

**DESAIN DAN IMPLEMENTASI CDN *TESTBED* UNTUK  
EVALUASI PERFORMA DISTRIBUSI KONTEN BERBASIS  
*NGINX***

**TUGAS AKHIR**

**Karya tulis sebagai salah satu syarat  
untuk memperoleh gelar Sarjana dari  
Institut Teknologi Bandung**

**Oleh  
AKBAR FEBRY WAHYU ANDRIAN  
NIM: 18121022  
(Program Studi Teknik Telekomunikasi)**



**INSTITUT TEKNOLOGI BANDUNG  
OKTOBER 2025**

## ABSTRAK

### DESAIN DAN IMPLEMENTASI CDN *TESTBED* UNTUK EVALUASI PERFORMA DISTRIBUSI KONTEN BERBASIS NGINX

Oleh  
**Akbar Febry Wahyu Andrian**  
**NIM: 18121022**  
**(Program Studi Teknik Telekomunikasi)**

Di era serba digital saat ini, internet telah menjadi salah satu kebutuhan primer untuk sebagian besar orang. Jumlah konten situs web yang semakin banyak dan jenis media yang semakin berat seperti gambar dan video ditambah permintaan akses dari pengguna ke situs tersebut yang terus bertumbuh menjadi tantangan yang sangat besar bagi penyedia layanan konten. Fenomena ini menyebabkan dua masalah utama: latensi yang tinggi bagi pengguna yang mengakses server dari lokasi yang jauh secara geografis, dan peningkatan biaya operasional terkait *bandwidth* bagi penyedia konten.

*Content Delivery Network* (CDN) telah menjadi solusi standar industri untuk mengatasi masalah ini dengan menggunakan jaringan server *cache* yang terdistribusi untuk mendekatkan konten dengan pengguna. Meskipun efektif, performa sebuah CDN sangat bergantung pada infrasrtuktur secara keseluruhan maupun strategi internalnya seperti kebijakan penghapusan *cache* (*cache eviction*). Kebijakan konvensional seperti *Least Recently Used* (LRU) yang umum digunakan seringkali terbukti suboptimal dalam menghadapi pola akses pengguna modern yang dinamis, sehingga perlu adanya pendekatan yang lebih dinamis. Di sisi lain, banyak penelitian canggih dalam optimasi CDN divalidasi hanya melalui simulasi, menciptakan kesenjangan antara teori dan implementasi praktis yang dapat direproduksi. Penelitian ini bertujuan untuk mengisi kesenjangan tersebut dengan merancang dan mengimplementasikan sebuah CDN *testbed* yang komprehensif, serta memanfaatkannya untuk mengevaluasi efektivitas kebijakan *cache eviction* cerdas berbasis *machine learning*.

Metodologi yang digunakan dalam penelitian ini meliputi perancangan dan implementasi sistem yang terdiri dari tiga subsistem utama: server, pemantauan, dan pembelajaran mesin. Subsistem server dibangun menggunakan NGINX sebagai *cache server* yang di-deploy pada tiga *virtual machine* di region berbeda (Asia Tenggara, Eropa, dan Amerika), dengan aplikasi web berbasis Flask sebagai *origin server* dan Gcore sebagai penyedia layanan GeoDNS. Subsistem pemantauan diimplementasikan menggunakan Grafana sebagai *dashboard* visualisasi, Prometheus sebagai pengumpul metrik, serta beberapa *exporter* tambahan seperti Fluentd untuk memastikan akurasi data *log*. Untuk subsistem pembelajaran mesin, digunakan algoritma *Learning from the Experts* (LRB) yang

diimplementasikan pada sebuah *virtual machine* terpisah untuk menganalisis *log* akses dan menentukan konten yang akan dihapus dari *cache*. Hipotesis utama dari penelitian ini adalah bahwa *testbed* yang dirancang mampu berfungsi sebagai platform evaluasi yang efektif, dibuktikan dengan kemampuannya untuk mengukur perbedaan performa secara kuantitatif antara kebijakan *cache* konvensional (LRU) dan berbasis *machine learning* (LRB).

Hasil pengujian menunjukkan bahwa ketiga subsistem berhasil diimplementasikan dan berfungsi sesuai rancangan. Subsistem pemantauan terbukti mampu menampilkan metrik server dengan akurasi tinggi, dimana selisih antara nilai pada dasbor dan nilai aktual pada server tidak melebihi 10%. Subsistem server juga terbukti secara signifikan mengurangi latensi akses dibandingkan dengan sistem tanpa CDN; sebagai contoh, waktu akses rata-rata untuk konten gambar turun dari 253 ms menjadi 51,75 ms pada pengujian ideal. Sebagai validasi utama dari kegunaan *testbed*, dilakukan pengujian komparatif pada kebijakan *cache eviction*. Pada skenario simulasi lalu lintas dengan distribusi Zipf yang merepresentasikan pola akses dunia nyata, algoritma LRB berhasil mencapai *Cache Hit Ratio* (CHR) rata-rata sebesar 90.15%, secara signifikan mengungguli algoritma *baseline* LRU (85.84%) dan LFU (83.26%). Hasil serupa juga ditunjukkan pada pengujian dengan distribusi uniform, di mana LRB mencapai CHR 92.41% dibandingkan LRU (90.32%). Keberhasilan ini membuktikan hipotesis bahwa *testbed* mampu mengukur dan memvalidasi keunggulan pendekatan cerdas dalam manajemen *cache*.

Kontribusi utama dari tugas akhir ini adalah terwujudnya sebuah CDN *testbed* yang fungsional, terukur, dan dapat direproduksi, yang dapat menjembatani kesenjangan antara penelitian berbasis simulasi dan implementasi skala produksi. *Testbed* ini tidak hanya berhasil memvalidasi bahwa kebijakan *cache eviction* berbasis *machine learning* lebih unggul, tetapi juga menyediakan platform terbuka bagi penelitian di masa depan untuk bereksperimen dengan berbagai teknik optimasi CDN lainnya. Dengan demikian, penelitian ini menyumbangkan sebuah platform praktis bagi komunitas akademik dan industri untuk mengembangkan solusi distribusi konten yang lebih efisien dan cerdas.

*Kata kunci:* Content Delivery Network, Testbed, Cache, Cache Eviction, NGINX, Machine Learning.

## ***ABSTRACT***

### ***DESIGN AND IMPLEMENTATION OF A CDN TESTBED FOR PERFORMANCE EVALUATION OF NGINX-BASED CONTENT DISTRIBUTION***

*By*

**Akbar Febry Wahyu Andrian**

**NIM: 18121022**

**(Telecommunication Engineering Program)**

*In today's digital era, the internet has become a primary necessity for most people. The growing amount of web content and heavier media types, such as images and videos, coupled with the increasing user demand for access, poses a significant challenge for content providers. This phenomenon leads to two main problems: high latency for users accessing servers from geographically distant locations and increased operational costs related to bandwidth for content providers.*

*A Content Delivery Network (CDN) has become the industry-standard solution to address this issue by using a distributed network of cache servers to bring content closer to users. Although effective, the performance of a CDN heavily depends on its overall infrastructure as well as its internal strategies, such as the cache eviction policy. Conventional policies like the commonly used Least Recently Used (LRU) often prove to be suboptimal in handling modern, dynamic user access patterns, thus necessitating a more dynamic approach. On the other hand, much of the advanced research in CDN optimization is validated solely through simulations, creating a gap between theory and reproducible practical implementations. This research aims to fill this gap by designing and implementing a comprehensive CDN testbed and utilizing it to evaluate the effectiveness of an intelligent, machine learning-based cache eviction policy.*

*The methodology used in this research includes the design and implementation of a system comprising three main subsystems: server, monitoring, and machine learning. The server subsystem is built using NGINX as a cache server deployed on three virtual machines in different regions (Southeast Asia, Europe, and America), with a Flask-based web application as the origin server and Gcore as the GeoDNS service provider. The monitoring subsystem is implemented using Grafana as a visualization dashboard, Prometheus as a metrics collector, and several additional exporters like Fluentd to ensure log data accuracy. For the machine learning subsystem, the Learning from the Experts (LRE) algorithm is implemented on a separate virtual machine to analyze access logs and determine which content to evict from the cache. The main hypothesis of this research is that the designed testbed can function as an effective evaluation platform, proven by its ability to quantitatively measure performance differences between a conventional cache policy (LRU) and a machine learning-based one (LRE).*

*Testing results show that all three subsystems were successfully implemented and function as designed. The monitoring subsystem proved capable of displaying server metrics with high accuracy, where the difference between the dashboard values and the actual server values did not exceed 10%. The server subsystem also proved to significantly reduce access latency compared to a system without a CDN; for example, the average access time for image content dropped from 253 ms to 51.75 ms in ideal tests. As the primary validation of the testbed's utility, a comparative test on cache eviction policies was conducted. In a traffic simulation scenario with a Zipf distribution, which represents real-world access patterns, the LRB algorithm achieved an average Cache Hit Ratio (CHR) of 90.15%, significantly outperforming the baseline LRU (85.84%) and LFU (83.26%) algorithms. Similar results were also shown in tests with a uniform distribution, where LRB achieved a CHR of 92.41% compared to LRU's 90.32%. This success proves the hypothesis that the testbed is capable of measuring and validating the superiority of an intelligent approach to cache management.*

*The main contribution of this final project is the creation of a functional, measurable, and reproducible CDN testbed, which can bridge the gap between simulation-based research and production-scale implementation. This testbed not only successfully validates that the machine learning-based cache eviction policy is superior but also provides an open platform for future research to experiment with various other CDN optimization techniques. Thus, this research contributes a practical platform for both the academic and industrial communities to develop more efficient and intelligent content delivery solutions.*

*Keywords:* Content Delivery Network, Testbed, Cache, Cache Eviction, NGINX, Machine Learning.

**DESAIN DAN IMPLEMENTASI CDN *TESTBED* UNTUK  
EVALUASI PERFORMA DISTRIBUSI KONTEN BERBASIS  
*NGINX***

**HALAMAN PERSETUJUAN**  
(untuk draft tugas akhir sebagai syarat sidang)

Oleh  
**Akbar Febry Wahyu Andrian**  
**NIM: 18121022**  
**(Program Studi Teknik Telekomunikasi)**

Institut Teknologi Bandung

Menyetujui  
Tim Pembimbing

Tanggal 28 Juli 2025

Ketua

Anggota

---

Dr. Ir. Mohammad Ridwan Effendi, Dr. Ir. Ian Josef Matheus Edward,  
M.A.Sc M.T.

NIP 196312011990011002

NIP 196811061994031003

**DESAIN DAN IMPLEMENTASI CDN *TESTBED* UNTUK  
EVALUASI PERFORMA DISTRIBUSI KONTEN BERBASIS  
NGINX**

**HALAMAN PENGESAHAN**  
**(untuk buku tugas akhir sebagai kelengkapan wisuda)**

Oleh  
**Akbar Febry Wahyu Andrian**  
**NIM: 18121022**  
**(Program Studi Teknik Telekomunikasi)**

Institut Teknologi Bandung

Menyetujui  
Tim Pembimbing

Tanggal 28 Juli 2025

Ketua

Anggota

---

Dr. Ir. Mohammad Ridwan Effendi, Dr. Ir. Ian Josef Matheus Edward,  
M.A.Sc M.T.

NIP 196312011990011002

NIP 196811061994031003

## **PEDOMAN PENGGUNAAN BUKU TUGAS AKHIR**

Buku Tugas Akhir yang tidak dipublikasikan terdaftar dan tersedia di Perpustakaan Institut Teknologi Bandung, dan terbuka untuk umum dengan ketentuan bahwa hak cipta ada pada penulis dengan mengikuti aturan HaKI yang berlaku di Institut Teknologi Bandung. Referensi kepustakaan diperkenankan dicatat, tetapi pengutipan atau peringkasan hanya dapat dilakukan seizin penulis dan harus disertai dengan kaidah ilmiah untuk menyebutkan sumbernya.

Situs hasil tugas akhir ini dapat ditulis dalam bahasa Indonesia sebagai berikut:

Andrian, Akbar Febry W. (2025): *Desain dan Implementasi CDN Testbed untuk Evaluasi Performa Distribusi Konten Berbasis NGINX*, Tugas Akhir Program Studi Teknik Telekomunikasi, Institut Teknologi Bandung.

dan dalam bahasa Inggris sebagai berikut:

Andrian, Akbar Febry W. (2025): *Design and Implementation of a CDN Testbed for Performance Evaluation of NGINX-based Content Distribution.*, Telecommunication Engineering Program, Institut Teknologi Bandung.

Memperbanyak atau menerbitkan sebagian atau seluruh buku tugas akhir haruslah seizin Dekan Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung.

*Untuk ibuku yang senantiasa mendoakan anaknya serta mendukung lahir dan  
batin untuk segala urusan.*

## KATA PENGANTAR

Puji dan syukur penulis panjatkan kepada ke hadirat Allah SWT yang telah memberikan rahmat dan karunia-Nya sehingga penulis dapat menyelesaikan masa mengenyam pendidikan tinggi yang diakhiri dengan dilakukannya Tugas Akhir ini. Tidak lupa penulis mengucapkan terima kasih kepada pihak-pihak yang telah membantu dan mendukung penulis dalam bentuk motivasi, doa, dan pertolongan lainnya selama masa kuliah. Ucapan terima kasih penulis sampaikan kepada:

1. Ibu, Bapak serta keluarga besar yang selalu mendukung dalam hal doa, materi dan lainnya sehingga penulis mampu menjalani perkuliahan dengan baik.
2. Ibu Dr.-Ing. Chairunnisa, S.T., M.T. selaku Kepala Program Studi Teknik Telekomunikasi ITB yang memberikan dukungan atas keberlangsungan Tugas Akhir ini.
3. Bapak Dr. Ir. Mohammad Ridwan Effendi, M.A.Sc. dan Bapak Dr. Ir. Ian Josef Matheus Edward, M.T. selaku dosen pembimbing penulis yang telah membantu dalam penggerjaan Tugas Akhir ini baik terutama dalam pembentukan laporan dan pengembangan ide.
4. Bapak Insan Praja yang turut membantu untuk banyak hal dalam memberikan masukan dan saran teknis saat penggerjaan Tugas Akhir ini.
5. Ibu Dr. Irma Zakia, S.T, M.Sc. selaku dosen pengampu Mata Kuliah Tugas Akhir yang mengawasi keberjalanan Tugas Akhir ini.
6. Donatur Beasiswa IA ITB'78 beserta pengurus yang telah memberikan penulis bantuan biaya hidup selama setengah masa kuliah, yang sangat membantu penulis menjalani perkuliahan secara finansial.
7. Hanif Al Falih selaku *partner* penulis dalam pelaksanaan Tugas Akhir ini.
8. Mutiara Qamiliah yang selalu bersama dan mendukung penulis dari pertengahan masa kuliah dan menjadi alasan lain hingga saat ini penulis masih bertahan.
9. Teman-teman seperjuangan Aegis ET'21 yang selalu membantu selama keberlangsungan kuliah yang tidak bisa penulis sebutkan satu persatu.

10. Teman-teman Rumah Belajar YASBIL dan Pembina yang memberikan penulis pelajaran hidup terutama dalam hal spiritual selama tinggal di Bandung.

Penulis menyadari masih banyak kekurangan yang luput dari perhatian pada Tugas Akhir ini. Oleh karena itu, penulis terbuka dengan adanya kritik serta saran sehingga penulis dapat menjadi lebih baik di kemudian hari.

## DAFTAR ISI

Abstrak .....	i
<i>Abstract</i> .....	iii
Halaman Persetujuan.....	v
Halaman Pengesahan .....	vi
Pedoman Penggunaan Buku Tugas Akhir.....	vii
Kata Pengantar .....	ix
Daftar Isi.....	xi
Daftar Gambar dan Ilustrasi.....	xiii
Daftar Tabel .....	xvi
Daftar Singkatan dan lambang .....	xvii
Bab I Pendahuluan .....	1
1.1 Latar Belakang .....	1
1.2 Rumusan Masalah.....	2
1.3 Tujuan dan Manfaat .....	2
1.4 Lingkup Permasalahan.....	2
1.5 Asumsi-asumsi .....	3
1.6 Hipotesis .....	3
1.7 Sistematika Buku Tugas Akhir .....	3
Bab II Proses dan pengembangan desain .....	5
2.1 Tinjauan Pustaka .....	5
2.1.1 Arsitektur dan Prinsip Kerja Content Delivery Network (CDN).....	5
2.1.2 Perancangan dan Implementasi CDN Testbed.....	6
2.1.3 Penerapan Machine Learning untuk Cache Eviction: Tinjauan Penelitian Terkini.....	7
2.2 Persyaratan Desain.....	8
2.2.1. Objektif Desain .....	8
2.2.2. Constraint Desain .....	9
2.2.3. Fungsi dan Tingkah Laku .....	10
2.2.3.1. Fungsi.....	10
2.2.3.2. Tingkah Laku .....	11
2.3 Konsep Desain .....	12
2.3.1. Pengukuran Persyaratan Desain.....	12
2.3.1.1. Metrik Objektif .....	12
2.3.1.2. Pengukuran Constraint Desain.....	12
2.3.1.2. Spesifikasi Fungsional .....	13
2.4 Pengembangan Desain .....	14
2.4.1. Alternatif Metode Desain .....	14
2.4.1.1. Subsistem Antarmuka .....	16
2.4.1.2. Subsistem Server.....	18
2.4.1.3. Subsistem Pembelajaran Mesin .....	20
2.4.2. Sistem Terintegrasi .....	23
2.4.3. Penentuan Solusi Desain Terbaik.....	23
2.4.3.1. Pembobotan Kategori dari Alternatif Metode Desain.....	23

2.4.3.2. Matriks Keputusan .....	24
Bab III Detil desain dan Implementasi.....	25
3.1    Model Desain .....	25
3.1.1 Desain Subsistem Pemantauan.....	27
3.1.2 Desain Subsistem Server.....	29
3.2    Implementasi.....	30
3.2.1 Konsep Pengujian Sistem.....	30
3.2.1 Implementasi Aplikasi Web.....	31
3.2.2 Implementasi <i>Cache Server NGINX</i> .....	34
3.2.2 Implementasi Sistem Pemantauan .....	40
Bab IV Pengujian dan Analisis .....	46
4.1    Pengujian Desain .....	46
4.1.1. Subsistem Pemantauan.....	46
4.1.2. Subsistem Server.....	50
4.2    Analisis Hasil Pengujian .....	69
4.2.1. Analisis Hasil Pengujian Subsistem Pemantauan .....	69
4.2.2. Analisis Hasil Pengujian Subsistem Server .....	69
4.2.3. Analisis Hasil Pengujian Sistem Terintegrasi.....	70
Bab V Simpulan dan Saran .....	72
Daftar Pustaka .....	74

## **DAFTAR GAMBAR DAN ILUSTRASI**

Gambar II-1	Arsitektur umum Content Delivery Network (CDN) [8].....	6
Gambar II-2	Pohon diagram objektif dan subobjektif .....	8
Gambar II-3	Objektif dan subobjektif bentuk indentasi .....	9
Gambar II-4	Constraint untuk objektif bentuk indentasi .....	9
Gambar II-5	Flowchart sistem secara keseluruhan .....	11
Gambar III-1	Gambar arsitektur diagram dan alur sistem CDN.....	25
Gambar III-2	Alur kerja dari sistem pemantauan .....	28
Gambar III-3	Desain tampilan sederhana pada Grafana .....	28
Gambar III-4 (a)	Desain tampilan utama yang sederhana pada website <i>dummy</i>	
	Desain tampilan salah satu kategori gambar sederhana pada website <i>dummy</i>	
	29	
Gambar III-5	Alur kerja dari subsistem server .....	30
Gambar III-6	Struktur folder pada aplikasi Flask .....	32
Gambar III-7	Program respon untuk permintaan ke <i>endpoint root</i> pada app.py ..	32
Gambar III-8	Program untuk merespon permintaan ke <i>endpoint /images</i> pada app.py	33
Gambar III-9 (a)	Tampilan antarmuka halaman utama dari aplikasi web (b)	
	Tampilan antarmuka untuk kategori <i>trending</i> dari aplikasi web untuk konten asli (c) Tampilan antarmuka untuk kategori <i>trending</i> dari aplikasi web untuk konten unggahan pengguna.....	
	34	
Gambar III-10	Cuplikan konfigurasi <i>nginx.conf</i> yang mendefinisikan <i>path cache</i>	
	35	
Gambar III-11	Contoh struktur direktori <i>cache</i> pada NGINX.....	35
Gambar III-12	Cuplikan konfigurasi <i>nginx.conf</i> yang mendefinisikan format <i>log main</i>	
	36	
Gambar III-13	Cuplikan konfigurasi <i>cdn.conf</i> yang mendefinisikan nama server, sertifikat SSL dan HTTP <i>header</i> untuk keamanan.....	38
Gambar III-14	Cuplikan konfigurasi <i>cdn.conf</i> yang mendefinisikan HTTP <i>header</i> untuk <i>cache</i> , kompresi konten, dan URL yang di <i>cache</i> .....	39
Gambar III-15	Alur kerja dari sistem pemantauan .....	40
Gambar III-16	Isi dari <i>node-exporter.service</i> .....	41
Gambar III-17	Cuplikan konfigurasi <i>fluent.conf</i> .....	42
Gambar III-18	Cuplikan konfigurasi <i>prometheus.yml</i> .....	43
Gambar III-19	Visualisasi dengan Grafana untuk <i>throughput</i> .....	44
Gambar III-20 (a)	Tampilan antarmuka Grafana untuk sistem pemantauan untuk metrik CPU, RAM penyimpanan, kapasitas dan latensi (b) Tampilan	

antarmuka Grafana untuk sistem pemantauan untuk penyimpanan, kapasitas, latensi, ketersediaan dan <i>cache hit ratio</i> .....	45
Gambar IV-1 Tampilan antarmuka Grafana .....	46
Gambar IV-2 Tampilan sistem pada Ubuntu Server SEA .....	47
Gambar IV-3 Tampilan Grafana setelah dilakukan pengujian terhadap perubahan nilai	47
Gambar IV-4 Hasil Grafana k6 yang menunjukkan nilai terukur setelah pengetesan	
48	
Gambar IV-5 Visualisasi <i>troughput</i> dengan opsi rentang waktu (a) 12 jam, (b) 3 jam dan (c) 5 menit terakhir pada Grafana.....	49
Gambar IV-6 Hasil akses ke halaman utama aplikasi web Flask .....	50
Gambar IV-7 Hasil akses ke halaman <i>trending images</i> aplikasi web Flask .....	51
Gambar IV-8 Hasil inspect element ke halaman <i>trending images</i> aplikasi web Flask	
52	
Gambar IV-9 Hasil HTTP <i>header</i> pada <i>browser</i> .....	52
Gambar IV-10 Hasil HTTP <i>header</i> dengan menggunakan metode <i>curl</i> .....	53
Gambar IV-11 Gambar (a) <i>form</i> pengunggahan konten dan (b) halaman <i>trending</i> setelah dilakukan pengunggahan konten.....	54
Gambar IV-12 Hasil <i>inspect element</i> untuk halaman <i>trending page</i> untuk konten yang diunggah pengguna.....	54
Gambar IV-13 Gambar (a) <i>form</i> pengubahan konten dan (b) halaman popular setelah dilakukan pengubahan konten dengan mengubah nama.....	55
Gambar IV-14 Gambar (a) <i>form</i> pengubahan konten dan (b) halaman <i>popular</i> setelah dilakukan pengubahan konten dengan gambar baru dengan mengubah gambar .....	55
Gambar IV-15 Gambar konfirmasi penghapusan konten dan hasil setelah konten dihapus .....	56
Gambar IV-16 Gambar peta server dan pengguna virtual .....	56
Gambar IV-17 Tabel hasil pengujian pada website <i>ping-admin</i> .....	57
Gambar IV-18 Tampilan Grafana setelah dilakukan pengujian (1).....	58
Gambar IV-19 Tampilan Grafana setelah dilakukan pengujian (2).....	59
Gambar IV-20 Laporan Grafana k6 setelah dilakukan pengujian untuk sistem dengan CDN .....	59
Gambar IV-21 Laporan Grafana k6 setelah dilakukan pengujian untuk sistem tanpa CDN	60
Gambar IV-22 Pengujian dengan <i>curl test</i> untuk salah satu konten video .....	61
Gambar IV-23 Pengujian dengan <i>curl test</i> untuk salah satu konten gambar .....	61
Gambar IV-24 Cuplikan pengaturan VM untuk ML .....	63
Gambar IV-25 Struktur folder VM .....	64
Gambar IV-26 Struktur folder VM ml-cdn .....	65

Gambar IV-27 File cache_index.json dari Akses CDN .....	66
Gambar IV-28 Grafik CHR dari Tiga Algoritma pada Pengujian. ....	67
Gambar IV-29 Grafik CHR dari Tiga Algoritma pada Pengujian Region EU .....	68

## DAFTAR TABEL

Tabel II-1 Penentuan persyaratan fungsional dan fungsi pada desain CDN <i>testbed</i>	10
Tabel II-2 Metrik pengukuran dan syarat pemenuhan subobjektif pada desain CDN <i>testbed</i>	12
Tabel II-3 Metrik pengukuran dan syarat pemenuhan subobjektif dengan <i>constraint</i> pada desain CDN <i>testbed</i> .....	12
Tabel II-4 Spesifikasi fungsi pada desain CDN <i>testbed</i> .....	13
Tabel II-5 Pemetaan persyaratan desain menjadi alternatif metode desain .....	14
Tabel II-6 Kombinasi alternatif desain sistem terintegrasi .....	23
Tabel II-7 Pembobotan pada kategori dari alternatif metode desain .....	23
Tabel II-8 Matriks keputusan alternatif desain sistem terintegrasi .....	24
Tabel III-1 Desain rinci subsistem dan pembagian kerja.....	26
Tabel III-2 Spesifikasi fungsi dan hasil yang diharapkan ketika pengujian .....	30
Tabel III-3 Nama dan deskripsi singkat format <i>access.log</i> .....	36
Tabel III-4 Kode HTTP respon dan penjelasan singkat.....	37
Tabel IV-1 Spesifikasi fungsi dan pengujian subsistem pemantauan .....	49
Tabel IV-2 Spesifikasi fungsi dan pengujian subsistem server .....	62
Tabel IV-3 Hasil CHR dari Pengujian Algoritma <i>Eviction</i> Distribusi Zipf dan Region US .....	67
Tabel IV-4 Hasil CHR dari Pengujian Algoritma <i>Eviction</i> Distribusi Uniform dan Region EU .....	69

## DAFTAR SINGKATAN DAN LAMBANG

SINGKATAN	Nama	Pemakaian pertama kali pada halaman
API	Application Programming Interface	17
CDN	Content Delivery Network	1
CHR	Cache Hit Ratio	2
CPU	Central Processing Unit	16
CRUD	Create, Read, Update, and Delete	31
CSS	Cascading Style Sheets	31
DNS	Domain Name System	7
GeoDNS	Geographic Domain Name System	29
HTML	HyperText Markup Language	6
HTTPS	Hypertext Transfer Protocol Secure	17
ICP	Internet Content Provider	1
ISP	Internet Service Provider	1
ISO	International Organization for Standardization	12
JSON	JavaScript Object Notation	65
LFU	Least Frequently Used	7
LRB	Learning from the Experts	2
LRU	Least Recently Used	7
LSTM	Long Short-Term Memory	19
ML	Machine Learning	1
NGINX	Engine X (Nama Proyek)	2
OTT	Over-the-Top	1
QoE	Quality of Experience	1
RAM	Random Access Memory	44
RNN	Recurrent Neural Network	19
SSH	Secure Shell	34
SSL	Secure Sockets Layer	37
TPS	Transactions Per Second	12
URL	Uniform Resource Locator	38
VCL	Varnish Caching Language	18
VM	Virtual Machine	26
WAN	Wide Area Network	20

# BAB I PENDAHULUAN

## 1.1 Latar Belakang

Di era serba digital saat ini, internet telah menjadi salah satu kebutuhan primer untuk sebagian besar orang. Jumlah konten situs web yang semakin banyak dan jenis media yang semakin berat seperti gambar dan OTT ditambah permintaan akses dari pengguna ke situs tersebut yang terus bertumbuh menjadi tantangan yang sangat besar bagi ICP(Li et al., 2020). Seperti pada tahun 2022, *traffic* internet global naik sebesar 23% dan konten yang berupa video berkontribusi pada 65% dari semua *traffic*(Vanerio et al., 2024).

