IMPLEMENTASI *CONTAINER LIVE MIGRATION* ANTAR-*CLOUD PROVIDER* MENGGUNAKAN PODMAN DAN CRIU

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Muhammad Abdul Aziz
NIM: 135150207111038

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2020

PENGESAHAN

IMPLEMENTASI *CONTAINER LIVE MIGRATION* ANTAR-*CLOUD PROVIDER*
MENGGUNAKAN PODMAN DAN CRIU

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh:
Muhammad Abdul Aziz
NIM: 135150207111038

Skripsi ini telah diuji dan dinyatakan lulus pada
28 Juli 2020
Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I	Dosen Pembimbing 2
Adhitya Bhawiyuga, S.Kom., M.Sc.	Fariz Andri Bakhtiar, S.T., M.Kom.
NIP: 19890720 201803 1 002	NIK: 2017098403141001

Mengetahui
Ketua Jurusan Teknik Informatika

Achmad Basuki, S.T., M.MG., Ph.D.
NIP: 19741118 200312 1 002

PERNYATAAN ORISINALITAS

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah skripsi ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis disitasi dalam naskah ini dan disebutkan dalam daftar referensi.

Apabila ternyata di dalam naskah skripsi ini dapat dibuktikan terdapat unsur-unsur plagiasi, saya bersedia skripsi ini digugurkan dan gelar akademik yang telah saya peroleh (sarjana) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 01 Agustus 2020

Muhammad Abdul Aziz

NIM: 135150207111038

ABSTRAK

Muhammad Abdul Aziz, Implementasi *Container Live Migration* Antar-*Cloud Provider* Menggunakan Podman dan CRIU

Pembimbing: Adhitya Bhawiyuga, S.Kom., M.Sc. dan Fariz Andri Bakhtiar, S.T., M.Kom.

Cloud computing merupakan teknologi yang penting dan cukup banyak digunakan saat ini. Beberapa perusahaan yang menyediakan layanan *cloud computing* saat ini antara lain Google, Microsoft, IBM, dan Amazon. Salah satu layanan *cloud computing* yang populer adalah virtualisasi. Virtualisasi memungkinkan sebuah mesin tunggal menjalankan berbagai peran dari suatu layanan yang berjalan. Dalam penerapannya, virtualisasi pada *cloud computing* tidak dapat terlepas dari proses *live migration*. *Live migration* mesin virtual memerlukan keseluruhan *state* dari mesin virtual sumber untuk dipindahkan ke mesin virtual tujuan. Disisi lain, terdapat teknik *live migration* yang bernama *container live migration* yang hanya membutuhkan *state* dari layanan yang akan dipindahkan. Penelitian ini menerapkan teknik *container live migration* untuk memindahkan layanan dari sebuah mesin virtual ke mesin virtual lainnya, tanpa harus memindahkan keseluruhan *state* dari mesin virtual sumber ke mesin virtual tujuan, menggunakan Podman dan CRIU. Dengan Podman, sebuah layanan dapat berjalan sebagai proses terisolasi, independen terhadap lingkungan sekitarnya. Dengan CRIU, layanan dapat dihentikan dan dipindahkan dari satu mesin ke mesin lainnya. Hasil pengujian fungsional menunjukkan bahwa metode *live migration* yang diusulkan mampu memindahkan layanan antar-*cloud provider* yang berbeda. Hasil pengujian implementasi *container live migration* antar *cloud provider* menggunakan Podman dan CRIU menunjukkan rata-rata waktu *downtime* sebesar 34,618 detik, dan rata-rata waktu *migration time* sebesar 71,627 detik.

Kata kunci: *cloud, cloud computing*, *container*, *live migration*, Podman, CRIU

ABSTRACT

Muhammad Abdul Aziz, Implementasi *Container Live Migration* Antar-*Cloud Provider* Menggunakan Podman dan CRIU

Supervisors: Adhitya Bhawiyuga, S.Kom., M.Sc. and Fariz Andri Bakhtiar, S.T., M.Kom.

Cloud computing is an important technology and is quite widely used today. Some companies that provide cloud computing services now are Google, Microsoft, IBM and Amazon. One of the popular cloud computing services is virtualization. Virtualization allows a single machine to perform multiple roles of a running service. In the application of virtualization in cloud computing, live migration is a very important process. Live migration of a virtual machine requires the entire state of the source virtual machine to be moved to the destination virtual machine. On the other hand, there is a live migration technique called container live migration which only requires the state of the service to be moved. This research applies container live migration techniques to move services from one virtual machine to another, without having to move the entire state from the source virtual machine to the destination virtual machine, using Podman and CRIU. With Podman, a service can run as an isolated process, independent of its surroundings. With CRIU, services can be stopped and transferred from one machine to another. The results of functional testing show that the proposed live migration method is capable of moving services between different cloud providers. The results of testing the implementation of container live migration between cloud providers using Podman and CRIU show an average downtime of 34.618 seconds, and an average migration time of 71.627 seconds.

Keywords: *cloud, cloud computing*, *container*, *live migration*, Podman, CRIU

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR LAMPIRAN

BAB 1 PENDAHULUAN

1.1 Latar Belakang

Cloud computing memiliki pengaruh yang cukup besar terhadap industri teknologi informasi, akademisi, dan pengguna individu karena kemudahan dalam penggunaannya, akses yang sesuai permintaan ke sumber daya jaringan, upaya manajemen yang minim, dan biaya penggunaan yang lebih rendah. Beberapa perusahaan teknologi informasi melihat *cloud computing* sebagai peluang dan mulai membangun infrastruktur *cloud computing* masing-masing. Saat ini, beberapa perusahaan teknologi informasi yang telah menyediakan layanan *cloud computing* antara lain, Google, Microsoft, IBM, dan Amazon. *Cloud computing* memiliki tiga karakteristik dasar, yakni manajemen sumber daya yang elastis dan cepat, akses jaringan luas, serta pelayanan mandiri yang sesuai permintaan dan terukur. Terdapat tiga model layanan pada *cloud computing*, yaitu *Software as a Service* (SaaS), *Platform as a Service* (PaaS), dan *Infrastructure as a Service* (IaaS) (Birje, et al., 2017).

Salah satu contoh model layanan IaaS pada *cloud computing* adalah virtualisasi. Virtualisasi merupakan *backbone* dari *cloud computing*. Dengan virtualisasi, perangkat keras dapat diemulasikan di atas sistem operasi utama, sehingga sebuah mesin tunggal dapat menjalankan beberapa sistem operasi sekaligus. Virtualisasi memiliki beberapa keunggulan, yaitu penggunaan sumber daya yang efisien, peningkatan keamanan, portabilitas, manajemen yang lebih mudah, peningkatan fleksibilitas, kemampuan isolasi, dan penyebaran yang cepat (Rashid & Chatuverdi, 2019). Dalam penerapan virtualisasi pada *cloud computing*, terdapat kondisi yang menyebabkan pemindahan sumber daya antar-mesin virtual harus dilakukan. Proses pemindahan ini disebut dengan migrasi.

Migrasi memiliki makna memindahkan atau menyalin seluruh layanan dan data dari server sumber ke server tujuan, kemudian melanjutkan prosesnya kembali pada server tujuan. Beberapa alasan yang menyebabkan proses migrasi dilakukan, antara lain persiapan sebelum melakukan *load balance*, pengalokasian sumber daya komputasi secara efektif, pertimbangan masalah keamanan, perawatan perangkat keras server, konsolidasi server, dan perpindahan antar-zona yang tersedia (Nadgowda, et al., 2017). Terdapat dua jenis migrasi pada mesin virtual, yaitu *cold migration* dan *live migration*. Pada *cold migration*, proses migrasi dilakukan setelah mesin virtual dihentikan sepenuhnya, yang mengakibatkan pengguna tidak dapat mengakses sumber daya pada mesin virtual hingga proses migrasi selesai. Sedangkan pada *live migration*, proses migrasi dilakukan saat mesin virtual masih berjalan, sehingga pengguna masih dapat mengakses sumber daya pada mesin virtual dengan gangguan yang minim (Tandel, et al., 2019).

Live migration memerlukan keseluruhan *state* dari mesin virtual sumber untuk dipindahkan ke mesin virtual tujuan, yang terdiri dari penyimpanan volatil,

penyimpanan permanen, *state* internal dari *Central Processing Unit* (CPU) virtual, dan perangkat yang terhubung (Mahfouz, et al., 2017). Selain *live migration* mesin virtual, terdapat teknik *live migration* lain bernama *container live migration*. *Container live migration* tidak memerlukan keseluruhan *state* dari mesin virtual sumber, melainkan hanya *state* dari layanan yang akan akan dipindahkan.

Container live migration dapat diterapkan menggunakan Docker dan CRIU (Ma, et al., 2019). Docker merupakan sebuah aplikasi *open source* yang menyediakan cara sistematis untuk mengotomatisasikan penyebaran aplikasi linux secara cepat di dalam *container* portabel. Pada dasarnya, Docker mengembangkan Linux *Container* (LXC) dengan menambahkan *Application Programming Interface* (API) pada level kernel dan aplikasi, yang bersama-sama menjalankan proses terisolasi (Bernstein, 2014). Selain Docker, terdapat alternatif lain bernama Podman. Podman merupakan *daemonless container engine* yang digunakan untuk mengembangkan, mengatur, dan menjalankan *Open Container Initiative* (OCI) *container* dan *container image* pada sistem operasi linux. Podman dapat dijalankan dengan atau tanpa otoritas pengguna 'root'. Jika dibandingkan dengan Docker, Podman memiliki dukungan yang lebih baik terhadap CRIU (Podman, 2019). CRIU merupakan sebuah perangkat lunak yang mampu membekukan proses pada sebuah aplikasi berjalan (sebagian atau seutuhnya) dan menyimpan *state* proses aplikasi tersebut ke dalam media penyimpanan. *State* yang telah disimpan dapat digunakan untuk memulihkan dan menjalankan kembali proses dari aplikasi yang dibekukan sebelumnya dalam keadaan sama persis seperti saat pembekuan proses aplikasi dilakukan (CRIU, 2019).

Nadgowda, et al. menggabungkan Docker dan CRIU dengan kemampuan *data federation* yakni *union mount* untuk meminimalkan *downtime* saat proses migrasi berlangsung. Hasil pengujian menunjukkan bahwa *downtime* yang terjadi saat proses migrasi berlangsung sebesar 2-3 detik dan *overhead* 1-3% untuk beban kerja *read/write* secara umum (Nadgowda, et al., 2017). Ma, et al. mengusulkan sebuah teknik *live migration* untuk layanan *edge computing* menggunakan arsitektur penyimpanan berlapis milik Docker. Dari evaluasi yang dilakukan, didapatkan pengurangan waktu *handoff* sebesar 56-80% dibandingkan dengan *state-of-the-art handoff* mesin virtual untuk platform *edge computing* (Ma, et al., 2019). Fan, et al. mengusulkan sebuah metode dan algoritme peningkatan kinerja migrasi secara umum. Hasil pengujian menunjukkan bahwa metode yang diusulkan mampu meningkatkan pemanfaatan dan keseimbangan kinerja sumber daya pada mesin fisik (Fan, et al., 2018). Govindaraj & Artemenko mengusulkan metode *live migration* baru bernama *redudancy migration* dengan memodifikasi *Linux Container Daemon* (LXD). Hasil pengujian menunjukkan bahwa *redudancy migration* mampu mengurangi *downtime* hingga skala 1.8 dibandingkan dengan LXD *live migration* (Govindaraj & Artemenko, 2018). Mirkin, et al. mengimplementasikan *live migration* pada OpenVZ. Proses *checkpoint* dan *restart* diimplementasikan sebagai *loadable kernel module* (Mirkin, et al., 2008).

1.2 Identifikasi Masalah

Berdasarkan pembahasan pada subbab 1.1 Latar belakang, dapat diketahui bahwa untuk melakukan *live migration* pada mesin virtual, membutuhkan keseluruhan *state* dari mesin virtual sumber untuk dipindahkan ke mesin virtual tujuan (Mahfouz, et al., 2017). Selain *live migration* mesin virtual, terdapat teknik *live migration* lain bernama *container live migration* yang hanya membutuhkan *state* dari layanan yang akan dipindahkan. Berlandaskan dari dua pernyataan sebelumnya, penelitian ini menerapkan teknik *container live migration* untuk memindahkan layanan dari sebuah mesin virtual ke mesin virtual lainnya, tanpa harus memindahkan keseluruhan *state* dari mesin virtual sumber ke mesin virtual tujuan, menggunakan Podman dan CRIU. Implementasi dan pengujian dilakukan pada dua *cloud provider* yang berbeda, yakni *Google Cloud Platform* (GCP) dan *Amazon Web Services* (AWS). Parameter pengujian yang digunakan adalah *downtime* dan *migration time*, karena kedua parameter tersebut merupakan satuan pengukuran yang sangat penting dalam proses migrasi (Govindaraj & Artemenko, 2018).

1.3 Rumusan Masalah

Rumusan masalah yang terdapat pada penelitian ini adalah:

1. Bagaimanakah implementasi *container live migration* antar-*cloud provider* menggunakan Podman dan CRIU?
2. Bagaimanakah hasil pengujian kinerja implementasi *container live migration* antar-*cloud provider* menggunakan Podman dan CRIU?

1.4 Tujuan

Tujuan yang ingin dicapai dalam penelitian ini antara lain:

1. Mengimplementasikan *container live migration* yang mampu memindahkan layanan antar-*cloud provider*, tanpa harus memindahkan keseluruhan *state* dari mesin virtual sumber ke mesin virtual tujuan, menggunakan Podman dan CRIU.
2. Mengetahui kinerja *container live migration* yang diimplementasikan dari segi *downtime* dan *migration time*.

1.5 Manfaat

Manfaat yang diperoleh dari penelitian ini adalah:

1. Menjaga availabilitas layanan dari sebuah server saat terjadinya keadaan di mana migrasi harus dilakukan, sebagai contoh saat perawatan atau peningkatan perangkat keras server.
2. Membantu pengguna layanan *cloud provider* agar lebih fleksibel dalam mendistribusikan layanannya, baik dalam satu atau antar-*cloud provider*.

1.6 Batasan Masalah

Agar fokus penelitian ini lebih jelas dan terarah, maka diberikan batasan masalah sebagai berikut:

1. Layanan *cloud* yang digunakan adalah *Google Cloud Platform* (GCP) milik Google dan *Amazon Web Services* (AWS) milik Amazon.
2. Akun AWS yang digunakan adalah akun AWS *Educate* yang memiliki batasan fitur yang dapat digunakan.
3. Layanan yang dibangun pada *cloud* adalah layanan web *video streaming* dan MySQL.
4. Parameter pengujian yang digunakan adalah *downtime* dan *migration time*.

1.7 Sistematika Penulisan

Sistematika penulisan pada dokumen penelitian ini terdiri dari enam bab dengan keterangan setiap bab dijelaskan sebagai berikut:

BAB 1 PENDAHULUAN

Bab ini berisi deskripsi umum penelitian yang terdiri dari latar belakang, identifikasi masalah, rumusan masalah, tujuan, manfaat, batasan masalah, serta sistematika penulisan dari implementasi *container live migration* antar-*cloud provider* menggunakan Podman dan CRIU.

BAB 2 LANDASAN KEPUSTAKAAN

Bab ini memuat penjelasan detail dari berbagai konsep, teori, dan referensi yang digunakan sebagai acuan penyelesaian implementasi *container live migration* antar-*cloud provider* menggunakan Podman dan CRIU.

BAB 3 METODOLOGI PENELITIAN

Bab ini menjabarkan tentang metodologi penelitian yang digunakan dalam mengimplementasikan *container live migration* antar-*cloud provider* menggunakan Podman dan CRIU yang terdiri dari kerangka penelitian, perancangan sistem, dan metode evaluasi.

BAB 4 IMPLEMENTASI

Bab ini memuat penjelasan keseluruhan proses implementasi *container live migration* antar-*cloud provider* menggunakan Podman dan CRIU yang berdasar pada perancangan sistem dan lingkungan implementasi yang telah dibuat.

BAB 5 PENGUJIAN DAN PEMBAHASAN

Bab ini menjabarkan proses dan data hasil pengujian yang diperoleh dari implementasi *container live migration* antar-*cloud provider* menggunakan Podman dan CRIU, beserta pembahasannya.

BAB 6 PENUTUP

Bab ini memuat kesimpulan yang diperoleh dari proses penelitian implementasi *container live migration* antar-*cloud provider* menggunakan Podman dan CRIU. Bab ini juga berisi saran yang dapat digunakan pada penelitian selanjutnya.

BAB 2 LANDASAN KEPUSTAKAAN

Bab ini berisikan uraian tinjauan pustaka beserta teori-teori pendukung penelitian implementasi *container live migration* antar-*cloud provider* menggunakan Podman dan CRIU. Tinjauan pustaka berisi pembahasan penelitian-penelitian sebelumnya yang relevan terhadap penelitian yang sedang dilakukan, serta memberikan gambaran umum persamaan dan perbedaannya. Dasar teori yang digunakan pada penelitian ini, antara lain *cloud computing*, virtualisasi, *container*, *live migration*, MySQL, dan CRIU.

2.1 Tinjauan Pustaka

Penelitian Nadgowda, et al. dengan judul "Voyager: *Complete Container State Migration*" menggabungkan Docker dan migrasi memori berbasis CRIU dengan kemampuan *data federation* yakni *union mount* untuk meminimalkan *downtime* saat proses migrasi berlangsung. Dengan tampilan *union* dari data antara *node* sumber dan *node* tujuan, *container* Voyager dapat melanjutkan operasi secara instan pada *node* tujuan sambil melakukan pemindahan *state* dari *disk* secara *lazy* (atau disebut juga sebagai *post-copy memory migration*) yang berjalan di latar belakang. Pengujian dilakukan menggunakan dua mesin virtual Ubuntu 14.04 *Long Term Support* (LTS) dengan 4 CPU virtual, 4 GB memori, 25 GB *disk*, dan *filesystem* ext4. Kedua mesin virtual terletak pada *datacenter* yang sama dengan rata-rata *bandwidth* 2.5 Gbps. Kinerja layanan MySQL diuji menggunakan aplikasi Yahoo! *Cloud Serving Benchmark* (YCSB). Hasil pengujian menunjukkan bahwa *downtime* yang terjadi saat proses migrasi berlangsung sebesar 2-3 detik dan *overhead* 1-3% untuk beban kerja *read/write* secara umum (Nadgowda, et al., 2017). Pada penelitian implementasi *container live migration* antar-*cloud provider* menggunakan Podman dan CRIU, pengujian dilakukan menggunakan dua mesin virtual yang terdapat pada *cloud provider* yang berbeda, yakni GCP dan AWS. Layanan yang dimigrasikan adalah layanan web *video streaming* dan MySQL.

Ma, et al. telah melakukan penelitian yang berjudul "*Efficient Live Migration of Edge Services Leveraging Container Layered Storage*". Mereka mengusulkan sebuah teknik *live migration* untuk meminimalkan waktu *handoff* pada layanan *edge computing* menggunakan arsitektur penyimpanan berlapis milik Docker, yang hanya memperbolehkan lapisan paling atas untuk diubah sepanjang masa hidup suatu *container*. Karena hal ini, lapisan di bawah lapisan teratas dibuat agar dapat diakses antar *container*. Saat proses migrasi berlangsung, hanya lapisan paling atas dan *incremental runtime memory* yang dipindahkan. Skenario migrasi dibangun dengan dua mesin virtual yang masing-masing menjalankan sebuah *instance* Docker. *Linux Traffic Control* (TC) digunakan untuk mengatur *traffic* jaringan. Lingkungan pengujian terdiri dari dua jenis, yakni lingkungan pengujian *Wide Area Network* (WAN) dengan alokasi *bandwidth* 5-45 Mbps; *latency* 50 ms; dan lingkungan pengujian *Local Area Network* (LAN) dengan alokasi *bandwidth* 50-500 Mbps; *latency* 6 ms. Layanan yang dimigrasikan adalah

Busybox sebagai beban kerja sederhana dan OpenFace sebagai beban kerja nyata. Dari evaluasi yang dilakukan, didapatkan pengurangan waktu *handoff* sebesar 56-80% dibandingkan dengan *state-of-the-art handoff* mesin virtual untuk platform *edge computing* (Ma, et al., 2019). Pada penelitian implementasi *container live migration* antar-*cloud provider* menggunakan Podman dan CRIU, lingkungan pengujian dibangun menggunakan dua mesin virtual yang terdapat pada *cloud provider* yang berbeda, yakni GCP dan AWS.

Penelitian yang dilakukan oleh Fan, et al. dengan judul "*A Locality Live Migration Strategy Based on* Docker *Containers*" mengusulkan sebuah metode dan algoritme peningkatan kinerja migrasi secara umum. Pengujian dilakukan dengan membentuk sebuah klaster berisi tiga server. Masing-masing server memiliki 8 *core* CPUs 2.6 GHz, 3 TB *disk*, 128 GB memori, dan *bandwidth* 1 Gbps. Satu server bertindak sebagai *master* dan dua server lainnya bertindak sebagai *slave*. Metode dan algoritme yang diusulkan dibandingkan dengan algoritme VectorDot dan RIAL. Hasil pengujian menunjukkan bahwa metode dan algoritme yang diusulkan mampu meningkatkan pemanfaatan dan keseimbangan kinerja sumber daya pada mesin fisik (Fan, et al., 2018). Pada penelitian implementasi *container live migration* antar-*cloud provider* menggunakan Podman dan CRIU, lingkungan pengujian dibangun menggunakan dua mesin virtual yang terdapat pada *cloud provider* yang berbeda, yakni GCP dan AWS. Tidak terdapat metode atau algoritme optimasi apapun yang diterapkan saat proses *live migration* dilakukan.

Govindaraj & Artemenko telah melakukan penelitian yang berjudul "*Container Live Migration for Latency Critical Industrial Applications on Edge Computing*". Mereka mengusulkan metode *live migration* baru bernama *redudancy migration* untuk mengurangi *downtime* saat proses migrasi berlangsung. Tidak seperti metode *live migration* dasar, *redudancy migration* tidak melakukan proses *stop-and-copy* yang menyebabkan *downtime* karena alasan pembuatan, transmisi, dan pemulihan sebuah *consistent state image* pada server tujuan. Selain itu, jika dibandingkan dengan metode optimasi *live migration* yang sudah ada, metode yang diusulkan bersifat *application agnostic*, independen terhadap lingkungan, dan cocok untuk aplikasi *Transmission Control Protocol* (TCP) maupun *User Datagram Protocol* (UDP). Migrasi dilakukan pada dua mesin virtual yang menjalankan kernel linux versi 4.13.0-rc7 dengan sistem operasi Ubuntu 16.04. Kedua mesin virtual saling terhubung menggunakan kabel *patch*. *Container engine* yang digunakan adalah LXD yang telah dimodifikasi. Hasil pengujian menunjukkan bahwa *redudancy migration* mampu mengurangi *downtime* hingga skala 1.8 dibandingkan dengan LXD *live migration* (Govindaraj & Artemenko, 2018). Pada penelitian implementasi *container live migration* antar-*cloud provider* menggunakan Podman dan CRIU, lingkungan pengujian dibangun menggunakan dua mesin virtual yang terdapat pada *cloud provider* yang berbeda, yakni GCP dan AWS. Proses *stop-and-copy* masih harus tetap dijalankan pada saat proses *live migration* dilakukan.

Penelitian yang dilakukan oleh Mirkin, et al. dengan judul "*Containers checkpointing and live migration*" mengimplementasikan *live migration* pada *container* OpenVZ. Fitur ini memungkinkan untuk menyimpan *state* dari sebuah *container* yang sedang berjalan dan memulihkannya pada *host* yang sama atau berbeda secara transparan terhadap aplikasi yang berjalan. Proses *checkpoint* dan *restart* diimplementasikan sebagai *loadable kernel module*. Efisiensinya telah terbukti di berbagai aplikasi yang ada di dunia nyata (Mirkin, et al., 2008). Mirkin, et al. tidak menjelaskan bagaimana proses pengujian dilakukan dalam penelitiannya, hanya deskripsi cara melakukan *live migration* beserta kemungkinan optimasi yang dapat dilakukan.

2.2 Dasar Teori

Terdapat enam dasar teori yang digunakan untuk mendukung penelitian implementasi *container live migration* antar-*cloud provider* menggunakan Podman dan CRIU, yaitu *cloud computing*, virtualisasi, *container*, *live migration*, MySQL, dan CRIU. Keterangan dari setiap dasar teori yang digunakan dijelaskan sebagai berikut:

2.2.1 *Cloud Computing*

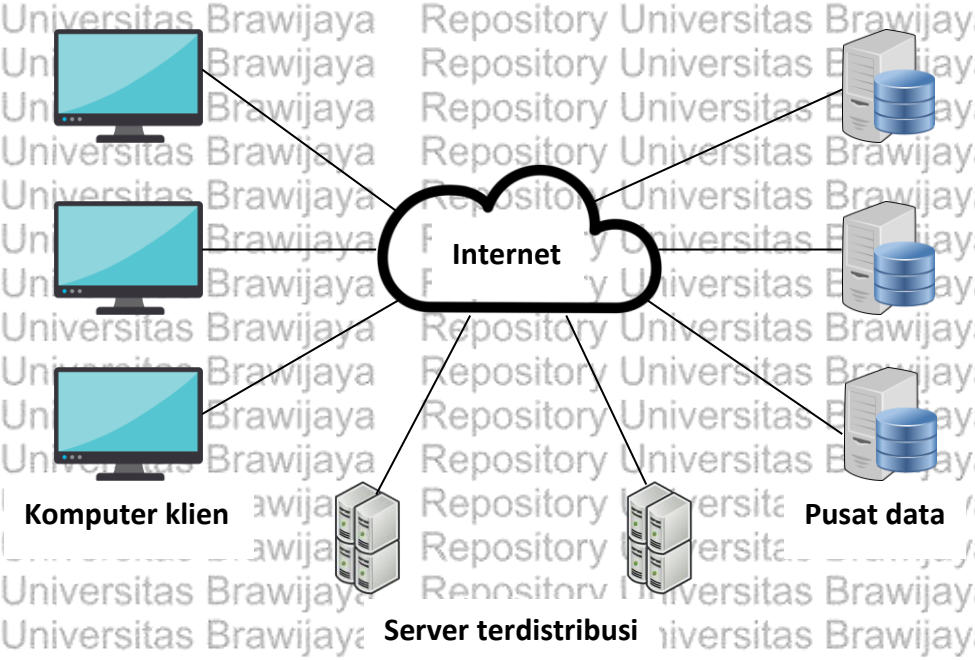
Istilah "*cloud*" pada *cloud computing* memiliki makna kumpulan dari banyak jaringan. Pengguna dapat menggunakan modalitas dari *cloud computing* secara bebas kapan pun diperlukan. Pada *cloud computing*, beban kerja dapat dipindahkan, sehingga tidak bertumpuk pada satu titik saja. Beban layanan ditangani oleh banyak jaringan yang membentuk *cloud*, sehingga beban kerja pada komputer lokal tidak terasa berat saat menjalankan aplikasi. Pengguna hanya membutuhkan aplikasi *web browser* untuk dapat menggunakan *cloud computing*. Beberapa fitur utama dari *cloud computing*, antara lain (Srivastava & Khan, 2018):

1. *Resource pooling and elasticity*
2. *Self-service and on-demand service*
3. *Pricing*
4. *Quality of service*

Terdapat tiga jenis layanan yang disediakan pada *cloud computing*, yaitu *Software as a Service* (SaaS), *Platform as a Service* (PaaS), dan *Infrastructure as a Service* (IaaS). Beberapa contoh dasar dari *cloud computing* yang digunakan secara umum diantaranya Facebook, YouTube, Dropbox, dan Gmail. Layanan-layanan tersebut menawarkan skalabilitas, fleksibilitas, kelincahan, dan kesederhanaan, sehingga penggunaannya meningkat pesat pada skala *enterprise*. *Cloud computing* memiliki tiga komponen dasar, antara lain (Srivastava & Khan, 2018):

1. Komputer klien: pengguna dapat berinteraksi dengan *cloud* menggunakan komputer klien.

2. Server terdistribusi: beberapa server didistribusikan di beberapa tempat berbeda, namun server-server tersebut terlihat layaknya bekerja dalam satu kesatuan.
3. Pusat data: pusat data merupakan kumpulan dari banyak server.



Gambar 2.1 Komponen dasar *cloud computing*

Sumber: Srivasta & Khan (2018)

Jenis-jenis *cloud computing* (Srivastava & Khan, 2018):

1. *Public cloud*: *public cloud* merupakan layanan komputasi yang disediakan oleh penyedia pihak ketiga yang berjalan di atas internet publik. Layanan ini tersedia untuk setiap pengguna yang ingin menggunakannya. Pengguna hanya perlu membayar layanan yang mereka gunakan saja.
2. *Private cloud*: Layanan komputasi yang disediakan melalui internet atau jaringan pribadi masuk ke dalam kategori *private cloud*. Layanan ini hanya dapat diakses oleh pengguna tertentu saja. Fitur keamanan dan privasi yang lebih tinggi diterapkan pada *private cloud* dengan menggunakan *firewall* dan *hosting* internal.
3. *Hybrid cloud*: *Hybrid cloud* merupakan kombinasi antara *public cloud* dengan *private cloud*. Pada *hybrid cloud*, setiap *cloud* dapat dikelola secara independen. Namun, data dan aplikasi tetap dapat dibagi antar-*cloud*.

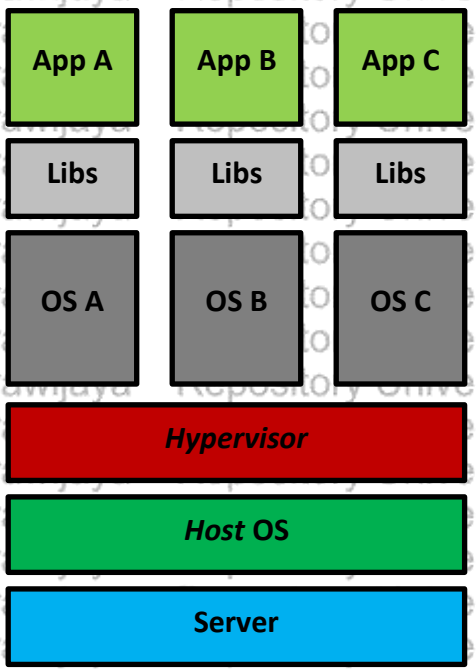
Keuntungan menggunakan *cloud computing* (Srivastava & Khan, 2018):

1. Penghematan biaya: pada *cloud computing*, pengguna hanya perlu membayar layanan yang mereka gunakan saja. Biaya perawatan cukup rendah karena pengguna tidak perlu membeli infrastrukturnya.

2. Fleksibilitas: *cloud computing* bersifat *scalable*. Peningkatan dan penurunan yang cepat dari operasi bisnis memerlukan penyesuaian perangkat keras dan sumber daya yang cepat pula. Oleh karena itu, *cloud computing* menyediakan fitur fleksibilitas.
3. Kemanan yang ditingkatkan: *cloud computing* menyediakan keamanan yang tinggi dengan menggunakan enkripsi data, akses kontrol yang kuat, manajemen kunci, dan intellijen keamanan.

2.2.2 Virtualisasi

Virtualisasi merupakan sebuah teknik yang memungkinkan untuk membuat lapisan abstrak sumber daya sistem dan menyembunyikan kompleksitas lingkungan kerja perangkat keras dan perangkat lunak. Virtualisasi meningkatkan independensi perangkat keras, isolasi sistem operasi *guest*, dan enkapsulasi seluruh mesin virtual yang dikelompokkan dalam satu berkas. Pada umumnya, virtualisasi diimplementasikan dengan teknologi *hypervisor* yang merupakan elemen perangkat lunak atau *firmware* yang mampu mengemulasi sumber daya sistem (Rashid & Chatuverdi, 2019).



Gambar 2.2 Arsitektur virtualisasi

Sumber: Bernstein (2014)

Keuntungan penggunaan virtualisasi (Rashid & Chatuverdi, 2019):

- Hemat biaya perawatan.
- Mengurangi beban kerja.
- Menawarkan *uptime* yang lebih baik.
- Memungkinkan penyebaran sumber daya yang lebih cepat.
- Mempromosikan kewirausahaan digital.
- Solusi pemulihan dari bencana yang lebih baik.

- Penggunaan energi yang efisien dan ekonomis.

Kelemahan penggunaan virtualisasi (Rashid & Chatuverdi, 2019):

- Dapat memiliki biaya implementasi yang tinggi.
- Memerlukan mesin yang kuat.
- Masih memiliki banyak keterbatasan.
- Meningkatkan resiko keamanan.
- Menciptakan masalah availabilitas dan skalabilitas.
- Memerlukan waktu mulai yang cenderung lama.

Untuk pemeliharaan sumber daya pada lingkungan *cloud computing*, virtualisasi adalah suatu keharusan karena membuatnya lebih mudah. Virtualisasi dalam *cloud computing* memungkinkan peningkatan keamanan dengan melindungi integritas komponen *cloud* begitu juga dengan mesin virtual *guest*. Mesin virtual komponen *cloud* juga dapat ditingkatkan atau diturunkan sesuai permintaan sehingga meningkatkan reliabilitas (Rashid & Chatuverdi, 2019).

Teknologi virtualisasi mengubah perspektif manusia untuk memanfaatkan sumber daya teknologi informasi dari fisik menjadi logis. Tujuan dari virtualisasi adalah untuk memanfaatkan sumber daya, seperti penyimpanan, prosesor, dan jaringan semaksimal mungkin dengan menggabungkan berbagai sumber daya yang tidak digunakan untuk menciptakan mesin virtual yang mengerjakan berbagai tugas berbeda secara serentak. Sumber daya dapat dialokasikan atau diubah secara dinamis. Ada beberapa teknik dalam melakukan virtualisasi, antara lain *hypervisor*, emulasi, *full virtualization*, *para virtualization*, dan *hardware assisted virtualization* (Rashid & Chatuverdi, 2019).

