

TUGAS AKHIR – EF234801

PENGELOLAAN PENGGUNAAN INFRASTRUKTUR GPU UNTUK PENGGUNA BERBASIS DOCKER CONTAINER MENGUNAKAN JUPYTERLAB

Gloriyano Cristho Daniel Pepuho
NRP 5025201121

Dosen Pembimbing 1

Ir. Ary Mazharuddin Shiddiqi, S.Kom., M.Comp.Sc., Ph.D.
NIP 19810620 200501 1 003

Dosen Pembimbing 2

Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D.
NIP 19770824 200604 1 001

Program Studi Strata 1 (S1) Teknik Informatika

Departemen Teknik Informatika

Fakultas Teknologi Elektro dan Informatika Cerdas

Institut Teknologi Sepuluh Nopember

Surabaya

2025



TUGAS AKHIR – EF234801

**PENGELOLAAN PENGGUNAAN INFRASTRUKTUR
GPU UNTUK PENGGUNA BERBASIS DOCKER
CONTAINER MENGGUNAKAN JUPYTERLAB**

Gloriyano Cristho Daniel Pepuho

NRP 5025201121

Dosen Pembimbing 1

Ir. Ary Mazharuddin Shiddiqi, S.Kom., M.Comp.Sc., Ph.D.

NIP 19810620 200501 1 003

Dosen Pembimbing 2

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

NIP 19770824 200604 1 001

Program Studi Strata 1 (S1) Teknik Informatika

Departemen Teknik Informatika

Fakultas Teknologi Elektro dan Informatika Cerdas

Institut Teknologi Sepuluh Nopember

Surabaya

2025

[Halaman ini sengaja dikosongkan]



FINAL PROJECT - EF234801

***MANAGING DISTRIBUTED GPU INFRASTRUCTURE
USAGE FOR USERS BASED ON DOCKER
CONTAINERS USING JUPYTERLAB***

Gloriyano Cristho Daniel Pepuho

NRP 5025201121

Advisor

Ir. Ary Mazharuddin Shiddiqi, S.Kom., M.Comp.Sc., Ph.D.

NIP 19810620 200501 1 003

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

NIP 19770824 200604 1 001

Undergraduate Study Program of Department of Informatics Engineering

Department of Department of Informatics Engineering

Faculty of Intelligent Electrical and Informatics Technology

Sepuluh Nopember Institute of Technology

Surabaya

2025

[Halaman ini sengaja dikosongkan]

LEMBAR PENGESAHAN

PENGELOLAAN PENGGUNAAN INFRASTRUKTUR GPU UNTUK PENGGUNA BERBASIS DOCKER CONTAINER MENGGUNAKAN JUPYTERLAB

TUGAS AKHIR

Diajukan untuk memenuhi salah satu syarat
memperoleh gelar Sarjana Teknik pada
Program Studi S-1 Teknik Informatika
Departemen Departemen Teknik Informatika
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember

Oleh: **Gloriyano Cristho Daniel Pepuho**
NRP. 5025201121

Disetujui oleh Tim Penguji Tugas Akhir:

Ir. Ary Mazharuddin Shiddiqi, S.Kom., M.Comp.Sc., Ph.D.
NIP: 19810620 200501 1 003

(Pembimbing I)

.....

Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D.
NIP: 19770824 200604 1 001

(Pembimbing II)

.....

Prof. Tohari Ahmad, S.Kom., M.IT., Ph.D.
NIP: 19750525 200312 1 002

(Penguji I)

.....

Baskoro Adi Pratomo, S.Kom., M.Kom., Ph.D.
NIP: 19870218 201404 1 001

(Penguji II)

.....

SURABAYA
Juli, 2025

[Halaman ini sengaja dikosongkan]

APPROVAL SHEET

MANAGING DISTRIBUTED GPU INFRASTRUCTURE USAGE FOR USERS BASED ON DOCKER CONTAINERS USING JUPYTERLAB

FINAL PROJECT

Submitted to fulfill one of the requirements
for obtaining a degree Bachelor of Engineering at
Undergraduate Study Program of Department of Informatics Engineering
Department of Department of Informatics Engineering
Faculty of Intelligent Electrical and Informatics Technology
Sepuluh Nopember Institute of Technology

By: **Gloriyano Cristho Daniel Pepuho**
NRP. 5025201121

Approved by Final Project Examiner Team:

Ir. Ary Mazharuddin Shiddiqi, S.Kom., M.Comp.Sc., Ph.D.
NIP: 19810620 200501 1 003

(Advisor I)

.....

Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D.
NIP: 19770824 200604 1 001

(Co-Advisor II)

.....

Prof. Tohari Ahmad, S.Kom., M.IT., Ph.D.
NIP: 19750525 200312 1 002

(Examiner I)

.....

Baskoro Adi Pratomo, S.Kom., M.Kom., Ph.D.
NIP: 19870218 201404 1 001

(Examiner II)

.....

SURABAYA
July, 2025

[Halaman ini sengaja dikosongkan]

PERNYATAAN ORISINALITAS

Yang bertanda tangan dibawah ini:

Nama Mahasiswa / NRP : Gloriyano Cristho Daniel Pepuho / 5025201121
Departemen : Departemen Teknik Informatika
Dosen Pembimbing / NIP : Ir. Ary Mazharuddin Shiddiqi, S.Kom., M.Comp.Sc., Ph.D. /
19810620 200501 1 003

Dengan ini menyatakan bahwa Tugas Akhir dengan judul "PENGELOLAAN PENGGUNAAN INFRASTRUKTUR GPU UNTUK PENGGUNA BERBASIS DOCKER CONTAINER MENGGUNAKAN JUPYTERLAB" adalah hasil karya sendiri, berfsifat orisinal, dan ditulis dengan mengikuti kaidah penulisan ilmiah.

Bilamana di kemudian hari ditemukan ketidaksesuaian dengan pernyataan ini, maka saya bersedia menerima sanksi sesuai dengan ketentuan yang berlaku di Institut Teknologi Sepuluh Nopember.

Surabaya, July 2025

Mengetahui
Dosen Pembimbing

Mahasiswa

Ir. Ary Mazharuddin Shiddiqi, S.Kom., M.Comp.Sc., Ph.D.
NIP. 19810620 200501 1 003

Gloriyano Cristho Daniel Pepuho
NRP. 5025201121

[Halaman ini sengaja dikosongkan]

STATEMENT OF ORIGINALITY

The undersigned below:

Name of student / NRP : Gloriyano Cristho Daniel Pepuho / 5025201121
Department : Department of Informatics Engineering
Advisor / NIP : Ir. Ary Mazharuddin Shiddiqi, S.Kom., M.Comp.Sc., Ph.D. /
19810620 200501 1 003

Hereby declared that the Final Project with the title of "*MANAGING DISTRIBUTED GPU INFRASTRUCTURE USAGE FOR USERS BASED ON DOCKER CONTAINERS USING JUPYTERLAB*" is the result of my own work, is original, and is written by following the rules of scientific writing.

If in future there is a discrepancy with this statement, then I am willing to accept sanctions in accordance with provisions that apply at Sepuluh Nopember Institute of Technology.

Surabaya, July 2025

Acknowledged
Advisor

Student

Ir. Ary Mazharuddin Shiddiqi, S.Kom., M.Comp.Sc., Ph.D. Gloriyano Cristho Daniel Pepuho
NIP. 19810620 200501 1 003 NRP. 5025201121

[Halaman ini sengaja dikosongkan]

ABSTRAK

Nama Mahasiswa : Gloriyano Cristho Daniel Pepuho
Judul Tugas Akhir : PENGELOLAAN PENGGUNAAN INFRASTRUKTUR GPU UNTUK PENGGUNA BERBASIS DOCKER CONTAINER MENGGUNAKAN JUPYTERLAB
Pembimbing : 1. Ir. Ary Mazharuddin Shiddiqi, S.Kom., M.Comp.Sc., Ph.D.
2. Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D.

