Perbandingan Kinerja Linux Container dan Docker Container untuk Skema Migrasi Layanan

Comparison of Linux Container and Docker Container Performance for Service Migration Scheme

Muhammad Valdi Harris, Ir. Dr. Rendy Munadi, M.T., Dr.Eng. Favian Dewanta, ST., M.Eng.
Prodi S1 Teknik Telekomunikasi, Fakultas Teknik Elektro, Universitas Telkom
valdihaha@student.telkomuniversity.ac.id, **rendymunadi@telkomuniversity.ac.id**
favian@telkomuniversity.ac.id.

Abstrak

Layanan yang berkualitas ditentukan dengan bagaimana sebuah layanan dapat tetap memiliki performansi yang baik atau mampu bekerja secara optimal meskipun terjadi sebuah perubahan, baik perubahan lokasi maupun perangkat yang digunakan oleh layanan tersebut. Namun terkadang proses perpindahan suatu layanan tersebut menyebabkan terjadinya penurunan Quality of Service (QoS) terutama untuk layanan-layanan yang bersifat kompleks. Maka dari itu, untuk menyederhanakan kompleksitas tersebut dilakukan penggunaan *container*. Karena container dapat langsung berjalan diatas Sistem Operasi (OS) tanpa menggunakan *hypervisor* sehingga container lebih bersifat fleksibel secara keseluruhan dibandingkan dengan penggunaan *server* secara fisik maupun *virtual machine*.

Dalam penelitian ini akan dilakukan perbandingan terhadap penggunaan *Linux Container* (LXC) dan *Docker Container* pada proses migrasi layanan saat dilakukan *live streaming Real-Time Message Protocol* (RTMP) dan saat database MySQL berjalan, dimana performansi kedua container tersebut dinilai dari aspek *application downtime*, QoS (*Throughput, Delay, Jitter dan Packet Loss)* serta utilisasi CPU dan *memory*. Perbandingan ini dilakukan untuk menentukan *container* mana yang lebih efektif dalam proses migrasi layanan.

Hasil penelitian ini menunjukkan bahwa penggunaan kedua *container* tersebut mampu mempertahankan kondisi layanan dinilai dari hasil QoS yang dapat dikategorikan baik. Namun dari segi utilisasi CPU dan *memory*, *docker* memiliki konsumsi CPU dan memory yang lebih kecil. Begitu juga dilihat dari segi *downtime docker* memiliki performansi yang lebih baik, dimana migrasi RTMP dengan menggunakan LXC memiliki *downtime* selama 52.5485s dan pada docker memiliki *downtime* selama 5.4345s. Sedangkan pada migrasi MySQL, LXC server memiliki *downtime* selama 35.3939s dan pada docker memiliki *downtime* selama 34.6957s. Sehingga dapat disimpulkan bahwa *docker* lebih efektif dalam proses migrasi layanan.

Kata kunci : Migrasi layanan, LXC, Docker Container

Abstract

Quality service is determined by how a service can still have good performance or is able to continue to work optimally even though there is a change, such as a change in location or the device used by the service. However, sometimes the process of moving a service causes a decrease in Quality of Service (QoS), one of the reasons is the complexity of the service environment. Containers are applications that can simplify this complexity by virtualizing a simpler machine based on what the service need only.

In this research, a comparison will be made of the use of Linux Container (LXC) and Docker Container in the service migration process in the form of live streaming Real-Time Message Protocol (RTMP) and a MySQL database, where the performance of the two containers is assessed from the aspects of application downtime and QoS (Throughput, Delay, Jitter and Packet Loss.) then CPU usage and memory usage. This comparison is done to determine which container is more effective for use in the service migration process.

The results of this research indicate that the use of the two containers is able to maintain service conditions judged by the QoS results which can be categorized as good, with each container having a QoS Throughput, delay, and packet loss index of 4, then the jitter for both

containers is 3. Then for CPU usage and memory usage of each container is not much different from Docker having much lighter resources.

However, in terms of downtime, the docker container has better performance, where the migration to the LXC on the RTMP server has 52.5485s of downtime, and the docker RTMP server is 5.434533333s. Where as the LXC MySQL server has a downtime of 35.39386667s, and the docker MySQL server has a downtime of 36.69573333s.

Keywords: Live Migration, LXC, Docker.

1. Pendahuluan

Quality of Service merupakan salah satu aspek penting yang belakangan semakin diperhatikan. Hal ini disebabkan oleh meningkatnya pengguna layanan tetapi tidak di imbangi dengan perkembangan infrastruktur jaringan. Sehingga dibutuhkan suatu jaringan yang dapat beradaptasi dengan kebutuhan pengguna tanpa mengurangi kualitas dari layanan tersebut [1]. Hal ini semakin diperkuat dengan semakin besarnya kebutuhan *user* terhadap layanan yang tidak terbatas pada suatu lokasi. Sebagai contoh, *end-user* yang biasanya berada di suatu lokasi dengan jaringan *cloud* dan *fog* dapat berpindah ke jaringan yang lain mengikuti pergerakan lokasi *end-user* tetapi tetap mendapatkan QoS yang sama. Hal itu disebut sebagai kemampuan jaringan untuk melakukan *handover*, dengan cara melakukan sebuah migrasi layanan [2][3].

Contoh lainya adalah layanan yang bersifat *real-time* dan memiliki jumlah data tidak terbatas seperti penggunaan *drone*. Sebagai contoh ketika menggunakan sebuah *drone* untuk melakukan observasi bencana alam [4][5]. Data yang direkam ketika melakukan observasi akan menggunakan komputasi *resource* yang cukup besar. Permasalahan muncul ketika kapasitas *resource* dari alat yang digunakan terbatas. Oleh karena itu dengan memanfaatkan sebuah migrasi layanan ketika *hardware* tersebut telah penuh, *drone* yang terkoneksi pada jaringan berupa *cloud* dan *fog* dapat memindahkan data ke suatu *server* penyimpanan atau mengirimkan instruksi ke *drone* lain untuk menggantikan tugas drone tersebut[6]. Proses tersebut dapat disebut dengan migrasi layanan. Agar dapat melakukan skenario migrasi tersebut dengan cara yang lebih sederhana, dapat digunakan dua layanan yaitu *real-time transfer protocol* (RTMP) yang mewakilkan layanan *real-time*, kemudian *server* MySQL yang mewakilkan migrasi data [7].