Pengguna yang melakukan permintaan ke server yang letaknya jauh cenderung mengalami penurunan pengalaman pengguna. Oleh karena itu, jarak dari penyedia konten dengan pengguna perlu didekatkan untuk alasan mengurangi latensi untuk kenyamanan pengguna(Fan et al., 2024). Lebih lanjut, jarak yang lebih jauh antara server dan pengguna mengakibatkan jumlah ISP yang dilalui semakin banyak, yang berimplikasi pada peningkatan biaya *bandwidth*(Adler et al., 2011).

CDN muncul sebagai jaringan yang muncul di atas jaringan Internet, yang merupakan server *cache* yang terdistribusi secara geografis untuk mendekatkan konten dengan pengguna(Stocker et al., 2017). Dengan mengalihkan permintaan pengguna ke server yang lebih dekat, akan meningkatkan *throughput*, mengurangi beban server utama dan mengurangi biaya *bandwidth*(Vanerio et al., 2024).

Beberapa penelitian menunjukkan adanya tren untuk meningkatkan kemampuan dari CDN, salah satunya adalah dalam hal penghapusan *cache* untuk menghemat penyimpanan karena berkaitan langsung dengan *capital expenditure* untuk *cloud service*. Penghapusan *cache* juga harus dilakukan dengan cermat untuk tidak mengurangi kualitas pengalaman (QoE) pengguna(Hu et al., 2022). Selain itu, pengembangan CDN *testbed* untuk menguji keandalan dari CDN untuk menguji hal-hal yang bersifat eksperimental sedang banyak dilakukan(Absur et al., 2024). Hal ini menunjukkan diperlukan adanya lingkungan untuk melakukan eksperimen optimasi CDN.

## 1.2 Rumusan Masalah

Rumusan masalah proyek tugas akhir ini adalah:

1. Bagaimana merancang dan mengimplementasikan sebuah CDN *testbed* yang fungsional dan dapat direproduksi menggunakan NGINX sebagai *cache server*?
2. Bagaimana merancang subsistem pemantauan yang mampu mengukur metrik-metrik performa dari CDN, seperti *Cache Hit Ratio* (CHR), latensi, dan *throughput* secara akurat?
3. Bagaimana *testbed* yang telah dibangun dapat dimanfaatkan untuk melakukan evaluasi dan perbandingan performa antara kebijakan *cache* konvensional dan yang berbasis *machine learning*?

## 1.3 Tujuan dan Manfaat

Tujuan utama dari tugas akhir ini adalah merancang dan mengimplementasikan sebuah Content Delivery Network (CDN) *Testbed* yang komprehensif. *Testbed* ini bertujuan untuk menyediakan lingkungan yang terkontrol dan terukur untuk melakukan evaluasi performa berbagai metode *caching*. Sebagai studi kasus untuk mendemonstrasikan kapabilitasnya, *testbed* ini akan digunakan untuk mengevaluasi dan membandingkan performa antara kebijakan *cache eviction* NGINX (LRU) dengan kebijakan cerdas berbasis *machine learning* (LRB).

## 1.4 Lingkup Permasalahan

Lingkup permasalahan dalam tugas akhir ini dibatasi oleh hal-hal berikut:

1. Lingkup Data
  - a. Data yang digunakan dibuat berdasarkan akses situs *dummy* yang penulis buat dengan isi media sebanyak 100 gambar dan 100 video sebagai sumber konten.
2. Lingkup Evaluasi
  - a. Performa sistem CDN dievaluasi berdasarkan metrik latensi dan *throughput*.
  - b. Perbandingan hanya dilakukan terhadap sistem tanpa *caching*
  - c. Analisis dibatasi hanya pada *cache eviction* dengan model *machine learning* LRB yang sudah dilatih.

## **1.5 Asumsi-asumsi**

Dalam perancangan dan pengujian *testbed* ini, beberapa asumsi digunakan adalah sebagai berikut:

1. Infrastruktur *public cloud* yang digunakan diasumsikan menyediakan sumber daya juga jaringan yang stabil dan konsisten selama periode pengujian, sehingga anomali performa dianggap berasal dari sistem yang diuji, bukan dari fluktuasi jaringan penyedia *cloud*.
2. Alat yang digunakan untuk pengujian beban (*load testing*) seperti Grafana k6 mampu menghasilkan pola trafik yang cukup representatif untuk mengukur metrik-metrik performa seperti latensi dan *throughput*.
3. Komponen *open-source* yang digunakan (NGINX, Prometheus, Grafana) diasumsikan berfungsi sesuai dengan dokumentasi resminya dalam konfigurasi yang diimplementasikan.

## **1.6 Hipotesis**

Hipotesis dari tugas akhir ini adalah *testbed* yang dirancang mampu berfungsi sebagai *platform* evaluasi yang efektif untuk menguji performa metode *caching*. Keefektifan *testbed* ini divalidasi dengan kemampuannya untuk secara kuantitatif mengukur dan menunjukkan perbedaan performa yang signifikan (terutama pada metrik *Cache Hit Ratio* dan latensi) antara sistem yang menggunakan kebijakan *cache* konvensional dengan sistem yang telah dioptimalkan menggunakan model *machine learning*.

## **1.7 Sistematika Buku Tugas Akhir**

Buku tugas akhir ini terdiri atas lima bab yang disusun secara sistematis dengan detil dari isetiap bab sebagai berikut:

### **1. BAB I PENDAHULUAN**

Bab ini terdiri atas latar belakang penelitian, rumusan masalah, tujuan dan manfaat, lingkup permasalahan, asumsi-asumsi, hipotesis penelitian, serta sistematika penulisan dari buku tugas akhir ini.

### **2. BAB II PROSES PENGEMBANGAN DESAIN**

Pada bab ini dijelaskan teori-teori serta konsep dasar yang dijelaskan pada bagian tinjauan pustaka, lalu persyaratan desain dari rancangan sistem yang terdiri atas objektif, fungsi dan tingkah laku desain, serta *constraint*. Selain itu, terdapat konsep desain yang mencakup pengembangan desain, alternatif desain, pemilihan desain, serta penjelasan subsistem desain.

**3. BAB III DETIL DESAIN DAN IMPLEMENTASI**

Bab ini berisi detil dari model desain yang dibuat berdasarkan desain terpilih serta implementasi dari sistem yang dibuat.

**4. BAB IV PENGUJIAN DAN ANALISIS**

Bab ini berisi pembahasan dari hasil pengujian untuk setiap subsistem yang ditentukan serta sistem terintegrasi, termasuk dengan analisis dari hasil pengujian tersebut.

**5. BAB V SIMPULAN DAN SARAN**

Pada bab ini terdapat kesimpulan dari seluruh pelaksanaan proyek tugas akhir serta saran untuk pengembangan lebih lanjut di masa depan.

## **BAB II PROSES DAN PENGEMBANGAN DESAIN**

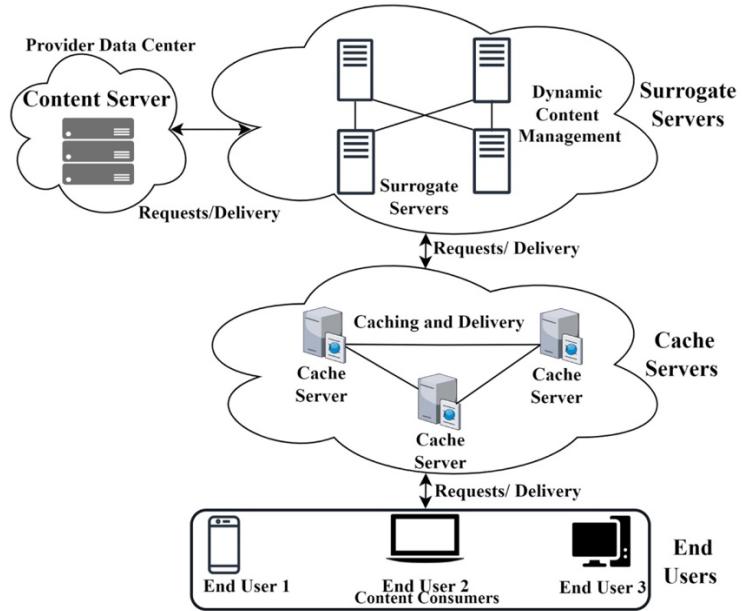
### **2.1 Tinjauan Pustaka**

#### **2.1.1 Arsitektur dan Prinsip Kerja Content Delivery Network (CDN)**

Content Delivery Network (CDN) adalah infrastruktur terdistribusi yang terdiri dari sekumpulan server *proxy* yang tersebar secara geografis untuk menyampaikan konten digital kepada pengguna akhir dengan performa tinggi(Ali et al., 2025). CDN terdiri dari infrastruktur terdistribusi server proxy yang dirancang untuk menyampaikan konten digital kepada pengguna akhir secara efektif.

CDN bekerja dengan prinsip menyimpan *cache* konten dekat dengan pengguna akhir untuk memungkinkan transfer asset yang diperlukan untuk memuat konten internet dengan cepat termasuk halaman HTML, file JavaScript, *stylesheet*, gambar, dan video(Ali et al., 2025). CDN didefinisikan sebagai sistem yang mereplikasi dan menyimpan *cache* data di seluruh jaringan server yang tersebar di berbagai area geografis.

Tujuan utama CDN adalah mengatasi keterbatasan internet dan meningkatkan pengalaman pengguna. Konsep utama teknologi ini adalah penyampaian di titik-titik *edge* jaringan, dekat dengan area permintaan, untuk meningkatkan performa yang dirasakan pengguna.



Gambar II-1 Arsitektur umum Content Delivery Network (CDN)(Ali et al., 2025)

Saat ini, terdapat beberapa jenis arsitektur CDN yang digunakan, seperti CDN-P2P, Mobile-based CDN, Cloud-based CDN, Multi-CDN, Software-based CDN, Blockchain-based CDN dan Fog-based CDN. Masing-masing arsitektur memiliki kegunaan, kelebihan dan kekurangannya. Pada sebuah jurnal yang meneliti *state-of-the-art* CDN terkhusus untuk Cloud-based CDN menunjukkan tren penelitian yang menggunakan pendekatan canggih namun dilakukan dalam simulasi ideal, bukan dalam praktik nyata. Hal ini menyoroti bahwa pengembangan lingkungan pengujian perlu dapat direproduksi sehingga perlu lingkungan yang lebih realistik. Fokus lainnya juga cenderung pada skala makro seperti optimasi biaya penyewaan *cloud* dan penempatan *edge* server secara geografis, penelitian untuk skala mikro yang terjadi pada setiap server seperti penghematan penyimpanan lebih sedikit adanya.

### 2.1.2 Perancangan dan Implementasi CDN Testbed

Selain optimasi algoritma, riset terkini juga menyoroti pembangunan platform *testbed* yang andal untuk mengevaluasi performa Content Delivery Network (CDN). Sebagai contoh, penelitian oleh (Yang et al., 2023) memperkenalkan arsitektur Software Defined Content Delivery Network (SDCDN) yang

mengintegrasikan konsep Software Defined Networking (SDN) dengan CDN guna meningkatkan skalabilitas dan efisiensi distribusi konten. Testbed yang dikembangkan berfokus pada kemampuan untuk menguji berbagai skenario layanan multimedia, seperti *video on demand* dan *live streaming*, dengan memanfaatkan kontrol terpusat di lapisan kontrol SDN untuk penyesuaian dinamis rute dan pengelolaan cache pada edge server. Pendekatan ini mendukung *deployment eksperimental* yang fleksibel dengan isolasi layanan dan monitoring *real-time* berdasarkan parameter Quality of Experience (QoE) dan pemodelan beban lalu lintas. Tinjauan ini menunjukkan bahwa meskipun sudah ada beberapa platform, masih ada ruang untuk pengembangan *testbed* yang secara spesifik dirancang untuk menguji dan membandingkan kebijakan *cache eviction* cerdas secara terintegrasi dengan sistem pemantauan modern.

### **2.1.3 Penerapan Machine Learning untuk Cache Eviction: Tinjauan Penelitian Terkini**

Untuk mengatasi keterbatasan kebijakan *cache* konvensional, berbagai penelitian internasional telah menerapkan *machine learning* untuk merancang strategi *cache eviction* yang cerdas dan adaptif. Pendekatan ini bertujuan memprediksi nilai atau kegunaan item dalam *cache* berdasarkan pola akses historis, sehingga keputusan penghapusan didasarkan pada model yang terus belajar dan disempurnakan. Salah satu pendekatan mutakhir adalah *preference learning* dengan *automated feedback* yang diterapkan pada HALP oleh (Gummadi, 2023), dimana sistem *cache* dimodelkan sebagai agen yang secara dinamis memperbarui model *neural reward* untuk mengoptimalkan performa *cache* secara *real-time*, mengungguli heuristik tradisional dalam produksi skala besar.

Selain itu, *reinforcement learning* dan *deep learning* juga digunakan secara luas. (Krishna, 2025) mengulas model-model seperti DeepCache, RL-Cache, dan lainnya yang secara adaptif mengelola *cache* dengan belajar dari pola penggunaan waktu nyata, memberi performa unggul pada jaringan *edge* dan sistem terdistribusi, serta menurunkan latensi sekaligus meningkatkan Cache Hit Ratio. Pendekatan *supervised learning* dan *imitation learning* juga dipakai untuk menyempurnakan prediksi konten mana yang harus disimpan atau dihapus, memperkuat kesimpulan

bahwa keberadaan *testbed* dan sistem pemantauan modern sangat penting untuk validasi dan perbandingan kebijakan *cache* secara realistik dan komprehensif.

Tinjauan ini mengindikasikan bahwa integrasi *machine learning* dalam *cache eviction* membuka peluang signifikan untuk meningkatkan efisiensi CDN dan sistem caching secara keseluruhan, dengan model yang mampu belajar secara berkelanjutan, beradaptasi dengan perubahan pola akses, dan mengatasi masalah kekurangan data historis. Namun, penelitian lebih lanjut terkait deployment praktis dan optimalisasi *overhead* komputasi tetap diperlukan agar solusi ini dapat diadopsi secara luas.

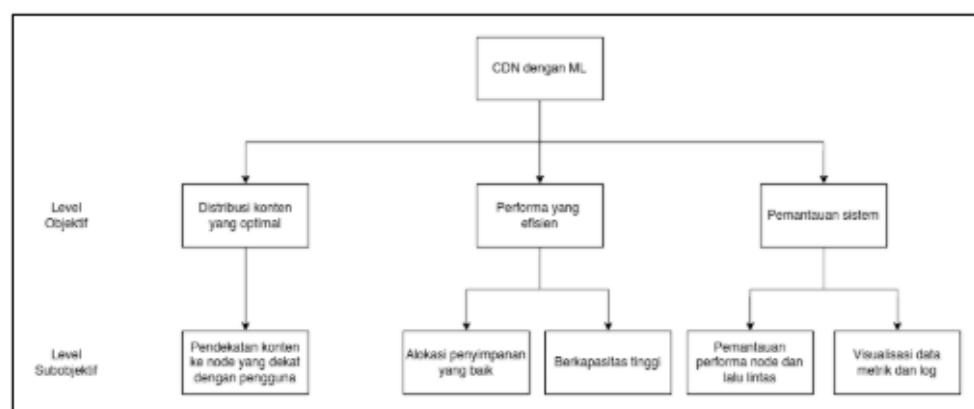
## 2.2 Persyaratan Desain

Pada penelitian tugas akhir ini, spesifikasi fungsionalitas dari sistem CDN dengan penambahan *machine learning* dipertimbangkan berdasarkan kemampuan dari penulis serta batasan-batasan yang sudah ditentukan di bab sebelumnya.

### 2.2.1. Objektif Desain

Desain yang dibuat diharapkan akan memiliki objektif sebagai berikut:

1. Distribusi konten yang optimal
2. Performa yang efisien
3. Pemantauan sistem



Gambar II-2 Pohon diagram objektif dan subobjektif

- |   |
|---|
| <ol style="list-style-type: none"> <li>1. Objektif 1: Distribusi konten yang optimal           <ol style="list-style-type: none"> <li>a. Subobjektif 1: Pendekatan konten ke <i>node</i> yang dekat dengan pengguna</li> </ol> </li> <li>2. Objektif 2: Performa yang efisien           <ol style="list-style-type: none"> <li>a. Subobjektif 1: Alokasi penyimpanan yang baik</li> <li>b. Subobjektif 2: Berkapasitas tinggi</li> </ol> </li> <li>3. Objektif 3: Pemantauan sistem           <ol style="list-style-type: none"> <li>a. Subobjektif 1: Pemantauan performa node dan lalu lintas</li> <li>b. Subobjektif 2: Visualisasi data metrik dan log</li> </ol> </li> </ol> |
|---|

Gambar II-3 Objektif dan subobjektif bentuk indentasi

Jumlah objektif ditentukan berdasarkan subsistem yang akan dibuat, yaitu subsistem antarmuka, subsistem server, dan subsistem pembelajaran mesin. Subsistem antarmuka berkolerasi dengan objektif 3, subsistem server berkolerasi dengan objektif 1, serta subsistem pembelajaran mesin berkolerasi dengan objektif 2. Ketiga objektif tersebut merupakan satu rangkaian yang harus terdapat pada sistem yang akan dibuat agar sistem dapat berjalan sesuai tujuan penulis.

### 2.2.2. Constraint Desain

*Constraint* desain adalah syarat atau batasan yang harus dipenuhi desain. Batasan ini digunakan agar desain dapat mudah dicapai. Selain itu, *constraint* digunakan untuk memfokuskan permasalahan agar dapat diselesaikan dengan lebih detil. *Constraint* pada sistem CDN dengan ML yang dibangun ada objektif kedua untuk setiap subobjektif. Berikut adalah constraint yang ditambahkan pada objektif dalam bentuk indentasi.

- |  |
|--|
| <ol style="list-style-type: none"> <li>1. Objektif 1: Distribusi konten yang optimal           <ol style="list-style-type: none"> <li>a. Subobjektif 1: Pendekatan konten ke <i>node</i> yang dekat dengan pengguna</li> </ol> </li> <li>2. Objektif 2: Performa yang efisien           <ol style="list-style-type: none"> <li>a. Subobjektif 1: Alokasi penyimpanan yang baik               <ol style="list-style-type: none"> <li>i. Constraint: Rasio <i>cache hit</i> harus mencapai minimal 80%</li> </ol> </li> <li>b. Subobjektif 2: Berkapasitas tinggi               <ol style="list-style-type: none"> <li>i. Constraint: Sistem harus mampu menangani permintaan dengan 95% dari permintaan tersebut (p95) memiliki latensi di bawah 200 milidetik.</li> </ol> </li> </ol> </li> <li>3. Objektif 3: Pemantauan sistem           <ol style="list-style-type: none"> <li>a. Subobjektif 1: Pemantauan performa node dan lalu lintas</li> <li>b. Subobjektif 2: Visualisasi data metrik dan log</li> </ol> </li> </ol> |
|--|

Gambar II-4 Constraint untuk objektif bentuk indentasi

Constraint desain adalah syarat atau batasan yang harus dipenuhi desain. Batasan ini digunakan agar desain dapat mudah dicapai. Selain itu, constraint digunakan untuk memfokuskan permasalahan agar dapat diselesaikan dengan lebih detil. Constraint pada sistem CDN dengan ML yang dibangun ada objektif kedua untuk setiap subobjektif. Berikut adalah constraint yang ditambahkan pada objektif dalam bentuk indentasi.

### **2.2.3. Fungsi dan Tingkah Laku**

#### **2.2.3.1. Fungsi**

Pada persyaratan desain, telah ditentukan. Agar persyaratan desain dapat dipenuhi, maka persyaratan fungsional yang harus dimiliki adalah sebagai berikut:

1. Melakukan analisis prediktif terhadap pola permintaan pengguna untuk optimasi *cache*
2. Menerima dan memproses permintaan pengguna
3. Menyalin konten yang sesuai ke berbagai *node*
4. Menampilkan metrik performa sistem dan log secara visual.
5. Mengarahkan permintaan pengguna ke *node* CDN terdekat
6. Mengelola siklus hidup konten di dalam *cache node* (menyimpan, memvalidasi, dan menghapus)

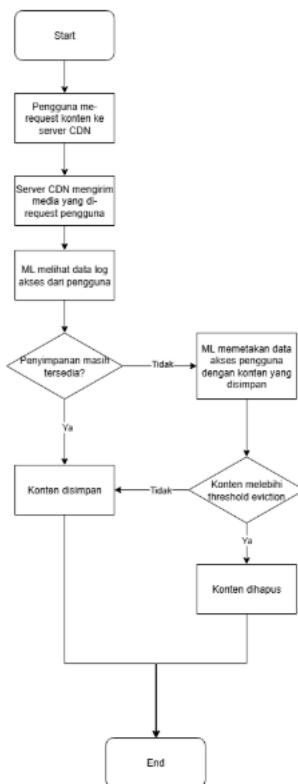
Pemetaan persyaratan desain subobjektif dan batasannya ke dalam persyaratan fungsional dan fungsi diperlihatkan pada Tabel II-1.