Dalam era teknologi yang semakin maju, kebutuhan akan komputasi berbasis GPU menjadi sangat penting, khususnya dalam bidang kecerdasan buatan (AI) dan analisis data skala besar. GPU memungkinkan pemrosesan paralel yang cepat dan efisien, sehingga sering digunakan untuk melatih model deep learning dan menjalankan tugas-tugas komputasi intensif. Namun, pengelolaan GPU di lingkungan multi-pengguna menghadapi tantangan besar, seperti alokasi sumber daya yang tidak merata dan potensi penurunan efisiensi sistem. Untuk mengatasi masalah ini, penelitian ini bertujuan untuk mengembangkan mekanisme penjadwalan GPU yang efisien dengan memanfaatkan teknologi Docker Container dan antarmuka JupyterLab. Docker digunakan untuk menciptakan lingkungan kerja yang terisolasi bagi setiap pengguna, sementara JupyterLab menyediakan platform interaktif yang memudahkan pengguna dalam mengakses dan menjalankan tugas berbasis GPU secara simultan. Penelitian ini dibagi kedalam beberapa tahap yang meliputi analisis kebutuhan, desain sistem, serta perancangan metode evaluasi. Rancangan sistem yang diusulkan akan diimplementasikan pada kluster GPU di lingkungan laboratorium atau institusi pendidikan. Evaluasi direncanakan mencakup pengujian efisiensi alokasi sumber daya, kemudahan akses pengguna, dan skalabilitas sistem dalam mendukung banyak pengguna secara bersamaan. Penelitian ini diharapkan dapat memberikan kontribusi terhadap pengelolaan sumber daya GPU dalam lingkungan komputasi terdistribusi, mendukung efisiensi dan keadilan alokasi, serta meningkatkan pengalaman pengguna dalam mengakses sumber daya GPU untuk kebutuhan komputasi modern.

Kata Kunci: *Kluster GPU, Docker Container, JupyterLab, Pengelolaan pengguna*

[Halaman ini sengaja dikosongkan]

ABSTRACT

Name : Gloriyano Cristho Daniel Pepuho
Title : *MANAGING DISTRIBUTED GPU INFRASTRUCTURE USAGE FOR USERS
BASED ON DOCKER CONTAINERS USING JUPYTERLAB*
Advisors : 1. Ir. Ary Mazharuddin Shiddiqi, S.Kom., M.Comp.Sc., Ph.D.
2. Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D.

In the era of advancing technology, the demand for GPU-based computing has become increasingly critical, particularly in the fields of artificial intelligence (AI) and large-scale data analysis. GPUs enable fast and efficient parallel processing, making them widely used for training deep learning models and performing computationally intensive tasks. However, managing GPUs in multi-user environments presents significant challenges, such as uneven resource allocation and potential system inefficiencies. To address these issues, this study aims to develop an efficient GPU scheduling mechanism utilizing Docker container technology and the JupyterLab interface. Docker creates isolated work environments for each user, while JupyterLab provides an interactive platform that simplifies simultaneous GPU-based task execution. The research consists of several phases, including requirement analysis, system design, and evaluation method planning. The proposed system design will be implemented on a GPU cluster in a laboratory or educational institution environment. Evaluation will include testing resource allocation efficiency, user accessibility, and system scalability in supporting multiple concurrent users. This study is expected to make a significant contribution to GPU resource management in distributed computing environments, promoting efficiency and fairness in resource allocation while enhancing the user experience in accessing GPU resources for modern computational needs.

Keywords: *GPU Cluster, Docker Container, JupyterLab, User Management*

[Halaman ini sengaja dikosongkan]

KATA PENGANTAR

Puji dan syukur kehadiran Tuhan Yang Maha Esa yang memberikan karunia, rahmat, dan pertolongan sehingga penulis dapat menyelesaikan penelitian tugas akhir yang berjudul 'PENGELOLAAN PENGGUNAAN INFRASTRUKTUR GPU UNTUK PENGGUNA BERBASIS DOCKER CONTAINER MENGGUNAKAN JUPYTERLAB'. Melalui kata pengantar ini, penulis mengucapkan terima kasih sebesar-besarnya kepada seluruh pihak yang telah membantu dan mendukung penulis selama mengerjakan penelitian tugas akhir ini, diantaranya adalah:

1. Tuhan Yang Maha Esa, atas karunia dan rahmat-Nya sehingga penulis dapat mencapai titik akhir perkuliahan strata satu di Departemen Teknik Informatika, Institut Teknologi Sepuluh Nopember.
2. Kedua orang tua yang telah mendukung penulis selama berkuliah di Departemen Teknik Informatika, Institut Teknologi Sepuluh Nopember.
3. Bapak Ir. Ary Mazharuddin Shiddiqi, S.Kom., M.Comp.Sc., Ph.D. dan Bapak Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D. sebagai dosen pembimbing yang telah membimbing, memberi arahan, dan masukan kepada penulis selama mengerjakan tugas akhir ini.
4. Dosen dan tenaga pendidik di Departemen Teknik Informatika, Institut Teknologi Sepuluh Nopember yang telah memberikan pengetahuan, wawasan, dan pengalaman yang sangat berarti selama masa studi.
5. Pihak-pihak lain yang tidak dapat disebutkan satu persatu yang telah membantu penulis dalam pelaksanaan penelitian tugas akhir ini.

Akhir kata, semoga penelitian tugas akhir ini dapat memberikan kontribusi yang bermanfaat. Terima kasih dan permohonan maaf atas kekurangan dan kesalahan dalam pelaksanaan tugas akhir ini.

Surabaya, Juli 2025

Gloriyano Cristho Daniel Pepuho

[Halaman ini sengaja dikosongkan]

DAFTAR ISI

ABSTRAK	i
ABSTRACT	iii
KATA PENGANTAR	v
DAFTAR ISI	vii
DAFTAR GAMBAR	ix
DAFTAR TABEL	xi
DAFTAR KODE SUMBER	xiii
DAFTAR SINGKATAN	xv
1 PENDAHULUAN	1
1.1 Latar Belakang	1
1.2 Rumusan Masalah	2
1.3 Batasan Masalah atau Ruang Lingkup	2
1.4 Tujuan	2
1.5 Manfaat	2
1.6 Sistematika Penulisan	3
2 DESAIN DAN IMPLEMENTASI	5
2.1 Perancangan Arsitektur Sistem	6
2.1.1 Service Discovery	8
2.1.2 Service Agent	8
2.1.3 JupyterHub	8
2.1.4 Ray Cluster	8
2.2 Implementasi Sistem	9
2.2.1 Service Discovery	9
2.2.2 Service Agent	21
2.2.3 JupyterHub	26

2.2.4	Ray Cluster	30
2.3	Peralatan Pendukung	31
3	PENGUJIAN DAN ANALISIS	33
3.1	Hasil dan Pengujian	33
3.1.1	Uji Akses dan Proses Spawn Server	33
3.1.2	Skenario 2: Multi-User Concurrent	36
4	PENUTUP	37
4.1	Kesimpulan	37
4.2	Saran	37
	DAFTAR PUSTAKA	39
	BIOGRAFI PENULIS	41

DAFTAR GAMBAR

2.1	Arsitektur Penelitian	7
2.2	Hasil build agent menjadi image	24
2.3	Log dari container agent ketika melakukan registrasi node	25
3.1	Proses Registrasi dan Login pada JupyterHub	34
3.2	User memilih profil dan environment sesuai kebutuhan pada halaman konfigurasi JupyterHub	34
3.3	Tahapan proses spawning container setelah konfigurasi dilakukan oleh user	35
3.4	JupyterLab berhasil dijalankan di lingkungan multi-node hingga mencapai tampilan akhir yang siap digunakan	35

[Halaman ini sengaja dikosongkan]

DAFTAR TABEL

2.1	Struktur Direktori Discovery Service	9
2.2	Kolom dan Tipe Data Model Node	12
2.3	Kolom dan Tipe Data Model NodeSelection	13
2.4	Kolom dan Tipe Data Model NodeMetric	13
2.5	Contoh Isi File <code>.env</code> dari <i>Discovery Service</i>	17
2.6	Daftar Endpoint REST API pada Discovery Service	20
2.7	Struktur Direktori Proyek JupyterHub	26
2.8	Spesifikasi Peralatan Pendukung	31
2.9	Daftar Perangkat Lunak Pendukung	32

[Halaman ini sengaja dikosongkan]

DAFTAR KODE SUMBER

2.1	Pseudocode untuk <code>app.py</code>	10
2.2	Pseudocode untuk Model Node	12
2.3	Pseudocode untuk Model NodeSelection	12
2.4	Pseudocode untuk Model NodeMetric	13
2.5	Pseudocode untuk Koneksi Redis Menggunakan ConnectionPool	14
2.6	Pseudocode Penyimpanan Data Node ke Redis dengan TTL	14
2.7	Pseudocode Fungsi Perhitungan Skor Node	15
2.8	Pseudocode untuk <code>config.py</code>	15
2.9	Pseudocode Proses Pembangunan Image Docker	18
2.10	Pseudocode untuk <code>docker-compose.yml</code>	18
2.11	Menjalankan Discovery Service via Docker Compose	20
2.12	Pseudocode Fungsi Register Agent	21
2.13	Pseudocode Kumpulan Informasi Sistem oleh Agent	22
2.14	Deteksi Container JupyterLab dan Ray	22
2.15	Deteksi GPU Menggunakan <code>gpustat</code>	22
2.16	Pseudocode Loop Registrasi Agent Tiap 15 Detik	23
2.17	Pseudocode untuk Dockerfile Agent Service	24
2.18	Perintah untuk Build dan Menjalankan Agent	24
2.19	Perintah untuk Build dan Menjalankan Agent	24
2.20	Perintah untuk Menjalankan Agent di setiap Node	25
2.21	<code>jupyterhub_config.py</code>	27
2.22	Pemanggilan API Seleksi Node	27
2.23	Inisialisasi Docker Client Berdasarkan IP Node	28
2.24	Pembuatan Container Ray Worker di Node Tambahan	28
2.25	Eksekusi Paralel untuk Container Jupyter dan Worker	28
2.26	Override URL pada PatchedMultiNodeSpawner	29