Migrasi layanan sendiri adalah suatu proses untuk menyimpan suatu kondisi layanan pada suatu perangkat kemudian kondisi tersebut dapat dijalankan kembali pada perangkat lain tanpa adanya perubahan kualitas. Namun, biasanya proses migrasi seringkali berpengaruh terhadap kualitas layanan, hal ini disebabkan oleh proses membangun kembali layanan yang membutuhkan waktu yang cukup lama karena memiliki susunan yang kompleks [2][3]. Agar dapat menyederhanakan kompleksitas tersebut, layanan dapat dijalankan di suatu platform berbentuk container [2][3].

Oleh karena itu, pada Tugas Akhir ini akan dilakukan simulasi untuk membandingkan dua container yang dapat digunakan dalam melakukan migrasi layanan, yaitu *Linux Container* (LXC) dan *Docker Container*. Kemudian akan dilakukan analisis performa terhadap layanan RTMP dan MySQL yang berjalan dan dimigrasikan diatas kedua container tersebut, dengan aspek penilaian *application downtime*, *Quality of Service* (QoS) serta utilisasi *memory* dan CPU [3][8], sehingga dapat menjadi referensi migrasi layanan bagi dua tipe container yang sering digunakan saat ini

2. Dasar Teori /Material dan Metodologi/perancangan

2.1 Container dan Linux Container

Container merupakan teknik virtualisasi sumber daya yang lebih fleksibel karena *container* melakukan virtualisasi *Operating System* (OS) dan *environment* dengan menggunakan sumber daya dari host di mana *container* berada sehingga tidak perlu dilakukan virtualisasi *hardware*. Berkat kemampuan ini, layanan yang digunakan dapat memanfaatkan semua sumber daya yang disediakan untuk layanan tersebut, sehingga sumber daya yang digunakan akan lebih hemat dan ringan serta layanan akan berjalan lebih cepat. Salah satu contoh container adalah *Linux container* (LXC) [9][10].

2.2 Docker Container

Docker Container merupakan pengembangan dari *container* LXC. Perbedaannya terdapat pada virtualisasi yang dilakukan *container* lebih spesifik kepada aplikasi. Dengan cara membentuk *template* dari berbagai macam OS dalam bentuk paling sederhana untuk menjalankan sebuah layanan. Hal ini membuat *docker* menjadi sebuah virtualisasi yang lebih ringan, karena *docker* tidak mengisolasi sebuah OS secara keseluruhan seperti yang dilakukan oleh LXC [11].

2.3 Migrasi layanan dan Container Migration menggunakan Checkpoint/Restore In Userspace (CRIU)

Kemampuan CRIU dalam membentuk suatu *checkpoint* yang menyimpan *state* dari *memory, network*, maupun *disk* yang kemudian dapat dipindahkan dan dijalankan di *host* lain disebut sebagai Migrasi layanan. CRIU dapat digunakan untuk migrasi layanan berbasis *container* [2]. Hal ini digunakan *pada cloud-based architecture network* dalam bentuk *live migration*, dimana layanan cloud dipindahkan dari satu host ke host lain secara live [2][3].

2.4 Real-Time Messaging Protocol (RTMP)

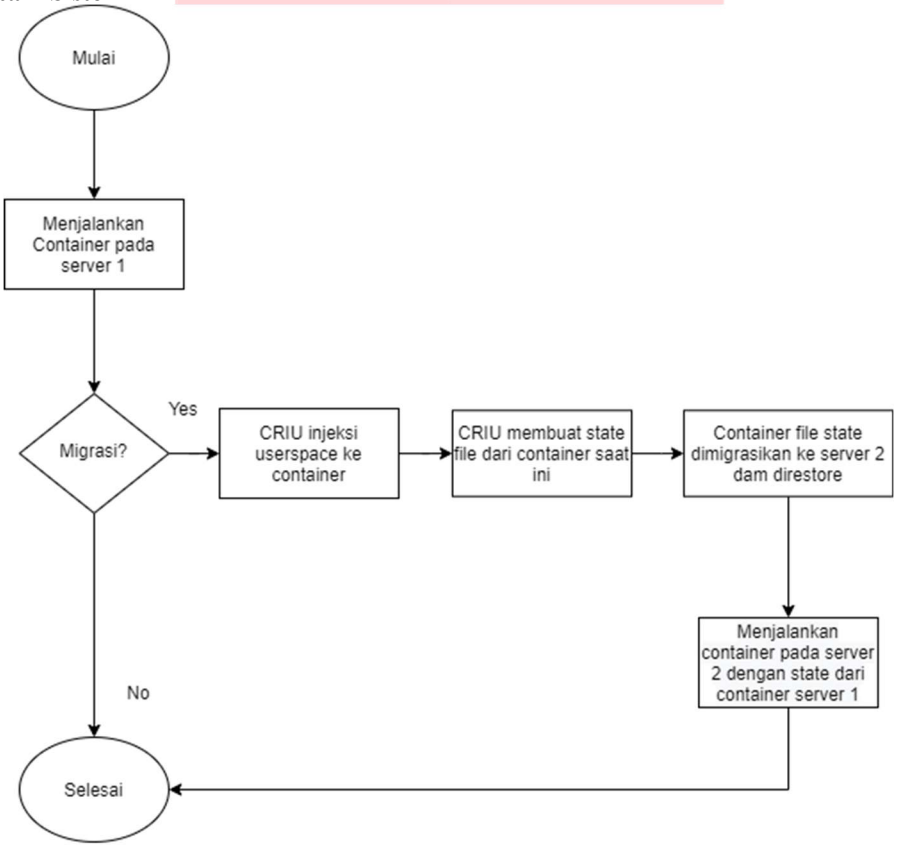
RTMP merupakan *application-level protocol* yang digunakan untuk *live streaming* dengan menggunakan *Transmission control protocol* (TCP) sebagai *transport protocol*. RTMP dapat melakukan *multiplexing* pada paket yang berisi data, *video*, dan *audio* untuk menjadi satu *chunk* paket [12]. *Server* RTMP sendiri menggunakan modul RTMP yang berbasis NGINX.

