Institut
Teknologi
Sepuluh Nopember

TUGAS AKHIR - IF184802

RANCANG BANGUN DISASTER RECOVERY SWITCH UNTUK BACKUP-RECOVERY VIRTUAL MACHINE SERVER

NAHDA FAUZIYAH ZAHRA
NRP 05111540000141

Dosen Pembimbing I
Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D

Dosen Pembimbing II
Bagus Jati Santoso, S.Kom., Ph.D

DEPARTEMEN TEKNIK INFORMATIKA
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya, 2020

(Halaman ini sengaja dikosongkan)

TUGAS AKHIR - IF184802

RANCANG BANGUN DISASTER RECOVERY SWITCH UNTUK BACKUP-RECOVERY VIRTUAL MACHINE SERVER

NAHDA FAUZIYAH ZAHRA
NRP 05111540000141

Dosen Pembimbing I
Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D

Dosen Pembimbing II
Bagus Jati Santoso, S.Kom., Ph.D

DEPARTEMEN TEKNIK INFORMATIKA
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya, 2020

(Halaman ini sengaja dikosongkan)

UNDERGRADUATE THESIS - IF184802

DESIGN OF DISASTER RECOVERY SWITCH FOR BACKUP-RECOVERY VIRTUAL MACHINE SERVER

NAHDA FAUZIYAH ZAHRA
NRP 05111540000141

Supervisor I
Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D

Supervisor II
Bagus Jati Santoso, S.Kom., Ph.D

DEPARTMENT OF INFORMATICS ENGINEERING
Faculty of ELECTICS
Institut Teknologi Sepuluh Nopember
Surabaya, 2020

(Halaman ini sengaja dikosongkan)

LEMBAR PENGESAHAN

RANCANG BANGUN DISASTER RECOVERY SWITCH UNTUK BACKUP-RECOVERY VIRTUAL MACHINE SERVER

TUGAS AKHIR

Diajukan Guna Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Arsitektur Jaringan dan Komputer
Program Studi S1 Departemen Teknik Informatika
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember

Oleh :
NAHDA FAUZIAH ZAHRA
NRP: 05111540000141

Disetujui oleh Dosen Pembimbing Tugas Akhir

Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D. .....
NIP: 197708242006041001 (Pembimbing 1)

Bagus Jati Santoso, S.Kom., Ph.D .....
NIP: 198611252018031001 (Pembimbing 2)

SURABAYA
Januari 2020

(Halaman ini sengaja dikosongkan)

RANCANG BANGUN DISASTER RECOVERY SWITCH UNTUK BACKUP-RECOVERY VIRTUAL MACHINE SERVER

Nama	:	**NAHDA FAUZIYAH ZAHRA**
NRP	:	**05111540000141**
Departemen	:	**Teknik Informatika FTEIC**
Pembimbing I	:	**Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D**
Pembimbing II	:	**Bagus Jati Santoso, S.Kom., Ph.D**

Abstrak

Virtual Machine merupakan teknik virtualisasi yang menyajikan perangkat keras yang dapat menjalankan perangkat lunak seperti perangkat keras fisik. Penyedia layanan Virtual Machine biasa disebut dengan Hypervisor. Hypervisor menangani manajemen Virtual Machine pada sebuah host. Penggunaan Virtual Machine dalam dunia teknologi sangat banyak dilakukan. Karena sangat banyaknya penggunaan Virtual Machine, maka dari itu banyak Hypervisor yang disediakan oleh pengembang, contohnya Vmware, Proxmox, Xen, Qemu dan lain-lain. Keragaman Hypervisor menyebabkan perbedaan cara pengoperasian. Hal ini menyebabkan sulitnya alokasi Virtual Machine. Masalah tersebut juga dialami oleh DPTSI ITS. Teknik virtualisasi pun tidak luput dari kegagalan sistem yang dapat menyebabkan kehilangan data pada server, maka diperlukan adanya prosedur Disaster Recovery salah satunya untuk menjaga konsistensi data pada server.

System Administrator perlu menjaga konsistensi data pada server secara berkala. Keragaman lingkungan hypervisor merupakan salah satu kesulitan yang dialami System Administrator untuk melakukan prosedur Disaster Recovery

Center salah satunya yaitu backup-restore virtual machine.

Dalam tugas akhir ini dibuat sebuah rancangan sistem yang memungkinkan menjembatani cara prosedur Disaster Recovery menggunakan abstraksi switch. Sistem diakses melalui antarmuka yang disediakan untuk pengguna melakukan proses backup-restore virtual machine. Dari hasil uji coba, sistem dapat menangani permintaan proses backup, restore, serta proses backup yang dilakukan secara terjadwal. Proses backup yang terjadwal dalam satu waktu menghasilkan waktu eksekusi yang relatif konstan karena proses dieksekusi secara sekuensial.

***Kata-Kunci**: middleware, hypervisor, proxmox, disaster recovery, virtual machine, switch*

DESIGN OF DISASTER RECOVERY SWITCH FOR BACKUP-RECOVERY VIRTUAL MACHINE SERVER

Name	:	**NAHDA FAUZIYAH ZAHRA**
NRP	:	**05111540000141**
Department	:	**Informatics Engineering FTEIC**
Supervisor I	:	**Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D**
Supervisor II	:	**Bagus Jati Santoso, S.Kom., Ph.D**

Abstract

Virtual Machine is a virtualization technique that presents hardware that can run software such as physical hardware. Virtual Machine service providers are commonly called Hypervisors. Hypervisor handles virtual machine management on a host. The use of Virtual Machines in the world of technology is very much done. Because there are so many uses of the Virtual Machine, many hypervisors are therefore provided by developers, for example Vmware, Proxmox, Xen, Qemu and others. Hypervisor diversity causes different ways of operating. This makes it difficult to allocate Virtual Machines. This problem was also experienced by DPTSI ITS. Virtualization techniques are not spared from system failures that can cause data loss on the server, so a Disaster Recovery procedure is needed, one of which is to maintain data consistency on the server. System Administrators need to maintain data consistency on the server periodically. The diversity of the hypervisor environment is one of the difficulties experienced by the System Administrator to perform the Disaster Recovery Center procedure, one of which is the backup-restore virtual machine. In this final project, a system design is made that allows bridging the way the Disaster Recovery procedure uses an abstraction switch. The system is

accessed through an interface that is provided for users to do a virtual machine backup-restore process. From the results of the trial, the system can handle requests for backup, restore and backup processes that are carried out on a scheduled basis. The scheduled backup process at one time produces a relatively constant execution time because the process is executed sequentially.

***Keywords**: middleware, hypervisor, vmware, proxmox, ahp, virtual machine*

KATA PENGANTAR

بِسْمِ ٱللَّهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ

Alhamdulillahirabbil'alamin, segala puji bagi Allah SWT, yang telah melimpahkan rahmat dan hidayah-Nya sehingga penulis dapat menyelesaikan Tugas Akhir yang berjudul **Rancang Bangun Manajemen Alokasi *Virtual Machine* dalam Lingkungan *Hypervisor* yang Heterogen**. Pengerjaan Tugas Akhir ini merupakan suatu kesempatan yang sangat baik bagi penulis. Dengan pengerjaan Tugas Akhir ini, penulis bisa belajar lebih banyak untuk memperdalam dan meningkatkan apa yang telah didapatkan penulis selama menempuh perkuliahan di Departemen Informatika ITS. Dengan Tugas Akhir ini penulis juga dapat menghasilkan suatu implementasi dari apa yang telah penulis pelajari. Selesainya Tugas Akhir ini tidak lepas dari bantuan dan dukungan beberapa pihak. Sehingga pada kesempatan ini penulis mengucapkan syukur dan terima kasih kepada:

1. Allah SWT atas anugerahnya yang tidak terkira kepada penulis dan Nabi Muhammad SAW.
2. Keluarga penulis yang selalu menyemangati.
3. Bapak Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D selaku pembimbing I yang telah membantu, membimbing dan memotivasi penulis mulai dari pengerjaan proposal hingga terselesaikannya Tugas Akhir ini.
4. Bapak Bagus Jati Santoso, S.Kom., Ph.D selaku pembimbing II yang juga telah membantu, membimbing dan memotivasi penulis mulai dari pengerjaan proposal hingga terselesaikannya Tugas Akhir ini.
5. Teman-teman *Administrator* laboratorium AJK.
6. Bapak Darlis Herumurti, S.Kom., M.Kom., selaku Kepala Departemen Teknik Informatika ITS pada masa pengerjaan Tugas Akhir, serta Bapak Radityo Anggoro, S.Kom.,

M.Sc., selaku koordinator Tugas Akhir dan segenap dosen Departemen Informatika yang telah memberikan ilmu dan pengalamannya.

7. Serta semua pihak yang telah turut membantu penulis dalam menyelesaikan Tugas Akhir ini.

Penulis menyadari bahwa Tugas Akhir ini masih memiliki banyak kekurangan. Sehingga dengan kerendahan hati, penulis mengharapkan kritik dan saran dari pembaca untuk perbaikan ke depannya.

Surabaya, Januari 2020

Nahda Fauziyah Z

DAFTAR ISI

(Halaman ini sengaja dikosongkan)

DAFTAR TABEL

(Halaman ini sengaja dikosongkan)

DAFTAR GAMBAR

(Halaman ini sengaja dikosongkan)

DAFTAR KODE SUMBER

(Halaman ini sengaja dikosongkan)

BAB I

PENDAHULUAN

Pada bab ini akan dipaparkan mengenai garis besar Tugas Akhir yang meliputi latar belakang, tujuan, rumusan dan batasan permasalahan, metodologi pembuatan Tugas Akhir dan sistematika penulisan.

1.1 Latar Belakang

Sistem komputer serta komponennya dapat mengalami kegagalan dan berisiko kehilangan data. Kegagalan pada sistem komputer yang berperan sebagai *server* akan mempengaruhi terhadap layanan atau proses bisnis yang dijalankannya. Untuk mengurangi risiko kehilangan data, biasanya akan dibuat data salinan untuk disimpan pada perangkat *server* lain yang disebut sebagai *server* sekunder. Hal tersebut juga dapat diterapkan dalam penggunaan *server* yang memanfaatkan teknik virtualisasi dalam bentuk *virtual machine* dengan membuat replika dari *virtual server* tersebut dan akan disimpan serta dijalankan sebagai *virtual server* pada *server* sekunder. Prosedur tersebut merupakan salah satu jenis proses *Disaster Recovery* untuk menjaga konsistensi data pada *server*.

Salah satu manfaat dari penggunaan teknik virtualisasi yaitu meminimalkan ruang yang diperlukan untuk *server* karena sebuah perangkat komputer fisik dapat menjalankan lebih dari satu *server*. Maka, sebuah perangkat komputer fisik harus dapat mengalokasikan sumber daya I/O yang dimiliki agar dapat digunakan secara bersamaan oleh *virtual machine* yang terdapat di dalamnya. *Virtual machine* yang terdapat dalam perangkat fisik berjalan di atas penyedia layanan virtualisasi yang disebut dengan *Hypervisor*. *Hypervisor* menangani manajemen *virtual machine* pada sebuah perangkat fisik. Maraknya pemanfaatan teknik virtualiasasi saat ini menjadikan munculnya beragam penyedia *Hypervisor*, seperti adalah VMware, Proxmox, Xen,

Qemu, dan lain-lain. Beberapa jenis hypervisor tersebut telah diterapkan pada *Data Center* DPTSI ITS.

Keragaman *hypervisor* menyebabkan perbedaan pengoperasian manajemen *virtual machine* maupun proses *backup-recovery*. Umumnya, *Hypervisor* menyediakan fitur untuk melakukan *backup* dan *restore* untuk *virtual machine* di dalamnya, tetapi dalam prosedur *Disaster Recovery* replika akan dijalankan pada perangkat *Hypervisor* yang berbeda untuk menjaga konsistensi data *server*. Hal ini menyebabkan sulitnya *System Adminsitrator* untuk melakukan proses *backup-recovery* karena membutuhkan beberapa langkah yang berbeda untuk setiap lingkungan *hypervisor* dan melakukan prosedur tersebut secara manual. Sebuah abstraksi *switch* dapat diterapkan pada prosedur *Disaster Recovery* untuk menjembatani perbedaan pengoperasian proses *backup-recovery virtual machine* melalui *Hypervisor* yang digunakan dalam perangkat fisik tersebut.

Oleh karena itu, diperlukan adanya jembatan untuk menjalankan otomasi prosedur *Disaster Recovery*. Salah satunya dengan membuat sebuah abstraksi *switch* yang akan berperan sebagai *middleware* untuk melakukan prosedur *Disaster Recovery* agar *System Administrator* dapat mengatur prosedur tersebut secara otomatis.

1.2 Rumusan Masalah

Rumusan masalah yang diangkat dalam tugas akhir ini adalah sebagai berikut :

1. Bagaimana membuat *middleware* dengan abstraksi *switch* untuk menjembatani *Hypervisor* di *Data Center* dan *Data Recovery Center*?
2. Bagaimana membuat dan mentransfer replika *virtual machine* menggunakan *middleware switch* dari *Data Center* ke *Data Recovery Center*?

