PERANCANGAN DAN IMPLEMENTASI OBJECT STORAGE BERBASIS PROTOKOL S3 PADA PENGEMBANGAN PRODUK COOFIS VERSE DI PT ARM SOLUSI

Design and Implementation of S3 Protocol-Based Object Storage on Coofis Verse Product Development at PT ARM Solusi

Nur Aisyah[1], Muhammad Iqbal[2], Bramandityo Prabowo3[3]

[1,2]Program Studi D3 Teknologi Telekomunikasi
[3]PT Andal Rancang Multi Solusi
[1]aisyhc@student.telkomuniversity.ac.id, [2]miqbal@telkomuniversity.ac.id, [3]bram3@armsolusi.com

Abstrak

Seiring dengan perkembangan teknologi informasi, kebutuhan penyimpanan data semakin meningkat. *Cloud computing* dengan model *scalable* dan *pay-as-you-go* memberikan manfaat signifikan untuk *Object Storage*. PT Andal Rancang Multi Solusi (PT ARM Solusi), yang mengembangkan teknologi seperti *big data*, *data analytics*, dan aplikasi administrasi, menggunakan *object storage* untuk produk Coofis Verse. Tujuan dari proyek ini adalah merancang dan mengimplementasikan *object storage* dengan *Simple Storage Service (S3)* untuk mendukung manajemen file yang lebih efisien. Proyek ini menggunakan platform *MinIO* yang mendukung ukuran objek hingga 50 TiB dan bagian hingga 5 GiB, serta *Ansible* untuk otomatisasi deployment dan operasi. Penulis juga menyediakan aplikasi alternatif yang terintegrasi dengan *MinIO* menggunakan *reimburse* untuk menyimpan file dalam bucket *MinIO*. Hasilnya adalah penyediaan *object storage* berbasis *S3* yang terintegrasi dengan Coofis Verse dan aplikasi alternatif *reimburse*, dengan pengaturan akses pengguna yang fleksibel dalam *MinIO*.

Kata Kunci: Coofis Verse, *Object Storage*, *Microservice*, *Simple Storage Service (S3)*

Abstract

Along with the development of information technology, data storage needs are increasing. Cloud computing with a scalable and pay-as-you-go model provides significant benefits for Object Storage. PT Andal Rancang Multi Solusi (PT ARM Solusi), which develops technology such as big data, data analytics and administration applications, uses object storage for Coofis Verse products. The goal of this project is to design and implement object storage with Simple Storage Service (S3) to support more efficient file management. This project uses the MinIO platform which supports object sizes up to 50 TiB and parts up to 5 GiB, as well as Ansible for deployment and operations automation. The author also provides an alternative application that is integrated with MinIO using reimbursement to store files in the MinIO bucket. The result is the provision of S3-based object storage integrated with Coofis Verse and alternative reimbursement applications, with flexible user access settings in MinIO.

Keywords: Coofis Verse, Object Storage, Microservice, Simple Storage Service (S3)

I. PENDAHULUAN

A. Latar Belakang

Kemajuan teknologi informasi dan komunikasi saat ini telah memengaruhi hampir semua aspek kehidupan, termasuk administrasi dan pengelolaan surat-menyurat di sektor publik. Sistem *tata persuratan* berbasis web adalah salah satu inovasi yang membantu mengelola surat-menyurat di perusahaan atau institusi, mencakup pembuatan, pengelolaan, penyimpanan, dan pengarsipan surat[1]. Di era digital, penggunaan *object storage* menjadi sangat penting karena kemampuannya untuk menyimpan dan mengelola data dalam skala besar dengan metadata yang komprehensif dan identifikasi unik. *Object storage* memungkinkan akses data secara efisien dari mana saja di internet, yang sangat bermanfaat bagi perusahaan seperti PT Andal Rancang Multi Solusi (PT ARM Solusi).

PT ARM Solusi adalah perusahaan *IT Consultant* yang mengembangkan teknologi seperti *big data, data analytics*, *kolaborasi*, dan aplikasi administrasi. Salah satu produknya adalah Coofis NDE (Nota Dinas Elektronik), aplikasi mobile yang menyediakan fitur pengelolaan surat secara elektronik termasuk penerimaan, persetujuan, disposisi, penomoran, pengarsipan, dan tanda tangan digital. Saat ini, Coofis NDE menggunakan layanan *monolitik* yang membatasi skalabilitas dan fleksibilitas[2]. Sistem

monolitik mengharuskan peningkatan kapasitas dilakukan secara keseluruhan dan membuat pengembangan serta pemeliharaan sulit karena ketergantungan antar komponen.

Untuk mengatasi keterbatasan ini, PT ARM Solusi berencana untuk beralih ke arsitektur *microservices*. Metode ini membagi aplikasi menjadi layanan-layanan kecil yang independen, masing-masing dengan fungsi spesifik dan antarmuka *API* (Application Programming Interface). Setiap layanan dalam arsitektur *microservices* dapat beroperasi secara mandiri, meningkatkan fleksibilitas, mendukung *DevOps*, dan memungkinkan penambahan fitur tanpa mengubah fungsi utama aplikasi. Dengan peralihan ini, Coofis NDE akan dikembangkan menjadi Coofis Verse, meningkatkan efisiensi dan skalabilitas.