2.5 MySQL

MySQL adalah sistem manajemen *database open source* yang sering dipakai. MySQL menggunakan *syntax Structured Query Language* (SQL) berbasis *client* dan *server* model. Cara kerja MySQL adalah dengan membentuk sebuah *server database* yang di akses oleh *client* untuk menggunakan informasi di dalam *database* tersebut [13].

3. Pembahasan

3.1. Desain Sistem

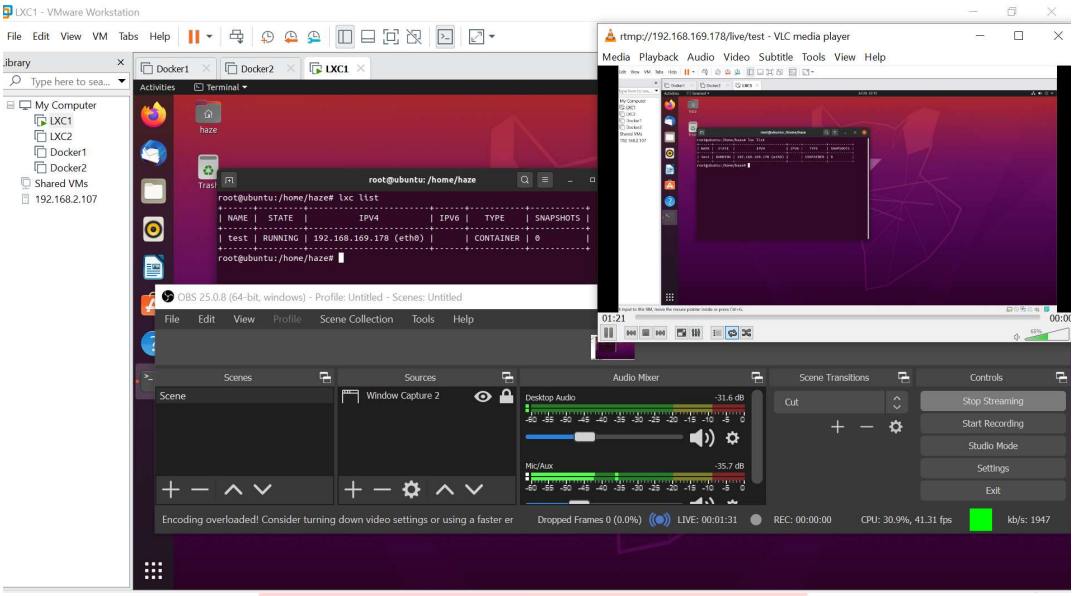


Gambar 3.1 Diagram Alir Sistem.

Dalam menjalankan sistem tersebut, Tugas Akhir ini menggunakan skenario dari **Gambar 3.1** seperti berikut :

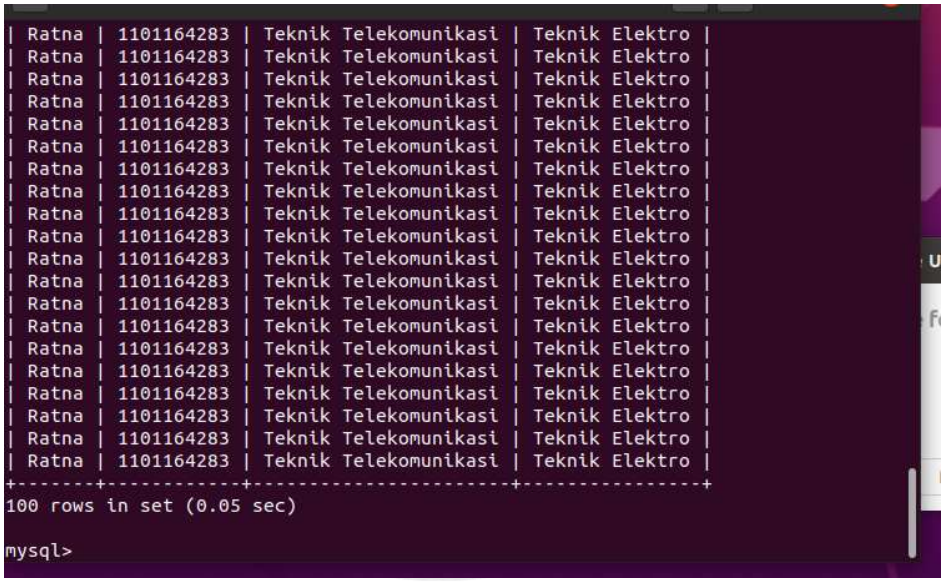
3.1.1 Menjalankan container pada server 1

Container pada server 1 melakukan *streaming* secara *real time* yang dapat diakses oleh user dengan menggunakan aplikasi *OBS* dan *VLC*. Durasi *live streaming* untuk pengambilan data dilakukan dalam 1 menit yang dapat dilihat pada **Gambar 3.2.**.



Gambar 3.2 Menjalankan *live stream*

Kemudian untuk MYSQL, disediakan database yang telah diisi dengan 100 data untuk menjadi beban. Database tersebut dapat dilihat pada **Gambar 3.3.**



Gambar 3.3 *Database* MYSQL

3.1.2 Migrasi

Hal ini dilakukan dengan mengeksekusi *shellscript* yang telah dibuat dengan menjalankan *command stop*, *create checkpoint*, migrasi, dan menjalankan kembali *container* setelah berada pada *server* tujuan, saat migrasi berlangsung akan terjadi *downtime* [2].

3.1.3 Migrasi pada LXC

Pada LXC, untuk dapat melakukan migrasi diperlukan konektivitas antara dua LXC dengan menggunakan fitur *remote list yang* dapat dilihat pada **Gambar 3.4**. Kemudian *container* akan berpindah dari suatu mesin ke mesin lain yang tidak memiliki

container. Migrasi menggunakan *shellscript* pada **Gambar 3.5** dengan hasil yang dapat dilihat pada **Gambar 3.6**.