3. Bagaimana mentransfer dan menjalankan kembali replika *virtual machine* menggunakan *middleware* dari *Data Center* ke *Data Recovery Center*?
4. Bagaimana performa transfer replika *virtual machine* dengan spesifikasi yang beragam?

1.3 Batasan Masalah

Dari permasalahan yang telah diuraikan di atas, terdapat beberapa batasan masalah pada tugas akhir ini, yaitu:

1. *Hypervisor* yang didukung oleh sistem adalah Proxmox.
2. Setiap *virtual machine server* yang dibuat replikanya akan dijalankan kembali pada *Data Recovery Center* dengan lingkungan *Hypervisor* yang sama dengan *Data Center*.

1.4 Tujuan

Tujuan pembuatan tugas akhir ini antara lain:

1. Membuat sebuah *middleware* yang digunakan untuk menjembatani prosedur *Disaster Recovery* dijalankan melalui antarmuka web.
2. Membuat sebuah mekanisme untuk mentransfer replika *virtual machine* menggunakan *middleware* dengan abstraksi *switch*.

1.5 Manfaat

Manfaat Tujuan pembuatan tugas akhir ini antara lain:

1. Mempermudah mekanisme *Disaster Recovery virtual machine*.
2. Memetakan mekanisme Disaster Recovery dari perangkat di *Data Center* ke perangkat di *Data Recovery Center*.

(Halaman ini sengaja dikosongkan)

BAB II

TINJAUAN PUSTAKA

2.1 Virtualisasi

Virtualisasi merupakan komponen terpenting dalam komputasi awan dengan memisahkan perangkat keras dengan sistem operasi yang berjalan. Virtualisasi memiliki kemampuan untuk membagi sumber daya fisik yang ada untuk dijadikan sumber daya *virtual* dan dapat menjadikan berbagai sumber daya fisik yang ada menjadi satu sumber daya *virtual*. Virtualisasi membawa banyak perubahan pada perusahaan IT saat ini.

Virtualisasi dan *multitasking* pada sistem operasi memiliki kemampuan untuk mengizinkan pemusatan berbagai server *virtual* pada sebuah komputer fisik. Ketika sebuah kelompok ingin mengerjakan sebuah pekerjaan tertentu pada dua atau lebih server dan salah satu server gagal karena sumber daya habis terpakai, virtualisasi dapat memindahkan pekerjaan dan server tersebut pada komputer fisik yang lain. Hal tersebut merupakan salah satu keuntungan menggunakan virtualisasi. Selain itu virtualisasi dapat menghemat energi yang dipakai dan biaya pembelian komputer fisik. Sehingga virtualisasi dapat meningkatkan keuntungan perusahaan[1].

2.2 *Hypervisor*

Pada virtualisasi, terdapat sebuah *layer* yang berada diantara perangkat keras dan *virtual machine*. *Layer* tersebut disebut *Hypervisor*. *Hypervisor* menyediakan standarisasi *CPU*, *memory* dan *storage* untuk *virtual machine*. *Hypervisor* memiliki dua jenis yaitu, *Bare-metal Hypervisor* dan *Hosted Hypervisor*[1].

2.2.1 *Bare-metal Hypervisor*

Jenis *hypervisor* ini dilakukan instalasi pada perangkat keras x86 secara langsung. Setelah melakukan instalasi *hypervisor*,

pengguna dapat melakukan instalasi sistem operasi pada (*virtual machine*) atau yang sering disebut *instance*. Jenis *hypervisor* ini lebih efisien dibanding *Hosted hypervisor*.

2.2.2 *Hosted Hypervisor*

Jenis *hypervisor* ini dilakukan instalasi pada sistem operasi yang sudah terinstalasi pada perangkat keras sebelumnya. *Instance* dibuat setelah melakukan instalasi *hypervisor*.

2.3 Python

Python adalah bahasa pemrograman interpretatif, interaktfi dan berorientasi objek. Python menggabungkan modul, pengecualian, penulisan secara dinamis, tipe data dinamis yang sangat tinggi dan kelas. Python memiliki antarmuka ke banyak *system call* dan pusataka diberbagai sistem dan dapat diperluas ke bahasa pemrograman C atau C++. Python dapat berjalan pada berbagai sistem operasi seperti Unix, Linux, Mac Os dan Windows.

Python adalah bahasa pemrograman tingkat tinggi yang dapat diterapkan pada berbagai masalah. Bahasa ini dilengkapi pustaka yang besar untuk melakukan pemrosesan *string*, protokol internet, rekayasa perangkat lunak dan antarmuka sistem operasi[2].

2.4 Flask

Flask adalah kerangka aplikasi web Python yang ringan. Flask dirancang untuk memulai membuat web dengan cepat dan mudah, dengan kemampuan untuk membuat aplikasi web sampai tingkat yang rumit. Flask dibuat dengan terintegrasi dengan modul Werkzeug dan Jinja. Flask termasuk salah satu kerangka aplikasi web Python yang populer.

Flask didesain tidak memiliki depedensi dan tata letak kerangka aplikasi, dengan demikian pengembang memiliki kebebasan untuk mengatur kerangka aplikasinya sendiri serta menambahkan modul yang diperlukan sesuai kebutuhan. Flask memiliki berbagai ekstensi yang dikembangkan oleh komunitas sehingga dapat menambahkan berbagai fungsi dengan mudah[3].

Flask memiliki kelebihan yaitu sangat ringan dan sangat sederhana dalam proses pengembangan. Sehingga Flask sangat cocok digunakan untuk pembuatan *HTTP Rest API*. Pada tugas akhir ini, Flask akan digunakan untuk pembuatan *HTTP Rest API*.

2.5 Rsync

Rsync merupakan salah satu utilitas *open source* yang menyediakan layanan *incremental file* transfer. Rsync juga dapat melakukan kompresi terhadap *file* yang dikirim untuk meminimalkan ukuran *file* yang dikirim[4].

2.6 VZDump

VZDump merupakan salah satu alat untuk membuat replika *virtual machine* dalam bentuk arsip untuk menjaga konsistensi *state* dari *virtual machine* maupun *container* pada *Hypervisor*. VZDump akan menyediakan beberapa opsi untuk menjalankan proses pembuatan replika, seperti tipe pengarsipan yang digunakan, direktori penyimpanan, mode untuk menjalankan pengarsipan, dan sebagainya. VZdump membuat replika pada *disk* dalam sebuah *file* yang harus diletakkan pada replika *disk* untuk pengarsipan[5].

2.7 Proxmoxer

Proxmoxer adalah Python *wrapper* untuk mengakses API Proxmox versi 2 melalui protokol HTTPS dan SSH. Modul ini dikembangkan oleh komunitas[6]. Proxmoxer sangat mudah digunakan dibanding modul yang lain. Selain itu, Proxmoxer mendukung berbagai akses fungsionalitas dalam manajemen Proxmox.

2.8 MySQL

MySQL adalah merupakan salah basis data resional terbuka yang awalnya dikembangkan oleh MySQL AB. Mulai tahun 2009, setelah Oracle Corporation mengakusisi Sun Microsystems, MySQL dikembangkan dan distribusikan oleh Oracle Corporation. MySQL tersedia sebagai basis data gratis di bawah lesensi *GNU General Public License* (GPL), tetapi juga tersedia lisensi komersial.

MySQL bekerja pada banyak *platform*, seperti Compaq Tru64, DEC OSF, FreeBSD, IBM AIX, HP-UX, Linux, Mac OS X, Novell NetWare, OpenBSD, QNX, SCO, SGI IRIX, Solaris (versions 8, 9 and 10) dan Microsoft Windows. MySQL juga menyediakan *source code* apabila MySQL dalam bentuk *binaries* tidak tersedia pada platform yang digunakan. MySQL menyediakan *API* untuk berbagai bahasa pemrograman seperti C, C++, Java, Perl, PHP, Ruby, Tcl dan lainnya.

MySQL juga menawarkan banyak jenis mekanisme untuk mengelola data, yang dikenal sebagai *storage engines*. Versi 5 menambahkan mesin ARCHIVE, BLACKHOLE, CSV, FEDERATED dan EXAMPLES[7].

BAB III

DESAIN DAN PERANCANGAN

Pada bab ini dibahas mengenai analisis dan perancangan sistem.

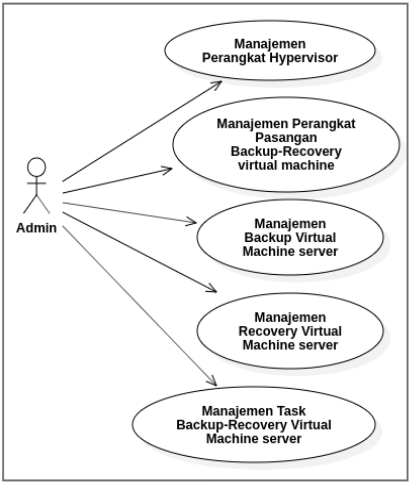
3.1 Deskripsi Umum Sistem

Sistem yang akan dibuat dalam tugas akhir ini adalah sistem yang digunakan untuk proses *backup-restore virtual machine* pada perangkat *hypervisor* di *Data Center* dan akan dikirim ke *Data Recovery Center* sebagai replika virtual machine yang berjalan di *Data Center*. Sistem memiliki *worker* yang mengeksekusi pekerjaan yang harus dilakukan. System Administrator akan mengirimkan pekerjaan *backup* atau *restore* melalui HTTP REST API dengan memberikan informasi perangkat yang dibutuhkan untuk proses tersebut lalu data akan dimasukkan ke dalam basis data. *Worker* akan melakukan pekerjaan berdasarkan daftar pekerjaan dan informasi yang ada pada basis data. Terdapat juga mekanisme untuk penjadwalan proses backup pada sis tem ini untuk melakukan otomasi untuk penjadwalan proses *backup-restore virtual machine* pada perangkat fisik yang berbeda dalam satu lingkungan *hypervisor*. Sistem yang dikembangkan oleh penulis mendukung proses *backup-restore virtual machine* pada *hypervisor Proxmox*.

Sistem ini melibatkan 3 (tiga) komponen *server*, yaitu *Data Center*, *Middleware*, dan *Data Recovery Center*.

3.2 Kasus Penggunaan

Terdapat satu aktor dalam sistem ini, yaitu admin (*System Administrator*). Diagram kasus penggunaan digambarkan pada Gambar 3.1.



Gambar 3.1: Diagram Kasus Penggunaan

Diagram kasus penggunaan pada Gambar 3.1 dideskripsikan masing-masing pada Tabel 3.1.

Tabel 3.1: Daftar Kode Kasus Penggunaan

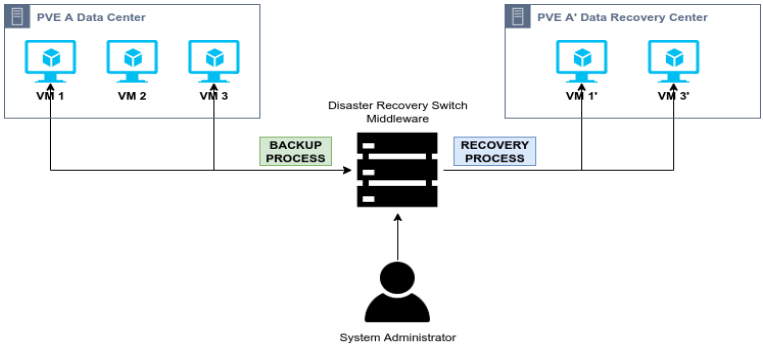
Kode Kasus Penggunaan	**Nama Kasus Penggunaan**	**Keterangan**
UC-0001	Manajemen Perangkat *Hypervisor*.	Admin dapat membuat, melihat, mengubah dan menghapus data *Hypervisor*.
UC-0002	Manajemen Perangkat Pasangan *backup-recovery virtual machine*.	Admin dapat membuat, melihat, mengubah dan menghapus data pasangan *backup-recovery virtual machine*.

Tabel 3.1: Daftar Kode Kasus Penggunaan

Kode Kasus Penggunaan	Nama Kasus Penggunaan	Keterangan
UC-0003	Manajemen *backup Virtual Machine server*.	Admin dapat mengatur waktu *backup virtual machine*.
UC-0004	Manajemen *Recovery Virtual Machine server*.	Admin dapat mengatur replika *recovery virtual machine* yang ingin di-*restore*.
UC-0005	Melihat *task backup-recovery Virtual Machine server*.	Admin dapat melihat *task backup-recovery virtual machine*.

3.3 Arsitektur Sistem

Pada sub-bab ini, dibahas mengenai tahap analisis dan kebutuhan bisnis dan desain dari sistem yang akan dibangun. Arsitektur sistem secara umum ditunjukkan pada Gambar 3.2.