B. Rumusan Masalah

Adapun rumusan masalah dari Proyek Akhir ini, sebagai berikut.

1. Bagaimana merancang arsitektur *object storage* berbasis protokol S3 yang sesuai dengan kebutuhan pengembangan produk Coofis Verse di PT ARM Solusi?
2. Apa saja tantangan yang mungkin dihadapi dalam mengimplementasikan *object storage* berbasis protokol S3 pada produk Coofis Verse, dan bagaimana cara mengatasinya?
3. Bagaimana evaluasi kinerja *object storage* berbasis protokol S3 setelah diimplementasikan pada produk Coofis Verse, dan sejauh mana peningkatan efisiensi dan kinerja yang dapat dicapai dengan solusi ini?

C. Tujuan

Adapun tujuan dari Proyek Akhir ini, sebagai berikut:

1. Merancang arsitektur dan infrastruktur penyimpanan *object storage* yang memanfaatkan protokol S3 menggunakan arsitektur *Microservice.*
2. Mengintegrasikan solusi penyimpanan objek menggunakan protokol S3 untuk mendukung manajemen *file* yang lebih efisien dan terstruktur untuk produk Coofis Verse.
3. Menerapkan lapisan keamanan yang kokoh untuk melindungi data yang disimpan menggunakan standar keamanan protokol S3.

D. Manfaat

Adapun maanfaat dari Proyek Akhir ini, sebagai berikut:

1. Mengelola dan mengakses data dengan lebih baik pada produk Coofis Verse menggunakan arsitektur *Microservice.*
2. Meningkatkan keandalan sistem dengan memanfaatkan protokol S3 yang telah teruji dan banyak digunakan.
3. Mengembangkan fitur penyimpanan objek yang lebih efisien untuk produk Coofis Verse.

II. KAJIAN TEORI

A. Microservice

Microservice adalah desain infrastruktur yang membagi aplikasi menjadi layanan-layanan kecil yang masing-masing memiliki fungsi spesifik, namun tetap saling terhubung. Berbeda dengan aplikasi *monolitik* yang biasanya terdiri dari satu unit besar, arsitektur *microservice* menyediakan pendekatan yang lebih fleksibel dan skalabel dalam pengembangan perangkat lunak. Sistem informasi dalam arsitektur ini dirancang untuk terdistribusi dan memberikan layanan dengan fokus dan spesifikasi yang lebih jelas [3].

Dengan *microservice*, setiap layanan memiliki cakupan yang lebih kecil dibandingkan dengan aplikasi *monolitik*, yang meningkatkan produktivitas dalam pengembangan dan pemeliharaan aplikasi. Ketika suatu masalah terjadi dalam salah satu layanan, hanya layanan yang bersangkutan yang perlu diperbaiki tanpa mempengaruhi layanan lain yang ada.

B. Object Storage

Ada beberapa jenis penyimpanan data pada *cloud storage* yaitu *block storage, file storage*, dan *object storage*. *Object storage* adalah model penyimpanan data yang menyimpan data sebagai objek, lengkap dengan metadata yang terkait. *Object storage* sering digunakan dalam skenario penyimpanan *cloud*, arsip data, penyimpanan besar-besaran, dan penyimpanan data yang perlu skalabilitas dan ketersediaan yang tinggi. Saat ini, penggunaan *object storage* sebagai media penyimpanan di *cloud* menjadi sangat penting untuk mengurangi biaya *server* dalam pengembangan atau implementasi aplikasi [4].

Beberapa contoh *object storage* yang bersifat *open-source* yang kompatible dengan antarmuka S3 adalah MinIO yang dapat diimplementasikan di berbagai lingkungan dan digunakan untuk menyimpan data secara *scalable* dengan peningkatan kapasitas atau kinerja sistem.

C. Amazon S3

Amazon *Simple Storage Service* (Amazon S3) adalah salah satu layanan penyimpanan protokol yang disediakan oleh AWS. Di Amazon S3, pengguna diberikan sebuah wadah bernama *bucket* untuk menyimpan objek yang mereka inginkan. Layanan Amazon S3 menggunakan arsitektur *Representational State Transfer* (REST) melalui protokol HTTP [5].

Protokol ini memungkinkan pengguna untuk mengakses, mengelola, dan menyimpan objek di S3

menggunakan *bucket*, *key*, dan operasi. Banyak aplikasi pihak ketiga, seperti *Cyberduck*, *Dragondisk*, dan S3 *Browser*, juga menggunakan protokol S3 untuk mengelola *file* dalam *bucket* S3. Protokol S3 memungkinkan pengguna untuk mengakses layanan penyimpanan objek S3 dari aplikasi atau skrip milik pribadi atau khusus, dan menyediakan beragam fungsi yang dapat digunakan untuk mengelola *file* di dalam *bucket*

D. MinIO

MinIO adalah alat *open source* populer yang kompatibel dengan protokol S3 dan memiliki kinerja tinggi dalam penyimpanan objek. Ditulis dalam bahasa GO dan assembly, MinIO memiliki fitur seperti kode penghapusan inline, perlindungan bitrot, kompresi, enkripsi, dan hanya mendukung sistem file xfs dan ext4 yang dipasang sebagai OSD [6].

