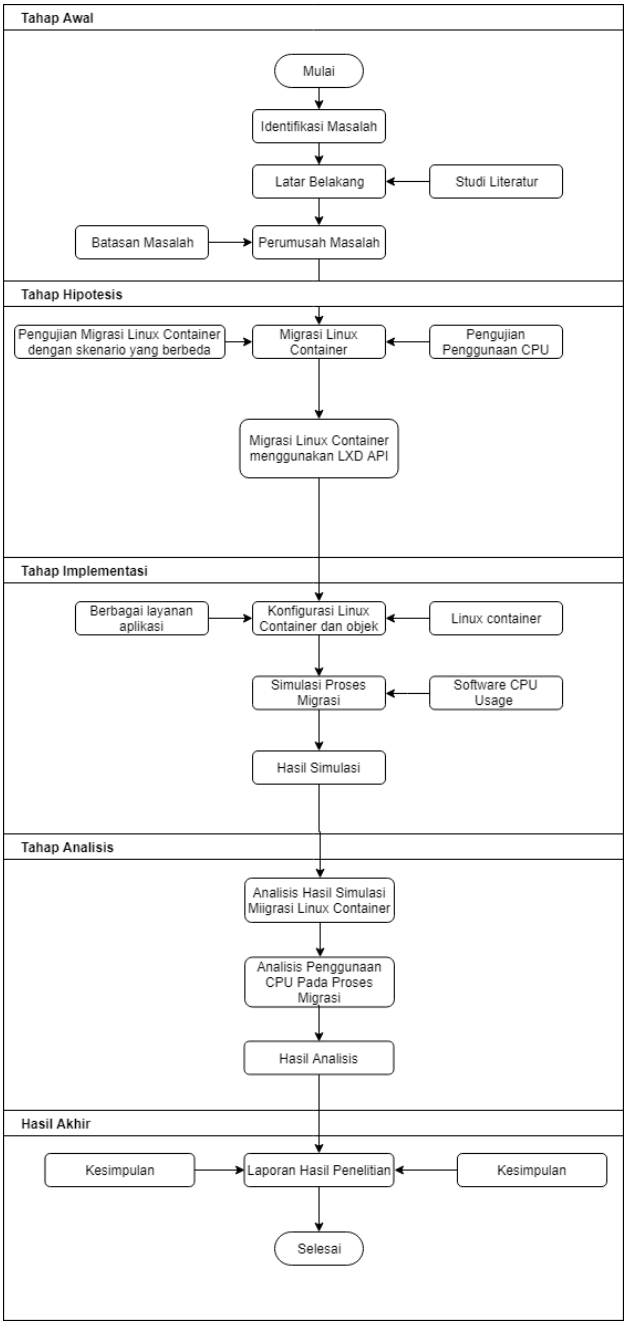
a) Fungsi pertama dari model konseptual sangat erat hubungannya dengan teori referensi/literatur yang digunakan
b) Fungsi kedua alah pembangunan model dapat membantu dalam penataan masalah, mengidentifikasi faktor-faktor relevan dan kemudian, memberikan koneksi yang membuatnya lebih mudah untuk memetakan bingkai masalahnya
c) Fungsi ketiga adalah menghubungkannya kedalam sistem teori



Gambar 1 Model Konseptual

3.2. Sistematika Penelitian

Sistematika peneilitan merupakan gambaran secara keseluruhan tentang hal apa saja yang akan dilakukan selama penelitian ini berlangsung. Tahapan yang dilakukan antara lain: rumusan masalah, hipotesa, implementasi, analisis dan tahap akhir.



Gambar 2 Sistematika Penulisan

4.1 Instrument pengujian

Instrument Program yaitu berisi komponen seperti *tools*, *software*, *operation system* yang akan digunakan untuk melakukan simulasi. Instrument pengujian biasanya berupa *software* atau *hardware*. Berikut ini instrument pengujian yang akan dilakukan sebagai berikut:

Tabel 1 Spesifikasi Hardware dan Software

	Komponen	Informasi
Hardware	Asus A456U	1. Processor: Intel® Core™ i5-7200U Processor (2.5 GHz, 3M Cache) up to 3.10 GHz 2. Memori: 8GB DDR4 3. GPU: Intel HD Graphics 620 dan NVIDIA GeForce GT930MX 2GB 4. Harddisk: 1TB HDD 5. OS: Windows 10 64-bit
	Asus X550V	1. Processor: Intel® Core™ i7-7700HQ Processor (Up to 3.8GHz) 2. Memori: 8GB DDR4 3. GPU: Intel HD Graphics 630 dan NVIDIA GeForce GTX950M 2GB 4. Harddisk: 1TB HDD 5. OS: Windows 10 64-bit
	Access Point	ZTE
Software	Linux Ubuntu	16.04.06 LTS
	Container	2.0.11
	LXD Container	2.0.11
	VMware Workstation Pro	12.5.4build-5192485
	Htop	2.6.3
	Nginx	1.10.0
	Odoo	v.11
	Apache	2.2.15
	MySQL	8.0.11
	phpMyAdmin	4.8.2
	Wordpress	4.4.2