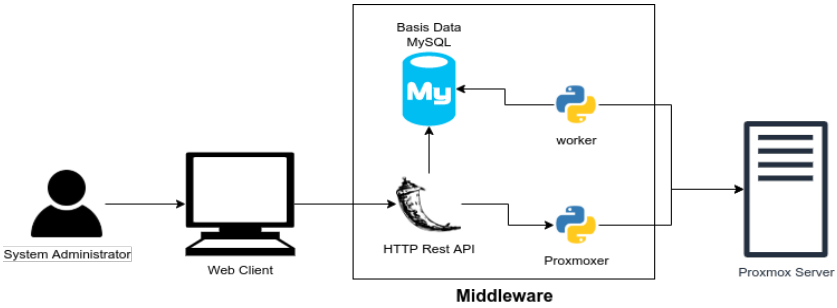


Gambar 3.2: Desain Umum Sistem

Middleware digunakan sebagai server yang akan mengirim *task backup* pada perangkat *Data Center* dan *task restore* pada perangkat *Data Recovery Center*, melakukan transfer replika dari *Data Center* dan ke *Data Recovery Center*, serta melakukan penjadwalan terhadap *tasks* yang dieksekusi oleh pengguna.

3.3.1 Desain Umum Sistem

Middleware terdiri dari beberapa komponen, yaitu *HTTP Rest API*, *worker*, basis data, dan *storage* penyimpanan *file* replika. Arsitektur *middleware* dapat dilihat pada Gambar 3.3.

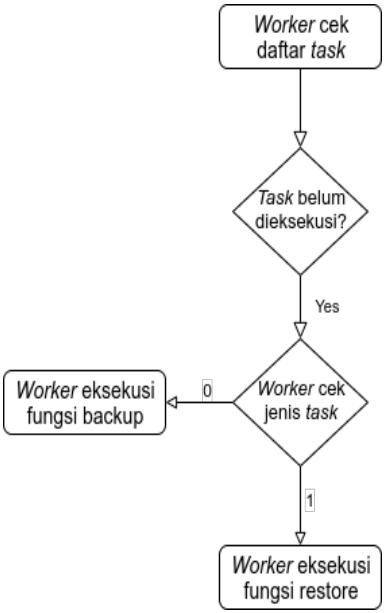


Gambar 3.3: Desain Middleware

3.3.2 Desain Proses *Backup-Restore Virtual Machine*

Untuk melakukan proses *backup* dan *restore*, *middleware* akan mengirimkan perintah ke perangkat *Data Center* untuk melakukan proses backup, lalu file backup tersebut akan dikirim menggunakan rsync dan disimpan di dalam *storage middleware*. Proses restore pun juga akan dilakukan dengan mengirimkan file replika virtual machine ke perangkat *Data Recovery Center* dan *middleware* akan mengirimkan perintah ke perangkat *Data Recovery Center* untuk melakukan proses *restore virtual machine*

terhadap *file* yang dikirim. Desain skenario proses *backup* dan *restore virtual machine* dapat dilihat pada Gambar 3.4.



Gambar 3.4: Desain Skenario Proses Backup dan *Restore Virtual Machine*

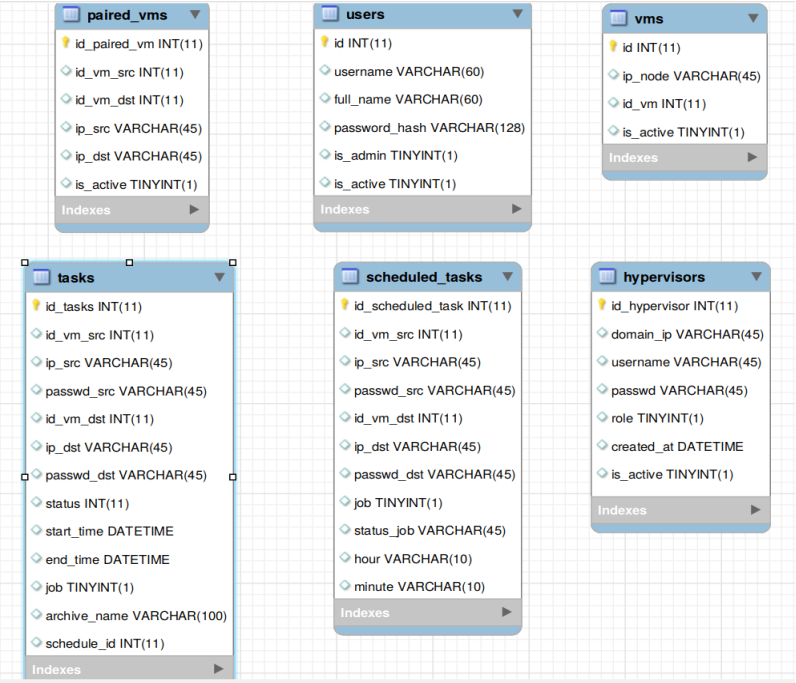
3.3.3 Desain Antrian *Tasks*

Proses *backup* maupun *restore* yang akan dieksekusi oleh pengguna akan dimasukkan ke dalam basis data. *Worker* akan mengambil data informasi dari tabel *Tasks* pada basis data dengan memeriksa status *task* sudah dijalankan atau belum. Jika belum, maka *worker* akan menjalankan *script* untuk mengeksekusi proses *backup* atau *restore*. Ketika *worker* selesai mengerjakan *task* tersebut, maka status *task* akan diperbaharui menjadi sudah tereksekusi. Untuk prosedur *task* yang dijadwalkan oleh pengguna, informasi *task* akan masuk terlebih

dahulu ke tabel *scheduled_task*, ketika informasi waktu sesuai dengan waktu terkini *task* akan didaftarkan ke tabel *tasks* agar menjadi task yang akan diekseskusi.

3.3.4 Desain Basis Data

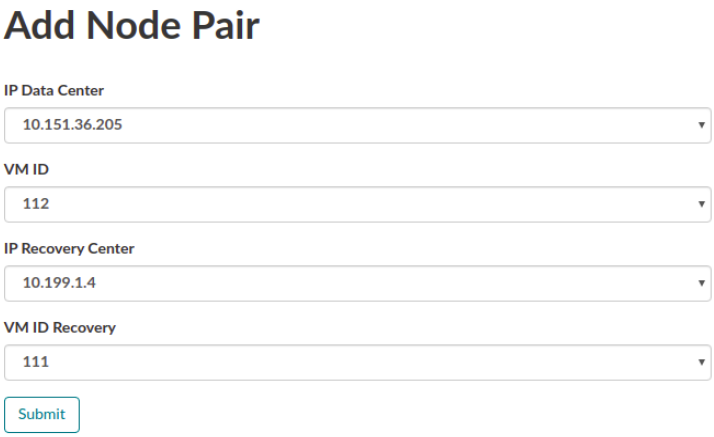
Desain basis data pada sistem ini adalah seperti yang digambarkan pada Gambar 3.5. Terdapat lima tabel yang digunakan untuk menyimpan data yang terlibat dalam sistem ini, yaitu: *users*, *tasks*, *hypervisors*, vms dan *paired_vms*. Arsitektur basis data pada *middleware* dapat dilihat pada gambar berikut.



Gambar 3.5: Desain Basis Data

3.3.5 Desain Antarmuka Web

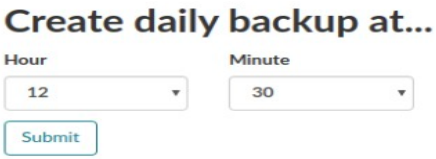
Desain antarmuka adalah desain untuk halaman yang nantinya akan digunakan oleh (*System Administrator*). Antarmuka akan dibuat berbasis web menggunakan Bootstrap 3 dan HTML. Pada halaman ini admin dapat melakukan manajemen pasangan *backup-restore virtual machine*, seperti: menambah, mengedit, menghapus, dan melihat daftar pasangan *backup-restore virtual machine*. Desain antarmuka web manajemen pasangan *backup-restore virtual machine* dapat dilihat pada Gambar 3.6.



Gambar 3.6: Desain Antarmuka Web Menambah Pasangan *backup-restore Virtual Machine*

System Administrator juga dapat melakukan penjadwalan untuk proses *backup virtual machine*. *System Administrator* dapat menjadwalkan dengan memilih tombol ”*Schedule Backup*” pada halaman daftar pasangan *backup-restore virtual machine*, lalu memilih waktu kapan proses backup ingin dijalankan per

harinya. Desain antarmuka web untuk melihat daftar *scheduled tasks* dapat dilihat pada Gambar 3.7.



Gambar 3.7: Desain Antarmuka Web Membuat *Scheduled Tasks*

BAB IV

IMPLEMENTASI

Bab ini membahas implementasi sistem Manajemen *Backup-Recovery* secara rinci. Pembahasan dilakukan secara rinci untuk setiap komponen yang ada, yaitu: *switch/middleware*, proses *backup virtual machine*, proses *recovery virtual machine*, Basis Data, dan antarmuka web.

4.1 Lingkungan Implementasi

Dalam mengimplementasikan sistem, digunakan beberapa perangkat pendukung sebagai ebrikut.

4.1.1 Perangkat Keras

Perangkat keras yang digunakan dalam pengembangan sistem adalah sebagai berikut:

1. *Middleware*, *processor* Intel(R) Xeon(R) CPU E5-2690 v4 @ 2.60GHz dan Ram 4GB.
2. *Server* Proxmox A dengan IP 10.151.36.205, *processor* Intel(R) Core(TM) i3-2120 CPU @ 3.30GHz dan Ram 12GB.
3. *Server* Proxmox B dengan IP 10.199.1.4, *processor* Intel(R) Core(TM) i5-3470 CPU @ 3.20GHz dan Ram 8GB.

4.1.2 Perangkat Lunak

Perangkat lunak yang digunakan dalam pengembangan adalah sebagai berikut:

- Sistem Operasi Ubuntu 18.04
- Proxmox VE
- Python 3.6
- Flask
- MySQL

4.2 Implementasi *Middleware*

Server Middleware merupakan *server* inti dari sistem. Pada *server* ini yang akan mengelola keseluruhan data dari sistem. Pada server ini, selain mengelola kesuluruhan data, *server middleware* memetakan *node* (*server virtual machine*) dari perangkat *hypervisor* yang berada di Data Center dengan perangkat *hypervisor* di *Data Recovery Center*. Selain itu semua permintaan dari pengguna akan diterjemahkan dalam bentuk *parameter* yang dibutuhkan untuk meneruskan permintaan dari pengguna ke hypervisor. *Server middleware* memiliki alamat IP 10.199.16.43.

4.2.1 Skema Basis Data Middleware Menggunakan MySQL

Untuk mengelola data yang ada pada sistem, dibutuhkan basis data sebagai tempat penyimpanannya, yaitu MySQL. MySQL *server* yang digunakan adalah versi 5.7.28. Data yang disimpan pada basis data adalah data *hypervisor*, data riwayat pekerjaan, data perangkat *hypervisor*, dan data pasangan *backup-recovery virtual machine*.

4.2.1.1 Tabel *Hypervisors*

Pada tabel *hypervisors* menyimpan data perangkat yang terinstall *hypervisor*. Berikut definisi tabel *hypervisors* pada Tabel 4.1.

Tabel 4.1: Tabel Hypervisors

No	Kolom	Tipe	Keterangan
1	id_hypervisor	int(11)	Sebagai primary key pada tabel, nilai pada kolom ini ada dalam bentuk *integer*, dengan ketentuan *auto increment*.
2	domain_ip	varchar(45)	Menunjukan nama *hypervisor* yang didukung oleh sistem.
3	passwd	varchar(45)	Menunjukkan *password* dari perangkat *hypervisor*.
4	role	boolean	Menunjukkan peran perangkat hypervisor sebagai Data Center atau Datarecovery Center.

4.2.1.2 Tabel *Tasks*

Pada tabel *tasks* menyimpan data-data riwayat pekerjaan yang pernah atau sedang berjalan pada sistem. Berikut definisi tabel *tasks* pada Tabel 4.2.

Tabel 4.2: Tabel *Tasks*

No	Kolom	Tipe	Keterangan
1	id_tasks	int(11)	Sebagai primary key pada tabel, nilai pada kolom ini ada dalam bentuk *integer*, dengan ketentuan *auto increment*.

Tabel 4.2: Tabel *Tasks*

No	Kolom	Tipe	Keterangan
2	*id_vm_src*	int(11)	Menunjukkan vmid dari *virtual machine* yang akan di-*backup*.
3	*ip_src*	char(36)	Menunjukan alamat IP atau domain dari perangkat hypervisor pada *Data Center*.
4	*passwd_src*	char(36)	Menunjukan *password* dari perangkat hypervisor pada *Data Center*.
5	*id_vm_dst*	int(11)	Menunjukkan vmid dari *virtual machine* yang akan di-*restore*.
6	*ip_src*	char(36)	Menunjukan alamat IP atau domain dari perangkat hypervisor pada *Data Recovery Center*.
7	*passwd_dst*	char(36)	Menunjukan *password* dari perangkat hypervisor pada *Data Recovery Center*.
8	*status*	int	Menunjukan status pekerjaan. Status bernilai 0 apabila pekerjaan masih berjalan, bernilai 1 apabila perkerjaan selesai dan sukses dan bernilai -1 apabila pekerjaan gagal dikerjakan.