Tabel II-1 Penentuan persyaratan fungsional dan fungsi pada desain CDN *testbed*

<b>Subobjektif dan Constraint</b>	<b>Persyaratan Fungsional</b>	<b>Fungsi</b>
<b>Subobjektif 1:</b> Pendekatan konten ke <i>node</i> yang dekat pengguna	1. Melakukan analisis prediktif terhadap pola permintaan pengguna untuk optimasi <i>cache</i>	4: Subsistem antarmuka 2, 3, 5 : Subsistem server
<b>Subobjektif 2:</b> Alokasi penyimpanan yang baik	2. Menerima dan memproses permintaan pengguna	1, 6: Subsistem pembelajaran mesin
<b>Subobjektif 3:</b> Berkapasitas tinggi	3. Menyalin konten yang sesuai ke berbagai <i>node</i>	

<p><b>Subobjektif 4:</b> Pemantauan performa <i>node</i> dan lalu lintas</p> <p><b>Subobjektif 5:</b> Visualisasi data metrik dan log</p> <p><b>Constraint:</b> Rasio <i>cache hit</i> harus mencapai minimal 80%</p> <p><b>Constraint:</b> Sistem harus mampu menangani permintaan dengan 95% dari permintaan tersebut (p95) memiliki latensi di bawah 200 milidetik.</p>	<ol style="list-style-type: none"> <li>4. Menampilkan metrik performa sistem dan log secara visual.</li> <li>5. Mengarahkan permintaan pengguna ke <i>node</i> CDN terdekat</li> <li>6. Mengelola siklus hidup konten di dalam <i>cache node</i> (menyimpan, memvalidasi, dan menghapus)</li> </ol>	
--	---	--

### 2.2.3.2. Tingkah Laku



Gambar II-5 Flowchart sistem secara keseluruhan

## 2.3 Konsep Desain

### 2.3.1. Pengukuran Persyaratan Desain

#### 2.3.1.1. Metrik Objektif

Tabel II-2 Metrik pengukuran dan syarat pemenuhan subobjektif pada desain CDN *testbed*

Subobjektif	Metrik Pengukuran	Syarat Pemenuhan
Pendekatan konten ke <i>node</i> yang dekat pengguna	Akurasi Geolokasi	Latensi tidak melebihi 100 ms untuk konten web (referensi: (Faradilla A, 2023) <a href="https://web.dev/articles/rail">https://web.dev/articles/rail</a> )
Alokasi penyimpanan yang baik	Nilai CHR	CHR bernilai 95-99% (referensi: ( <i>What Is a Cache Hit Ratio?</i> , n.d.) <a href="https://www.cloudflare.com/learning/cdn/what-is-a-cache-hit-ratio/">https://www.cloudflare.com/learning/cdn/what-is-a-cache-hit-ratio/</a> )
Berkapasitas tinggi	Throughput	Throughput minimum 100 TPS (referensi: (ISO, 2011) Standard ISO 25010; System and software quality models)
Pemantauan performa <i>node</i> dan lalu lintas	Akurasi Data Monitoring	Selisih nilai yang ditampilkan Grafana dengan nilai aktual pada server tidak melebihi 10%
Visualisasi data metrik dan log		

#### 2.3.1.2. Pengukuran Constraint Desain

Tabel II-3 Metrik pengukuran dan syarat pemenuhan subobjektif dengan *constraint* pada desain CDN *testbed*

Subobjektif dan Constraint	Cara Pengukuran	Syarat Pemenuhan
<b>Subobjektif:</b> Alokasi penyimpanan yang baik	Accuracy testing	<i>Cache miss ratio</i> berdasarkan 1-5% (referensi: ( <i>What Is a Cache Hit Ratio?</i> ,

<b>Constraint:</b> Rasio <i>cache hit</i> harus mencapai minimal 80%		n.d.) <a href="https://www.cloudflare.com/learning/cdn/what-is-a-cache-hit-ratio/">https://www.cloudflare.com/learning/cdn/what-is-a-cache-hit-ratio/</a> )
<b>Subobjektif:</b> Berkapasitas tinggi  <b>Constraint:</b> Sistem harus mampu menangani permintaan dengan 95% dari permintaan tersebut (p95) memiliki latensi di bawah 200 milidetik.	Load testing	(Faradilla A, 2023)Latensi tidak melebihi 100 ms untuk konten web (referensi: <a href="https://web.dev/articles/rail">https://web.dev/articles/rail</a> )

### 2.3.1.2. Spesifikasi Fungsional

1. Subsistem antarmuka
2. Subsistem server
3. Subsistem pembelajaran mesin

Tabel II-4 Spesifikasi fungsi pada desain CDN *testbed*

Fungsi	Persyaratan Fungsional	Spesifikasi
Subsistem antarmuka	<ul style="list-style-type: none"> <li>• Menampilkan metrik performa sistem dan log secara visual</li> </ul>	<ul style="list-style-type: none"> <li>• Tampilan metrik server yang berupa utilitas CPU, RAM dan <i>Storage</i></li> <li>• Memiliki tampilan <i>log</i> dan metrik jaringan seperti latensi dan <i>cache hit ratio</i></li> </ul>
Subsistem server	<ul style="list-style-type: none"> <li>• Menerima dan memproses permintaan pengguna</li> <li>• Mengarahkan permintaan pengguna ke <i>node</i> CDN terdekat</li> <li>• Menyalin konten yang sesuai ke berbagai <i>node</i></li> </ul>	<ul style="list-style-type: none"> <li>• Memproses permintaan pengguna dan sesuai dengan lokasi</li> <li>• Menyalin konten ke semua <i>node</i></li> </ul>

Subsistem pembelajaran mesin	<ul style="list-style-type: none"> <li>Melakukan analisis prediktif terhadap pola permintaan pengguna untuk optimasi <i>cache</i></li> <li>Mengelola siklus hidup konten di dalam <i>cache node</i> (menyimpan, memvalidasi, dan menghapus)</li> </ul>	<ul style="list-style-type: none"> <li>Menganalisa pola permintaan pengguna untuk optimasi penyimpanan</li> <li>Menghapus konten dalam siklus hidup <i>cache</i></li> </ul>

## 2.4 Pengembangan Desain

### 2.4.1. Alternatif Metode Desain

Tabel II-5 Pemetaan persyaratan desain menjadi alternatif metode desain

Fungsi	Relevansi Fungsi		Metode	
	Persyaratan dan Spesifikasi Fungsional	Subobjektif dan Constraint, beserta Syarat Pemenuhanannya	Kategori	Alternatif
Subsistem Antarmuka	<p><b>Rangkuman persyaratan fungsional 4:</b> Menampilkan metrik performa sistem dan log secara visual.</p> <p><b>Rangkuman spesifikasi:</b> Memiliki tampilan metrik utilitas seperti CPU, RAM, dan <i>Storage</i> juga menampilkan metrik jaringan</p>	<p><b>Rangkuman subobjektif 4, 5:</b> Pemantauan performa <i>node</i> dan visualisasi data metrik jaringan</p> <p><b>Rangkuman syarat pemenuhan:</b> Selisih nilai yang ditampilkan Grafana dengan nilai aktual pada server tidak melebihi 10%.</p>	<i>Dashboard</i> monitoring	1. Grafana 2. Datadog

Subsistem Server	<p><b>Rangkuman persyaratan fungsional 2,3,5:</b></p> <p>Menerima dan memproses permintaan pengguna, menyalin konten yang sesuai ke berbagai <i>node</i> dan mengarahkan permintaan pengguna ke <i>node</i> CDN terdekat</p> <p><b>Rangkuman spesifikasi:</b></p> <p>Memproses permintaan pengguna dan sesuai dengan Lokasi dan menyalin konten ke semua <i>node</i></p>	<p><b>Rangkuman subobjektif 1,3:</b></p> <p>Pendekatan konten ke <i>node</i> yang dekat pengguna dan berkapasitas tinggi</p> <p><b>Rangkuman syarat pemenuhan:</b></p> <p>Latensi tidak melebihi 100 ms</p> <p>Throughput minimum 100 TPS</p>	Cache Server	1.NGINX 2.Varnish Cache
Subsistem Pembelajaran Mesin	<p><b>Rangkuman persyaratan fungsional 1 dan 6:</b></p> <p>Melakukan analisis prediktif terhadap pola permintaan pengguna untuk optimasi <i>cache</i> dan mengelola siklus hidup konten di dalam <i>cache node</i></p>	<p><b>Rangkuman subobjektif 2:</b></p> <p>Alokasi penyimpanan yang baik</p> <p><b>Rangkuman syarat pemenuhan:</b></p> <p><i>Cache miss ratio</i> berdasarkan 1-5%</p>	Algoritma komputasi	1. Teknik LSTM 2. Teknik LRB

	<p><b>Rangkuman spesifikasi:</b></p> <p>Menganalisa pola permintaan pengguna untuk optimasi penyimpanan dan menghapus konten dalam siklus hidup <i>cache</i></p>			
--	--	--	--	--

#### 2.4.1.1. Subsistem Antarmuka

##### Alternatif framework 1: Grafana

Grafana adalah platform *open source* yang dikembangkan oleh Grafana Labs untuk monitoring dan visualisasi data yang interaktif. Grafana memungkinkan pengguna untuk *query*, visualisasi, melihat metrik, *log*, dan jejak dari berbagai sumber data [1].

Keunggulan Grafana adalah:

1. Visualisasi data yang beragam dan interaktif, dengan pengguna yang dapat membuat berbagai macam visualisasi seperti grafik, tabel, dan lain-lainnya dari utilitas server.
2. Sistem *alerting real time*, dengan pengguna dapat mengatur untuk memunculkan peringatan jika CPU atau memori dari server sudah melebihi batas ambang yang telah ditentukan.
3. Grafana memiliki skalabilitas yang baik sehingga pengguna dapat memantau beberapa server dalam satu *dashboard*.

Kekurangan Grafana adalah:

1. Pengaturan Grafana memerlukan konfigurasi yang cukup rumit dan pemahaman yang mendalam sehingga ini akan menyulitkan pengguna baru.
2. Grafana memiliki model lisensi yang mahal untuk fitur *enterprise* dan pengaturan alert yang rumit untuk implementasi yang kompleks

3. Grafana memiliki keterbatasan pada fitur keamanan dan kemampuan reporting yang memerlukan konfigurasi tambahan untuk lingkungan *enterprise*

Dengan kemampuannya untuk menyediakan visualisasi data dengan minim delay dan kustomisasi *dashboard*, Grafana memenuhi persyaratan fungsional 4, yaitu Menampilkan metrik performa sistem dan log secara visual. Grafana juga memungkinkan tampilan metrik utilitas seperti CPU, memori, dan penyimpanan, juga disertai dengan *alerting* yang membantu pemantauan kinerja server agar tetap dalam batas yang aman.

### **Alternatif framework 2: Datadog**

Datadog adalah *platform monitoring* dan *observability* berbasis *cloud* yang dikembangkan oleh Datadog Inc. untuk memantau aplikasi, infrastruktur, dan layanan dalam skala *enterprise*. Datadog menyediakan solusi *monitoring* terintegrasi yang mencakup *metrics*, *logs*, *traces*, dan *synthetic monitoring* dalam satu platform terpusat.

Keunggulan Datadog adalah:

1. Integrasi yang luas dan mudah, dengan Datadog mendukung lebih dari 600+ integrasi *out-of-the-box* termasuk cloud providers (AWS, Azure, GCP), *database*, web server, dan berbagai teknologi modern seperti Kubernetes dan Docker.
2. Machine learning dan AI-powered insights, dengan Datadog menyediakan *anomaly detection* otomatis, *predictive analytics*, dan *intelligent alerting* yang dapat mengurangi *noise* dan memberikan *insight* yang lebih akurat tentang performa sistem.
3. *Unified observability platform*, dengan Datadog menggabungkan *metrics*, *logs*, *traces*, dan *synthetic monitoring* dalam satu *dashboard* yang koheren, memungkinkan *correlation analysis* yang mendalam untuk *troubleshooting* yang lebih efektif.

Kekurangan Datadog adalah:

1. Biaya yang sangat mahal terutama untuk organisasi besar, dengan model *pricing* berdasarkan volume data dan jumlah host yang dapat mengakibatkan *bill shock* ketika traffic atau infrastruktur berkembang pesat.
2. Vendor lock-in yang kuat karena Datadog menggunakan *proprietary* format dan API yang membuat migrasi ke platform lain menjadi sulit dan mahal, serta ketergantungan pada layanan *cloud* mereka.
3. Kompleksitas konfigurasi untuk *use case advanced* dan *learning curve yang curam* untuk memaksimalkan fitur-fitur canggih seperti *custom metrics*, *advanced dashboards*, dan *correlation analysis* yang memerlukan keahlian khusus.

#### **2.4.1.2. Subsistem Server**

##### **Alternatif Cache Server 1: NGINX**

NGINX (*engine x*) adalah HTTP *web server*, *reverse proxy*, *content cache*, *load balancer*, TCP/UDP *proxy server*, dan *mail proxy*. NGINX terkenal dengan kemampuan fleksibilitas dan berperforma tinggi dengan utilisasi sumber daya yang rendah.

Keuntungan nginx:

1. Web server yang serba bisa: Seperti yang dijelaskan di awal, nginx merupakan web server komplit yang dapat meng-*handle* permintaan HTTP/HTTPS.
2. Penyediaan konten statis yang baik: nginx efisien dalam menyediakan konten *cache* statis seperti gambar, CSS dan JavaScript.
3. Konfigurasi fleksibel: nginx menawarkan kontrol granular untuk *caching policy* yang dapat diatur pada file konfigurasi

Kekurangan nginx:

1. NGINX tidak memiliki kemampuan pemrosesan konten dinamis bawaan dan harus menggunakan prosesor eksternal seperti FastCGI.
2. Manipulasi dan analisis *cache* yang terbatas: Setelah konten di-*cache*, nginx tidak menyediakan alat untuk memanipulasi atau menganalisa *cache*.
3. Keterbatasan dalam *debugging* dan *monitoring cache*

Dengan sifatnya yang menyediakan banyak layanan web server, dan menyediakan konfigurasi *caching* terutama untuk konten statis, nginx memenuhi subobjektif 1 dan memenuhi persyaratan fungsional 2 dan 3.

### **Alternatif Cache Server 2: Varnish Cache**

Varnish cache adalah aplikasi akselerator web yang diketahui juga sebagai HTTP *reverse proxy*. Aplikasi ini dipasang di depan server menggunakan protokol HTTP dan dikonfigurasi untuk menyimpan konten dalam bentuk *cache*. Varnish Cache memiliki fleksibilitas pada konfiugrasinya dengan menggunakan bahasa VCL. VCL memungkinkan adanya peraturan dari permintaan pengguna yang masuk, seperti penentuan konten yang ingin dikirimkan [3].

Kelebihan Varnish Cache:

1. Fungsi utama *caching*: Varnish dirancang khusus sebagai *server caching*, menawarkan performa dan fitur *caching* yang lebih baik. Varnish mendukung fitur bawaan seperti *cache invalidation*, *menghapus cache*, dan manipulasi *cache*.
2. VCL: Varnish Caching Language menyediakan cara yang fleksibel untuk mendefinisikan logika caching. Aturan caching khusus dapat ditulis yang mengintegrasikan prediksi dengan pembelajaran mesin.
3. Analitik *cache* yang kaya fitur: Varnish menyediakan statistik dan *monitoring* yang terperinci secara bawaan yang membantu menganalisis performa *cache* dan mengoptimalkan algoritma *caching*.

Kekurangan Varnish Cache:

1. Membutuhkan sumber daya tinggi: Varnish dapat lebih intensif sumber daya dibandingkan NGINX, terutama dalam penggunaan memori.
2. Kompleksitas dalam set up dan manajemen: Varnish dengan pengaturan caching tertentu bisa menjadi lebih rumit, memerlukan keahlian dalam VCL dan kemungkinan modul kustom untuk mengimplementasikan algoritma prediksi *caching*.
3. Keterbatasan sebagai web server lengkap: Varnish adalah *reverse proxy cache* murni dan tidak dapat berfungsi sebagai web server yang berdiri sendiri seperti NGINX atau Apache. Untuk implementasi dalam skala

*production*, Varnish memerlukan *backend* web server tambahan, sehingga menambah kompleksitas arsitektur dan *overhead* infrastruktur.

Dengan sifatnya yang dikhususkan untuk layanan *caching* dan memiliki kelebihan dalam manipulasi *cache*, Varnish Cache memenuhi subobjektif 1, dan persyaratan fungsional 2 dan 3. Tidak adanya fitur enkripsi SSL/TLS membuat Varnish Cache membutuhkan modifikasi tambahan berupa nginx untuk menghandle trafik yang dienkripsi.

#### 2.4.1.3. Subsistem Pembelajaran Mesin

##### Alternatif Algoritma Komputasi 1: Teknik LSTM

LSTM adalah salah satu tipe RNN yang dapat mengatasi dan mempelajari dependensi jangka panjang pada data sekuensial. LSTM juga dapat memproses dan menganalisis data sekuensial seperti deret waktu, teks, dan data ucapan. Algoritma ini menggunakan sel memori dan gerbang untuk mengontrol laju informasi sehingga informasi yang diproses dapat diseleksi dan ditolak sesuai yang dibutuhkan. LSTM umum digunakan di beberapa aplikasi seperti NLP, *speech recognition*, dan *time series forecasting* [4].

Keunggulan teknik LSTM adalah:

1. LSTM efektif untuk menangkap pola atau ketergantungan jangka panjang dalam data sekuensial
2. Adanya memori jangka panjang dan pendek sehingga teknik ini dapat memprediksi berdasarkan pola historis dan tren.
3. Teknik ini cocok digunakan untuk tipe data yang berurutan seperti *log* dari trafik atau *request* dari pengguna.
4. LSTM dapat mencegah *overfitting* pada data kompleks dan besar dengan kemampuan untuk belajar dari data sekuensial yang panjang.

Kekurangan teknik LSTM adalah:

1. Kompleksitas yang tinggi karena memerlukan komputasi yang lebih besar dibanding model lain yang berbasis regresi sederhana atau ARIMA, khususnya untuk dataset yang besar.
2. Memerlukan data pelatihan yang banyak agar teknik LSTM dapat berjalan secara optimal yang tidak selalu tersedia di CDN.

3. Banyak pengaturan *hyperparameter* yang harus dilakukan untuk mencapai performa yang optimal.

Algoritma LSTM dirancang untuk memahami pola urutan data yang kompleks, menjadikannya pilihan yang sangat baik untuk predictive caching di Content Delivery Network dengan pola akses yang dinamis. Dengan mempelajari hubungan jangka panjang dalam data historis, LSTM dapat menghasilkan prediksi yang sangat akurat mengenai permintaan konten mendatang. Model ini bekerja dengan mengolah data historis yang terstruktur dalam urutan waktu untuk memprediksi konten yang akan diakses berikutnya, sehingga memungkinkan pre-caching yang lebih efektif. Untuk menjaga nilai MAPE tidak lebih dari 5%, model LSTM memerlukan proses pelatihan yang cermat, termasuk normalisasi data, pemilihan fitur yang tepat, serta tuning hyperparameter seperti learning rate dan jumlah unit dalam lapisan LSTM. Meski memerlukan lebih banyak sumber daya komputasi, LSTM sangat cocok untuk aplikasi dengan pola akses yang kompleks dan volume data yang besar

### **Alternatif Algoritma Komputasi 2: Teknik LRB**

Teknik LRB adalah algoritma machine learning yang didesain untuk meningkatkan performa caching di CDN. algoritma ini melakukan pendekatan menggunakan Belady Algorithm yang mana dapat meminimalkan cache miss menggunakan teknik machine learning. Ide utama dari teknik ini adalah menggunakan data historis untuk memprediksi konten yang akan di-request selanjutnya oleh pengguna yang nantinya akan mengurangi jumlah cache miss dan meningkatkan performa secara keseluruhan [5].

Keunggulan teknik LRB:

1. Pendekatan mendekati algoritma optimal LRB dirancang untuk meniru algoritma Belady MIN, yang secara teoretis merupakan algoritma penggantian cache paling efisien. Dengan menggunakan pembelajaran mesin, LRB berupaya mendekati kinerja optimal ini tanpa memerlukan pengetahuan sempurna tentang permintaan di masa depan.
2. Pengurangan lalu lintas WAN implementasi LRB dalam simulasi dengan enam jejak produksi CDN menunjukkan pengurangan lalu lintas WAN sebesar 4–25% dibandingkan dengan desain cache CDN produksi tipikal.

Hal ini menunjukkan efisiensi LRB dalam mengurangi kebutuhan pengambilan konten dari server asal.

3. Adaptabilitas terhadap pola akses yang beragam dengan memanfaatkan data historis dan pembelajaran mesin, LRB dapat menyesuaikan keputusan caching berdasarkan pola akses yang berubah-ubah, meningkatkan efisiensi cache dalam berbagai skenario penggunaan.

Kekurangan teknik LRB:

1. Kebutuhan data pelatihan yang signifikan untuk mencapai prediksi yang akurat, LRB memerlukan sejumlah besar data historis untuk melatih model pembelajaran mesinnya. Pengumpulan dan pemrosesan data ini dapat menjadi tantangan dalam lingkungan dengan sumber daya terbatas.
2. Overhead komputasi meskipun LRB dirancang untuk efisien, integrasi pembelajaran mesin dalam proses penggantian cache menambah beban komputasi. Hal ini dapat mempengaruhi kinerja sistem, terutama jika sumber daya komputasi tidak memadai.
3. Kompleksitas implementasi dengan penerapan LRB memerlukan integrasi antara algoritma caching tradisional dengan model pembelajaran mesin, yang dapat meningkatkan kompleksitas sistem dan memerlukan keahlian khusus dalam pengembangan dan pemeliharaan.

Algoritma LRB dapat memenuhi persyaratan untuk menjalankan model machine learning predictive caching dengan menghasilkan prediksi konten untuk pre-caching, asalkan pola akses historis dianalisis secara efektif. LRB menggunakan pendekatan berbasis aturan untuk memprioritaskan konten yang akan disimpan di cache dengan mempertimbangkan prediksi probabilitas akses. Dengan integrasi model pembelajaran mesin sederhana, seperti regresi linear atau pohon keputusan, LRB dapat meningkatkan akurasi prediksinya. Untuk menjaga nilai MAPE tidak lebih dari 5%, fitur historis yang relevan harus dirancang dengan baik, dan evaluasi kinerja prediksi dilakukan secara rutin untuk memastikan keakuratan dalam pemilihan konten yang di-cache. Selain itu, karena kompleksitasnya yang lebih rendah dibandingkan dengan model seperti LSTM, LRB lebih efisien untuk implementasi pada sistem dengan sumber daya terbatas.

#### **2.4.2. Sistem Terintegrasi**

Tentukan kombinasi alternatif dari tiap subsistem, sehingga menjadi alternatif desain sistem terintegrasi, seperti terlihat pada Tabel 2.

Tabel II-6 Kombinasi alternatif desain sistem terintegrasi

No.	Alternatif Desain Sistem Terintegrasi
1.	Grafana-NGINX-LSTM
2.	Grafana-NGINX-LRB
3.	Grafana -Varnish Cache-LSTM
4.	Grafana -Varnish Cache-LRB
5.	Datadog -NGINX-LSTM
6.	Datadog -NGINX-LRB
7.	Datadog-Varnish Cache-LSTM
8.	Datadog-Varnish Cache-LRB

#### **2.4.3. Penentuan Solusi Desain Terbaik**

##### **2.4.3.1. Pembobotan Kategori dari Alternatif Metode Desain**

Dalam skala 5, dasbor *monitoring* diberi bobot 1 karena tidak mempengaruhi langsung terhadap performa dari CDN, jika terjadi kesalahan hanya akan berefek pada tampilan dari utilisasi server dll. Tentu saja tampilan yang baik dan mudah dimengerti tetap diperlukan jika memungkinkan.

Penentuan *cache server* yang sesuai juga sangat menentukan performa dan kompleksitas dari CDN karena merupakan komponen inti, sehingga diberikan bobot 3. Penentuan dan konfigurasi *cache server* yang baik akan berdampak pada kesesuaian sistem dengan yang diharapkan.

Algoritma komputasi juga memiliki peran penting untuk menentukan akurasi dari *caching* dan performa sehingga diberikan bobot 2. Algoritma komputasi yang efisien akan membantu meningkatkan kesesuaian sistem dengan yang diharapkan.

Tabel II-7 Pembobotan pada kategori dari alternatif metode desain

Kategori	Bobot
Dashboard monitoring	1
Cache server	3
Algoritma komputasi	3

#### 2.4.3.2. Matriks Keputusan

Dari berbagai alternatif desain sistem terintegrasi yang diperlihatkan pada Tabel 2, kita perlu memutuskan metode desain yang terbaik. Untuk itu, perlu dibuat matriks keputusan, dimana setiap alternatif desain diberikan bobot nilai antara 1 – 5. Nilai tersebut lalu dikalikan dengan pembobotan kategori yang telah ditentukan sebelumnya pada Tabel 3.

Matriks keputusan dari setiap alternatif desain sistem terintegrasi diperlihatkan pada Tabel 4.

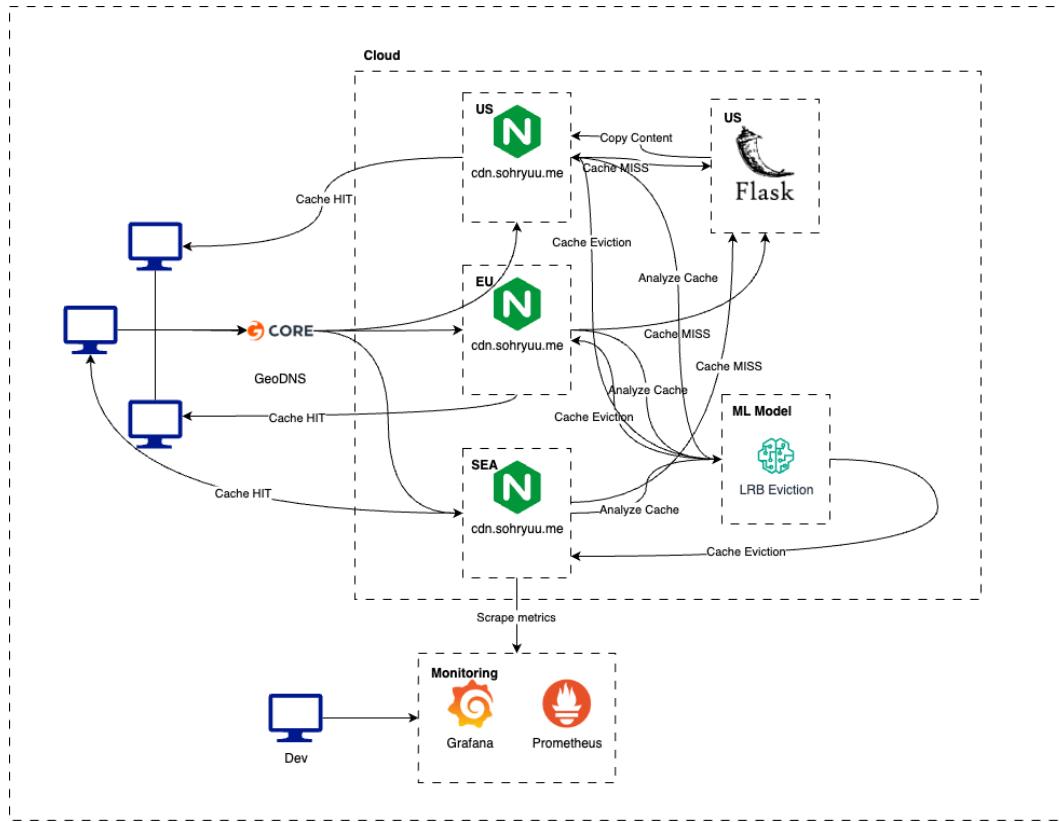
Tabel II-8 Matriks keputusan alternatif desain sistem terintegrasi

Kategori		Alter natif 1	Alter natif 2	Alter natif 3	Alter natif 4	Alter natif 5	Alter natif 6	Alter natif 7	Alter natif 8
Nama	Bob ot								
Dashboard monitoring	1	3	3	3	3	2	2	2	2
Cache server	3	4	4	3	3	4	4	3	3
Algoritma komputasi	3	3	4	3	4	3	4	3	4
<b>Total Bobot</b>		<b>24</b>	<b>27</b>	<b>21</b>	<b>24</b>	<b>23</b>	<b>26</b>	<b>20</b>	<b>23</b>

Dari hasil penilaian setiap alternatif sistem terintegrasi, alternatif desain yang dipilih adalah alternatif ke-2 dengan kombinasi Grafana-NGINX-LRB.