- *Hypervisor* merupakan lapisan perangkat lunak yang dapat memonitor dan memvirtualisasikan sumber daya dari mesin *host* sesuai dengan kebutuhan pengguna. *Hypervisor* merupakan lapisan perantara antara sistem operasi dan perangkat keras. Pada dasarnya, *hypervisor* diklasifikasikan menjadi dua, yaitu *native* dan *hosted*. *Hypervisor native* berjalan langsung pada perangkat keras, sedangkan *hypervisor hosted* berjalan pada sistem operasi *host*. *Hypervisor* menciptakan sumber daya virtual, seperti penyimpanan, CPU, memori, dan *driver*.
- Emulasi mengubah perilaku perangkat keras komputer menjadi program perangkat lunak yang terletak pada lapisan sistem operasi. Emulasi memberikan fleksibilitas yang sangat tinggi untuk sistem operasi *guest*, namun kecepatan proses penerjemahan yang cukup rendah dibandingkan dengan *hypervisor*, juga membutuhkan konfigurasi sumber daya perangkat keras yang tinggi.
- *Full virtualization* merupakan jenis virtualisasi yang umum digunakan dan hemat biaya. Pada dasarnya, *full virtualization* memisahkan permintaan layanan komputer dengan perangkat keras fisik yang memfasilitasinya.

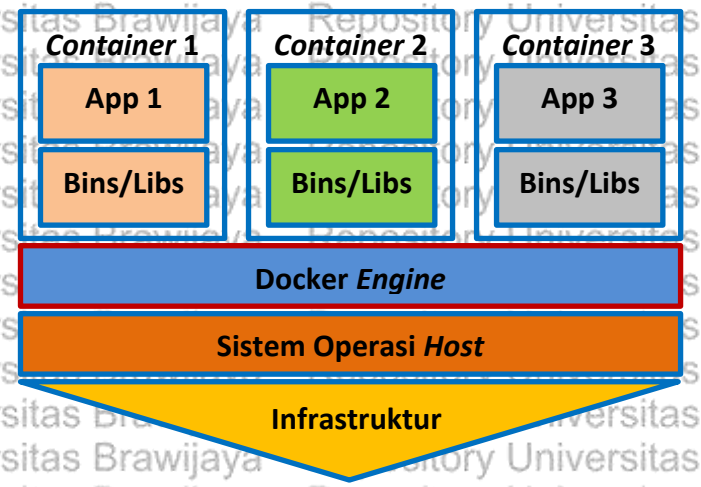
Dengan *full virtualization*, sistem operasi dan perangkat lunak yang di *hosting* dijalankan pada perangkat keras virtual.

- Pada *para virtualization*, *hypercall* khusus disediakan untuk menggantikan arsitektur set instruksi dari mesin *host*. *Para virtualization* menghubungkan komunikasi antara sistem operasi *guest* dan *hypervisor* untuk meningkatkan kinerja dan efisiensi. Pengaksesan sumber daya pada *para virtualization* lebih baik daripada model *full virtualization*. Karena pada model *full virtualization* semua sumber daya harus divirtualisasikan secara penuh. Kekurangan dari *para virtualization* adalah model ini memodifikasi kernel sistem operasi *guest* menggunakan *hypercall*, sehingga hanya cocok digunakan dengan sistem operasi *open source*.

2.2.3 *Container*

Container menyediakan lingkungan terisolasi mirip dengan virtualisasi tetapi tanpa emulasi perangkat keras virtual. Istilah *container* sudah cukup lama dalam dunia komputasi. Namun, *container* tidak pernah diimplementasikan dalam skala besar. *Container* berjalan pada *userspace* di atas kernel sistem operasi. Oleh karena itu, virtualisasi *container* dikenal dengan sebutan OS-*level virtualization*. Teknologi *container* memungkinkan beberapa *userspace instance* terisolasi dijalankan pada suatu *host*. *Container* dapat diklasifikasikan menjadi dua kategori, yaitu *container* sistem dan *container* aplikasi. *Container* sistem mirip dengan sistem operasi utuh dan menjalankan semua proses seperti *init*, *inetd*, *sshd*, *syslogd*, dan *cron*. Sedangkan *container* aplikasi hanya dapat menjalankan aplikasi saja. Kedua jenis *container* tersebut memiliki manfaat dalam keadaan yang berbeda. Beberapa contoh teknologi *container* yang tersedia saat ini, antara lain OpenVZ, LXC, Solaris Zones, systemd-nspawn, *let me contain that for you* (lmctfy), Warden, dan Docker (Naik, 2016).

2.2.3.1 Docker



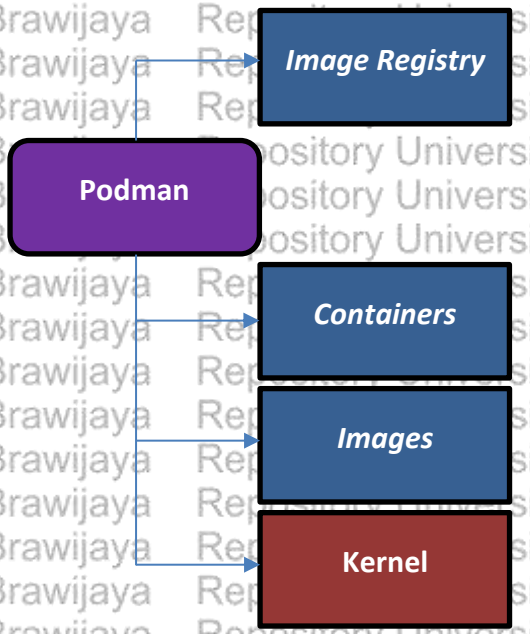
Gambar 2.3 *Container* Docker pada *host* linux

Sumber: Naik (2016)

Docker merupakan sebuah perangkat lunak *open-source* yang mengotomatisasikan penyebaran aplikasi ke dalam *container*. Docker dikembangkan oleh Docker, Inc., yang sebelumnya dikenal sebagai dotCloud, Inc. yang merupakan salah satu perusahaan perintis di pasar *platform-as-a-service*. Docker menyediakan *container* terisolasi berdasarkan fitur kernel pada kebanyakan linux yang dikenal sebagai *cgroup* dan *namespace*. Sehingga, setiap *container* dapat memiliki isolasi yang kuat, memiliki *stack* jaringan tersendiri, *stack* penyimpanan, *file system*, dan kemampuan manajemen sumber daya yang memungkinkan beberapa *container* sejenis dijalankan pada satu *host*. Sebuah *container* tidak memiliki sistem operasinya sendiri karena *container* tersebut berbagi kernel yang sama dengan *host*. Namun, *container* memiliki semua *binary* dan *library* untuk menjalankan aplikasi tertentu di dalamnya (Naik, 2016).

2.2.3.2 Podman

Podman adalah sebuah proyek *open-source* yang tersedia di sebagian besar platform linux yang bisa diunduh melalui GitHub. Podman merupakan *daemonless container engine* yang digunakan untuk mengembangkan, mengatur, dan menjalankan OCI *container* dan *container image* pada sistem linux. Podman menyediakan *syntax Command Line Interface* (CLI) yang serupa dengan Docker (Podman, 2019).



Gambar 2.4 Arsitektur Podman

Sumber: Red Hat Developer (2019)

Container pada Podman dapat dijalankan dengan atau tanpa otoritas root. Podman mengatur keseluruhan ekosistem *container* yang terdiri dari *pod*, *container*, *container image*, dan *container volume* menggunakan *library* yang bernama *libpod*. Podman menyediakan semua fitur yang membantu pengguna dalam memelihara atau memodifikasi OCI *container image*, seperti menarik

image dan memberikan *tag* pada *image*. Pengguna juga dapat membuat, menjalankan, dan melakukan pemeliharaan pada *container* yang terbuat dari *image* tersebut pada lingkungan produksi (Podman, 2019).

Pada tingkatan yang tinggi, ruang lingkup dari *libpod* dan Podman adalah sebagai berikut (Podman, 2019):

- Mendukung berbagai format *image* termasuk format *image* OCI dan Docker.
- Mendukung berbagai cara mengunduh *image* secara aman, termasuk verifikasi *image*.
- Manajemen *container image* (mengatur lapisan *image*, *overlay filesystem*, dan sebagainya).
- Manajemen penuh terhadap siklus hidup *container*.
- Dukungan terhadap *pod* untuk mengatur kumpulan *container*.
- Isolasi sumber daya dari *container* dan *pod*.

2.2.4 *Live Migration*

Live migration merupakan sebuah konsep memindahkan sebuah layanan, independen terhadap lingkungan virtualisasi, dari satu server ke server lain tanpa kehilangan *state* dari aplikasi yang sedang berjalan. Hal ini juga melibatkan migrasi CPU, memori, dan status jaringan. Server tujuan memerlukan ketiga hal tersebut agar dapat memulihkan layanan dengan benar. Selama proses migrasi berlangsung, *state* memori akan terus berubah. Tingkat kecepatan perubahan dan ukuran *state* tergantung dari layanan yang akan dipindahkan (Govindaraj & Artemenko, 2018).

Proses *live migration* diukur menggunakan dua satuan pengukuran, yakni *downtime* dan *migration time*. *Downtime* berkaitan dengan durasi layanan dihentikan sepenuhnya pada saat proses migrasi berlangsung. Selama periode ini, layanan tidak dapat diakses oleh pengguna, sehingga berpengaruh pada *Service Level Agreement* (SLA) dari layanan tersebut. *Migration time* merupakan durasi total dari dimulainya proses migrasi hingga layanan berhasil dipulihkan pada server tujuan dan mulai berjalan. Pada titik ini, layanan di server utama sudah dapat dinonaktifkan (Govindaraj & Artemenko, 2018).

Terdapat tiga skema untuk mengurangi *downtime* pada *live migration*, yaitu *pre-copy migration, post-copy migration*, dan *hybrid*. Pada *pre-copy migration*, *downtime* bergantung pada ukuran *state* dari memori yang akan dipindahkan saat fase *stop-and-copy*. Skema ini memiliki pengaruh yang minim terhadap kinerja layanan selama proses migrasi berlangsung. Selain itu, jika server tujuan mengalami kegagalan dalam proses pemulihan layanan, tidak akan memengaruhi kinerja dari server sumber dan pengguna tetap bisa menggunakan layanan secara normal. Dibandingkan dengan *pre-copy migration*, *post-copy migration* dapat menyelesaikan masalah *non-converging working sets* dan memiliki *downtime* yang lebih kecil dikarenakan ukuran *state* CPU lebih kecil dari *state* memori. Namun, klien akan mengalami penurunan kinerja layanan selama fase

pull berlangsung. Selain itu, karena server sumber di *suspend* setelah fase *stop-and-copy*, tidak ada mekanisme penanganan kegagalan jika server tujuan gagal untuk melanjutkan layanan kembali. Pada skema migrasi *hybrid*, *downtime* dan *migration time* dapat dioptimalkan berdasarkan aturan tertentu. Namun, fase *stop-and-copy* masih tetap harus dijalankan (Govindaraj & Artemenko, 2018).

2.2.5 MySQL

MySQL merupakan *Relational Database Management System* (RDBMS) yang digunakan sebagai layanan SaaS pada *cloud*. MySQL dirilis pada tahun 1995, dan sekarang dipegang oleh perusahaan Oracle. MySQL merupakan sistem manajemen basis data yang paling populer untuk penyedia layanan *hosting* seperti Rackspace, GoDaddy, Bluehost, dan WHM. Selain itu, Facebook, Twitter, Yahoo, Wikipedia, dan YouTube juga memanfaatkan MySQL (Dawodi, et al., 2019).

MySQL mampu mendukung 4096 kolom per tabel. Ukuran *BLOB* adalah 16777215 karakter dan ukuran *TEXTS* adalah 65535 karakter. MySQL *table engines* digunakan untuk membuat berkas sesuai dengan nama tabel, kemudian data yang terdapat pada tabel disimpan ke dalam berkas tersebut. Hal ini dilakukan ketika mendapat permintaan dari pengguna, yang kemudian MySQL menjalankan dan menampilkan hasilnya. MySQL mendukung tabel *transactional* dan *non-transactional* secara sederhana. Hanya *engine* InnoDB yang mendukung proses transaksi. Dalam suatu transaksi, pada dasarnya sebuah *engine* dapat menggunakan beberapa perintah seperti *rollback, commit,* dan *savepoint*. *Engine* yang umum digunakan pada MySQL adalah InnoDB dan MyISAM (Dawodi, et al., 2019).

2.2.6 CRIU

Checkpoint/Restore In Userspace (CRIU) merupakan sebuah *package* untuk melakukan *checkpoint* dan memulihkan *state* dari sebuah proses. *Checkpoint* merupakan tindakan pembuatan *snapshot state* lengkap dari sebuah proses, termasuk konten memori, *file descriptor*, dan koneksi TCP yang sedang berlangsung. CRIU dapat digunakan untuk menunda, melanjutkan, dan/atau memindahkan suatu proses dari satu mesin ke mesin lainnya (Huang & Lee, 2017).

- *Checkpoint*: prosedur *checkpoint* bergantung pada sistem berkas /proc untuk mendapatkan informasi dari sebuah proses. Prosedur ini memerlukan parameter *pipe*, informasi deskriptor, dan *memory map*. Proses *checkpoint* terdiri dari tiga langkah, yaitu:

  1. Menjelajahi direktori /proc/$pid/task secara rekursif dan mengumpulkan *process tree*.
  2. Mengumpulkan sumber daya tugas dan melakukan *dump* untuk dijadikan berkas *image*. Jika terdapat permintaan *pre-dump*, CRIU hanya akan mengumpulkan *process tree* dan *memory page*. Jika tidak, CRIU akan mengumpulkan *process tree*, *memory page*, *namespace*,

cgroup, *shared memory*, dan jaringan. Dengan parameter *track-memory*, CRIU akan membandingkan *checkpoint image* dan *memory page* untuk mengurangi ruang penyimpanan.

3. Setelah melakukan proses *dump* pada *image*, CRIU memisahkan diri dari *task*, dan *task* tersebut dapat terus berjalan.

- *Restore*: Ketika melakukan pemulihan suatu proses, CRIU akan merubah dirinya sendiri menjadi *task* tersimpan. Proses pemulihan terdiri dari empat tahapan, yakni:

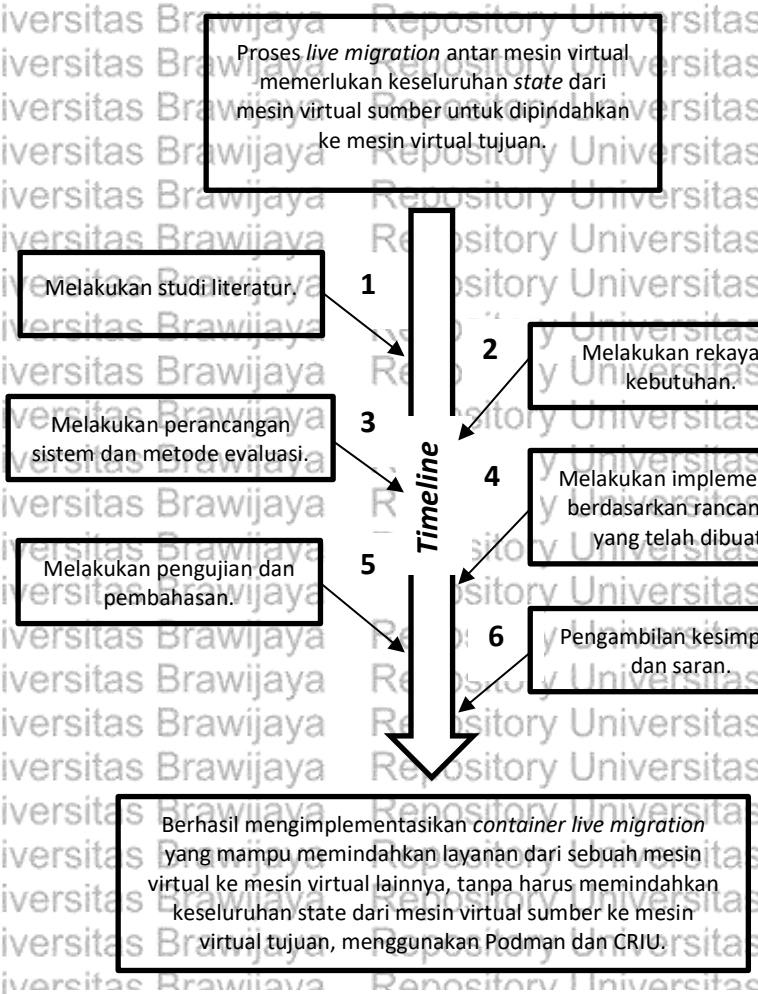
1. CRIU membaca berkas *image* dan menyelesaikan pemetaan antar proses dan sumber daya. Sumber daya yang dibagi, dipulihkan oleh suatu proses; yang lainnya mewarisi sumber daya pada tingkatan kedua (seperti *session*) atau mendapatkannya dengan cara lain.
2. CRIU memanggil fungsi fork() sebanyak *process tree* untuk membuat proses yang akan dipulihkan.
3. Memulihkan sumber daya seperti pemetaan memori, perhitungan waktu, *namespace*, *cgroup*, *thread*, dan lainnya.
4. CRIU merubah dirinya sendiri menjadi proses dan melanjutkan pekerjaan dari proses tersebut.

BAB 3 METODOLOGI PENELITIAN

Bab ini menjelaskan tentang metodologi penelitian yang digunakan dalam implementasi *container live migration* antar-*cloud provider* menggunakan Podman dan CRIU yang terdiri dari tiga bagian, yakni kerangka penelitian, perancangan sistem, dan metode evaluasi. Kerangka penelitian mengulas identifikasi masalah, rumusan masalah, dan tujuan penelitian yang akan dicapai. Perancangan sistem berisi gambaran umum sistem yang dikembangkan. Metode evaluasi berisi parameter dan skenario uji sebagai dasar dalam melakukan tahap pengujian.

3.1 Kerangka Penelitian

Kerangka penelitian digunakan sebagai ulasan dari identifikasi masalah, rumusan masalah, dan tujuan penelitian yang akan dicapai. Kerangka penelitian yang digunakan dalam penelitian implementasi *container live migration* antar-*cloud provider* menggunakan Podman dan CRIU ditunjukkan pada Gambar 3.1.



Gambar 3.1 Diagram kerangka penelitian

Proses penelitian dimulai dengan tahap identifikasi masalah yang akan diselesaikan. Pencarian masalah dilakukan dengan membaca dan mempelajari berbagai literatur yang berfokus pada topik *container*, migrasi, dan *cloud computing*. Metode ini disebut dengan metode deduksi, di mana penentuan masalah dimulai dari masalah umum ke masalah yang lebih spesifik. Dari studi literatur awal yang dilakukan, ditemukan sebuah masalah spesifik, yaitu proses *live migration* antar mesin virtual memerlukan keseluruhan *state* dari mesin virtual sumber untuk dipindahkan ke mesin virtual tujuan. Setelah masalah ditemukan, selanjutnya dilakukan studi literatur kembali untuk memahami konsep, istilah, dan metode yang akan digunakan dalam penelitian. Langkah-langkah berikutnya adalah melakukan rekayasa kebutuhan, melakukan perancangan sistem dan metode evaluasi, melakukan implementasi berdasarkan rancangan yang telah dibuat, melakukan pengujian dan pembahasan, serta pengambilan kesimpulan dan saran. Masing-masing dari setiap langkah tersebut akan dijelaskan lebih detail pada Subbab 3.1.1 Studi Literatur, 3.1.2 Rekayasa Kebutuhan, 3.1.3 Perancangan, 3.1.4 Implementasi, 3.1.5 Pengujian, dan 3.1.6 Pengambilan Kesimpulan Dan Saran.

3.1.1 Studi Literatur

Studi literatur dilakukan untuk menemukan pustaka dan dasar teori yang sesuai guna menunjang seluruh kegiatan penelitian implementasi *container live migration* antar-*cloud provider* menggunakan Podman dan CRIU. Pustaka berisi pembahasan penelitian-penelitian sebelumnya yang relevan terhadap penelitian yang dilakukan, serta memberikan gambaran umum persamaan dan perbedaannya. Dasar teori yang digunakan pada penelitian ini, antara lain *cloud computing*, virtualisasi, *container*, *live migration*, MySQL, dan CRIU. Literatur-literatur tersebut bersumber dari hasil konferensi, jurnal, situs web resmi, dan hasil penelitian sebelumnya yang relevan.

3.1.2 Rekayasa Kebutuhan

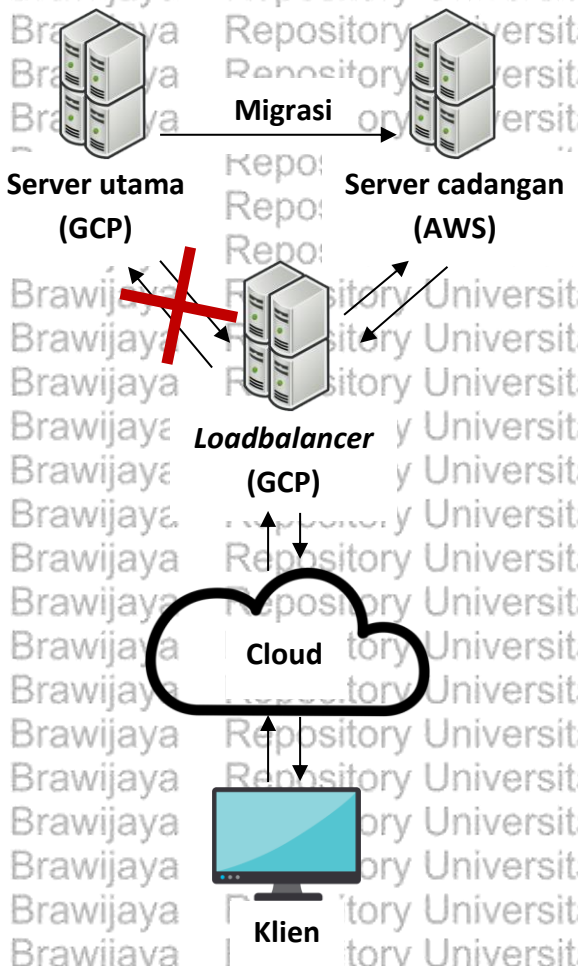
Rekayasa kebutuhan diperlukan untuk mengetahui hal-hal yang dibutuhkan pada penelitian implementasi *container live migration* antar-*cloud provider* menggunakan Podman dan CRIU, serta digunakan sebagai acuan pada tahap perancangan dan implementasi. Rekayasa kebutuhan terdiri dari dua jenis, yakni kebutuhan fungsional dan kebutuhan non-fungsional. Kebutuhan fungsional merupakan semua kebutuhan yang wajib dipenuhi oleh sistem terkait hal-hal yang berfokus pada tingkah laku data batasan fitur pada sistem. Beberapa kebutuhan fungsional yang terdapat di dalam penelitian ini antara lain, *load balancer* dapat mengarahkan *traffic* permintaan dari klien menuju server yang aktif, server mampu merespon permintaan dari klien, server yang memiliki layanan mampu memigrasikan layanannya ke server lainnya, dan server mampu menjaga keseluruhan *state* dari layanan sesudah proses *live migration* dilakukan.

Kebutuhan non-fungsional dapat dibedakan menjadi dua jenis, yaitu kebutuhan perangkat keras dan kebutuhan perangkat lunak. Kebutuhan perangkat keras yang diperlukan pada penelitian ini adalah sebuah komputer

personal sebagai media untuk terhubung pada layanan *cloud* dan tempat untuk mengembangkan *script live migration*. Kebutuhan perangkat lunak yang dibutuhkan dalam penelitian ini, antara lain sistem operasi Fedora 30 Workstation x64 sebagai sistem operasi komputer personal, sistem operasi Red Hat Enterprise Linux 8 sebagai sistem operasi server utama dan cadangan pada *cloud*, sistem operasi CentOS 8 sebagai sistem operasi *load balancer* pada *cloud*, Podman sebagai *container engine*, CRIU sebagai aplikasi yang membantu dalam implementasi *live migration*, server *engine* Apache untuk menjalankan layanan web pada *cloud*, MySQL sebagai media penyimpanan data pengguna, NGINX sebagai *load balancer* pada *cloud,* dan Python2.7 sebagai bahasa pemrograman yang digunakan untuk membuat *script live migration.*

3.1.3 Perancangan

Perancangan dilakukan untuk mempermudah tahap implementasi dan pengujian agar menjadi lebih terstruktur dan terarah. Pada penelitian implementasi *container live migration* antar-*cloud provider* menggunakan Podman dan CRIU, perancangan terdiri dari lima bagian, yakni perancangan topologi, perancangan server utama, perancangan server cadangan, perancangan mekanisme *live migration*, dan perancangan pengujian.



Gambar 3.2 Gambaran umum sistem

Perancangan topologi diperlukan untuk memperjelas alur pembangunan lingkungan sistem yang dikembangkan, mulai dari penamaan, pengalamatan IP, peran, dan alur komunikasi setiap *node* yang terdapat di dalam sistem. Perancangan server utama dilakukan pada layanan *cloud* milik Google, yakni Google Cloud Platform (GCP). Perancangan server cadangan dilakukan pada layanan *cloud* milik Amazon, yakni Amazon Web Service (AWS). Perancangan mekanisme *live migration* dilakukan dengan membuat *load balancer* dan *script live migration*. Dengan *load balancer* memungkinkan pengguna untuk terhubung ke satu alamat IP saja, tanpa perlu tau keadaan sebenarnya dibalik proses *live migration* yang sedang terjadi. Perancangan pengujian dilakukan untuk menetapkan batasan minimal yang wajib dipenuhi oleh sistem, serta skenario pengujian *live migration* antar server. Pada penelitian ini, perancangan pengujian terdiri dari tiga bagian, yakni perancangan pengujian fungsional, *downtime* dan *migration time*.

3.1.4 Implementasi

Implementasi merupakan proses penerapan rancangan sesuai dengan rekayasa kebutuhan yang telah dilakukan. Pada penelitian ini, tahap implementasi dimulai dengan membangun *instance* server utama pada Google Cloud Platform (GCP) menggunakan fitur Google *Compute Engine*. *Instance* server utama menggunakan sistem operasi Red Hat Enterprise Linux (RHEL) 8 dan dipasangkan beberapa aplikasi, yakni Podman, CRIU, dan beberapa depedensi yang diperlukan. Pada *instance* server utama juga dilakukan proses pembuatan alamat IP menjadi statis dan proses pembuatan pasangan *Secure Shell* (SSH) *key* agar komputer pribadi peneliti dapat terhubung. Langkah berikutnya adalah membangun *instance load balancer* pada GCP. Sistem operasi pada *instance load balancer* menggunakan CentOS 8, alamat IP dibuat statis, juga dilakukan proses pembuatan pasangan SSH *key*. Pada *instance load balancer* dipasangkan aplikasi NGINX yang akan bertindak sebagai layanan *load balancer*. Tahap selanjutnya adalah membangun *instance* server cadangan di AWS. Proses dan pengaturan untuk *instance* server cadangan sama seperti *instance* server utama. Setelah pembangunan dan pengaturan ketiga *instance* telah selesai, dilakukan pembangunan layanan berupa layanan web *video streaming* menggunakan bahasa pemrograman *HyperText Markup Language* (HTML) dan *Hypertext Preprocessor* (PHP). Pada *instance* server utama dan cadangan juga dipasangkan aplikasi MySQL untuk menyimpan data pengguna. Setelah semua tahapan selesai dilakukan, *instance* server utama siap dijalankan dan pengguna sudah dapat menggunakan layanan dari *instance* server utama. Langkah terakhir adalah melakukan pengujian dari sistem yang sudah dibangun.

3.1.5 Pengujian

Terdapat tiga jenis pengujian pada penelitian ini, yaitu pengujian fungsional, pengujian *downtime*, dan pengujian *migration time*. Pengujian fungsional dilakukan untuk mengetahui apakah sistem yang dibangun telah memenuhi seluruh kebutuhan fungsional yang ditetapkan. Pengujian berikutnya hanya

dapat dilakukan apabila semua kebutuhan fungsional telah terpenuhi. Apabila terdapat kebutuhan fungsional yang belum terpenuhi, maka dilakukan pengecekan kembali terhadap rancangan dari kode sistem, kemudian dilakukan pengujian ulang hingga semua kebutuhan terpenuhi.

Pengujian *downtime* berkaitan dengan durasi layanan dihentikan sepenuhnya pada saat proses migrasi berlangsung. Selama periode ini, layanan tidak dapat diakses oleh pengguna, sehingga berpengaruh pada *Service Level Agreement* (SLA) dari layanan tersebut. Pengujian *migration time* merupakan durasi total dari dimulainya proses migrasi hingga layanan berhasil dipulihkan pada server tujuan dan mulai berjalan. Baik *downtime* maupun *migration time* diukur menggunakan satuan detik.

3.1.6 Pengambilan Kesimpulan Dan Saran

Pengambilan kesimpulan dan saran merupakan tahapan paling akhir pada sebuah penelitian. Pengambilan kesimpulan dilakukan setelah seluruh tahapan penelitian selesai dilaksanakan mulai dari tahap studi literatur, rekayasa kebutuhan, perancangan, implementasi, serta pengujian. Kesimpulan yang dibuat harus mampu menjawab rumusan masalah yang diutarakan pada BAB 1 PENDAHULUAN. Saran merupakan masukan terhadap kekurangan-kekurangan yang terdapat di dalam penelitian ini sehingga dapat digunakan sebagai pertimbangan pada penelitian selanjutnya.

3.2 Perancangan Sistem

Perancangan pada penelitian implementasi *container live migration* antar-*cloud provider* menggunakan Podman dan CRIU terdiri dari perancangan topologi, perancangan server utama, perancangan server cadangan, perancangan mekanisme *live migration*, dan perancangan pengujian. Perancangan pengujian akan dibahas tersendiri pada Subbab 3.3 Metode Evaluasi. Sebelum masuk ke tahap perancangan, dijelaskan kebutuhan-kebutuhan yang diperlukan oleh sistem.

3.2.1 Deskripsi Umum Sistem

Penelitian yang dilakukan berfokus pada kemampuan Podman dan CRIU dalam melakukan migrasi antar-*cloud provider*, yakni GCP dan AWS. Proses nya dimulai dengan membangun dan melakukan konfigurasi sebuah server utama pada GCP. Server utama dipasangi perangkat lunak sistem operasi Red Hat Enterprise Linux 8, Podman, CRIU, dan beberapa depedensi yang diperlukan. Kemudian dilanjutkan dengan membangun dan melakukan konfigurasi *load balancer* pada GCP. *Load balancer* dipasangi perangkat lunak sistem operasi CentOs 8, NGINX dan beberapa depedensi yang diperlukan. Kemudian dilanjutkan lagi dengan membangun dan melakukan konfigurasi server cadangan pada AWS. Konfigurasi pada server cadangan sama dengan konfigurasi server utama. Python2.7 digunakan dalam pembuatan *script live migration*.

3.2.2 Rekayasa Kebutuhan

Rekayasa kebutuhan diperlukan untuk mengetahui hal-hal yang dibutuhkan pada penelitian implementasi *container live migration* antar-*cloud provider* menggunakan Podman dan CRIU. Rekayasa kebutuhan terdiri dari dua jenis, yakni kebutuhan fungsional dan kebutuhan non-fungsional.

3.2.2.1 Kebutuhan Fungsional

Kebutuhan fungsional merupakan semua kebutuhan yang wajib dipenuhi oleh sistem terkait hal-hal yang mampu dilakukannya. Kebutuhan fungsional pada penelitian ini dijabarkan pada Tabel 3.1.

Tabel 3.1 Kebutuhan fungsional

No.	Kebutuhan Fungsional
1	*Load balancer* dapat mengarahkan *traffic* permintaan dari klien menuju server yang memiliki layanan.
2	Server yang memiliki layanan mampu merespon permintaan dari klien.
3	Server yang memiliki layanan mampu memigrasikan layanannya ke server lainnya.
4	Server yang memiliki layanan mampu menjaga keseluruhan *state* dari layanan sesudah proses *live migration* dilakukan.

3.2.2.2 Kebutuhan Non-Fungsional

Kebutuhan non-fungsional merupakan kebutuhan yang berfokus pada tingkah laku dan batasan fitur pada sistem. Kebutuhan non-fungsional dapat dibedakan menjadi dua jenis, yaitu kebutuhan perangkat keras dan kebutuhan perangkat lunak.

❖ **Kebutuhan Perangkat Keras**

Perangkat keras yang dibutuhkan dalam implementasi *container live migration* antar-*cloud provider* menggunakan Podman dan CRIU, yaitu sebuah komputer personal dengan spesifikasi sebagai berikut:

- CPU: Intel Pentium 4 atau prosesor yang lebih baru, dengan kecepatan 1 GHz atau lebih cepat, yang mendukung SSE2.
- VGA: *Graphic adapter* dengan dukungan DirectX 9 atau versi yang lebih baru dengan *driver* WDDM 1.0, apabila menggunakan sistem operasi Windows.
- Resolusi layar: 800 x 600 atau lebih tinggi, untuk dapat menggunakan *Graphical User Interface* (GUI) dari sistem operasi yang digunakan.
- HDD: Minimal 20 GB untuk sistem operasi, dan 1 GB atau lebih untuk penyimpanan data yang digunakan.

- RAM: Minimal 1 GB untuk sistem operasi 32 bit dan minimal 2 GB untuk sistem operasi 64 bit.

❖ **Kebutuhan Perangkat Lunak**

Perangkat lunak yang dibutuhkan dalam implementasi *container live migration* antar-*cloud provider* menggunakan Podman dan CRIU, ditunjukkan pada Tabel 3.2.