Tabel 4.2: Tabel *Tasks*

No	Kolom	Tipe	Keterangan
9	*start_time*	timestamp	Menunjukan waktu pekerjaan dimulai.
10	*end_time*	timestamp	Menunjukan waktu pekerjaan berakhir.
11	*job*	boolean	Menunjukan status pesan pekerjaan, bernilai 0 untuk backup dan *bernilai* 1 untuk *restore*.
12	*archive_name*	varchar(100)	Menunjukan nama *file backup* yang akan digunakan untuk proses *restore*.
13	*schedule_id*	int(11)	Menunjukkan *id_task* dari tabel *scheduled_task*.

4.2.1.3 Tabel *Scheduled Tasks*

Pada tabel *Scheduled tasks* menyimpan data-data pekerjaan dijalankan sebagai pekerjaan yang dijadwalkan pada sistem. Berikut definisi tabel *scheduled tasks* pada Tabel 4.3.

Tabel 4.3: Tabel *Scheduled_Tasks*

No	Kolom	Tipe	Keterangan
1	id_scheduled_task	int(11)	Sebagai primary key pada tabel, nilai pada kolom ini ada dalam bentuk *integer*, dengan ketentuan *auto increment*.

Tabel 4.3: Tabel *Scheduled_Tasks*

No	Kolom	Tipe	Keterangan
2	*id_vm_src*	int(11)	Menunjukkan vmid dari *virtual machine* yang akan di-*backup*.
3	*ip_src*	char(36)	Menunjukan alamat IP atau domain dari perangkat hypervisor pada *Data Center*.
4	*passwd_src*	char(36)	Menunjukan *password* dari perangkat hypervisor pada *Data Center*.
5	*id_vm_dst*	int(11)	Menunjukkan vmid dari *virtual machine* yang akan di-*restore*.
6	*ip_src*	char(36)	Menunjukan alamat IP atau domain dari perangkat hypervisor pada *Data Recovery Center*.
7	*passwd_dst*	char(36)	Menunjukan *password* dari perangkat hypervisor pada *Data Recovery Center*.
8	*job*	boolean	Menunjukan status pesan pekerjaan, bernilai 0 untuk backup dan *bernilai* 1 untuk *restore*.

Tabel 4.3: Tabel *Scheduled_Tasks*

No	Kolom	Tipe	Keterangan
9	*status_job*	varchar(45)	Menunjukan status pekerjaan apakah sudah dimasukkan ke dalam scheduler, belum dimasukkan ke scheduler, atau merupakan pekerjaan yang dihapus.
10	*hour*	char(10)	Menunjukan waktu jam pekerjaan dijadwalkan.
11	*minute*	char(10)	Menunjukan waktu menit pekerjaan dijadwalkan.

4.2.1.4 Tabel *Users*

Pada tabel *users* menyimpan data-data pengguna yang dapat mengakses sistem. Berikut definisi tabel *users* pada Tabel 4.4.

Tabel 4.4: Tabel *Users*

No	Kolom	Tipe	Keterangan
1	id	int(11)	Sebagai primary key pada tabel, nilai pada kolom ini ada dalam bentuk *integer*, dengan ketentuan *auto increment*.
2	*username*	varchar(45)	Menunjukan *username* pengguna yang digunakan untuk *login*.
3	*password*	varchar(100)	Menunjukan *password* pengguna yang digunakan untuk *login*.

Tabel 4.4: Tabel *Users*

No	Kolom	Tipe	Keterangan
4	*fullname*	varchar(45)	Menunjukan nama dari pengguna.
5	*is_admin*	int	Menunjukan jabatan dari pengguna. Apabila kolom *is_admin* tidak diisi maka akan bernilai 0.

4.2.1.5 Tabel *VMs*

Pada tabel *VMs* menyimpan data virtual machine yang akan di-*backup* maupun di-*restore*. Berikut definisi tabel *ip_address* pada Tabel 4.5.

Tabel 4.5: Tabel *VMs*

No	Kolom	Tipe	Keterangan
1	id	char(36)	Sebagai primary key pada tabel, nilai pada kolom ini ada dalam bentuk *Universally Unique Identifier (UUID)*.
2	*ip_node*	char(36)	Menunjukan *foreign key* alamat IP hypervisor.
3	*id_vm*	int(11)	Menunjukan vmid dari *virtual machine*.

4.2.1.6 Tabel *Paired VMs*

Pada tabel *paired_vms* menyimpan data pasangan virtual machine yang akan dieksekusi untuk proses *backup-restore*. Berikut definisi tabel *paired_vms* pada Tabel 4.6.

Tabel 4.6: Tabel *Paired_VMs*

No	Kolom	Tipe	Keterangan
1	id_paired_vm	int(11)	Sebagai primary key pada tabel, nilai pada kolom ini ada dalam bentuk *integer*, dengan ketentuan *auto increment*.
2	*id_vm_src*	int(11)	Menunjukkan vmid dari *virtual machine* yang akan di-*backup*.
3	*ip_src*	char(36)	Menunjukan alamat IP atau domain dari perangkat hypervisor pada *Data Center*.
4	*id_vm_dst*	int(11)	Menunjukkan vmid dari *virtual machine* yang akan di-*restore*.
5	*ip_src*	char(36)	Menunjukan alamat IP atau domain dari perangkat hypervisor pada *Data Recovery Center*.

4.2.2 Implementasi *Endpoint* pada *Middleware*

Untuk mempermudah manajemen sistem, penulis menyediakan *end-point* pada *middleware*.

4.2.2.1 *End-Point* Autentikasi *User*

Untuk autentikasi *user*, 3 operasi yang didapat dilakukan yaitu registrasi *user*, *login* akun *user*, dan *logout* akun *user*. Untuk detail *end-point* dapat dilihat pada Tabel 4.7.

Tabel 4.7: Tabel *End-point* Autentikasi *User*

No	***End-point***	***Method***	**Keterangan**
1	`/hypervisors`	*GET*	Untuk melihat daftar perangkat *hypervisor*.
2	`/hypervisors/add`	POST	Untuk menambah perangkat *hypervisor*.
3	`/hypervisors/edit/<id_hypervisor>`	POST	Untuk mengedit perangkat *virtual machine*.

4.2.2.2 *End-Point* Manajemen Perangkat *Hypervisor*

Untuk manajemen perangkat *hypervisor*, terdapat 4 operasi yang didapat dilakukan yaitu melihat daftar perangkat *hypervisor*, menambah perangkat *hypervisor*, menghapus perangkat *hypervisor*, dan mengedit perangkat *hypervisor*. Untuk detail *end-point* dapat dilihat pada Tabel 4.8.

Tabel 4.8: Tabel *End-point* Manajemen Perangkat *Hypervisor*

No	***End-point***	***Method***	**Keterangan**
1	`/hypervisors`	*GET*	Untuk melihat daftar perangkat *hypervisor*.
2	`/hypervisors/add`	POST	Untuk menambah perangkat *hypervisor*.
3	`/hypervisors/edit/<id_hypervisor>`	POST	Untuk mengedit perangkat *virtual machine*.
4	`/hypervisors/delete/<id_hypervisor>`	DELETE	Untuk menghapus perangkat *hypervisor*.

4.2.2.3 *End-Point* Manajemen Pasangan *Backup-Recovery Virtual Machine*

Untuk manajemen pasangan *backup-recovery virtual machine*, terdapat 4 operasi yang dapat dilakukan yaitu melihat daftar pasangan *backup-recovery virtual machine*, menambah pasangan *backup-recovery virtual machine*, menghapus pasangan *backup-recovery virtual machine*, dan mengedit pasangan *backup-recovery virtual machine*. Untuk detail *end-point* dapat dilihat pada Tabel 4.9.

Tabel 4.9: Tabel *End-point* Manajemen Pasangan *Virtual Machine*

No	*End-point*	*Method*	Keterangan
1	`/node-pairs`	*GET*	Untuk melihat daftar pasangan *backup-recovery virtual machine*.
2	`/node-pairs/add`	POST	Untuk menambah pasangan *backup-recovery virtual machine*.
3	`/node-pairs/edit/<id_paired_vm>`	POST	Untuk mengedit pasangan *backup-recovery virtual machine*.
4	`/node-pairs/delete/<id_paired_vm>`	DELETE	Untuk menghapus pasangan *backup-recovery virtual machine*.

4.2.2.4 *End-Point* Melihat *Task*

Untuk melihat daftar *task*, pengguna dapat mengakses *end-point /tasks* dengan *method GET*.

4.2.2.5 *End-Point* Manajemen *Scheduled Task*

Untuk manajemen *scheduled task* daftar *scheduled task*, terdapat 3 operasi yang dapat dilakukan oleh pengguna, yaitu melihat daftar *scheduled tasks*, menambah *scheduled task*, dan menghapus *scheduled task*.Untuk detail *end-point* dapat dilihat pada Tabel 4.10.

Tabel 4.10: Tabel *End-point* Manajemen *Scheduled Tasks*

No	*End-point*	*Method*	Keterangan
1	`/scheduled_tasks/`	*GET*	Untuk melihat daftar *scheduled tasks* yang telah dibuat oleh admin.
2	`/tasks/create_scheduled_backup/<id_paired_vm>`	POST	Untuk menambah *scheduled task*.
3	`/delete_scheduled_task/<id_paired_vm>`	DELETE	Untuk menghapus *scheduled task*.

4.2.3 Implementasi Proses *Backup-Recovery Virtual Machine*

Proses *backup* atau *recovery*, akan diekseskusi oleh sebuah *worker* berupa proses dalam middleware yang akan melakukan iterasi pada *query* untuk memeriksa *task* yang belum dieksekusi pada tabel *tasks* di basis data MySQL. *Worker* merupakan sebuah proses yang berjalan terpisah dari layanan *HTTP Rest API*.

Worker mempunyai beberapa fungsi untuk mengeksekusi proses *backup* dari *Data Center* maupun *restore* ke *Data*

Recovery Center. Untuk melakukan proses *backup*, worker akan menjalankan fungsi yang dapat dilihat pada Kode Sumber IV.1.

```
FUNCTION backup(ip, password, id_vm, schedule_id=
    None):
  status <- None
  start_time <- time.time()
  try:
    subprocess.check_output([PATH_SCRIPT_BACKUP,
       ip, password, id_vm])
    IF schedule_id:
      None
    ENDIF
    status <- 1
  except subprocess.CalledProcessError as e:
    OUTPUT e.output
    status <- -1
  end_time <- time.time()
  export_log(ip, None, password, id_vm, None,
     start_time, end_time)
  RETURN status
ENDFUNCTION
```

Kode Sumber IV.1: Fungsi Proses *Backup*

Worker akan mengirimkan perintah ke perangkat *hypervisor Data Center* untuk mengeksekusi backup menggunakan vzdump. Perintah yang dijalankan oleh perangkat *hypervisor* dapat dilihat pada Kode Sumber **??**.

```
vzdump --dumpdir /home/backups/<VMID_BACKUP> --
    compress lzo VMID_BACKUP
```

Kode Sumber IV.2: Fungsi Proses *Restore*

VMID_BACKUP merupakan argumen untuk mendapatkan vmid dari virtual machine yang akan di-*backup*. Untuk menjalankan

proses *restore*, *worker* akan mengirimkan perintah menggunakan fungsi. Pseudocode yang dijalankan fungsi *restore* dapat dilihat pada Kode Sumber IV.3.

```
1  FUNCTION restore(ip_src, ip_dst, password,
       id_vm_src, id_vm_dst, archive_name):
2    status <- None
3    start_time <- time.time()
4    try:
5      subprocess.check_output([PATH_SCRIPT_RESTORE,
           ip_src, ip_dst, password, id_vm_src,
          id_vm_dst, archive_name])
6      status <- 1
7    except subprocess.CalledProcessError as e:
8      OUTPUT e.output
9      status <- -1
10   end_time <- time.time()
11   export_log(ip_src, ip_dst, password, id_vm_src,
         id_vm_dst, start_time, end_time)
12   RETURN status
13 ENDFUNCTION
```

Kode Sumber IV.3: Fungsi Proses *Restore*

Setelah mengeksekusi *task*, *worker* akan memperbaharui status task pada tabel, menggunakan fungsi. Pseudocode yang dijalankan untuk memperbaharui status *task* dapat dilihat pada Kode Sumber IV.4.

```
1  FUNCTION update_status(id_task, status):
2    db <- MySQLdb.connect(host=HOST,user=USERNAME,
       passwd=PASS,db=DB)
3    cursor <- db.cursor()
4    sql <- "UPDATE tasks SET status <- " + str(
       status) + " WHERE id_tasks <- " + str(
       id_task) + ";"
```

```
  try:
    cursor.execute(sql)
    db.commit()
    OUTPUT "Number of rows updated: " + str(
        cursor.rowcount)
  except:
    OUTPUT "update_status Error! Unable to fetch
        data"
  cursor.close()
  db.close()
  RETURN
ENDFUNCTION
```