## BAB III DETIL DESAIN DAN IMPLEMENTASI

### 3.1 Model Desain



Gambar III-1 Gambar arsitektur diagram dan alur sistem CDN

Gambar III-1 merupakan diagram desain sistem dari CDN Testbed yang terdiri dari 3 sistem yaitu sistem antarmuka, sistem server dan sistem pembelajaran mesin. Sistem antarmuka menggunakan Prometheus yang akan mengambil metrik dan log dari server NGINX yang kemudian akan divisualisasikan oleh Grafana. Log yang akan dimonitor adalah ketersediaan (*availability*), *throughput*, latensi dan Cache Hit Ratio (CHR) juga untuk metriknya berupa CPU, RAM dan penyimpanan untuk masing-masing CDN. Subsistem server menggunakan Gcore yang menjadi GeoDNS, merutekan permintaan pengguna ke letak server CDN yang tersedia. Server CDN menggunakan NGINX yang di-deploy pada *virtual machine* di *public cloud* yang berbeda region. Sedangkan aplikasi utamanya menggunakan Flask yang berbasis bahasa pemrograman Python yang juga di-deploy pada *public cloud* region Amerika. Konten-konten pada aplikasi utama tersimpan pada *bucket storage* di

*public cloud*. Untuk sistem pembelajaran mesinnya sendiri di-deploy pada VM di Google Cloud yang akan terintegrasi dengan NGINX untuk menganalisa *cache*.

Mekanisme kerjanya secara umum adalah sebagai berikut; Pertama, pengguna melakukan permintaan HTTPS ke aplikasi utama yang akan direspon dengan tampilan antarmuka website. Kemudian, ketika ingin mengambil konten, yang biasa berupa gambar/video, permintaan dari pengguna akan mengarah ke GeoDNS terlebih dahulu untuk menentukan server CDN mana yang paling dekat dengan pengguna. Server CDN akan mengecek terlebih dahulu apakah konten yang diminta tersedia di *cache* atau tidak. Jika iya, maka konten yang ada di CDN tersebut akan langsung diteruskan ke pengguna. Jika tidak, CDN akan meminta konten ke *storage bucket*, melakukan *caching* untuk konten tersebut, lalu mengirimkannya ke pengguna. Setiap aktivitas akses ke server NGINX akan dipantau dan tersimpan pada suatu file log bernama *access.log*. Log ini akan diekspor oleh Prometheus yang kemudian akan divisualisasi oleh Grafana. Tim pengembang dapat melihat secara langsung metrik dan log server yang disimpan oleh server seperti berapa banyak koneksi aktif, latensi pengguna dan apakah permintaan pengguna direspon dengan *cache* atau tidak. *Cache* yang tersimpan kemudian diekspor dan diolah oleh model pembelajaran mesin untuk penentuan *cache eviction*. Hasil dari *machine learning* akan menentukan *cache* mana yang akan dihapus.

Dalam pelaksanaan tugas akhir ini, secara garis besar terdiri dari 3 subsistem yang dikembangkan secara berkelompok. Masing-masing sistem terintegrasi yang menjadi suatu sistem yang utuh. Rincian subsistem dan pembagian kerja untuk setiap anggota kelompok adalah sebagai berikut:

Tabel III-1 Desain rinci subsistem dan pembagian kerja

Subsistem	Rincian Subsistem	Pembagian Kerja
Subsistem Antarmuka	Mengkonfigurasi Prometheus dan Grafana, integrasi server dengan	Akbar Febry Wahyu Andrian

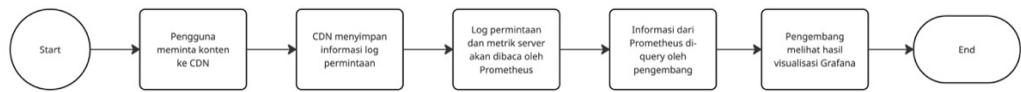
	<i>monitoring, test dan debugging</i>	
Subsistem Server	Merancang arsitektur, mengkonfigurasi NGINX dan GeoDNS, membuat <i>dummy</i> website, <i>test</i> dan <i>debugging</i>	Akbar Febry Wahyu Andrian
Subsistem Pembelajaran Mesin	Melatih model pembelajaran mesin, integrasi dengan sistem CDN, <i>test</i> dan <i>debugging</i> ,	Hanif Al Falih

### 3.1.1 Desain Subsistem Pemantauan

Tampilan antarmuka dari *monitoring* oleh Grafana berupa beberapa grafik dari parameter yang diukur terhadap waktu. Untuk hal yang akan dilihat dipantau adalah

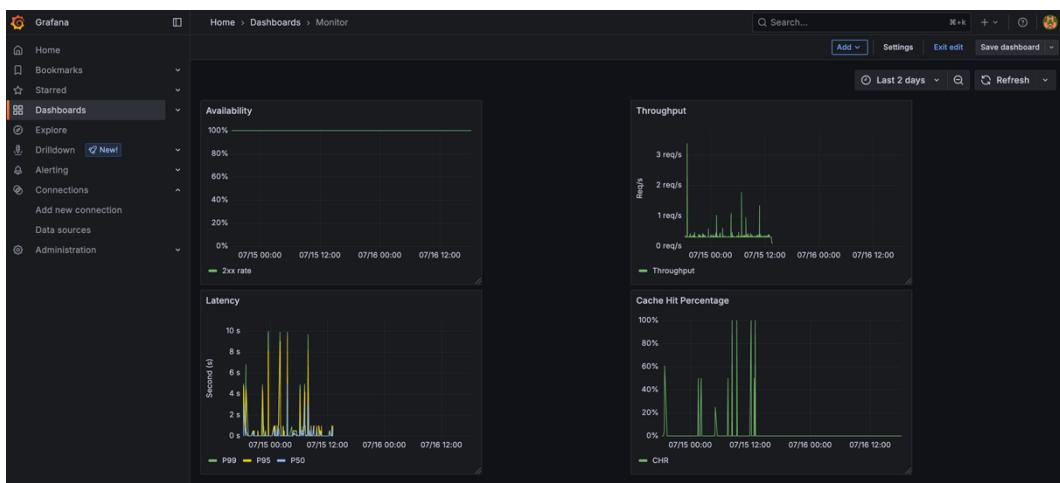
1. latensi akses web;
2. *throughput*;
3. Cache Hit Ratio (CHR);
4. ketersediaan;
5. CPU;
6. RAM;
7. dan penyimpanan.

Hal yang akan dipantau tidak terbatas pada yang sudah disebutkan di atas karena seiring berjalannya proyek, jika ada kebutuhan baru tertentu maka akan bertambah. Sebagai tambahan, Grafana juga menyediakan fitur “Import” yang memungkinkan pengguna untuk mengimport *template dashboard* yang orang lain buat. Hal ini memungkinkan hasil implementasi yang tidak sesuai dengan desain awal. Namun, tetap dengan fungsionalitas yang sama atau lebih.



Gambar III-2 Alur kerja dari sistem pemantauan

Gambar III-2 menunjukkan desain alur kerja dari sistem pemantauan, dimulai dari pengguna yang meminta konten ke CDN, informasi mengenai permintaan tersebut akan disimpan pada suatu file log. Log tersebut kemudian akan diekspor oleh Prometheus. Selain log, Prometheus akan mengekspor data-data metrik seperti penggunaan CPU, RAM dan penyimpanan yang dipakai oleh server. Data-data tersebut dapat di-*query* dan divisualisasikan dengan menggunakan Grafana. Setelah dilakukan pengaturan terhadap tampilan dari Grafana, selebihnya tim pengembang hanya memantau info dari sistem.

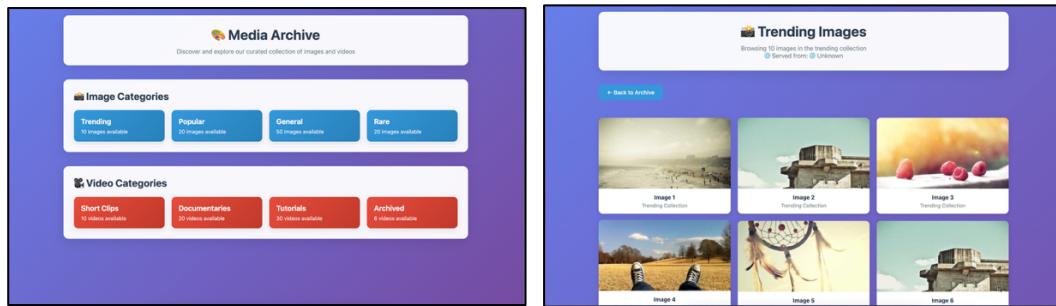


Gambar III-3 Desain tampilan sederhana pada Grafana

Gambar III-3 menunjukkan desain tampilan sederhana dari Grafana yang merupakan subsistem antarmuka untuk memonitor performa dari subsistem server. Desain seperti ini dipilih karena bersifat sederhana dan mudah diimplementasikan, namun tetap memberikan informasi yang relevan.

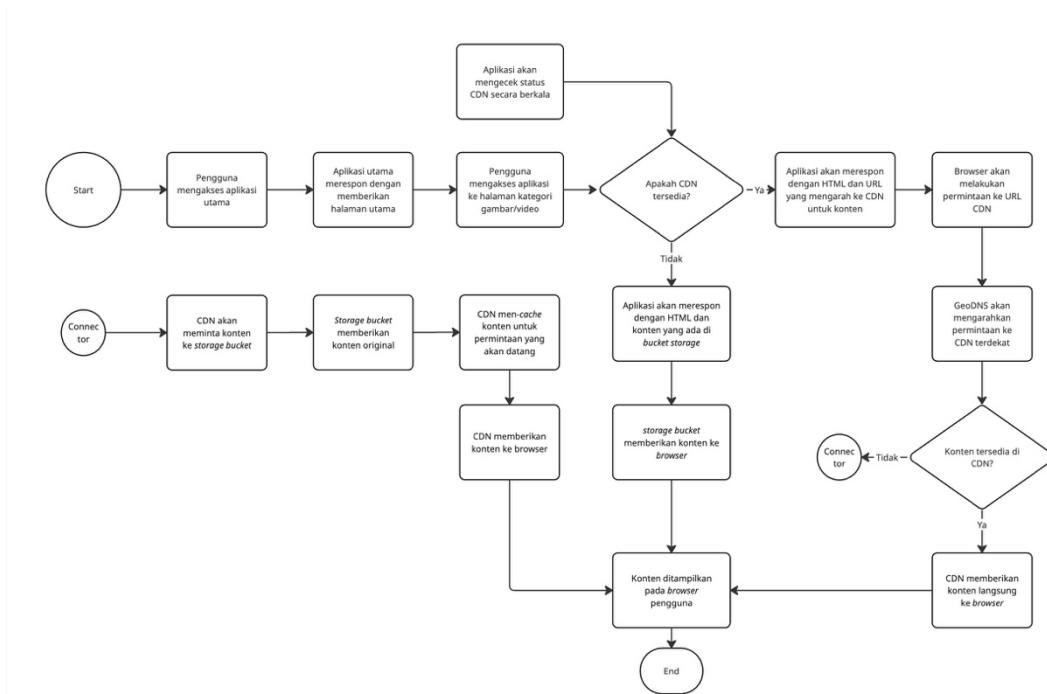
### 3.1.2 Desain Subsistem Server

Subsistem server memiliki beberapa bagian yaitu, aplikasi web yang berbasis Flask, *cache* server dengan NGINX dan GeoDNS yang menggunakan Gcore. Aplikasi web *dummy* memiliki desain sebagai berikut



Gambar III-4 (a) Desain tampilan utama yang sederhana pada website *dummy* (b) Desain tampilan salah satu kategori gambar sederhana pada website *dummy*

Desain tersebut digunakan karena sistem hanya memerlukan suatu aplikasi sederhana yang menyajikan konten berupa gambar dan video sehingga fitur canggih dan tampilan yang estetik bukanlah prioritas. Karena tidak memiliki tampilan, untuk nginx sebagai server CDN dan Gcore GeoDNS tidak memiliki desain tertentu.



Gambar III-5 Alur kerja dari subsistem server

Gambar III-5 menunjukkan rancangan alur kerja untuk subsistem server. Dimulai dari pengguna yang mengakses aplikasi Flask, kemudian meminta konten yang ada. Sebelumnya, aplikasi akan mengecek ketersediaan dari CDN, jika tidak ada maka konten akan dikirimkan dari *bucket storage* yang merupakan letak asli dari konten gambar dan video. Hal ini diperlukan karena jika CDN server sedang gada gangguan atau perbaikan, aplikasi tetap dapat melayani pengguna. Lalu, jika CDN memang tersedia, aplikasi akan memberikan URL ke CDN. *Browser* yang menerima respon tersebut akan melakukan permintaan ke URL CDN yang akan dirutekan oleh GeoDNS ke CDN server terdekat. Jika CDN memiliki konten tersebut, maka konten akan langsung dikirimkan sedangkan jika tidak tersedia, CDN akan melakukan *request* ke *bucket storage*. Konten dari *bucket storage* akan *di-cache* dan dikirimkan ke pengguna.

### 3.2 Implementasi

Implementasi terhadap desain sistem yang sudah dirancang dilakukan. Lebih lanjut, konsep pengujian juga didefinisikan untuk memudahkan pengujian sistem dan penentuan pemenuhan objektif, subobjektif dan persyaratan fungsionalnya.

#### 3.2.1 Konsep Pengujian Sistem

Pengujian yang akan dilakukan untuk subsistem pemantauan berupa kesesuaian hasil pemantauan terhadap *testing*. Untuk subsistem server dilakukan *user testing* secara manual dan *load testing* untuk melihat apakah permintaan direspon dengan baik, kesesuaian letak server yang melayani terhadap lokasi pengguna, jumlah *throughput*, dan latensi pengguna. Untuk lebih jelasnya, berikut adalah fungsi dan deskripsi yang harus dipenuhi

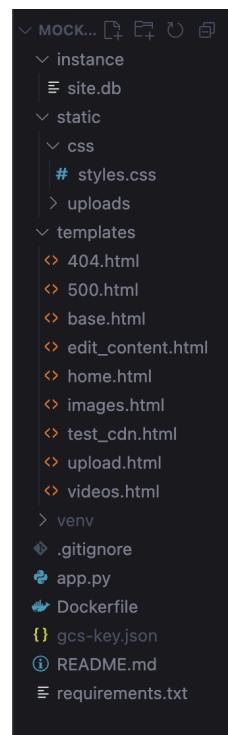
Tabel III-2 Spesifikasi fungsi dan hasil yang diharapkan ketika pengujian

Fungsi	Deskripsi

Menampilkan metrik	Sistem pemantauan mampu menampilkan hasil yang sesuai dengan pengetesan
Caching konten	CDN mampu memberikan cache status “HIT” ketika dilakukan <i>request</i>
Perutean permintaan pengguna	GeoDNS mampu merutekan permintaan pengguna ke server dengan jarak terdekat
Menerima dan merespon permintaan pengguna	Aplikasi web mampu memberikan respon yang sesuai terhadap permintaan pengguna

### 3.2.1 Implementasi Aplikasi Web

Flask adalah sebuah framework *back-end* dengan bahasa Python. Fungsinya adalah untuk merutekan permintaan pengguna ke *endpoint* yang sesuai dan memastikan permintaan tersebut direspon dengan baik. Untuk proyek ini, struktur file dari aplikasi adalah sebagai berikut



Gambar III-6 Struktur folder pada aplikasi Flask

File utama dari flask adalah app.py yang berisikan *logic* dari aplikasi. Untuk antarmuka/*front-end* dari aplikasi hanya menggunakan *template* file HTML biasa yang ditambah styling dengan CSS. Terdapat SQLite database untuk mendukung fitur Create, Read, Update, and Delete (CRUD). Dockerfile digunakan untuk CI/CD karena aplikasi di-deploy pada Cloud Run. Requirements.txt berisikan *dependency* proyek yang perlu diinstal.

```
@app.route("/")
def home():
    try:
        response = make_response(render_template('home.html',
                                                image_groups=IMAGE_GROUPS,
                                                video_groups=VIDEO_GROUPS))
        return add_cache_headers(response, 'page')
    except Exception as e:
        logger.error(f"Error in home route: {e}")
        return "Internal server error", 500
```

Gambar III-7 Program respon untuk permintaan ke *endpoint root* pada app.py

Berikut adalah *Flask route decorator* yang memberitahu aplikasi untuk menjalankan program di bawahnya ketika ada permintaan ke *endpoint root* atau ke *endpoint* utama. Dimulai dengan menggunakan *try block* untuk *catch* dan mengatasi jika ada error (*exceptions*) yang muncul. Penggunaan *try block* bermanfaat untuk mencegah server mengalami *crash* jika ada *error*. Jika tidak ada masalah, server akan mengirimkan *template* home.html yang akan menampilkan tampilan utama dari aplikasi dengan tambahan respon *cache headers*. *Cache headers* ditambahkan oleh fungsi *add\_cache\_headers* yang berfungsi untuk menambahkan *headers* pada respon agar konten dapat di-*cache*, aman dan teroptimasi untuk browser dan CDN.

```

@app.route("/images/<group>")
def show_images(group):
    try:
        if group not in IMAGE_GROUPS:
            logger.warning(f"Invalid image group requested: {group}")
            return "Invalid image group", 404

        cdn_region = request.headers.get('X-CDN-Region', 'Origin')
        if 'EU' in cdn_region:
            cdn_flag = '🇪🇺'
        elif 'Asia' in cdn_region:
            cdn_flag = '🇯🇵'
        elif 'US' in cdn_region:
            cdn_flag = '🇺🇸'
        else:
            cdn_flag = '🌐'

        start, end = IMAGE_GROUPS[group]
        IMAGE_BASE = "https://storage.googleapis.com/bucket-main-ta/static/images/image_{}.jpg"
        urls = make_urls(IMAGE_BASE, start, end)

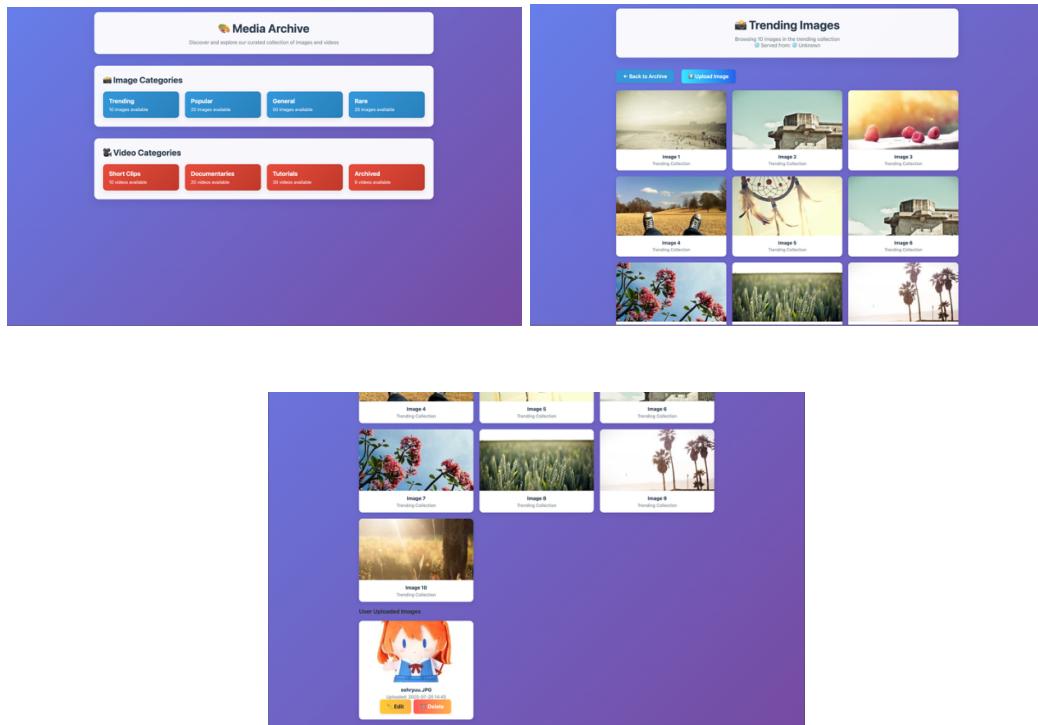
        # Safely query the database
        try:
            user_images = UserContent.query.filter_by(content_type='image', group=group).order_by(UserContent.uploaded_at.desc()).all()
        except Exception as e:
            logger.error(f"Database error in show_images: {e}")
            user_images = []

        response = make_response(render_template('images.html',
                                                group=group,
                                                urls=urls,
                                                CDN_REGION=cdn_region,
                                                CDN_FLAG=cdn_flag,
                                                CDN_ACTIVE=CDN_AVAILABLE,
                                                user_images=user_images))
        return add_cache_headers(response, 'page')
    except Exception as e:
        logger.error(f"Error in show_images route: {e}")
        return "Internal server error", 500

```

Gambar III-8 Program untuk merespon permintaan ke *endpoint* /images pada app.py

Gambar III-8 adalah program yang meng-*handle* permintaan untuk melihat konten yang berupa gambar, memiliki struktur dan logika yang sama dengan program yang meng-*handle* permintaan untuk melihat video. Secara umum, program mengakses konten secara langsung dari *bucket storage*, atau dari CDN jika tersedia dengan mengubah URL yang diberikan ke *browser* pengguna. Tambahan, aplikasi akan men-*query* konten dari *database* untuk konten yang diunggah oleh pengguna. Diatur pula *logic* untuk menampilkan bendera dan nama *region* dari penyedia konten yaitu CDN, yang hanya akan muncul ketika dilakukan permintaan melalui domain CDN itu sendiri, yakni *cdn.sohryuu.me*. Hal ini dikarenakan konten dikirim langsung ke *browser* pengguna sehingga aplikasi tidak tahu CDN mana yang melayani permintaan pengguna. Hasil akhir adalah respon yang berupa halaman HTML yang berisikan URL konten dengan *cache header*.



Gambar III-9 (a) Tampilan antarmuka halaman utama dari aplikasi web (b) Tampilan antarmuka untuk kategori *trending* dari aplikasi web untuk konten asli (c) Tampilan antarmuka untuk kategori *trending* dari aplikasi web untuk konten unggahan pengguna.

Gambar di atas menunjukkan hasil akhir dari aplikasi web yang sudah dihosting pada Cloud Run, suatu layanan dari Google Cloud. Aplikasi dapat diakses pada alamat <https://sohryuu.me>.

### 3.2.2 Implementasi *Cache Server NGINX*

*Cache server* nginx yang diimplementasikan akan di-host pada Virtual Machine di Google Cloud yang berbasis Ubuntu Server 24.04 LTS. Mayoritas konfigurasi dilakukan dengan cara melakukan koneksi SSH ke server. Berikut adalah bagian dari konfigurasi nginx.conf (/etc/nginx/nginx.conf) yang berlaku secara global

```
http {  
    ##  
    # Basic Settings  
    ##  
  
    sendfile on;  
    tcp_nopush on;  
    types_hash_max_size 2048;  
    # server_tokens off;  
  
    # server_names_hash_bucket_size 64;  
    # server_name_in_redirect off;  
  
    # Cache configuration  
    proxy_cache_path /var/cache/nginx/media levels=2:2 keys_zone=media_cache:100m  
        | inactive=7d max_size=100m use_temp_path=off;  
  
    proxy_cache_path /var/cache/nginx/pages levels=1:2 keys_zone=page_cache:10m  
        | inactive=1h max_size=1g use_temp_path=off;  
}
```

Gambar III-10 Cuplikan konfigurasi *nginx.conf* yang mendefinisikan *path cache*

Cuplikan konfigurasi di atas utamanya untuk mendefinisikan *path* dimana *cache* akan disimpan. Untuk sistem ini, dibagi ke dalam dua jenis, yakni *cache* untuk konten yang berupa gambar/video dan *cache* yang berupa halaman web statis yang memiliki *path* dimana *cache*-nya masing-masing. Parameter *levels* mengatur 2 level direktori di bawah */path/to/cache*. Hal ini diatur untuk mencegah adanya terlalu banyak file disimpan pada satu direktori yang sama yang akan memperlambat akses ke file tersebut.

```
sohryuuasuka@cdn-sea:~$ sudo ls -a /var/cache/nginx/media  
.. 19 3e 3f 62 7f 99 a8 d9 ec ff  
sohryuuasuka@cdn-sea:~$ sudo ls -a /var/cache/nginx/pages  
.. 0 1 8 9 b c e f  
sohryuuasuka@cdn-sea:~$
```

Gambar III-11 Contoh struktur direktori *cache* pada NGINX

Untuk memberikan gambaran lebih, gambar di atas adalah contoh dari struktur direktori *cache* pada nginx. NGINX akan menggunakan *hash* dari URL nya sebagai *cache key* untuk membuat nama subdirektori di bawah */path/to/cache*. Angka level menspesifikasikan jumlah karakter *hash* yang digunakan untuk nama subdirektori.

Parameter *keys\_zone* mendefinisikan *shared memory zone* untuk menyimpan *cache keys* dan *metadata*. Secara sederhana dapat dikatakan mengatur besar penyimpanan yang dapat digunakan oleh nginx untuk men-*cache* konten. Jika sudah mencapai

batasan, maka *cache manager* akan menghapus konten dengan algoritma bawaan LRU.

```
##  
# Logging Settings  
##  
  
log_format main '$remote_addr - $remote_user [$time_local] "$request" '  
    '$status $body_bytes_sent "$http_referer" '  
    '"$http_user_agent" "$http_x_forwarded_for" '  
    'cache_status=$upstream_cache_status '  
    'cdn_region=$http_x_cdn_region '  
    'lat=$upstream_response_time '  
    'rt=$request_time';  
  
access_log /var/log/nginx/access.log main;
```

Gambar III-12 Cuplikan konfigurasi *nginx.conf* yang mendefinisikan format *log main*

Gambar di atas menunjukkan cuplikan lain untuk konfigurasi nginx yang mendefinisikan format dari log *main* untuk *access.log* (/var/log/nginx/access.log). File log ini memiliki peranan penting untuk mencatat setiap permintaan yang menuju CDN. Berikut adalah penjelasan singkat mengenai informasi yang disimpan pada file *access.log*

Tabel III-3 Nama dan deskripsi singkat format *access.log*

Variabel	Deskripsi Singkat
\$remote_addr	Alamat IP pengguna
\$remote_user	Username dari <i>authenticated user</i> (jika ada)
\$time_local	Waktu lokal dilakukan permintaan
\$request	<i>HTTP request line</i> (GET /home HTTP1.1)
\$status	Kode status HTTP (200, 304, 404, dll.)
\$body_bytes_sent	Bytes terkirim ke pengguna (body)
\$http_referer	<i>Referrer URL</i>
\$http_user_agent	<i>User agent string</i> ( <i>browser/info pengguna</i> )
\$http_x_forwarded_for	Alamat IP asli (jika dibelakangi <i>proxy</i> )
\$upstream_cache_status	<i>Cache status</i> (HIT, MISS, dll.)
\$http_x_cdn_region	CDN <i>region</i>
\$upstream_response_time	Waktu untuk mendapatkan respon dari <i>upstream server</i>
\$request_time	Total waktu untuk memproses permintaan

Beberapa hal yang perlu diperhatikan adalah *upstream cache status*, *upstream response time* dan *request time*. *Upstream cache status* memberikan informasi bagaimana status dari *cache* konten yang dikirim, *upstream response time* merupakan waktu yang dibutuhkan oleh nginx menunggu respon dari *upstream server* atau aplikasi utama, dan *request time* adalah total waktu yang diperlukan nginx untuk memproses permintaan.