MinIO tersedia sebagai perangkat lunak *open-source* yang dapat diunduh dan digunakan secara gratis, dan sering digunakan untuk menyediakan performa tinggi dan skalabilitas horizontal yang tanpa batas dengan mendukung klaster penyimpanan objek, serta MinIO mendukung API S3 sehingga kompatibel dengan aplikasi dan alat yang sudah ada yang dibangun untuk menggunakan layanan AWS S3. Terdapat beberapa keuntungan dalam penggunaan MinIO seperti:

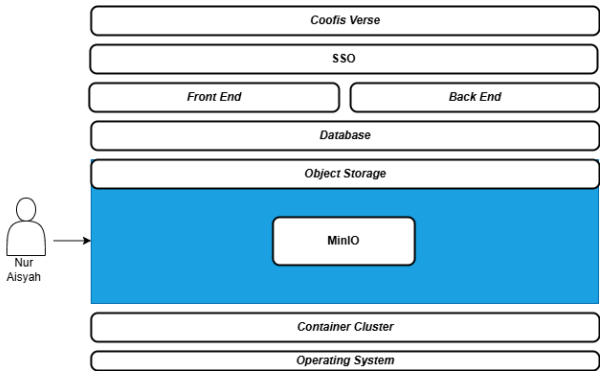
1. Dapat di-*deploy* di infrastruktur yang ada atau *cloud* publik dengan biaya operasional yang lebih rendah.
2. Kompabilitas dengan API S3 memudahkan integrasi dengan aplikasi dan alat yang menggunakan AWS S3.
3. Memungkinkan penambahan kapasitas penyimpanan tanpa batasan horizontal, sehingga dapat memenuhi kebutuhan pertumbuhan data yang cepat.

III. PERANCANGAN OBJECT STORAGE

A. Deskripsi Proyek Akhir

Dalam proyek ini, direncanakan dan diimplementasikan *Object Storage* berbasis *S3* untuk produk *Coofis Verse* menggunakan platform *MinIO*. *MinIO* dipilih karena kelebihannya yang sesuai dengan kebutuhan *Coofis Verse* di PT ARM Solusi. *Coofis Verse* adalah layanan pengelolaan tata persuratan yang dilengkapi dengan fitur seperti dashboard, surat internal, dan surat disposisi. Pada Gambar 3.1, merupakan blok diagram lebih rinci yang dimana penulis berfokus dalam penyediaan *server* layanan *object storage* dengan menggunakan MinIO.

Dalam hal ini, penulis menyiapkan *server* layanan MinIO dengan membuat *bucket* yang nantinya akan digunakan untuk penyimpanan *object* yang diterima.



Gambar 3. 1 Blok Diagram Sistem

B. Analisis Kebutuhan Perangkat

Dalam pengerjaan proyek akhir ini, perangkat yang digunakan terdiri dari perangkat keras (*hardware*) dan perangkat lunak (*software*) dengan spesifikasi sebagai berikut:

1. Spesifikasi Perangkat Keras (*Hardware*)

Perangkat keras yang digunakan dalam pengerjaan proyek akhir ini adalah laptop dengan spesifikasi sebagai berikut:

a. Device name: MSI
b. Processor: 11th Gen Intel(R) Core (TM) i5-11400H @ 2.70GHz 2.69 GHz
c. Installed RAM: 8.00 GB (7.71 GB usable)
d. System type: 64-bit operating system, x64-based processor

2. Spesifikasi Perangkat Lunak (*Software*)

Perangkat lunak yang digunakan dalam pengerjaan protek akhir ini, memiliki spesifikasi sebagai berikut:

Tabel 3. 1 Spesifikasi Perangkat Lunak (Software)

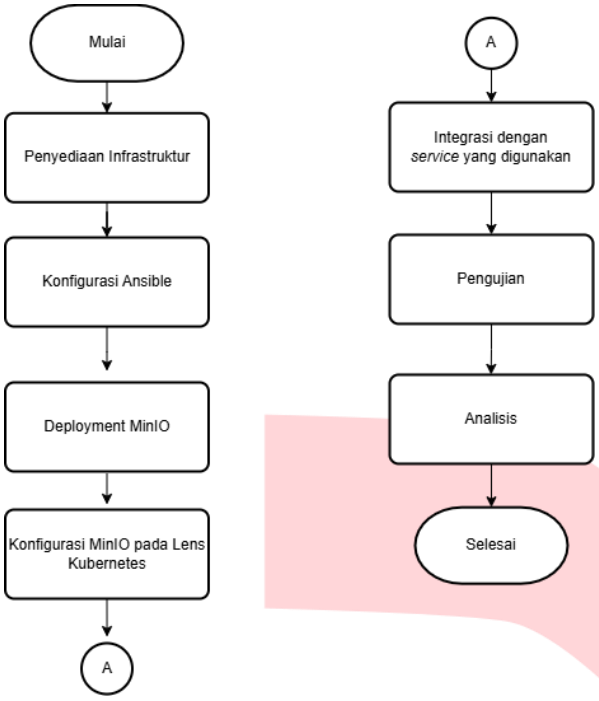
SPESIFIKASI	VERSI
Termius	8.12.7
Lens Kubernetes IDE	2024.4.230844-latest
Apache JMeter	5.6.3

C. Perancangan Sistem

Pada proyek akhir ini dengan judul perancangan dan implementasi *object storage* berbasis S3 menggunakan platform MinIO akan dilakukan dengan beberapa tahapan.