Kode Sumber IV.4: Fungsi Memperbaharui Status *Task*

Inisialisasi *worker* sebagai proses terdapat pada fungsi utama. Pseudocode worker sebagai proses dapat dilihat pada Kode Sumber IV.5.

```
add_all_schedule()
try:
  while True:
    OUTPUT "Current Jobs:"
    OUTPUT schedule.jobs
    schedule.run_pending()
    add_new_schedule()
    delete_schedule()
    new_task <- check_new_task()
    IF is_empty(new_task):
      OUTPUT str(datetime.now()) + " no new task
          ."
    ELSE:
      id_task <- new_task[0]['id_tasks']
      status <- 2
      update_status(id_task, status)
```

```
16        update_time(id_task, 0)
17        IF int(new_task[0]['job']) = 0:
18          ip <- str(new_task[0]['ip_src'])
19          password <- str(new_task[0]['passwd_src']
                )
20          id_vm <- str(new_task[0]['id_vm_src'])
21          OUTPUT "Backing up " + ip + " VM " +
                id_vm + "..."
22          status <- backup(ip, password, id_vm)
23        ELSE:
24          ip_src <- str(new_task[0]['ip_src'])
25          ip_dst <- str(new_task[0]['ip_dst'])
26          password <- str(new_task[0]['passwd_dst']
                )
27          id_vm_src <- str(new_task[0]['id_vm_src']
                )
28          id_vm_dst <- str(new_task[0]['id_vm_dst']
                )
29          archive_name <- str(new_task[0]['
                archive_name'])
30          OUTPUT "Restoring from " + ip_src + " VM
                " + id_vm_src + " to " + ip_dst + " VM
                 " + id_vm_dst + "..."
31          status <- restore(ip_src, ip_dst,
                password, id_vm_src, id_vm_dst,
                archive_name)
32
33        ENDIF
34        update_status(id_task, status)
35        update_time(id_task, 1)
36      ENDIF
37      time.sleep(10)
38    ENDWHILE
```

```
except KeyboardInterrupt:
  pass
```

Kode Sumber IV.5: Fungsi Inisialisasi *Worker*

4.2.4 Implementasi *Scheduled Tasks*

Scheduled tasks yang telah dijadwalkan oleh pengguna akan dimasukkan ke tabel scheduled_tasks. *Worker* menggunakan modul *schedule* dari Pyhton untuk melakukan penjadwalan. Modul *schedule* berguna untuk mendaftarkan *task* yang terjadwal ke scheduler agar dapat dieksekusi sesuai dengan waktu yang dijadwalkan. *Worker* akan menjalankan sebuah fungsi untuk memasukkan data dari tabel *scheduled_tasks* ke *scheduler*. Pseudocode fungsi tersebut dapat dilihat pada Kode Sumber IV.6.

```
FUNCTION add_all_schedule():
  db <- MySQLdb.connect(host=HOST,user=USERNAME,
     passwd=PASS,db=DB)
  cursor <- db.cursor()
  sql <- "SELECT * FROM scheduled_tasks WHERE
     status_job <- 'inserted';"
  try:
    cursor.execute(sql)
    results <- cursor.fetchall()
    for row in results:
      id_schedule <- row[0]
      id_vm_src <- row[1]
      ip_src <- row[2]
      pass_src <- row[3]
      hour <- row[9]
      minute <- row[10]
      schedule.every().day.at(hour+':'+minute).do
```

```
            (insert_new_schedule_to_task,
            schedule_id=id_schedule).tag(str(
            id_schedule))
        OUTPUT "Add schedule " + str(id_schedule)
      ENDFOR
    except:
      OUTPUT "add_all_schedule Error! Unable to
          fetch data"
    cursor.close()
    db.close()
    RETURN
ENDFUNCTION
```

Kode Sumber IV.6: Fungsi Mendaftarkan *Scheduled Tasks* ke *Scheduler*

Ketika waktu yang dijadwalkan sesuai dengan waktu pada sistem, worker menjalankan fungsi untuk memasukkan task tersebut ke tabel *tasks* untuk dieksekusi. Fungsi untuk mendaftarkan *scheduled task* dapat dilihat pada Kode Sumber IV.7.

```
FUNCTION insert_new_schedule_to_task(schedule_id)
    :
db <- MySQLdb.connect(host=HOST,user=USERNAME,
    passwd=PASS,db=DB)
cursor <- db.cursor()
  sql <- "SELECT * FROM scheduled_tasks WHERE
      id_scheduled_task <- " + str(schedule_id) +
      ";"
  cursor.execute(sql)
  row <- cursor.fetchone()
  id_schedule <- row[0]
  id_vm_src <- row[1]
  ip_src <- row[2]
  pass_src <- row[3]
```

```
11    id_vm_dst <- row[4]
12    ip_dst <- row[5]
13    pass_dst <- row[6]
14    cursor_ins <- db.cursor()
15    sql <- "INSERT INTO tasks (id_vm_src,ip_src,
          passwd_src,id_vm_dst,ip_dst,passwd_dst,job,
          schedule_id,status) VALUES ('" + str(
          id_vm_src) + "','" + \
16    str(ip_src) + "','" + str(pass_src) + "'," +
          str(id_vm_dst) + ",'" + str(ip_dst) + "','"
          + \
17    str(pass_dst) + "',0," + str(id_schedule) +",0)
          ;"
18    cursor_ins.execute(sql)
19    cursor_ins.close()
20    db.commit()
21  ENDFUNCTION
```

Kode Sumber IV.7: Fungsi Mengeksekusi *Scheduled Task*

(Halaman ini sengaja dikosongkan)

BAB V

PENGUJIAN DAN EVALUASI

5.1 Lingkungan Uji Coba

Lingkungan pengujian menggunakan komponen-komponen yang terdiri dari: satu server *middleware*, dua server *proxmox*, dan Komputer Penguji. Spesifikasi untuk setiap komponen yang digunakan ditunjukkan pada Tabel 5.1.

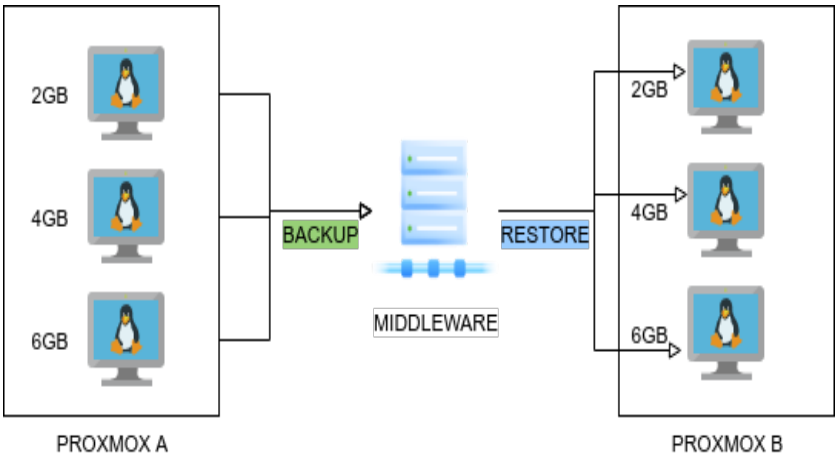
Tabel 5.1: Spesifikasi Komponen

No	Komponen	Perangkat Keras	Perangkat Lunak
1	Middleware	8 core processor, 4GB RAM	Ubuntu-server 18.04, Python 3.6
2	Proxmox A	4 core processor, 12GB RAM	*Hypervisor* Proxmox
3	Proxmox B	4 core processor, 8GB RAM	*Hypervisor* Proxmox

Spesifikasi *virtual machine* yang digunakan untuk uji coba adalah :

1. Sistem Operasi Ubuntu-server 16.04.
2. 4 core processor.
3. 2GB RAM.

Arsitektur lingkungan uji coba dapat dilihat pada Gambar 5.1.



Gambar 5.1: Arsitektur *Guest Virtual Machine* Pengujian

5.2 Skenario Uji Coba

Uji coba akan dilakukan untuk mengetahui keberhasilan sistem yang telah dibangun. Skenario pengujian dibedakan menjadi 2 bagian, yaitu:

- **Uji Fungsionalitas**
  Pengujian ini didasrkan pada fungsionalitas yang disajikan sistem.
- **Uji Performa**
  Pengujian ini untuk menguji performa sistem dengan spesifikasi *virtual machine* yang beragam terhadap waktu eksekusi *backup* dan *restore*, memori, dan lama waktu transfer. Pengujian dilakukan dengan melakukan *benchmark* pada sistem.

5.2.1 Skenario Uji Coba Fungsionalitas

Uji fungsionalitas akan diuji dengan mengelola sistem menggunakan antarmuka web.

5.2.1.1 Uji Fungsionalitas Mengelola Sistem Menggunakan Antarmuka Web

Pengujian fungsionalitas selanjutnya, penulis menguji fitur-fitur pengelolaan sistem melalui *interface* web. Rancangan pengujian dan hasil yang diharapkan ditunjukkan pada Tabel 5.2.

Tabel 5.2: Skenario Uji Fungsionalitas Mengelola Sistem Menggunakan Antarmuka Web

No	Menu	Uji Coba	Hasil Harapan
1	*Hypervisors*	Menambahkan perangkat *hypervisor* baru.	Data perangkat *hypervisor* baru dapat diteruskan oleh *interface* web ke middleware dan disimpan pada basis data sistem.
		Melihat daftar perangkat *hypervisor* yang tersedia.	Pengguna dapat melihat daftar perangkat *hypervisor* yang tersedia pada sistem.
		Memperbaharui data perangkat *hypervisor* terpilih.	Pengguna dapat melakukan perubahan data pada perangkat *hypervisor* yang dipilih.

Tabel 5.2: Skenario Uji Fungsionalitas Mengelola Sistem Menggunakan Antarmuka Web

No	Menu	Uji Coba	Hasil Harapan
		Menghapus data perangkat *hypervisor* terpilih.	Pengguna dapat menghapus data perangkat *hypervisor* yang dipilih.
2	*Node Pairs*	Menambahkan pasangan *backup-recovery virtual machine* baru.	Data pasangan *backup-recovery virtual machine* baru dapat diteruskan oleh *interface* web ke middleware dan disimpan pada basis data sistem.
		Melihat daftar pasangan *backup-recovery virtual machine* yang tersedia.	Pengguna dapat melihat daftar pasangan *backup-recovery virtual machine* yang tersedia pada sistem.
		Memperbaharui data pasangan *backup-recovery virtual machine* terpilih.	Pengguna dapat melakukan perubahan data pada pasangan *backup-recovery virtual machine* yang dipilih.

Tabel 5.2: Skenario Uji Fungsionalitas Mengelola Sistem Menggunakan Antarmuka Web

No	Menu	Uji Coba	Hasil Harapan
		Menghapus data pasangan *backup-recovery virtual machine* terpilih.	Pengguna dapat menghapus data pasangan *backup-recovery virtual machine* yang dipilih.
3	*User*	Mendaftarkan pengguna pada sistem.	Data pengguna baru dapat disimpan pada basis data sistem.
		Autentikasi pengguna menggunakan akun yang didaftarkan	Pengguna dapat *login* dengan akun untuk menggunakan sistem.
		Keluar autentikasi pengguna menggunakan akun yang sedang digunakan	Pengguna dapat *logout* dari sitem menggunakan akun yang sedang digunakan.
4	*Task*	Menambah *task backup* dengan pasangan *virtual machine* yang terdaftar.	Pengguna dapat menambahkan *task backup* ke *list tasks*.
		Menambah *task recovery* dengan pasangan *virtual machine* yang terdaftar.	Pengguna dapat menambahkan *task recovery* ke *list tasks*.

Tabel 5.2: Skenario Uji Fungsionalitas Mengelola Sistem Menggunakan Antarmuka Web

No	Menu	Uji Coba	Hasil Harapan
		Menambah *scheduled task backup* dengan pasangan *virtual machine* yang terdaftar.	Pengguna dapat menambahkan *scheduled task backup* ke *list scheduled tasks*.
		Melihat daftar *tasks*	Pengguna dapat melihat riwayat daftar *tasks*.

5.2.2 Skenario Uji Coba Performa

Uji performa digunakan untuk menguji bagaimana ketahanan sistem dalam proses *backup* dan *restore virtual machine* dari Proxmox A ke Proxmox B. Uji performa dilakukan untuk mengukur proses waktu backup-recovery dan penanganan scheduled tasks yang terjadwal pada satu waktu.

5.2.2.1 Uji Performa Kecepatan Proses *Backup-Recovery Virtual Machine*

Pengujian dilakukan dengan mengukur waktu proses *backup*, ukuran *file backup*, proses transfer *file* dan mengukur waktu proses *restore virtual machine* dengan variasi ukuran *storage* yang digunakan oleh *virtual machine* yaitu 2GB, 4GB, dan 6GB. Hasil yang diharapkan dari pengujian ini adalah sistem memiliki kecepatan yang baik dan *storage* yang cukup untuk menangani proses *backup* sampai *recovery virtual machine*.

5.2.2.2 Uji Performa Penanganan *Scheduled Tasks*

Pengujian dilakukan dengan mengatur *tasks* pada jadwal yang sama untuk variasi jumlah *virtual machine* yang berbeda yaitu penjadwalan 1, 2, 3, dan 4 *virtual machine* ke satu penjadwalan waktu yang sama. Hasil yang diharapkan dari pengujian ini adalah sistem dapat menjalankan semua *scheduled tasks* berdasarkan antrian.