[Halaman ini sengaja dikosongkan]

DAFTAR SINGKATAN

API	Application Programming Interface
CPU	Central Processing Unit
CLI	Command Line Interface
GPU	Graphics Processing Unit
GUI	Graphical User Interface
JEG	Jupyter Enterprise Gateway
PAAS	Platform as a Service

[Halaman ini sengaja dikosongkan]

BAB I

PENDAHULUAN

1.1 Latar Belakang

Peningkatan kebutuhan komputasi untuk machine learning (ML) dan kecerdasan buatan (AI) telah mendorong penggunaan Graphics Processing Unit (GPU) secara eksponensial. Urgensi pergeseran ini ditegaskan oleh CEO NVIDIA, Jensen Huang, dalam pidato utamanya di NVIDIA GPU Technology Conference (GTC) 2023, di mana ia menyatakan, "**The starting point of the new world is a new type of computer, a new computing model... This is the accelerated computing model.**" Kutipan ini menggaris bawahi bahwa era komputasi modern menuntut arsitektur baru di mana GPU menjadi elemen krusial berkat kemampuannya dalam pemrosesan paralel yang cepat dan efisien untuk *training model deep learning*. Namun, seiring meningkatnya kebutuhan ini, muncul tantangan baru dalam pengelolaannya.

Salah satu tantangan utama adalah tingginya biaya atau *cost* dalam menginvestasi dan pemeliharaan infrastruktur GPU. Bagi banyak institusi, terutama di lingkungan pendidikan dan riset, pengadaan GPU dalam jumlah besar seringkali tidak memungkinkan. Di sisi lain, tidak jarang komputer dengan GPU yang sudah ada baik di laboratorium atau milik perorangan tidak dimanfaatkan secara optimal dan seringkali dalam keadaan *idle*. Untuk memaksimalkan potensi sumber daya yang ada, muncul kebutuhan untuk memanfaatkan infrastruktur GPU yang tersebar ini secara kolektif.

Untuk menjawab tantangan tersebut, pengelolaan sumber daya dalam lingkungan multi-pengguna menjadi sangat penting. Pengelolaan yang tidak efisien dapat menyebabkan alokasi sumber daya yang tidak merata dan penurunan kinerja sistem. Diperlukan sebuah mekanisme alokasi dan penjadwalan sumber daya (*resource allocation and scheduling*) yang dinamis, yang dapat memastikan bahwa setiap pengguna mendapatkan akses yang adil dan efisien. Pendekatan penjadwalan tradisional seringkali tidak cukup untuk menangani karakteristik unik dari beban kerja AI.

Dalam tugas akhir ini, dikembangkan sebuah sistem untuk mengelola penggunaan infrastruktur GPU yang tersebar bagi banyak pengguna. Sistem ini memanfaatkan teknologi containerisasi menggunakan Docker untuk menciptakan lingkungan kerja yang terisolasi dan konsisten. Sebagai antarmuka utama, JupyterLab digunakan untuk menyediakan platform yang interaktif dan mudah diakses, memungkinkan pengguna menjalankan tugas komputasi tanpa perlu konfigurasi yang rumit.

Dengan demikian, penelitian ini berfokus pada pengembangan mekanisme penjadwalan GPU yang efektif dalam lingkungan terdistribusi. Sistem yang diusulkan dirancang untuk dapat mengelola dan mendistribusikan beban kerja secara dinamis ke node-node yang tersedia, termasuk yang sebelumnya dalam keadaan idle, guna mendukung kebutuhan komputasi AI modern yang semakin kompleks dan kolaboratif.

1.2 Rumusan Masalah

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

1. Bagaimana arsitektur integrasi antara JupyterHub dan service discovery dirancang untuk memungkinkan setiap instance JupyterLab dapat memanfaatkan sumber daya komputasi (GPU dan CPU) yang tersebar di berbagai *node*?
2. Bagaimana sistem menangani dan mendistribusikan antrian permintaan dari banyak pengguna yang masuk secara bersamaan (*concurrent requests*), dan bagaimana pengaruhnya terhadap alokasi sumber daya pada setiap node?

1.3 Batasan Masalah atau Ruang Lingkup

Batasan dalam pengerjaan tugas akhir ini adalah sebagai berikut:

1. Infrastruktur yang digunakan adalah komputer yang tersedia di lingkungan laboratorium atau institusi pendidikan, bukan *cloud platform* berskala besar
2. Implementasi sistem berfokus pada integrasi JupyterHub dengan Docker, serta pengembangan mekanisme *service discovery* untuk penjadwalan dinamis. Arsitektur ini tidak mencakup perbandingan mendalam dengan orkestrator lain seperti Kubernetes.
3. Algoritma penjadwalan (scheduling) yang dikembangkan berfokus pada implementasi satu model fungsional (misalnya, berbasis skor beban atau round-robin) dan tidak melakukan analisis komparatif terhadap semua kemungkinan algoritma penjadwalan.

1.4 Tujuan

Tujuan dari pembuatan Tugas Akhir ini adalah sebagai berikut:

1. Merancang dan mengimplementasikan sebuah mekanisme penjadwalan sumber daya (resource scheduling) dinamis yang dapat mengalokasikan kontainer Docker ke node dengan beban paling optimal dalam kluster.
2. Membangun arsitektur sistem yang mengintegrasikan JupyterHub dengan service discovery, sehingga setiap pengguna dapat secara mudah memperoleh lingkungan kerja (instance JupyterLab) yang berjalan di atas sumber daya terdistribusi.
3. Menganalisis dan mengevaluasi kinerja sistem dalam skenario multi-pengguna, terutama dalam hal manajemen antrian, waktu eksekusi tugas, dan efisiensi alokasi sumber daya.

1.5 Manfaat

Manfaat dari penelitian ini adalah sebagai berikut:

1. Sistem ini dapat meningkatkan efisiensi pemanfaatan infrastruktur GPU yang terbatas, mendukung penelitian, dan pengembangan berbasis komputasi AI.
2. Memberikan akses yang mudah, terstruktur, dan terisolasi ke sumber daya GPU tanpa memerlukan pengetahuan teknis mendalam tentang manajemen server atau Docker.

1.6 Sistematika Penulisan

Laporan penelitian tugas akhir ini terbagi menjadi:

1. **BAB I Pendahuluan**

Bab ini berisi latar belakang penelitian yang menjelaskan pentingnya pengelolaan infrastruktur GPU terdistribusi, rumusan masalah yang dihadapi dalam penggunaan GPU multi-user, batasan masalah dan ruang lingkup penelitian, tujuan yang ingin dicapai, manfaat penelitian, serta sistematika penulisan laporan.

2. **BAB II Tinjauan Pustaka**

Bab ini berisi tinjauan terhadap penelitian-penelitian terdahulu yang relevan dengan topik penelitian, teori dan konsep dasar yang meliputi kluster GPU, teknologi Docker, Ray Framework, penjadwalan GPU, JupyterLab, dan JupyterHub. Bab ini menjadi landasan teoritis untuk melakukan pengembangan sistem.

3. **BAB III Desain dan Implementasi Sistem**

Bab ini berisi perancangan arsitektur sistem yang mencakup service discovery, integrasi JupyterHub, dan konfigurasi Ray cluster. Selain itu bab ini membahas peralatan apa saja yang digunakan pada saat penelitian serta setiap detail implementasi komponen yang dikembangkan.

4. **BAB IV Pengujian dan Analisa**

Bab ini berisi bab ini dirancang untuk memvalidasi fungsionalitas sistem, evaluasi perform dalam berbagai kondisi beban, analisis efisiensi penggunaan resource, serta pembahasan hasil pengujian terhadap tujuan penelitian yang telah ditetapkan

5. **BAB V Penutup**

Bab ini berisi kesimpulan dari penelitian yang merangkum pencapaian tujuan penelitian, kontribusi yang diberikan, serta saran untuk pengembangan dan penelitian lebih lanjut yang dapat dilakukan berdasarkan penelitian ini.