Tabel III-4 Kode HTTP respon dan penjelasan singkat

Kode	Deskripsi Singkat
2xx	Permintaan berhasil
3xx	<i>Redirection request</i>
4xx	<i>Error</i> dari sisi pengguna
5xx	<i>Error</i> dari sisi server

Penjelasan singkat mengenai kode respon HTTP dijelaskan pada Tabel III-4. Kode respon yang berawal 2, seperti 200, 201, 202 berarti permintaan berhasil direspon dengan baik. Kode yang berawal 3 seperti 301, 302 biasanya mengindikasikan kalau permintaan di-*redirect* ke server lain. Kode yang berawalan 4 seperti 400, 401, 403, 404 mengindikasikan kalau terdapat kesalahan pada sisi pengguna sedangkan kode yang berawalan 5 seperti 500, 502 mengindikasikan kalau ada kesalahan pada sisi server.

Selanjutnya adalah mengkonfigurasi blok server untuk CDN. File cdn.conf disimpan pada /etc/nginx/conf.d/cdn.conf. Untuk mengaktifkan file konfigurasi ini, tidak lupa dilakukan *include* pada file nginx.conf. Berikut adalah cuplikan isi dari file konfigurasi CDN

```
server {
    server_name cdn.sohryuu.me;

    # Redirect HTTP to HTTPS

    listen [::]:443 ssl ipv6only=on; # managed by Certbot
    listen 443 ssl; # managed by Certbot
    ssl_certificate /etc/letsencrypt/live/cdn.sohryuu.me/fullchain.pem; # managed by Certbot
    ssl_certificate_key /etc/letsencrypt/live/cdn.sohryuu.me/privkey.pem; # managed by Certbot
    include /etc/letsencrypt/options-ssl-nginx.conf; # managed by Certbot
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by Certbot

    # Security headers
    add_header X-Frame-Options "SAMEORIGIN" always;
    add_header X-Content-Type-Options "nosniff" always;
    add_header X-XSS-Protection "1; mode=block" always;
}
```

Gambar III-13 Cuplikan konfigurasi *cdn.conf* yang mendefinisikan nama server, sertifikat SSL dan HTTP *header* untuk keamanan

Cuplikan konfigurasi di atas secara singkat adalah untuk mengatur nama server (domain) server beserta port nya juga diatur sertifikat SSL agar server dapat berkomunikasi dengan protokol HTTPS dan *security headers*. Sertifikat SSL dapat di-*issue* dengan menggunakan Let's Encrypt yang merupakan suatu proyek non-profit sehingga website dapat menggunakannya secara gratis.

```

server {
    proxy_set_header X-CDN-Region "Southeast-Asia";

    # Compression
    gzip on;
    gzip_vary on;
    gzip_min_length 1024;
    gzip_types text/plain text/css text/xml text/javascript application/javascript \
        | application/xml+rss application/json image/svg+xml;

    # Static assets with long cache
    location ~* ^/static/(images|videos)/ {
        limit_req zone=media_limit burst=20 nodelay;

        # Cache configuration
        proxy_cache media_cache;
        proxy_cache_valid 200 304 30d;
        proxy_cache_valid 404 1m;
        proxy_cache_use_stale error timeout updating http_500 http_502 http_503 http_504;
        proxy_cache_background_update on;
        proxy_cache_lock on;

        set $gcs_path "/bucket-main-ta$request_uri";
        proxy_pass https://gcs_storage$gcs_path;
        proxy_ssl_server_name on;
        proxy_ssl_verify off;

        proxy_set_header Host storage.googleapis.com;
        proxy_set_header Authorization "";
        proxy_set_header X-CDN-Region "Southeast-Asia";

        add_header X-Cache-Status $upstream_cache_status always;
        add_header X-CDN-Region "Southeast-Asia" always;
        add_header X-Served-From "GCS-via-CDN" always;

        expires 30d;
        add_header Cache-Control "public, max-age=2592000";

        proxy_set_header Range $http_range;
        proxy_set_header If-Range $http_if_range;
    }
}

```

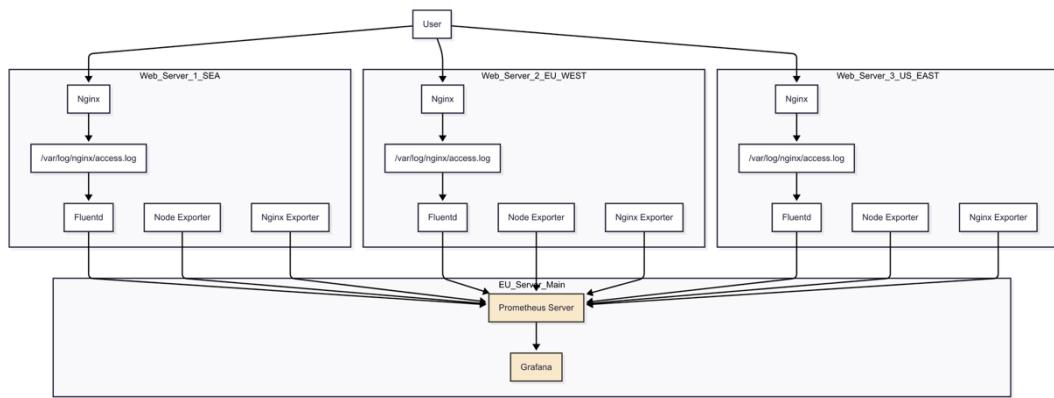
Gambar III-14 Cuplikan konfigurasi *cdn.conf* yang mendefinisikan HTTP *header* untuk *cache*, kompresi konten, dan URL yang di *cache*.

Gambar III-14 pada blok server, diatur *header* berupa X-CDN-Region untuk memberi tahu dari server CDN mana konten diberikan dan dikonfigurasikan kompresi file teks dengan menggunakan gzip. Kemudian diatur pada blok *location* yang berisi media berupa gambar dan video untuk pengaturan *cache*, *proxy* ke *bucket storage* dan penambahan respon *header*. Server akan men-*cache* respon 200 dan 304 selama 30 hari, respon 404 selama 1 menit dan menggunakan *cache* yang tersimpan (*stale*) jika aplikasi utama memberikan respon 5xx. *Proxy\_cache\_background\_update* mengatur ketika ada konten yang *expired* dan ada *request* ke konten tersebut, nginx akan memberikan respon dari *cache* lamanya (*stale*) dan akan mengambil *cache* baru ke backend pada proses *background*. *Proxy\_cache\_lock* berfungsi ketika ada beberapa pengguna yang mengakses suatu konten yang *cache*-nya tidak ada, hanya 1 permintaan pengguna yang akan

mengambil *cache* dari *backend*, selebihnya menunggu permintaan pertama itu di-*cache*. Hal ini mengurangi adanya pekerjaan yang dilakukan berkali-kali dan dapat mengurangi beban server, *backend* sekaligus mengurangi *bandwidth*. Selebihnya konfigurasi mengatur meneruskan permintaan ke *bucket storage* dan penambahan *header* baik ke *bucket storage* maupun ke *browser* pengguna.

### 3.2.2 Implementasi Sistem Pemantauan

Aplikasi pemantauan berupa Prometheus dan Grafana diinstal pada Virtual Machine pada regional Eropa, selebihnya hanya diinstal pengekspornya saja.



Gambar III-15 Alur kerja dari sistem pemantauan

Gambar III-15 menunjukkan alur kerja dari sistem pemantauan yang lebih mendetail, menyertakan layanan yang terlibat. Pertama, melakukan proses instalasi untuk semua *service* terkait, yaitu *nginx-exporter* dan *node-exporter* terlebih dahulu. *NGINX exporter* adalah proyek *official* dari nginx untuk mengekspor metrik-metrik yang ada pada nginx ke Prometheus yang dapat dilihat pada Github <https://github.com/nginx/nginx-prometheus-exporter>. *Node exporter* adalah proyek dari Prometheus untuk dapat mengekspor metrik-metrik pada \*NIX kernel (yang berarti termasuk UNIX kernel pada Linux OS) yang dapat dilihat pada Github [https://github.com/prometheus/node\\_exporter](https://github.com/prometheus/node_exporter). Dari kedua sumber tersebut, terdapat sumber *binaries* dalam bentuk .tar.gz yang dapat diinstal pada server Ubuntu. Setelah diunduh dan memindahkannya ke direktori /usr/local/bin, disiapkan *process*-nya agar bisa berjalan di *background*, mulai ketika awal *boot* dan dapat diatur dengan baik oleh *operating system*.

```

[Unit]
Description=Node Exporter
Wants=network-online.target
After=network-online.target

[Service]
User=node_exporter
Group=node_exporter
Type=simple
ExecStart=/usr/local/bin/node_exporter \
    --web.listen-address=0.0.0.0:9100 \
    --collector.systemd \
    --collector.processes \
    --collector.filesystem.ignored-mount-points=^(sys|proc|dev|host|etc)(\$|/)

Restart=always

[Install]
WantedBy=multi-user.target

```

Gambar III-16 Isi dari *node-exporter.service*

Gambar di atas adalah isi dari *node-exporter.service*, file konfigurasi untuk system, digunakan untuk menjalankan dan mengatur proses. Komponen pentingnya untuk [Service] nya adalah ExecStart yang mendefinisikan perintah untuk menjalankan node-exporter itu sendiri, User/Group yang menjalankan proses tersebut sebagai node\_exporter , Restart=always yang memastikan proses otomatis *restart* ketika terjadi *crash*. Untuk file *nginx-exporter.service* memiliki format yang sama dengan kegunaan yang sama pula.

Tak cukup sampai disitu, karena nginx versi *open-source* memiliki keterbatasan dalam pengeksposan metriknya, maka dibutuhkan pengekspor tambahan untuk melihat informasi dari log nya. Digunakan Fluentd, suatu *open source data collector* berbasis Ruby dan C. Dengan adanya Fluentd, access.log yang berisikan *upstream cache status, delay*, dan data-data penting lainnya pada log dapat diekspor dan diolah.

```

<filter nginx.access>
  <metric>
    name nginx_http_status_code_total
    type counter
    desc Total HTTP requests by status code
    <labels>
      method ${method}
      status ${status}
      path ${path}
    </labels>
  </metric>

  <metric>
    name nginx_cache_status_total
    type counter
    desc Total requests by cache status
    <labels>
      method ${method}
      status ${status}
      cache_status ${cache_status}
      cdn_region ${cdn_region}
    </labels>
  </metric>

  <metric>
    name nginx_cache_requests_by_status
    type counter
    desc Cache requests grouped by status for ratio calculations
    <labels>
      cache_status ${cache_status}
      cdn_region ${cdn_region}
      method ${method}
    </labels>
  </metric>
</filter>

```

Gambar III-17 Cuplikan konfigurasi *fluent.conf*

Dari cuplikan konfigurasi di atas, Fluentd dapat mengekspor *nginx\_cache\_status\_total*, *nginx\_cache\_request\_by\_status* dan *nginx\_http\_status\_code\_total* yang diambil dari file access.log. Seperti *service* sebelumnya, Fluentd memiliki prosesnya sendiri yang berjalan di *background* dengan *systemd*.

Langkah selanjutnya adalah menginstall Prometheus dan mengkonfigurasikannya pada /etc/prometheus/prometheus.yml.

```

# my global config
global:
  scrape_interval: 15s
  evaluation_interval: 15s

rule_files:
  # - "first_rules.yml"
  # - "second_rules.yml"

scrape_configs:
  # Prometheus itself
  - job_name: 'prometheus'
    static_configs:
      - targets: ['localhost:9090']

  # Node Exporter – System metrics
  - job_name: 'node'
    static_configs:
      - targets: ['localhost:9100']

  # nginx Prometheus Exporter
  - job_name: 'nginx'
    static_configs:
      - targets: ['localhost:9113']
    metrics_path: /metrics
    scrape_interval: 10s

  # nginx Status (if enabled)
  - job_name: 'nginx-status'
    static_configs:
      - targets: ['localhost:80']
    metrics_path: /nginx_status
    scrape_interval: 10s
  # Note: This requires nginx status module to be enabled

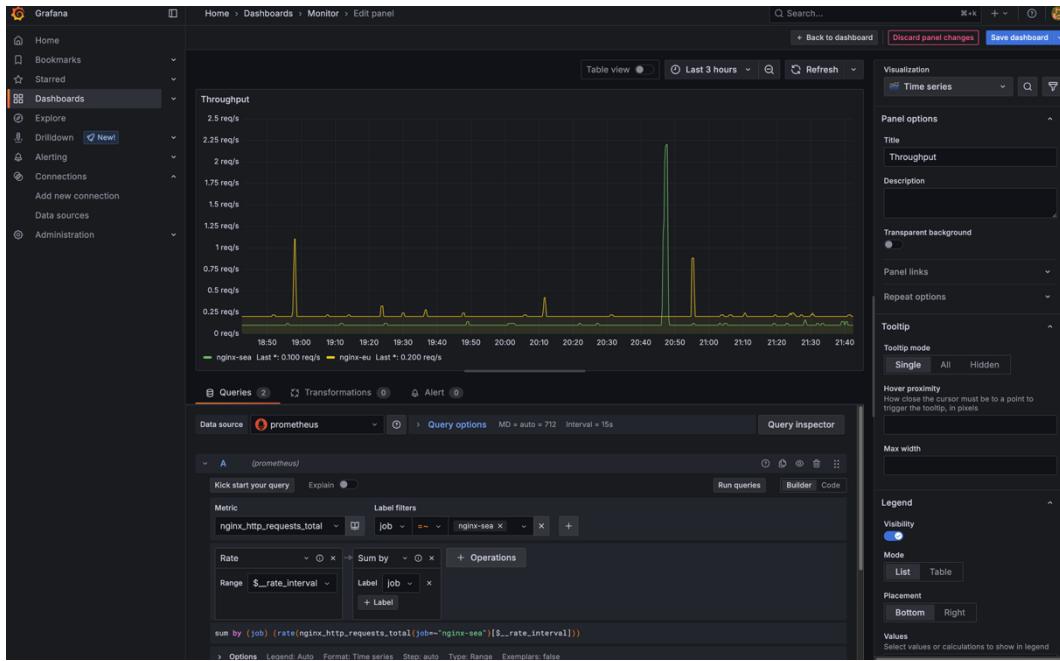
```

Gambar III-18 Cuplikan konfigurasi *prometheus.yml*

Gambar di atas menunjukkan konfigurasi dari Prometheus, hal utamanya adalah *scrape\_interval : 15s* yang berarti Prometheus mengekspor setiap 15 detik sekali, dan juga pengaturan pada *scrape\_configs*. *Scrape configs* mengatur target yang metriknya akan dikumpulkan, bagaimana caranya dan frekuensi mengumpulkan metrik. Terlihat pada gambar di atas *job* untuk mengumpulkan metrik dari *node exporter* dan *nginx exporter*.

Langkah berikutnya adalah instalasi Grafana yang berfungsi sebagai *dashboard* visualisasi. Komponen ini dipasang pada server *monitoring* menggunakan manajer paket apt. Setelah diinstal, Grafana dapat diakses pada web dengan port 3000. Lalu

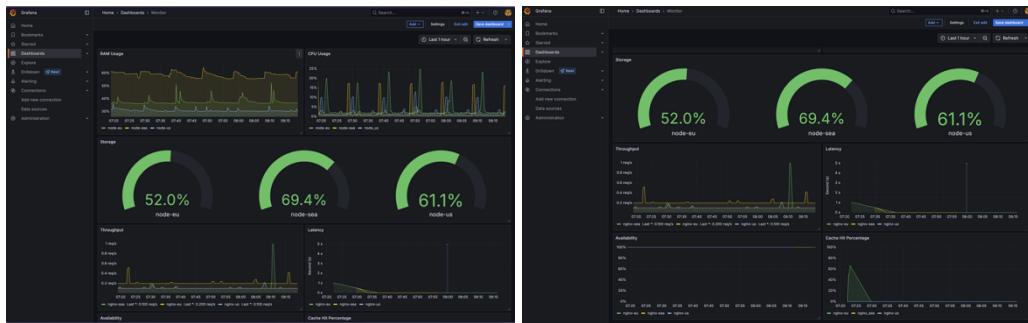
pada bagian *connection*, ditambahkan sumber data baru yakni Prometheus. *Dashboard* baru dibuat berdasarkan sumber tersebut kemudian dibuat visualisasi baru. Saat membuat visualisasi, digunakan PromQL, untuk meng-*query* data dari Prometheus. Grafana sudah menyediakan *builder* yang mempermudah menggunakan jika tidak mengetahui struktur dari PromQL.



Gambar III-19 Visualisasi dengan Grafana untuk *throughput*

Contoh gambar di atas adalah visualisasi *time-series* untuk *throughput* atau kapasitas. *Troughput* dapat dipantau dengan melihat metrik *nginx\_http\_request\_total* yang diekspos oleh *nginx-prometheus-exporter*. Contoh diatas menampilkan *builder* untuk *query* metrik yang ada pada Prometheus. *Query* tersebut dapat dikustomisasi sesuai dengan kebutuhan, seperti pada gambar mengatur filter yang dilakukan oleh job “nginx-sea” yang berarti yang sedang dilakukan *query* adalah metrik *nginx\_http\_request\_total* untuk server CDN Asia Tenggara. Digunakan *rate* yang merupakan rata-rata per detik perubahan dari metrik tersebut dan *sum* yang menjumlahkan *request rates* untuk semua *instance* dengan *job* yang sama. Grafana juga menyediakan fitur untuk menjelaskan *query* dan melakukan *query* dengan PromQL *code*. Untuk semua visualisasi dilakukan 3 *query* sehingga menampilkan 3 grafik yang berbeda dalam 1 visualisasi yang sama

yang kemudian diberikan legenda untuk memberi info bahwa grafik dimiliki oleh CDN tertentu.



Gambar III-20 (a) Tampilan antarmuka Grafana untuk sistem pemantauan untuk metrik CPU, RAM penyimpanan, kapasitas dan latensi (b) Tampilan antarmuka Grafana untuk sistem pemantauan untuk penyimpanan, kapasitas, latensi, ketersediaan dan *cache hit ratio*

Gambar III-20 menunjukkan tampilan antarmuka Grafana untuk keseluruhan metrik yang dipantau, yaitu tingkat penggunaan RAM, penggunaan CPU, penggunaan penyimpanan, *troughput*, latensi, ketersediaan dan Cache Hit Rate. Untuk CPU, *troughput*, latensi, ketersediaan dan Cache Hit Rate bersifat menampilkan *rate* yang berarti menggunakan perubahan rata-rata per detik sedangkan RAM dan penyimpanan tidak memerlukan *rate*.

## BAB IV PENGUJIAN DAN ANALISIS

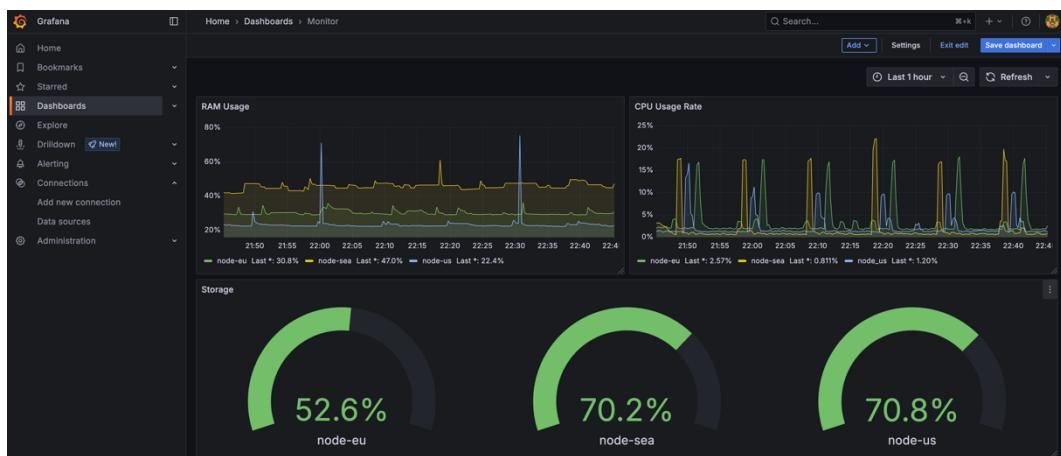
Bab ini membahas data-data pengujian beserta analisis pemenuhan kebutuhan/persyaratan sistem.

### 4.1 Pengujian Desain

Pengujian desain tentunya bergantung pada setiap subsistem. Untuk subsistem pemantauan berupa kesesuaian hasil pemantauan terhadap *testing*. Untuk subsistem server dilakukan *user testing* secara manual dan *load testing* untuk melihat apakah permintaan direspon dengan baik, kesesuaian letak server yang melayani terhadap lokasi pengguna, jumlah *throughput*, dan latensi pengguna.

#### 4.1.1. Subsistem Pemantauan

Pengujian untuk subsistem pemantauan secara umum berupa melihat kesesuaian hasil pemantauan terhadap yang sebenarnya. Untuk itu dilakukan pengamatan objektif terhadap fungsi-fungsi dari subsistem.