Tabel 3.2 Kebutuhan perangkat lunak

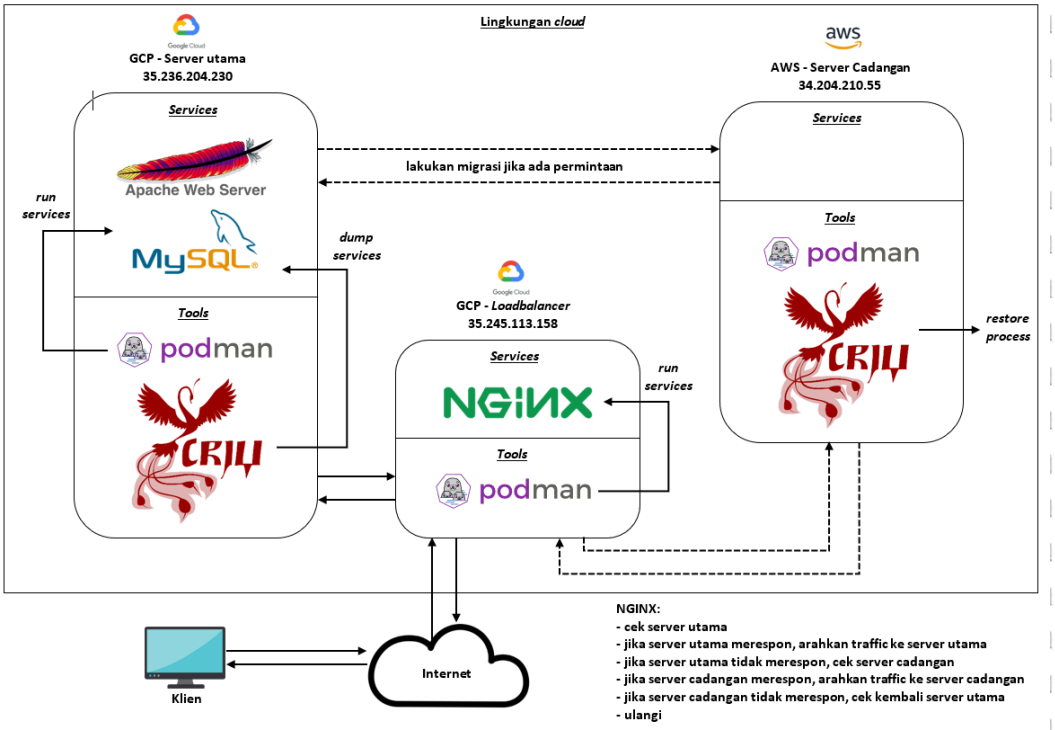
Perangkat Lunak	Keterangan
Fedora 30 Workstation x64	Sistem operasi yang digunakan pada komputer personal peneliti.
Red Hat Enterprise Linux 8 x64	Sistem operasi yang digunakan pada server utama (GCP) dan server cadangan (AWS).
CentOS 8 x64	Sistem operasi yang digunakan pada *loadbalancer* pada GCP.
Python 2.7.x	Bahasa pemrograman yang digunakan dalam pembuatan *script live migration*.
Podman 1.6.4	*Container engine* yang digunakan.
CRIU 3.13	*Library* yang digunakan untuk menyimpan dan memulihkan *state* dari sebuah aplikasi.
Buildah 1.11.6	Depedensi yang diperlukan untuk menjalankan Podman.
Nano 2.9.8	*Text editor* sederhana yang digunakan untuk mempermudah proses perubahan sebuah berkas teks.
Rsync 3.1.3	Aplikasi yang digunakan untuk melakukan sinkronisasi berkas antar *host* yang berbeda.
NGINX (*latest* - Docker hub *repository*)	Digunakan sebagai *loadbalancer* yang diletakkan pada GCP.
Apache server (*latest* - Docker hub *repository*)	Digunakan sebagai webserver untuk menjalankan layanan utama.
MySQL (*latest* - Docker hub *repository*)	Digunakan untuk menyimpan data pengguna.
Sublime Text 3.x	*Text editor* yang digunakan untuk membantu dalam pembuatan *script live migration*.

Pada Tabel 3.2 dapat dilihat bahwa semua sistem operasi yang digunakan merupakan distro Linux dengan basis RHEL. RHEL merupakan salah satu sistem operasi berskala *enterprise* yang dibangun dan dioptimasi untuk pengembangan

sistem dan perangkat lunak, manajemen server, serta hal-hal berskala *enterprise* lainnya (Red Hat Enterprise Linux, 2019), sehingga sangat cocok digunakan dalam penelitian ini. Selain itu, versi yang digunakan dari setiap perangkat lunak merupakan versi stabil terbaru saat penelitian dimulai. Hal ini dilakukan untuk meminimalkan kendala akibat *bug* yang terdapat pada perangkat lunak, dan pemanfaatan fitur-fitur baru yang dapat membantu proses penelitian.

3.2.3 Perancangan Topologi

Perancangan topologi diperlukan untuk mendapatkan gambaran jelas terkait alur dan hubungan antar entitas dalam penelitian implementasi *container live migration* antar-*cloud provider* menggunakan Podman dan CRIU. Dengan adanya perancangan topologi, proses implementasi akan jadi lebih terstruktur dan terarah. Rancangan topologi ini menggambarkan bagaimana *node* klien terhubung ke *load balancer* yang kemudian diarahkan ke salah satu server yang memiliki layanan yang diminta. Selain itu, perancangan topologi juga menggambarkan secara detail penamaan, pengalamatan IP, peran, dan alur komunikasi setiap *node* yang terdapat di dalam sistem. Perancangan topologi ditunjukkan pada Gambar 3.3.



Gambar 3.3 Perancangan topologi

Klien terhubung pada *load balancer* yang ada pada GCP dengan IP publik 35.245.113.158 melalui internet. *Load balancer* mengarahkan permintaan dari klien ke server utama pada GCP atau server cadangan pada AWS, tergantung server mana yang memiliki layanan sesuai dengan permintaan klien. Server yang memiliki layanan memberikan respon ke *load balancer* yang diteruskan pada klien. IP publik dari server utama adalah 35.236.204.230 dan IP publik dari server

cadangan adalah 34.204.210.55. IP publik dari setiap *instance* didapatkan secara otomatis dari *cloud provider* dan telah dibuat statis agar tidak berubah saat *instance* dinyalakan ulang. Proses sinkronisasi data antar server dilakukan dengan sebuah *script* yang memanfaatkan aplikasi rsync.

3.2.4 Perancangan Server Utama

Perancangan server utama dilakukan pada layanan *cloud* milik Google yakni GCP. Server utama harus mampu merespon permintaan dari klien, mampu memigrasikan layanan saat diperlukan, dan mampu menjalankan layanan setelah menerima permintaan migrasi dari server cadangan. Perancangan server utama ditunjukkan pada Gambar 3.4.

Gambar 3.4 Perancangan server utama

Proses perancangan server utama dimulai dengan membuat pasangan SSH *key* pada komputer lokal agar dapat terhubung ke server utama pada *cloud*. Untuk membuat pasangan SSH *key* sebagai pengguna biasa menggunakan perintah `ssh-keygen -t rsa -f ~/.ssh/gcp-user-1 -C abdulazizm41` dan untuk root menggunakan perintah `ssh-keygen -t rsa -f ~/.ssh/gcp-root-1 -C root`. "abdulazizm41" merupakan nama pengguna yang digunakan pada *instance* server utama dan pengguna root diperlukan dalam proses migrasi. Langkah berikutnya adalah menambahkan *public key* yang telah dibuat pada projek GCP. Caranya adalah dengan masuk ke pilihan *compute engine* -> *metadata* -> SSH. Selanjutnya menambahkan *rule firewall* dev-allow-all pada pilihan VPC *network* -

> *firewall rules*. *Rule* ini digunakan untuk mempermudah pengaksesan *instance* server utama. Spesifikasi *rule firewall* yang digunakan terdapat pada Tabel 3.3.

Tabel 3.3 *Rule firewall* GCP dev-allow-all

Nama	development-allow-all
Jenis	*ingress*
Tujuan	allow-all-traffic
Filter	IP *range:* 0.0.0.0/0
Protokol / port	*all*
Tindakan	*allow*
Prioritas	1000
Jaringan	*default*

Langkah berikutnya adalah pembuatan *instance* server utama pada GCP. Caranya adalah dengan masuk ke pilihan *compute engine* -> VM *instance* -> *create instance*. Spesifikasi dari *instance* yang dibuat ditunjukkan pada Tabel 3.4.

Tabel 3.4 Spesifikasi *instance* server utama GCP

Name	gcp-server-1
Region	us-east4 (*North* Virgiana)
Zone	us-east4c
Machine series	N1
Machine type	*custom*, 1 *core*, 1 GB *memory*
CPU *platform*	Intel Skylake++
Boot disk	Red Hat Enterprise Linux 8 x64 (10 GB)
Firewall tags	allow-all-traffic

Langkah selanjutnya adalah melakukan *promote* alamat IP agar menjadi statis. Caranya adalah dengan masuk ke pilihan VPC *network* -> *external* IP *address* -> *reserve static address*. Pada penelitian ini, IP statis yang didapatkan adalah 35.236.204.230. Setelah *promote* IP selesai dilakukan, jalankan *instance* server utama. Lakukan *update* dan instalasi perangkat lunak yang diperlukan. Perangkat lunak yang di instal pada *instance* server utama beserta penjelasan kegunaannya terdapat pada Tabel 3.5.

Tabel 3.5 Perangkat lunak pada server utama

Perangkat Lunak	Keterangan
Python 2.7.x	Bahasa pemrograman yang digunakan dalam pembuatan *script live migration*.

Tabel 3.5 (lanjutan)

Perangkat Lunak	Keterangan
Podman 1.6.4	*Container engine* yang digunakan.
CRIU 3.13	*Library* yang digunakan untuk menyimpan dan memulihkan *state* dari sebuah aplikasi.
Buildah 1.11.6	Depedensi yang diperlukan untuk menjalankan Podman.
Nano 2.9.8	*Text editor* sederhana yang digunakan untuk mempermudah proses perubahan sebuah berkas teks.
Rsync 3.1.3	Aplikasi yang digunakan untuk melakukan sinkronisasi berkas antar *host* yang berbeda.
Apache server (*latest* – Docker hub *repository*)	Digunakan sebagai webserver untuk menjalankan layanan utama.
MySQL (*latest* – Docker hub *repository*)	Digunakan untuk menyimpan data pengguna.

Setelah proses *update* dan instalasi perangkat lunak selesai dilakukan, langkah berikutnya adalah melakukan konfigurasi sshd dan selinux. Konfigurasi sshd dilakukan dengan melakukan perubahan pada berkas /etc/ssh/sshd_config, yakni pada baris "PermitRootLogin yes" dan "PasswordAuthentication no". Konfigurasi selinux dilakukan dengan melakukan perubahan pada berkas /etc/selinux/config, dengan merubahnya menjadi "disabled". Konfigurasi selinux dapat dicek menggunakan perintah `getenforce` dan `sestatus`.

Langkah terakhir adalah memindahkan data yang digunakan pada layanan server utama, dari komputer lokal ke *cloud*. Proses ini dilakukan dengan menggunakan perintah `rsync`. Setelah semua data selesai dipindahkan, layanan siap dijalankan pada server utama menggunakan *script* memulai layanan yang telah dibuat. Pengguna hanya dapat mengakses layanan yang disediakan server utama melalui *load balancer*.

3.2.5 Perancangan Server Cadangan

Perancangan server cadangan dilakukan pada layanan *cloud* milik Amazon, yakni AWS EC2. Server cadangan harus mampu menjalankan layanan setelah menerima permintaan migrasi dari server utama dan merespon semua permintaan dari klien. Perancangan server cadangan ditunjukkan pada Gambar 3.5.

Gambar 3.5 Perancangan server cadangan

Perancangan server cadangan dimulai dengan membuat *instance* server cadangan pada AWS. Caranya adalah dengan memilih pilihan *launch instance* pada halaman EC2 *Dashboard*. Spesifikasi *instance* server cadangan yang dibuat ditunjukkan pada Tabel 3.6.

Tabel 3.6 Spesifikasi *instance* server cadangan AWS

Name	aws-server-1
Region	us-east-1 (*North* Virgiana)
Zone	us-east-1b
***Amazon Machine Image* (AMI)**	Red Hat Enterprise Linux 8 (HVM), SSD *Volume Type*
Instance type	t2.micro
Machine type	1 vCPUs, 1 GiB *memory*
CPU *platform*	Intel Xeon Family

Saat langkah terakhir pada proses pembuatan *instance* server cadangan, *generate* pasangan SSH *key* baru atau gunakan yang sudah ada, kemudian unduh ke komputer lokal. Langkah berikutnya adalah melakukan konfigurasi *security*

group pada AWS. Hal ini dilakukan untuk mempermudah pengaksesan *instance* server cadangan. Caranya adalah dengan memilih pilihan *security group* pada EC2 *dashboard*, kemudian lakukan perubahan pada *security group instance* server cadangan *rule inbound*, dengan detail ditunjukkan pada Tabel 3.7.

Tabel 3.7 *Security group instance* server cadangan AWS

Mode	*inbound*
Type	*all traffic*
Protocol	*all*
Port range	*all*
Source	0.0.0.0/0

Langkah selanjutnya adalah melakukan konfigurasi *elastic* IP untuk mendapatkan IP statis pada AWS. Pada penelitian ini, IP statis AWS yang didapatkan adalah 34.204.210.55. Setelah konfigurasi *elastic* IP selesai dilakukan, jalankan *instance* server cadangan. Lakukan *update* dan instalasi perangkat lunak yang diperlukan. Perangkat lunak yang di instal pada *instance* server cadangan beserta penjelasan kegunaannya terdapat pada Tabel 3.8.

Tabel 3.8 Perangkat lunak pada server cadangan

Perangkat Lunak	Keterangan
Python 2.7.x	Bahasa pemrograman yang digunakan dalam pembuatan *script live migration*.
Podman 1.6.4	*Container engine* yang digunakan.
CRIU 3.13	*Library* yang digunakan untuk menyimpan dan memulihkan *state* dari sebuah aplikasi.
Buildah 1.11.6	Depedensi yang diperlukan untuk menjalankan Podman.
Nano 2.9.8	*Text editor* sederhana yang digunakan untuk mempermudah proses perubahan sebuah berkas teks.
Rsync 3.1.3	Aplikasi yang digunakan untuk melakukan sinkronisasi berkas antar *host* yang berbeda.
Apache server (*latest* - Docker hub *repository*)	Digunakan sebagai webserver untuk menjalankan layanan utama.
MySQL (*latest* - Docker hub *repository*)	Digunakan untuk menyimpan data pengguna.

Setelah proses *update* dan instalasi perangkat lunak selesai dilakukan, langkah berikutnya adalah melakukan konfigurasi sshd dan selinux. Konfigurasi sshd dilakukan dengan melakukan perubahan pada berkas /etc/ssh/sshd_config,

yakni pada baris "PermitRootLogin yes" dan "PasswordAuthentication no". Konfigurasi selinux dilakukan dengan melakukan perubahan pada berkas /etc/selinux/config, dengan merubahnya menjadi "disabled". Konfigurasi selinux dapat dicek menggunakan perintah `getenforce` dan `sestatus`.

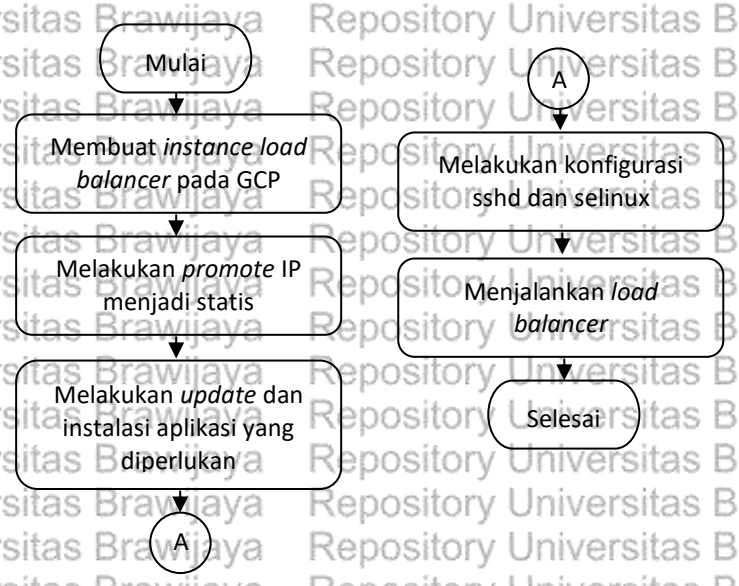
Langkah terakhir adalah melakukan uji coba layanan yang akan dimigrasikan. Pertama-tama pindahkan data yang diperlukan dari komputer lokal ke *cloud* menggunakan perintah `rsync`. Setelah semua data selesai dipindahkan, jalankan layanan menggunakan *script* yang telah dibuat. Apabila layanan berhasil dijalankan, maka layanan dapat langsung dimatikan dan server cadangan siap menerima permintaan migrasi dari server utama.

3.2.6 Perancangan Mekanisme *Live Migration*

Perancangan mekanisme *live migration* dilakukan dengan membuat *load balancer* dan *script live migration*. *Load balancer* memungkinkan pengguna untuk terhubung ke satu alamat IP saja, tanpa perlu tahu keadaan sebenarnya dibalik proses *live migration* yang terjadi. *Script live migration* dibuat untuk mengotomatisasikan langkah-langkah melakukan *live migration*.

3.2.6.1 Perancangan *Load Balancer*

Perancangan *load balancer* dilakukan pada layanan *cloud* milik Google, yakni GCP. *Load balancer* harus mampu mengarahkan permintaan dari pengguna ke server yang memiliki layanan. Perancangan *load balancer* ditunjukkan pada Gambar 3.6.



Gambar 3.6 Perancangan *load balancer*

Pada perancangan *load balancer*, langkah pembuatan dan penambahan SSH *key* di projek pada GCP tidak perlu dilakukan. Begitu juga dengan langkah penambahan *rule firewall* dev-allow-all. Ketiga langkah tersebut sudah dilakukan saat perancangan server utama.

Perancangan *load balancer* dimulai dengan membuat *instance load balancer* pada GCP. Caranya adalah dengan masuk ke pilihan *compute engine* -> VM *instance* -> *create instance*. Spesifikasi dari *instance load balancer* yang dibuat ditunjukkan pada Tabel 3.9.

Tabel 3.9 Spesifikasi *instance load balancer GCP*

Name	gcp-lb-1
Region	us-east4 (*North* Virgiana)
Zone	us-east4c
Machine series	N1
Machine type	g1-small (1 vCPUs, 1.7 GB *memory*)
CPU *platform*	Intel Skylake++
Boot disk	CentOS 8 x64 (10 GB)
Firewall tags	allow-all-traffic

Langkah selanjutnya adalah melakukan *promote* alamat IP agar menjadi statis. Caranya adalah dengan masuk ke pilihan VPC *network* -> *external* IP *address* -> *reserve static address*. Pada penelitian ini, IP statis yang didapatkan adalah 35.245.113.158. Setelah *promote* IP selesai dilakukan, jalankan *instance load balancer*. Lakukan *update* dan instalasi perangkat lunak yang diperlukan. Perangkat lunak yang di instal pada *instance load balancer* beserta penjelasan kegunaannya terdapat pada Tabel 3.10.

Tabel 3.10 Perangkat lunak pada *load balancer*

Perangkat Lunak	Keterangan
Python 2.7.x	Bahasa pemrograman yang digunakan dalam pembuatan *script live migration*.
Podman 1.6.4	*Container engine* yang digunakan.
Buildah 1.11.6	Depedensi yang diperlukan untuk menjalankan Podman.
Nano 2.9.8	*Text editor* sederhana yang digunakan untuk mempermudah proses perubahan sebuah berkas teks.
NGINX (*latest* - Docker hub *repository)*	Digunakan sebagai *loadbalancer* yang diletakkan pada GCP.

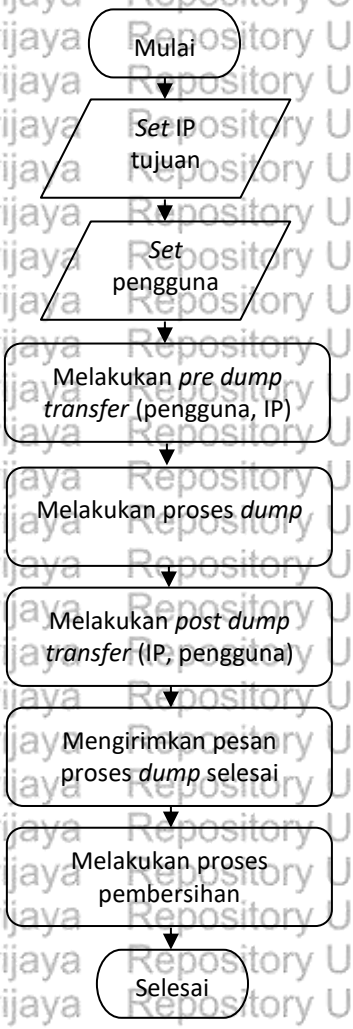
Setelah proses *update* dan instalasi perangkat lunak selesai dilakukan, langkah berikutnya adalah melakukan konfigurasi sshd dan selinux. Konfigurasi sshd dilakukan dengan melakukan perubahan pada berkas /etc/ssh/sshd_config, yakni pada baris "PermitRootLogin yes" dan "PasswordAuthentication no". Konfigurasi selinux dilakukan dengan melakukan perubahan pada berkas

/etc/selinux/config, dengan merubahnya menjadi "disabled". Konfigurasi selinux dapat dicek menggunakan perintah `getenforce` dan `sestatus`.

Langkah terakhir adalah membuat berkas konfigurasi *load balancer* yang dibuat dengan nama loadbalance.conf. Isi dari berkas loadbalance.conf adalah konfigurasi alamat IP yang dijadikan sebagai acuan dalam melakukan proses *load balance*. IP yang digunakan adalah IP dari server utama yakni 35.236.204.230 dan IP dari server cadangan yaitu 34.204.210.55. Setelah berkas konfigurasi selesai dibuat, *load balancer* dapat dijalankan menggunakan *script* yang telah dibuat.

3.2.6.2 Perancangan *Script Live Migration*

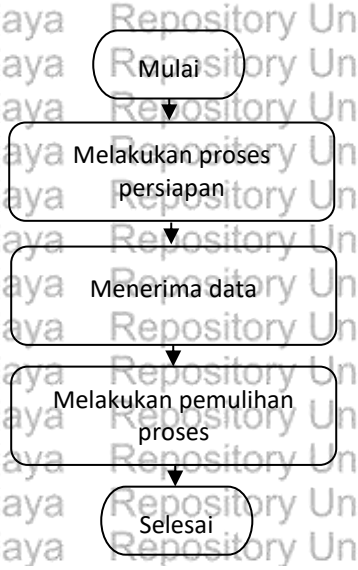
Script live migration terdiri dari dua jenis, yaitu *script* untuk mengirimkan permintaan *live migration* dan *script* untuk menerima permintaan *live migration*. *Script live migration* dikembangkan dan dijalankan menggunakan Python 2.7.x. Perancangan *script* permintaan *live migration* ditunjukkan pada Gambar 3.7.



Gambar 3.7 Rancangan *script* permintaan *live migration*

Proses permintaan *live migration* dimulai dengan menentukan alamat IP tujuan dan pengguna yang digunakan. Alamat IP disesuaikan dengan *instance* tujuan. Apabila migrasi dilakukan dari server utama ke server cadangan, maka alamat IP tujuannya adalah 34.204.210.55. Sebaliknya, jika migrasi dilakukan dari server cadangan ke server utama, maka alamat IP tujuannya adalah 35.236.204.230. Pengguna yang digunakan untuk menjalankan *script live migration* adalah pengguna root, karena akses root dibutuhkan untuk memigrasikan suatu proses dari satu *instance* ke *instance* lain. Langkah selanjutnya adalah *pre dump transfer*. *Pre dump transfer* merupakan proses pemindahan data yang diperlukan dari *instance* sumber ke *instance* tujuan menggunakan `rsync`. Setelah semua data yang diperlukan selesai dipindahkan, langkah berikutnya adalah *dump*. *Dump* merupakan proses pengompresan proses aplikasi menjadi berkas dengan menggunakan CRIU. Setelah proses *dump* selesai berjalan, dilakukan proses *post dump transfer*. *Post dump transfer* merupakan proses pemindahan berkas hasil *dump* dan data-data yang mengalami perubahan, dari *instance* sumber ke *instance* tujuan menggunakan `rsync`. Setelah berkas hasil *dump* dan data yang berubah terkirim, *instance* sumber mengirimkan pesan "OK" kepada *instance* tujuan, kemudian melakukan proses pembersihan.

Script penerimaan permintaan *live migration* dijalankan pada *instance* tujuan. *Script* ini harus dijalankan terlebih dahulu sebelum *script* permintaan *live migration*. Perancangan *script* penerimaan permintaan *live migration* ditunjukkan pada Gambar 3.8.



Gambar 3.8 Rancangan *script* penerimaan *live migration*

Proses penerimaan *live migration* dimulai dengan proses persiapan. Pada proses persiapan, dilakukan pembersihan pada direktori tempat data dan berkas hasil *dump* diletakkan. Setelah proses persiapan selesai dilakukan, *instance* tujuan siap menerima data dan berkas hasil *dump* dari *instance* sumber. Setelah semua data, berkas hasil *dump*, dan pesan "OK" diterima, *instance* tujuan melakukan pemulihan proses menggunakan CRIU.

3.3 Metode Evaluasi

Metode evaluasi berisi parameter dan skenario uji dari penelitian implementasi *container live migration* antar-*cloud provider* menggunakan Podman dan CRIU. Pada penelitian ini, metode evaluasi terdiri dari tiga bagian, yaitu evaluasi pengujian fungsional, *downtime*, dan *migration time*.

3.3.1 Pengujian Fungsional

Pengujian fungsional dilakukan untuk mengetahui apakah sistem yang dibangun telah memenuhi seluruh kebutuhan fungsional yang ditetapkan. Pengujian berikutnya hanya dapat dilakukan apabila semua kebutuhan fungsional telah terpenuhi. Apabila terdapat kebutuhan fungsional yang belum terpenuhi, maka dilakukan pengecekan kembali terhadap rancangan dan kode sistem, kemudian dilakukan pengujian ulang hingga semua kebutuhan terpenuhi. Skenario pengujian fungsional dari sistem yang dikembangkan dapat dilihat pada Tabel 3.11.

Tabel 3.11 Skenario pengujian fungsional

Kode	Kebutuhan Fungsional	Skenario
LMF_01	*Load balancer* dapat mengarahkan *traffic* permintaan dari klien menuju server yang memiliki layanan.	1. Layanan *load balancer* sudah berjalan. 2. Layanan pada server utama atau cadangan sudah berjalan. 3. Pengguna mengakses alamat IP *instance load balancer*. 4. Pengguna dapat mengakses layanan yang disediakan oleh server utama atau cadangan.
LMF_02	Server yang memiliki layanan mampu merespon permintaan dari klien.	1. Layanan *load balancer* sudah berjalan. 2. Layanan pada server utama atau cadangan sudah berjalan. 3. Pengguna mengakses alamat IP *instance load balancer*. 4. Pengguna dapat mengakses layanan yang disediakan oleh server utama atau cadangan.
LMF_03	Server yang memiliki layanan mampu memigrasikan layanannya ke server lainnya.	1. Layanan *load balancer* sudah berjalan. 2. Layanan pada server utama atau cadangan sudah berjalan. 3. Server yang tidak memiliki layanan menjalankan *script* menerima permintaan migrasi. 4. Server yang memiliki layanan mengirimkan permintaan migrasi. 5. Server tujuan berhasil menerima dan menjalankan layanan yang dimigrasikan. 6. Pengguna mengakses alamat IP *instance load balancer*. 7. Pengguna dapat mengakses layanan yang disediakan oleh server tujuan.

Tabel 3.11 (lanjutan)

Kode	Kebutuhan Fungsional	Skenario
LMF_04	Server yang memiliki layanan mampu menjaga keseluruhan *state* dari layanan sesudah proses *live migration* dilakukan.	1. Layanan *load balancer* sudah berjalan. 2. Layanan pada server utama atau cadangan sudah berjalan. 3. Server yang tidak memiliki layanan menjalankan *script* menerima permintaan migrasi. 4. Server yang memiliki layanan mengirimkan permintaan migrasi. 5. Server tujuan berhasil menerima dan menjalankan layanan yang dimigrasikan. 6. Pengguna mengakses alamat IP *instance load balancer*. 8. Pengguna dapat mengakses layanan yang disediakan oleh server tujuan, dengan *session* yang masih terjaga.

3.3.2 Pengujian *Downtime*

Pengujian *downtime* berkaitan dengan durasi layanan dihentikan sepenuhnya pada saat proses migrasi berlangsung. Selama periode ini, layanan tidak dapat diakses oleh pengguna, sehingga berpengaruh pada *Service Level Agreement* (SLA) dari layanan tersebut. Pada sistem yang dibangun, *downtime* mulai dihitung saat proses *dump* dilakukan pada *instance* sumber, hingga proses *restore* selesai dilakukan pada *instance* tujuan. Untuk melakukan pengujian *downtime*, pada *script* yang dibuat telah disisipkan kode untuk menampilkan waktu dan *timestamp* dari setiap proses. Satuan waktu yang digunakan adalah detik. Pengujian *downtime* dilakukan dengan banyak data 100, 500, 1000, 5000, dan 10000, di 3 lingkungan pengujian berbeda, sebanyak 10 kali. Skenario uji ini diambil dan disesuaikan dari penelitian milik Govindaraj & Artemenko (2018). Data yang digunakan merupakan data pengguna yang tersimpan pada MySQL. Tahapan pengujian *downtime* dari sistem yang dikembangkan ditunjukkan pada Tabel 3.12.

Tabel 3.12 Tahapan pengujian *downtime*

Kode	Tahapan	Keterangan
LMD_01	Layanan *load balancer* sudah berjalan.	Layanan NGINX telah berjalan di *instance load balancer* pada GCP menggunakan Podman.
LMD_02	Layanan pada server utama atau cadangan sudah berjalan.	Layanan web *video streaming* dan MySQL telah berjalan di salah satu server menggunakan Podman.
LMD_03	Pengguna mengakses alamat IP *instance load balancer*.	Pengguna mengakses alamat IP 35.245.113.158 menggunakan aplikasi web *browser*.
LMD_04	Pengguna dapat mengakses layanan yang disediakan oleh server utama atau cadangan.	Pengguna dapat melihat halaman awal dari layanan yang disediakan menggunakan aplikasi web *browser*.

Tabel 3.12 (lanjutan)

Kode	Tahapan	Keterangan
LMD_05	Pengguna melakukan *login* pada layanan yang disediakan.	Pengguna melakukan *login* dengan memasukkan *username* dan *password* dan diarahkan ke halaman utama layanan.
LMD_06	Server yang tidak memiliki layanan menjalankan *script* menerima permintaan migrasi.	Server yang tidak memiliki layanan menjalankan *script* penerimaan permintaan migrasi menggunakan Python 2.7.x.
LMD_07	Server yang memiliki layanan mengirimkan permintaan migrasi.	Server yang memiliki layanan menjalankan *script* permintaan migrasi menggunakan Python 2.7.x.
LMD_08	Server tujuan berhasil menerima dan menjalankan layanan yang dimigrasikan.	Layanan berhasil dipindah dan dijalankan di server tujuan.
LMD_09	Pengguna dapat mengakses layanan yang disediakan oleh server tujuan, dengan *session* yang masih terjaga.	Pengguna melakukan *refresh* pada halaman web yang dibuka dan masih dalam keadaan *login*.
LMD_10	Dilakukan pencatatan *downtime* yang terjadi.	Perhitungan *downtime* dilakukan dengan melakukan pengurangan variabel UP_TIMESTAMP yang terdapat pada server tujuan, dengan variabel DOWN_TIMESTAMP yang terdapat pada server sumber.

3.3.3 Pengujian *Migration Time*

Pengujian *migration time* merupakan perhitungan total durasi mulai dari proses migrasi dilakukan pada *instance* sumber, hingga layanan berhasil dipulihkan dan berjalan pada *instance* tujuan. Pada sistem yang dibangun, *migration time* mulai dihitung saat proses *pre dump transfer* dilakukan pada *instance* sumber, hingga proses *restore* selesai dilakukan pada *instance* tujuan. Untuk melakukan pengujian *migration time*, pada *script* yang dibuat telah disisipkan kode untuk menampilkan waktu dan *timestamp* dari setiap proses. Satuan waktu yang digunakan adalah detik. Pengujian *migration time* dilakukan dengan banyak data 100, 500, 1000, 5000, dan 10000, di 3 lingkungan pengujian berbeda, sebanyak 10 kali. Skenario uji ini diambil dan disesuaikan dari penelitian milik Govindaraj & Artemenko (2018). Data yang digunakan merupakan data pengguna yang tersimpan di MySQL. Tahapan pengujian *migration time* dari sistem yang dikembangkan dapat dilihat pada Tabel 3.13.

Tabel 3.13 Tahapan pengujian *migration time*

Kode	Tahapan	Keterangan
LMM_01	Layanan *load balancer* sudah berjalan.	Layanan NGINX telah berjalan di *instance load balancer* pada GCP menggunakan Podman.
LMM_02	Layanan pada server utama atau cadangan sudah berjalan.	Layanan web *video streaming* dan MySQL telah berjalan di salah satu server menggunakan Podman.

Tabel 3.14 (lanjutan)

Kode	Tahapan	Keterangan
LMM_03	Pengguna mengakses alamat IP *instance load balancer*.	Pengguna mengakses alamat IP 35.245.113.158 menggunakan aplikasi web *browser*.
LMM_04	Pengguna dapat mengakses layanan yang disediakan oleh server utama atau cadangan.	Pengguna dapat melihat halaman awal dari layanan yang disediakan menggunakan aplikasi web *browser*.
LMM_05	Pengguna melakukan *login* pada layanan yang disediakan.	Pengguna melakukan *login* dengan memasukkan *username* dan *password* dan diarahkan ke halaman utama layanan.
LMM_06	Server yang tidak memiliki layanan menjalankan *script* menerima permintaan migrasi.	Server yang tidak memiliki layanan menjalankan *script* penerimaan permintaan migrasi menggunakan Python 2.7.x.
LMM_07	Server yang memiliki layanan mengirimkan permintaan migrasi.	Server yang memiliki layanan menjalankan *script* permintaan migrasi menggunakan Python 2.7.x.
LMM_08	Server tujuan berhasil menerima dan menjalankan layanan yang dimigrasikan.	Layanan berhasil dipindah dan dijalankan di server tujuan.
LMM_09	Pengguna dapat mengakses layanan yang disediakan oleh server tujuan, dengan *session* yang masih terjaga.	Pengguna melakukan *refresh* pada halaman web yang dibuka dan masih dalam keadaan *login*.
LMM_10	Dilakukan pencatatan *migration time* yang terjadi.	Perhitungan *migration time* dilakukan dengan melakukan penjumlahan total waktu proses pada server sumber dan tujuan.