[Halaman ini sengaja dikosongkan]

BAB II

DESAIN DAN IMPLEMENTASI

Bab ini menjelaskan perancangan dan implementasi sistem pengelolaan sumber daya GPU secara terdistribusi menggunakan container Docker, JupyterHub, dan Ray. Sistem dirancang untuk mendukung penggunaan secara multi-pengguna dengan penjadwalan node berbasis beban kerja dan integrasi antarkomponen melalui Discovery Service.

Pembahasan pada bab ini meliputi arsitektur sistem secara keseluruhan, implementasi masing-masing komponen utama, serta perangkat lunak pendukung yang digunakan selama proses pengembangan.

2.1 Perancangan Arsitektur Sistem

Penelitian ini diawali dengan proses perancangan sistem yang bertujuan untuk memungkinkan pengelolaan sumber daya GPU secara efisien dan adil bagi banyak pengguna. Sistem dikembangkan untuk dapat berjalan dalam lingkungan terdistribusi dengan infrastruktur multi-node berbasis container Docker. Untuk itu, dibutuhkan arsitektur yang mampu mengintegrasikan manajemen autentikasi pengguna, alokasi kontainer secara dinamis, serta orkestrasi workload komputasi berbasis GPU maupun CPU.

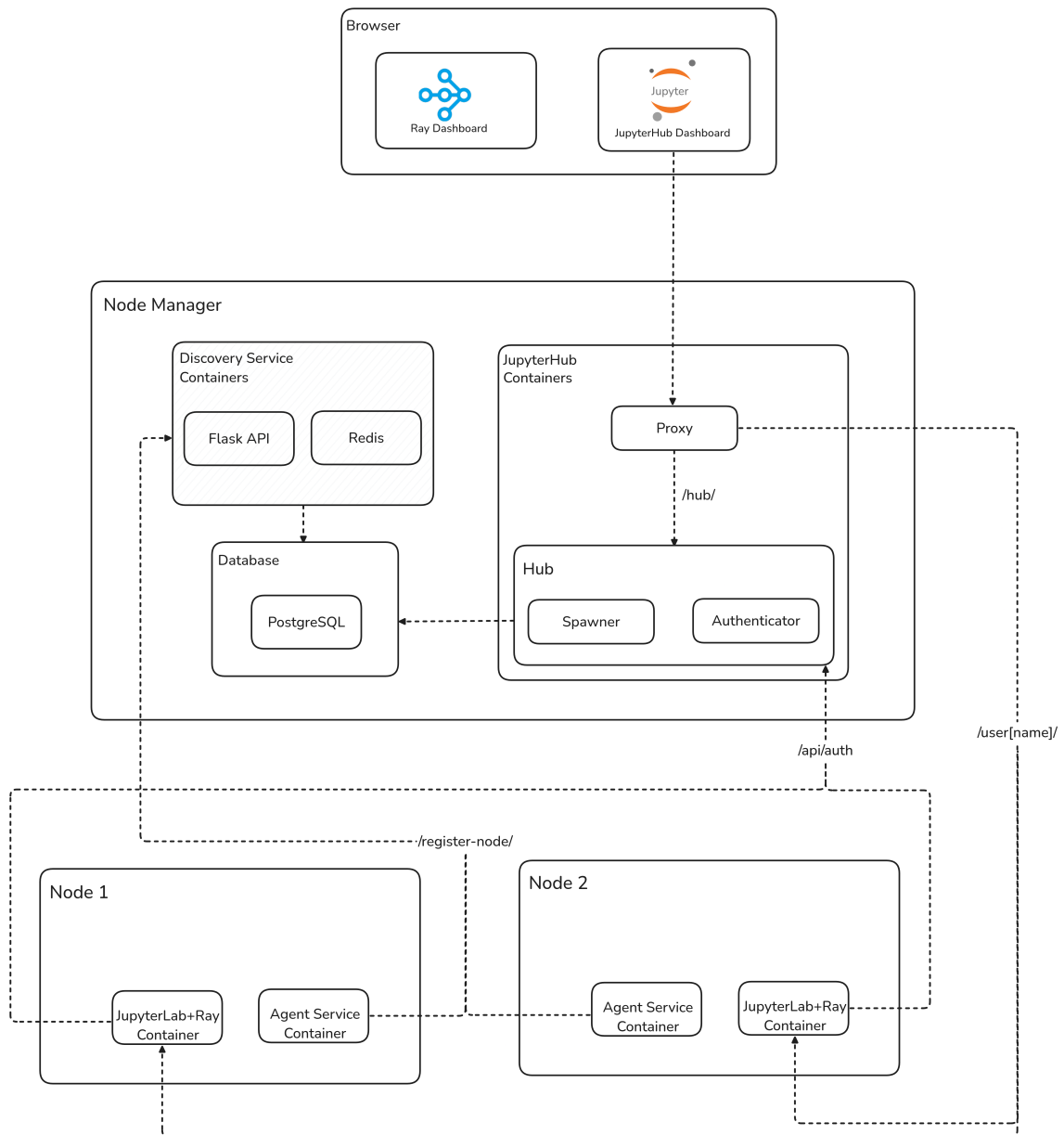
Langkah awal dalam perancangan adalah merancang sistem service discovery yang berfungsi sebagai pusat pengumpulan dan penyimpanan status sumber daya dari seluruh node yang tersedia dalam kluster. Data ini mencakup informasi penggunaan CPU, RAM, keberadaan GPU, serta jumlah container aktif, yang dikirim secara periodik oleh agent service dari masing-masing node. Informasi ini disimpan ke dalam basis data in-memory Redis yang akan digunakan untuk mendukung pengambilan keputusan secara real-time saat pemilihan node dilakukan.

Selanjutnya, dilakukan integrasi antara JupyterHub sebagai antarmuka utama pengguna dan DockerSpawner yang telah dimodifikasi untuk mendukung penjadwalan multi-node. Dengan adanya integrasi ini, setiap permintaan pengguna akan diproses melalui skema load balancing, yang kemudian menentukan node terbaik untuk menjalankan *environment* JupyterLab. Skor dihitung berdasarkan tingkat utilisasi CPU, RAM, dan jumlah container aktif. Pemilihan node dilakukan secara dinamis dan adaptif, bergantung pada profil sumber daya yang dibutuhkan oleh masing-masing pengguna.

Dalam *environment* kontainer yang di-spawn, pengguna akan mendapatkan akses ke antarmuka JupyterLab secara personal dan terisolasi. Di dalam container tersebut, Ray Worker dijalankan secara otomatis dan terhubung ke Ray Head Node. Dengan demikian, setiap pengguna dapat langsung menjalankan komputasi paralel menggunakan framework Ray tanpa memerlukan konfigurasi manual tambahan.

Untuk mengimplementasikan arsitektur tersebut, dilakukan konfigurasi Discovery Service, Agent Service, JupyterHub, dan Ray secara terintegrasi. Masing-masing komponen dikemas dalam Docker container untuk memudahkan deployment dan orkestrasi antar layanan. Komunikasi antar komponen dilakukan melalui protokol HTTP REST API dan jaringan internal antar container.

Gambar 3.1 di bawah ini menggambarkan secara keseluruhan hubungan antar komponen sistem. Diagram tersebut memperlihatkan arsitektur sistem yang dirancang, mulai dari proses pelaporan status node oleh Agent Service, pemrosesan dan pengambilan keputusan di Discovery Service, hingga peluncuran kontainer pengguna di JupyterHub dan orkestrasi komputasi dengan Ray.



Gambar 2.1: Arsitektur Penelitian

Adapun komponen utama yang membentuk arsitektur sistem ini terdiri atas:

2.1.1 Service Discovery

Service discovery bertugas sebagai pusat informasi status terkini dari setiap node dalam kluster. Informasi yang dikumpulkan meliputi status CPU, RAM, GPU, dan jumlah container aktif. Data ini dikumpulkan secara periodik oleh agen yang berjalan di setiap node dan dikirim ke Redis melalui REST API Flask. Redis berfungsi sebagai basis data cepat (in-memory) untuk mendukung pengambilan keputusan real-time saat pemilihan node terbaik untuk menjalankan JupyterLab user.

2.1.2 Service Agent

Service Agent merupakan komponen ringan yang berjalan secara periodik di setiap node dalam kluster. Tugas utama Agent adalah memantau kondisi sistem secara lokal, kemudian mengirimkan informasi tersebut ke Discovery Service melalui endpoint **/register-node**. Informasi yang dikirim mencakup:

- Informasi jumlah CPU, kapasitas memori dan penyimpanan, serta tingkat penggunaan (usage) masing-masing sumber daya secara real-time.
- Deteksi keberadaan GPU (terutama NVIDIA) beserta detail spesifikasinya seperti kapasitas memori dan tingkat utilisasi.
- Informasi tambahan seperti *hostname*, alamat IP, dan metadata node lainnya.