Gambar IV-1 Tampilan antarmuka Grafana

Berdasarkan grafik pada Grafana, terlihat bahwa penggunaan RAM nya untuk server Asia adalah 47%, untuk penggunaan CPU nya adalah 0.811% dan penyimpanannya sudah terisi sebanyak 70.2%.

```
Welcome to Ubuntu 24.04.2 LTS (GNU/Linux 6.11.0-1017-gcp x86_64)

* Documentation: https://help.ubuntu.com
* Management: https://landscape.canonical.com
* Support: https://ubuntu.com/pro

System information as of Sat Jul 26 15:41:01 UTC 2025

System load: 0.01      Processes:          121
Usage of /: 70.0% of 8.65GB  Users logged in: 0
Memory usage: 47%        IPv4 address for ens4: 10.184.0.3
Swap usage: 0%          Swap usage: 0

Expanded Security Maintenance for Applications is not enabled.

46 updates can be applied immediately.
To see these additional updates run: apt list --upgradable

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

*** System restart required ***
Last login: Sat Jul 26 15:37:43 2025 from 103.171.30.58
sohryuuasuka@cdn-sea:~$ 
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
351610	node_ex+	20	0	726216	13456	4548	S	0.3	1.4	6:19.41	node_exporter
765698	root	20	0	1191684	18764	8000	S	0.3	1.9	37:13.07	fluent-bit
1463752	root	20	0	12344	5928	3752	R	0.3	0.6	0:00.10	top
	1 root	20	0	22794	12024	7544	S	0.0	1.2	5:85.53	systemd
	2 root	20	0	0	0	0	S	0.0	0.0	0:00.63	kthreadd

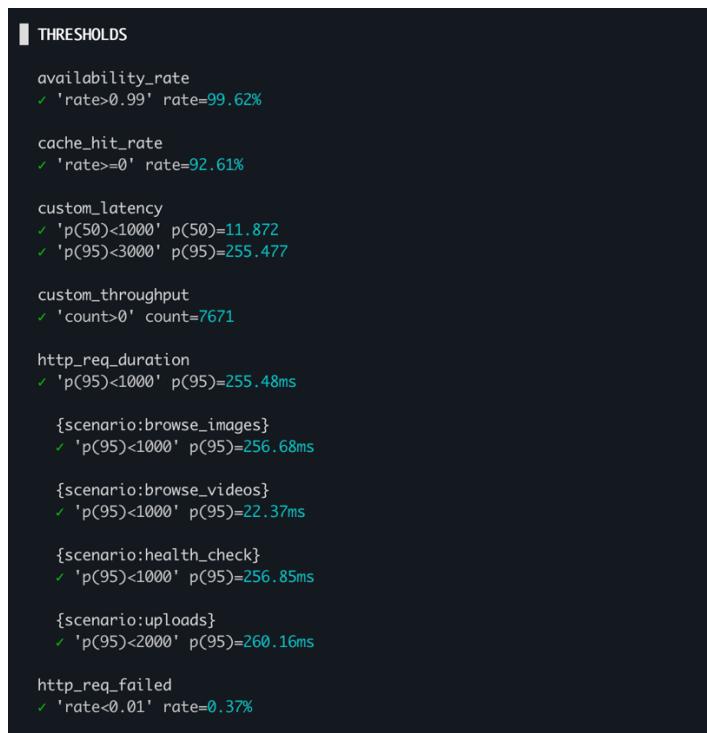
Gambar IV-2 Tampilan sistem pada Ubuntu Server SEA

Sebagai perbandingan, nilai CPU, RAM dan penyimpanan yang digunakan dibandingkan dengan nilai yang tertera pada Ubuntu Server region SEA. Nilai RAM dan penyimpanan tertera secara langsung ketika melakukan SSH ke server sedangkan untuk penggunaan CPU dilihat dengan menjalankan program *top* yang memunculkan proses yang berjalan pada server sekaligus jumlah CPU dan memori yang digunakan proses tersebut. Nilai CPU, RAM dan penyimpanan yang tertera pada server berturut-turut adalah 0.9%, 47% dan 70%. Karena nilainya yang tidak jauh dengan yang tertera pada Grafana.



Gambar IV-3 Tampilan Grafana setelah dilakukan pengujian terhadap perubahan nilai

Gambar IV-3 menunjukkan visualisasi dasbor pada Grafana setelah dilakukan pengujian dengan menggunakan *traffic* buatan untuk melihat apakah nilai yang ditampilkan berubah sesuai dengan *traffic* yang ada. Visualisasi menunjukkan adanya perubahan nilai pada grafik berdasarkan *traffic* yang ada sehingga dapat dinyatakan visualisasi grafik responsive terhadap perubahan nilai yang diukur.



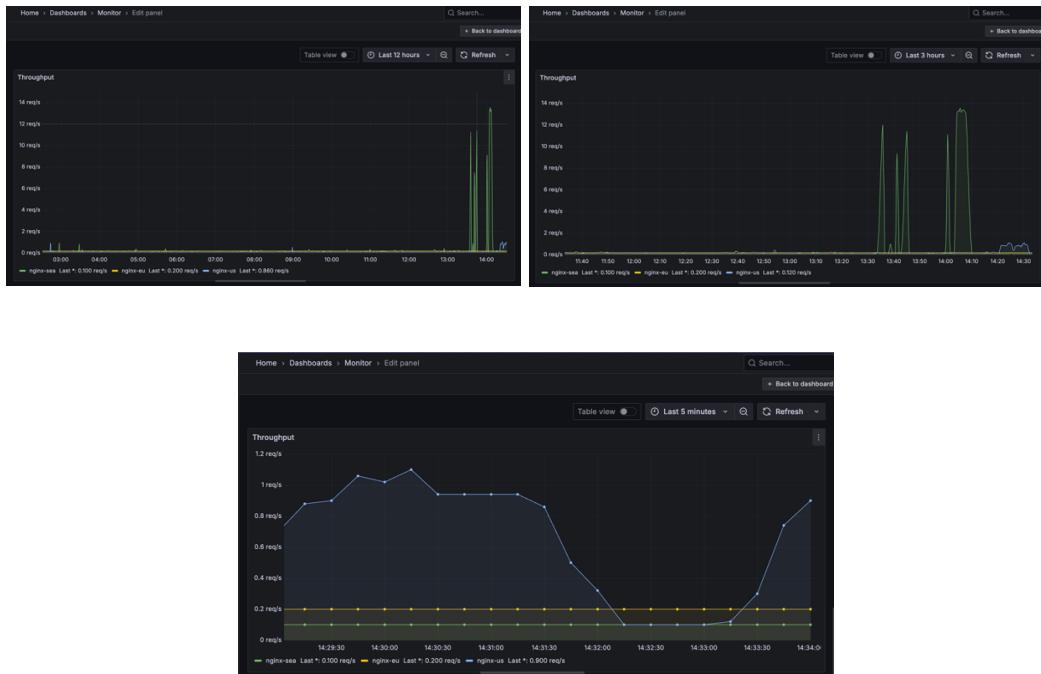
The screenshot shows a 'THRESHOLDS' section from a Grafana dashboard. It lists several metrics with their respective thresholds and current values:

- availability\_rate**:
  - ✓ 'rate>0.99' rate=99.62%
- cache\_hit\_rate**:
  - ✓ 'rate>=0' rate=92.61%
- custom\_latency**:
  - ✓ 'p(50)<1000' p(50)=11.872
  - ✓ 'p(95)<3000' p(95)=255.477
- custom\_throughput**:
  - ✓ 'count>0' count=7671
- http\_req\_duration**:
  - ✓ 'p(95)<1000' p(95)=255.48ms
- {scenario:browse\_images}**:
  - ✓ 'p(95)<1000' p(95)=256.68ms
- {scenario:browse\_videos}**:
  - ✓ 'p(95)<1000' p(95)=22.37ms
- {scenario:health\_check}**:
  - ✓ 'p(95)<1000' p(95)=256.85ms
- {scenario:uploads}**:
  - ✓ 'p(95)<2000' p(95)=260.16ms
- http\_req\_failed**:
  - ✓ 'rate<0.01' rate=0.37%

Gambar IV-4 Hasil Grafana k6 yang menunjukkan nilai terukur setelah pengetesan

Gambar IV-4 menunjukkan beberapa nilai yang dapat menjadi acuan yang seharusnya terukur pada Grafana. Rata-rata latensi yang terukur adalah 12 ms, total koneksi yang terjadi selama 9 menit adalah 7671 koneksi dan *cache hit rate* sekitar 92%. Adanya perbedaan nilai antara Grafana dan hasil laporan Grafana k6 dikarenakan beberapa faktor terutama perbedaan cara menghitung dari kedua metode. Namun, perbedaan ini tetap dalam batas yang dapat ditolerir.

Selanjutnya adalah pengetesan fitur Grafana untuk melihat sistem secara historis dalam rentang waktu yang sempit atau luas.



Gambar IV-5 Visualisasi *troughput* dengan opsi rentang waktu (a) 12 jam, (b) 3 jam dan (c) 5 menit terakhir pada Grafana

Gambar di atas menunjukkan visualisasi *troughput* pada rentang waktu variatif dari yang cenderung lama hingga lebih terkini yaitu 12 jam, 3 jam dan 5 menit dari waktu setempat. Lebih dari itu, Visualisasi pada Grafana dapat dimanipulasi untuk menampilkan data historis pada rentang tanggal tertentu yang sudah lama lewat dan dapat menampilkan rentang waktu yang relatif lebih singkat seperti 30 detik dari sekarang. Hal ini menunjukkan bahwa Grafana dapat memvisualisasikan data historis dalam rentang waktu yang beragam.

Berdasarkan hasil pengujian yang telah dilakukan, berikut adalah rangkuman dari pengujian fungsional pada Tabel IV-1

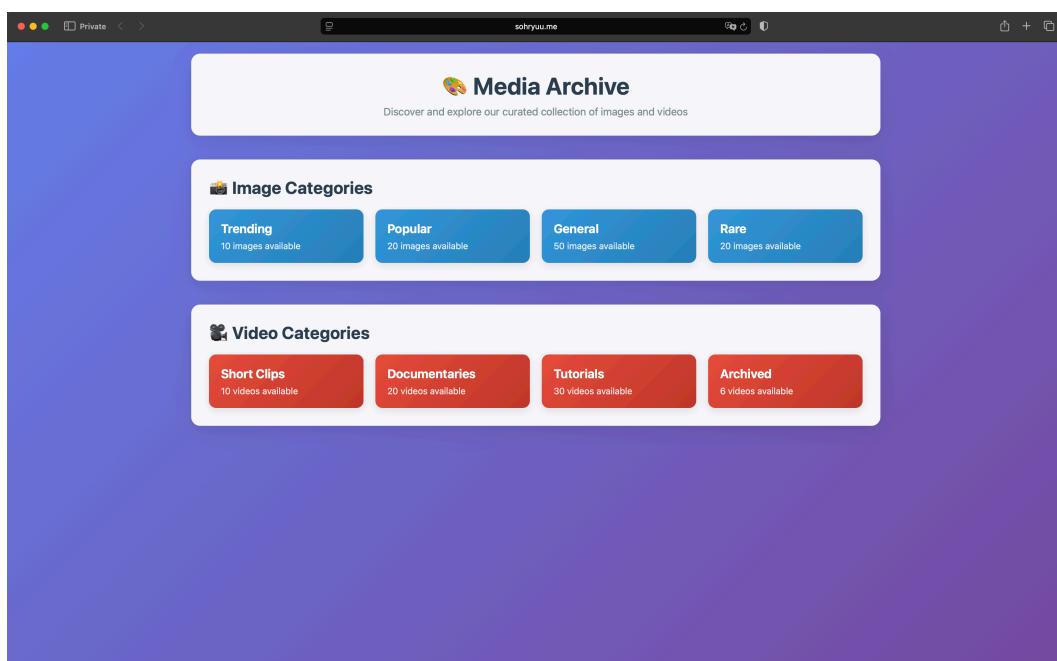
Tabel IV-1 Spesifikasi fungsi dan pengujian subsistem pemantauan

No	Deskripsi	Prosedur	Hasil	Hasil yang diharapkan	Keterangan
----	-----------	----------	-------	-----------------------	------------

1	Menampilkan metrik performa sistem dan log secara visual	<ul style="list-style-type: none"> <li>-Mengakses dasbor Grafana dan melihat visualisasi</li> <li>-Membandingkan hasil visualisasi dengan <i>load test</i></li> <li>-Mengakses visualisasi historis dan dalam rentang waktu variatif</li> </ul>	Pengembang dapat melihat metrik performa sistem dan log secara visual	Pengembang dapat melihat metrik performa sistem dan log secara visual	Berhasil
---	--	---	---	---	----------

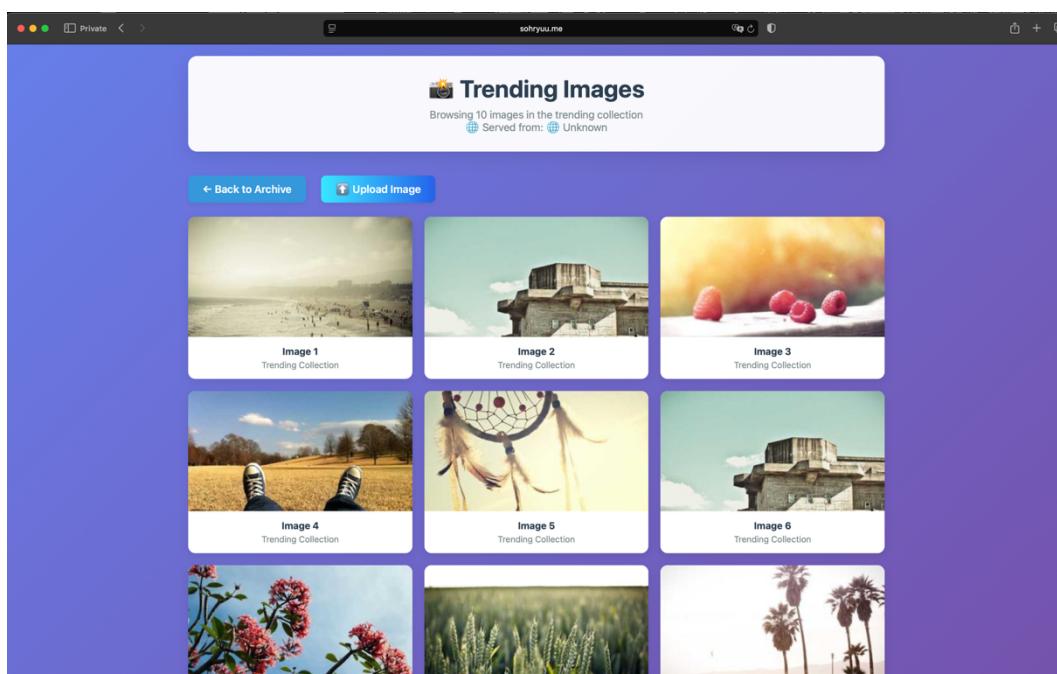
#### 4.1.2. Subsistem Server

Seperti yang sudah disebutkan sebelumnya, untuk subsistem server dilakukan pengujian berdasarkan apakah permintaan direspon dengan baik, kesesuaian letak server yang melayani terhadap lokasi pengguna, dan performa server. Untuk menguji bahwa server (aplikasi web) dapat merespon permintaan pengguna dengan baik, dilakukan pengetesan secara manual fitur-fiturnya.



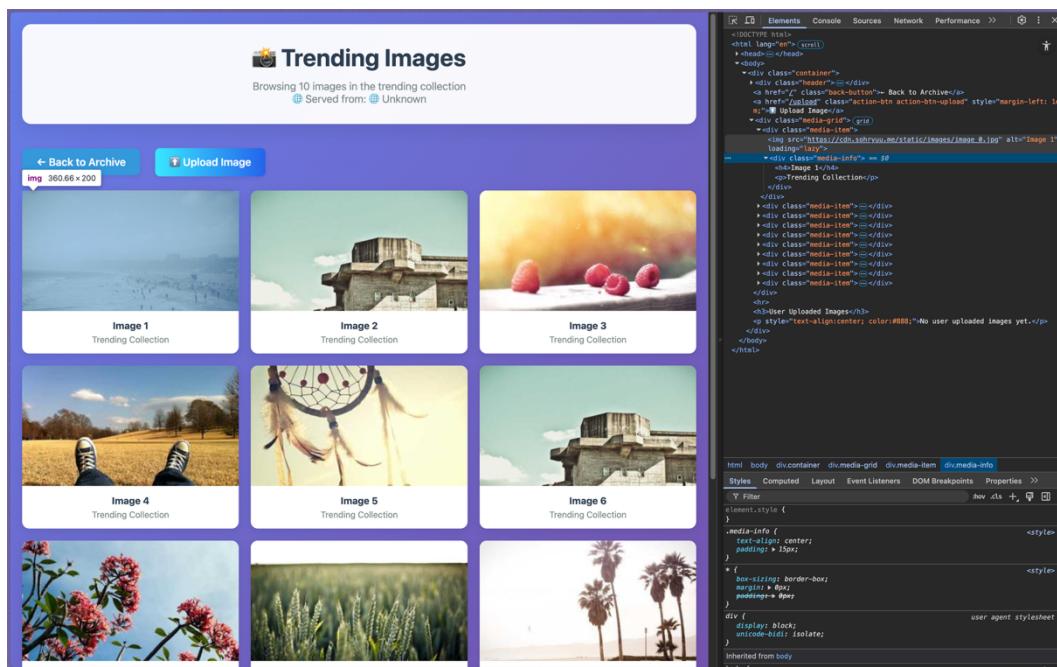
Gambar IV-6 Hasil akses ke halaman utama aplikasi web Flask

Gambar di atas menunjukkan hasil akses ke halaman utama dari aplikasi web Flask dengan menggunakan browser Safari dengan mode Private sehingga tidak menggunakan *cache* dan *cookie* pada browser.



Gambar IV-7 Hasil akses ke halaman *trending images* aplikasi web Flask

Gambar di atas menunjukkan hasil akses ke halaman *trending images* aplikasi web Flask. Hasil menunjukkan bahwa aplikasi dapat merespon dengan memberikan konten yang sesuai.

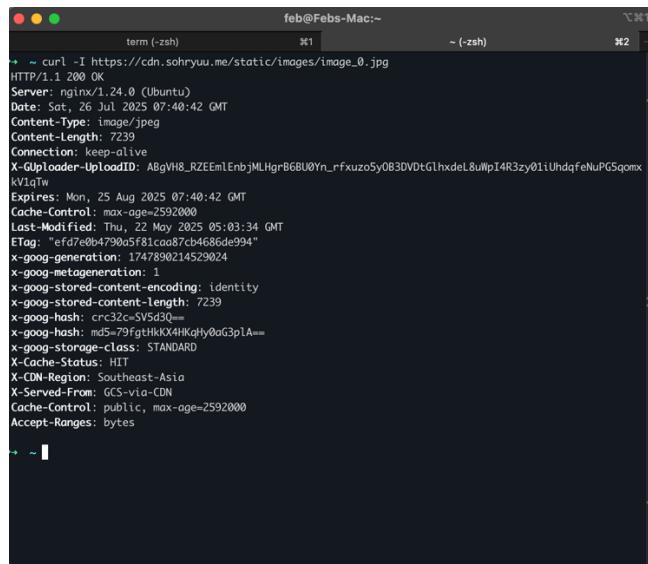


Gambar IV-8 Hasil inspect element ke halaman *trending images* aplikasi web Flask

Pada browser terdapat fitur *inspect element* yang memungkinkan pengguna untuk melihat *source code* aplikasi web. Dari *inspect element* dapat diketahui bahwa konten gambar diambil dari URL CDN, bukan *bucket storage* utama. Hal ini membuktikan bahwa aplikasi Flask dapat mendeteksi keberadaan CDN kemudian mengganti URL asli dengan URL CDN. Untuk mengetahui CDN mana yang memberikan konten, dilakukan pengecekan terhadap *header* dari respon HTTP karena sudah diatur sedemikian rupa sehingga CDN di *region* berbeda memiliki *header* yang berbeda terkhusus untuk *custom header X-CDN-Region*.

Gambar IV-9 Hasil HTTP header pada browser

Gambar di atas menunjukkan *header* dengan mengeceknya melalui *browser*. Hal ini dapat dilakukan dengan membuat *custom endpoint* pada aplikasi Flask. Namun, untuk melihat hal ini menggunakan domain dari CDN ([cdn.sohryuu.me](https://cdn.sohryuu.me)). Tampak pada gambar bahwa nilai untuk *header X-CDN-Region* adalah *Southeast-Asia*. Pengujian juga dapat dilakukan dengan menggunakan *curl* pada terminal perangkat pengguna.

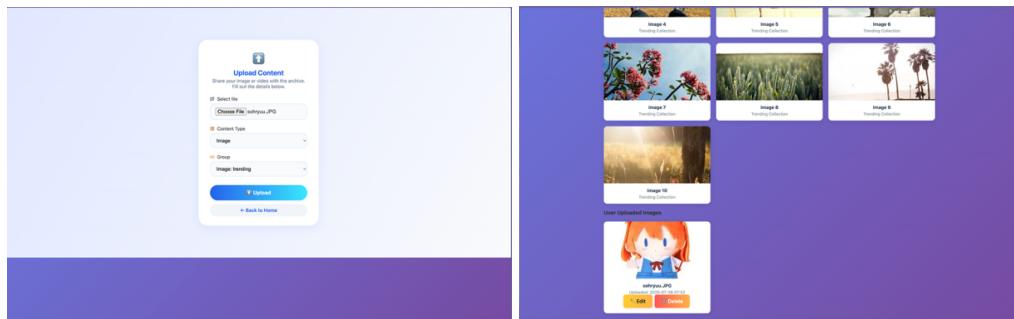
A screenshot of a Mac OS terminal window titled "term (-zsh)". It shows the command "curl -I https://cdn.sohryuu.me/static/images/image\_0.jpg" being run. The output displays various HTTP headers, including "X-CDN-Region: Southeast-Asia".