4.2 Skenario Pengujian.

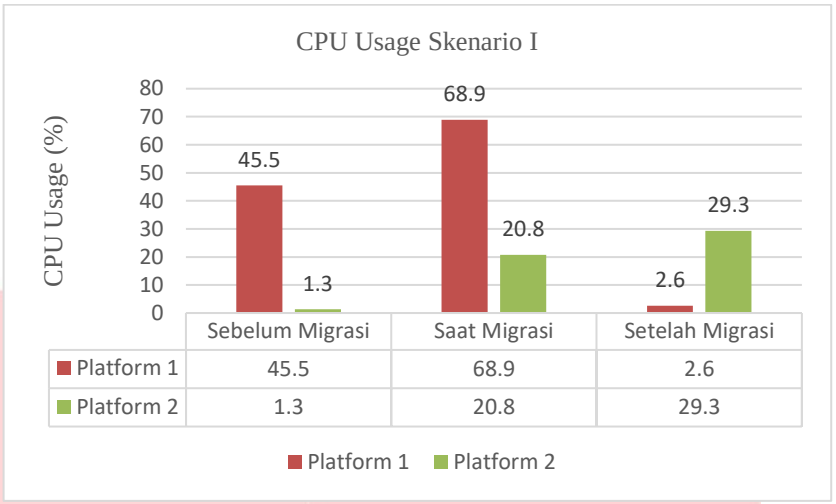
Untuk pengujian analisa proses sistem pada linux container dengan menggunakan LXD API ada beberapa scenario untuk menunjang proses migrasi *container* selama berjalan, proses pengambilan data dalam *system process*. Skenario tersebut diantaranya:

1. Pengujian migrasi satu arah (*platform* 1 ke *platform* 2)
2. Pengujian migrasi satu arah (*platform* 2 ke *platform* 1)
3. Pengujian migrasi dua arah (*platform* 1 ke *platform* 2 dan *platform* 2 ke *platform* 1)
4. Pengujian migrasi dua arah (*platform* 2 ke *platform* 1 dan *platform* 1 ke *platform* 2)

4.3 Hasil Pengujian dan analisis

4.3.1 Skenario 1: pengujian migrasi satu arah (*Platform* 1 ke *Platform* 2)

pengujian dilakukan dengan cara melihat CPU *usage* di *platform* 1 dan *platform* 2 dengan ukuran *container* berisi 2 aplikasi.

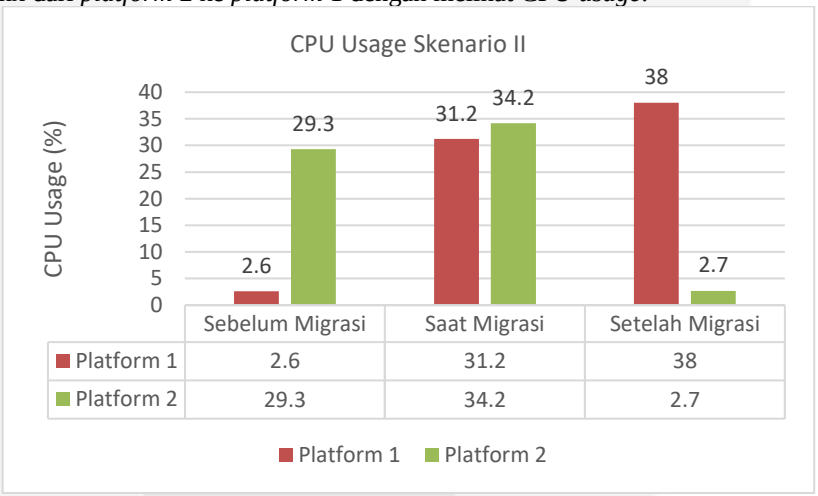


Gambar 3 Grafik CPU Usage Skenario 1

Pada pengujian *processor* dengan *core* 1 sebelum migrasi aplikasi 2 layanan yaitu odoo dan nginx dan mengakses layanan pada *platform* 1 menggunakan CPU paling tinggi sekitar 45,5% dan *platform* 2 menggunakan CPU paling tinggi 1,3%, pada saat migrasi *container*, *container* di stop terlebih dahulu supaya dapat dimigrasikan, *platform* 1 menggunakan CPU paling tinggi 68,9 dan *platform* 2 menggunakan CPU paling tinggi 20,8%, proses pengiriman container memakan waktu 12,25 menit. Dan setelah melakukan migrasi *container*, pada *platform* 1 menggunakan CPU paling tinggi 2,6% dan *platform* 2 menggunakan CPU paling tinggi 29,3%.