5.3 Hasil Uji Coba dan Evaluasi

Berikut dijelaskan hasil uji coba dan evaluasi berdasarkan skenario yang telah dijelaskan pada subbab 5.2.

5.3.1 Hasil Uji Fungsionalitas

Berikut dijelaskan hasil pengujian fungsionalitas pada sistem yang dibangun.

5.3.1.1 Hasil Uji Mengelola Sistem Menggunakan Antarmuka Web

Pengujian dilakukan sesuai dengan skenario yang dijelaskan pada subbab 5.2.1.1 dan pada Tabel 5.2. Hasil pengujian seperti tertera pada Tabel 5.3.

Tabel 5.3: Hasil Uji Coba Mengelola Sistem Menggunakan Antarmuka Web

No	Menu	Uji Coba	Hasil
1	*Hypervisors*	Menambahkan perangkat *hypervisor* baru.	Berhasil.
		Melihat daftar perangkat *hypervisor* yang tersedia.	Berhasil.

Tabel 5.3: Hasil Uji Coba Mengelola Sistem Menggunakan Antarmuka Web

No	Menu	Uji Coba	Hasil
		Memperbaharui data perangkat *hypervisor* terpilih.	Berhasil.
		Menghapus data perangkat *hypervisor* terpilih.	Berhasil.
2	*Node Pairs*	Menambahkan pasangan *backup-recovery virtual machine* baru.	Berhasil.
		Melihat daftar pasangan *backup-recovery virtual machine* yang tersedia.	Berhasil.
		Memperbaharui data pasangan *backup-recovery virtual machine* terpilih.	Berhasil.
		Menghapus data pasangan *backup-recovery virtual machine* terpilih.	Berhasil.
3	*User*	Mendaftarkan pengguna pada sistem.	Berhasil.
		Autentikasi pengguna menggunakan akun yang didaftarkan	Berhasil.

Tabel 5.3: Hasil Uji Coba Mengelola Sistem Menggunakan Antarmuka Web

No	Menu	Uji Coba	Hasil
		Keluar autentikasi pengguna menggunakan akun yang sedang digunakan	Berhasil.
4	*Task*	Menambah *task backup* dengan pasangan *virtual machine* yang terdaftar.	Berhasil.
		Menambah *task recovery* dengan pasangan *virtual machine* yang terdaftar.	Berhasil.
		Menambah *scheduled task backup* dengan pasangan *virtual machine* yang terdaftar.	Berhasil.
		Melihat daftar *tasks*	Berhasil.

Sesuai dengan skenario uji coba yang diberikan pada Tabel 5.2, hasil uji coba menunjukkan semua skenario berhasil ditangani.

5.3.2 Hasil Uji Performa

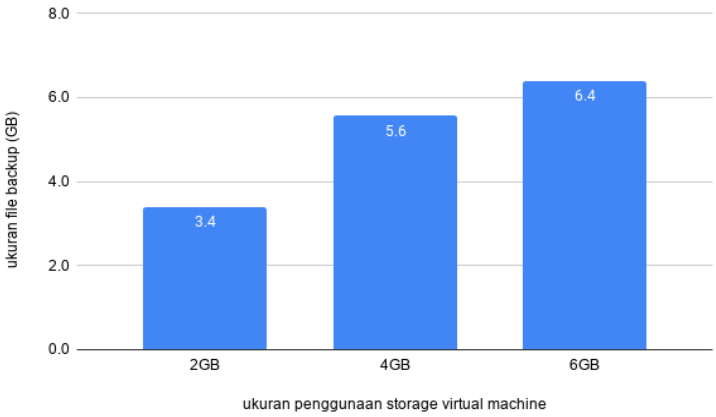
Berikut dijelaskan hasil pengujian performa pada sistem yang dibangun.

5.3.2.1 Uji Performa Kecepatan Proses *Backup-Recovery Virtual Machine*

Dari hasil uji coba, lama waktu *backup-recovery virtual machine* dapat mempengaruhi hasil uji coba performa sistem. Hasil uji coba pengukuran *size file backup*, dapat dilihat pada Tabel 5.4 dalam satuan *Gigabyte* (GB).

Tabel 5.4: Hasil Uji Coba Ukuran *File Backup* terhadap Ukuran *Storage* yang Digunakan *Virtual Machine*

No	Ukuran *Storage* yang digunakan	Ukuran *File Backup*
1	2GB	3.4GB
2	4GB	5.6GB
3	6GB	6.4GB



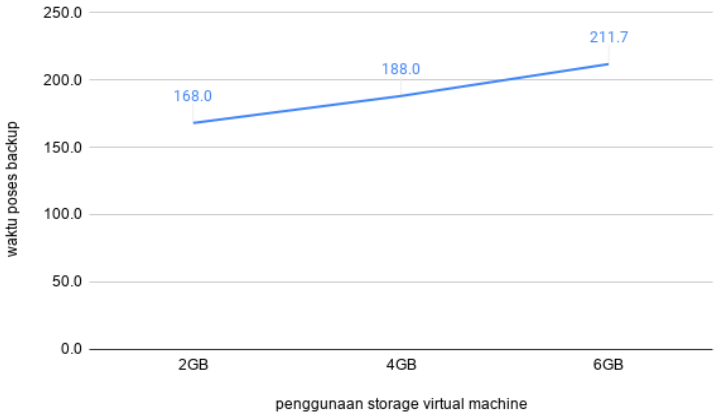
Gambar 5.2: Ukuran *File Backup* terhadap *Storage* yang Digunakan *Virtual Machine*

Hasil uji coba kecepatan melakukan proses *backup virtual*

machine, dapat dilihat pada Tabel 5.5 dalam satuan detik.

Tabel 5.5: Hasil Uji Coba Kecepatan Proses *Backup* terhadap *Storage* yang Digunakan *Virtual Machine*

No	Ukuran *Storage* yang digunakan	Waktu *Backup*
1	2GB	168.0s
2	4GB	188.0s
3	6GB	211.7s

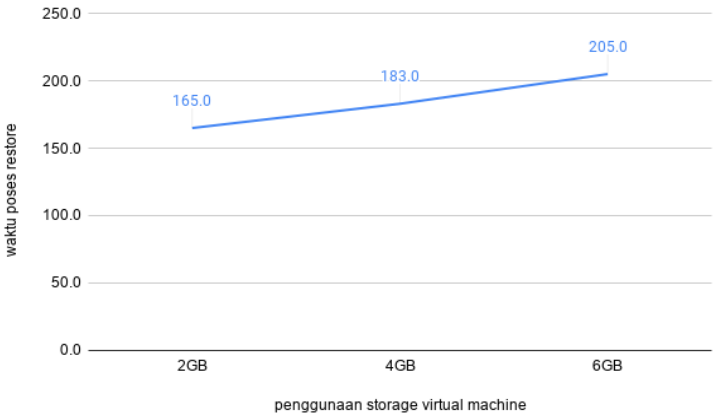


Gambar 5.3: Waktu Rata-rata Proses *Backup* terhadap *Storage* yang Digunakan *Virtual Machine*

Hasil uji coba kecepatan proses *restore virtual machine*, dapat dilihat pada Tabel 5.6 dalam satuan detik.

Tabel 5.6: Hasil Uji Coba Kecepatan Proses *Restore* terhadap *Storage* yang digunakan *Virtual Machine*

No	Ukuran *Storage* yang digunakan	Waktu *Restore*
1	2GB	165.0s
2	4GB	183.0s
3	6GB	205.0s



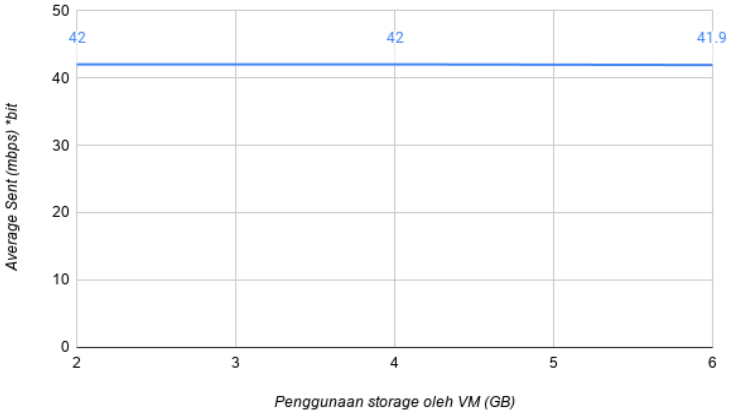
Gambar 5.4: Waktu Rata-rata Proses *Restore* terhadap Ukuran *Storage* yang Digunakan *Virtual Machine*

Gambar 5.2, Gambar 5.3, dan Gambar 5.4 menunjukkan bahwa penggunaan *storage* pada *virtual machine* berpengaruh terhadap proses *backup* sampai *restore virtual machine*. Perbedaan ukuran *file backup* dengan ukuran penggunaan *storage virtual machine* tersebut terjadi karena *file backup* dari *virtual machine* terdiri dari konfigurasi *hypervisor* untuk teknik virtualisasi terhadap *virtual machine* tersebut. Hasil pengujian membuktikan adanya kenaikan waktu proses *backup* dan *storage*

secara linear sejalan dengan bertambahnya ukuran penggunaan *storage* di *virtual machine*. Lama waktu *backup* tersebut dipengaruhi oleh penggunaan *storage* oleh *virtual machine* karena membutuhkan waktu untuk melakukan kompresi terhadap konfigurasi virtualisasi dan *virtual disk* dari *virtual machine* tersebut. Sama halnya dengan waktu *backup*, untuk melakukan *restore virtual machine*, *hypervisor* harus melakukan dekompresi dari *file backup* untuk menjalankan *virtual machine* pada perangkat *Data Recovery Center*. Proses transfer backup dimulai setelah proses kompresi selesai yang ditandai dengan menurunnya aktivitas penggunaan CPU dan naiknya utilitas jaringan secara signifikan. Hasil uji coba kecepatan transfer file backup terhadap Ukuran *Storage* yang Digunakan *Virtual Machine* dapat dilihat pada Tabel 5.7 dalam satuan *Megabit per second* (Mbps).

Tabel 5.7: Hasil Uji Coba Kecepatan Transfer *File Backup* terhadap *Storage* yang digunakan *Virtual Machine*

No	**Ukuran *Storage* yang digunakan**	**Kecepatan Transfer**
1	2GB	42.0s
2	4GB	42.0s
3	6GB	41.9s



Gambar 5.5: Kecepatan Transfer *File Backup* terhadap Ukuran *Storage* yang Digunakan *Virtual Machine*

Pada pengujian ini, perangkat yang digunakan memiliki kecepatan download sebesar 42.36 Mbps. Jika dibandingkan dengan hasil pengujian proses transfer, utilitas jaringan adalah 99%. Dengan demikian, proses transfer menggunakan *bandwidth* maksimal yang dapat dipakai oleh kedua perangkat pengirim dan penerima *file backup*.

5.3.2.2 Uji Performa Penanganan *Scheduled Tasks*

Hasil uji coba penjadwalan *tasks backup* dalam satu waktu dapat dilihat pada Tabel 5.8 dalam satuan detik.

Tabel 5.8: Hasil Uji Coba Penanganan *Scheduled Tasks*

No	Jumlah *Virtual Machine* Terjadwal	Total Waktu Penanganan
1	1	257s

Tabel 5.8: Hasil Uji Coba Penanganan *Scheduled Tasks*

No	Jumlah *Virtual Machine* Terjadwal	Total Waktu Penanganan
2	2	510s
3	3	765s
4	4	1000s

Gambar 5.6: Waktu Penanganan *Scheduled Tasks* terhadap Jumlah Penjadwalan Virtual Machine dalam Satu Waktu

Gambar 5.6 menunjukkan bahwa waktu penanganan *scheduled tasks* pada satu waktu memiliki kenaikan secara linear sesuai dengan bertambahnya jumlah *virtual machine* yang dieksekusi. Hal ini terjadi karena *worker* bekerja secara sekuensial untuk menangani *task* yang ada pada antrian.

(Halaman ini sengaja dikosongkan)

BAB VI

PENUTUP

Bab ini membahas kesimpulan yang dapat diambil dari tujuan pembuatan sistem dan hubungannya dengan hasil uji coba dan evaluasi yang telah dilakukan. Selain itu, terdapat beberapa saran yang bisa dijadikan acuan untuk melakukan pengembangan dan penelitian lebih lanjut.