BAB 4 IMPLEMENTASI

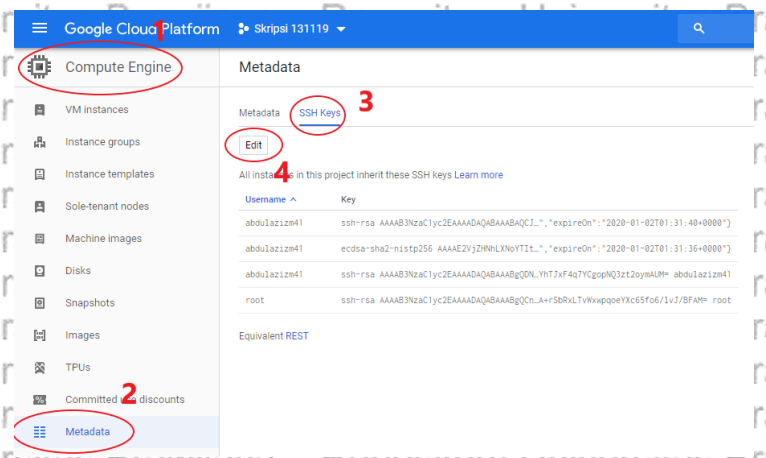
Bab ini memuat penjelasan keseluruhan proses implementasi *container live migration* antar-*cloud provider* menggunakan Podman dan CRIU yang berdasar pada perancangan sistem dan lingkungan implementasi yang telah dibuat. Implementasi pada penelitian ini terdiri dari implementasi server utama, server cadangan, *load balancer*, dan *script live migration*. *Script live migration* terdiri dari dua jenis, yakni *script* permintaan *live migration* dan *script* penerimaan *live migration*.

4.1 Implementasi Server Utama

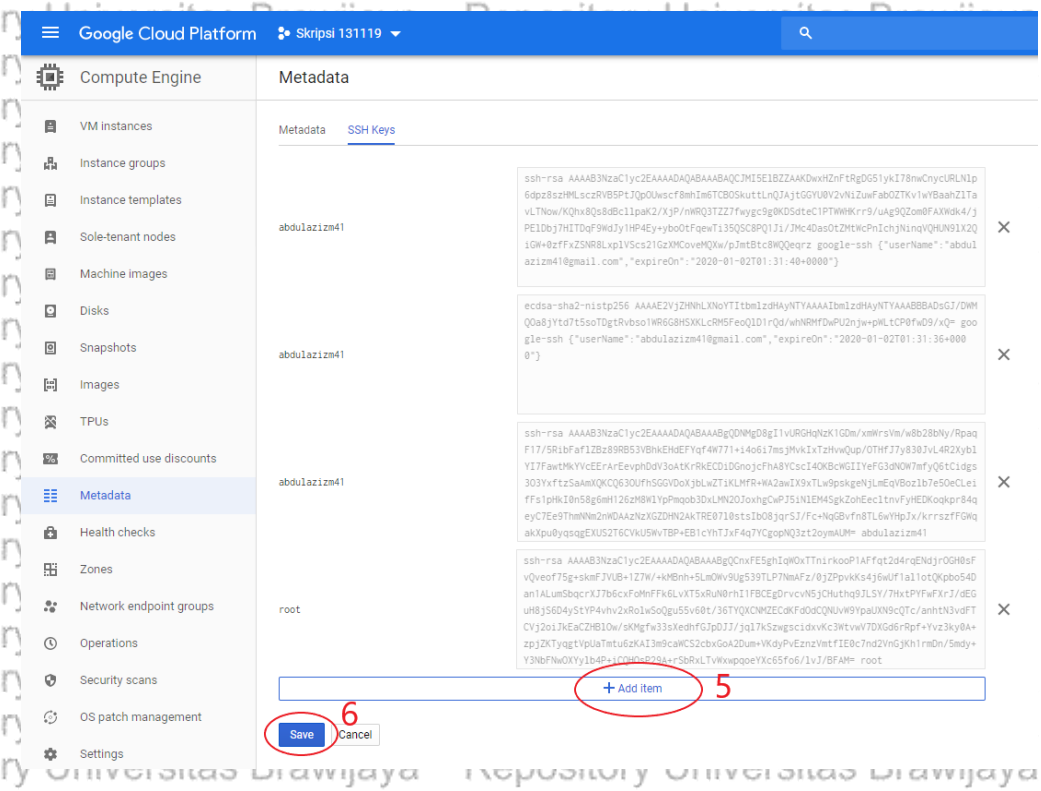
Server utama merupakan server yang terdapat pada layanan *cloud* milik Google yaitu GCP. Server utama menjalankan layanan *video streaming* dan MySQL. Pengguna hanya dapat mengakses layanan *video streaming* setelah melakukan *login*. Implementasi server utama dilakukan sesuai dengan rancangan yang telah dibuat pada Subbab 3.2.4 Perancangan Server Utama, dengan penjelasan setiap langkah sebagai berikut.

4.1.1 Pembuatan Pasangan SSH *Key*

Implementasi server utama dimulai dengan membuat pasangan SSH *key* pada komputer lokal agar dapat terhubung ke server utama pada *cloud*. SSH *key* yang dibuat ada dua pasang, yang pertama adalah SSH *key* untuk pengguna biasa dan kedua adalah SSH *key* untuk pengguna root. SSH *key* untuk pengguna biasa dibuat menggunakan perintah `ssh-keygen -t rsa -f ~/.ssh/gcp-user-1 -C abdulazizm41` dan untuk pengguna root menggunakan perintah `ssh-keygen -t rsa -f ~/.ssh/gcp-root-1 -C root`. abdulazizm41 merupakan nama pengguna yang digunakan pada *instance* server utama, dan pengguna root diperlukan dalam proses *live migration*. Langkah berikutnya adalah menambahkan *public key* yang telah dibuat pada projek di GCP. Caranya ditunjukkan pada Gambar 4.1 dan 4.2.



Gambar 4.1 Menambahkan *public key* di GCP

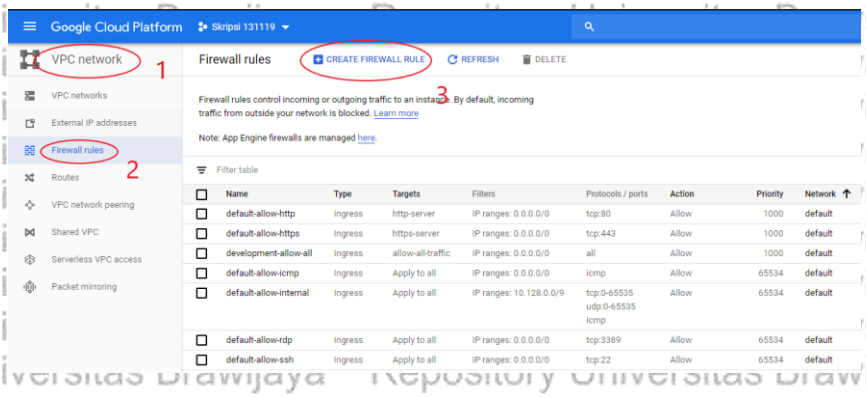


Gambar 4.2 Menambahkan *public key* di GCP (lanjutan)

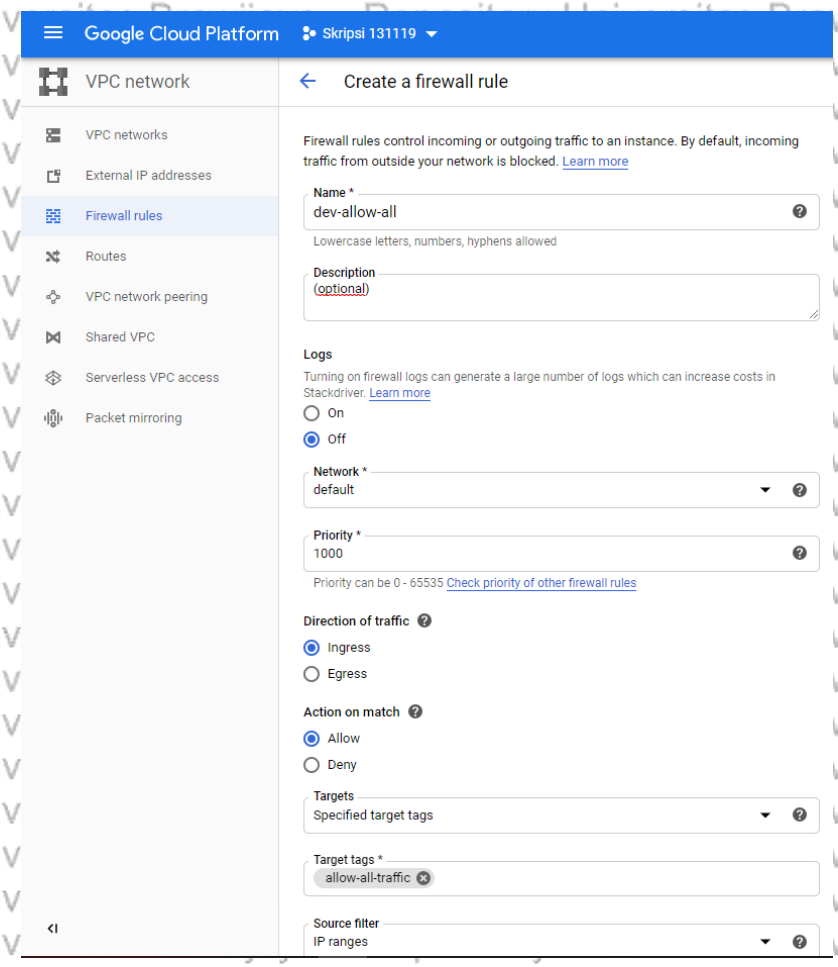
Langkah pertama adalah masuk ke pilihan *compute engine*, kemudian pilih *metadata*. Setelah itu pilih tab SSH *key* dan tekan tombol *edit*. Pada halaman *edit*, tekan tombol *add item*, kemudian masukkan *public key* untuk pengguna biasa dan *public key* untuk pengguna root yang telah dibuat sebelumnya. Setelah selesai, tekan tombol *save*.

4.1.2 Konfigurasi *Firewall* Pada GCP

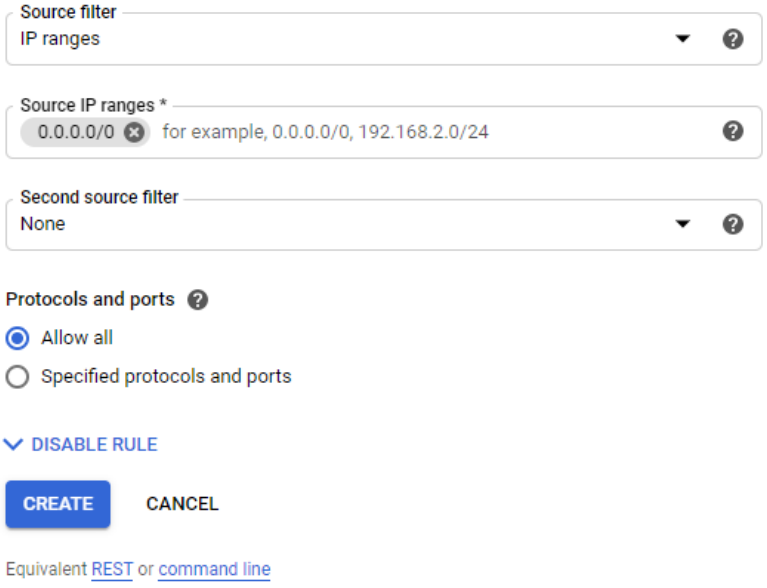
Langkah berikutnya dalam implementasi server utama adalah melakukan konfigurasi *firewall* pada GCP. Untuk mempermudah pengaksesan *instance* server utama, ditambahkan sebuah *rule firewall* bernama dev-allow-all yang mengizinkan semua akses masuk ke *instance* server utama. Langkah-langkahnya ditunjukkan pada Gambar 4.3, 4.4, dan 4.5.



Gambar 4.3 Konfigurasi *firewall* pada GCP



Gambar 4.4 Konfigurasi *firewall* pada GCP (lanjutan)

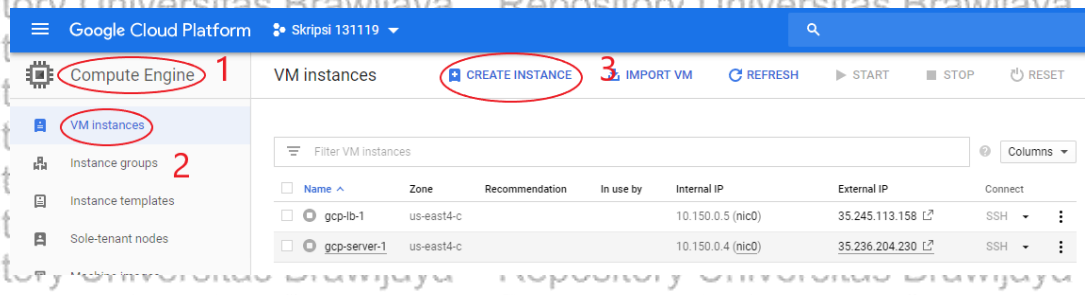


Gambar 4.5 Konfigurasi *firewall* pada GCP (lanjutan)

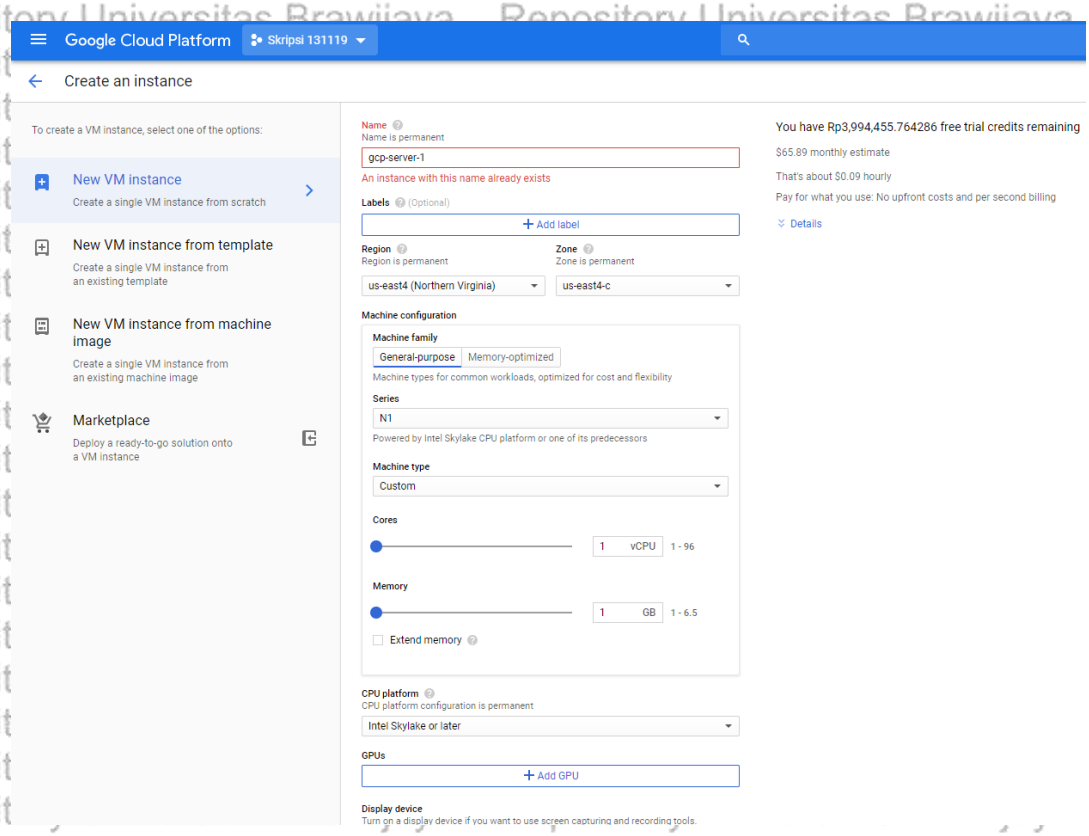
Langkah pertama adalah masuk ke pilihan VPC *network*, kemudian pilih *firewall rule* dan tekan tombol *create firewall rule*. Pada halaman *create firewall rule*, masukkan konfigurasi *firewall* sesuai dengan Tabel 3.3 yang terdapat pada Subbab 3.2.4 Perancangan Server Utama. Setelah konfigurasi *firewall* selesai dilakukan, tekan tombol *create* yang terdapat di bagian paling bawah halaman *create firewall rule*.

4.1.3 Pembuatan *Instance* Server Utama

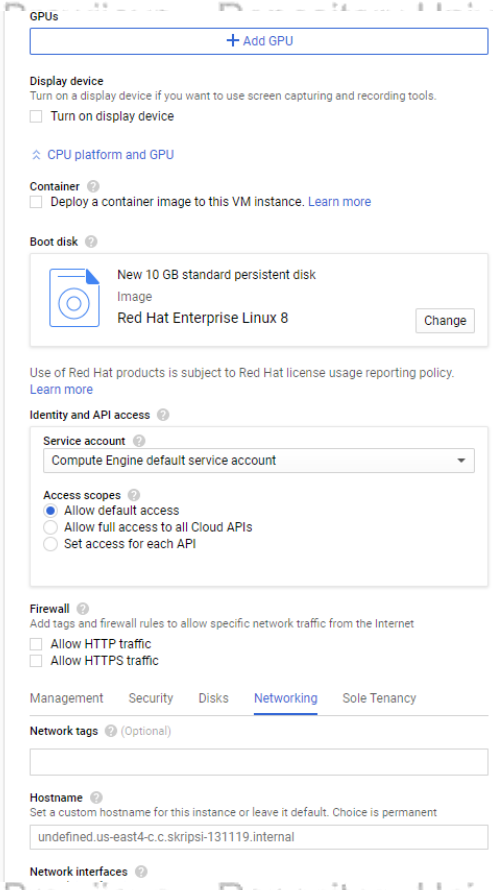
Langkah selanjutnya dalam implementasi server utama adalah membuat *instance* server utama pada GCP. Langkah-langkah pembuatan *instance* server utama ditunjukkan pada Gambar 4.6, 4.7, 4.8, dan 4.9.



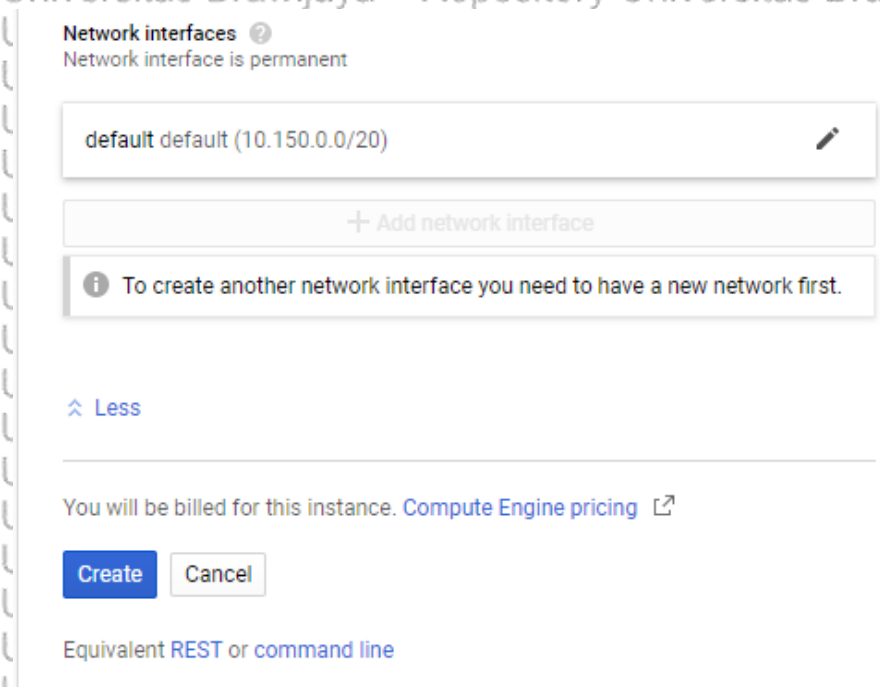
Gambar 4.6 Pembuatan *instance* server utama



Gambar 4.7 Pembuatan *instance* server utama (lanjutan)



Gambar 4.8 Pembuatan *instance* server utama (lanjutan)

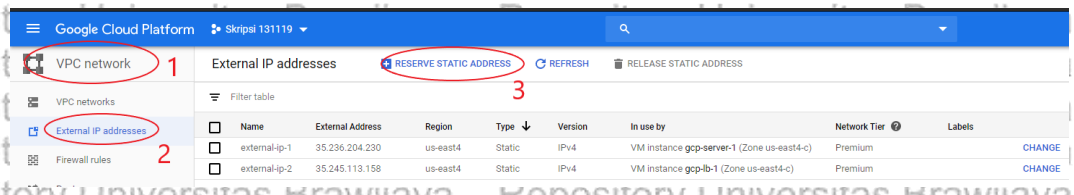


Gambar 4.9 Pembuatan *instance* server utama (lanjutan)

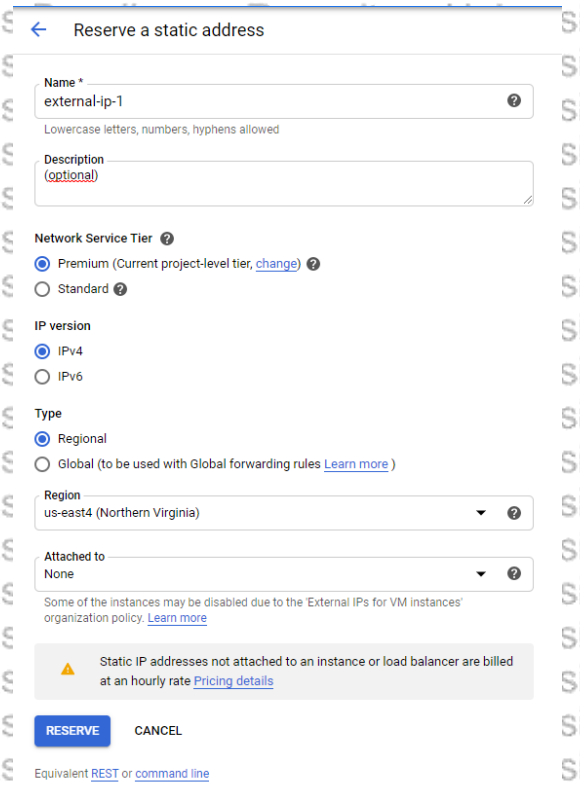
Pembuatan *instance* server utama dimulai dengan masuk ke pilihan *compute engine*, kemudian pilih VM *instance* dan klik tombol *create instance*. Pada halaman *create instance*, masukkan spesifikasi *instance* sesuai dengan Tabel 3.4 yang terdapat pada Subbab 3.2.4 Perancangan Server Utama. Pada bagian *networking*, masukkan *network tag* sesuai dengan *firewall tag* yang sudah dibuat sebelumnya, yaitu allow-all-traffic. Setelah konfigurasi selesai dilakukan, tekan tombol *create* yang terletak di bagian paling bawah halaman *create instance*.

4.1.4 Konfigurasi IP Statis Pada GCP

Setelah pembuatan *instance* server utama selesai dilakukan, langkah berikutnya adalah melakukan konfigurasi IP statis pada GCP. Hal ini dilakukan agar *load balancer* dapat mengarahkan *traffic* pengguna ke alamat IP yang konsisten. Langkah-langkah konfigurasi IP statis pada GCP ditunjukkan pada Gambar 4.10 dan 4.11.



Gambar 4.10 Konfigurasi IP statis pada GCP



Gambar 4.11 Konfigurasi IP statis pada GCP (lanjutan)

Langkah pertama dalam melakukan konfigurasi IP statis pada GCP adalah dengan masuk ke pilihan VPC *network*, kemudian pilih *external* IP *address* dan tekan pilihan *reserve static address*. Pada halaman *reserve static address*, masukkan konfigurasi IP statis sesuai dengan Tabel 4.1.

Tabel 4.1 Konfigurasi IP statis pada GCP

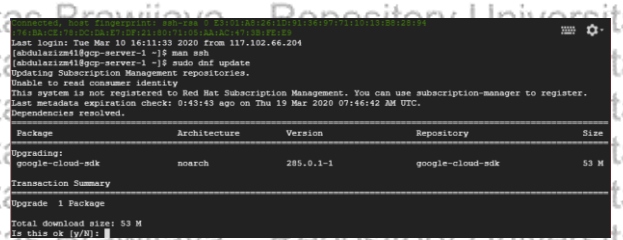
Nama	external-ip-1
Deskripsi	(opsional)
Network service tier	Premium
Versi IP	IPv4
Tipe	*Regional*
Wilayah	us-east-4 (*Northern* Virgiana)
Terhubung ke	gcp-server-1

Alamat IP statis dipilihkan secara otomatis oleh Google. Dalam penelitian ini, IP statis yang didapatkan adalah 35.236.204.230. Setelah konfigurasi selesai dilakukan, tekan tombol *reserve* yang terletak di bagian paling bawah halaman *reserve static address*.

4.1.5 Melakukan *Update* Dan Instalasi Aplikasi

Setelah konfigurasi IP statis pada GCP selesai dilakukan, langkah selanjutnya adalah melakukan *update* dan instalasi aplikasi yang diperlukan oleh *instance* server utama. Pertama-tama masuk ke *instance* server utama melalui SSH dengan menggunakan perintah `ssh -i /home/abdulazizm41/.ssh/gcp-user-1 abdulazizm41@35.236.204.230`. Opsi `-i` adalah singkatan dari *identity_file* yang merupakan *private key* yang digunakan untuk masuk pada *instance* server utama. `/home/abdulazizm41/.ssh/` merupakan direktori letak *private key* pada komputer lokal dan `gcp-user-1` merupakan nama *private key* yang digunakan.

Setelah masuk ke *instance* server utama, dilakukan proses *update* dengan menggunakan perintah `sudo dnf update -y`. *Dandified* YUM (DNF) merupakan *package manager* yang terdapat pada distro linux RHEL dan turunannya. Opsi `-y` merupakan singkatan dari *yes* yang digunakan untuk menjalankan sebuah perintah tanpa tanggapan dari pengguna. Proses *update* ditunjukkan pada Gambar 4.12.



Gambar 4.12 Proses *update instance* server utama

Setelah proses *update* selesai, dilanjutkan dengan proses instalasi aplikasi yang diperlukan. Aplikasi-aplikasi yang diperlukan oleh *instance* server utama dapat dilihat pada Tabel 3.5 yang terdapat pada Subbab 3.2.4 Perancangan Server Utama. Perintah yang digunakan dan proses instalasi aplikasi ditunjukkan pada Gambar 4.13.

```
[abdulazizm41@gcp-server-1 ~]$ sudo dnf install buildah criu nano podman python2 rsync -y
Updating Subscription Management repositories.
Unable to read consumer identity
This system is not registered to Red Hat Subscription Management. You can use subscription-manager to register.
Last metadata expiration check: 1:23:54 ago on Thu 19 Mar 2020 07:46:42 AM UTC.
Package buildah-1.11.6-4.module+el8.1.1+5259+bcdd613a.x86_64 is already installed.
Package criu-3.12-9.el8.x86_64 is already installed.
Package nano-2.9.8-1.el8.x86_64 is already installed.
Package podman-1.6.4-2.module+el8.1.1+5363+bf8ff1af.x86_64 is already installed.
Package python2-2.7.16-12.module+el8.1.0+4148+33a50073.x86_64 is already installed.
Package rsync-3.1.3-6.el8.x86_64 is already installed.
Dependencies resolved.
Nothing to do.
Complete!
[abdulazizm41@gcp-server-1 ~]$
```

Gambar 4.13 Proses instalasi aplikasi *instance* server utama

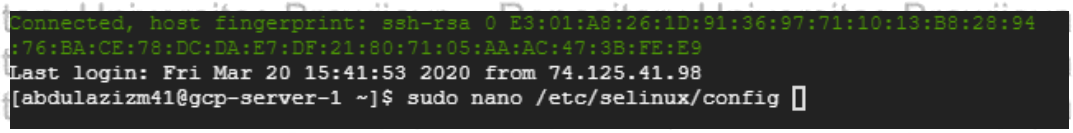
Setelah proses instalasi aplikasi yang diperlukan selesai, dilanjutkan dengan mengecek apakah setiap aplikasi sudah terinstal dengan benar. Caranya adalah dengan mengecek versi dari masing-masing aplikasi seperti yang ditunjukkan pada Gambar 4.14.

```
[abdulazizm41@gcp-server-1 ~]$ podman --version
podman version 1.6.4
[abdulazizm41@gcp-server-1 ~]$ criu --version
Version: 3.12
[abdulazizm41@gcp-server-1 ~]$ python --version
-bash: python: command not found
[abdulazizm41@gcp-server-1 ~]$ python2 --version
Python 2.7.16
[abdulazizm41@gcp-server-1 ~]$ nano --version
 GNU nano, version 2.9.8
 (C) 1999-2011, 2013-2018 Free Software Foundation, Inc.
 (C) 2014-2018 the contributors to nano
 Email: nano@nano-editor.org    Web: https://nano-editor.org/
 Compiled options: --enable-utf8
[abdulazizm41@gcp-server-1 ~]$ buildah --version
buildah version 1.11.6 (image-spec 1.0.1-dev, runtime-spec 1.0.1-dev)
[abdulazizm41@gcp-server-1 ~]$ rsync --version
rsync  version 3.1.3  protocol version 31
Copyright (C) 1996-2018 by Andrew Tridgell, Wayne Davison, and others.
Web site: http://rsync.samba.org/
Capabilities:
    64-bit files, 64-bit inums, 64-bit timestamps, 64-bit long ints,
    socketpairs, hardlinks, symlinks, IPv6, batchfiles, inplace,
    append, ACLs, xattrs, iconv, symtimes, prealloc

rsync comes with ABSOLUTELY NO WARRANTY.  This is free software, and you
are welcome to redistribute it under certain conditions.  See the GNU
General Public Licence for details.
[abdulazizm41@gcp-server-1 ~]$
```

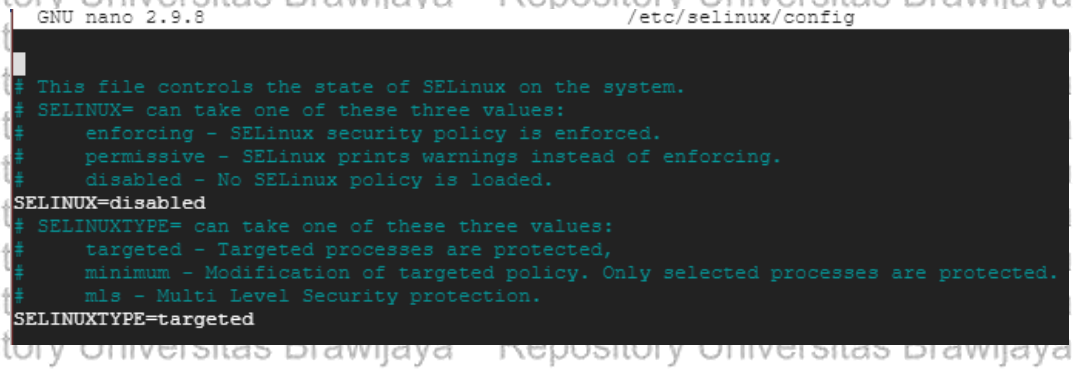
Gambar 4.14 Versi dari setiap aplikasi yang diinstal

4.1.6 Konfigurasi SSHD Dan Selinux

Langkah berikutnya dalam implementasi server utama adalah melakukan konfigurasi SSHD dan selinux. Konfigurasi selinux dibuat menjadi *disabled* untuk mempermudah proses *live migration*. Konfigurasi selinux dilakukan dengan melakukan perubahan pada berkas `/etc/selinux/config`. Proses konfigurasi selinux ditunjukkan pada Gambar 4.15 dan 4.16.

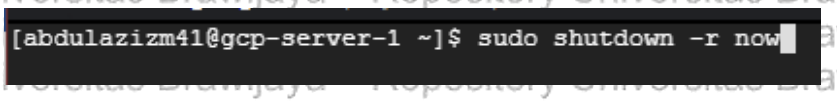


Gambar 4.15 Konfigurasi selinux

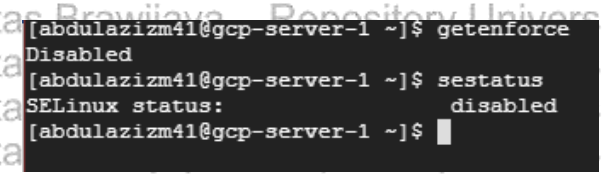


Gambar 4.16 Konfigurasi selinux (lanjutan)

Setelah konfigurasi selinux selesai dilakukan, nyalakan ulang komputer agar konfigurasi dapat dibaca oleh sistem. Setelah itu, cek menggunakan perintah `getenforce` dan/atau `sestatus` seperti yang ditunjukkan pada Gambar 4.17 dan 4.18. Apabila muncul bacaan *disabled*, maka konfigurasi selinux selesai dilakukan.



Gambar 4.17 Menyalakan ulang komputer



Gambar 4.18 Mengecek status selinux

Langkah selanjutnya adalah melakukan konfigurasi SSHD. Konfigurasi SSHD dilakukan karena tiga alasan, yaitu:

- (Opsional) Agar koneksi dari klien SSH ke server SSH tidak sering terputus akibat tidak ada permintaan dari klien SSH dalam jangka waktu tertentu.
- Mengizinkan login sebagai pengguna root melalui SSH. Hal ini diperlukan untuk proses *live migration*.
- Menonaktifkan autentikasi sandi. Hal ini diperlukan agar klien SSH dapat terhubung ke server SSH hanya dengan menggunakan pasangan SSH *key* yang telah dibuat.

Konfigurasi SSHD dilakukan dengan melakukan perubahan pada berkas `/etc/ssh/sshd_config`. Parameter-parameter yang diubah beserta keterangan setiap parameter dijelaskan sebagai berikut:

- **ClientAliveInterval** 60
  Server akan menunggu selama 60 detik sebelum mengirimkan *null packet* kepada klien untuk menjaga koneksi tetap aktif.
- **TCPKeepAlive** yes
  Untuk memastikan bahwa tidak ada *rule firewall* tertentu yang menghapus koneksi *idle*.
- **ClientAliveCountMax** 10000
  Server mengirimkan pesan kepada klien, walaupun server tidak menerima pesan apapun dari klien.
- **PermitRootLogin** yes
  Mengizinkan login sebagai pengguna root melalui SSH.
- **PasswordAuthentication** no
  Menonaktifkan autentikasi sandi.