Agent dirancang dalam bentuk container mandiri berbasis Python yang berjalan otomatis sejak node aktif. Dengan mengandalkan pustaka seperti `psutil`, `gpustat`, dan Docker SDK, Agent mampu menangkap informasi sistem secara akurat. Interval pengiriman data diatur setiap 15 detik agar status node tetap mutakhir di Redis tanpa membebani sumber daya secara signifikan.

Komponen ini sangat krusial dalam menjaga keakuratan data load balancing, karena JupyterHub akan memilih node berdasarkan data yang dikumpulkan oleh Agent. Dengan adanya Agent, sistem dapat secara otomatis mengetahui jika suatu node mengalami kelebihan beban, tidak tersedia, atau sedang dalam kondisi idle.

2.1.3 JupyterHub

JupyterHub bertindak sebagai sistem autentikasi dan pengelola sesi pengguna. Setiap pengguna dapat memulai server JupyterLab pribadi yang dijalankan sebagai container terisolasi. Dengan bantuan spawner khusus yang terintegrasi dengan discovery service, JupyterHub akan secara otomatis memilih node dengan resource teringan. Spawner ini juga bertugas melakukan konfigurasi container secara otomatis, termasuk setting IP, port, dan image sesuai kebutuhan pengguna.

2.1.4 Ray Cluster

Ray digunakan untuk mengatur workload komputasi paralel. Dalam perancangannya, setiap container JupyterLab pengguna akan menjalankan Ray Worker secara otomatis dan terhubung ke Ray Head Node. Dengan cara ini, pengguna dapat langsung menggunakan fitur komputasi terdistribusi seperti `ray.remote()` tanpa konfigurasi manual. Ray menjembatani antar-node agar task berat bisa dijalankan dengan efisien di GPU atau CPU sesuai kapasitas.

2.2 Implementasi Sistem

Implementasi sistem terdiri dari beberapa komponen utama yang saling terintegrasi, sebagai berikut:

2.2.1 Service Discovery

Discovery Service merupakan komponen pusat dalam sistem yang bertugas menerima, menyimpan, dan menyediakan informasi status sumber daya dari setiap node GPU. Layanan ini dibangun dengan framework Flask REST API dan mengimplementasikan pendekatan penyimpanan hybrid menggunakan Redis dan PostgreSQL. Redis digunakan untuk data real-time dengan TTL (time-to-live), sedangkan PostgreSQL menyimpan data historis dan metadata yang lebih persisten.

Pada bagian ini akan dijelaskan struktur proyek, konfigurasi sistem, serta implementasi fitur-fitur utama dari layanan Discovery Service secara teknis.

A. Arsitektur Aplikasi

Struktur proyek Discovery Service disusun secara modular dengan pembagian tanggung jawab yang jelas. Tabel 3.1 menjelaskan file dan direktori utama:

Tabel 2.1: Struktur Direktori Discovery Service

Nama File/Folder	Deskripsi
app.py	Titik masuk aplikasi Flask.
config.py	Konfigurasi environment dan database.
redis_client.py	Utilitas koneksi Redis.
Dockerfile	Definisi image Docker.
docker-compose.yml	Orkestrasi Redis, PostgreSQL, dan Flask.
init.sql	Skrip inisialisasi database PostgreSQL.
redis.conf	Konfigurasi Redis kustom.
models/	ORM SQLAlchemy untuk Node, Profile, dsb.
routes/	Endpoint API untuk node dan profile.
services/	Logika bisnis seperti pendaftaran node.
utils/	Load balancer dan skoring node.

B. Inisialisasi Proyek dan Registrasi Layanan

Selain menginisialisasi konfigurasi dasar seperti CORS, database, dan blueprint, file *app.py* juga mencakup integrasi awal dengan basis data PostgreSQL dan Redis. Salah satu endpoint bawaan adalah */health-check* yang digunakan untuk memastikan apakah API telah berjalan dan memverifikasi status koneksi ke kedua database tersebut secara real-time. Redis digunakan melalui kelas *RedisService* untuk pengecekan konektivitas dan pengelolaan data status node yang volatile.

Selain itu, fungsi *run_periodic_task* digunakan untuk menjalankan tugas latar belakang yang membersihkan node-node yang tidak aktif berdasarkan data dari Redis. Hal ini meningkatkan reliabilitas data yang tersimpan dan mengurangi beban layanan.

```
1 FUNCTION create_app():
2     INITIALIZE Flask app
3     LOAD configuration from Config
4     INITIALIZE extensions:
5         - CORS
6         - SQLAlchemy database
7         - Flask Migrate
8
9     REGISTER blueprints:
10        - node_bp
11        - profile_bp
12
13    DEFINE route "/health-check":
14        RETURN JSON status (ok) with database connection status
15
16    WITH app context:
17        CREATE all database tables
18        INITIALIZE default profiles
19
20    RETURN app
21 END FUNCTION
22
23 FUNCTION run_periodic_tasks(app):
24    START background thread:
25        WHILE True:
26            CALL mark_nodes_inactive() to cleanup inactive nodes
27            SLEEP 300 seconds
28 END FUNCTION
29
30 IF __name__ == '__main__':
31     app = create_app()
32     CALL run_periodic_tasks(app)
33     RUN app on host 0.0.0.0 port 15002 (debug mode)
34 END IF
```

Kode Sumber 2.1: Pseudocode untuk *app.py*

C. Integrasi dengan Basis Data

Discovery Service menggunakan dua jenis sistem basis data untuk mendukung performa dan keandalan layanan: **PostgreSQL** sebagai basis data relasional permanen dan **Redis** sebagai penyimpanan data sementara (in-memory) untuk status sistem.

PostgreSQL diakses melalui ORM SQLAlchemy yang telah terhubung di dalam file `app.py` menggunakan `db.init_app(app)`. Tabel-tabel utama yang dimodelkan dalam sistem ini antara lain:

- **Node:** Menyimpan informasi node seperti hostname, kapasitas CPU, RAM, dan keberadaan GPU.
- **NodeMetric:** Menyimpan riwayat pemantauan beban node seperti penggunaan CPU, memori, jumlah kontainer aktif, dan skoring beban.
- **Profile:** Mendefinisikan konfigurasi profil pengguna yang menentukan kebutuhan resource.
- **NodeSelection:** Mencatat hasil seleksi node berdasarkan profil dan pengguna.

Semua model didefinisikan secara modular dalam direktori `models/`. Skema database dapat diinisialisasi dan dimigrasi menggunakan `Flask-Migrate`.

```

1 CLASS Node:
2     ATTRIBUTE id : Integer
3     ATTRIBUTE hostname : String(255)
4     ATTRIBUTE ip : String(45)
5     ATTRIBUTE cpu_cores : Integer
6     ATTRIBUTE ram_gb : Float
7     ATTRIBUTE has_gpu : Boolean = False
8     ATTRIBUTE gpu_info : JSON
9     ATTRIBUTE is_active : Boolean
10    ATTRIBUTE max_containers : Integer
11    ATTRIBUTE created_at : Datetime
12    ATTRIBUTE updated_at : Datetime
13    ATTRIBUTE last_heartbeat : Datetime
14 END CLASS

```

Kode Sumber 2.2: Pseudocode untuk Model Node

Tabel 2.2: Kolom dan Tipe Data Model Node

Nama Kolom	Tipe Data
id	INTEGER (PRIMARY KEY)
hostname	STRING(255)
ip	STRING(45)
cpu_cores	INTEGER
ram_gb	FLOAT
has_gpu	BOOLEAN
gpu_info	JSON
is_active	BOOLEAN
max_containers	INTEGER
created_at	DATETIME
updated_at	DATETIME
last_heartbeat	DATETIME

```

1 CLASS NodeSelection:
2     ATTRIBUTE id : Integer
3     ATTRIBUTE profile_id : Integer
4     ATTRIBUTE user_id : String(255)
5     ATTRIBUTE session_id : String(255)
6     ATTRIBUTE selected_nodes : JSON
7     ATTRIBUTE selection_reason : String(50)
8     ATTRIBUTE created_at : Datetime
9 END CLASS

```

Kode Sumber 2.3: Pseudocode untuk Model NodeSelection

Tabel 2.3: Kolom dan Tipe Data Model NodeSelection

Nama Kolom	Tipe Data
id	INTEGER (PRIMARY KEY)
profile_id	INTEGER
user_id	STRING(255)
session_id	STRING(255)
selected_nodes	JSON
selection_reason	STRING(50)
created_at	DATETIME

```

1 CLASS NodeMetric:
2     ATTRIBUTE id : Integer
3     ATTRIBUTE node_id : Integer
4     ATTRIBUTE cpu_usage_percent : Float
5     ATTRIBUTE memory_usage_percent : Float
6     ATTRIBUTE disk_usage_percent : Float
7     ATTRIBUTE active_jupyterlab : Integer
8     ATTRIBUTE active_ray : Integer
9     ATTRIBUTE total_containers : Integer
10    ATTRIBUTE load_score : Float
11    ATTRIBUTE recorded_at : Datetime
12 END CLASS

```

Kode Sumber 2.4: Pseudocode untuk Model NodeMetric

Tabel 2.4: Kolom dan Tipe Data Model NodeMetric

Nama Kolom	Tipe Data
id	INTEGER (PRIMARY KEY)
node_id	INTEGER
cpu_usage_percent	FLOAT
memory_usage_percent	FLOAT
disk_usage_percent	FLOAT
active_jupyterlab	INTEGER
active_ray	INTEGER
total_containers	INTEGER
load_score	FLOAT
recorded_at	DATETIME

Redis digunakan untuk menyimpan status terkini node yang dilaporkan oleh agent secara periodik. Redis ini tidak menyimpan data permanen, tetapi digunakan untuk:

- Menyimpan metrik real-time seperti CPU, RAM, dan disk usage.
- Menentukan apakah node masih aktif berdasarkan heartbeat agent.