4.3.2 Skenario 2: pengujian migrasi satu arah (*Platform* 2 ke *Platform* 1)

Pengujian dilakukan dengan cara mengirim kembali *container* yang berisi 2 aplikasi yaitu odoo dan nginx dari *platform* 2 ke *platform* 1 dengan melihat CPU *usage*.

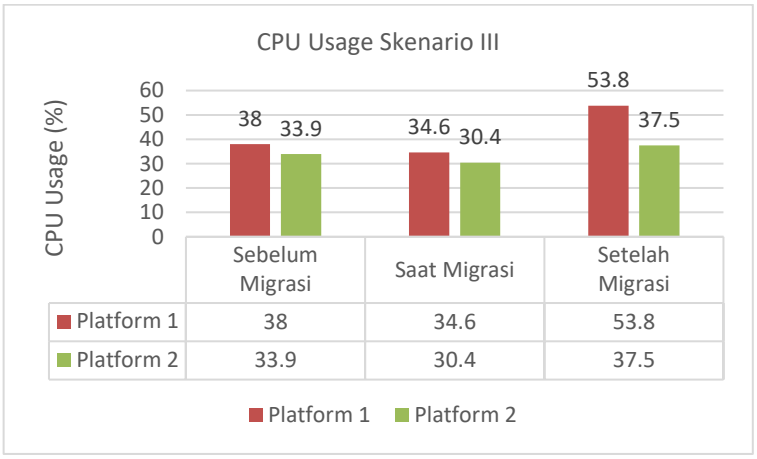


Gambar 4 Grafik CPU Usage Skenario II

Pada pengujian *processor* dengan *core* 1 sebelum migrasi aplikasi 2 layanan yaitu odoo dan nginx dan mengakses layanan pada *platform* 1 menggunakan CPU paling tinggi sekitar 2,6% dan *platform* 2 menggunakan CPU paling tinggi 29,3%, pada saat migrasi *container*, *container* di stop terlebih dahulu supaya dapat dimigrasikan, *platform* 1 menggunakan CPU paling tinggi 31,2 dan *platform* 2 menggunakan CPU paling tinggi 34.2%, proses pengiriman container memakan waktu 9,55s. Dan setelah melakukan migrasi container, pada *platform* 1 menggunakan CPU paling tinggi 38% dan *platform* 2 menggunakan CPU paling tinggi 2,7%.

4.3.3 Skenario 3: pengujian migrasi dua arah mengirim (*Platform* 1 ke *Platform* 2 dan *Platform* 2 ke *Platform* 1)

Pengujian dilakukan dengan cara mengirim kembali *container* yang berisi 2 aplikasi dari kedua *platform* yang sama-sama mengirim *container* dengan melihat CPU *usage*.

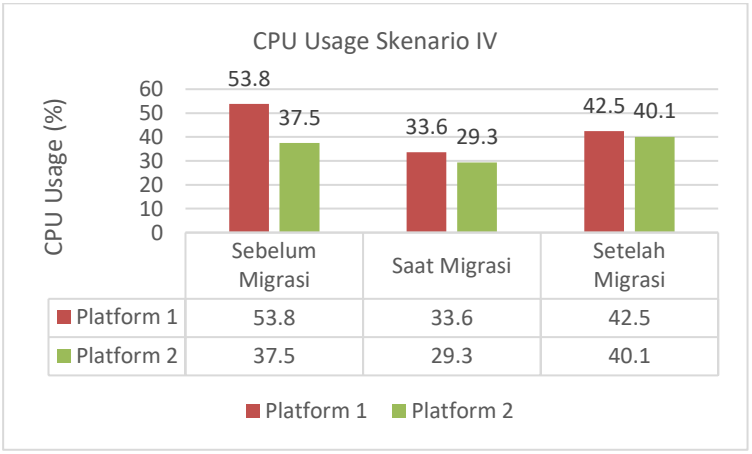


Gambar 5 Grafik CPU Usage Skenario III

Pada pengujian *processor* dengan *core* 1 sebelum migrasi aplikasi 2 layanan yaitu odoo dan nginx dan mengakses layanan pada *platform* 1 menggunakan CPU paling tinggi sekitar 38% dan *platform* 2 menggunakan CPU paling tinggi 33,9%, pada saat migrasi *container*, *container* di stop terlebih dahulu supaya dapat dimigrasikan, *platform* 1 menggunakan CPU paling tinggi 34,6% dan *platform* 2 menggunakan CPU paling tinggi 30,4%, proses pengiriman *container* dari *platform* 1 ke *platform* 2 dan *platform* 2 ke *platform* 1 memakan waktu sekitar 19,19 menit dan 18,27 menit. Dan setelah melakukan migrasi container, pada *platform* 1 menggunakan CPU paling tinggi 53,8% dan *platform* 2 menggunakan CPU paling tinggi 37,5%.