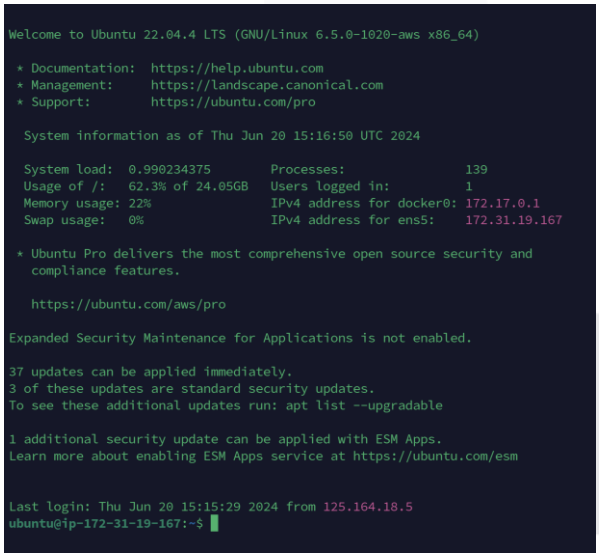
Di bawah ini merupakan penjelasan *flowchart* perancangan sistem *object storage* berbasis S3 yang akan dilakukan dengan dua proses yaitu bagian *deployment* juga *operations*.

1) Flowchart Deployment



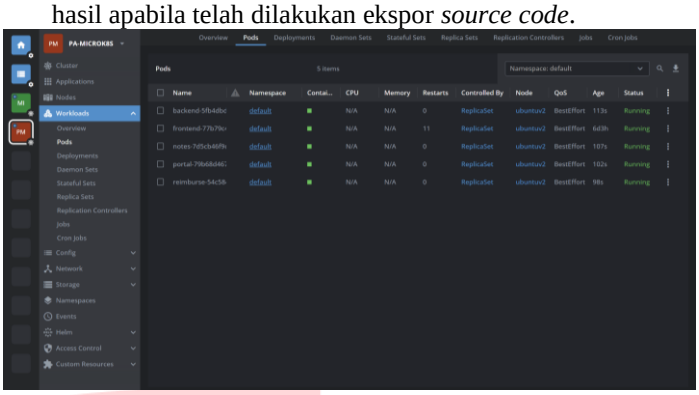
Gambar 3. 2 Deployment Storage

1. Tahap pertama yaitu melakukan penyediaan infrastruktur berupa memastikan bahwa server *virtual machine* sudah berjalan, dan melakukan peng*install*-an Ansible.



Gambar 3. 3 Penyediaan Server

2. Tahap kedua, yaitu melakukan konfigurasi pada *file inventory* yang berisi *IP Address* target *tools* yang di*install* dan *source code* file .yaml yang berisikan perintah untuk melakukan instalasi pada *tools* MinIO.
3. Tahap ketiga, melakukan *deployment* pada MinIO dengan cara mempersiapkan *source code* seperti *file deployment*, *service*, dan *ingress* untuk di ekspor dan mempersiapkan IP Address yang akan dikonfigurasi pada Lens Kubernetes.
4. Tahap keempat, melakukan ekspor *source code* yang telah disiapkan sebelumnya. Berikut adalah hasil apabila telah dilakukan ekspor *source code*.

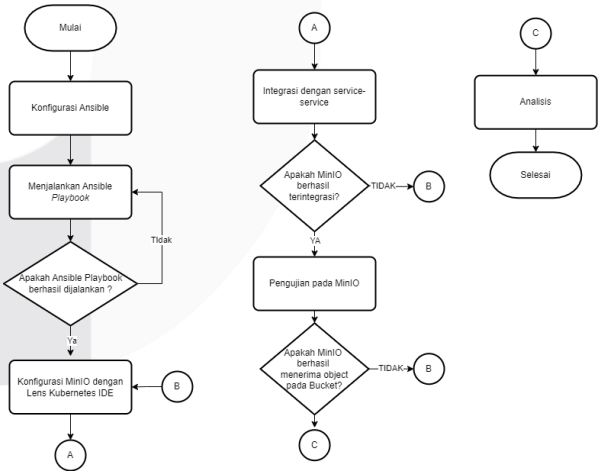


Gambar 3. 4 Eskpor Source Code

5. Tahap kelima, melakukan pengintegrasian terhadap *service* yang akan dihubungkan dengan MinIO melalui lens Kubernetes.