Koneksi ke Redis dilakukan melalui file `redis_client.py` menggunakan *connection pool* untuk efisiensi koneksi. File ini diakses melalui kelas `RedisService` pada `services/redis_service`

```
1 SET REDIS_HOST = getenv("REDIS_HOST", "localhost")
2 SET REDIS_PORT = getenv("REDIS_PORT", 6379) AS Integer
3 SET REDIS_PASSWORD = getenv("REDIS_PASSWORD")
4 SET REDIS_EXPIRE_SECONDS = getenv("REDIS_EXPIRE_SECONDS", 45) AS Integer
5
6 INITIALIZE ConnectionPool with:
7     host = REDIS_HOST
8     port = REDIS_PORT
9     password = REDIS_PASSWORD
10    decode_responses = True
11
12 INITIALIZE RedisClient with ConnectionPool
```

Kode Sumber 2.5: Pseudocode untuk Koneksi Redis Menggunakan ConnectionPool

Agent akan mengirimkan data dalam interval tertentu, dan informasi tersebut disimpan sementara dalam Redis menggunakan TTL selama 45 detik. Berikut contoh potongan penyimpanan data pada saat registrasi node:

```
1 CALL RedisClient.set(
2     key = "node:{hostname}:info",
3     value = JSON.stringify(data),
4     expire = Config.REDIS_EXPIRE_SECONDS
5 )
```

Kode Sumber 2.6: Pseudocode Penyimpanan Data Node ke Redis dengan TTL

D. Seleksi Node

Discovery Service menggunakan pendekatan modular dalam proses seleksi node, yang diimplementasikan dalam file `load_balancer.py` pada direktori *utils/*. Pemilihan node dilakukan berdasarkan algoritma yang dapat disesuaikan, seperti *round robin*, *best fit*, dan *random selection*, dengan *round robin* sebagai metode default untuk mendistribusikan beban kerja antar node secara merata.

Sebelum pemilihan dilakukan, setiap node dihitung nilai beban-nya melalui fungsi `calculate_node_score` yang berada pada file `scoring.py`. Fungsi ini menghitung skor berdasarkan kombinasi tingkat utilisasi CPU dan memori. Node yang melebihi ambang batas penggunaan sumber daya akan dikenakan penalti tambahan, sehingga menghasilkan skor yang lebih tinggi dan cenderung tidak diprioritaskan.

```
1 FUNCTION calculate_node_score(node_data) RETURNS Float:
2   SET cpu_usage = node_data["cpu_usage_percent"] OR 100
3   SET memory_usage = node_data["memory_usage_percent"] OR 100
4
5   SET score = (cpu_usage * CPU_WEIGHT) + (memory_usage * MEMORY_WEIGHT)
6
7   IF cpu_usage > 90 OR memory_usage > 90 THEN
8     score = score + HEAVY_PENALTY
9   ELSE IF cpu_usage > 80 OR memory_usage > 80 THEN
10    score = score + MEDIUM_PENALTY
11  END IF
12
13  RETURN Round(score, 2)
14 END FUNCTION
```

Kode Sumber 2.7: Pseudocode Fungsi Perhitungan Skor Node

Selain itu, fungsi `select_nodes_by_algorithm()` digunakan untuk memilih node terbaik sesuai algoritma yang ditentukan, sedangkan `distribute_load()` digunakan untuk mendistribusikan workload berdasarkan kapasitas maksimal per node.

E. Konfigurasi Environment

Discovery Service menggunakan pendekatan berbasis konfigurasi eksternal agar sistem dapat dengan mudah dijalankan di berbagai lingkungan seperti *development*, maupun *production*. Semua pengaturan disatukan dalam satu file `config.py` yang memanfaatkan *library python-dotenv* untuk membaca variabel dari file `.env`.

```
1 CLASS Config:
2   ATTRIBUTE SECRET_KEY = getenv("SECRET_KEY", "secret-service-1111")
3   ATTRIBUTE DEBUG = getenv("DEBUG", "True") == "true"
4
5   // Database Configuration
6   ATTRIBUTE POSTGRES_HOST = getenv("POSTGRES_HOST", "localhost")
7   ATTRIBUTE POSTGRES_PORT = getenv("POSTGRES_PORT", "5432")
8   ATTRIBUTE POSTGRES_DB = getenv("POSTGRES_DB", "discovery")
9   ATTRIBUTE POSTGRES_USER = getenv("POSTGRES_USER", "postgres")
10  ATTRIBUTE POSTGRES_PASSWORD = getenv("POSTGRES_PASSWORD", "postgres")
11
```

```

12  ATTRIBUTE SQLALCHEMY_DATABASE_URI =
13      "postgresql://" + POSTGRES_USER + ":" + POSTGRES_PASSWORD +
14      "@" + POSTGRES_HOST + ":" + POSTGRES_PORT + "/" + POSTGRES_DB
15
16  ATTRIBUTE SQLALCHEMY_TRACK_MODIFICATIONS = False
17  ATTRIBUTE SQLALCHEMY_ECHO = getenv("SQLALCHEMY_ECHO", "false") == "↵
    true"
18
19  // Redis Configuration
20  ATTRIBUTE REDIS_HOST = getenv("REDIS_HOST", "localhost")
21  ATTRIBUTE REDIS_PORT = getenv("REDIS_PORT", 6379) AS Integer
22  ATTRIBUTE REDIS_PASSWORD = getenv("REDIS_PASSWORD", "redis@pass")
23  ATTRIBUTE REDIS_EXPIRE_SECONDS = getenv("REDIS_EXPIRE_SECONDS", 45) ↵
    AS Integer
24
25  // Load Balancer Thresholds
26  ATTRIBUTE DEFAULT_MAX_CPU_USAGE = 80.0
27  ATTRIBUTE DEFAULT_MAX_MEMORY_USAGE = 85.0
28  ATTRIBUTE STRICT_MAX_CPU_USAGE = 60.0
29  ATTRIBUTE STRICT_MAX_MEMORY_USAGE = 60.0
30  ATTRIBUTE STRICT_MAX_CONTAINERS = 5
31
32  // Scoring Weights & Penalties
33  ATTRIBUTE CPU_WEIGHT = 0.8
34  ATTRIBUTE MEMORY_WEIGHT = 0.8
35  ATTRIBUTE HEAVY_PENALTY = 80
36  ATTRIBUTE MEDIUM_PENALTY = 20
37  END CLASS

```

Kode Sumber 2.8: Pseudocode untuk config.py

Seluruh konfigurasi di atas bersifat dinamis dan dapat disesuaikan melalui file `.env` tanpa perlu mengubah kode Python. Contoh isi file konfigurasi lingkungan dapat dilihat pada Tabel 2.5 berikut:

Tabel 2.5: Contoh Isi File `.env` dari *Discovery Service*

Variabel	Deskripsi
FLASK_DEBUG=True	Mengaktifkan mode debug pada aplikasi Flask.
SECRET_KEY=secret-service111111	Kunci rahasia untuk keperluan autentikasi Flask.
POSTGRES_HOST=127.0.0.1	Alamat host untuk koneksi ke database PostgreSQL.
POSTGRES_PORT=5432	Port yang digunakan PostgreSQL.
POSTGRES_DB=voyager	Nama database utama yang digunakan.
POSTGRES_USER=postgres	Nama pengguna untuk mengakses PostgreSQL.
POSTGRES_PASSWORD=postgres	Password pengguna PostgreSQL.
REDIS_HOST=127.0.0.1	Alamat host untuk server Redis.
REDIS_PORT=6379	Port Redis yang digunakan.
REDIS_PASSWORD=redis@pass	Password autentikasi ke Redis.
REDIS_EXPIRE_SECONDS=45	Waktu kedaluwarsa (dalam detik) untuk data Redis.