```
feb@Feb-Mac:~ term (-zsh) ~ (-zsh)
HTTP/1.1 200 OK
Server: nginx/1.24.0 (Ubuntu)
Date: Sat, 26 Jul 2025 07:40:42 GMT
Content-Type: image/jpeg
Content-Length: 7239
Connection: keep-alive
X-Cloud-Upload-ID: ABgVH8_RZEEmlEnbjMLHgrB6BU0Yn_rfxuzo5yOB3DVtGlxdeL8uWpT4R3zy01iUhdqfeNuPG5qomxKViqjW
Expires: Mon, 25 Aug 2025 07:40:42 GMT
Cache-Control: max-age=2592000
Last-Modified: Thu, 22 May 2025 05:03:34 GMT
ETag: "efdf7e0b4790ad5f81caa87cb4686de994"
x-goog-generation: 1747890214529024
x-goog-metageneration: 1
x-goog-stored-content-encoding: identity
x-goog-stored-content-length: 7239
x-goog-hash: crc32c-SV5s30m
x-goog-hash: md5-79fgfHkX4HKdHy0aG3pla==
x-goog-storage-class: STANDARD
X-Cache-Status: HIT
X-CDN-Region: Southeast-Asia
X-Served-From: GCS-via-CDN
Cache-Control: public, max-age=2592000
Accept-Ranges: bytes
```

Gambar IV-10 Hasil HTTP *header* dengan menggunakan metode *curl*

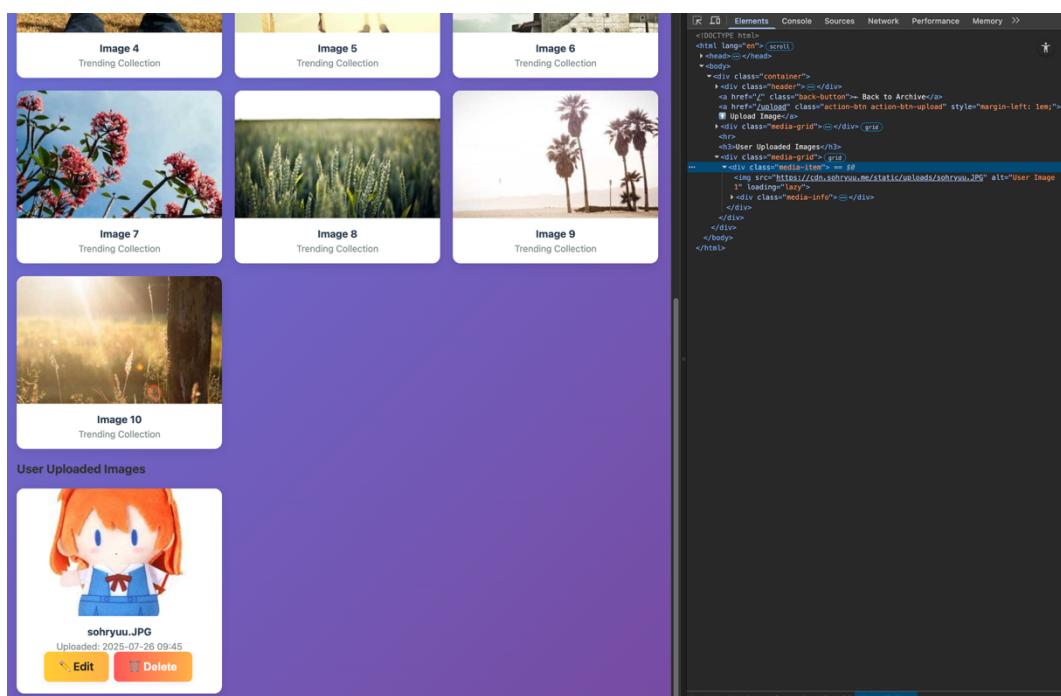
Dari gambar di atas yang merupakan hasil pengujian dengan *curl* ke *endpoint* CDN, nampak bahwa *X-CDN-Region* bernilai *Southeast-Asia* dan *X-Cache-Status* bernilai HIT. Hal ini membuktikan bahwa untuk pengguna yang berada di Indonesia GeoDNS merutekan permintaan yang menuju domain [cdn.sohryuu.me](https://cdn.sohryuu.me) ke server yang berada di Jakarta, Asia Tenggara.

Selanjutnya adalah pengetesan terhadap fitur unggah konten. Pengguna mengisi form singkat yang berisikan file, tipe konten dan grup.



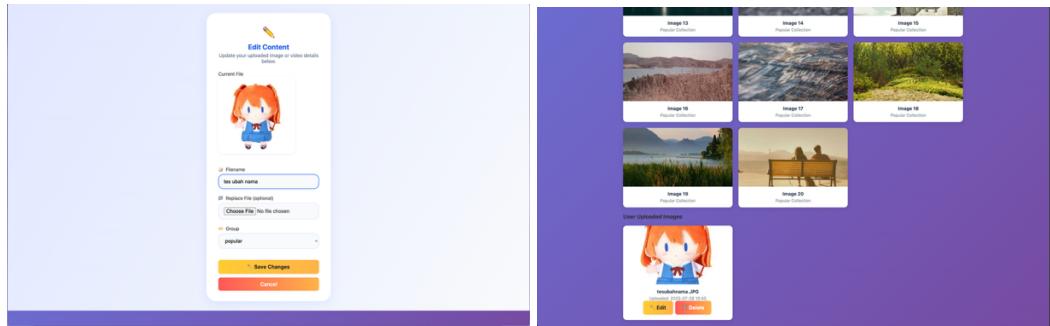
Gambar IV-11 Gambar (a) *form* pengunggahan konten dan (b) halaman *trending* setelah dilakukan pengunggahan konten

Dari hasil pengujian berdasarkan gambar di atas, aplikasi web dapat mengunggah konten gambar dengan baik dan dapat ditampilkan dengan segera. Selanjutnya dicek apakah konten tersebut juga diakses dari CDN atau tidak.



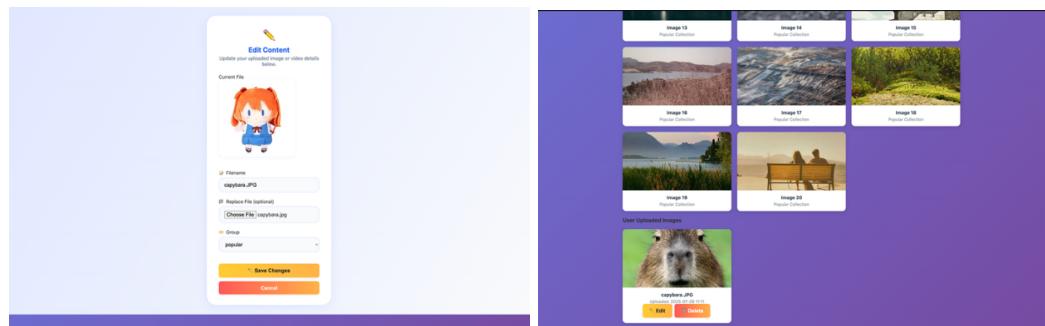
Gambar IV-12 Hasil *inspect element* untuk halaman *trending page* untuk konten yang diunggah pengguna

Dari hasil *inspect element* di atas, terbukti bahwa konten gambar juga diakses dengan URL CDN, bukan *bucket storage*.



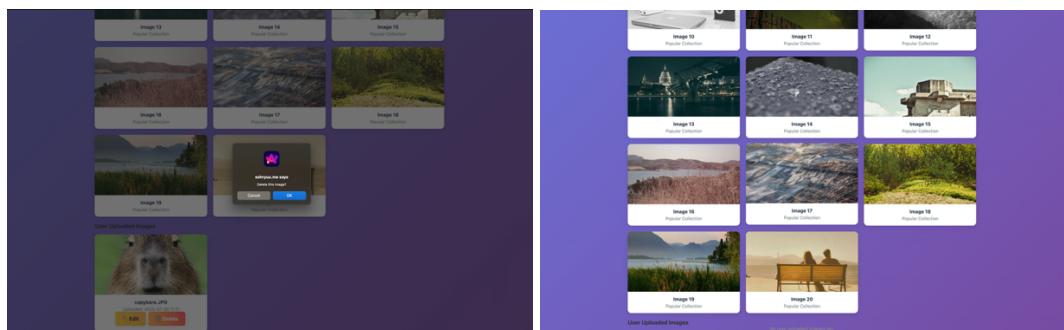
Gambar IV-13 Gambar (a) form pengubahan konten dan (b) halaman popular setelah dilakukan pengubahan konten dengan mengubah nama

Pengujian di atas berhasil menunjukkan bahwa aplikasi web dapat melakukan pengubahan konten yang berupa nama dan kategori (*trending/popular/general* dll.).



Gambar IV-14 Gambar (a) form pengubahan konten dan (b) halaman popular setelah dilakukan pengubahan konten dengan gambar baru dengan mengubah gambar

Pengujian di atas berhasil menunjukkan bahwa aplikasi web dapat melakukan pengubahan konten yang berupa pengubahan gambar dan nama. Kemudian dilakukan pengetesan untuk menghapus konten tersebut.



Gambar IV-15 Gambar konfirmasi penghapusan konten dan hasil setelah konten dihapus

Pengujian di atas berhasil menunjukkan bahwa aplikasi web dapat menghapus konten yang diunggah oleh pengguna dan tidak menampilkan lagi.

Selanjutnya adalah pengetesan lebih lanjut untuk perutean permintaan pengguna dengan menguji seberapa akurat GeoDNS merutekan permintaan ke server terdekat yang nantinya juga berkorelasi dengan latensi akses pengguna. Salah satu cara sederhana untuk melihatnya adalah dengan menggunakan aplikasi web pihak ketiga seperti ping-admin yang menggunakan virtual server di berbagai belahan dunia untuk melakukan ping ICMP *request* ke domain situs web.



Gambar IV-16 Gambar peta server dan pengguna virtual

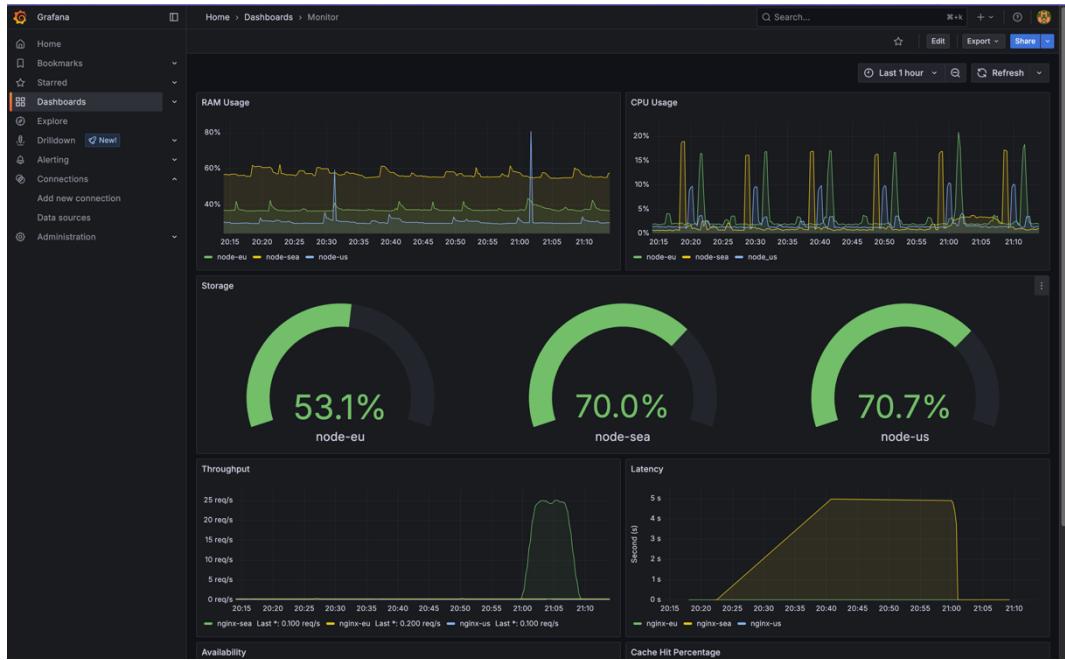
Berdasarkan gambar IV-16, terlihat bahwa server CDN yang melayani permintaan sudah sesuai secara letak geografisnya. Untuk wilayah Amerika Utara dan Selatan dilayani oleh server yang berada di Amerika Utara, untuk wilayah Eropa dan Rusia dilayani oleh server yang berada di Eropa dan terakhir untuk wilayah Asia Timur dan Tenggara dilayani oleh server yang berada di Asia Tenggara. Untuk selengkapnya dapat melihat informasi berikut

Monitoring node	IP of your site	Total time, sec.	DNS, sec.	Connection, sec.	SSL, sec.	Waiting for answer, sec.	Load speed	Page size
Russia, Irkutsk Powered by <a href="#">BAIKAL-IX</a> .	35.197.236.92	0.572066	0.179226 (31.33%)	0.115814 (20.24%)	0.162314 (28.37%)	0.114562 (20.03%)	5.47 MB/c	2 KB Compressed to 602 B (72%)
Russia, Krasnodar, north Powered by <a href="#">General-IT</a> .	35.197.236.92	0.412137	0.146697 (35.59%)	0.077754 (18.87%)	0.112068 (27.19%)	0.075311 (18.27%)	2.44 MB/c	2 KB Compressed to 602 B (72%)
Russia, Rostov-on-Don	35.197.236.92	0.459226	0.102911 (22.41%)	0.077478 (16.87%)	0.2032 (44.25%)	0.075350 (16.41%)	3.17 MB/c	2 KB Compressed to 602 B (72%)
Austria, Vienna Powered by <a href="#">geza</a> .	35.197.236.92	0.086607	0.008928 (10.31%)	0.020619 (23.81%)	0.038232 (44.14%)	0.018726 (21.62%)	6.83 MB/c	2 KB Compressed to 602 B (72%)
Vietnam, Hanoi Powered by <a href="#">GreenCloudVPS</a> .	34.101.140.128	4.144021	3.872236 (93.44%)	0.062866 (1.52%)	0.144967 (3.50%)	0.063849 (1.54%)	7.46 MB/c	2 KB Compressed to 602 B (72%)
Germany, Erfurt Powered by <a href="#">Keyweb</a> .	35.197.236.92	0.210854	0.060459 (28.67%)	0.021033 (9.98%)	0.109547 (51.95%)	0.019635 (9.31%)	5.57 MB/c	2 KB Compressed to 602 B (72%)
Spain, Madrid Powered by <a href="#">Meibicom</a> .	35.197.236.92	0.179137	0.038093 (21.26%)	0.026097 (14.57%)	0.089721 (50.09%)	0.024920 (13.91%)	2.68 MB/c	2 KB Compressed to 602 B (72%)
Italy, Arezzo	35.197.236.92	0.320853	0.124565 (38.82%)	0.029331 (9.14%)	0.138435 (43.15%)	0.028341 (8.83%)	4.35 MB/c	2 KB Compressed to 602 B (72%)
Canada, Beauharnois Powered by <a href="#">MrHost</a> .	34.23.29.132	0.143239	0.023441 (16.36%)	0.026379 (18.42%)	0.066132 (46.17%)	0.027200 (18.99%)	9.41 MB/c	2 KB Compressed to 602 B (72%)
China, Jiangsu Powered by <a href="#">Compevo</a> .	34.101.140.128	0.419626	0.012985 (3.09%)	0.085406 (20.35%)	0.235097 (56.03%)	0.085845 (20.46%)	2.73 MB/c	2 KB Compressed to 602 B (72%)
Mexico, Puebla Powered by <a href="#">FullVPServer</a> .	34.23.29.132	0.363171	0.027533 (7.58%)	0.075220 (20.71%)	0.184756 (50.87%)	0.074829 (20.60%)	960 KB/c	2 KB Compressed to 602 B (72%)
Singapore Powered by <a href="#">iLDC</a> .	34.101.140.128	0.113436	0.010904 (9.61%)	0.017268 (15.22%)	0.068917 (60.75%)	0.016071 (14.17%)	2.79 MB/c	2 KB Compressed to 602 B (72%)
USA, GA, Atlanta, south Powered by <a href="#">RackNerd</a> .	34.23.29.132	0.12411	0.013669 (11.01%)	0.020375 (16.42%)	0.069535 (56.03%)	0.020326 (16.38%)	3.50 MB/c	2 KB Compressed to 602 B (72%)
USA, CA, Los Angeles 1 Powered by <a href="#">Friendhosting.net</a> .	34.23.29.132	0.240592	0.012936 (5.38%)	0.064898 (26.97%)	0.09759 (40.56%)	0.065029 (27.03%)	5.52 MB/c	2 KB Compressed to 602 B (72%)
USA, NY, Garden City Powered by <a href="#">FirstByte</a> .	34.23.29.132	0.112749	0.002933 (2.60%)	0.018843 (16.71%)	0.071821 (63.70%)	0.018842 (16.71%)	2.71 MB/c	2 KB Compressed to 602 B (72%)
Taiwan, Taipei	34.101.140.128	0.3741	0.101779 (27.21%)	0.068288 (18.25%)	0.137097 (36.65%)	0.066587 (17.80%)	2.07 MB/c	2 KB Compressed to 602 B (72%)
France, Roubaix Powered by <a href="#">Z-Tv Corporation</a> .	35.197.236.92	0.125701	0.018050 (14.36%)	0.010727 (8.53%)	0.087383 (69.52%)	0.009390 (7.47%)	7.46 MB/c	2 KB Compressed to 602 B (72%)
Japan, Tokyo	34.101.140.128	0.504053	0.004165 (0.83%)	0.089559 (17.77%)	0.321599 (63.80%)	0.088535 (17.56%)	4.04 MB/c	2 KB Compressed to 602 B (72%)

Gambar IV-17 Tabel hasil pengujian pada website *ping-admin*

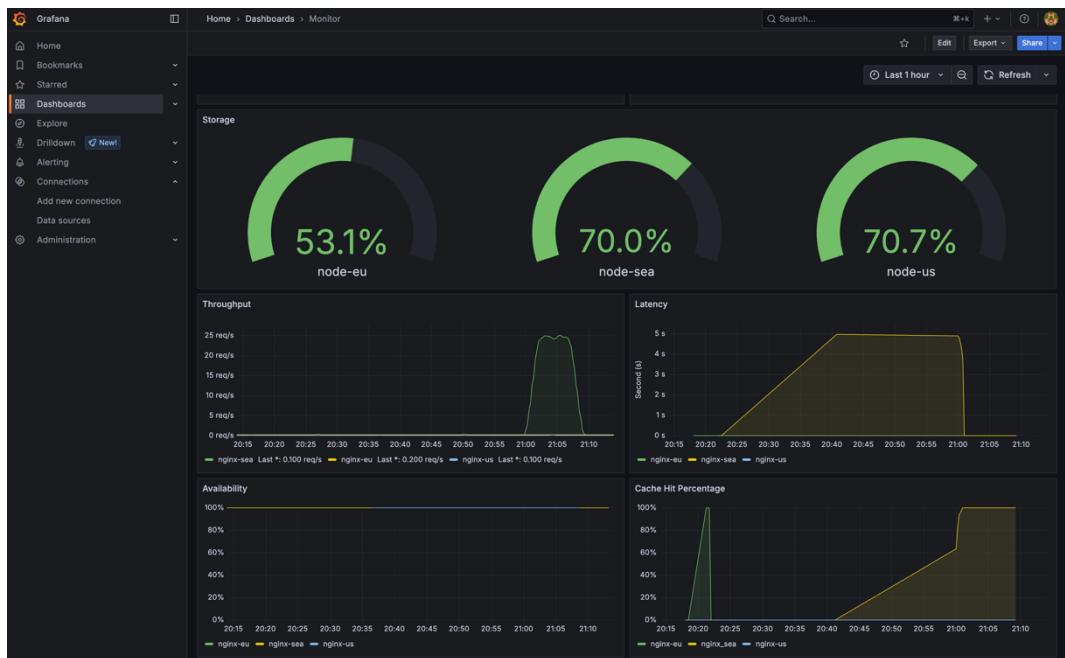
Lebih jelasnya dapat terlihat pada gambar di atas untuk IP server yang melayani pengguna. IP server Amerika, Eropa dan Asia berturut-turut adalah 34.23.29.132, 35.197.236.92 dan 34.101.140.128. Pada hasil di atas juga menunjukkan informasi seperti total waktu akses yang terdiri dari DNS *resolution*, *Connection*, SSL dan *waiting time*.

Pengujian untuk melihat kapasitas dan bagaimana perilaku sistem dalam *traffic* tinggi dapat dilakukan dengan melakukan *load testing* menggunakan Grafana k6. Skrip pengujian dibuat berupa kode JavaScript kemudian dijalankan pada mesin lokal, yang artinya permintaan yang dibuat ditujukan untuk server di SEA.



Gambar IV-18 Tampilan Grafana setelah dilakukan pengujian (1)

Gambar di atas menunjukkan metrik yang terekam dan divisualisasikan oleh Grafana. Pengetesan dilakukan sekitar pukul 21.00 waktu setempat. Untuk RAM dan CPU tidak memiliki kenaikan yang signifikan dan cenderung stabil. Untuk throughput, tercatat nilai tertingginya mencapai 25 req/s. Ketika terjadi banyak permintaan, latensi turun mendekati 0 ms.



Gambar IV-19 Tampilan Grafana setelah dilakukan pengujian (2)

Gambar di atas menunjukkan metrik ketersediaan dan Cache Hit Ratio. Ketika dilakukan pengujian, Cache Hit Ratio menunjukkan bahwa hampir semua permintaan yang diterima direspon dengan diberikan *cache* yang tersimpan pada CDN.

```
TOTAL RESULTS
checks_total.....: 32613 59.72621/s
checks_succeeded.....: 100.00% 32613 out of 32613
checks_failed.....: 0.00% 0 out of 32613

✓ health check ok
✓ health check response time OK
✓ health check response time measured
✓ home page loaded (app)
✓ home page response time OK
✓ image category rare loaded (cdn)
✓ image category response time OK
✓ image category tutorials loaded (cdn)
✗ image category trending loaded (cdn)
✗ image category video loaded (cdn)
✗ video category response time measured
✗ image category popular loaded (cdn)
✓ static image request completed
✓ static image response time measured
✓ video category archived loaded (cdn)
✓ image category trending loaded (cdn)
✓ image category general loaded (cdn)
✓ upload page loaded (app)
✓ upload page response time OK
✓ upload page response time measured
✓ video category documentaries loaded (cdn)
✓ video category short-clips loaded (cdn)

HTTP
http_req.duration.....: avg=95.51ms min=6.02ms med=26.99ms max=1.38s p(90)=261.87ms p(95)=272.35ms
{ expected.response=true }
{ scenario:browse_images }
{ scenario:health_check }
{ scenario:uploads }
http_req.failed.....: 0.00% 0 out of 14857
http_reqs.....: 14857 27.208546/s

EXECUTION
iteration.duration.....: avg=4.8s min=2.01s med=4.79s max=10.45s p(90)=6.4s p(95)=6.88s
iterations.....: 6887 12.61259/s
vus.....: 1 min=0 max=77
vus_max.....: 77 min=77 max=77

NETWORK
data_received.....: 109 MB 199 KB/s
data_sent.....: 1.6 MB 3.0 KB/s
```

Gambar IV-20 Laporan Grafana k6 setelah dilakukan pengujian untuk sistem dengan CDN

Setelah dilakukan pengujian *load testing*, Grafana k6 akan memberikan suatu laporan seperti pada gambar di atas. Dari laporan tersebut, diketahui nilai eksak untuk latensi ketika mengakses gambar di angka 102.29 ms dan untuk video di angka 25.15 ms untuk rata-ratanya. Juga diberikan report singkat mengenai status fungsi dan performa untuk keseluruhan sistem CDN. Karena hasil yang tidak meyakinkan, yaitu pengetesan akses video lebih cepat dari gambar, dilakukan pengujian yang lebih granular terhadap akses ke video.

Sebagai perbandingan, dilakukan *load test* jika CDN dinonaktifkan sehingga permintaan dilayani oleh server utama dan *bucket storage* yang terletak di Amerika Serikat sedangkan permintaan tetap dilakukan dari Indonesia.

```

TOTAL RESULTS
checks_total ..... : 25080 45.662183/s
checks_succeeded ..... : 100.00% 25080 out of 25080
checks_failed ..... : 0.00% 0 out of 25080

✓ Health check ok
✓ Health check response time OK
✓ Home page loaded (app)
✓ Home page response time OK
✓ Image category general loaded (app)
✓ Image category response time OK
✓ Video category archived loaded (app)
✓ Video category response time OK
✓ Image category popular loaded (app)
✓ Static image request completed
✓ Static image response time measured
✓ Image category trending loaded (app)
✓ Upload page loaded (app)
✓ Upload page response time OK
✓ Static video request completed
✓ Static video response time measured
✓ Video category tutorials loaded (app)
✓ Video category documentaries loaded (app)
✓ Image category rare loaded (app)
✓ Video category short-clips loaded (app)

HTTP
http_req_duration ..... : avg=725.91ms min=0.29ms med=255.66ms max=34.44s p(90)=296.88ms p(95)=372.76ms
  { expected_response:true }
  { scenario:browse_images_no_cdn }
  { scenario:browse_videos_no_cdn }
  { scenario:health_check }
  { scenario:uploads }
  http_req_failed ..... : 0.00% 0 out of 12541
  http_reqs ..... : 12541 22.832912/s

EXECUTION
iteration_duration ..... : avg=7.14s min=3.24s med=5.71s max=37.98s p(90)=7.93s p(95)=24.27s
iterations ..... : 4645 8.456971/s
vus ..... : 1 min=1 max=77
vus_max ..... : 77 min=77 max=77

NETWORK
data_received ..... : 9.1 GB 17 MB/s
data_sent ..... : 18 MB 32 kB/s

running (0m00.3s), 00/77 VUs, 4645 complete and 0 interrupted iterations
browse_main_and_images ✓ [=====] 00/58 VUs 9m0s
browse_videos ✓ [=====] 00/28 VUs 9m0s
health_checks ✓ [=====] 2 VUs 9m0s
simulate_uploads ✓ [=====] 0/5 VUs 9m0s
INFO[0549]: thresholds on metrics "http_req_duration{scenario:browse_videos_no_cdn}" have been crossed
└ testing git:curl0 ✘

```

Gambar IV-21 Laporan Grafana k6 setelah dilakukan pengujian untuk sistem tanpa CDN

Dari laporan tersebut, diketahui nilai eksak untuk latensi ketika mengakses gambar di angka 197.44 ms dan untuk video di angka 10.15 s untuk rata-ratanya. Hal ini menunjukkan bahwa dengan mengarahkan konten lebih dekat ke pengguna secara fisik akan mengurangi waktu akses dengan signifikan. Lalu untuk throughput nya sendiri untuk sistem tanpa CDN lebih kecil di angka 25080 permintaan dalam waktu 9 menit dibanding sistem dengan CDN dengan 32613 permintaan dalam kurun waktu yang sama.

```

↳ ~ # Test from multiple locations to find edge server issues
for i in {1..5}; do
    echo "Test $i:"
    curl -w "Time: ${time_total}s, Connect: ${time_connect}s, Speed: ${speed_download} B/s\n" \
        -o /dev/null -s "https://storage.googleapis.com/bucket-main-ta/static/videos/video_0.mp4"
    sleep 2
done
Test 1:
Time: 4.249348s, Connect: 0.030212s, Speed: 6351335 B/s
Test 2:
Time: 7.375526s, Connect: 0.024425s, Speed: 3659269 B/s
Test 3:
Time: 3.734755s, Connect: 0.027897s, Speed: 7226454 B/s
Test 4:
Time: 3.771023s, Connect: 0.026231s, Speed: 7156953 B/s
Test 5:
Time: 3.731738s, Connect: 0.024854s, Speed: 7232296 B/s
↳ ~ # Test from multiple locations to find edge server issues
for i in {1..5}; do
    echo "Test $i:"
    curl -w "Time: ${time_total}s, Connect: ${time_connect}s, Speed: ${speed_download} B/s\n" \
        -o /dev/null -s "https://cdn.sohryuu.me/static/videos/video_0.mp4"
    sleep 2
done
Test 1:
Time: 1.365672s, Connect: 0.074233s, Speed: 19762458 B/s
Test 2:
Time: 0.813747s, Connect: 0.012241s, Speed: 331665372 B/s
Test 3:
Time: 0.874258s, Connect: 0.017176s, Speed: 30870791 B/s
Test 4:
Time: 0.800683s, Connect: 0.016435s, Speed: 33707517 B/s
Test 5:
Time: 1.806383s, Connect: 0.016734s, Speed: 14940926 B/s

```

Gambar IV-22 Pengujian dengan *curl test* untuk salah satu konten video

Lebih lanjut, dilakukan pengujian dengan menggunakan metode *curl* untuk melihat latensi akses jika dalam kondisi ideal (tidak ada traffic berlebih seperti pada *load test*) saat mengakses file video. Hasil menunjukkan dalam 5 kali percobaan, waktu rata-rata yang diperlukan ketika mengakses langsung ke *bucket* adalah 4,5 detik sedangkan ketika menggunakan CDN, waktu rata-rata adalah 1,4 detik.

```

↳ ~ # Test from multiple locations to find edge server issues
for i in {1..5}; do
    echo "Test $i:"
    curl -w "Time: ${time_total}s, Connect: ${time_connect}s, Speed: ${speed_download} B/s\n" \
        -o /dev/null -s "https://cdn.sohryuu.me/static/images/image_0.jpg"
    sleep 2
done
Test 1:
Time: 0.040478s, Connect: 0.015767s, Speed: 178837 B/s
Test 2:
Time: 0.037905s, Connect: 0.014896s, Speed: 190977 B/s
Test 3:
Time: 0.050002s, Connect: 0.018872s, Speed: 144774 B/s
Test 4:
Time: 0.036235s, Connect: 0.012357s, Speed: 199779 B/s
Test 5:
Time: 0.042606s, Connect: 0.018129s, Speed: 169905 B/s
↳ ~ # Test from multiple locations to find edge server issues
for i in {1..5}; do
    echo "Test $i:"
    curl -w "Time: ${time_total}s, Connect: ${time_connect}s, Speed: ${speed_download} B/s\n" \
        -o /dev/null -s "https://storage.googleapis.com/bucket-main-ta/static/images/image_0.jpg"
    sleep 2
done
Test 1:
Time: 0.326414s, Connect: 0.030566s, Speed: 22177 B/s
Test 2:
Time: 0.413240s, Connect: 0.028584s, Speed: 17517 B/s
Test 3:
Time: 0.093196s, Connect: 0.034337s, Speed: 77675 B/s
Test 4:
Time: 0.087476s, Connect: 0.027999s, Speed: 82754 B/s
Test 5:
Time: 0.089632s, Connect: 0.029189s, Speed: 80763 B/s

```

Gambar IV-23 Pengujian dengan *curl test* untuk salah satu konten gambar

Pengujian dengan menggunakan metode *curl* dilakukan juga untuk melihat latensi jika dalam kondisi ideal saat mengakses file gambar. Hasil menunjukkan dalam 5 kali percobaan, waktu rata-rata yang diperlukan ketika mengakses langsung ke *bucket* adalah 253 milidetik sedangkan ketika menggunakan CDN, waktu rata-rata adalah 51,75 milidetik. Dari hasil pengujian dengan *curl*, masih dapat disimpulkan bahwa dengan menggunakan CDN akses konten menjadi lebih cepat.

Berdasarkan hasil pengujian yang telah dilakukan, berikut adalah rangkuman dari pengujian fungsional pada Tabel IV-2

Tabel IV-2 Spesifikasi fungsi dan pengujian subsistem server

No	Deskripsi	Prosedur	Hasil	Hasil yang diharapkan	Keterangan
1	Menerima dan memproses permintaan pengguna	<ul style="list-style-type: none"> <li>-Mengakses halaman utama aplikasi</li> <li>-Mengakses halaman foto dan video berdasarkan kategori</li> <li>-Mengupload konten baru</li> <li>-Mengedit konten yang diupload</li> <li>-Menghapus konten yang diupload</li> </ul>	Pengguna menerima respon yang sesuai berdasarkan hal yang dilakukan	Pengguna menerima respon yang sesuai berdasarkan hal yang dilakukan	Berhasil
2	Mengarahkan permintaan pengguna ke <i>node</i> CDN terdekat	-Mengakses konten web dari letak geografis yang berbeda-beda	Pengguna dilayani oleh server CDN terdekat	Pengguna dilayani oleh server CDN terdekat	Berhasil

3	Menyalin konten yang sesuai ke berbagai node	-Mengakses konten web dan melihat HTTP header cache status	Pengguna mengakses <i>cached content</i> dan merasakan latensi yang lebih rendah	Pengguna mengakses <i>cached content</i> dan merasakan latensi yang lebih rendah	Berhasil
---	--	--	--	--	----------

#### 4.1.3. Sistem Terintegrasi

Subsistem server CDN dan pembelajaran mesin diintegrasikan menggunakan VM yang di-deploy melalui layanan *public cloud*, yaitu GCP seperti yang ditunjukkan pada Gambar III-4. Berikut adalah spesifikasi yang digunakan untuk VM ML.

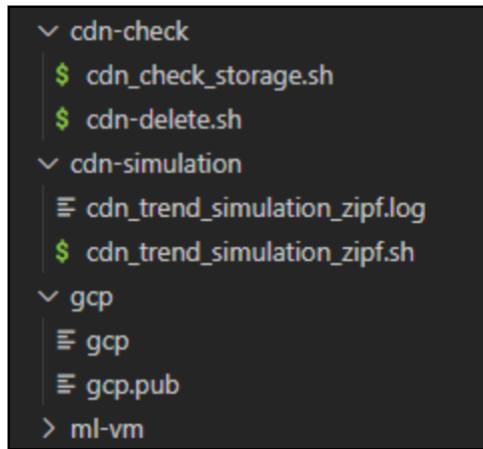
Basic information	
Name	ml-cdn
Instance Id	4749557405160614206
Description	None
Type	Instance
Status	Running
Creation time	Jun 5, 2025, 11:06:44 AM UTC+07:00
Location	us-central1-c
Boot disk source image	<a href="#">debian-12-bookworm-v20250513</a>
Boot disk architecture	X86_64
Instance template	None
In use by	None
Physical host	None
Maintenance status	—
Reservations	Automatically choose
Labels	goog-ops-a... : v2-x86-tem...
Tags	—
Deletion protection	Disabled
Confidential VM service	Disabled
Preserved state size	0 GB

Machine configuration	
Machine type	e2-medium (2 vCPUs, 4 GB Memory)
CPU platform	Intel Broadwell
Minimum CPU platform	None
Architecture	x86/64
vCPUs to core ratio	—
Custom visible cores	—
All-core turbo-only mode	—

Gambar IV-24 Cuplikan pengaturan VM untuk ML

Terdapat beberapa VM yang di-deploy di GCP dan salah satunya adalah VM untuk ML yang diberi nama ml-cdn. Gambar IV-2 menampilkan konfigurasi yang dibuat untuk menjalankan model ML. VM ini di-deploy di region US dengan sistem operasi Debian 12 yang berbasis Linux. Agar mengantisipasi pemrosesan yang berat, mesin ini menggunakan spesifikasi 2 virtual CPU dan memori 4 GB.