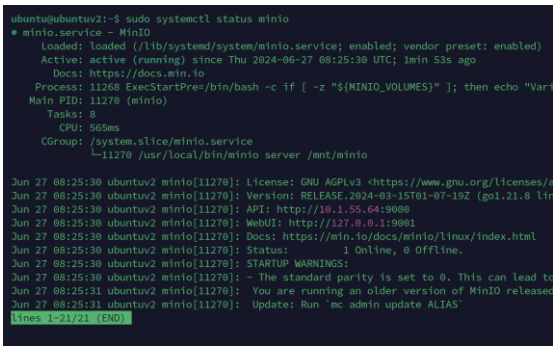
```
DEFAULT_FILE_STORAGE = "xxxxx"

MINIO_STORAGE_ENDPOINT= os.environ.get("xxxxx")

MINIO_STORAGE_SECRET_KEY=
os.environ.get("xxxxx")

MINIO_STORAGE_ACCESS_KEY=
os.environ.get("xxxxxx")

MINIO_STORAGE_USE_HTTPS= os.environ.get("xxxxx)

MINIO_STORAGE_MEDIA_BUCKET_NAME=
os.environ.get("xxxxx")
```

2) Flowchart Operations



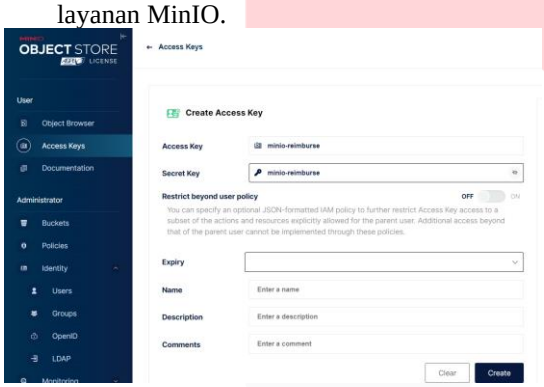
Gambar 3. 5 Operations Storage

1. Tahap pertama yaitu melakukan *running* terhadap ansible yang telah dikonfigurasi pada tahap *deployment* sebelumnya. Tahap *running* ansible ini dilakukan dengan menggunakan *command* "ansible-*playbook* -I <file inventory> <nama .yaml yang akan dijalankan>. Selanjutnya, hal yang dilakukan selanjutnya adalah dengan memastikan bahwa *tools* MinIO yang kita *install* telah berjalan.



Gambar 3. 6 Status tools MinIO

2. Tahap kedua, melakukan *login* terlebih dahulu ke MinIO Console. Setelah *login*, selanjutnya melakukan konfigurasi MinIO dengan membuat *Access Key & Secret Key* serta *Bucket* pada MinIO sesuai dengan konfigurasi yang telah di buat pada tahap *Deployment* yang nantinya akan digunakan untuk mengakses layanan MinIO.



Gambar 3. 7 Pembuatan Access Key

Gambar 3. 8 Pembuatan Bucket MinIO

3. Tahap ketiga, integrasi *service* pada MinIO dengan melakukan eskpor *source code deployment* MinIO yang telah dibuat di tahap *deployment* kedalam Lens Kubernetes IDE.

1) *Deployment*

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: reimburse
  labels:
    app: reimburse
```

2) *Service*

```
apiVersion: v1
kind: Service
metadata:
  name: reimburse
```

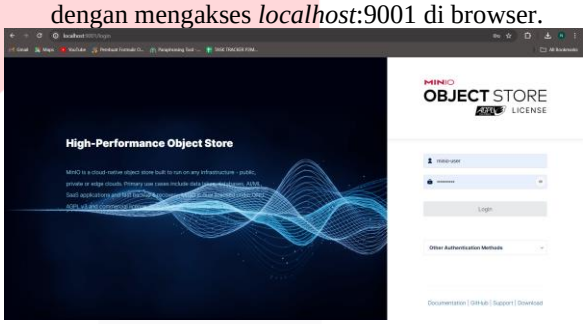
3) *Ingress*

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: reimburse
```

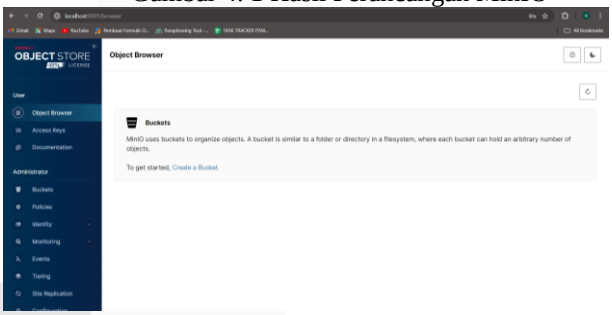
IV. HASIL DAN PENGUJIAN

A. Hasil Perancangan

1. Pada perancangan MinIO, hal pertama yang dilakukan adalah mengakses *localhost* dengan port yang telah dikonfigurasi sebelumnya untuk memastikan bahwa MinIO berhasil diakses yaitu dengan mengakses *localhost*:9001 di browser.