Dengan struktur seperti ini, sistem dapat dengan mudah dipindahkan antar server atau dijalankan dalam konteks kontainer Docker tanpa harus mengubah kode utama aplikasi.

F. Deployment Service Discovery dengan Docker

Untuk memudahkan proses deployment dan reproduksibilitas lingkungan, Discovery Service dikemas dalam sebuah image menggunakan Docker. Layanan ini selanjutnya diatur dengan Docker Compose untuk menjalankan seluruh komponen (Flask API, Redis, PostgreSQL) secara terorkestrasi.

Dockerfile. Berkas Dockerfile berikut akan membangun image Python 3.12, menginstal dependensi dari `requirements.txt`, dan menjalankan `app.py` sebagai aplikasi utama.

```
1 PROCEDURE BuildDockerImage
2
3     // Inisialisasi Konfigurasi Dasar
4     SET BaseImage TO "python:3.12-slim"
5     SET WorkingDirectory TO "/app"
6
7     // Konfigurasi Environment Variables
8     SET EnvVariable "PYTHONDONTWRITEBYTECODE" TO "1"
9     SET EnvVariable "PYTHONUNBUFFERED" TO "1"
10    SET EnvVariable "TZ" TO "Asia/Jakarta"
11
12    // Proses Instalasi Dependensi
13    COPY "requirements.txt" FROM source TO "/app/"
14    EXECUTE "pip install --no-cache-dir -r requirements.txt"
15
16    // Salin Kode Aplikasi
17    COPY all_files FROM source TO "/app/"
18
19    // Konfigurasi Jaringan dan Eksekusi
20    EXPOSE Port 15002
21    SET DefaultCommand TO ["python", "app.py"]
22
23 END PROCEDURE
```

Kode Sumber 2.9: Pseudocode Proses Pembangunan Image Docker

Docker Compose. Untuk menjalankan layanan ini secara bersamaan dengan Redis dan PostgreSQL, digunakan `docker-compose.yml` berikut:

```
1 COMPOSE VERSION: 3.8
2
3 DEFINE SERVICES:
4
5     SERVICE discovery:
6         BUILD from Dockerfile in current context
7         CONTAINER NAME = discovery-api
8         RESTART POLICY = unless-stopped
9         EXPOSE PORTS: 15002:15002
10
11     SET ENVIRONMENT VARIABLES:
12         POSTGRES_HOST = postgres
13         POSTGRES_PORT = 5432
14         POSTGRES_DB = voyager
15         POSTGRES_USER = postgres
16         POSTGRES_PASSWORD = postgres
17
18         REDIS_HOST = redis
```

```

19     REDIS_PORT = 16379
20     REDIS_PASSWORD = redis@pass
21     REDIS_EXPIRE_SECONDS = 45
22
23     API_HOST = 0.0.0.0
24     API_PORT = 15002
25     DEBUG = True
26     SECRET_KEY = secret-service111111
27
28     DEPENDS ON: postgres, redis
29     NETWORKS: discovery-network
30
31     SERVICE postgres:
32         IMAGE = postgres:14-alpine
33         CONTAINER NAME = postgres
34         RESTART POLICY = unless-stopped
35         EXPOSE PORTS: 5432:5432
36
37     SET ENVIRONMENT VARIABLES:
38         POSTGRES_DB = ds
39         POSTGRES_USER = postgres
40         POSTGRES_PASSWORD = postgres
41         POSTGRES_INITDB_ARGS = --encoding=UTF-8
42         TZ = Asia/Jakarta
43
44     VOLUMES:
45         postgres_data -> /var/lib/postgresql/data
46         ./init.sql -> /docker-entrypoint-initdb.d/init.sql
47
48     HEALTHCHECK:
49         COMMAND = pg_isready -U postgres -d discovery_db
50         INTERVAL = 10s
51         TIMEOUT = 5s
52         RETRIES = 5
53         START_PERIOD = 30s
54
55     NETWORKS: discovery-network
56
57     SERVICE redis:
58         IMAGE = redis:7-alpine
59         CONTAINER NAME = redis
60         RESTART POLICY = unless-stopped
61         EXPOSE PORTS: 16379:16379
62
63     VOLUMES:
64         redis_data -> /data
65         ./redis.conf -> /usr/local/etc/redis/redis.conf
66
67     COMMAND: redis-server /usr/local/etc/redis/redis.conf
68
69     SET ENVIRONMENT VARIABLES:
70         TZ = Asia/Jakarta
71
72     HEALTHCHECK:
73         COMMAND = redis-cli -p 16379 ping
74         INTERVAL = 10s
75         TIMEOUT = 3s

```



```

76         RETRIES = 3
77
78     NETWORKS: discovery-network
79
80 DEFINE VOLUMES:
81     postgres_data
82     redis_data
83
84 DEFINE NETWORKS:
85     discovery-network (driver: bridge)

```

Kode Sumber 2.10: Pseudocode untuk docker-compose.yml

docker-compose.yml mendefinisikan tiga layanan utama: discovery, postgres, dan redis, yang saling terhubung melalui jaringan internal discovery-network. Untuk menjalankan seluruh services, gunakan perintah:

```
1 docker-compose up -d --build
```

Kode Sumber 2.11: Menjalankan Discovery Service via Docker Compose

G. List API Endpoint

Tabel 2.6: Daftar Endpoint REST API pada Discovery Service

Metode	Endpoint	Fungsi
GET	/health-check	Mengecek status koneksi layanan, termasuk status Redis dan PostgreSQL.
POST	/register-node	Menerima informasi node dari Agent dan menyimpan status terbaru ke Redis serta basis data.
GET	/available-nodes	Mengambil daftar node aktif beserta skor beban terkini.
POST	/select-nodes	Memilih sejumlah node berdasarkan algoritma load balancing tertentu.
GET	/all-nodes	Menampilkan semua node yang pernah terdaftar, termasuk node yang tidak aktif.
GET	/profiles	Menampilkan daftar seluruh profil user yang tersedia.
POST	/profiles	Menambahkan profil baru ke sistem.
PUT	/profiles/<id>	Memperbarui konfigurasi profil berdasarkan ID.
DELETE	/profiles/<id>	Menghapus profil dari sistem berdasarkan ID.

2.2.2 Service Agent

Setelah layanan Discovery Service diimplementasikan, sistem memerlukan komponen tambahan yang berjalan secara periodik di setiap node. Komponen ini disebut sebagai *Agent Service*. Agent bertanggung jawab untuk mengumpulkan informasi sistem dan mengirimkannya secara berkala ke endpoint **/register-node** pada Discovery API. Informasi tersebut mencakup pemanfaatan CPU, memori, disk, deteksi GPU, serta jumlah container yang sedang aktif. Bagian ini akan menjelaskan konfigurasi dari Agent Service dan deployment-menggunakan Docker secara lebih mendalam.

A. Arsitektur dan Fungsi Agent

Agent dikembangkan sebagai skrip Python mandiri yang berjalan sebagai *container* pada setiap node. Agent ini dirancang agar:

- Mengirimkan data sistem setiap 15 detik.
- Menangkap informasi hardware dan aktivitas container.
- Tetap ringan dan tidak membebani node secara signifikan.

B. Implementasi dan Pengumpulan Data

Agent diimplementasikan dalam bahasa Python dan berjalan sebagai *container* terpisah di setiap node. Agent secara berkala mengumpulkan informasi sistem dan mengirimkannya ke Discovery API melalui endpoint **/register-node**. Seluruh proses berlangsung setiap 15 detik, memastikan bahwa data yang dikirim tetap *up-to-date*.