Proses konfigurasi SSHD ditunjukkan pada Gambar 4.19, 4.20, dan 4.21.

```
[abdulazizm41@gcp-server-1 ~]$ sudo nano /etc/ssh/sshd_config
```

Gambar 4.19 Konfigurasi SSHD

```
#LoginGraceTime 2m
PermitRootLogin yes
#StrictModes yes
#MaxAuthTries 6
#MaxSessions 10

#PubkeyAuthentication yes

# The default is to check both .ssh/authorized_keys and .ssh/authorized_keys2
# but this is overridden so installations will only check .ssh/authorized_keys
AuthorizedKeysFile      .ssh/authorized_keys

#AuthorizedPrincipalsFile none

#AuthorizedKeysCommand none
#AuthorizedKeysCommandUser nobody

# For this to work you will also need host keys in /etc/ssh/ssh_known_hosts
#HostbasedAuthentication no
# Change to yes if you don't trust ~/.ssh/known_hosts for
# HostbasedAuthentication
#IgnoreUserKnownHosts no
# Don't read the user's ~/.rhosts and ~/.shosts files
#IgnoreRhosts yes

# To disable tunneled clear text passwords, change to no here!
#PasswordAuthentication yes
#PermitEmptyPasswords no
PasswordAuthentication no
```

Gambar 4.20 Konfigurasi SSHD (lanjutan)

```
#PrintLastLog yes
TCPKeepAlive yes
#PermitUserEnvironment no
#Compression delayed
ClientAliveInterval 60
ClientAliveCountMax 10000
#ShowPatchLevel no
#UseDNS no
#PidFile /var/run/sshd.pid
#MaxStartups 10:30:100
#PermitTunnel no
#ChrootDirectory none
#VersionAddendum none
```

Gambar 4.21 Konfigurasi SSHD (lanjutan)

Setelah proses konfigurasi SSHD selesai dilakukan, nyalakan ulang layanan SSHD dengan perintah `sudo systemctl restart sshd.service`.

4.1.7 Pemindahan Data Ke *Cloud*

Pemindahan data dilakukan dengan menggunakan utilitas rsync. Data yang dipindahkan merupakan data yang diperlukan untuk menjalankan layanan pada *instance* server utama, yakni data web, video, dan beberapa *script* yang diperlukan. Perintah yang digunakan dalam proses pemindahan data dapat dilihat pada Tabel 4.2.

Tabel 4.2 Perintah memindahkan data ke *cloud*

Data web dan video:

```
sudo rsync -avz /home/abdulazizm41/Downloads/shared/docker-compose-lamp/
-e "sudo ssh -i /home/abdulazizm41/.ssh/gcp-user-1"
abdulazizm41@35.236.204.230:/home/abdulazizm41/docker-compose-lamp/
```

***Script*:**

```
sudo rsync -avz /home/abdulazizm41/Downloads/shared/script/ -e "sudo ssh
-i /home/abdulazizm41/.ssh/gcp-user-1"
abdulazizm41@35.236.204.230:/home/abdulazizm41/script/
```

Direktori data web, video, dan *script* disesuaikan dengan direktori komputer lokal dan direktori *instance* server utama. Pada penelitian ini, data web, video, dan *script* terletak di sebuah direktori bernama *shared* pada komputer lokal, yang akan dipindahkan ke direktori *home* pada *instance* server utama.

4.1.8 Menjalankan Layanan Menggunakan *Script*

Script dibuat untuk mengotomatisasikan proses-proses yang terdapat di dalam sistem, mulai dari menjalankan, menghentikan, dan memigrasikan layanan antar *instance*. *Script* menjalankan layanan terdiri dari tiga bagian, yakni bagian utama *script*, bagian menjalankan perintah, dan bagian menampilkan pesan kesalahan. *Pseudo-code* masing-masing bagian secara berurutan ditunjukkan pada Tabel 4.3, 4.4, dan 4.5.

Tabel 4.3 *Pseudo-code* bagian utama *script*

Pseudo-code 1: Bagian Utama *Script*

```
1  DO print process information
2  SET command = 'sudo podman run -d --name webserver ...
   docker.io/abdulazizm41/webserver:7.3.x'
3  DO run_command(command)
4  DO print process status
5
6  DO print process information
7  SET command = 'sudo podman run -d --name mysql ...
   docker.io/abdulazizm41/mysql:5.7'
8  DO run_command(command)
9  DO print process status
```

Bagian utama *script* berisi perintah untuk menjalankan dua proses utama yang digunakan pada *instance* server utama, yakni proses webserver dan MySQL. Perintah tersebut disimpan pada variabel `command` yang kemudian dikirimkan sebagai parameter fungsi `run_command`. *Script* ini juga menampilkan informasi dan status proses yang dijalankan. Perintah yang digunakan merupakan perintah *bash* yang dipanggil melalui *script* Python. Hal ini dapat dilakukan dengan melakukan impor *library* `subprocess` dan `sys` dari Python 2.7.x.

Tabel 4.4 *Pseudo-code script* menjalankan perintah

Pseudo-code 2: *Script* Menjalankan Perintah

```
1  DEFINE function run_command(command)
2      SET process = open bash shell as subprocess
3      SET retur_n = get return value from process
4      IF retur_n != 0 THEN
5          DO error()
6  END
```

Script menjalankan perintah merupakan *script* yang digunakan untuk menjalankan perintah *bash* yang terdapat pada parameter `command`. Agar perintah *bash* dapat dieksekusi, *bash shell* dibuka sebagai *subprocess* pada Python 2.7.x dengan bantuan *library* `subprocess` dan `sys`. Setelah perintah *bash* selesai dieksekusi, nilai kembalian dari perintah tersebut disimpan kedalam variabel `retur_n`. Apabila nilai dari variabel `retur_n` tidak sama dengan nol, maka pesan kesalahan akan ditampilkan.

Tabel 4.5 *Pseudo-code script* menampilkan pesan kesalahan

Pseudo-code 3: *Script* Menampilkan Pesan Kesalahan

```
1  DEFINE function error()
2      DO print error message
3      DO send error code
4      DO stop script execution
5  END
```

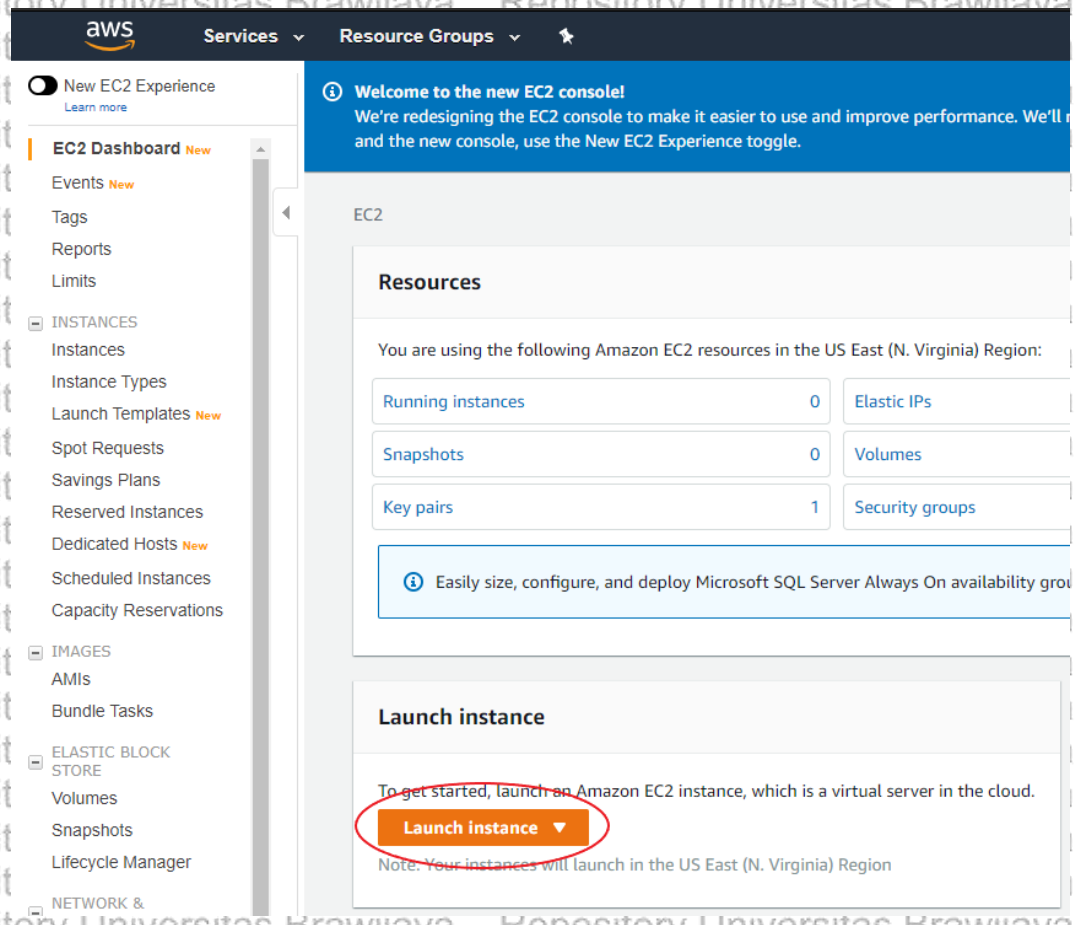
Script menampilkan pesan kesalahan merupakan *script* yang digunakan untuk menampilkan pesan kesalahan dan menghentikan *script* pada saat terjadinya suatu kesalahan saat mengeksekusi perintah *bash* tertentu. Atau dengan kata lain, nilai kembalian pada variabel `retur_n` di fungsi `run_command` tidak sama dengan nol.

4.2 Implementasi Server Cadangan

Server cadangan merupakan server yang terdapat pada layanan *cloud* milik Amazon yakni AWS. Dilakukan persiapan pada server cadangan agar dapat menerima permintaan migrasi dari server utama dan melanjutkan layanannya. Langkah-langkah implementasi server cadangan terdiri dari pembuatan *instance* server cadangan, konfigurasi *security group*, konfigurasi *elastic* IP, melakukan *update* dan instalasi aplikasi yang diperlukan, konfigurasi SSHD dan selinux, pemindahan data ke *cloud*, dan melakukan percobaan menjalankan layanan sejenis layanan server utama.

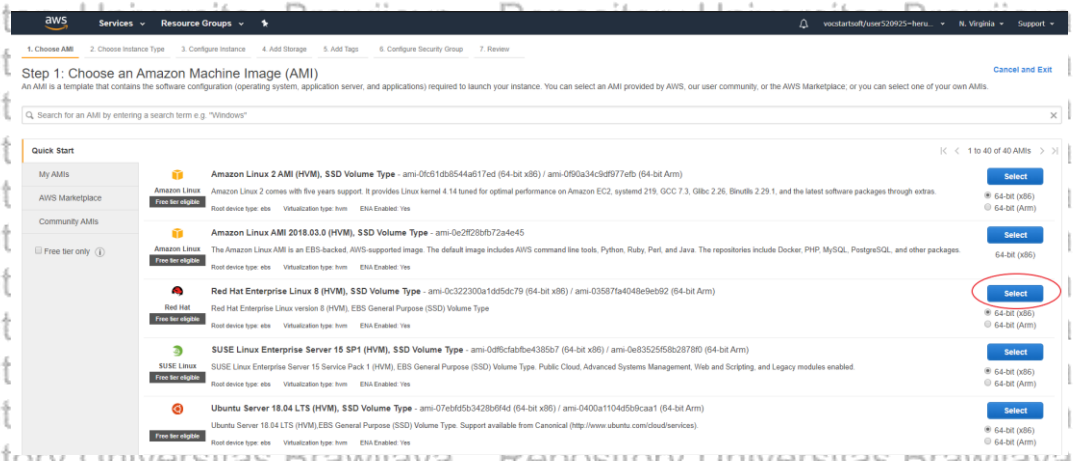
4.2.1 Pembuatan *Instance* Server Cadangan

Implementasi server cadangan dimulai dengan proses pembuatan *instance* server cadangan. Caranya adalah dengan masuk ke AWS *console*, kemudian masuk ke EC2 *dashboard*. Selanjutnya tekan tombol *launch instance* seperti yang ditunjukkan pada Gambar 4.22.



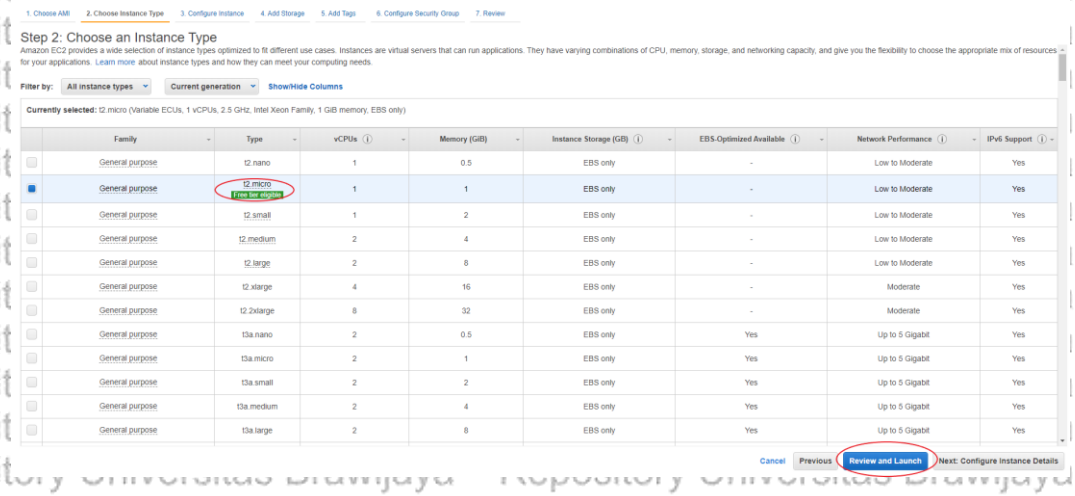
Gambar 4.22 Pembuatan *instance* server cadangan

Pada halaman berikutnya, pilih *Amazon Machine Image* (AMI) yang akan digunakan. Dalam penelitian ini digunakan RHEL 8, karena RHEL merupakan salah satu distro linux yang dibangun, dikembangkan, dan dioptimasi untuk keperluan manajemen server. Proses pemilihan AMI ditunjukkan pada Gambar 4.23.



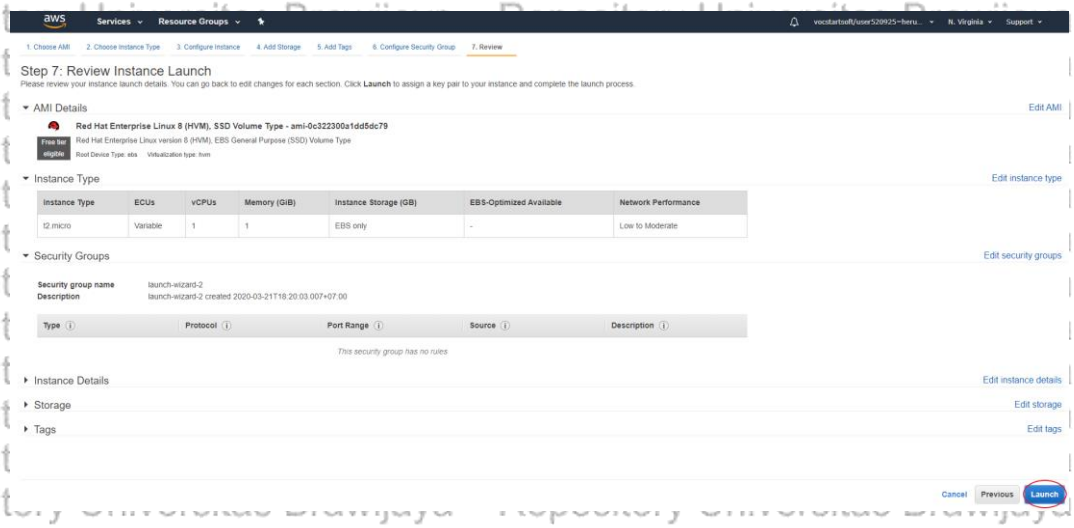
Gambar 4.23 Pemilihan AMI pada AWS

Setelah pemilihan AMI selesai dilakukan, langkah berikutnya adalah memilih tipe *instance* yang akan digunakan. AWS menyediakan berbagai tipe *instance* sesuai keperluannya masing-masing. Dalam penelitian ini, digunakan AWS *educate* yang menyebabkan terbatasnya tipe *instance* yang dapat dipilih, yaitu hanya tipe t2.micro saja. Tipe-tipe *instance* yang terdapat pada AWS ditunjukkan pada Gambar 4.24.



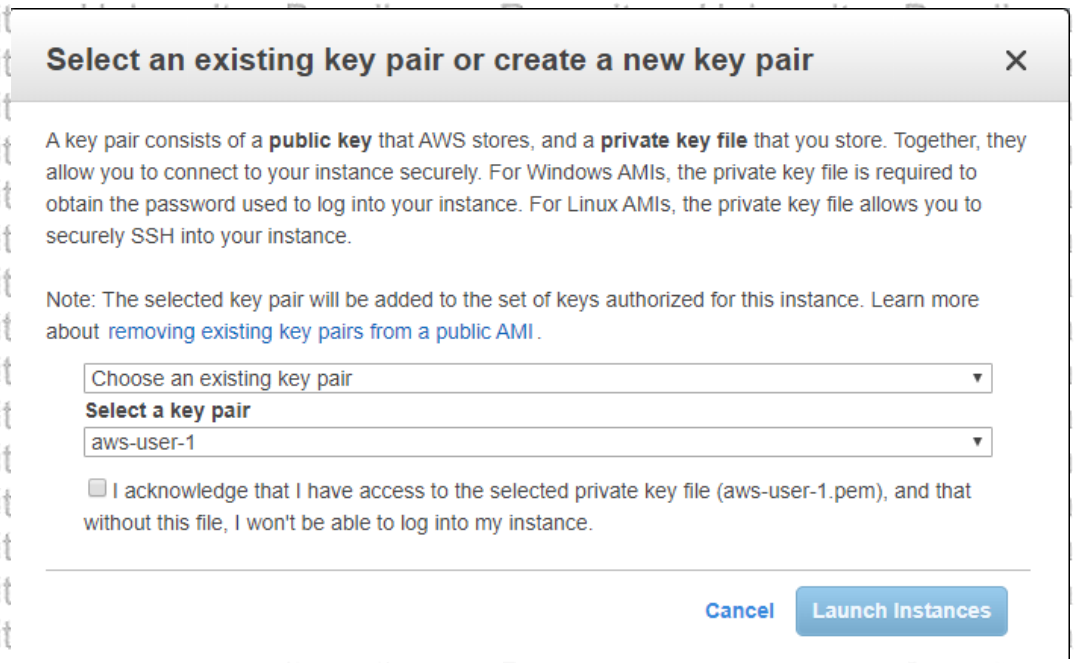
Gambar 4.24 Tipe-tipe *instance* pada AWS

Setelah pemilihan tipe *instance* selesai dilakukan, terdapat dua pilihan untuk langkah selanjutnya, yakni *review and launch* atau *next*. Apabila memilih pilihan *next*, maka pengguna dapat melakukan konfigurasi lebih detail dari *instance* yang dibuat, mulai dari penyimpanan yang digunakan, hingga pengaturan *security group*. Untuk menyederhanakan proses penelitian, digunakan pilihan *review and launch*. Dengan memilih pilihan *review and launch*, konfigurasi *default* akan diterapkan oleh AWS pada *instance* yang dibuat, dan pengguna diarahkan ke halaman *review*. Pada halaman *review*, ditunjukkan detail dari konfigurasi yang dilakukan, sehingga pengguna dapat melihat dan melakukan penyesuaian terhadap konfigurasi *instance* sesuai keperluan. Halaman *review* ditunjukkan pada Gambar 4.25.



Gambar 4.25 Halaman *review* konfigurasi *instance* AWS

Setelah memastikan konfigurasi *instance* yang dilakukan telah sesuai, tekan tombol *launch* yang kemudian memunculkan sebuah dialog mengenai SSH *key* yang akan digunakan untuk mengakses *instance* yang dibuat, seperti yang ditunjukkan pada Gambar 4.26.

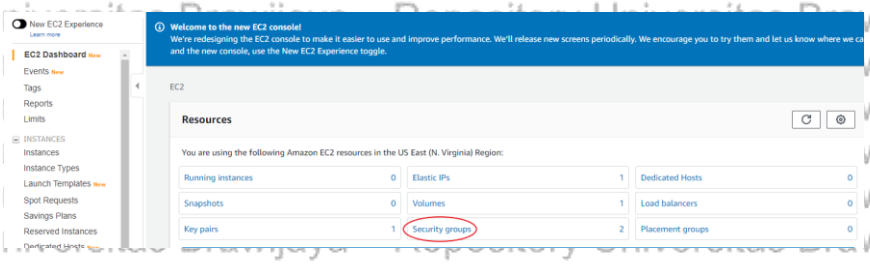


Gambar 4.26 Dialog SSH *key* pada AWS

Pada dialog SSH *key* ini, pengguna dapat memilih untuk membuat atau menggunakan pasangan SSH *key* yang sudah dibuat sebelumnya. Apabila pengguna memilih untuk membuat pasangan SSH *key* yang baru, maka setelah pengguna menekan tombol *launch instance*, pengguna akan secara otomatis mengunduh *private key* dari *instance* yang dibuat, dan dialihkan ke halaman *running instance* untuk melihat status dari *instance* yang dibuat.

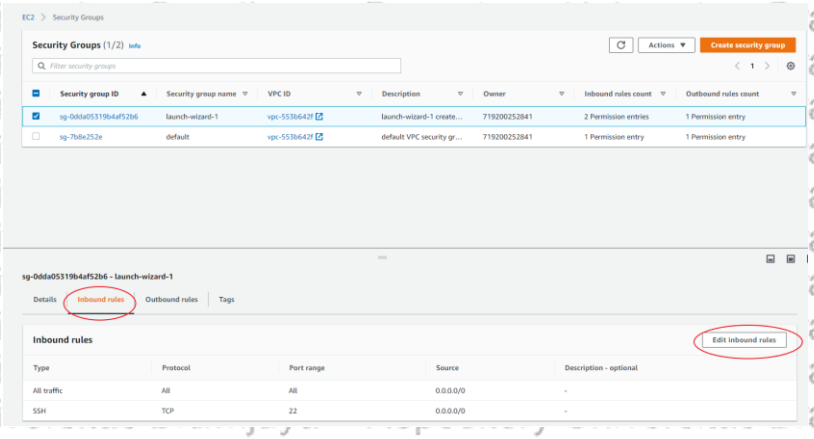
4.2.2 Konfigurasi *Security Group* Pada AWS

Setelah proses pembuatan *instance* server cadangan selesai dilakukan, langkah berikutnya adalah melakukan konfigurasi *security group*. Ditambahkan sebuah *rule* yang mengizinkan semua *traffic* masuk ke *instance* server cadangan. Hal ini dilakukan untuk mempermudah pengaksesan *instance* server cadangan. Caranya adalah dengan masuk ke EC2 *dashboard*, kemudian pilih pilihan *security group* seperti yang ditunjukkan pada Gambar 4.27.

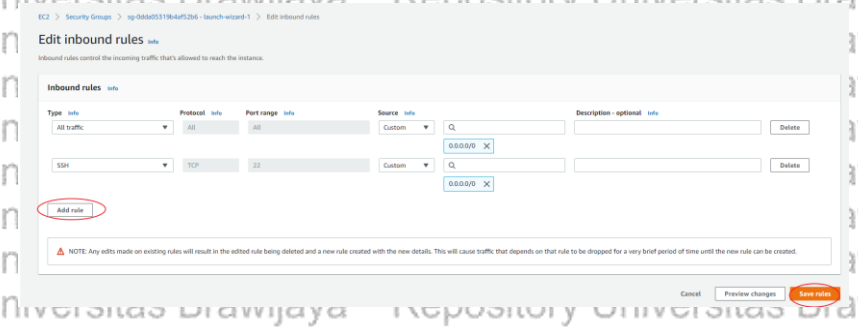


Gambar 4.27 Konfigurasi *security group* pada AWS

Pada halaman *security group*, pilih *security group* yang digunakan oleh *instance* server cadangan, kemudian pilih pilihan *inbound* dibagian bawah halaman. Selanjutnya tekan tombol *edit inbound rule* yang akan mengarahkan pengguna ke halaman berikutnya, seperti yang ditunjukkan pada Gambar 4.28 dan 4.29.



Gambar 4.28 Konfigurasi *security group* pada AWS (lanjutan)

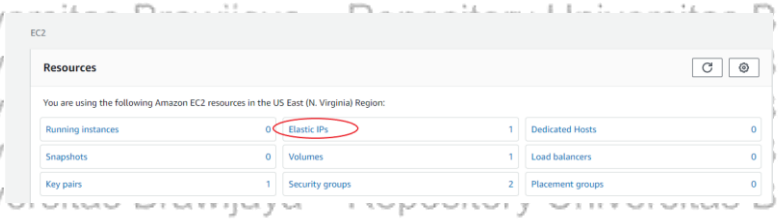


Gambar 4.29 Konfigurasi *security group* pada AWS (lanjutan)

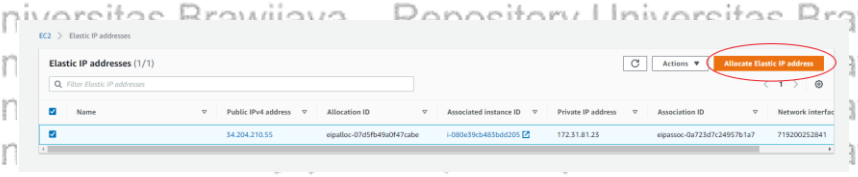
Pada halaman *edit inbound rule*, tekan tombol *add rule* untuk menambahkan *rule* baru pada *security group*. *Rule* yang ditambahkan sesuai dengan Tabel 3.7 yang terdapat pada Subbab 3.2.5 Perancangan Server Cadangan. Setelah *rule* selesai ditambahkan, tekan tombol *save rule* yang terletak di bagian kanan bawah halaman *edit inbound rule*.

4.2.3 Konfigurasi *Elastic* IP Pada AWS

Setelah konfigurasi *security group* selesai dilakukan, langkah selanjutnya adalah melakukan konfigurasi *elastic* IP. *Elastic* IP merupakan fitur yang disediakan oleh AWS untuk mendapatkan IP publik statis. Konfigurasi *elastic* IP dilakukan agar *load balancer* dapat mengarahkan *traffic* pengguna ke alamat IP yang konsisten. Langkah-langkah konfigurasi *elastic* IP pada AWS ditunjukkan pada Gambar 4.30 dan 4.31.

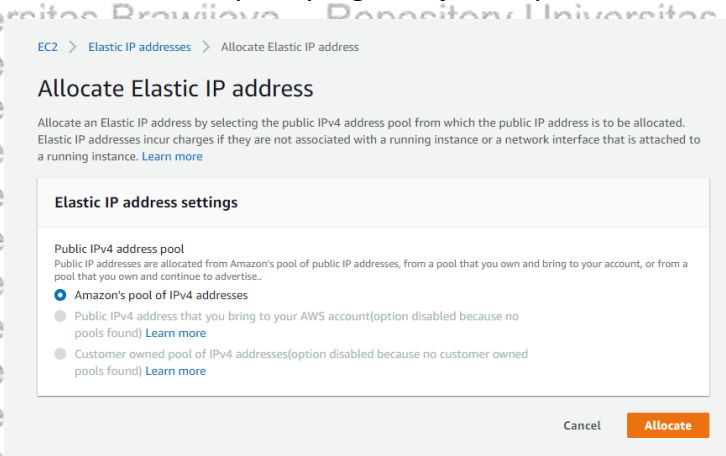


Gambar 4.30 Konfigurasi *elastic* IP pada AWS



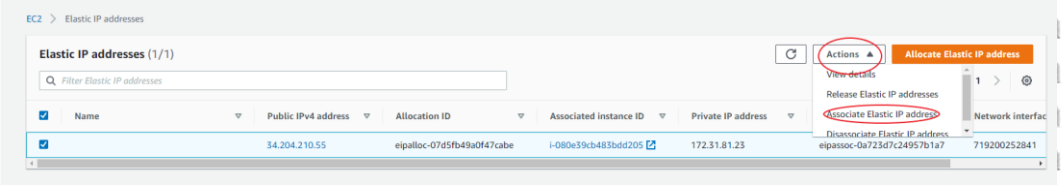
Gambar 4.31 Konfigurasi *elastic* IP pada AWS (lanjutan)

Langkah pertama dalam melakukan konfigurasi *elastic* IP pada AWS adalah dengan masuk ke EC2 *dashboard*, kemudian pilih pilihan *elastic* IP. Pada halaman berikutnya, tekan tombol *allocate elastic* IP *address* yang mengarahkan pengguna ke halaman baru, seperti yang ditunjukkan pada Gambar 4.32.

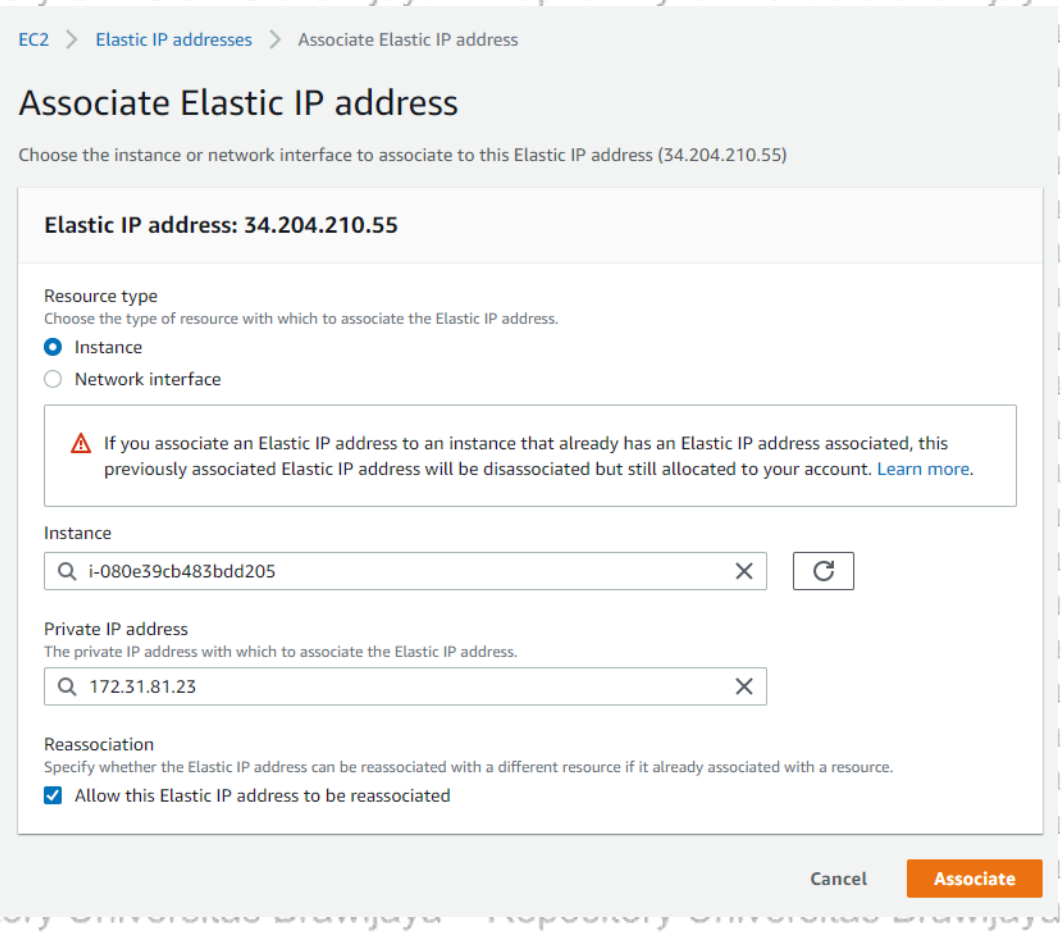


Gambar 4.32 Konfigurasi *elastic* IP pada AWS (lanjutan)

Pada halaman *allocate elastic* IP *address*, tekan tombol *allocate*. AWS akan memberikan sebuah IP publik statis secara otomatis, yang dapat dikaitkan ke salah satu *instance* EC2 yang telah dibuat. Dalam penelitian ini, IP yang didapatkan adalah 34.204.210.55. Untuk mengaitkan IP publik statis yang telah didapatkan ke *instance* server cadangan, kembali ke halaman *elastic* IP *address* dan pilih pilihan *action*. Kemudian, pilih *associate elastic* IP *address* seperti yang ditunjukkan pada Gambar 4.33.



Gambar 4.33 Konfigurasi *elastic* IP pada AWS (lanjutan)



Gambar 4.34 Konfigurasi *elastic* IP pada AWS (lanjutan)

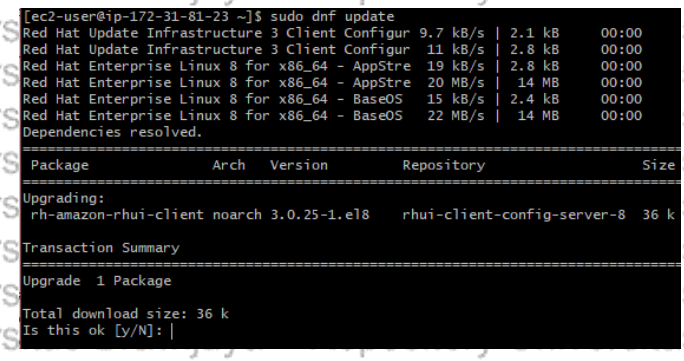
Pada halaman *associate elastic* IP *address*, pilih pilihan *instance* pada *resource type*, karena alamat IP ini akan dikaitkan ke *instance* server cadangan yang telah dibuat sebelumnya. Kemudian pada bagian *instance*, masukkan nomor identitas *instance* server cadangan, dan pada bagian *private* IP *address*, masukkan IP privat dari *instance* server cadangan. Langkah terakhir, berikan

tanda centang pada pilihan *allow this elastic* IP *address to be reassociated*, agar alamat IP ini dapat dikaitkan ulang ke *instance* lain jika diperlukan. Kemudian tekan tombol *associate* untuk menyelesaikan proses pengalokasian IP statis ke *instance* server cadangan.