4.3.4 Skenario 4: pengujian migrasi dua arah membalikkan container (*Platform* 1 ke *Platform* 2 dan *Platform* 2 ke *Platform* 1)

Pengujian dilakukan dengan cara mengirim kembali *container* yang berisi 2 aplikasi yaitu odoo dan nginx dari kedua *platform* yang sama-sama mengirim kembali *container* dengan melihat CPU *usage*.



Gambar 6 Grafik CPU Usage Skenario IV

Pada pengujian *processor* dengan *core* 1 sebelum migrasi aplikasi 2 layanan yaitu odoo dan nginx dan mengakses layanan pada *platform* 1 menggunakan CPU paling tinggi sekitar 53,8% dan *platform* 2 menggunakan cpu paling tinggi 37,5%, pada saat migrasi *container*, *container* di stop terlebih dahulu supaya dapat dimigrasikan, *platform* 1 menggunakan CPU paling tinggi 33,6% dan *platform* 2 menggunakan CPU paling tinggi 29,3%, saat proses pengiriman *container* dari *platform* 1 ke *platform* 2 dan *platform* 2 ke *platform* 1 memakan waktu sekitar 19,54 menit dan 20,13 menit. Dan setelah melakukan migrasi *container*, pada *platform* 1 menggunakan CPU paling tinggi 42,5% dan *platform* 2 menggunakan CPU paling tinggi 40,1%.

4. Kesimpulan

Berdasarkan analisis dari pengujian dan *monitoring* yang dilakukan pada skenario pada CPU yang digunakan ketika migrasi *container* dapat diambil kesimpulan sebagai berikut:

- Proses migrasi *container* dengan menggunakan LXD API yang dilakukan dengan melakukan *remote* antar *platform* melalui pengenalan IP *Address*, untuk kedua *platform* harus berada didalam satu jaringan yang sama dan pada masing-masing *platform* sudah installasi LXD *container*. LXD API berfungsi untuk proses migrasi *container* yang berisi aplikasi dan pada saat dimigrasikan harus dalam keadaan *stop* terlebih dahulu supaya bisa di migrasikan.
- Jumlah *Core* dan spesifikasi *processor* juga mempengaruhi penggunaan CPU saat mengakses *container*, semakin banyak jumlah *Core* yang digunakan akan semakin rendah penggunaan CPU dan Jumlah *Core processor* tidak mempengaruhi cepat atau lambatnya pada proses migrasi container.
- Pengimplementasian migrasi *container* pada Linux Container juga harus memperhatikan stabilitas dan kompatibel suatu aplikasi pada sistem operasi maupun versi *container* yang digunakan.

Daftar Pustaka:

[1] Martin, Brown, DeHayes, Hoffer, dan Perkins. 2005. Diakses tanggal 13 juli 2019, http://nicodarmawan.blog.ugm.ac.id/2011/09/

[2] Petter Svard, S. W. (2014). *A The Noble Art of Live VM Migration - Principles and Performance of precopy, postcopy and hybrid migration of demanding workloads*. Retrieved 06 23, 2019, from https://pdfs.semanticscholar.org/e31d/f36d516d88d9233f4eb6de9660cce23cfe7c.pdf?_ga=2.10145251.328105095.1563841025-54939759.1563841025

[3] R. Morabito, "Virtualization on Internet of Things Edge Devices with Container Technologies : a Performance Evaluation," vol. 3536, no. c, 2017.

[4] Linux Container.[Online]. Diakses tanggal 12 juli 2019. Available: https://www.redhat.com/en/topics/containers/whats-a-linux-container

[5] Yudianto, M. Jafar Noor. 2007. Diakses tanggal 2 November 2018, http://ilmukomputer.org/wp-content/uploads/2013/01/Ilmu-komputer-Jaringan-Komputer-Dan-Pengertiannya.pdf.

[6] Pengurus Forum Inovasi Bisnis dan Manajemen Indonesia . Jurnal INOBIS Vol. 1 No. 2 Hal Maret 2018 ISSN (Online).pdf

[7] LXD API. [Online]. Diakses tanggal 12 juli 2019. Available : https://linuxcontainers.org/lxd/rest-api/

[8] S. Singh Ghuman, "Comparison of Single-Core and Multi-Core Processor," *International Journal of Advanced Research in*, vol. 6, no. 6, 2016.

[9] Chrisnado, E. (2015, May 27). *Apa itu Core di Processor*. Retrieved July 23, 2019

[10] Setiawan, R. (2017). Diakses Agustus 21, 2019, from Sistem Operasi: https://books.google.co.id/books/about/Sistem_Operasi.html?id=rqg-DwAAQBAJ&redir_esc=y