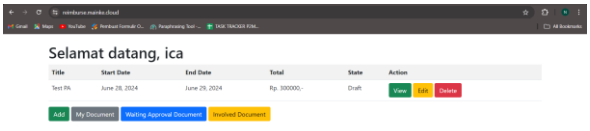
Gambar 4. 1 Hasil Perancangan MinIO



Gambar 4. 2 Dashboard MinIO

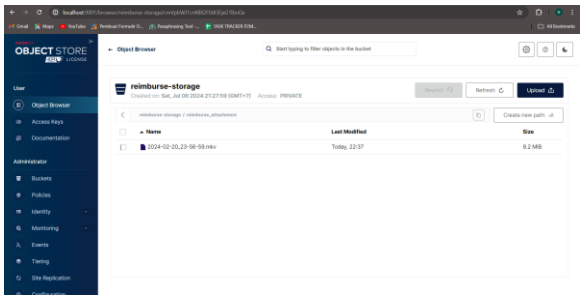
B. Pengujian MinIO

1. Pada pengujian MinIO dilakukan percobaan pengiriman *file* dari reimburse ke MinIO dengan cara mengirimkan sebuah *file* yang nantinya akan masuk ke *bucket* pada MinIO jika pengujian tersebut berhasil.



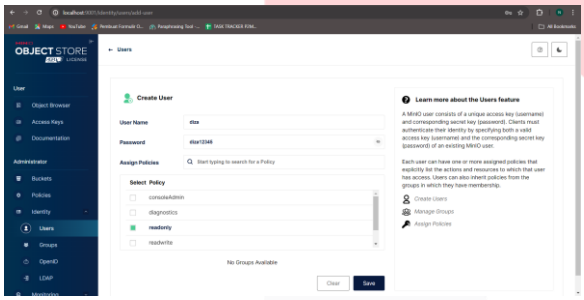
Gambar 4. 3 Pengujian MinIO

2. Kemudian jika *file* tersebut berhasil dikirimkan ke MinIO maka *file* tersebut akan masuk kedalam bucket seperti gambar 4.4



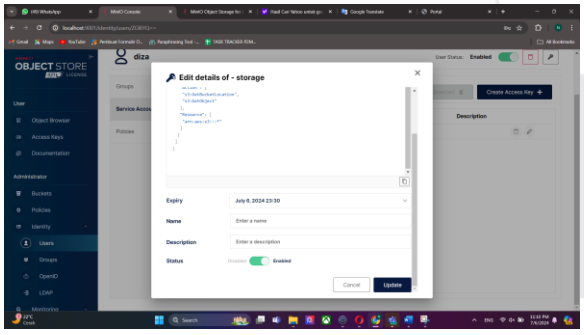
Gambar 4. 4 Bucket MinIO

3. Selanjutnya membuat *user* yang dimana nantinya *user* ini memiliki kebijakan yang dapat diatur oleh admin. Pada *case* ini admin mengatur *user* ini hanya untuk membaca (*readonly*).



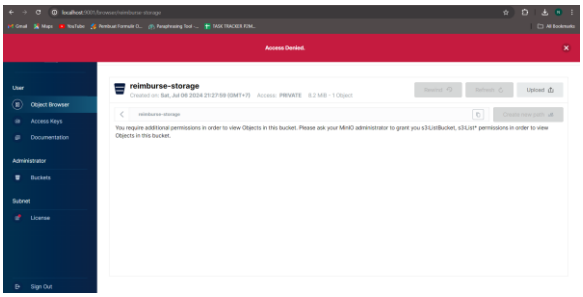
Gambar 4. 5 Membuat User MinIO

4. Kemudian admin juga bisa melakukan kebijakan seperti berapa lama *user* tersebut bisa mengakses console MinIO atau dapat dikatakan admin mengatur waktu *expired user* tersebut untuk bisa mengakses MinIO.



Gambar 4. 6 Konfigurasi User

5. Pada gambar dibawah ini adalah tampilan Ketika *user* tersebut sudah melewati batas waktu yang di atur oleh admin untuk mengakses MinIO.



Gambar 4. 7 Hasil Konfigurasi User

C. Pengujian Kinerja Sistem MinIO

Tabel 4. 1 Pengujian Kinerja Sistem MinIO

No	Deskripsi Pengujian	Hasil
1	Mengirimkan *file* dengan ukuran 50 MB	Dalam mengirimkan 1 *file* berukuran 50 MB dibutuhkan waktu kurang lebih 10 Detik untuk *file* masuk ke *bucket* dalam MinIO
2	Mengirimkan *file* dengan ukuran 40 MB	Dalam mengirimkan 1 *file* berukuran 40 MB dibutuhkan waktu kurang lebih 07 Detik untuk *file* masuk ke *bucket* dalam MinIO
3	Mengirimkan *file* dengan ukuran 30 MB	Dalam mengirimkan 1 *file* berukuran 30 MB dibutuhkan waktu kurang lebih 05 Detik untuk *file* masuk ke *bucket* dalam MinIO
4	Mengirimkan *file* dengan ukuran 20 MB	Dalam mengirimkan 1 *file* berukuran 20 MB dibutuhkan waktu kurang lebih 06 Detik untuk *file* masuk ke *bucket* dalam MinIO
5	Mengirimkan *file* dengan ukuran 10 MB	Dalam mengirimkan 1 *file* berukuran 10 MB dibutuhkan waktu kurang lebih 03 Detik untuk *file* masuk ke *bucket* dalam MinIO
6	Mengirimkan 3 *file* secara bersamaan dengan ukuran rata-rata 10 MB	Dalam mengirimkan 3 *file* dengan ukuran rata-rata 10 MB dibutuhkan waktu kurang lebih 05 Detik untuk *file* masuk ke *bucket* dalam MinIO