NAME	URL	PROTOCOL	AUTH TYPE	PUBLIC	STATIC
images	https://images.linuxcontainers.org	simplestreams	none	YES	NO
local (default)	unix://	lxd	file access	NO	YES
server2	https://192.168.169.156:8443	lxd	tls	NO	NO
ubuntu	https://cloud-images.ubuntu.com/releases	simplestreams	none	YES	YES
ubuntu-daily	https://cloud-images.ubuntu.com/daily	simplestreams	none	YES	YES

Gambar 3.4 Fitur r*emote list* pada LXC

```
 GNU nano 4.8                                    migration.sh
#!/bin/bash

lxc stop test

lxc move test server2:test

sshpass -p ' ' ssh gravel@192.168.169.156 'echo " " | sudo -S lxc start test'
```

Gambar 3.5 *Shellscript* untuk migrasi

Fungsi setiap *command* adalah sebagai berikut:

1. *lxc stop nama_container* untuk memberhentikan *container.*
2. *lxc move test server2:nama_container* untuk migrasi *container*.
3. *SSHPASS -p ‘ ‘ ssh username @ ip.address ‘echo “ “ | sudo -s lxc start nama_container* untuk menjalankan *container* pada server 2.

```
root@ubuntu:/home/haze# time ./migration.sh
[sudo] password for gravel:
real    0m36.951s
user    0m0.163s
sys     0m0.189s
root@ubuntu:/home/haze#
```

Gambar 3.6 Migrasi pada LXC

3.1.4 Migrasi pada Docker

Berbeda dengan LXC, pada *Docker* kedua mesin telah memiliki *container* sehingga yang dipindahkan adalah kondisi dari mesin tersebut ketika berjalan (*checkpoint). Checkpoint* tersebut dikirimkan dan diterapkan kembali pada server yang dituju, migrasi dilakukan dengan menggunakan *shellscript* pada **Gambar 3.7.** kemudian hasilnya dapat dilihat pada **Gambar 3.8**.

```
docker checkpoint create --checkpoint-dir /home/docker1/Checkpoints tes checkpoint1

docker container stop tes

sshpass -p ' ' scp -r /home/docker1/Checkpoints/checkpoint1/ docker2@192.168.169.177:/home/docker2

sshpass -p ' ' ssh docker2@192.168.169.177 'echo " " | sudo -S docker start --checkpoint-dir /home/docker2 --checkpoint checkpoint1 tes'
```

Gambar 3.7 *Shellscript* pada Docker

Fungsi setiap *command* pada **Gambar 3.7** adalah sebagai berikut:

1. *Docker checkpoint create –checkpoint-dir directory nama_container nama_checkpoint* untuk membuat *checkpoint* pada kondisi *container* saat ini.

2. *Docker container stop nama_container* untuk memberhentikan *container.*
3. *SSHPASS –p ‘ ‘ scp -r /directory asal/ username @ ip:/directory tujuan/* untuk memindahkan *checkpoint container*.
4. *SSHPASS -p ‘ ‘ ssh username @ ip.address ‘echo “ “ | sudo -s docker start –checkpoint-dir /lokasi checkpoint/ --checkpoint nama_checkpoint nama_container* untuk menjalankan *container* pada server 2 dengan *checkpoint*.

Gambar 3.8 *Migrasi* pada Docker

3.1.5 CRIU melakukan injeksi *userspace* ke *container*

CRIU melakukan injeksi *userpace* dengan menggunakan *parasite code* untuk dapat mengakses OS. Hal ini dilakukan untuk mendapatkan *state* dari *container* yang kemudian dibuat menjadi *checkpoint.*

3.1.6 CRIU membuat state file dari container saat ini

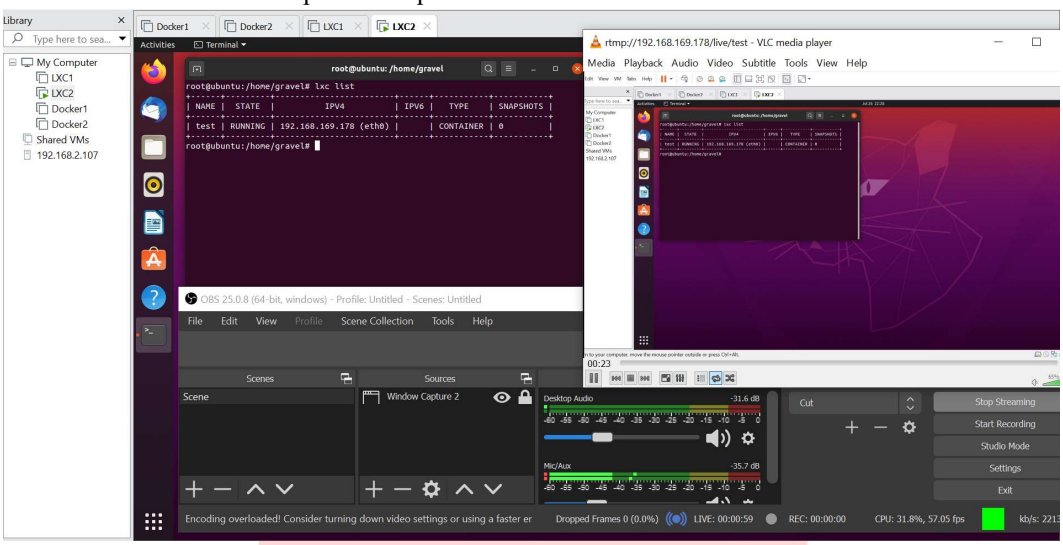
Memanfaatkan akses *userspace,* CRIU membuat *checkpoint* untuk dimigrasikan. *Checkpoint* berupa *images* yang berisi informasi dan data yang dapat dimigrasikan.

3.1.7 Checkpoint dimigrasikan ke server 2 dan direstore

Checkpoint dipindahkan ke *server 2* kemudian di*restore* kembali bersamaan dengan menjalankan *container server* 2.

3.1.8 Menjalankan *Container* pada server 2 dengan state dari container 1

Live streaming yang semula diakses pada server 1 sekarang dapat diakses pada server 2. Hal ini dapat dilihat pada **Gambar 3.9**.