4.2.4 Melakukan *Update* Dan Instalasi Aplikasi

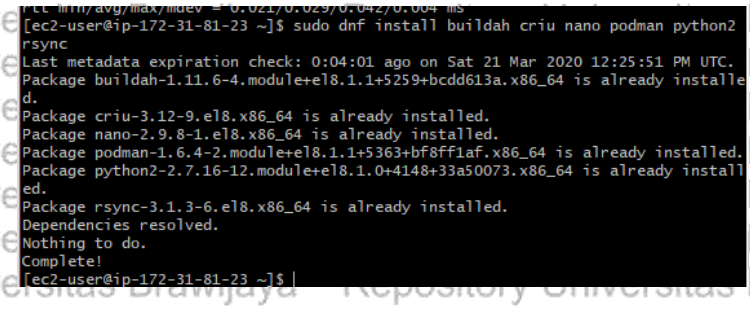
Setelah konfigurasi *elastic* IP selesai dilakukan, langkah selanjutnya adalah melakukan *update* dan instalasi aplikasi yang diperlukan oleh *instance* server cadangan. Pertama-tama masuk ke *instance* server cadangan melalui SSH dengan menggunakan perintah `ssh -i /home/abdulazizm41/.ssh/aws-user-1.pem ec2-user@34.204.210.55`. Opsi `-i` adalah singkatan dari *identity_file* yang merupakan *private key* yang digunakan untuk masuk pada *instance* server cadangan. `/home/abdulazizm41/.ssh/` merupakan direktori letak *private key* pada komputer lokal dan `aws-user-1.pem` merupakan nama *private key* yang digunakan.

Setelah masuk ke *instance* server cadangan, dilakukan proses *update* dengan menggunakan perintah `sudo dnf update -y`. DNF merupakan *package manager* yang terdapat pada distro linux RHEL dan turunannya. Opsi `-y` merupakan singkatan dari *yes* yang digunakan untuk menjalankan sebuah perintah tanpa tanggapan dari pengguna. Proses *update* ditunjukkan pada Gambar 4.35.



Gambar 4.35 Proses *update instance* server cadangan

Setelah proses *update* selesai, dilanjutkan dengan proses instalasi aplikasi yang diperlukan. Aplikasi-aplikasi yang diperlukan oleh *instance* server cadangan dapat dilihat pada Tabel 3.8 yang terdapat pada Subbab 3.2.5 Perancangan Server Cadangan. Perintah yang digunakan dan proses instalasi aplikasi ditunjukkan pada Gambar 4.36.



Gambar 4.36 Proses instalasi aplikasi *instance* server cadangan

Setelah proses instalasi aplikasi yang diperlukan selesai, dilanjutkan dengan mengecek apakah setiap aplikasi sudah terinstal dengan benar. Caranya adalah dengan mengecek versi dari masing-masing aplikasi seperti yang ditunjukkan pada Gambar 4.37.

```
[ec2-user@ip-172-31-81-23 ~]$ podman --version
podman version 1.6.4
[ec2-user@ip-172-31-81-23 ~]$ criu --version
Version: 3.12
[ec2-user@ip-172-31-81-23 ~]$ python2 --version
Python 2.7.16
[ec2-user@ip-172-31-81-23 ~]$ nano --version
 GNU nano, version 2.9.8
 (C) 1999-2011, 2013-2018 Free Software Foundation, Inc.
 (C) 2014-2018 the contributors to nano
 Email: nano@nano-editor.org    Web: https://nano-editor.org/
 Compiled options: --enable-utf8
[ec2-user@ip-172-31-81-23 ~]$ buildah --version
buildah version 1.11.6 (image-spec 1.0.1-dev, runtime-spec 1.0.1-dev)
[ec2-user@ip-172-31-81-23 ~]$ rsync --version
rsync  version 3.1.3  protocol version 31
Copyright (C) 1996-2018 by Andrew Tridgell, Wayne Davison, and others.
Web site: http://rsync.samba.org/
Capabilities:
    64-bit files, 64-bit inums, 64-bit timestamps, 64-bit long ints,
    socketpairs, hardlinks, symlinks, IPv6, batchfiles, inplace,
    append, ACLs, xattrs, iconv, symtimes, prealloc

rsync comes with ABSOLUTELY NO WARRANTY.  This is free software, and you
are welcome to redistribute it under certain conditions.  See the GNU
General Public Licence for details.
[ec2-user@ip-172-31-81-23 ~]$
```

Gambar 4.37 Versi dari setiap aplikasi yang diinstall

4.2.5 Konfigurasi SSHD Dan Selinux

Langkah berikutnya dalam implementasi server cadangan adalah melakukan konfigurasi SSHD dan selinux. Konfigurasi selinux dibuat menjadi *disabled* untuk mempermudah proses *live migration*. Konfigurasi selinux dilakukan dengan melakukan perubahan pada berkas `/etc/selinux/config`. Proses konfigurasi selinux ditunjukkan pada Gambar 4.38 dan 4.39.

```
[ec2-user@ip-172-31-81-23 ~]$ sudo nano /etc/selinux/config
```

Gambar 4.38 Konfigurasi selinux

```
 GNU nano 2.9.8                  /etc/selinux/config

# This file controls the state of SELinux on the system.
# SELINUX= can take one of these three values:
#     enforcing - SELinux security policy is enforced.
#     permissive - SELinux prints warnings instead of enforcing.
#     disabled - No SELinux policy is loaded.
SELINUX=disabled
# SELINUXTYPE= can take one of these three values:
#     targeted - Targeted processes are protected,
#     minimum - Modification of targeted policy. Only selected processes are pr$
#     mls - Multi Level Security protection.
SELINUXTYPE=targeted
```

Gambar 4.39 Konfigurasi selinux (lanjutan)

Setelah konfigurasi selinux selesai dilakukan, nyalakan ulang komputer agar konfigurasi dapat dibaca oleh sistem. Setelah itu, cek menggunakan perintah `getenforce` dan/atau `sestatus` seperti yang ditunjukkan pada Gambar 4.40 dan 4.41. Apabila muncul bacaan *disabled*, maka konfigurasi selinux selesai dilakukan.

```
[ec2-user@ip-172-31-81-23 ~]$ sudo shutdown -r now
```

Gambar 4.40 Menyalakan ulang komputer

```
[ec2-user@ip-172-31-81-23 ~]$ getenforce
Disabled
[ec2-user@ip-172-31-81-23 ~]$ sestatus
SELinux status:                 disabled
[ec2-user@ip-172-31-81-23 ~]$
```

Gambar 4.41 Mengecek status selinux

Langkah selanjutnya adalah melakukan konfigurasi SSHD. Konfigurasi SSHD dilakukan karena tiga alasan, yaitu:

- (Opsional) Agar koneksi dari klien SSH ke server SSH tidak sering terputus akibat tidak ada permintaan dari klien SSH dalam jangka waktu tertentu.
- Mengizinkan login sebagai pengguna root melalui SSH. Hal ini diperlukan untuk proses *live migration*.
- Menonaktifkan autentikasi sandi. Hal ini diperlukan agar klien SSH dapat terhubung ke server SSH hanya dengan menggunakan pasangan SSH *key* yang telah dibuat.

Konfigurasi SSHD dilakukan dengan melakukan perubahan pada berkas `/etc/ssh/sshd_config`. Parameter-parameter yang diubah beserta keterangan setiap parameter dijelaskan sebagai berikut:

- **ClientAliveInterval** 60

  Server akan menunggu selama 60 detik sebelum mengirimkan *null packet* kepada klien untuk menjaga koneksi tetap aktif.
- **TCPKeepAlive** yes

  Untuk memastikan bahwa tidak ada *rule firewall* tertentu yang menghapus koneksi *idle*.
- **ClientAliveCountMax** 10000

  Server mengirimkan pesan kepada klien, walaupun server tidak menerima pesan apapun dari klien.
- **PermitRootLogin** yes

  Mengizinkan login sebagai pengguna root melalui SSH.
- **PasswordAuthentication** no

  Menonaktifkan autentikasi sandi.

Proses konfigurasi SSHD ditunjukkan pada Gambar 4.42, 4.43, dan 4.44.

```
[ec2-user@ip-172-31-81-23 ~]$ sudo nano /etc/ssh/sshd_config
```

Gambar 4.42 Konfigurasi SSHD

```
#LoginGraceTime 2m
PermitRootLogin yes
#StrictModes yes
#MaxAuthTries 6
#MaxSessions 10

#PubkeyAuthentication yes

# The default is to check both .ssh/authorized_keys and .ssh/authorized_keys2
# but this is overridden so installations will only check .ssh/authorized_keys
AuthorizedKeysFile .ssh/authorized_keys

#AuthorizedPrincipalsFile none

#AuthorizedKeysCommand none
#AuthorizedKeysCommandUser nobody

# For this to work you will also need host keys in /etc/ssh/ssh_known_hosts
#HostbasedAuthentication no
# Change to yes if you don't trust ~/.ssh/known_hosts for
# HostbasedAuthentication
#IgnoreUserKnownHosts no
# Don't read the user's ~/.rhosts and ~/.shosts files
#IgnoreRhosts yes

# To disable tunneled clear text passwords, change to no here!
#PasswordAuthentication yes
#PermitEmptyPasswords no
PasswordAuthentication no

# Change to no to disable s/key passwords
```

Gambar 4.43 Konfigurasi SSHD (lanjutan)

```
#PrintLastLog yes
TCPKeepAlive yes
#PermitUserEnvironment no
#Compression delayed
ClientAliveInterval 60
ClientAliveCountMax 10000
#ShowPatchLevel no
#UseDNS no
#PidFile /var/run/sshd.pid
#MaxStartups 10:30:100
#PermitTunnel no
#ChrootDirectory none
#VersionAddendum none
```

Gambar 4.44 Konfigurasi SSHD (lanjutan)

Setelah proses konfigurasi SSHD selesai dilakukan, nyalakan ulang layanan SSHD dengan perintah `sudo systemctl restart sshd.service`.

4.2.6 Pemindahan Data Ke *Cloud*

Pemindahan data dilakukan dengan menggunakan utilitas rsync. Data yang dipindahkan merupakan data yang diperlukan untuk melakukan percobaan menjalankan layanan pada *instance* server cadangan, yakni data web, video, dan beberapa *script* yang diperlukan. Perintah yang digunakan dalam proses pemindahan data dapat dilihat pada Tabel 4.6.

Tabel 4.6 Perintah memindahkan data ke *cloud*

Data web dan video:

```
sudo rsync -avz /home/abdulazizm41/Downloads/shared/docker-compose-lamp/
-e "sudo ssh -i /home/abdulazizm41/.ssh/aws-user-1.pem" ec2-
user@34.204.210.55:/home/ec2-user/
```

***Script*:**

```
sudo rsync -avz /home/abdulazizm41/Downloads/shared/script/ -e "sudo ssh
-i /home/abdulazizm41/.ssh/aws-user-1.pem" ec2-
user@34.204.210.55:/home/ec2-user/script/
```

Direktori data web, video, dan *script* disesuaikan dengan direktori komputer lokal dan direktori *instance* server cadangan. Pada penelitian ini, data web, video, dan *script* terletak di sebuah direktori bernama *shared* pada komputer lokal, yang akan dipindahkan ke direktori *home* pada *instance* server cadangan.

4.2.7 Percobaan Menjalankan Layanan

Langkah terakhir dalam implementasi server cadangan adalah melakukan percobaan menjalankan layanan yang sama dengan server utama, untuk mengetahui bagaimana server cadangan dapat mengatasi layanan yang sejenis sebelum proses migrasi dilakukan. Untuk mempermudah prosesnya, digunakan *script* untuk menjalankan, menghentikan, dan menerima permintaan migrasi dari server utama. *Script* yang digunakan untuk menjalankan layanan pada server cadangan terdiri dari tiga bagian, yakni bagian utama *script*, bagian menjalankan perintah, dan bagian menampilkan pesan kesalahan. *Pseudo-code* masing-masing bagian secara berurutan ditunjukkan pada Tabel 4.7, 4.8, dan 4.9.

Tabel 4.7 *Pseudo-code* bagian utama *script*

Pseudo-code 4: Bagian Utama *Script*

```
1  DO print process information
2  SET command = 'sudo podman run -d --name webserver ...
   docker.io/abdulazizm41/webserver:7.3.x'
3  DO run_command(command)
4  DO print process status
5
6  DO print process information
7  SET command = 'sudo podman run -d --name mysql ...
   docker.io/abdulazizm41/mysql:5.7'
8  DO run_command(command)
9  DO print process status
```

Bagian utama *script* berisi perintah untuk menjalankan dua proses yang sama seperti proses pada *instance* server utama, yaitu proses webserver dan MySQL. Perintah tersebut disimpan pada variabel `command` yang kemudian dikirimkan

sebagai parameter fungsi `run_command`. *Script* ini juga menampilkan informasi dan status proses yang dijalankan. Perintah yang digunakan merupakan perintah *bash* yang dipanggil melalui *script* Python. Hal ini dapat dilakukan dengan melakukan impor *library* `subprocess` dan `sys` dari Python 2.7.x.

Tabel 4.8 *Pseudo-code script* menjalankan perintah

Pseudo-code 5: *Script* Menjalankan Perintah	
1	DEFINE function run_command(command)
2	SET process = open bash shell as subprocess
3	SET retur_n = get return value from process
4	IF retur_n != 0 THEN
5	DO error()
6	END

Script menjalankan perintah merupakan *script* yang digunakan untuk menjalankan perintah *bash* yang terdapat pada parameter `command`. Agar perintah *bash* dapat dieksekusi, *bash shell* dibuka sebagai *subprocess* pada Python 2.7.x dengan bantuan *library* `subprocess` dan `sys`. Setelah perintah *bash* selesai dieksekusi, nilai kembalian dari perintah tersebut disimpan kedalam variabel `retur_n`. Apabila nilai dari variabel `retur_n` tidak sama dengan nol, maka pesan kesalahan akan ditampilkan.

Tabel 4.9 *Pseudo-code script* menampilkan pesan kesalahan

Pseudo-code 6: *Script* Menampilkan Pesan Kesalahan	
1	DEFINE function error()
2	DO print error message
3	DO send error code
4	DO stop script execution
5	END

Script menampilkan pesan kesalahan merupakan *script* yang digunakan untuk menampilkan pesan kesalahan dan menghentikan *script* pada saat terjadinya suatu kesalahan saat mengeksekusi perintah *bash* tertentu. Atau dengan kata lain, nilai kembalian pada variabel `retur_n` di fungsi `run_command` tidak sama dengan nol.

Setelah semua layanan dapat berjalan dengan baik pada server cadangan, selanjutnya layanan dihentikan menggunakan *script* yang hampir sama dengan *script* untuk menjalankan layanan. Perbedaannya hanya terdapat pada bagian utamanya saja. *Pseudo-code* bagian utama *script* untuk menghentikan layanan pada server cadangan ditunjukkan pada Tabel 4.10.

Tabel 4.10 *Pseudo-code script* menghentikan layanan

Pseudo-code 7: *Script* Menghentikan Layanan	
1	DO print process information
2	SET command = 'sudo podman container stop --all'
3	DO run_command(command)
4	DO print process status
5	
6	DO print process information
7	SET command = 'sudo podman container prune'
8	DO run_command(command)
9	DO print process status
10	
11	DO print process information

Tabel 4.10 (lanjutan)

Pseudo-code 7: *Script* Menghentikan Layanan	
12	SET command = 'sudo podman volume prune -f --force'
13	DO run_command(command)
14	DO print process status

Bagian utama *script* menghentikan layanan berisi perintah untuk mencetak informasi dan status proses yang dihentikan. *Script* ini juga berisi perintah untuk menghentikan dua proses utama pada *instance* server cadangan, yakni webserver dan MySQL, menghapus seluruh *container* yang tidak berjalan, dan menghapus seluruh *volume* yang tidak terkait ke *container* mana pun.

4.3 Implementasi *Load Balancer*

Load balancer merupakan *instance* yang dibuat pada layanan *cloud* milik Google, yakni GCP. *Load balancer* bertugas untuk mengarahkan *traffic* pengguna ke salah satu server yang memiliki layanan. Langkah-langkah implementasi *load balancer* hampir sama dengan implementasi server utama, dengan penyesuaian sebagai berikut:

- IP publik statis yang didapatkan adalah 35.345.113.158.
- Berkas konfigurasi *load balancer* yang diletakkan pada direktori *home instance load balancer*. Berkas konfigurasi *load balancer* berisi konfigurasi berbasis Nginx yang di dalamnya terdapat daftar server yang akan menerima *traffic* pengguna. Isi dari berkas konfigurasi *load balancer* ditunjukkan pada Tabel 4.11.

Tabel 4.11 Konfigurasi *load balancer*

```
upstream app1 {
    server 35.236.204.230;
    server 34.204.210.55;
}
server {
    listen 80;
    location / {
        proxy_pass http://app1;
    }
}
```

- *Script* untuk memulai layanan Nginx sebagai *load balancer*. *Script* untuk memulai layanan *load balancer* hampir sama dengan *script* untuk menjalankan layanan pada server utama. Perbedaannya terdapat pada bagian utama *script*. *Pseudo-code* bagian utama *script* menjalankan *load balancer* ditunjukkan pada Tabel 4.12.

Tabel 4.12 *Pseudo-code* bagian utama *script load balancer*

Pseudo-code 8: Bagian Utama *Script Load Balancer*	
1	DO print process information
2	SET command = 'sudo podman run -d --name loadbalancer ... docker.io/library/nginx:latest'
3	DO run_command(command)
4	DO print process status

Bagian utama *script load balancer* berisi perintah untuk mencetak informasi dan status proses yang dijalankan. *Script* ini juga berisi perintah untuk menjalankan proses *load balancer* pada *instance load balancer*.

4.4 Implementasi *Script Live Migration*

Script live migration terdiri dari dua jenis, yakni *script* permintaan *live migration* dan *script* penerimaan *live migration*. *Script* permintaan *live migration* dijalankan oleh server yang akan melakukan migrasi, sedangkan *script* penerimaan *live migration* dijalankan oleh server yang akan menerima permintaan migrasi. *Script* penerimaan *live migration* harus dijalankan terlebih dahulu sebelum proses permintaan *live migration* dapat dilakukan.

4.4.1 *Script* Permintaan *Live Migration*

Script permintaan *live migration* dijalankan oleh server yang akan melakukan migrasi. *Script* permintaan *live migration* hanya dapat dijalankan setelah *script* penerimaan *live migration* berjalan. *Script* permintaan *live migration* terdiri dari enam fungsi, antara lain fungsi `prepare()`, `pre_dump_transfer()`, `dump()`, `post_dump_transfer()`, `migrate()`, dan `cleanup()`. *Pseudo-code* masing-masing fungsi secara berurutan ditunjukkan pada Tabel 4.13 hingga Tabel 4.18.

Tabel 4.13 *Pseudo-code* fungsi `prepare()`

Pseudo-code 9: Fungsi prepare()

```
1  DEFINE function prepare()
2      SET command = 'sudo rm --recursive --force /sharedy/cp/'
3      DO run_command(command)
4
5      SET command = 'sudo mkdir /sharedy/cp/'
6      DO run_command(command)
7  END
```

Fungsi `prepare()` berisikan dua perintah, yaitu perintah untuk menghapus direktori tempat berkas *checkpoint* hasil *dump* diletakkan dan perintah untuk membuatnya kembali dalam keadaan kosong. Hal ini dilakukan untuk menghindari tertumpuknya berkas *checkpoint* hasil *dump* dan mengurangi penggunaan media penyimpanan yang tersedia.

Tabel 4.14 *Pseudo-code* fungsi `pre_dump_transfer()`

Pseudo-code 10: Fungsi pre_dump_transfer()

```
1  DEFINE function pre_dump_transfer(user, destination)
2      SET start = time.time()
3      SET command = 'sudo rsync -aqz /sharedy/docker-compose-lamp/ ...
:/sharedy/docker-compose-lamp/'
4      DO run_command(command)
5      SET end = time.time()
6      DO print (end - start)
7  END
```

Fungsi `pre_dump_transfer()` berisi perintah untuk melakukan transfer data sebelum proses migrasi dilakukan. Di antara proses tersebut dilakukan pencatatan waktu mulai dan waktu selesai dari proses transfer, yang kemudian ditampilkan saat proses transfer selesai dilakukan.

Tabel 4.15 *Pseudo-code* fungsi `dump()`

Pseudo-code 11: Fungsi dump()

```
1  DEFINE function dump()
2      DO print down_timestamp
3      DO print process information
4      SET command = 'sudo podman container checkpoint --export
   /sharedy/cp/mysql.tar.gz --keep --tcp-established mysql'
5      DO run_command(command)
6      DO print process status
7
8      DO print process information
9      SET command = 'sudo podman container checkpoint --export
   /sharedy/cp/webserver.tar.gz --keep --tcp-established webserver'
10     DO run_command(command)
11     DO print process status
12 END
```

Fungsi `dump()` berisi perintah untuk memigrasikan seluruh proses pada server utama ke server cadangan, atau sebaliknya. Ada dua proses yang dimigrasikan, yakni proses webserver dan MySQL. Hasil *dump* dari kedua proses disimpan dalam bentuk berkas .tar.gz.

Tabel 4.16 *Pseudo-code* fungsi `post_dump_transfer()`

Pseudo-code 12: Fungsi post_dump_transfer()

```
1  DEFINE function post_dump_transfer(destination, user)
2      SET start = time.time()
3      SET command = 'sudo rsync -aqz /sharedy/cp/ -e "sudo ssh -i
   /home/abdulazizm41/.ssh/aws-user-1.pem" ' + user + '@' + destination
   + ':/sharedy/cp/'
4      DO run_command(command)
5
6      SET command = 'sudo rsync -aqz /sharedy/docker-compose-lamp/ ...
   :/sharedy/docker-compose-lamp/'
7      DO run_command(command)
8      SET end = time.time()
9
10     DO print (end - start)
11 END
```

Fungsi `post_dump_transfer()` berisi perintah untuk melakukan transfer data setelah proses migrasi selesai dilakukan. Data yang ditransfer hanyalah data yang berubah sejak data awal dikirimkan pada proses `pre_dump_transfer()`. Hal ini dapat dilakukan dengan bantuan utilitas rsync.

Tabel 4.17 *Pseudo-code* fungsi `migrate()`

Pseudo-code 13: Fungsi migrate()

```
1  DEFINE function migrate(destination)
2      SET socke_t = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
3      SET port = 12345
4      DO socke_t.connect((destination, port))
5
6      DO print process information
7      DO socke_t.send("OK")
8      DO print process status
9  END
```

Fungsi `migrate()` berisikan perintah untuk mengirimkan sebuah pesan kepada server yang menerima permintaan migrasi, bahwa proses *dump* dan transfer data telah selesai dilakukan.

Tabel 4.18 *Pseudo-code* fungsi `cleanup()`

Pseudo-code 14: Fungsi cleanup()

```
1  DEFINE function cleanup()
2      DO print process information
3      SET command = 'sudo ./stop.py'
4      DO run_command(command)
5      DO print process status
6
7      DO print process information
8      SET command = 'sudo rm -recursive -force /sharedy/*/'
9      DO run_command(command)
10     DO print process status
11 END
```

Fungsi `cleanup()` berisikan perintah untuk membersihkan server yang melakukan migrasi dengan menjalankan *script* stop.py. *Pseudo-code script* stop.py telah dituliskan pada Tabel 4.10 yang terdapat pada Subbab 4.2.7 Percobaan Menjalankan Layanan. Setelah *script* stop.py selesai dieksekusi, dilakukan penghapusan direktori tempat data layanan diletakkan pada server yang melakukan migrasi.

4.4.2 *Script* Penerimaan *Live Migration*

Script penerimaan *live migration* dijalankan oleh server yang akan menerima permintaan migrasi. *Script* penerimaan *live migration* harus dijalankan terlebih dahulu sebelum *script* permintaan *live migration*. *Script* penerimaan *live migration* terdiri dari dua fungsi, yaitu fungsi `prepare()` dan fungsi `restore()`. *Pseudo-code* dari kedua *script* tersebut dapat dilihat pada Tabel 4.19 dan 4.20.

Tabel 4.19 *Pseudo-code* fungsi `prepare()`

Pseudo-code 15: Fungsi prepare()

```
1  DEFINE function prepare()
2      SET command = 'sudo rm --recursive --force /sharedy/*'
3      DO run_command(command)
4  END
```

Fungsi `prepare()` berisi perintah untuk menghapus isi direktori letak data hasil migrasi akan ditempatkan. Hal ini dilakukan untuk menghindari penumpukan data dan penghematan penggunaan kapasitas media penyimpanan yang tersedia.

Tabel 4.20 *Pseudo-code* fungsi `restore()`

Pseudo-code 16: Fungsi restore()

```
1  DEFINE function restore()
2      DO print process information
3      SET command = 'sudo podman container restore --import
   /sharedy/cp/webserver.tar.gz --keep --tcp-established'
5      DO run_command(command)
6      DO print process status
7
8      DO print process information
9      SET command = 'sudo podman container restore --import
   /sharedy/cp/mysql.tar.gz --keep --tcp-established'
10     DO run_command(command)
11     DO print process status
12
13     DO print up_timestamp
```

Tabel 4.20 (lanjutan)

Pseudo-code 16: Fungsi restore()
14 END

Fungsi `restore()` berisi perintah untuk melakukan pemulihan proses pada server yang menerima permintaan migrasi. Terdapat dua proses yang dipulihkan, yaitu proses webserver dan MySQL. Pada bagian akhir fungsi `restore()`, terdapat perintah untuk menampilkan *timestamp* saat layanan mulai berjalan kembali.

BAB 5 PENGUJIAN DAN PEMBAHASAN

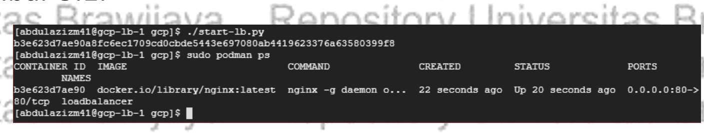
Bab ini menjabarkan proses dan data hasil pengujian yang diperoleh dari implementasi *container live migration* antar-*cloud provider* menggunakan Podman dan CRIU, beserta pembahasannya. Pada penelitian ini, pengujian terdiri dari tiga bagian, yaitu pengujian fungsional, *downtime*, dan *migration time*.

5.1 Pengujian Fungsional

Pengujian fungsional dilakukan untuk mengetahui apakah sistem yang dibangun telah memenuhi seluruh kebutuhan fungsional yang ditetapkan. Pengujian berikutnya hanya dapat dilakukan apabila semua kebutuhan fungsional telah terpenuhi. Apabila terdapat kebutuhan fungsional yang belum terpenuhi, maka dilakukan pengecekan kembali terhadap rancangan dan kode sistem, kemudian dilakukan pengujian ulang hingga semua kebutuhan terpenuhi.

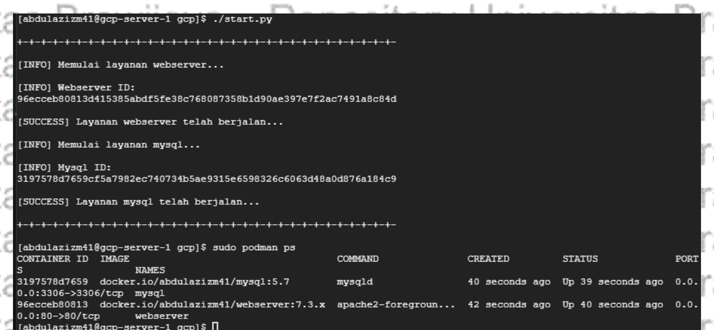
1. [LMF_01] *Load balancer* dapat mengarahkan *traffic* permintaan dari klien menuju server yang memiliki layanan dan [LMF_02] server yang memiliki layanan mampu merespon permintaan dari klien.

   Pengujian ini dilakukan untuk melihat apakah *load balancer* dapat mengarahkan *traffic* permintaan dari klien menuju server yang memiliki layanan dan meresponnya. Sesuai dengan Tabel 3.11 yang terdapat pada Subbab 3.3.1 Pengujian Fungsional, langkah pertama adalah dengan memastikan layanan *load balancer* telah berjalan pada *instance load balancer* di GCP. Layanan *load balancer* yang telah berjalan dapat dilihat pada Gambar 5.1.



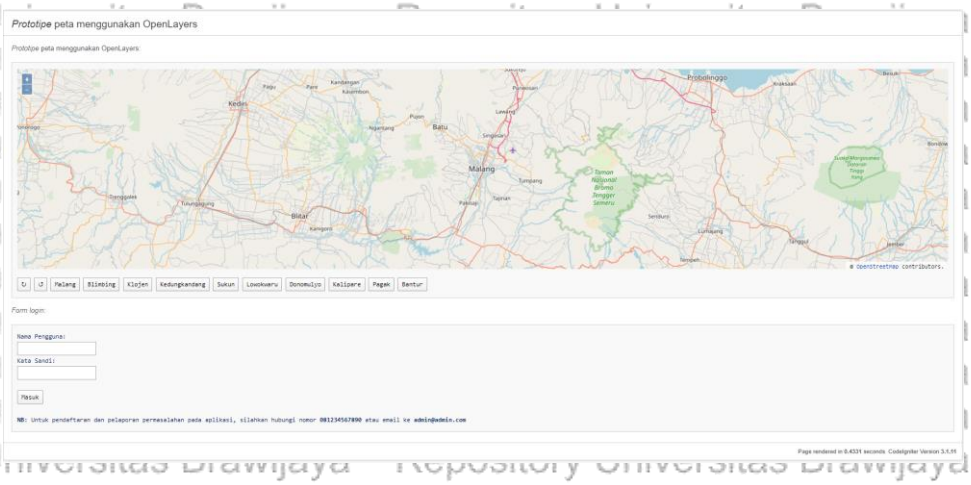
Gambar 5.1 Layanan *load balancer* pada *instance* GCP

   Langkah kedua adalah memastikan layanan telah berjalan di server utama atau server cadangan. Pada pengujian ini layanan dijalankan pada server utama. Layanan pada server utama yang telah berjalan dapat dilihat pada Gambar 5.2.



Gambar 5.2 Layanan pada *instance* server utama

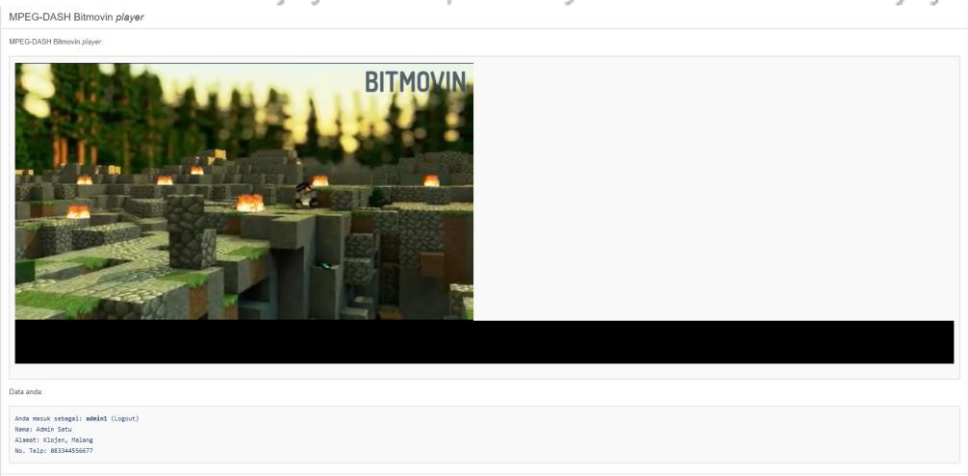
Langkah berikutnya adalah pengguna mengakses alamat IP dari *instance load balancer* yakni 35.345.113.158. Apabila pengguna dapat mengakses layanan yang telah dijalankan pada *instance* server utama melalui IP tersebut, maka pengujian ini dapat dikatakan berhasil.



Gambar 5.3 Halaman utama layanan *instance* server utama

Dapat dilihat pada Gambar 5.3 pengguna dapat mengakses halaman utama dari layanan yang terdapat pada *instance* server utama melalui IP *instance load balancer*. Dengan demikian, dapat disimpulkan bahwa pengujian pada poin ini berhasil dilakukan.

2. [LMF_03] Server yang memiliki layanan mampu memigrasikan layanannya ke server lainnya dan [LMF_04] server yang memiliki layanan mampu menjaga keseluruhan *state* dari layanan sesudah proses *live migration* dilakukan.