Berdasarkan hasil pengujian yang disajikan pada tabel diatas, dapat disimpulkan bahwa rata-rata waktu yang dibutuhkan dalam proses pengiriman *file* ke *bucket* MinIO cukup cepat meskipun pengiriman objek tersebut menggunakan ukuran *file* yang berbeda-beda.

Pada percobaan pertama, untuk *file* dengan ukuran 50 MB waktu yang dibutuhkan adalah sekitar 10 detik. Sementara untuk *file* yang berukuran lebih kecil, yaitu menggunakan ukuran 10 MB waktu pengiriman yang dibutuhkan adalah sekitar 3 detik. Serta, pengiriman dengan skema menggunakan 3 *file* secara bersamaan dengan ukuran rata-rata *file* yang digunakan adalah 10 MB memerlukan waktu sekitar 5 detik.

Sehingga, dapat disimpulkan bahwa MinIO dapat menangani beban pengiriman objek dengan rentang ukuran *file* 10 MB – 50 MB secara cepat dengan rata-rata waktu nya adalah 3 – 10 detik. Namun, hal ini perlu dilakukan pengujian lebih lanjut dengan menggunakan objek yang berukuran lebih besar untuk mengetahui kestabilan kinerja MinIO. Pengujian ini

juga perlu dilakukan untuk memastikan bahwa MinIO dapat digunakan untuk kebutuhan penyimpanan dan pengiriman *file* dalam skala yang lebih besar.

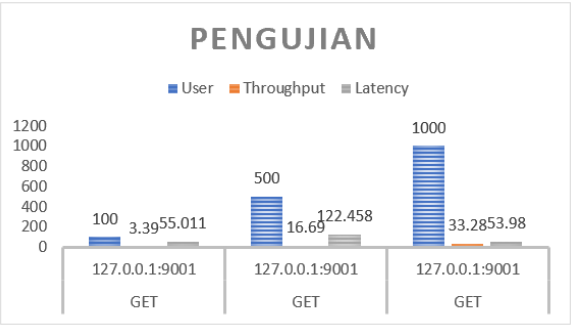
D. Pengujian Fungsionalitas

Tabel 4. 2 Pengujian Fungsionalitas

No.	Fitur yang Diuji	Deskripsi Pengujian	Hasil yang diharapkan
1	Penyimpanan Objek	Menguji kemampuan menyimpan *file* ke dalam *object storage* menggunakan protokol S3.	*File* berhasil diunggah dan disimpan dalam *object storage*, serta bisa dilihat melalui antarmuka pengguna.
2	Pengambilan Objek	Menguji kemampuan mengambil dan mengunduh *file* dari *object storage* menggunakan protokol S3.	*File* berhasil diunduh dari *object storage* ke komputer pengguna.
3	Penghapusan Objek	Menguji kemampuan menghapus *file* dari *object storage* menggunakan protokol S3.	*File* berhasil dihapus dari *object storage* dan tidak lagi muncul di antarmuka pengguna.
4	Pembatasan Akses Objek	Menguji kemampuan membatasi akses ke *file* tertentu di dalam *object storage* menggunakan protokol S3.	*File* hanya bisa diakses oleh pengguna yang diizinkan sesuai dengan pengaturan yang telah dibuat.
5	Pembuatan *Bucket*	Menguji kemampuan membuat *bucket* baru di dalam *object storage* menggunakan protokol S3.	*Bucket* baru berhasil dibuat dan muncul di daftar *bucket* yang tersedia.
6	Integrasi dengan Sistem Autentikasi	Menguji integrasi sistem *object storage* dengan sistem autentikasi pengguna pada Coofis Verse.	Pengguna tidak dapat mengakses *file* tanpa *login*, tetapi dapat mengakses *file* setelah login.
7	Kinerja dan Skalabilitas	Menguji kinerja dan skalabilitas *object storage* saat menangani beban tinggi, seperti banyaknya *file* yang diunggah dan diunduh secara bersamaan.	Sistem mampu menangani beban tinggi tanpa penurunan kinerja yang signifikan, dan tetap responsif.

E. Pengujian Kinerja Sistem

Dari keseluruhan pengujian, dapat disimpulkan bahwa nilai *throughput* akan bertambah seiring bertambahnya juga jumlah *user*. Sedangkan, nilai *latency latency* dapat meningkat atau bahkan menurun sesuai dengan peningkatan jumlah pengguna serta jika server dapat menangani beban dengan baik.