Gambar 3.9 Menjalankan kembali sesi di *server 2*

4. Hasil Dan Analisis

Pada bab ini akan dibahas mengenai hasil pengujian penelitian dengan membandingkan LXC dan Docker. Adapun kriteria yang diujikan adalah sebagai berikut

1. Pengujian *Quality of Service* (QoS) berupa *throughput, delay, jitter, dan packet loss* pada *container* LXC dan Docker saat dilakukan *live stream.* Pengujian ini dilakukan sebanyak tiga puluh kali dalam kurun waktu satu menit. Pengambilan data dilakukan dengan menggunakan tools *wireshark.*
2. Mengukur waktu *downtime* ketika dilakukan migrasi *container.* Migrasi dilakukan dengan dua scenario, pertama saat dilakukan *live stream* dan kedua saat *running database*. Pengujian dilakukan pada server LXC dan Docker dengan jumlah pengujian sebanyak tiga puluh kali dalam kurun waktu satu menit.
3. Mengukur utilisasi CPU dan Memory pada *container* LXC dan docker..Pengujian dilakukan dalam kurun waktu satu jam dengan periode pencatatan setiap lima menit menggunakan skenario berikut:
   a. Pengujian CPU dan Memory saat dilakukan
      1) Running live streaming
      2) Running *database*
      3) Server idle
   b. Pengujian CPU dan Memory saat dilakukan
      1) Running live streaming saat dilakukan migrasi
      2) Running database saat dilakukan migrasi

4.1 Hasil Pengujian *Quality of Service*

Rata rata *QoS* dengan kriteria *Throughput*, *jitter, delay, dan packet loss* untuk kedua *container* memenuhi indeks yang cukup *reliable* dapat dilihat pada **Tabel 4.1** dan **Tabel 4.2**. Perbedaan *QoS*

pada *docker* dan LXC disebabkan oleh beberapa faktor diantaranya koneksi yang tidak selalu stabil saat dilakukan *live stream*, perubahan ukuran paket yang mempengaruhi *throughput,* variasi *waktu* paket dikirim, ukuran paket dan isi paket untuk setiap sesi serta lama waktu *uptime yang* mempengaruhi *packet loss*.

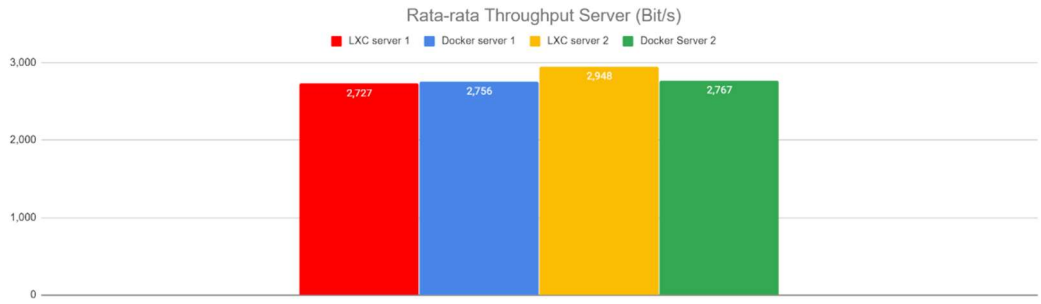
Tabel 4.1 Hasil rata-rata QoS Server 1

Hasil rata-rata QoS Server 1		
Parameter/Server	LXC	Docker
Throghput (Bit/s)	2727	2756
Delay (ms)	4.819	4.787
Jitter (ms)	0.1551173	0.0000223
Packet loss (%)	0.03	0.04

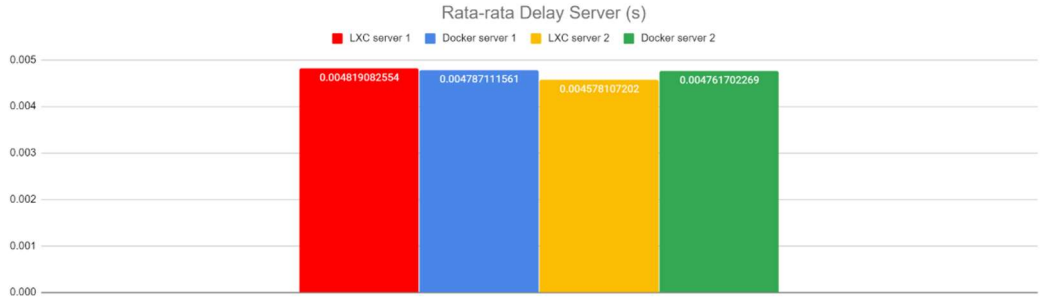
Tabel 4.2 Hasil rata-rata QoS pada server 2 setelah migrasi

Hasil rata-rata QoS Server 2		
Parameter/Server	LXC	Docker
Throghput (Bit/s)	2,948k	2,767k
Delay (ms)	4.578	0. 4.762
Jitter (ms)	0.0000114	0.0000625
Packet loss (%)	0.04	0.07

Grafik *Throughput* pada **Gambar 4.1** memperlihatkan hasil yang *reliable, throughput* pada masing-masing server tidak memiliki perbedaan yang terlalu signifikan, begitu pula hasil rata-rata *delay* yang ditunjukkan oleh grafik *delay* pada **Gambar 4.2** memiliki selisih nilai yang tidak terlalu tinggi.