Gambar 5.4 Halaman layanan *video streaming*

Pengujian ini dilakukan untuk melihat apakah server yang memiliki layanan mampu memigrasikan layanannya ke server lainnya dan apakah server yang menerima migrasi layanan mampu menjaga keseluruhan *state* layanan sesudah proses *live migration* dilakukan. Pada pengujian ini layanan akan dimigrasikan dari *instance* server utama ke *instance* server

cadangan. Sesuai dengan Tabel 3.11 yang terdapat pada Subbab 3.1.1 Pengujian Fungsional, langkah pertama dan kedua sama dengan poin pengujian fungsional sebelumnya. Langkah ketiga adalah pengguna mengakses alamat IP dari *instance load balancer* dan melakukan *login* pada layanan yang disediakan. Setelah melakukan *login,* pengguna akan diarahkan ke halaman layanan *video streaming* seperti yang ditunjukkan pada Gambar 5.4. Langkah keempat adalah menjalankan *script* penerimaan migrasi pada *instance* server cadangan yang ditunjukkan pada Gambar 5.5.

```
[ec2-user@ip-172-31-81-23 aws]$ ./migrate-server.py

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-

[INFO] Listening on port 12345 ...
```

Gambar 5.5 *Script* penerimaan migrasi server cadangan

Setelah *script* penerimaan migrasi dijalankan pada *instance* server cadangan, langkah kelima adalah menjalankan *script* permintaan migrasi pada *instance* server utama seperti yang ditunjukkan pada Gambar 5.6.

```
[abdulazizm41@gcp-server-1 gcp]$ ./migrate.py

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-

[INFO] Melakukan pre-dump transfer...

[SUCCESS] Transfer selesai dalam 37.8743040562 detik...

[DEBUG] DOWN_TIMESTAMP: 1585140752.87

[INFO] Melakukan dump pada layanan mysql...
3197578d7659cf5a7982ec740734b5ae9315e6598326c6063d48a0d876a184c9

[SUCCESS] Melakukan dump pada layanan mysql (sukses)...

[INFO] Melakukan dump pada layanan webserver...
96ecceb80813d415385abdf5fe38c768087358b1d90ae397e7f2ac7491a8c84d

[SUCCESS] Melakukan dump pada layanan webserver (sukses)...

[INFO] Proses dump selesai dalam 7.99929499626 detik...

[INFO] Melakukan post-dump transfer...

[SUCCESS] Transfer selesai dalam 1.65069508553 detik...

[INFO] Mengirimkan pesan bahwa dump telah selesai dilakukan...

[SUCCESS] Mengirimkan pesan bahwa dump telah selesai dilakukan (sukses)...

[INFO] Memanggil script stop.py...

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-

[INFO] Menghentikan semua container pada podman...
3197578d7659cf5a7982ec740734b5ae9315e6598326c6063d48a0d876a184c9
96ecceb80813d415385abdf5fe38c768087358b1d90ae397e7f2ac7491a8c84d
```

Gambar 5.6 *Script* permintaan migrasi server utama

Setelah *script* permintaan migrasi dijalankan pada *instance* server utama, secara otomatis layanan akan dimigrasikan ke *instance* server cadangan. Proses yang terjadi saat *live migration* berlangsung akan ditampilkan pada layar *instance* server utama dan cadangan. Saat proses *live migration* selesai dilakukan, ditampilkan pesan selesai pada layar *instance* server utama dan cadangan seperti yang ditunjukkan pada Gambar 5.7 dan 5.8.

```
[INFO] Menghentikan semua container pada podman...
3197578d7659cf5a7982ec740734b5ae9315e6598326c6063d48a0d876a184c9
96ecceb80813d415385abdf5fe38c768087358b1d90ae397e7f2ac7491a8c84d

[SUCCESS] Menghentikan semua container pada podman (sukses)...

[INFO] Menghapus semua container pada podman...
3197578d7659cf5a7982ec740734b5ae9315e6598326c6063d48a0d876a184c9
96ecceb80813d415385abdf5fe38c768087358b1d90ae397e7f2ac7491a8c84d

[SUCCESS] Menghapus semua container pada podman (sukses)...

[INFO] Menghapus semua volume pada podman...


[SUCCESS] Menghapus semua volume pada podman (sukses)...

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-

[SUCCESS] Memanggil script stop.py (done)...

[INFO] Membersihkan...

[SUCCESS] Membersihkan (sukses)...

[INFO] Keseluruhan proses selesai dalam 48.2092831135 detik...

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-

[abdulazizm41@gcp-server-1 gcp]$
```

Gambar 5.7 *Script* permintaan migrasi server utama (lanjutan)

```
[INFO] Listening on port 12345 ...

[INFO] Connected with 35.236.204.230:46960

[INFO] Restore layanan webserver..
96ecceb80813d415385abdf5fe38c768087358b1d90ae397e7f2ac7491a8c84d
96ecceb80813d415385abdf5fe38c768087358b1d90ae397e7f2ac7491a8c84d

[SUCCESS] Restore layanan webserver (sukses)...

[INFO] Restore layanan mysql..
3197578d7659cf5a7982ec740734b5ae9315e6598326c6063d48a0d876a184c9
3197578d7659cf5a7982ec740734b5ae9315e6598326c6063d48a0d876a184c9

[SUCCESS] Restore layanan mysql (sukses)...

[DEBUG] UP_TIMESTAMP: 1585140785.42

Proses restore selesai dalam 22.914192915 detik...

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-

[ec2-user@ip-172-31-81-23 aws]$
```

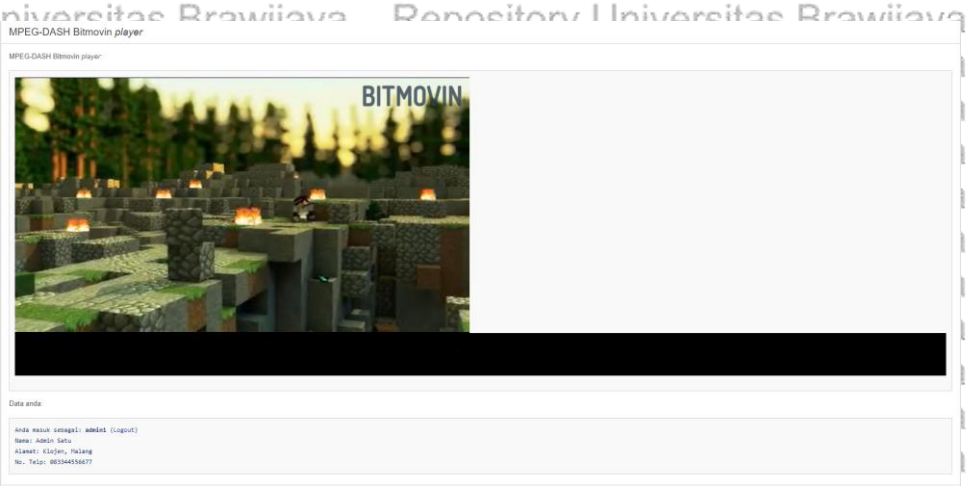
Gambar 5.8 *Script* penerimaan migrasi server cadangan (lanjutan)

Langkah keenam adalah mengecek apakah layanan telah berpindah dan berjalan pada *instance* server cadangan seperti yang ditunjukkan pada Gambar 5.9.

```
[ec2-user@ip-172-31-81-23 aws]$ sudo podman ps
CONTAINER ID  IMAGE                                COMMAND               CRE
TED            STATUS                  PORTS                   NAMES
3197578d7659  docker.io/abdulazizm41/mysql:5.7      mysqld                Abo
t a minute ago  Up About a minute ago  0.0.0.0:3306->3306/tcp  mysql
96ecceb80813  docker.io/abdulazizm41/webserver:7.3.x  apache2-foregroun...  Abo
t a minute ago  Up About a minute ago  0.0.0.0:80->80/tcp      webserver
[ec2-user@ip-172-31-81-23 aws]$
```

Gambar 5.9 Layanan pada *instance* server cadangan

Langkah terakhir adalah pengguna kembali mengakses alamat IP *instance load balancer* yakni 35.345.113.158. Apabila pengguna dapat mengakses layanan yang telah dijalankan pada *instance* server cadangan melalui IP tersebut dan *session* pengguna tetap terjaga, maka pengujian ini dapat dikatakan berhasil.



Gambar 5.10 Halaman layanan *video streaming* (lanjutan)

Dapat dilihat pada Gambar 5.10 pengguna dapat mengakses halaman utama dari layanan yang terdapat pada *instance* server cadangan melalui IP *instance load balancer* dengan *session* pengguna yang masih terjaga. Dengan demikian, dapat disimpulkan bahwa pengujian pada poin ini berhasil dilakukan.

Berdasarkan pengujian fungsional yang telah dilakukan, dapat disimpulkan bahwa sistem yang dibangun telah memenuhi seluruh kebutuhan fungsional yang telah ditetapkan. Analisis hasil pengujian fungsional dapat dilihat pada Tabel 5.1.

Tabel 5.1 Analisis hasil pengujian fungsional

Kode	Kebutuhan Fungsional	Status
LMF_01	*Load balancer* dapat mengarahkan *traffic* permintaan dari klien menuju server yang memiliki layanan.	Sesuai

Tabel 5.1 (lanjutan)

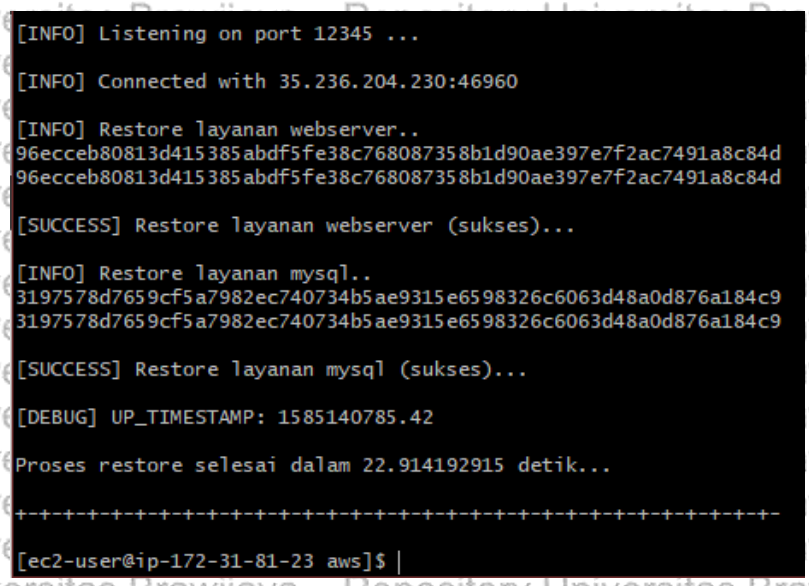
Kode	Kebutuhan Fungsional	Status
LMF_02	Server yang memiliki layanan mampu merespon permintaan dari klien.	Sesuai
LMF_03	Server yang memiliki layanan mampu memigrasikan layanannya ke server lainnya.	Sesuai
LMF_04	Server yang memiliki layanan mampu menjaga keseluruhan *state* dari layanan sesudah proses *live migration* dilakukan.	Sesuai

5.2 Pengujian *Downtime*

Pengujian *downtime* berkaitan dengan durasi layanan dihentikan sepenuhnya pada saat proses migrasi berlangsung. Selama periode ini, layanan tidak dapat diakses oleh pengguna, sehingga berpengaruh pada *Service Level Agreement* (SLA) dari layanan tersebut. Pada sistem yang dibangun, *downtime* mulai dihitung saat proses *dump* dilakukan pada *instance* sumber, hingga proses *restore* selesai dilakukan pada *instance* tujuan. Untuk melakukan pengujian *downtime*, pada *script* yang dibuat telah disisipkan kode untuk menampilkan waktu dan *timestamp* dari setiap proses, seperti yang ditunjukkan pada Gambar 5.11 dan 5.12. Satuan waktu yang digunakan adalah detik. Pengujian *downtime* dilakukan pada tiga lingkungan yang berbeda, yakni lokal *Network File System* (NFS), lokal, dan *cloud*. Masing-masing lingkungan terdiri dari lima skenario pengujian, antara lain skenario pengujian dengan 100, 500, 1000, 5000, dan 10000 baris data. Kemudian, masing-masing skenario diuji sebanyak 10 kali.

```
[abdulazizm41@gcp-server-1 gcp]$ ./migrate.py

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-

[INFO] Melakukan pre-dump transfer...

[SUCCESS] Transfer selesai dalam 37.8743040562 detik...

[DEBUG] DOWN_TIMESTAMP: 1585140752.87

[INFO] Melakukan dump pada layanan mysql...
3197578d7659cf5a7982ec740734b5ae9315e6598326c6063d48a0d876a184c9

[SUCCESS] Melakukan dump pada layanan mysql (sukses)...

[INFO] Melakukan dump pada layanan webserver...
96ecceb80813d415385abdf5fe38c768087358b1d90ae397e7f2ac7491a8c84d

[SUCCESS] Melakukan dump pada layanan webserver (sukses)...

[INFO] Proses dump selesai dalam 7.99929499626 detik...

[INFO] Melakukan post-dump transfer...

[SUCCESS] Transfer selesai dalam 1.65069508553 detik...

[INFO] Mengirimkan pesan bahwa dump telah selesai dilakukan...

[SUCCESS] Mengirimkan pesan bahwa dump telah selesai dilakukan (sukses)...

[INFO] Memanggil script stop.py...

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-

[INFO] Menghentikan semua container pada podman...
3197578d7659cf5a7982ec740734b5ae9315e6598326c6063d48a0d876a184c9
96ecceb80813d415385abdf5fe38c768087358b1d90ae397e7f2ac7491a8c84d
```

Gambar 5.11 DOWN_TIMESTAMP pada *script* permintaan migrasi



Gambar 5.12 UP_TIMESTAMP pada *script* penerimaan migrasi

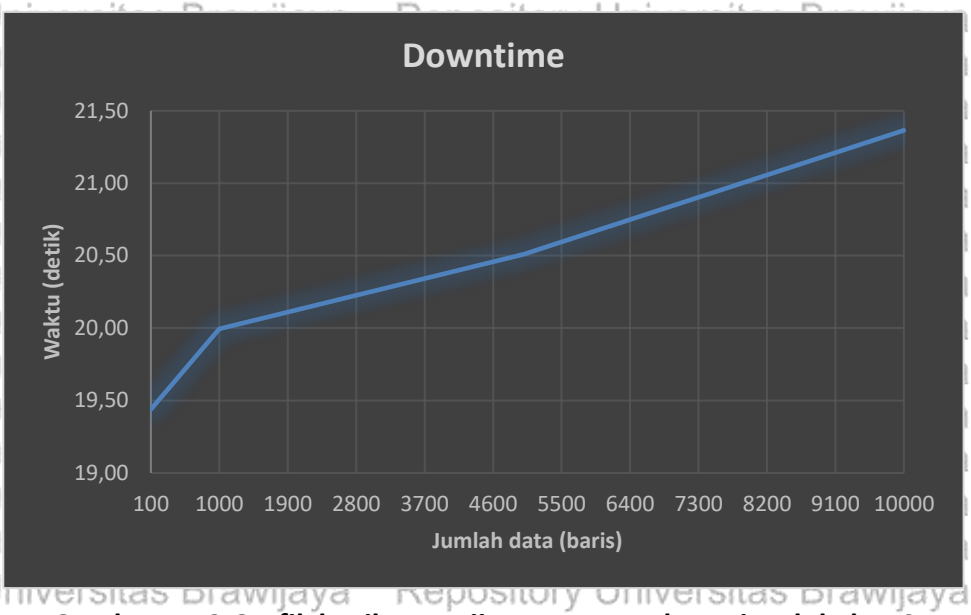
Perhitungan waktu *downtime* didapatkan dengan mengurangkan waktu UP_TIMESTAMP yang terdapat pada *script* penerimaan migrasi dengan waktu DOWN_TIMESTAMP yang terdapat pada *script* permintaan migrasi. DOWN_TIMESTAMP merupakan waktu di mana layanan mulai di *dump* pada *instance* sumber, dan UP_TIMESTAMP merupakan waktu di mana layanan sudah mulai berjalan pada *instance* tujuan.

1. Lokal NFS

   Pengujian pada lokal NFS dilakukan di dua buah mesin virtual Fedora 30 Server x64 yang bertindak sebagai server utama dan server cadangan. Masing-masing mesin virtual memiliki 2vCPUs, 2 GB memori, dan 20 GB *disk* dengan *filesystem* ext4. Rata-rata *bandwidth* antar *instance* sebesar 3.65 Gbits/detik, dihitung menggunakan iperf3. Keepalived digunakan untuk memberikan alamat IP virtual pada kedua mesin virtual agar dapat diakses oleh pengguna. NFS yang digunakan adalah NFSv4. Hasil pengujian rata-rata *downtime* lokal NFS dapat dilihat pada Tabel 5.2 dan direpresentasikan dalam bentuk grafik pada Gambar 5.13. Hasil pengujian *downtime* lokal NFS secara lengkap terdapat pada Lampiran A.

Tabel 5.2 Hasil pengujian rata-rata *downtime* lokal NFS

Jumlah Data (Baris)	Rata-Rata *Downtime* (Detik)
100	19,4390000343323
500	19,6859999656677
1000	19,9950000047683
5000	20,5089999675751
10000	21,3650000333786



Gambar 5.13 Grafik hasil pengujian rata-rata *downtime* lokal NFS

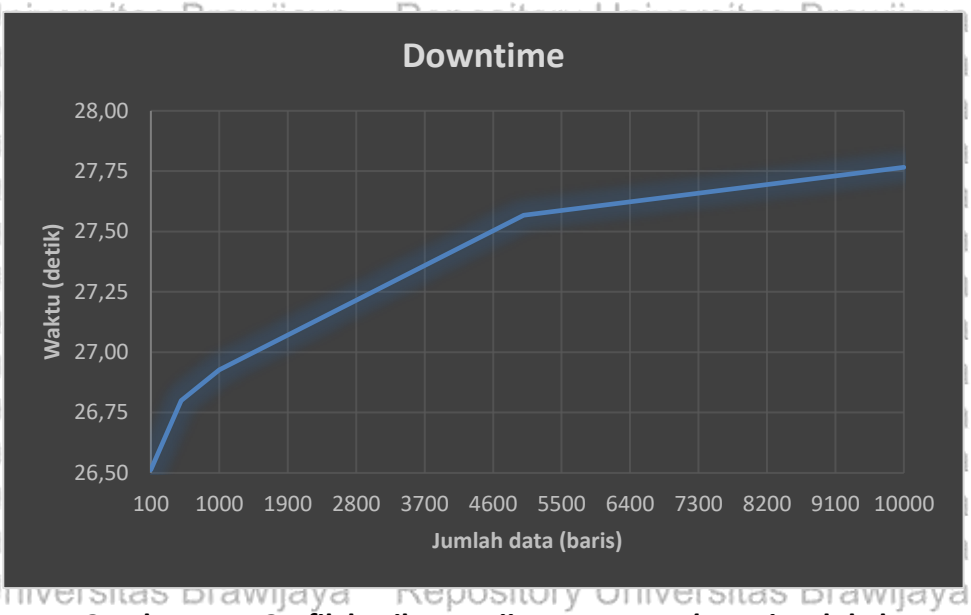
Hasil pengujian rata-rata *downtime* lokal NFS untuk jumlah data 100, 500, 1000, 5000, dan 10000 secara berturut-turut adalah 19,439, 19,686, 19,995, 20,509, dan 21,365 detik. Dapat dilihat bahwa nilai *dowtime* terus meningkat seiring dengan bertambahnya jumlah data. Hal ini dapat terjadi karena dengan bertambahnya jumlah data, maka waktu yang diperlukan untuk melakukan proses *dump* layanan dan proses *restore* layanan menjadi lebih lama.

2. Lokal

Pengujian pada lokal dilakukan di dua buah mesin virtual Fedora 30 Server x64 yang bertindak sebagai server utama dan server cadangan. Masing-masing mesin virtual memiliki 2vCPUs, 2 GB memori, dan 20 GB *disk* dengan *filesystem* ext4. Rata-rata *bandwidth* antar *instance* sebesar 3.65 Gbits/detik, dihitung menggunakan iperf3. Keepalived digunakan untuk memberikan alamat IP virtual pada kedua mesin virtual agar dapat diakses oleh pengguna. Hasil pengujian rata-rata *downtime* lokal dapat dilihat pada Tabel 5.3 dan direpresentasikan dalam bentuk grafik pada Gambar 5.14. Hasil pengujian *downtime* lokal secara lengkap terdapat pada Lampiran B.

Tabel 5.3 Hasil pengujian rata-rata *downtime* lokal

Jumlah Data (Baris)	Rata-Rata *Downtime* (Detik)
100	26,5130000114440
500	26,7990000724792
1000	26,9269999980927
5000	27,5669999837875
10000	27,7660000085831



Gambar 5.14 Grafik hasil pengujian rata-rata *downtime* lokal

Hasil pengujian rata-rata *downtime* lokal untuk jumlah data 100, 500, 1000, 5000, dan 10000 secara berturut-turut adalah 26,513, 26,799, 26,927, 27,567, dan 27,766 detik. Dapat dilihat bahwa nilai *dowtime* terus meningkat seiring dengan bertambahnya jumlah data. Hal ini dapat terjadi karena dengan bertambahnya jumlah data, maka waktu yang diperlukan untuk melakukan proses *dump* layanan dan proses *restore* layanan menjadi lebih lama. Selain itu, rata-rata waktu *downtime* yang didapatkan lebih besar jika dibandingkan dengan hasil pengujian rata-rata *downtime* lokal NFS. Hal tersebut dapat terjadi karena adanya penambahan waktu dari proses transfer data dari server sumber ke server cadangan. Semakin besar data yang ditransfer, semakin lama pula waktu transfer yang dibutuhkan.

3. *Cloud*

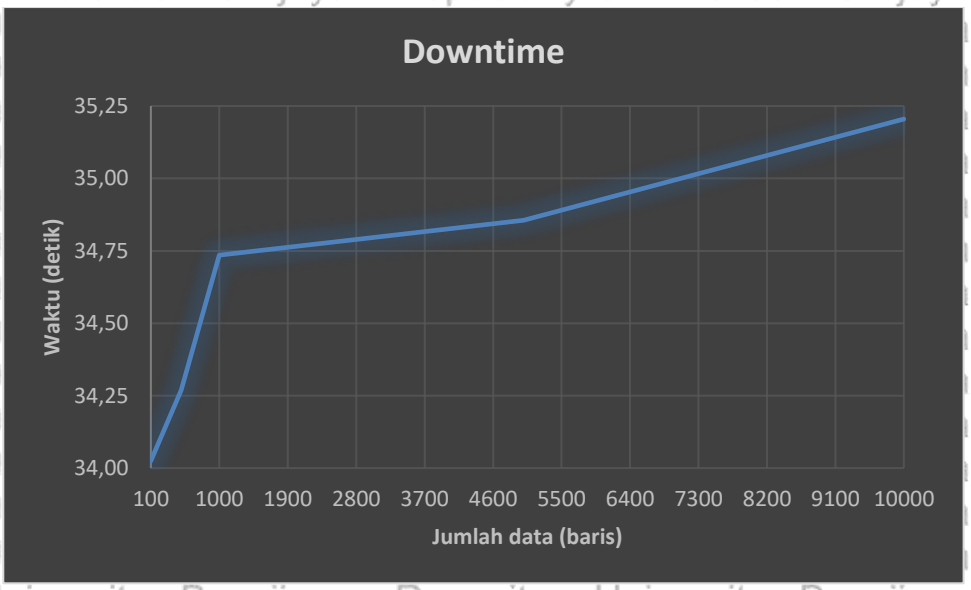
Pengujian pada *cloud* dilakukan pada *instance* server utama yang terletak pada GCP dengan spesifikasi yang dapat dilihat pada Tabel 3.4 Subbab 3.2.4 Perancangan Server Utama dan *instance* server cadangan yang terletak pada AWS dengan spesifikasi yang dapat dilihat pada Tabel 3.6 Subbab 3.2.5 Perancangan Server Cadangan. Rata-rata *bandwidth* antar *instance* sebesar 1.75 Gbits/detik, dihitung menggunakan iperf3. Hasil pengujian rata-rata *downtime cloud* dapat dilihat pada Tabel 5.4 dan direpresentasikan dalam bentuk grafik pada Gambar 5.15. Hasil pengujian *downtime cloud* secara lengkap terdapat pada Lampiran C.

Tabel 5.4 Hasil pengujian rata-rata *downtime cloud*

Jumlah Data (Baris)	Rata-Rata *Downtime* (Detik)
100	34,0249999761581
500	34,2690000295639

Tabel 5.4 (lanjutan)

Jumlah Data (Baris)	Rata-Rata *Downtime* (Detik)
1000	34,7349999904633
5000	34,8560000181198
10000	35,2050000190735



Gambar 5.15 Grafik hasil pengujian rata-rata *downtime cloud*

Hasil pengujian rata-rata *downtime cloud* untuk jumlah data 100, 500, 1000, 5000, dan 10000 secara berturut-turut adalah 34,025, 34,269, 34,735, 34,856, dan 35,205 detik. Dapat dilihat bahwa nilai *dowtime* terus meningkat seiring dengan bertambahnya jumlah data. Hal ini dapat terjadi karena dengan bertambahnya jumlah data, maka waktu yang diperlukan untuk melakukan proses *dump* layanan dan proses *restore* layanan menjadi lebih lama. Selain itu, rata-rata waktu *downtime* yang didapatkan lebih besar jika dibandingkan dengan hasil pengujian rata-rata *downtime* lokal dan lokal NFS. Hal tersebut dapat terjadi karena *bandwidth* antar *instance* pada *cloud* lebih kecil, sehingga terdapat penambahan waktu dari proses transfer data dari server sumber ke server cadangan.

Pengujian yang dilakukan oleh Nadgowda, et al. dengan judul "Voyager: *Complete Container State Migration*" mendapatkan hasil pengujian rata-rata *downtime* sekitar 2-3 detik. Hasil ini cukup rendah jika dibandingkan dengan hasil pengujian rata-rata *downtime* pada penelitian implementasi *container live migration* antar-*cloud provider* menggunakan Podman dan CRIU. Hal ini dapat terjadi karena beberapa alasan, antara lain:

- Tidak terdapat proses optimasi pada penelitian implementasi *container live migration* antar-*cloud provider* menggunakan Podman dan CRIU, sedangkan pada penelitian yang dilakukan oleh Nadgowda, et al. menggunakan teknik *lazy migration* digabungkan dengan *remote page server,* sehingga mengurangi waktu transfer data antar *instance*.
- Kemampuan sumber daya perangkat keras yang berbeda. Penelitian yang dilakukan oleh Ma, et al. menyatakan bahwa ketika tingkat transfer data jaringan menjadi tinggi, kemampuan CPU yang digunakan untuk melakukan kompresi, dan penyimpanan *disk* mesin menjadi *performance bottlenecks*.
- Perbedaan ukuran data layanan yang dipindahkan dari server sumber ke server tujuan.

5.3 Pengujian *Migration Time*

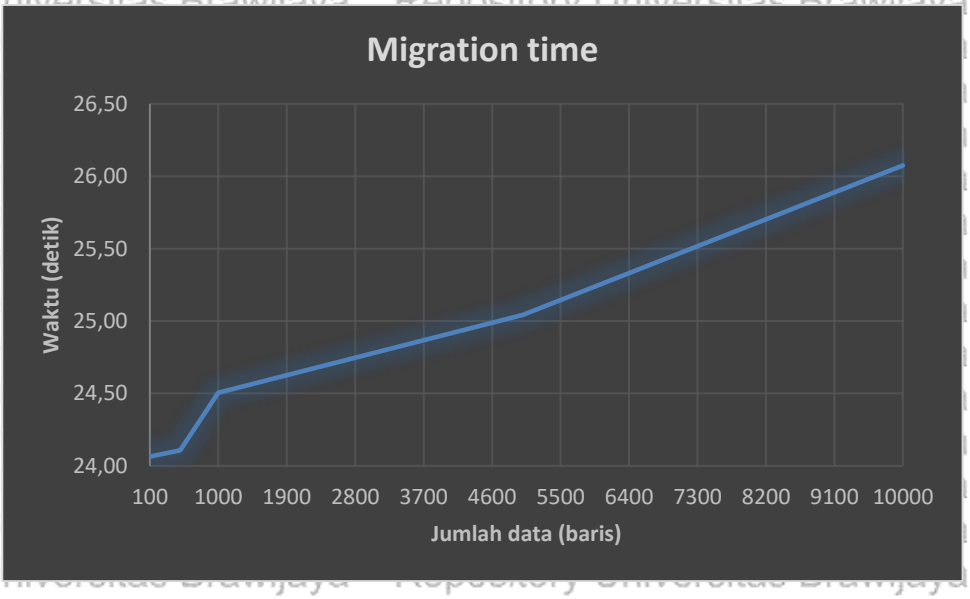
Pengujian *migration time* merupakan perhitungan total durasi mulai dari proses migrasi dilakukan pada *instance* sumber, hingga layanan berhasil dipulihkan dan berjalan pada *instance* tujuan. Pada sistem yang dibangun, *migration time* mulai dihitung saat proses *pre dump transfer* dilakukan pada *instance* sumber, hingga proses *restore* selesai dilakukan pada *instance* tujuan. Untuk melakukan pengujian *migration time*, pada *script* yang dibuat telah disisipkan kode untuk menampilkan waktu dan *timestamp* dari setiap proses, seperti yang ditunjukkan pada Gambar 5.11 dan 5.12. Satuan waktu yang digunakan adalah detik. Sama halnya dengan pengujian *downtime*, pengujian *migration time* dilakukan pada tiga lingkungan yang berbeda, yakni lokal *Network File System* (NFS), lokal, dan *cloud*. Masing-masing lingkungan terdiri dari lima skenario pengujian, antara lain skenario pengujian dengan 100, 500, 1000, 5000, dan 10000 baris data. Kemudian, masing-masing skenario diuji sebanyak 10 kali. Perhitungan waktu *migration time* didapatkan dengan menjumlahkan total waktu yang diperlukan oleh *script* permintaan migrasi untuk selesai, dengan total waktu yang diperlukan oleh *script* penerimaan migrasi untuk selesai.

1. Lokal NFS

   Pengujian pada lokal NFS dilakukan di dua buah mesin virtual Fedora 30 Server x64 yang bertindak sebagai server utama dan server cadangan. Masing-masing mesin virtual memiliki 2vCPUs, 2 GB memori, dan 20 GB *disk* dengan *filesystem* ext4. Rata-rata *bandwidth* antar *instance* sebesar 3.65 Gbits/detik, dihitung menggunakan iperf3. Keepalived digunakan untuk memberikan alamat IP virtual pada kedua mesin virtual agar dapat diakses oleh pengguna. NFS yang digunakan adalah NFSv4. Hasil pengujian rata-rata *migration time* lokal NFS dapat dilihat pada Tabel 5.5 dan direpresentasikan dalam bentuk grafik pada Gambar 5.16. Hasil pengujian *migration time* lokal NFS secara lengkap terdapat pada Lampiran A.

Tabel 5.5 Hasi pengujian rata-rata *migration time* lokal NFS

Jumlah Data (Baris)	Rata-Rata *Migration Time* (Detik)
100	24,064440965653
500	24,106957244891
1000	24,504961228382
5000	25,041837167753
10000	26,074268651015



Gambar 5.16 Grafik hasil pengujian rata-rata *migration time* lokal NFS

Hasil pengujian rata-rata *migration time* lokal NFS untuk jumlah data 100, 500, 1000, 5000, dan 10000 secara berturut-turut adalah 24,064, 24,107, 24,505, 25,042, dan 26,074 detik. Dapat dilihat bahwa nilai *migration time* terus meningkat seiring dengan bertambahnya jumlah data. Hal ini dapat terjadi karena dengan bertambahnya jumlah data, maka waktu yang diperlukan untuk melakukan proses *dump* layanan dan proses *restore* layanan menjadi lebih lama.

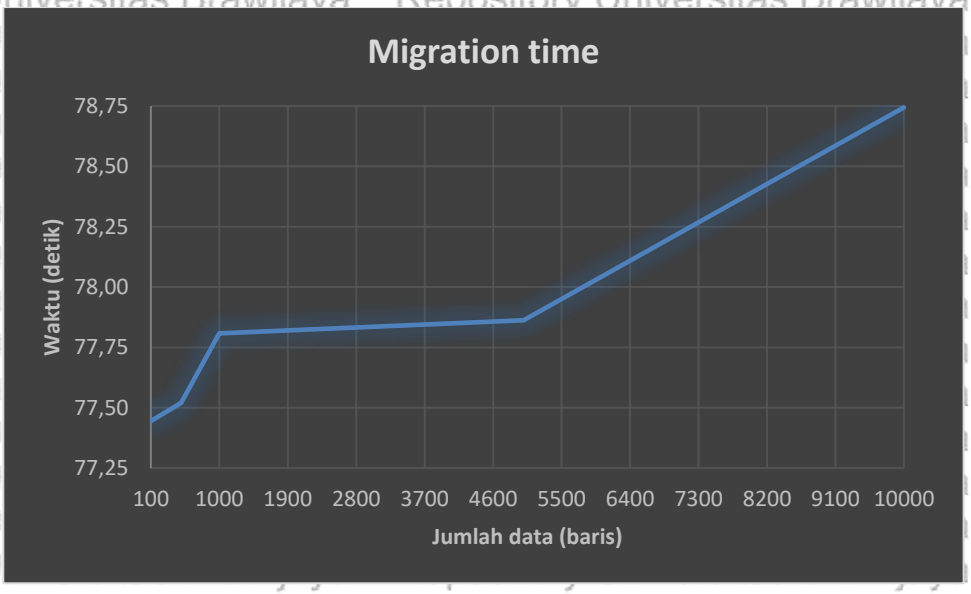
2. Lokal

Pengujian pada lokal dilakukan di dua buah mesin virtual Fedora 30 Server x64 yang bertindak sebagai server utama dan server cadangan. Masing-masing mesin virtual memiliki 2vCPUs, 2 GB memori, dan 20 GB *disk* dengan *filesystem* ext4. Rata-rata *bandwidth* antar *instance* sebesar 3.65 Gbits/detik, dihitung menggunakan iperf3. Keepalived digunakan untuk memberikan alamat IP virtual pada kedua mesin virtual agar dapat diakses oleh pengguna. Hasil pengujian rata-rata *migration time* lokal dapat dilihat pada Tabel 5.6 dan direpresentasikan dalam bentuk grafik

pada Gambar 5.17. Hasil pengujian *migration time* lokal secara lengkap terdapat pada Lampiran B.

Tabel 5.6 Hasil pengujian rata-rata *migration time* lokal

Jumlah Data (Baris)	Rata-Rata *Migration Time* (Detik)
100	77,444750618920
500	77,520793223410
1000	77,808551669120
5000	77,862855716680
10000	78,743962574020



Gambar 5.17 Grafik hasil pengujian rata-rata *migration time* lokal

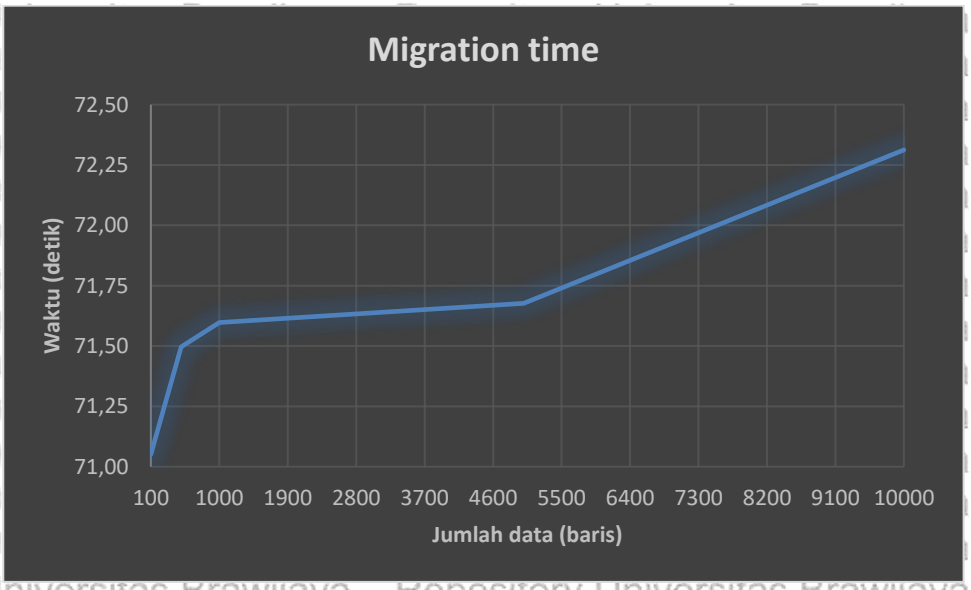
Hasil pengujian rata-rata *migration time* lokal untuk jumlah data 100, 500, 1000, 5000, dan 10000 secara berturut-turut adalah 77,445, 77,521, 77,809, 77,863, dan 78,744 detik. Dapat dilihat bahwa nilai *migration time* terus meningkat seiring dengan bertambahnya jumlah data. Hal ini dapat terjadi karena dengan bertambahnya jumlah data, maka waktu yang diperlukan untuk melakukan proses *dump* layanan dan proses *restore* layanan menjadi lebih lama. Selain itu, rata-rata waktu *migration time* yang didapatkan jauh lebih besar jika dibandingkan dengan hasil pengujian rata-rata *migration time* lokal NFS. Hal tersebut dapat terjadi karena adanya penambahan waktu dari proses transfer data dari server sumber ke server cadangan. Semakin besar data yang ditransfer, semakin lama pula waktu transfer yang dibutuhkan. Selain itu, seperti yang disampaikan pada penelitian yang dilakukan oleh Ma, et al. bahwasanya kemampuan perangkat keras juga menentukan kecepatan dari proses transfer yang dilakukan.