6.1 Kesimpulan

Dari proses perancangan, implementasi dan pengujian terhadap sistem, dapat diambil beberapa kesimpulan berikut:

1. Abstraksi switch dapat diimplementasikan pada *middleware* dengan menjalankan proses berupa *worker* yang akan mengirimkan perintah ke perangkat *Data Center* dan *Data Recovery Center* berdasarkan antrian *tasks* dari basis data MySQL yang didapatkan melalui layanan *HTTP Rest API*.
2. Proses *backup-recovery virtual machine* pada suatu lingkungan hypervisor yang sama, yaitu Proxmox dapat diimplementasikan dengan mengirimkan perintah oleh *worker* untuk menjalankan *backup* dan *restore* menggunakan utilitas VZdump dan melakukan transfer *file* menggunakan Rsync.
3. Hal yang mempengaruhi waktu proses *backup* maupun *recovery virtual machine* yaitu ukuran penyimpanan data yang digunakan oleh *virtual machine* tersebut. Hal tersebut juga berpengaruh terhadap ukuran file backup dari virtual machine.
4. Dari hasil pengujian, semakin besar ukuran *storage* yang digunakan oleh *virtual machine*, maka proses *backup* maupun *recovery* akan semakin lama dan mempunyai ukuran *file backup* yang semakin besar.

6.2 Saran

Berikut beberapa saran yang diberikan untuk pengembangan lebih lanjut:

1. Perlu penanganan untuk melakukan proses *backup-recovery* dengan ragam *hypervisor* selain Proxmox.
2. Perlu mekanisme untuk proses *provisioning* jaringan terhadap *virtual machine* yang ditangani pada proses *recovery virtual machine*.

DAFTAR PUSTAKA

[1] N. Jain dan S. Choudhary, "Overview of Virtualization in Cloud Computing," in *Colossal Data Analysis and Networking (CDAN), Symposium on*, 2012.

[2] "General Python FAQ," 3 Mei 2018. [Daring]. Tersedia pada: https://docs.python.org/3/faq/general.html#what-is-python. [Diakses: 3 Mei 2018].

[3] "A simple framework for building complex web applications," 3 Mei 2018. [Daring]. Tersedia pada: https://pypi.org/project/Flask/. [Diakses: 3 Mei 2018].

[4] "rsync," Jan. 2020, 20 Januari 2020. [Daring]. Tersedia pada: https://rsync.samba.org/. [Diakses: 20 Januari 2020].

[5] "Backup of a running container with vzdump," Jan. 2020, 20 Januari 2020. [Daring]. Tersedia pada: https://wiki.openvz.org/Backup_of_a_running_container_with_vzdump. [Diakses: 20 Januari 2020].

[6] "Python Wrapper for the Proxmox 2.x API (HTTP and SSH)," 3 Mei 2018. [Daring]. Tersedia pada: https://pypi.org/project/proxmoxer/. [Diakses: 3 Mei 2018].

[7] W. J. Gilmore, "Beginning PHP and MySQL From Novice to Professional," *Apress*, vol. 4th, hal. 477–480, 2010.

(Halaman ini sengaja dikosongkan)

LAMPIRAN A

INSTALASI PERANGKAT LUNAK

Instalasi Pustaka Python

Dalam pengembangan sistem ini, digunakan berbagai pustaka pendukung. Pustaka pendukung yang digunakan merupakan pustaka untuk bahasa pemrograman Python. Berikut adalah daftar pustaka yang digunakan dan cara pemasangannya:

- Python Dev
  $ `sudo apt-get install python-dev`
- Flask
  $ `sudo pip install Flask`
- Proxmoxer
  $ `sudo pip install proxmoxer`
- Schedule
  $ `sudo pip install schedule`
- MySQLdb
  $ `sudo apt-get install python-mysqldb`

Instalasi Rsync

Dalam pengembangan sistem ini, digunakan utilitas untuk proses transfer *file* antar perangkat, yaitu Rsync. Rsync dapat dipasang pada setiap perangkat dalam komponen sistem ini menggunakan perintah berikut:

$ `sudo apt-get install rsync`

(Halaman ini sengaja dikosongkan)

LAMPIRAN B

KODE SUMBER

File Environment Middleware

```
DB_USERNAME=root
DB_PASSWORD=node
DB_NAME=db_drs
DB_HOST=localhost

BACKUP_DIR=/home/nahda/
```

Kode Sumber B.1: File Environment Middleware (.env)

Inisialisasi Perangkat Hypervisor Proxmox

```
1  from proxmoxer import ProxmoxAPI
2
3  class Connect_Proxmox():
4  """docstring for Connect_Proxmox"""
5  def __init__(self, ip, username, password):
6    self.ip = ip
7    self.username = username
8    self.password = password
9
10 def connect(self):
11 try:
12   proxmox = ProxmoxAPI(self.ip, user=self.
         username+"@pam",
13   password=self.password, verify_ssl=False)
14 except Exception as e:
15   return False, e, None
16 return True, None, proxmox
17
18 def get_vm_id(ip, username, password):
```

```
19    vm_id = []
20    status, message, proxmox = Connect_Proxmox(ip,
          username, password).connect()
21    if status is False:
22      return vm_id
23    try:
24      for vm in proxmox.cluster.resources.get(type
            ='vm'):
25      vm_id.append(vm['vmid'])
26    except Exception as e:
27      raise e
28    return vm_id
```

Kode Sumber B.2: File Inisialisasi Proxmox (proxmox.py)

Inisialisasi Model Basis Data

```
1  from flask_login import UserMixin
2  from werkzeug.security import
       generate_password_hash, check_password_hash
3
4  from app import db, login_manager
5  from os import listdir, getenv
6  from os.path import isfile, join
7
8  class User(UserMixin, db.Model):
9  """
10 Create a User table
11 """
12 __tablename__ = 'users'
13
14 id = db.Column(db.Integer, primary_key=True)
```

```
15 username = db.Column(db.String(60), index=True,
       unique=True)
16 full_name = db.Column(db.String(60), index=True)
17 password_hash = db.Column(db.String(128))
18 is_admin = db.Column(db.Boolean, default=True)
19 is_active = db.Column(db.Boolean, default=True)
20
21 @property
22 def password(self):
23 """
24 Prevent pasword from being accessed
25 """
26 raise AttributeError('password is not a readable
       attribute.')
27
28 @password.setter
29 def password(self, password):
30 """
31 Set password to a hashed password
32 """
33 self.password_hash = generate_password_hash(
       password)
34
35 def verify_password(self, password):
36 """
37 Check if hashed password matches actual password
38 """
39 return check_password_hash(self.password_hash,
       password)
40
41 def __repr__(self):
42 return '<User: {}>'.format(self.username)
43
```

```
44
45 # Set up user_loader
46 @login_manager.user_loader
47 def load_user(user_id):
48 return User.query.get(int(user_id))
49
50
51 class Hypervisor(db.Model):
52 """
53 Create a Node table
54 """
55 __tablename__ = 'hypervisors'
56
57 id_hypervisor = db.Column(db.Integer, nullable=
       False, primary_key=True)
58 domain_ip = db.Column(db.String(45))
59 username = db.Column(db.String(45))
60 passwd = db.Column(db.String(45))
61 role = db.Column(db.Boolean)
62 created_at = db.Column(db.DateTime)
63 is_active = db.Column(db.Boolean, default=True)
64
65 def __repr__(self):
66 return self.domain_ip
67
68
69 class VM(db.Model):
70 """
71 Create a Node table
72 """
73 __tablename__ = 'vms'
74
```

```
75 id = db.Column(db.Integer, nullable=False,
       primary_key=True)
76 ip_node = db.Column(db.String(45))
77 id_vm = db.Column(db.Integer)
78 is_active = db.Column(db.Boolean, default=True)
79
80 def __repr__(self):
81 return str(self.id_vm)
82
83
84 class Paired_VM(db.Model):
85 """
86 Create a Paired VM table
87 """
88 __tablename__ = 'paired_vms'
89
90 id_paired_vm = db.Column(db.Integer, nullable=
       False, primary_key=True)
91 id_vm_src = db.Column(db.Integer)
92 id_vm_dst = db.Column(db.Integer)
93 ip_src = db.Column(db.String(45))
94 ip_dst = db.Column(db.String(45))
95 is_active = db.Column(db.Boolean, default=True)
96
97 def __repr__(self):
98 return '<Paired VM: {} {}>'.format(self.id_vm_src
       , self.id_vm_dst)
99
100
101 class Task(db.Model):
102 """
103 Create a Task table
104 """
```

```
105 __tablename__ = 'tasks'
106
107 id_tasks = db.Column(db.Integer, nullable=False,
        primary_key=True)
108 id_vm_src = db.Column(db.Integer)
109 ip_src = db.Column(db.String(45))
110 passwd_src = db.Column(db.String(45))
111 id_vm_dst = db.Column(db.Integer)
112 ip_dst = db.Column(db.String(45))
113 passwd_dst = db.Column(db.String(45))
114 status = db.Column(db.Integer)
115 start_time = db.Column(db.DateTime)
116 end_time = db.Column(db.DateTime)
117 job = db.Column(db.Boolean)
118 archive_name = db.Column(db.String(100))
119 schedule_id = db.Column(db.Integer)
120
121 def __repr__(self):
122 return '<Task: {}>'.format(self.job)
123
124 class Scheduled_Task(db.Model):
125 __tablename__ = 'scheduled_tasks'
126 id_scheduled_task = db.Column(db.Integer,
        nullable=False, primary_key=True)
127 id_vm_src = db.Column(db.Integer)
128 ip_src = db.Column(db.String(45))
129 passwd_src = db.Column(db.String(45))
130 id_vm_dst = db.Column(db.Integer)
131 ip_dst = db.Column(db.String(45))
132 passwd_dst = db.Column(db.String(45))
133 job = db.Column(db.Boolean) # 0: backup, 1:
        restore
134
```

```
135 status_job = db.Column(db.String(45)) # new,
        inserted, deleted
136 hour = db.Column(db.String(10))
137 minute = db.Column(db.String(10))
138
139 def __repr__(self):
140 return '<Task: {}>'.format(self.id_scheduled_task
        )
141
142 class Archive:
143 @staticmethod
144 def get_archives(ip_src, id_vm_src):
145 backup_dir = getenv('BACKUP_DIR')
146 archive_path = join(backup_dir, ip_src + '_' +
        id_vm_src)
147 onlyfiles = [f for f in listdir(archive_path) if
        isfile(join(archive_path, f)) and join(
        archive_path, f).endswith(".vma.lzo")]
148 return onlyfiles
```

Kode Sumber B.3: File Inisialisasi Model (models.py))

Script Proses *Backup*

```
1 #!/bin/bash
2
3 SRC=$1
4 PASS=$2
5 VMID_B=$3
6
7 sshpass -p "${PASS}" ssh -i /home/nahda/.ssh/
    id_rsa -o StrictHostKeyChecking=no root@${SRC}
     "mkdir -p /home/backups/${VMID_B} && vzdump
```

```
    --dumpdir /home/backups/${VMID_B} --compress
    lzo ${VMID_B} && echo $? || echo 1"
sshpass -p "${PASS}" rsync -v --stats --progress
    -auzhe ssh root@${SRC}:/home/backups/${VMID_B}
    / /home/nahda/${SRC}_${VMID_B}
```

Kode Sumber B.4: File Script Backup (backup.sh)

Script Proses *Restore*

```
#!/bin/bash

SRC=$1
DEST=$2
PASS=$3
VMID_B=$4
VMID_R=$5
FILE=$6

sshpass -p "${PASS}" rsync -e"ssh -i /home/nahdec
    /.ssh/id_rsa -o StrictHostKeyChecking=no" -v
    --stats --progress -Patz /home/nahdec/${SRC}_$
    {VMID_B}/ root@${DEST}:/var/lib/vz/dump/
sshpass -p "${PASS}" ssh root@${DEST} "qm list |
    grep -v VMID" > /tmp/vmlist01
awk -F' ' '{print $1}' /tmp/vmlist01 > /tmp/
    vmlist
if grep -q "${VMID_R}" /tmp/vmlist
then
echo "VMID Already exist"
sshpass -p "${PASS}" ssh root@${DEST} "qm
    shutdown ${VMID_R} && qm destroy ${VMID_R} &&
```

```
    qmrestore local:backup/${FILE} ${VMID_R} && qm
     start ${VMID_R}"
17 else
18 sshpass -p "${PASS}" ssh root@${DEST} "qmrestore
    local:backup/${FILE} ${VMID_R} && qm start ${
    VMID_R}"
19 fi
20 echo "Cleanup tmp directory."
21 rm /tmp/vmlist*
```

Kode Sumber B.5: File Script Restore (restore.sh)