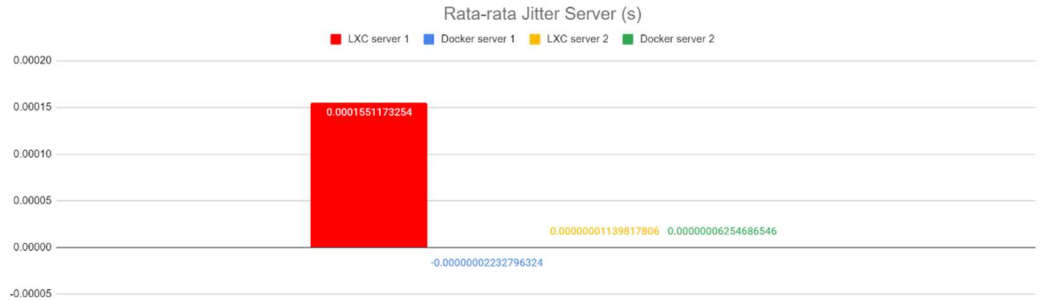


Gambar 4.1 Grafik perbandingan *throughput*



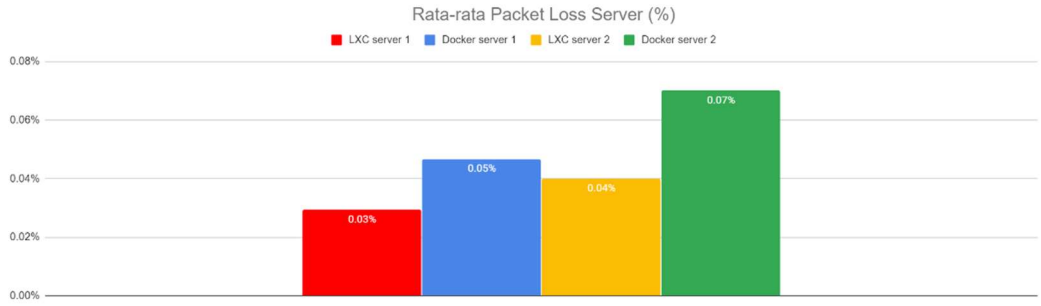
Gambar 4.2 Grafik perbandingan *delay*

Berdasarkan **Gambar 4.3**, keempat *server* menghasilkan kualitas *jitter* yang baik, dengan rata-rata jitter yang dihasilkan oleh LXC *server* 1 lebih tinggi dari server yang lain. Namun, LXC *server 2* yang merupakan hasil migrasi dengan konfigurasi yang sama dengan LXC *server* 1 menghasilkan rata-rata *jitter* dibawah LXC *server* 1. Hasil rata-rata jitter LXC *server* 2 tidak jauh berbeda dengan rata-rata jitter yang dihasilkan server docker. Perbedaan nilai rata-rata tersebut dapat berasal dari *delay* koneksi *jaringan,* RAM dan lamanya respon OBS dalam beberapa percobaan.



Gambar 4.3 Grafik perbandingan *jitter*

Pada grafik di ***Gambar 4.4***, Rata-rata *packet loss* yang dihasilkan server LXC jauh lebih baik apabila dibandingkan dengan rata-rata *packet loss* server *docker.* Namun, packet loss yang dihasilkan kedua *server* tersebut masih dalam kategori baik.. Faktor yang menyebabkan *packet loss* pada *docker* sendiri berasal dari bentuk migrasinya, pada *docker checkpoint* migrasi dilakukan dengan hanya mengambil *state container* kemudian dilakukan penyesuaian agar *container* dapat berjalan sesuai dengan *state* yang disimpan oleh *checkpoint,* serta beberapa faktor lainnya seperti *koneksi jaringan,* RAM, dan lamanya OBS untuk melanjutkan *livesteam* tersebut.



Gambar 4.4 Grafik perbandingan *packet loss*

4.2 Downtime

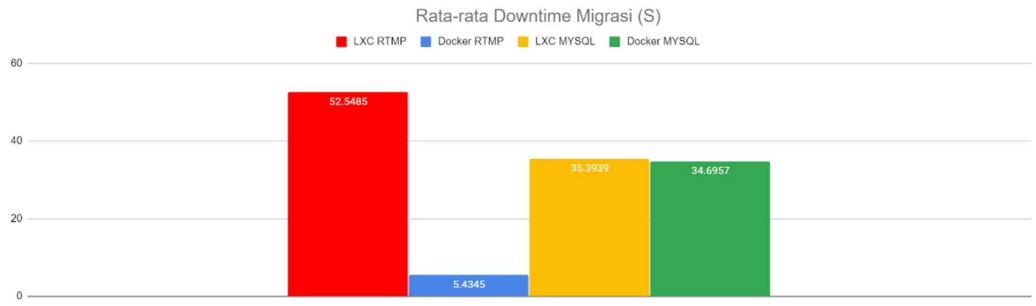
Berdasarkan **Tabel 4.3** dan **Gambar 4.5** terlihat perbedaan *downtime* yang signifikan antara LXC dan *docker.* Untuk pengujian RTMP, LXC memiliki rata-rata *downtime* selama 53 detik dan *docker* memiliki rata-rata *downtime* selama 6 detik. Sedangkan untuk pengujian MySQL

downtime yang dihasilkan LXC selama 35 detik dan *downtime* yang dihasilkan docker selama 21 detik.

Faktor yang dapat mempengaruhi lama downtime adalah penggunaan *Random Access Memory* (RAM) dan Teknik virtualisasinya. Pada server *docker* penggunaan RAM lebih kecil dibandingkan server LXC sehingga *downtime* yang dihasilkan lebih cepat. Kemudian teknik virtualisasi dan migrasi pada *docker* yang menggunakan *file* dengan ukuran yang lebih kecil berdampak pada waktu pengiriman yang lebih cepat.

Tabel 4.3 Hasil rata-rata *Downtime* pada *server*

Hasil rata-rata Downtime pada Server (s)				
Parameter/Server	LXC RTMP	Docker RTMP	LXC MYSQL	Docker MYSQL
Downtime (s)	52.5485	5.4345	35.3939	36.6957
Size (MB)	601.3695	2.0264	720.0751	543.5147



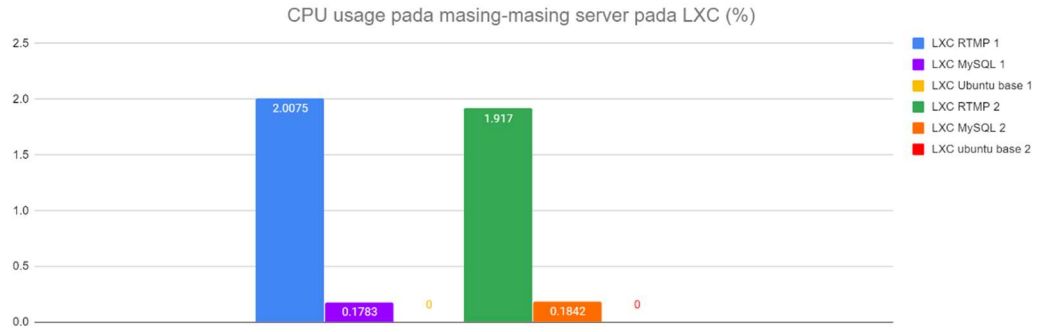
Gambar 4.5 Grafik perbandingan *downtime*