```
└── cdn-check
    ├── cdn_check_storage.sh
    └── cdn-delete.sh
└── cdn-simulation
    └── cdn_trend_simulation_zipf.log
    └── cdn_trend_simulation_zipf.sh
└── gcp
    └── gcp
        └── gcp.pub
└── ml-vm
```

Gambar IV-25 Struktur folder VM

Gambar IV-25 menunjukkan struktur file dan folder yang digunakan untuk implementasi ML pada sistem CDN. Terdapat empat folder utama yang digunakan pada proses pengujian sistem terintegrasi, yaitu cdn-check, cdn-simulation, gcp, dan ml-vm. Folder cdn-check terdiri atas dua *script* untuk menghapus dan mengecek *storage* yang ada pada server CDN secara *remote* menggunakan SSH yang mana *key* untuk SSH tersebut disimpan di folder gcp. *Script* cdn-delete.sh digunakan untuk menghapus file-file *cache* dan me-reset proses pengujian sehingga *storage* CDN tidak terisi file yang sudah di-*cache*, sedangkan *script* cdn\_check\_storage.sh digunakan untuk melihat dan memvalidasi hasil *eviction* yang sudah dilakukan dalam proses pengujian.

Setelah itu terdapat folder simulasi yang terdiri atas cdn\_trend\_simulation\_zipf.sh yang mana *script* ini memungkinkan simulasi dari pengguna untuk me-request konten-konten ke situs *dummy* yang sudah dirancang dan log hasil *request*. Simulasi ini menggunakan distribusi zipf yang memodelkan *request* pengguna di dunia nyata. *Request* disimulasikan sebanyak 500 *request* terhadap file video dan gambar, serta kategori situs yang sudah dirancang sebelumnya, seperti kategori ‘trending’,

‘popular’, dan lainnya. Di bagian kategori tersebut dideklarasikan bobot popularitas secara manual dengan kategori yang paling populer adalah ‘trending’.

```
ml-vm
  +-- deprecated
  +-- lfu
    +-- cache_index_lfu_eu.json
    +-- cache_index_lfu_sea.json
    +-- cache_index_lfu_us.json
    +-- cdn_eviction_lfu.py
    +-- cdn_trend_simulation.log
    +-- lfu_eviction_metrics_eu.json
    +-- lfu_eviction_metrics_sea.json
    +-- pull_index_lfu.sh
  +-- lrb-new
    +-- backup
    +-- cache_index_us.json
    +-- cdn_eviction_update.py
    +-- cdn_trend_simulation_SEA.log
    +-- cdn_trend_simulation.log
    +-- pull_index.sh
    +-- web_lrb_model_binary.pkl
    +-- web_lrb_model_distance.pkl
    +-- web_lrb_model_info.json
    +-- web_lrb_model_reuse.pkl
  +-- lru
    +-- cache_index_lru_us.json
    +-- cdn_eviction_lru.py
    +-- cdn_trend_simulation.log
    +-- lru_eviction_metrics_eu.json
    +-- lru_eviction_metrics_sea.json
    +-- pull_index_lru.sh
```

Gambar IV-26 Struktur folder VM ml-cdn

Gambar IV-26 menunjukkan isi dari folder ml-cdn. Pada folder ini terdapat tiga algoritma yang digunakan sebagai pengujian, yaitu LRB sebagai objek utama pengujian, LRU sebagai algoritma *baseline* dari CDN yang digunakan oleh NGINX, lalu LFU sebagai algoritma pembanding ketiga. Sesuai dengan diagram pada gambar II-6, sistem *eviction* ini akan mengambil data akses yang diubah

menjadi file json menggunakan *script* pull\_index.sh. File json akan diberi nama menjadi cache\_index\_[REGION].json, yang mana pada gambar IV-4 region yang digunakan adalah US. Berikut adalah cuplikan dari file json yang dibuat.

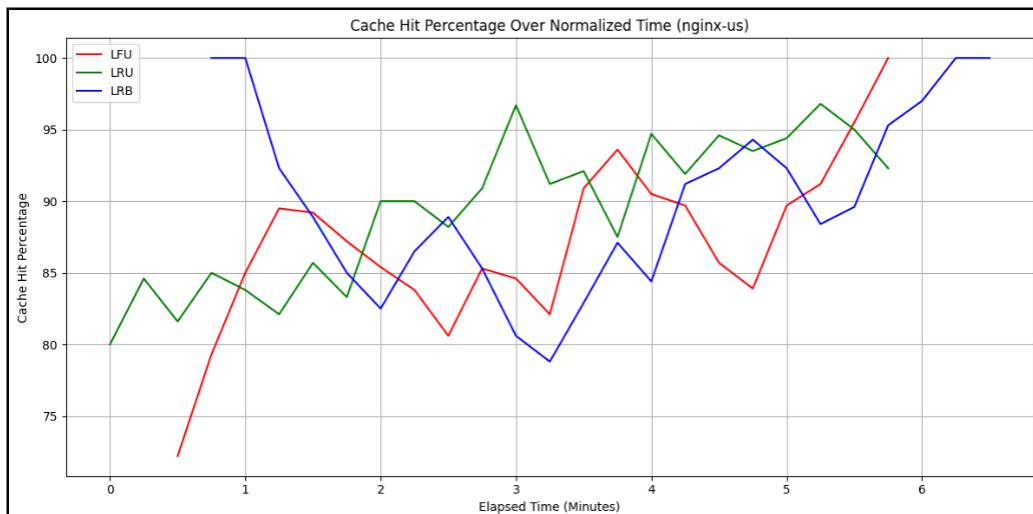
```
ml-vm > lrb-new > { cache_index_us.json > {} 60/b1/186b52fa56d3140aa054c98bf745b160
1  {
2    "60/b1/186b52fa56d3140aa054c98bf745b160": {
3      "last_access": "2025-07-27T07:20:55.434333",
4      "size": 26990108,
5      "type": "video",
6      "access_count": 5,
7      "age_since_last_access": 714.716839,
8      "age_hours": 0.1985324552777778,
9      "log_file_size": 25,
10     "access_rate": 4.171768547428425,
11     "recency_score": 4.399314012713039,
12     "stack_distance_approx": 4498351.333333333
13   },
14   "fb/7e/877fe1a102f88c6b3ca6c13e69c97efb": {
15     "last_access": "2025-07-27T07:26:54.808829",
16     "size": 17251,
17     "type": "image",
18     "access_count": 2,
19     "age_since_last_access": 355.342343,
20     "age_hours": 0.0987062063888889,
21     "log_file_size": 15,
22     "access_rate": 1.8203228382347891,
23     "recency_score": 1.870032339463494,
24     "stack_distance_approx": 5750.333333333333
25   },
26   "19/31/18853c0966a09b9caabe37e56b823119": {
27     "last_access": "2025-07-27T07:27:35.568401",
28     "size": 10824,
29     "type": "image",
30     "access_count": 7,
31     "age_since_last_access": 314.582771,
32     "age_hours": 0.08738410305555555,
33     "log_file_size": 14,
34     "access_rate": 6.43746766237428,
35     "recency_score": 6.593223819322769,
36     "stack_distance_approx": 1353.0
37   },
38 }
```

Gambar IV-27 File cache\_index.json dari Akses CDN

Gambar IV-27 menunjukkan isi dari file index json yang dipetakan berdasarkan file yang ada tersimpan di *storage* CDN dan file dari log akses pengguna di server CDN. File ini memiliki key berupa lokasi di mana file tersebut disimpan di dalam *storage* CDN. Lalu, file ini juga memiliki value-value yang dibutuhkan untuk perhitungan pada *script eviction* menggunakan ML dan disesuaikan dengan proses *training* yang telah dilakukan, seperti last\_access, size, type, access\_count, age\_since\_last\_access, age\_hours, log\_file\_size, access\_rate, recency\_score, dan stack\_distance\_approx. Value-value tersebut digunakan untuk perhitungan skor *eviction* yang dihitung berdasarkan probabilitas akses, jarak, dan probabilitas reuse konten. Semakin tinggi file tersebut maka prioritas file tersebut untuk dihapus akan semakin besar. Akan tetapi, batas penghapusan file datur berdasarkan *maximum*

*size* yang didefinisikan pada *script*, yaitu 500 MB yang ditentukan berdasarkan ukuran file dan ukuran *cache* pada *storage CDN*.

Proses pengujian dilakukan dengan urutan menghapus isi dari *storage CDN*, lalu menjalankan simulasi lalu lintas terhadap situs *dummy* yang sudah dibuat sebagai sasaran, lalu proses *eviction* dilakukan menggunakan *script* yang sudah dibuat, dan akhirnya simulasi lalu lintas dijalankan kembali untuk memeriksa dan mengevaluasi hasil *eviction* berdasarkan nilai CHR yang dipantau melalui pemantauan menggunakan Grafana. Pengujian juga dilakukan hanya di satu *node* server CDN, yaitu CDN region US karena hanya digunakan untuk melihat performa dari *eviction*. Berikut adalah hasil yang diperoleh.



Gambar IV-28 Grafik CHR dari Tiga Algoritma pada Pengujian.

Seperti yang sudah disebutkan sebelumnya, pengujian dilakukan terhadap tiga algoritma, yaitu LRB sebagai objek utama, LRU sebagai algoritma *baseline*, dan LFU sebagai algoritma pembanding ketiga. Gambar IV-28 menunjukkan hasil simulasi lalu lintas yang dijalankan setelah proses *eviction* untuk masing-masing algoritma. Berikut adalah hasil ringkasan data pengujian:

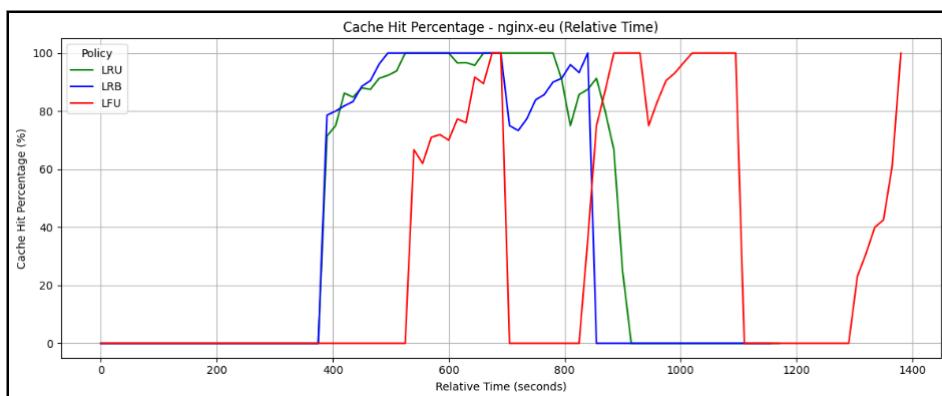
Tabel IV-3 Hasil CHR dari Pengujian Algoritma *Eviction* Distribusi Zipf dan Region US

Metrik	LFU	LRU	LRB
Rata-Rata	83.26%	85.84%	90.15%

Standar Deviasi	19.05	18.56	6.38
Min	0.00	0.00	78.80
Max	100	96.8	100

Tabel IV-3 menunjukkan data statistik dari hasil pengujian CHR untuk tiga algoritma, yaitu LRB, LFU, dan LRU. Berdasarkan tabel terlihat bahwa hasil LRB memiliki nilai yang cukup baik jika dibandingkan dengan dua algoritma lainnya dengan rata-rata CHR yang diperoleh lebih tinggi serta standar deviasi terkecil yang menandakan bahwa nilai-nilai CHR cenderung dekat dengan rata-ratanya.

Selain pengujian menggunakan simulasi lalu lintas dengan distribusi zipf, pengujian juga dilakukan dengan simulasi lalu lintas menggunakan distribusi uniform. Pengujian dengan *traffic* distribusi uniform dilakukan untuk menguji algoritma menggunakan pola *traffic* yang tidak biasa. Berikut adalah hasil pengujian dari ketiganya.



Gambar IV-29 Grafik CHR dari Tiga Algoritma pada Pengujian Region EU

Pengujian pada *traffic* dengan distribusi uniform dilakukan pada Region EU berdasarkan ketersediaan server ketika pengujian dilakukan. Meski begitu, perbedaan server tidak begitu berpengaruh ketika proses pengujian dan perbandingan antara ketiga algoritma karena ketiganya dilakukan menggunakan metode pengujian yang sama. Data statistik dari pengujian tersebut dapat dilihat pada tabel berikut.

Tabel IV-4 Hasil CHR dari Pengujian Algoritma *Eviction* Distribusi Uniform dan Region EU

<b>Metrik</b>	<b>LFU</b>	<b>LRU</b>	<b>LRB</b>
Rata-Rata	80.28%	90.32%	92.41%
Standar Deviasi	22.89	14.73	9.07
Min	23.10	25.00	73.30
Max	100	100	100

Tabel IV-4 menunjukkan hasil pengujian algoritma untuk ketiga jenis algoritma *eviction*, yaitu LFU, LRU, dan LRB. Terlihat dari ketiganya bahwa LRB masih memiliki nilai yang lebih baik dibanding dua algoritma lainnya meskipun dengan algoritma *baseline* perbedaannya tidak begitu jauh.

#### 4.2 Analisis Hasil Pengujian

Pada bagian ini diulas kembali persyaratan desain dan konsep desain, apakah terbukti dari hasil pengujian beserta interpretasinya.

##### 4.2.1. Analisis Hasil Pengujian Subsistem Pemantauan

Berdasarkan hasil pengujian dan rangkumannya pada Tabel IV-1 Spesifikasi fungsi dan pengujian subsistem pemantauan, semua komponennya telah diuji dan menunjukkan keberhasilan fungsi. Tujuan utama dari adanya subsistem pemantauan yang juga tertera pada persyaratan dan spesifikasi fungsional untuk menampilkan metrik performa sistem dan log secara visual telah tercapai. Hal ini juga berkorelasi dengan tercapainya subobjektif dan objektif sistem yaitu pemantauan sistem untuk dapat melihat metrik-metrik yang diamati untuk proyek ini.

##### 4.2.2. Analisis Hasil Pengujian Subsistem Server

Berdasarkan hasil pengujian dan rangkumannya pada Tabel IV-2 Spesifikasi fungsi dan pengujian subsistem server, semua komponennya telah diuji dan menunjukkan keberhasilan fungsi. Subsistem server mampu untuk menerima dan memproses permintaan pengguna, mengarahkan permintaan pengguna ke server CDN terdekat dan menyalin konten yang sesuai ke berbagai server sebagai *cache*. Dalam perbandingan hasil pengujian untuk sistem dengan CDN dan tanpa CDN, terlihat bahwa sistem dengan CDN terbukti unggul secara performa. Hal ini dikarenakan

letak konten yang lebih dekat secara fisik mengurangi waktu tunggu secara signifikan sekaligus mengurangi *bandwidth* internasional. Selain mengurangi latensi, CDN juga dapat mempertahankan atau bahkan meningkatkan kapasitas.

Untuk syarat pemenuhan akurasi gelokasi yang diukur berdasarkan latensi yang tidak melebihi 100 ms untuk konten web masih belum tercapai sepenuhnya dengan hasil 102.29 ms dan 25.15 ms untuk konten video. Hal ini dikarenakan oleh banyak faktor seperti keadaan jaringan internet secara umum, efisiensi kode aplikasi, infrastruktur dan beban server. Hal yang dapat dilakukan agar hasil dapat mendekati standar, diperlukan identifikasi *bottleneck* terlebih dahulu, optimasi kode dan mekanisme *caching* dan optimasi infrastruktur. Akurasi geolokasi tetap terukur dengan baik dari hasil tes *ping* dengan *ping-admin* yang menunjukkan permintaan tetap dirutekan ke server terdekat.

Hal serupa juga berlaku untuk syarat pemenuhan *troughput* minimum 100 TPS yang masih belum tercapai. Sistem dengan CDN secara umum memang meningkatkan kapasitas dari server utama karena seperti memiliki “salinan server”. Namun, ini juga bergantung pada infrastruktur server dan spesifikasi server (seperti CPU dan RAM). Spesifikasi server CDN yang digunakan adalah *e2-small tier* (0.25-2 vCPU, 2 GB *shared memory*) yang terhitung kecil.

Tujuan utama dari adanya subsistem server berkorelasi dengan subobjektif sistem yaitu pendekatan konten ke *node* yang lebih dekat ke pengguna dan berkapasitas tinggi yang juga memenuhi objektif berupa distribusi konten yang optimal. Dapat disimpulkan bahwa sistem berhasil memenuhi subobjektif dan objektif tersebut dengan baik walau belum memenuhi syarat pemenuhan.

#### **4.2.3. Analisis Hasil Pengujian Sistem Terintegrasi**

Sistem terintegrasi yang menggabungkan model ML dengan kebijakan *eviction* pada CDN berhasil melakukan proses *eviction* berdasarkan skor *eviction*. Skor *eviction* dihitung dari kombinasi *next access distance*, *reuse probability*, dan *stack distance approximation*. Hasil pengujian juga dapat terlihat pada grafik

Dari hasil pengujian tiga algoritma *eviction* pada sistem yang sudah terintegrasi, data pengujian LRB menunjukkan hasil yang lebih unggul dibandingkan ketiganya dalam melakukan *eviction* terhadap file *cache* di dalam storage CDN. Pada pengujian di region US yang disimulasikan dengan *traffic* berdistribusi zipf, LRB memiliki peningkatan rata-rata CHR terhadap LRU sebesar 5.02%, bahkan jika dibandingkan dengan LFU, CHR dari LRB memiliki peningkatan sebesar 8.27%. hal tersebut dapat terjadi karena algoritma yang digunakan oleh LRB memiliki keunggulan dalam strategi yang lebih kompleks dan informasi yang lebih banyak sehingga dapat melakukan *eviction* terhadap konten yang kurang relevan. LFU hanya melihat jumlah frekuensi akses sehingga algoritma ini sulit untuk beradaptasi ketika terdapat perubahan tren. LRU memiliki algoritma yang lebih unggul dibanding LFU karena algoritma ini melihat akses terakhir sehingga akan lebih responsif terhadap perubahan tren.

Keunggulan LRB dalam hal responsivitas terhadap perubahan tren dapat terlihat pada simulasi *traffic* di *region* EU yang menggunakan distribusi uniform dengan LRB memiliki peningkatan sebesar 2.31% terhadap LRU dan 15.11% terhadap LFU. Selain responsivitas yang lebih baik, LRB juga memiliki nilai CHR yang paling stabil di antara ketiga algoritma yang diuji. Pada pengujian US dengan distribusi zipf, LRB memiliki standar deviasi paling rendah di nilai 6.38, sedangkan algoritma lain memiliki nilai di atasnya. Sama halnya dengan pengujian EU dengan distribusi uniform yang mana meski sedikit meningkat dari sebelumnya, LRB memiliki nilai standar deviasi terendah di nilai 9.07 dibanding dua algoritma lainnya yang berada pada nilai 22.89 untuk LFU dan 14.73 untuk LRU.

## BAB V SIMPULAN DAN SARAN

Berdasarkan hasil perancangan, implementasi, dan pengujian sistem, dapat disimpulkan bahwa tujuan Tugas Akhir untuk merancang, mengimplementasikan kemudian menganalisis performa distribusi konten, ditambah kebijakan *cache eviction* pada CDN *testbed* telah berhasil dicapai. Sistem terintegrasi yang terdiri dari subsistem server berbasis NGINX, subsistem pemantauan menggunakan Prometheus dan Grafana, dan subsistem pembelajaran mesin telah berhasil diimplementasikan dan diuji secara fungsional. Hipotesis penelitian yang menyatakan bahwa *cache eviction* berbasis *machine learning* dapat meningkatkan pemanfaatan penyimpanan dan performa terbukti valid. Hal ini ditunjukkan melalui hasil pengujian dimana algoritma LRB secara konsisten menghasilkan *Cache Hit Ratio* (CHR) rata-rata yang lebih tinggi (90.15% pada distribusi Zipf dan 92.41% pada distribusi uniform) dibandingkan dengan algoritma konvensional LRU dan LFU. Hipotesis lain yang menyatakan bahwa penggunaan CDN mampu meningkatkan performa sistem seperti dalam hal meningkatkan *throughput* dan menurunkan latensi yang dialami oleh pengguna terbukti valid. Hal ini ditunjukkan oleh hasil pengujian bahwa sistem CDN terbukti mampu mengurangi latensi akses konten secara signifikan dibandingkan sistem tanpa CDN meskipun beberapa syarat pemenuhan seperti *throughput* minimum 100 TPS dan latensi di bawah 100 ms belum tercapai sepenuhnya karena keterbatasan spesifikasi infrastruktur server yang digunakan.

Berdasarkan kesimpulan dan keseluruhan proses penggerjaan Tugas Akhir, penulis memiliki beberapa saran untuk pengembangan di masa depan. Pertama, penelitian lanjutan dapat mengeksplorasi algoritma *machine learning* yang lebih kompleks, seperti LSTM yang telah dipertimbangkan sebagai alternatif, untuk membandingkan potensi peningkatan akurasi prediksi dan dampaknya terhadap CHR. Kedua, pengujian dapat dilakukan pada skala infrastruktur yang lebih besar dengan spesifikasi server yang lebih tinggi untuk memvalidasi ulang pencapaian

metrik performa seperti *throughput* dan latensi agar sesuai dengan standar industri. Terakhir, *testbed* yang telah dikembangkan ini dapat dijadikan platform dasar bagi peneliti selanjutnya untuk bereksperimen dengan berbagai strategi optimasi CDN lainnya, tidak hanya terbatas pada *cache eviction*, tetapi juga mencakup *request routing* atau *load balancing*.

## DAFTAR PUSTAKA

- Absur, M. N., Saha, S., Nova, S. N., Nasif, K. F. A., & Nasib, M. R. U. (2024). *Optimizing CDN Architectures: Multi-Metric Algorithmic Breakthroughs for Edge and Distributed Performance*. <http://arxiv.org/abs/2412.09474>
- Adler, M., Sitaraman, R. K., & Venkataramani, H. (2011). Algorithms for optimizing the bandwidth cost of content delivery. *Computer Networks*, 55(18), 4007–4020. <https://doi.org/10.1016/j.comnet.2011.07.015>
- Ali, W., Fang, C., & Khan, A. (2025). A survey on the state-of-the-art CDN architectures and future directions. *Journal of Network and Computer Applications*, 236, 104106. <https://doi.org/10.1016/J.JNCA.2025.104106>
- Fan, J., Liu, D., Tang, G., Wu, K., & Shao, X. (2024). Intelligent edge CDN with smart contract-aided local IoT sharing. *High-Confidence Computing*, 100225. <https://doi.org/10.1016/j.hcc.2024.100225>
- Faradilla A. (2023, January 19). *Apa Itu Latency? Definisi, Penyebab, dan Cara Mengatasinya*. Hostinger. <https://www.hostinger.co.id/tutorial/latency-adalah>
- Gummadi, R. (2023). *Preference learning with automated feedback for cache eviction*. <https://research.google/blog/preference-learning-with-automated-feedback-for-cache-eviction/>
- Hu, X., Ramadan, E., Ye, W., Tian, F., & Zhang, Z. L. (2022). Raven: Belady-Guided, Predictive (Deep) Learning for In-Memory and Content Caching. *CoNEXT 2022 - Proceedings of the 18th International Conference on Emerging Networking EXperiments and Technologies*, 72–90. <https://doi.org/10.1145/3555050.3569134>
- ISO. (2011). System and software quality models, . *ISO/IEC 25010*. <https://www.iso.org/standard/35733.html>

Krishna, K. (2025). Advancements in cache management: a review of machine learning innovations for enhanced performance and security. In *Frontiers in Artificial Intelligence* (Vol. 8). Frontiers Media SA. <https://doi.org/10.3389/frai.2025.1441250>

Li, H., He, L., Zhang, H., Zhang, K., Li, X., & He, C. (2020). CDN-hosted Domain Detection with Supervised Machine Learning through DNS Records. *ACM International Conference Proceeding Series*, 144–149. <https://doi.org/10.1145/3388176.3388206>

Stocker, V., Smaragdakis, G., Lehr, W., & Bauer, S. (2017). The growing complexity of content delivery networks: Challenges and implications for the Internet ecosystem. *Telecommunications Policy*, 41(10), 1003–1016. <https://doi.org/10.1016/j.telpol.2017.02.004>

Vanerio, J., Hügerich, L., & Schmid, S. (2024). Tero: Offloading CDN Traffic to Massively Distributed Devices. *ACM International Conference Proceeding Series*, 186–198. <https://doi.org/10.1145/3631461.3631556>

*What is a cache hit ratio?* (n.d.). Cloudflare. Retrieved November 7, 2024, from <https://www.cloudflare.com/learning/cdn/what-is-a-cache-hit-ratio/>

Yang, H., Pan, H., & Ma, L. (2023). A Review on Software Defined Content Delivery Network: A Novel Combination of CDN and SDN. *IEEE Access*, 11, 43822–43843. <https://doi.org/10.1109/ACCESS.2023.3267737>