Gambar 4. 8 Grafik Pengujian

Dari grafik pengujian diatas, dengan jumlah pengguna 100, nilai *throughput* relatif rendah tetapi nilai *latency* cukup stabil di sekitar 55ms. Sehingga, hal ini menunjukkan bahwa server 127.0.0.1:9001 mampu menangani beban dari 100 pengguna dengan *latency* yang masih dapat diterima. Ketika jumlah pengguna meningkat menjadi 500, nilai *throughput* meningkat secara signifikan hingga sekitar 16.69 *request per second* dan nilai *latency* juga meningkat menjadi 122.458ms. Sehingga, hal ini menunjukkan bahwa server 127.0.0.1:9001 dapat memproses lebih banyak permintaan, waktu respons rata-rata juga akan meningkat. Sedangkan, dengan jumlah pengguna 1000, nilai *throughput* mencapai 33.28 *request per second* meningkat dua kali lipat dari hasil pengujian 500 pengguna. Meskipun demikian, nilai *latency* menurun menjadi 53.98ms. Sehingga, hal ini menunjukkan bahwa server 127.0.0.1:9001 dapat menangani beban yang lebih besar dengan *latency* yang masih relatif rendah.

Sehingga, dari pengujian diatas dapat disimpulkan bahwa kinerja server dikatakan baik karena *throughput* dapat menerima beban *request* dengan banyak *user* dengan dibuktikan nilai *throughput* meningkat sesuai dengan jumlah *user*. Serta, *latency* yang didapat menurun sesuai dengan peningkatan jumlah *user*, yang dimana secara umum nilai *latency* yang rendah adalah kondisi kinerja sistem yang baik.

V. KESIMPULAN DAN SARAN

A. Kesimpulan

Berdasarkan hasil perancangan, pengujian dan analisa yang telah dilakukan maka dapat diambil beberapa kesimpulan sebagai berikut:

1. Sistem *object storage* berbasis protokol S3 telah berhasil diimplementasikan dan diintegrasikan dengan produk Coofis Verse dan aplikasi alternatif *reimburse*.
2. Fitur seperti penyimpanan, pengambilan, penghapusan objek, serta pembatasan akses telah berfungsi sesuai dengan yang diharapkan.
3. Sistem ini mampu memastikan keamanan data melalui pembatasan akses dan autentikasi pengguna. Hanya pengguna yang memiliki izin yang dapat mengakses file tertentu, menjaga kerahasiaan dan integritas data.

B. Saran

Berdasarkan hasil pengujian dan implementasi yang telah dilakukan, beberapa saran untuk pengembangan lebih lanjut dari sistem ini adalah:

1. Tambahkan fitur-fitur canggih seperti versioning (penyimpanan beberapa versi dari file yang sama), enkripsi end-to-end untuk peningkatan keamanan, dan fitur tagging untuk pengelolaan file yang lebih baik.
2. Lakukan optimisasi lebih lanjut untuk meningkatkan kinerja sistem, terutama untuk operasi penyimpanan dan pengambilan file berukuran besar. Penerapan teknologi caching dapat membantu dalam hal ini.
3. Pertimbangkan untuk mengadopsi arsitektur microservices dan containerization (misalnya menggunakan Docker dan Kubernetes) untuk

meningkatkan skalabilitas dan fleksibilitas sistem dalam penanganan beban yang lebih dinamis.

DAFTAR PUSTAKA

[1] J. Riset, M. Bidang, T. Informasi, D. Radya Briantama, and N. D. Hendrawan, "'Bimasakti' Aplikasi Persuratan Digital Berbasis Web Untuk Manajemen Dokumen Dengan Metode Addie." [Online]. Available: https://ejournal.unikama.ac.id/index.php/JFTI

[2] K. F. Ribawanto and D. Pramono, "Pengembangan Sistem Nota Dinas Elektronik dengan Tanda Tangan Elektronik Studi Kasus PT Andal Rancang Multi Solusi (Arm Solusi)," 2022. [Online]. Available: http://j-ptiik.ub.ac.id

[3] A. Sinambela and F. Farady Coastera, "Implementasi Arsitektur Microservices Pada Rancang Bangun Aplikasi Marketplace Berbasis Web," 2021. [Online]. Available: http://ejournal.unib.ac.id/index.php/rekursif/1

[4] B. Kristomoyo Kristanto, S. Widyayuningtias, P. Listio, Y. A. Kanthi, P. I. Baskoro, and M. Z. Fauzi, "Implementasi Object Storage Meningkatkan Efisiensi Biaya Penggunanan Server (StudiKasus Sistem Manajemen Lomba)."

[5] S. Ginata, A. Kusyanti, and R. Primananda, "Implementasi Algoritme Kriptografi SIMON pada Arsitektur Amazon Web Services," 2019. [Online]. Available: http://j-ptiik.ub.ac.id

[6] P. Mishra, R. Pitchumani, and Y.-S. Kee, "Learnings from an Under the Hood Analysis of an Object Storage Node IO Stack."