4.3 CPU usage dan Memory Usage

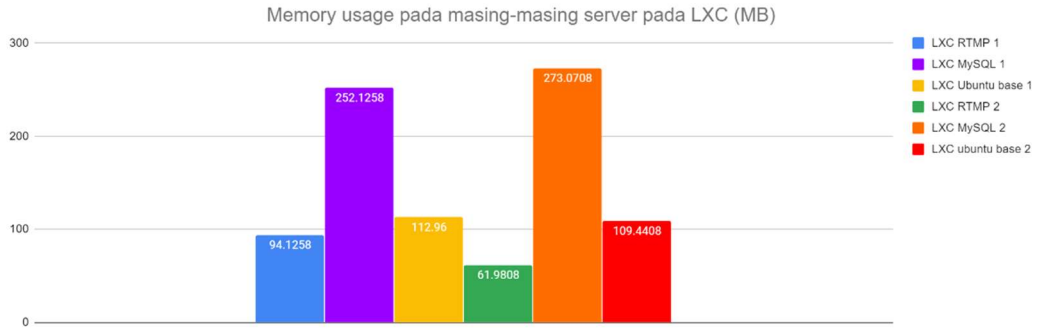
Pengujian utilisasi CPU dan *Memory* dilakukan pada masing-masing *container* dalam kurun waktu satu jam dan dengan periode pencatatan setiap lima menit. Utilisasi CPU dan Memory tersebut dipengaruhi oleh bentuk virtualisasi dan kapasitas *hardware*. Pengujian pertama dilakukan dengan skenario *running live streaming, running database* dan *server idle.* Hasil pengujian untuk utilisasi CPU dan Memory bagi LXC dapat dilihat pada **Gambar 4.6**, **Gambar 4.7**, serta pada **Tabel 4.4**. Sedangkan utilisasi bagi docker diperlihatkan pada **Gambr 4.8**, **Gambar 4.9**, dan pada **Tabel 4.5**.

Tabel 4.4 Hasil rata-rata CPU *usage* dan *memory usage* pada LXC.

Hasil rata-rata CPU usage dan Memory Usage Container LXC						
Server	LXC RTMP 1	LXC MySQL 1	LXC base 1	LXC RTMP 2	LXC MySQL 2	LXC base 2
CPU (%)	2.0075	0.1783	0	1.9168	0.1847	0
Memory (MB)	94.1258	252.1258	112.96	61.9808	273.0708	109.4408



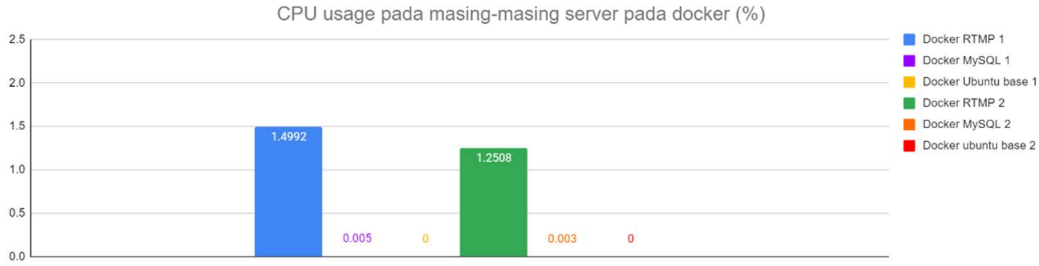
Gambar 4.6 Grafik perbandingan CPU *usage* pada LXC



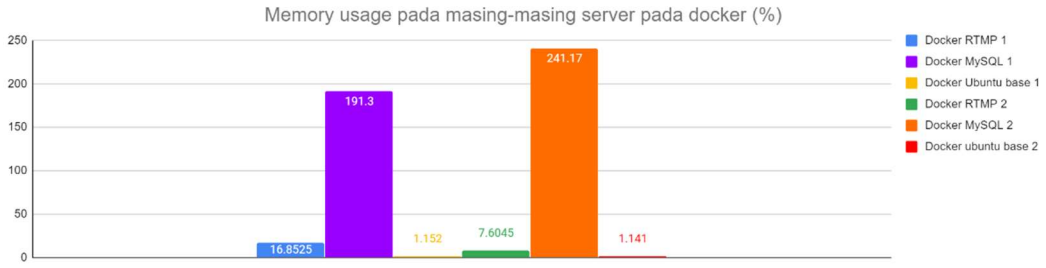
Gambar 4.7 Grafik perbandingan *Memory usage* pada LXC

Tabel 4.5 Hasil rata-rata CPU *usage* dan *memory usage* pada *docker*.

Hasil rata-rata CPU usage dan Memory Usage Container						
Server	Docker RTMP 1	Docker MySQL 1	Docker base 1	Docker RTMP 2	Docker MySQL 2	Docker base 2
CPU (%)	1.4992	0.005	0	1.2508	0.003	0
Memory (MB)	16.8525	191.3	1.152	7.6045	241.17	1.141