Script *Worker*

```
1  import MySQLdb
2  import subprocess
3  import time
4  import schedule
5  import os
6  import csv
7  from datetime import datetime
8  from dotenv import load_dotenv
9
10 dir_path = os.path.dirname(os.path.realpath(
    __file__))
11 load_dotenv(dotenv_path=os.path.join(dir_path,
    "../.env"))
12
13 PATH_SCRIPT_BACKUP = os.path.join(dir_path, "
    backup.sh")
14 PATH_SCRIPT_RESTORE = os.path.join(dir_path, "
    restore.sh")
15 PATH_LOG = os.path.join(dir_path, "../log.csv")
```

```
16
17 HOST = os.getenv("DB_HOST")
18 USERNAME = os.getenv("DB_USERNAME")
19 PASS = os.getenv("DB_PASSWORD")
20 DB = os.getenv("DB_NAME")
21
22 def check_new_task():
23 db = MySQLdb.connect(host=HOST,
24 user=USERNAME,
25 passwd=PASS,
26 db=DB)
27 cursor = db.cursor()
28 sql = "SELECT * FROM tasks WHERE status = 0 ORDER
       BY id_tasks ASC LIMIT 1;"
29 try:
30 cursor.execute(sql)
31 return [dict(zip([column[0] for column in cursor.
      description], row)) for row in cursor.fetchall
      ()]
32 except:
33 print("check_new_task Error! Unable to fetch data
      ")
34 cursor.close()
35 db.close()
36 return
37
38
39 def update_status(id_task, status):
40 db = MySQLdb.connect(host=HOST,
41 user=USERNAME,
42 passwd=PASS,
43 db=DB)
44 cursor = db.cursor()
```

```
45 sql = "UPDATE tasks SET status = " + str(status)
       + " WHERE id_tasks = " + str(id_task) + ";"
46 try:
47 cursor.execute(sql)
48 db.commit()
49 print("Number of rows updated: " + str(cursor.
       rowcount))
50 except:
51 print("update_status Error! Unable to fetch data
       ")
52 cursor.close()
53 db.close()
54 return
55
56
57 def update_time(id_task, time_type):
58 db = MySQLdb.connect(host=HOST,
59 user=USERNAME,
60 passwd=PASS,
61 db=DB)
62 cursor = db.cursor()
63 time = datetime.now()
64
65 if time_type == 0:
66   sql = "UPDATE tasks SET start_time = '" + str(
         time) + "' WHERE id_tasks = " + str(id_task)
          + ";"
67 else:
68   sql = "UPDATE tasks SET end_time = '" + str(
         time) + "' WHERE id_tasks = " + str(id_task)
          + ";"
69   print(sql)
70 try:
```

```
71 cursor.execute(sql)
72 db.commit()
73 print("Number of rows updated: " + str(cursor.
       rowcount))
74 except:
75 print("update_time Error! Unable to fetch data")
76 cursor.close()
77 db.close()
78 return
79
80
81 def backup(ip, password, id_vm, schedule_id=None)
       :
82 status = None
83 start_time = time.time()
84 try:
85 subprocess.check_output([PATH_SCRIPT_BACKUP, ip,
       password, id_vm])
86 if schedule_id:
87 None
88 status = 1
89 except subprocess.CalledProcessError as e:
90 print(e.output)
91 status = -1
92 end_time = time.time()
93 export_log(ip, None, password, id_vm, None,
       start_time, end_time)
94 return status
95
96
97 def insert_new_schedule_to_task(schedule_id):
98 db = MySQLdb.connect(host=HOST,
99 user=USERNAME,
```

```
100 passwd=PASS,
101 db=DB)
102 cursor = db.cursor()
103 sql = "SELECT * FROM scheduled_tasks WHERE
        id_scheduled_task = " + str(schedule_id) + ";"
104 cursor.execute(sql)
105 row = cursor.fetchone()
106
107 id_schedule = row[0]
108 id_vm_src = row[1]
109 ip_src = row[2]
110 pass_src = row[3]
111 id_vm_dst = row[4]
112 ip_dst = row[5]
113 pass_dst = row[6]
114
115 cursor_ins = db.cursor()
116 sql = "INSERT INTO tasks (id_vm_src,ip_src,
        passwd_src,id_vm_dst,ip_dst,passwd_dst,job,
        schedule_id,status) VALUES ('" + str(id_vm_src
        ) + "','" + \
117 str(ip_src) + "','" + str(pass_src) + "'," + str(
        id_vm_dst) + ",'" + str(ip_dst) + "','" + \
118 str(pass_dst) + "',0," + str(id_schedule) +",0);"
119 cursor_ins.execute(sql)
120 cursor_ins.close()
121 db.commit()
122
123 def restore(ip_src, ip_dst, password, id_vm_src,
        id_vm_dst, archive_name):
124 status = None
125 start_time = time.time()
126 try:
```

```
127 subprocess.check_output([PATH_SCRIPT_RESTORE,
        ip_src, ip_dst, password, id_vm_src, id_vm_dst
        , archive_name])
128 status = 1
129 except subprocess.CalledProcessError as e:
130 print(e.output)
131 status = -1
132 end_time = time.time()
133 export_log(ip_src, ip_dst, password, id_vm_src,
        id_vm_dst, start_time, end_time)
134 return status
135
136 def is_empty(my_function):
137 return len(my_function) == 0
138
139 def export_log(ip_src, ip_dst, password,
        id_vm_src, id_vm_dst, start_time, end_time):
140 res_title = ['ts_start', 'ts_end', 'ip_src', '
        ip_dst', 'password', 'id_vm_src', 'id_vm_dst',
         'run_time']
141 res, res_data = [], []
142 res_data.append(time.strftime("%H:%M:%S", time.
        gmtime(start_time)))
143 res_data.append(time.strftime("%H:%M:%S", time.
        gmtime(end_time)))
144 res_data.append(ip_src)
145 res_data.append(ip_dst)
146 res_data.append(password)
147 res_data.append(id_vm_src)
148 res_data.append(id_vm_dst)
149 elapsed_time = end_time - start_time
150 res_data.append(time.strftime("%H:%M:%S", time.
        gmtime(elapsed_time)))
```

```
151 res.append(res_data)
152 if not os.path.exists(PATH_LOG):
153 with open(PATH_LOG, 'a') as output:
154 writer = csv.writer(output, lineterminator='\n')
155 writer.writerow(res_title)
156 with open(PATH_LOG, 'a') as output:
157 writer = csv.writer(output, lineterminator='\n')
158 writer.writerows(res)
159 print('logged')
160
161 def add_all_schedule():
162 db = MySQLdb.connect(host=HOST,
163 user=USERNAME,
164 passwd=PASS,
165 db=DB)
166 cursor = db.cursor()
167 sql = "SELECT * FROM scheduled_tasks WHERE
        status_job = 'inserted';"
168 try:
169 cursor.execute(sql)
170 results = cursor.fetchall()
171 for row in results:
172 id_schedule = row[0]
173 id_vm_src = row[1]
174 ip_src = row[2]
175 pass_src = row[3]
176 hour = row[9]
177 minute = row[10]
178 schedule.every().day.at(hour+':'+minute).do(
        insert_new_schedule_to_task, schedule_id=
        id_schedule).tag(str(id_schedule))
179
180 print("Add schedule " + str(id_schedule))
```

```
181 except:
182 print("add_all_schedule Error! Unable to fetch
        data")
183 cursor.close()
184 db.close()
185 return
186 
187 def add_new_schedule():
188 db = MySQLdb.connect(host=HOST,
189 user=USERNAME,
190 passwd=PASS,
191 db=DB)
192 cursor = db.cursor()
193 sql = "SELECT * FROM scheduled_tasks WHERE
        status_job = 'new';"
194 try:
195 cursor.execute(sql)
196 results = cursor.fetchall()
197 for row in results:
198 id_schedule = row[0]
199 id_vm_src = row[1]
200 ip_src = row[2]
201 pass_src = row[3]
202 hour = row[9]
203 minute = row[10]
204 print(hour)
205 print(minute)
206 schedule.every().day.at(hour+':'+minute).do(
        insert_new_schedule_to_task, schedule_id=
        id_schedule).tag(str(id_schedule))
207 update_status_job(id_schedule)
208 
209 print("Add new schedule " + str(id_schedule))
```

```
except e:
print(e)
print("add new Error! Unable to fetch data")
cursor.close()
db.close()
return

def schedule_to_task():
db = MySQLdb.connect(host=HOST,
user=USERNAME,
passwd=PASS,
db=DB)
cursor = db.cursor()
sql = "SELECT * FROM scheduled_tasks WHERE
    status_job = 'inserted';"
try:
cursor.execute(sql)
results = cursor.fetchall()
for row in results:
id_schedule = row[0]
id_vm_src = row[1]
ip_src = row[2]
pass_src = row[3]
id_vm_dst = row[4]
ip_dst = row[5]
pass_dst = row[6]

schedule.clear(str(id_schedule))
cursor_ins = db.cursor()
sql = "INSERT INTO tasks (id_vm_src,ip_src,
    passwd_src,id_vm_dst,ip_dst,passwd_dst,job,
    schedule_id) VALUES (" + str(id_vm_src) + ","
    + str(ip_src) + "," + str(pass_src) + "," +
```

```
    str(id_vm_dst) + "," + str(ip_dst) + "," + str
    (pass_dst) + ",0," + str(id_schedule) +");"
cursor_ins.execute(sql)
cursor_ins.close()
db.commit()

print("Inserted schedule to tasks " + str(
    id_schedule))
except:
print("schedule to tasks Error! Unable to fetch
    data")
cursor.close()
db.close()
return

def update_status_job(id_schedule):
db = MySQLdb.connect(host=HOST,
user=USERNAME,
passwd=PASS,
db=DB)
cursor = db.cursor()
sql = "UPDATE scheduled_tasks SET status_job = '
    inserted' WHERE id_scheduled_task = " + str(
    id_schedule) + ";"
try:
cursor.execute(sql)
db.commit()
print("Number of rows updated: " + str(cursor.
    rowcount))
except:
print("update Error! Unable to fetch data")
cursor.close()
db.close()
```

```
265 return
266
267 def delete_schedule():
268 db = MySQLdb.connect(host=HOST,
269 user=USERNAME,
270 passwd=PASS,
271 db=DB)
272 cursor = db.cursor()
273 sql = "SELECT * FROM scheduled_tasks WHERE
        status_job = 'deleted';"
274 try:
275 cursor.execute(sql)
276 results = cursor.fetchall()
277 for row in results:
278 id_schedule = row[0]
279 schedule.clear(str(id_schedule))
280 cursor_del = db.cursor()
281 sql = "DELETE FROM scheduled_tasks WHERE
        id_scheduled_task = " + str(id_schedule) + ";"
282 cursor_del.execute(sql)
283 cursor_del.close()
284 db.commit()
285
286 print("Delete schedule " + str(id_schedule))
287 except:
288 print("Error delete! Unable to fetch data")
289 cursor.close()
290 db.close()
291 return
292
293 if __name__ == "__main__":
294 add_all_schedule()
295 try:
```

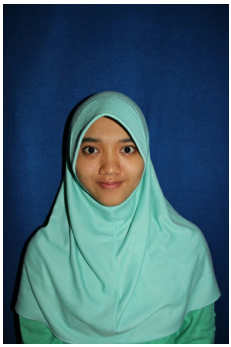
```
while True:
print("Current Jobs:")
print(schedule.jobs)
# print("")
schedule.run_pending()

add_new_schedule()
delete_schedule()

new_task = check_new_task()
if is_empty(new_task):
print(str(datetime.now()) + " no new task.")
else:
id_task = new_task[0]['id_tasks']
status = 2
# update status jadi pending dan starttime
update_status(id_task, status)
update_time(id_task, 0)
# check backup
if int(new_task[0]['job']) == 0:
# get variable
ip = str(new_task[0]['ip_src'])
password = str(new_task[0]['passwd_src'])
id_vm = str(new_task[0]['id_vm_src'])
# get status backup

print("Backing up " + ip + " VM " + id_vm +
    "...")

status = backup(ip, password, id_vm)
# check restore
else:
# get variable
```

```
328 ip_src = str(new_task[0]['ip_src'])
329 ip_dst = str(new_task[0]['ip_dst'])
330 password = str(new_task[0]['passwd_dst'])
331 id_vm_src = str(new_task[0]['id_vm_src'])
332 id_vm_dst = str(new_task[0]['id_vm_dst'])
333 archive_name = str(new_task[0]['archive_name'])
334 # get status restore
335
336 print("Restoring from " + ip_src + " VM " +
        id_vm_src + " to " + ip_dst + " VM " +
        id_vm_dst + "...")
337
338 status = restore(ip_src, ip_dst, password,
        id_vm_src, id_vm_dst, archive_name)
339 # update status and endtime
340 update_status(id_task, status)
341 update_time(id_task, 1)
342 time.sleep(10)
343 except KeyboardInterrupt:
344 pass
```

Kode Sumber B.6: File Script Worker (worker.py)

(Halaman ini sengaja dikosongkan)

BIODATA PENULIS



Nahda Fauziyah Zahra, akrab dipanggil Nahda lahir pada tanggal 28 Maret 1998 di Kota Jakarta Timur, DKI Jakarta. Penulis merupakan seorang mahasiswi yang sedang menempuh studi di Departemen Informatika Institut Teknologi Sepuluh Nopember. Selama menempuh pendidikan di kampus, penulis juga aktif dalam kepanitiaan dan organisasi kemahasiswaan, antara lain sebagai Sekretaris dalam kepanitiaan Schematics ITS 2016 dan Schematics ITS 2017, menjadi Staf Departemen Dalam Negeri (Dagri) Himpunan Mahasiswa Teknik Computer-Informatika dan staf Departemen External Affair BEM FTIf ITS. Selain itu penulis pernah menjadi asisten dosen pada mata kuliah Sistem Operasi, Jaringan Komputer, dan Sistem Terdistribusi. Penulis dapat dihubungi melalui email di **znahda@gmail.com**.