3. *Cloud*

Pengujian pada *cloud* dilakukan pada *instance* server utama yang terletak pada GCP dengan spesifikasi yang dapat dilihat pada Tabel 3.4 Subbab 3.2.4 Perancangan Server Utama dan *instance* server cadangan yang terletak pada AWS dengan spesifikasi yang dapat dilihat pada Tabel 3.6 Subbab 3.2.5 Perancangan Server Cadangan. Rata-rata *bandwidth* antar *instance* sebesar 1.75 Gbits/detik, dihitung menggunakan iperf3. Hasil pengujian rata-rata *migration time cloud* dapat dilihat pada Tabel 5.7 dan direpresentasikan dalam bentuk grafik pada Gambar 5.18. Hasil pengujian *migration time cloud* secara lengkap terdapat pada Lampiran C.

Tabel 5.7 Hasil pengujian rata-rata *migration time cloud*

Jumlah Data (Baris)	Rata-Rata *Migration Time* (Detik)
100	71,051493048680
500	71,496828556040
1000	71,597207474700
5000	71,676707625370
10000	72,312255358690



Gambar 5.18 Grafik hasil pengujian rata-rata *migration time cloud*

Hasil pengujian rata-rata *migration time cloud* untuk jumlah data 100, 500, 1000, 5000, dan 10000 secara berturut-turut adalah 71,051, 71,497, 71,597, 71,677, dan 72,312 detik. Dapat dilihat bahwa nilai *migration time* terus meningkat seiring dengan bertambahnya jumlah data. Hal ini dapat terjadi karena dengan bertambahnya jumlah data, maka waktu yang diperlukan untuk melakukan proses *dump* layanan dan proses

restore layanan menjadi lebih lama. Selain itu, rata-rata waktu *migration time* yang didapatkan jauh lebih besar jika dibandingkan dengan hasil pengujian rata-rata *migration time* lokal NFS. Hal tersebut dapat terjadi karena *bandwidth* antar *instance* pada *cloud* lebih kecil, sehingga terdapat penambahan waktu dari proses transfer data dari server sumber ke server cadangan. Namun, jika dibandingkan dengan hasil pengujian rata-rata *migration time* lokal, hasil pengujian yang didapatkan sedikit lebih kecil. Hal ini terjadi karena kemampuan perangkat keras *cloud* sedikit lebih baik dibandingkan dengan kemampuan perangkat keras lokal.

Jika diamati dengan seksama, hasil pengujian rata-rata *downtime* memiliki kaitan dengan hasil pengujian rata-rata *migration time*. Semakin lama waktu *downtime* yang terjadi, maka semakin lama pula waktu *migration time*. Selain itu, kemampuan perangkat keras dan ukuran data layanan yang dipindahkan juga memengaruhi durasi *migration time* yang terjadi.

BAB 6 PENUTUP

6.1 Kesimpulan

Berdasarkan perancangan dan implementasi yang telah dilakukan, serta hasil pengujian yang diperoleh, maka dapat disimpulkan bahwa:

1. Implementasi *live migration* antar-*cloud provider* menggunakan Podman dan CRIU terbukti mampu memindahkan layanan antar-*cloud provider*, tanpa harus memindahkan keseluruhan *state* dari mesin virtual sumber ke mesin virtual tujuan, menggunakan Podman dan CRIU. Dengan menggunakan Podman, sebuah layanan dapat dijalankan berupa proses terisolasi layaknya mesin virtual ringan, yang independen terhadap lingkungannya. Dan dengan CRIU, proses *live migration* dapat dilakukan. CRIU mampu melakukan *checkpoint* dan *restore* dari sebuah proses. *Checkpoint* membuat *snapshot state* lengkap dari sebuah proses, termasuk konten memori, *file descriptor*, dan koneksi TCP yang sedang berlangsung, menjadi berkas-berkas. Berkas-berkas tersebut kemudian dipindahkan dan di *restore* pada *instance* tujuan.
2. Hasil pengujian rata-rata *downtime* lokal NFS untuk jumlah data 100, 500, 1000, 5000, dan 10000 secara berturut-turut adalah 19,439, 19,686, 19,995, 20,509, dan 21,365 detik. Hasil pengujian rata-rata *downtime* lokal untuk jumlah data 100, 500, 1000, 5000, dan 10000 secara berturut-turut adalah 26,513, 26,799, 26,927, 27,567, dan 27,766 detik. Hasil pengujian rata-rata *downtime cloud* untuk jumlah data 100, 500, 1000, 5000, dan 10000 secara berturut-turut adalah 34,025, 34,269, 34,735, 34,856, dan 35,205 detik.
3. Hasil pengujian rata-rata *migration time* lokal NFS untuk jumlah data 100, 500, 1000, 5000, dan 10000 secara berturut-turut adalah 24,064, 24,107, 24,505, 25,042, dan 26,074 detik. Hasil pengujian rata-rata *migration time* lokal untuk jumlah data 100, 500, 1000, 5000, dan 10000 secara berturut-turut adalah 77,445, 77,521, 77,809, 77,863, dan 78,744 detik. Hasil pengujian rata-rata *migration time cloud* untuk jumlah data 100, 500, 1000, 5000, dan 10000 secara berturut-turut adalah 71,051, 71,497, 71,597, 71,677, dan 72,312 detik.
4. Waktu rata-rata durasi *downtime* berbanding lurus dengan waktu rata-rata durasi *migration time*.
5. Proses optimasi, kemampuan perangkat keras, dan ukuran data layanan berpengaruh terhadap durasi *downtime* dan *migration time* yang terjadi.

6.2 Saran

Berdasarkan hasil kesimpulan yang diperoleh, maka direkomendasikan beberapa saran untuk penelitian berikutnya, antara lain:

1. Menambahkan metode optimasi dalam proses *live migration* untuk mengurangi durasi waktu *downtime* dan *migration time* yang terjadi. Beberapa metode optimasi yang dapat digunakan, yaitu *iterative migration*, *disk-less migration*, dan *lazy migration*.
2. Menambahkan variasi parameter pengujian untuk mendapatkan hasil pengujian yang lebih variatif. Beberapa parameter pengujian yang dapat digunakan, yakni pengujian *overhead* dan *consistency*.
3. Menggunakan *container engine* lain selain Podman sebagai perbandingan performa kinerja dalam melakukan *live migration*. Beberapa *container engine* yang dapat digunakan, antara lain OpenVZ, LXC/LXD, dan Docker.

DAFTAR REFERENSI

Bernstein, D., 2014. Containers and Cloud: From LXC to Docker to Kubernetes. *IEEE Cloud Computing,* 1(3), pp. 81-84.

Birje, M. N., Challagidad, P. S., Goudar, R. & Tapale, M. T., 2017. Cloud computing review: concept, technology, challenges and security. *International Journal of Cloud Computing,* 6(1), p. 32.

CRIU, 2019. *CRIU.* [Online]
Available at: https://criu.org/Main_Page
[Accessed 11 September 2019].

Dawodi, M., Hedayati, D. M. H., Baktash, D. J. A. & Erfan, A. L., 2019. *Facebook MySQL Performance vs MySQL Performance.* Vancouver, IEEE.

Dua, R., Raja, A. R. & Kakadia, D., 2014. *Virtualization vs Containerization to support PaaS.* Boston, IEEE.

Elliott, D., Otero, C., Ridley, M. & Merino, X., 2018. *A Cloud-Agnostic Container Orchestrator for Improving Interoperability.* San Fransisco, IEEE.

Fan, W., Han, Z., Zhang, Y. & Wang, R., 2018. *A Locality Live Migration Strategy Based on Docker Containers.* Omaha, IEEE.

Govindaraj, K. & Artemenko, A., 2018. *Container Live Migration for Latency Critical Industrial Applications on Edge Computing.* Turin, IEEE.

Huang, C.-H. & Lee, C.-R., 2017. *Enhancing the Availability of Docker Swarm Using Checkpoint-and-Restore.* Exeter, IEEE.

Mahfouz, A. M., Rahman, M. L. & Shiva, S. G., 2017. *Secure Live Virtual Machine Migration through Runtime Monitors.* Noida, IEEE.

Ma, L., Yi, S., Carter, N. & Li, Q., 2019. Efficient Live Migration of Edge Services Leveraging Container Layered Storage. *IEEE Transactions on Mobile Computing,* 18(9), pp. 2020-2033.

Mirkin, A., Kuznetsov, A. & Kolyshkin, K., 2008. *Containers checkpointing and live migration.* Ottawa, Proceedings of the Linux Symposium.

Nadgowda, S., Suneja, S., Bila, N. & Isci, C., 2017. *Voyager: Complete Container State Migration.* Atlanta, IEEE.

Naik, N., 2016. *Applying Computational Intelligence for enhancing the dependability of multi-cloud systems using Docker Swarm.* Athens, IEEE.

Podman, 2019. *Podman.* [Online]
Available at: https://podman.io/whatis.html
[Accessed 20 December 2019].

Rashid, A. & Chatuverdi, A., 2019. Virtualization and its Role in Cloud Computing Environment. *International Journal of Computer Sciences and Engineering,* 7(4), pp. 1131-1136.

Red Hat Developer, 2019. *Podman and Buildah for Docker users.* [Online] Available at: https://developers.redhat.com/blog/2019/02/21/podman-and-buildah-for-docker-users/ [Accessed 22 December 2019].

Red Hat Enterprise Linux, 2019. *Red Hat Enterprise Linux.* [Online] Available at: https://www.redhat.com/en/technologies/linux-platforms/enterprise-linux [Accessed 10 Juli 2020].

Srivastava, P. & Khan, R., 2018. A Review Paper on Cloud Computing. *International Journal of Advanced Research in Computer Science and Software Engineering,* 8(6), pp. 17-20.

Tandel, P., Parmar, P. A. & Shah, P. J., 2019. *An Analysis of Live VM Migration Based on OpenStack Cloud: A Survey.* Tirunelveli, IEEE.

LAMPIRAN A HASIL PENGUJIAN LOKAL NFS

Lokal NFS (625 MB; 201 MB [100 *row*]; 826 MB)							*Downtime*	*Migration time*
Pengujian ke-	**Sumber**	**Tujuan**	**DOWN_TIMESTAMP**	**Total waktu (sumber)**	**UP_TIMESTAMP**	**Total waktu (tujuan)**		
1	server1	server2	1587295379,44	14,6383800507	1587295398,79	9,09018898010	19,3499999046325	23,728569030800
2	server2	server1	1587295453,34	15,4816200733	1587295472,63	8,91981506350	19,2900002002716	24,401435136800
3	server1	server2	1587295527,70	15,1337320805	1587295547,44	9,45312404633	19,7400000095367	24,586856126830
4	server2	server1	1587295590,85	14,1296830177	1587295609,98	9,88091897964	19,1300001144409	24,010601997340
5	server1	server2	1587295662,84	13,9853961468	1587295681,35	8,94374299049	18,5099999904632	22,929139137290
6	server2	server1	1587295730,09	14,6714570522	1587295749,31	8,98033308983	19,2200000286102	23,651790142030
7	server1	server2	1587295808,12	14,5393760204	1587295827,71	9,33244419098	19,5900001525878	23,871820211380
8	server2	server1	1587295875,07	15,1676402092	1587295894,84	9,53748488426	19,7699999809265	24,705125093460
9	server1	server2	1587295954,10	14,2162718773	1587295974,09	10,54177999500	19,9900000095367	24,758051872300
10	server2	server1	1587296025,40	13,5116999149	1587296045,20	10,48932099340	19,7999999523162	24,001020908300
Rata-rata							19,4390000343323	24,064440965653

Lokal NFS (625 MB; 201 MB [500 *row*]; 826 MB)							***Downtime***	***Migration time***
Pengujian ke-	**Sumber**	**Tujuan**	**DOWN_TIMESTAMP**	**Total waktu (sumber)**	**UP_TIMESTAMP**	**Total waktu (tujuan)**		
1	server1	server2	1587296506,47	15,8340198994	1587296526,23	8,21525216103	19,7599999904632	24,049272060430
2	server2	server1	1587296589,67	13,9054219723	1587296608,59	9,63924884796	18,9199998378753	23,544670820260
3	server1	server2	1587296652,45	14,4792711735	1587296672,03	9,29317808151	19,5799999237060	23,772449255010
4	server2	server1	1587296720,20	14,3321039677	1587296739,86	10,12352013590	19,6599998474121	24,455624103600
5	server1	server2	1587296789,97	15,4728960991	1587296809,90	9,51085901260	19,9300000667572	24,983755111700
6	server2	server1	1587296848,79	13,4879679680	1587296868,06	9,90970110890	19,2699999809265	23,397669076900
7	server1	server2	1587296920,61	12,7430188656	1587296940,57	10,10834193230	19,9600000381469	22,851360797900
8	server2	server1	1587296982,98	15,2422840595	1587297002,79	9,63804316521	19,8099999427795	24,880327224710
9	server1	server2	1587297039,93	14,8493649960	1587297059,45	9,37553095820	19,5199999809265	24,224895954200
10	server2	server1	1587297089,05	15,2936530113	1587297109,50	9,61589503290	20,4500000476837	24,909548044200
Rata-rata							19,6859999656677	24,106957244891

Lokal NFS (625 MB; 201 MB [1000 *row*]; 826 MB)							***Downtime***	***Migration time***
Pengujian ke-	**Sumber**	**Tujuan**	**DOWN_TIMESTAMP**	**Total waktu (sumber)**	**UP_TIMESTAMP**	**Total waktu (tujuan)**		
1	server1	server2	1587298482,71	14,0639040470	1587298502,20	10,77462506290	19,4900000095367	24,838529109900
2	server2	server1	1587298551,45	14,7720940113	1587298571,28	9,43033289909	19,8299999237060	24,202426910390
3	server1	server2	1587298616,84	15,0059709549	1587298636,62	9,46953797340	19,7799999713897	24,475508928300
4	server2	server1	1587298678,77	14,7222480774	1587298699,05	10,48726511000	20,2799999713897	25,209513187400
5	server1	server2	1587298728,95	15,7752439976	1587298749,93	9,99209213257	20,9800000190734	25,767336130170
6	server2	server1	1587298793,34	14,9311408997	1587298813,46	9,98296809200	20,1200001239776	24,914108991700
7	server1	server2	1587298853,11	13,1494450569	1587298873,46	10,23118400570	20,3500001430511	23,380629062600
8	server2	server1	1587298904,15	14,2526650429	1587298923,44	9,51665782928	19,2899999618530	23,769322872180
9	server1	server2	1587298955,50	15,1158139706	1587298975,69	9,31728601460	20,1900000572204	24,433099985200
10	server2	server1	1587299003,64	14,4158751965	1587299023,28	9,64326190948	19,6399998664855	24,059137105980
Rata-rata							19,9950000047683	24,504961228382

Lokal NFS (625 MB; 202 MB [5000 *row*]; 827 MB)							*Downtime*	*Migration time*
Pengujian ke-	**Sumber**	**Tujuan**	**DOWN_TIMESTAMP**	**Total waktu (sumber)**	**UP_TIMESTAMP**	**Total waktu (tujuan)**		
1	server1	server2	1587300896,49	14,4500069618	1587300917,51	10,82143998150	21,0199999809265	25,271446943300
2	server2	server1	1587300962,72	15,0800080299	1587300984,12	10,53518986700	21,3999998569488	25,615197896900
3	server1	server2	1587301023,58	15,2784779072	1587301043,87	9,42310976982	20,2899999618530	24,701587677020
4	server2	server1	1587301079,85	14,1440160275	1587301100,57	11,12412595750	20,7200000286102	25,268141985000
5	server1	server2	1587301147,63	15,1380338669	1587301167,74	10,10476398470	20,1099998950958	25,242797851600
6	server2	server1	1587301206,21	15,8739278316	1587301227,33	9,75137996674	21,1199998855590	25,625307798340
7	server1	server2	1587301261,76	15,1712439060	1587301282,17	9,91558098793	20,4100000858306	25,086824893930
8	server2	server1	1587301316,74	14,2594149113	1587301337,57	11,40963983540	20,8299999237060	25,669054746700
9	server1	server2	1587301369,95	13,8716719151	1587301389,71	10,35665297510	19,7599999904632	24,228324890200
10	server2	server1	1587301419,52	14,4006140232	1587301438,95	9,30907297134	19,4300000667572	23,709686994540
Rata-rata							20,5089999675751	25,041837167753

Lokal NFS (625 MB; 219 MB [10000 *row*]; 844 MB)							***Downtime***	***Migration time***
Pengujian ke-	**Sumber**	**Tujuan**	**DOWN_TIMESTAMP**	**Total waktu (sumber)**	**UP_TIMESTAMP**	**Total waktu (tujuan)**		
1	server1	server2	1587304174,48	15,0410699844	1587304197,29	11,72824597360	22,8099999427795	26,769315958000
2	server2	server1	1587304233,03	14,5231559277	1587304254,52	11,29012107850	21,4900000095367	25,813277006200
3	server1	server2	1587304290,82	15,9803810120	1587304312,34	10,60515499110	21,5199999809265	26,585536003100
4	server2	server1	1587304345,20	15,1516060829	1587304368,53	12,21352195740	23,3299999237060	27,365128040300
5	server1	server2	1587304397,32	14,6485190392	1587304417,03	9,97538399696	19,7100000381469	24,623903036160
6	server2	server1	1587304452,78	14,6287231445	1587304474,17	11,55511617660	21,3900001049041	26,183839321100
7	server1	server2	1587304505,33	14,8186590672	1587304524,88	9,58473300934	19,5500001907348	24,403392076540
8	server2	server1	1587304553,05	15,0454030037	1587304575,54	12,59344100950	22,4900000095367	27,638844013200
9	server1	server2	1587304613,86	16,1216590405	1587304634,59	9,44473314285	20,7300000190734	25,566392183350
10	server2	server1	1587304659,75	14,7534818649	1587304680,38	11,03957700730	20,6300001144409	25,793058872200
Rata-rata							21,3650000333786	26,074268651015

LAMPIRAN B HASIL PENGUJIAN LOKAL

Lokal (625 MB; 201 MB [100 *row*]; 826 MB)							*Downtime*	*Migration time*
Pengujian ke-	**Sumber**	**Tujuan**	**DOWN_TIMESTAMP**	**Total waktu (sumber)**	**UP_TIMESTAMP**	**Total waktu (tujuan)**		
1	server1	server2	1587355426,38	64,6364848614	1587355452,99	14,71610903740	26,6099998950958	79,352593898800
2	server2	server1	1587355567,77	65,5880501270	1587355593,50	12,91274809840	25,7300000190734	78,500798225400
3	server1	server2	1587355676,48	64,2107348442	1587355702,81	13,23966908450	26,3299999237060	77,450403928700
4	server2	server1	1587355782,65	62,7845051289	1587355810,01	14,33791399000	27,3599998950958	77,122419118900
5	server1	server2	1587355887,38	63,7235331535	1587355913,89	12,44710111620	26,5099999904632	76,170634269700
6	server2	server1	1587356011,73	64,9600329399	1587356038,60	15,00722384450	26,8699998855590	79,967256784400
7	server1	server2	1587356128,29	67,0305318832	1587356153,90	13,81188893320	25,6100001335144	80,842420816400
8	server2	server1	1587356344,62	61,7940800190	1587356371,38	15,86049008370	26,7600002288818	77,654570102700
9	server1	server2	1587356445,84	59,1607890129	1587356473,02	14,67378306390	27,1800000667572	73,834572076800
10	server2	server1	1587356545,96	60,0397570133	1587356572,13	13,51207995410	26,1700000762939	73,551836967400
Rata-rata							26,5130000114440	77,444750618920

Lokal (625 MB; 201 MB [500 *row*]; 826 MB)							*Downtime*	*Migration time*
Pengujian ke-	**Sumber**	**Tujuan**	**DOWN_TIMESTAMP**	**Total waktu (sumber)**	**UP_TIMESTAMP**	**Total waktu (tujuan)**		
1	server1	server2	1587357761,05	65,5849349499	1587357788,69	13,85864210130	27,6400001049041	79,443577051200
2	server2	server1	1587357873,58	63,7248029709	1587357902,13	13,12608909610	28,5500001907348	76,850892067000
3	server1	server2	1587357974,07	62,0661149025	1587357999,56	14,31155490880	25,4900000095367	76,377669811300
4	server2	server1	1587358071,75	63,1096591949	1587358097,72	16,55444002150	25,9700000286102	79,664099216400
5	server1	server2	1587358167,27	63,3095979691	1587358195,59	13,21844315530	28,3199999332427	76,528041124400
6	server2	server1	1587358271,03	62,3569660187	1587358296,96	13,04246401790	25,9300000667572	75,399430036600
7	server1	server2	1587358370,12	63,6299519539	1587358398,99	12,78546786310	28,8700001239776	76,415419817000
8	server2	server1	1587358488,32	65,9530041218	1587358513,18	13,33005690570	24,8600001335144	79,283061027500
9	server1	server2	1587358593,59	63,0234920979	1587358618,52	12,35645008090	24,9300000667572	75,379942178800
10	server2	server1	1587358690,75	64,5840799809	1587358718,18	15,28171992300	27,4300000667572	79,865799903900
Rata-rata							26,7990000724792	77,520793223410

Lokal (625 MB; 201 MB [1000 *row*]; 826 MB)							*Downtime*	*Migration time*
Pengujian ke-	**Sumber**	**Tujuan**	**DOWN_TIMESTAMP**	**Total waktu (sumber)**	**UP_TIMESTAMP**	**Total waktu (tujuan)**		
1	server1	server2	1587359921,68	64,2098288536	1587359950,81	15,68762493130	29,1299998760223	79,897453784900
2	server2	server1	1587360037,60	65,0633330345	1587360062,57	14,05428195000	24,9700000286102	79,117614984500
3	server1	server2	1587360150,75	62,8513951302	1587360176,42	13,63780403140	25,6700000762939	76,489199161600
4	server2	server1	1587360260,00	65,5127279758	1587360286,31	13,89702081680	26,3099999427795	79,409748792600
5	server1	server2	1587360367,86	61,8533358574	1587360396,62	17,33875203130	28,7599999904632	79,192087888700
6	server2	server1	1587360471,34	62,1339929104	1587360496,44	13,42618298530	25,1000001430511	75,560175895700
7	server1	server2	1587360567,87	60,3966569901	1587360595,23	14,99359393120	27,3600001335144	75,390250921300
8	server2	server1	1587360672,71	63,7032411098	1587360701,56	14,93435597420	28,8499999046325	78,637597084000
9	server1	server2	1587360786,21	62,3318669796	1587360814,03	15,54651117320	27,8199999332427	77,878378152800
10	server2	server1	1587360882,31	64,2281610966	1587360907,61	12,28484892850	25,2999999523162	76,513010025100
Rata-rata							26,9269999980927	77,808551669120

Lokal (625 MB; 202 MB [5000 *row*]; 827 MB)							***Downtime***	***Migration time***
Pengujian ke-	**Sumber**	**Tujuan**	**DOWN_TIMESTAMP**	**Total waktu (sumber)**	**UP_TIMESTAMP**	**Total waktu (tujuan)**		
1	server1	server2	1587362447,41	62,6730241776	1587362478,09	17,47264985080	30,6799998283386	80,145674028400
2	server2	server1	1587362583,19	65,5207050514	1587362612,49	13,81102009770	29,2999999523162	79,331725149100
3	server1	server2	1587362706,75	63,7724289894	1587362735,66	16,53235602380	28,9100000858306	80,304785013200
4	server2	server1	1587362812,30	61,7193749046	1587362841,06	13,93027809140	28,7599999904632	75,649652996000
5	server1	server2	1587362919,77	58,3938379288	1587362946,32	15,77968192100	26,5499999523162	74,173519849800
6	server2	server1	1587363019,16	65,9371550083	1587363046,92	17,11552500720	27,7599999904632	83,052680015500
7	server1	server2	1587363120,26	59,6315281391	1587363146,67	16,90245213510	26,4100000858306	76,533980274200
8	server2	server1	1587363224,31	63,9652090073	1587363251,92	14,39269804950	27,6100001335144	78,357907056800
9	server1	server2	1587363320,45	57,5976529121	1587363345,42	15,92246198650	24,9700000286102	73,520114898600
10	server2	server1	1587363416,14	62,5714849949	1587363440,86	14,98703289030	24,7199997901916	77,558517885200
Rata-rata							27,5669999837875	77,862855716680

Lokal (625 MB; 219 MB [10000 *row*]; 844 MB)							***Downtime***	***Migration time***
Pengujian ke-	**Sumber**	**Tujuan**	**DOWN_TIMESTAMP**	**Total waktu (sumber)**	**UP_TIMESTAMP**	**Total waktu (tujuan)**		
1	server1	server2	1587365605,69	61,8586080074	1587365634,73	15,26130390170	29,0399999618530	77,119911909100
2	server2	server1	1587365721,07	63,0880379677	1587365746,27	14,69212698940	25,2000000476837	77,780164957100
3	server1	server2	1587365820,63	67,8588690758	1587365851,71	14,91449689870	31,0799999237060	82,773365974500
4	server2	server1	1587365925,81	68,4470038414	1587365957,67	17,60547804830	31,8600001335144	86,052481889700
5	server1	server2	1587366037,95	63,3318879604	1587366063,41	15,06795787810	25,4600000381469	78,399845838500
6	server2	server1	1587366133,94	64,9550230503	1587366160,86	15,20461106300	26,9199998378753	80,159634113300
7	server1	server2	1587366234,86	61,6602210999	1587366258,98	14,13825583460	24,1200001239776	75,798476934500
8	server2	server1	1587366326,84	59,7384500504	1587366356,79	16,60599994660	29,9500000476837	76,344449997000
9	server1	server2	1587366427,69	61,9362001419	1587366453,53	15,51318001750	25,8399999141693	77,449380159400
10	server2	server1	1587366522,31	58,0541329384	1587366550,50	17,50778102870	28,1900000572204	75,561913967100
Rata-rata							27,7660000085831	78,743962574020

LAMPIRAN C HASIL PENGUJIAN *CLOUD*

Cloud (625 MB; 201 MB [100 *row*]; 826 MB)							*Downtime*	*Migration time*
Pengujian ke-	**Sumber**	**Tujuan**	**DOWN_TIMESTAMP**	**Total waktu (sumber)**	**UP_TIMESTAMP**	**Total waktu (tujuan)**		
1	server1	server2	1587434784,16	47,3851668835	1587434815,59	23,06412315370	31,4299998283386	70,449290037200
2	server2	server1	1587434881,90	40,4616758823	1587434917,52	29,64938402180	35,6199998855590	70,111059904100
3	server1	server2	1587434981,26	47,4667940140	1587435013,66	23,02895307540	32,4000000953674	70,495747089400
4	server2	server1	1587435074,66	41,3089079857	1587435110,98	30,55087780950	36,3199999332427	71,859785795200
5	server1	server2	1587435176,15	48,3849360943	1587435208,10	22,55291318890	31,9499998092651	70,937849283200
6	server2	server1	1587435278,72	41,0046181679	1587435314,75	30,15680599210	36,0299999713897	71,161424160000
7	server1	server2	1587435383,44	47,4763801098	1587435416,40	23,92699503900	32,9600000381469	71,403375148800
8	server2	server1	1587435483,86	41,8133208752	1587435521,98	32,18496608730	38,1200001239776	73,998286962500
9	server1	server2	1587435592,09	49,5866980553	1587435620,99	18,45913004880	28,9000000953674	68,045828104100
10	server2	server1	1587435685,67	41,2799298763	1587435722,19	30,77235412600	36,5199999809265	72,052284002300
Rata-rata							34,0249999761581	71,051493048680

Cloud (625 MB; 201 MB [500 *row*]; 826 MB)							*Downtime*	*Migration time*
Pengujian ke-	**Sumber**	**Tujuan**	**DOWN_TIMESTAMP**	**Total waktu (sumber)**	**UP_TIMESTAMP**	**Total waktu (tujuan)**		
1	server1	server2	1587436665,73	47,0805249214	1587436695,91	21,72936987880	30,1800000667572	68,809894800200
2	server2	server1	1587436768,33	41,8216707706	1587436803,43	29,16190981860	35,1000001430511	70,983580589200
3	server1	server2	1587436872,16	48,6426439285	1587436903,99	21,63313984870	31,8299999237060	70,275783777200
4	server2	server1	1587436963,94	40,8410949707	1587437001,35	31,58953094480	37,4099998474121	72,430625915500
5	server1	server2	1587437064,18	47,1558778286	1587437097,37	24,05601501460	33,1899998188018	71,211892843200
6	server2	server1	1587437173,60	41,2994821072	1587437208,76	29,41950201990	35,1600000858306	70,718984127100
7	server1	server2	1587437269,85	47,0072209835	1587437303,13	24,57420492170	33,2800002098083	71,581425905200
8	server2	server1	1587437399,84	41,8413109779	1587437434,70	28,86975598340	34,8600001335144	70,711066961300
9	server1	server2	1587437500,53	49,1349449158	1587437535,14	24,72428393360	34,6100001335144	73,859228849400
10	server2	server1	1587437597,93	43,3877658844	1587437635,00	30,99803590770	37,0699999332427	74,385801792100
Rata-rata							34,2690000295639	71,496828556040

Cloud (625 MB; 201 MB [1000 *row*]; 826 MB)							*Downtime*	*Migration time*
Pengujian ke-	**Sumber**	**Tujuan**	**DOWN_TIMESTAMP**	**Total waktu (sumber)**	**UP_TIMESTAMP**	**Total waktu (tujuan)**		
1	server1	server2	1587438677,51	47,6928601265	1587438711,51	25,27908587460	34,0000000000000	72,971946001100
2	server2	server1	1587438779,30	40,9482259750	1587438814,78	29,63144087790	35,4800000190734	70,579666852900
3	server1	server2	1587438880,84	48,9442949295	1587438911,84	22,28230905530	31,0000000000000	71,226603984800
4	server2	server1	1587438987,00	40,7094268799	1587439023,87	30,69826483730	36,8699998855590	71,407691717200
5	server1	server2	1587439089,56	49,2051119804	1587439123,58	23,49906897540	34,0199999809265	72,704180955800
6	server2	server1	1587439189,80	40,5414240360	1587439226,53	30,92761182790	36,7300000190734	71,469035863900
7	server1	server2	1587439288,90	47,5978040695	1587439320,38	22,26270794870	31,4800000190734	69,860512018200
8	server2	server1	1587439376,44	40,8028259277	1587439412,77	30,77285718920	36,3299999237060	71,575683116900
9	server1	server2	1587439479,53	47,8907501698	1587439514,15	25,15099310870	34,6200001239776	73,041743278500
10	server2	server1	1587439569,96	40,1429610252	1587439606,78	30,99204993250	36,8199999332427	71,135010957700
Rata-rata							34,7349999904633	71,597207474700

Cloud (625 MB; 202 MB [5000 *row*]; 827 MB)							***Downtime***	***Migration time***
Pengujian ke-	**Sumber**	**Tujuan**	**DOWN_TIMESTAMP**	**Total waktu (sumber)**	**UP_TIMESTAMP**	**Total waktu (tujuan)**		
1	server1	server2	1587441069,95	47,0812890530	1587441100,74	22,22011089320	30,7899999618530	69,301399946200
2	server2	server1	1587441180,51	42,1705999374	1587441216,67	30,12169504170	36,1600000858306	72,292294979100
3	server1	server2	1587441277,57	49,0422639847	1587441312,68	23,58348298070	35,1100001335144	72,625746965400
4	server2	server1	1587441370,08	41,7256441116	1587441408,28	31,18483901020	38,2000000476837	72,910483121800
5	server1	server2	1587441459,43	46,4600629807	1587441493,65	22,70866918560	34,2200000286102	69,168732166300
6	server2	server1	1587441550,02	41,0643470287	1587441586,26	31,29671502110	36,2400000095367	72,361062049800
7	server1	server2	1587441646,45	49,4793009758	1587441679,15	23,50385713580	32,7000000476837	72,983158111600
8	server2	server1	1587441734,16	42,0298531055	1587441770,96	30,86140894890	36,7999999523162	72,891262054400
9	server1	server2	1587441831,03	47,1670069695	1587441863,79	23,46700787540	32,7599999904632	70,634014844900
10	server2	server1	1587441918,53	39,8741929531	1587441954,11	31,72472906110	35,5799999237060	71,598922014200
Rata-rata							34,8560000181198	71,676707625370

Cloud (625 MB; 219 MB [10000 *row*]; 844 MB)							*Downtime*	*Migration time*
Pengujian ke-	**Sumber**	**Tujuan**	**DOWN_TIMESTAMP**	**Total waktu (sumber)**	**UP_TIMESTAMP**	**Total waktu (tujuan)**		
1	server1	server2	1587444395,07	48,3595628738	1587444429,56	25,11684894560	34,4900000095367	73,476411819400
2	server2	server1	1587444489,03	41,7700641155	1587444527,24	30,64388489720	38,2100000381469	72,413949012700
3	server1	server2	1587444589,20	48,6561448574	1587444621,67	23,21381783490	32,4700000286102	71,869962692300
4	server2	server1	1587444672,54	41,7468590736	1587444709,15	30,23440909390	36,6100001335144	71,981268167500
5	server1	server2	1587444766,54	48,7401759624	1587444799,90	23,91518616680	33,3600001335144	72,655362129200
6	server2	server1	1587444856,28	41,3513238430	1587444892,53	30,36534190180	36,2500000000000	71,716665744800
7	server1	server2	1587444957,24	48,4817969799	1587444989,10	22,36683201790	31,8599998950958	70,848628997800
8	server2	server1	1587445041,23	41,9728112221	1587445079,08	30,97921299930	37,8499999046325	72,952024221400
9	server1	server2	1587445147,92	48,0558190346	1587445181,14	23,50140786170	33,2200000286102	71,557226896300
10	server2	server1	1587445237,90	41,7406539917	1587445275,63	31,91039991380	37,7300000190734	73,651053905500
Rata-rata							35,2050000190735	72,312255358690