Gambar 4.8 Grafik perbandingan CPU *usage* pada *docker*

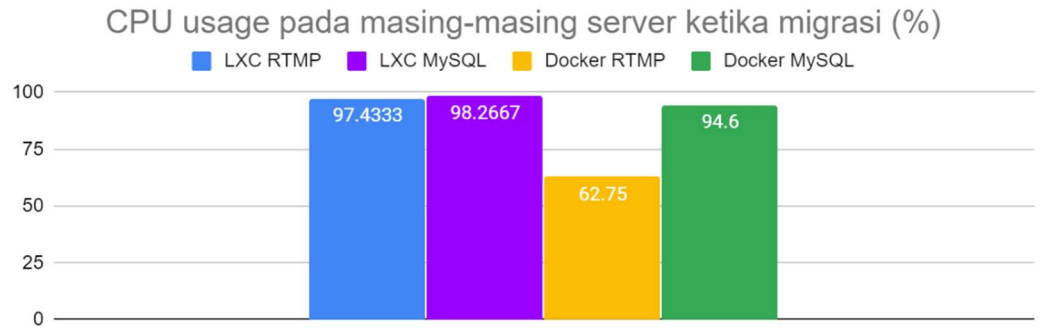


Gambar 4.9 Grafik perbandingan *Memory usage* pada *docker*

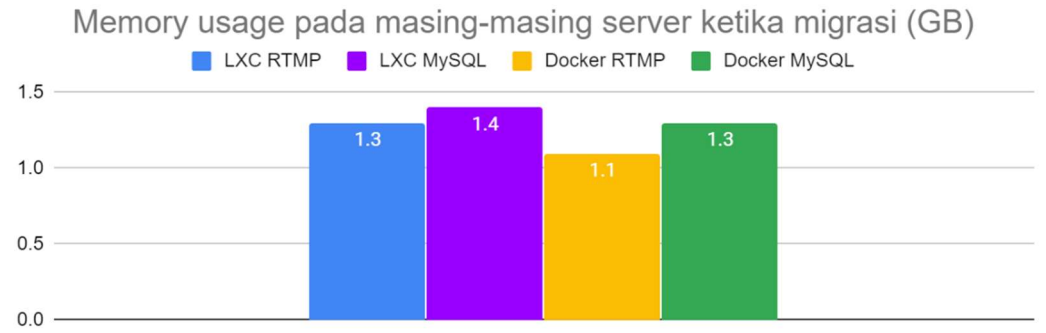
Sedangkan pengujian kedua dilakukan ketika migrasi berlangsung untuk kedua *container*. Hasil pengujian dapat dilihat pada **Gambar 4.10, Gambar 4.11,** dan **Tabel 4.6.**

Tabel 4.6 Hasil rata-rata CPU *usage* dan *memory usage* saat migrasi.

Hasil rata-rata CPU usage dan Memory Usage ketika migrasi				
Server	LXC RTMP	LXC MySQL	Docker RTMP	Docker MySQL
CPU (%)	97.4333	98.2667	62.75	94.6
Memory (GB)	1.3	1.4	1.1	1.3



Gambar 4.10 Grafik perbandingan CPU *usage* pada saat migrasi



Gambar 4.11 Grafik perbandingan *Memory usage* pada saat migrasi

Berdasarkan grafik diatas dapat dilihat bahwa utilisasi CPU dan *memory* tidak terlalu jauh berbeda untuk kedua jenis *container* tersebut. Namun dapat dilihat bahwa pada skenario

server *idle* maupun saat dilakukan migrasi, *docker* memiliki utilisasi CPU dan *memory* yang lebih baik. Hal ini dikarenakan bentuk virtualisasi *docker* yang lebih ringan dibandingkan dengan LXC.

5. Kesimpulan dam saran

Setelah dilakukan pengujian dan analisis terhadap perbandingan migrasi layanan, berikut ini merupakan kesimpulan dan saran yang dihasilkan berdasarkan hasil pengujian dan analisis sistem diantaranya:

5.1 Kesimpulan

1. Sistem migrasi layanan untuk kedua *docker container* dan *Linux Container* (LXC) berjalan dengan baik.
2. Perbandingan *Quality of Service* (QoS) dari kedua *container* tidak jauh berbeda.
3. Namun, *Docker container* memiliki *downtime* yang lebih baik daripada LXC dikarenakan bentuk virtualisasi yang lebih ringan maupun penggunaan *CPU usage* dan *memory usage*.
4. *Downtime* migrasi pada docker di antara layanan RTMP dan MySQL dipengaruhi oleh *docker volume* yang digunakan oleh *server* MySQL

5.2 Saran

1. Disarankan untuk menggunakan hardware yang lebih kuat.
2. Kedua *container* telah memiliki pengembangan untuk digunakan yang memiliki kemampuan migrasi lebih canggih seperti *Podman* dan *Proxmox VE*.
3. Menggunakan layanan yang memiliki beban lebih tinggi seperti *object-detection*.
4. Menambah parameter pengujian, terutama dalam layanan.

Daftar Pustaka:

[1] J. F. Kurose and K. W. Ross, *COMPUTER NETWORKING A Top-Down Approach*. 2013.

[2] S. Oh and J. Kim, "Stateful Container Migration employing Checkpoint-based Restoration for Orchestrated Container Clusters," *2018 Int. Conf. Inf. Commun. Technol. Converg.*, pp. 25–30, 2018.

[3] A. Machen, S. Wang, K. K. Leung, B. J. Ko, T. Salonidis, and D. C. Aug, "Live Service Migration in Mobile Edge Clouds," *IEEE Wirel. Commun.*, pp. 1–8, 2018.

[4] P. D.Patel, M. Karamta, M. D. Bhavsar, and M. B. Potdar, "Live Virtual Machine Migration Techniques in Cloud Computing: A Survey," *Int. J. Comput. Appl.*, vol. 86, no. 16, pp. 18–21, 2014, doi: 10.5120/15070-3453.

[5] S. Vaddi, C. Kumar, and A. Jannesari, "Efficient Object Detection Model for Real-Time UAV Applications," 2019.

[6] N. An, S. Yoon, T. Ha, Y. Kim, and H. Lim, "Seamless virtualized controller migration for drone applications," *IEEE Internet Comput.*, vol. 23, no. 2, pp. 51–58, 2019, doi: 10.1109/MIC.2018.2884670.

[7] S. Mahmoud, N. Mohamed, and J. Al-Jaroodi, "Integrating UAVs into the Cloud Using the Concept of the Web of Things," *J. Robot.*, vol. 2015, 2015, doi: 10.1155/2015/631420.

[8] Y. Qiu, C. Lung, and S. Ajila, "LXC Container Migration in Cloudlets under Multipath TCP," *2017 IEEE 41st Annu. Comput. Softw. Appl. Conf. LXC*, pp. 31–36, 2017, doi: 10.1109/COMPSAC.2017.163.

[9] B. David, "Containers and Cloud : From LXC to Docker to Kubernetes," *IEEE Cloud Comput.*, pp. 81–84, 2014.

[10] T. Banerjee, "Understanding the key differences between LXC and Docker," 2014. [Online]. Available: https://archives.flockport.com/lxc-vs-docker/#:~:text=LXC is a container technology,cannot be treated as such.

[11] Docker, *The Definitive Guide to Container Platforms*, no. August. 2018.

[12] P. Guan, J. Li, and A. Yan, "Design and implementation of streaming media system based on RTP protocol family," *J. Adv. Oxid. Technol.*, vol. 21, no. 2, pp. 192–196, 2018, doi: 10.26802/jaots.2018.09047.

[13] C. Győrödi, R. Gyorodi, G. Pecherle, and A. Olah, "A Comparative Study: MongoDB vs. MySQL Energetical sustainability of a local community using air flows View project Convergence of university practical training for integration with success in the labor market View project," no. June, 2015, doi: 10.13140/RG.2.1.1226.7685.