Fungsi utama agent dimulai dari `register()`, seperti ditunjukkan pada Listing 2.12. Fungsi ini bertugas mengumpulkan data menggunakan `collect_node_info()` dan mengirimkannya ke API.

```
1 FUNCTION register():
2     SET payload = collect_node_info()
3
4     IF payload IS NOT EMPTY THEN
5         CALL HTTP POST to DISCOVERY_URL with JSON payload
6     END IF
7 END FUNCTION
```

Kode Sumber 2.12: Pseudocode Fungsi Register Agent

Fungsi `collect_node_info()` bertanggung jawab untuk membaca informasi hardware dan beban kerja node. Data yang dikumpulkan meliputi:

- Penggunaan CPU, memori, dan disk saat ini.
- Informasi jumlah container (JupyterLab dan Ray).
- Deteksi GPU NVIDIA.

```
1 FUNCTION collect_node_info():
2     SET hostname = GET system hostname
3     SET ip_address = GET primary IP address of system
4
5     SET ram_gb = total RAM in GB (rounded to 2 decimals)
6     SET cpu_usage = current CPU usage percent (measured over 1 second)
7     SET memory = current memory usage summary
8     SET disk = disk usage summary for root directory "/"
9
10    // Return or assemble data dictionary (not shown)
11 END FUNCTION
```

Kode Sumber 2.13: Pseudocode Kumpulan Informasi Sistem oleh Agent

Agent juga menghitung jumlah container yang berjalan dengan membaca nama dan image-nya. Hal ini dilakukan oleh fungsi `get_container_info()`, yang akan mengenali apakah container tersebut merupakan JupyterLab atau Ray Worker.

```
1 def get_container_info():
2     containers = docker_client.containers.list()
3     for container in containers:
4         if "jupyter" in container.name or "jupyter" in container.image.tags:
5             ...
```

Kode Sumber 2.14: Deteksi Container JupyterLab dan Ray

Untuk mendeteksi keberadaan GPU, agent menggunakan pustaka `gpustat`. Jika GPU NVIDIA tersedia, maka informasi seperti penggunaan memori, suhu, dan load GPU akan dikirimkan. Jika tidak tersedia, akan dilakukan fallback untuk deteksi AMD GPU.

```
1 def get_gpu_stats():
2     stats = gpustat.GPUStatCollection.new_query()
3     for gpu in stats.gpus:
4         gpu_info.append({
5             "name": gpu.name,
6             "memory_used_mb": gpu.memory_used,
7             "utilization_gpu_percent": gpu.utilization
8         })
```

Kode Sumber 2.15: Deteksi GPU Menggunakan gpustat

Akhirnya, agent akan menjalankan proses ini dalam loop tak hingga, mengirimkan data ke API setiap 15 detik. Hal ini memungkinkan Discovery Service selalu memiliki data terbaru untuk pengambilan keputusan.

```
1 IF this script is executed as main program THEN
2     WHILE True DO
3         CALL register()
4         WAIT 15 seconds
5     END WHILE
6 END IF
```

Kode Sumber 2.16: Pseudocode Loop Registrasi Agent Tiap 15 Detik

C. Deployment Agent

Agar dapat berjalan secara independen di setiap node, Agent dibungkus ke dalam sebuah *container* menggunakan Docker. Hal ini memungkinkan deployment yang konsisten di seluruh lingkungan tanpa bergantung pada konfigurasi sistem host. Kode sumber 2.17 menunjukkan isi file Dockerfile yang digunakan untuk membangun image Agent.

```
1 BUILD IMAGE:
2   BASE IMAGE = python:3.12-slim
3
4   SET ENVIRONMENT VARIABLES:
5       PYTHONDONTWRITEBYTECODE = 1
6       PYTHONUNBUFFERED = 1
7       TZ = Asia/Jakarta
8
9   SET WORKING DIRECTORY = /app
10
11  COPY file requirements.txt to /app
12
13  RUN pip install requirements from requirements.txt (no cache)
14
15  COPY all files from build context to /app
16
17  EXPOSE PORT 15002
18
19  SET DEFAULT COMMAND to run:
20      "python agent.py"
21 END BUILD
```

Kode Sumber 2.17: Pseudocode untuk Dockerfile Agent Service

Struktur file sangat sederhana. Base image yang digunakan adalah `python:3.12-slim` untuk memastikan image tetap ringan. *Working Directory* di-set ke `/app`, dan seluruh kode serta dependensi diinstal melalui `requirements.txt`. Command akhir akan menjalankan file `agent.py`.

Selanjutnya agent akan di-build menjadi Docker image dan akan di-push ke Docker registry:

```
1 # Build image agent
2 docker build -t danielcristh0/agent:1.1 .
```

Kode Sumber 2.18: Perintah untuk Build dan Menjalankan Agent



Repository	Tag	Image ID	Time since build	Size
danielcristh0/agent	1.1	0c646f042897	25 hours ago	134MB

Gambar 2.2: Hasil build agent menjadi image

```
1
2 # Push image agent
3 docker push danielcristh0/agent:1.1
```

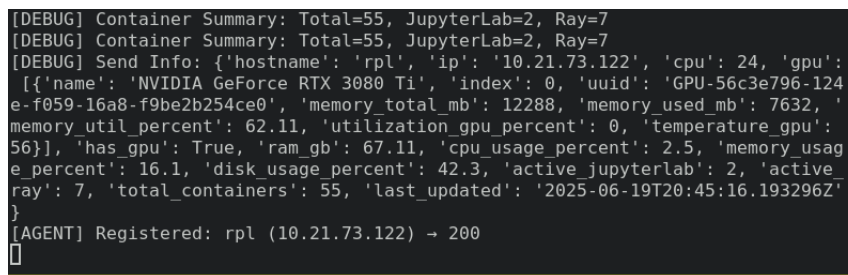
Kode Sumber 2.19: Perintah untuk Build dan Menjalankan Agent

```
1
2 # Menjalankan agent di setiap node
3 docker run --name agent -d \
4     --net=host \
5     -e DISCOVERY_URL=http://192.168.122.1:15002/register-node \
6     danielcristh0/agent:1.1
7
8 # Tambahkan "--gpus all" di node yang memiliki GPU
9 docker run --name agent -d \
10    --net=host \
11    -e DISCOVERY_URL=http://10.21.73.116:15002/register-node \
12    -v /var/run/docker.sock:/var/run/docker.sock \
13    --gpus all \
14    danielcristh0/agent:1.1
```

Kode Sumber 2.20: Perintah untuk Menjalankan Agent di setiap Node

Agent akan mengirimkan informasi ke endpoint **/register-node** dan menggunakan IP Address node manager. Penggunaan mode **-net=host** memungkinkan agent mengakses informasi IP node dengan benar serta membaca container aktif melalui Docker daemon lokal.

Docker logs dari container agent:



```
[DEBUG] Container Summary: Total=55, JupyterLab=2, Ray=7
[DEBUG] Container Summary: Total=55, JupyterLab=2, Ray=7
[DEBUG] Send Info: {'hostname': 'rpl', 'ip': '10.21.73.122', 'cpu': 24, 'gpu':
[{'name': 'NVIDIA GeForce RTX 3080 Ti', 'index': 0, 'uuid': 'GPU-56c3e796-124
e-f059-16a8-f9be2b254ce0', 'memory_total_mb': 12288, 'memory_used_mb': 7632, '
memory_util_percent': 62.11, 'utilization_gpu_percent': 0, 'temperature_gpu':
56}], 'has_gpu': True, 'ram_gb': 67.11, 'cpu_usage_percent': 2.5, 'memory_usag
e_percent': 16.1, 'disk_usage_percent': 42.3, 'active_jupyterlab': 2, 'active_
ray': 7, 'total_containers': 55, 'last_updated': '2025-06-19T20:45:16.193296Z'
}
[AGENT] Registered: rpl (10.21.73.122) → 200
```

Gambar 2.3: Log dari container agent ketika melakukan registrasi node

2.2.3 JupyterHub

JupyterHub bertindak sebagai sistem autentikasi dan pengelola sesi pengguna. Setiap pengguna dapat memulai server JupyterLab pribadi yang dijalankan sebagai container terisolasi. Dengan bantuan spawner khusus yang terintegrasi dengan Discovery Service, JupyterHub akan secara otomatis memilih node dengan resource teringan. Spawner ini juga bertugas melakukan konfigurasi container secara otomatis, termasuk setting IP, port, dan image sesuai kebutuhan pengguna.

Pada bagian ini akan dijelaskan struktur proyek, konfigurasi sistem, serta integrasi multi-node spawner dengan Service Discovery.

A. Arsitektur Aplikasi

Proyek JupyterHub ini dibangun dengan struktur modular yang memisahkan konfigurasi, spawner, dan form HTML dalam direktori berbeda. Hal ini memudahkan pemeliharaan dan pengembangan fitur baru.

Tabel 2.7: Struktur Direktori Proyek JupyterHub

Nama File/Folder	Deskripsi
jupyterhub_config.py	Titik masuk konfigurasi utama JupyterHub.
config/	Konfigurasi modular auth, spawner, hooks, dan proxy.
spawner/	Implementasi custom MultiNodeSpawner.
form/	Template HTML dan JS untuk interface pemilihan node.
singleuser/	Dockerfile dan skrip build image JupyterLab.
docker-compose.yml	Orkestrasi layanan JupyterHub dan reverse proxy.

B. Inisialisasi Konfigurasi dan Komponen

File `jupyterhub_config.py` berperan sebagai titik masuk konfigurasi yang memanggil fungsi-fungsi konfigurasi modular dari direktori `config/`. Ini termasuk konfigurasi environment, autentikasi, proxy, spawner, serta hook yang diperlukan saat spawn dan terminasi container.

```
1 from config.env import load_environment
2 from config.hub import configure_hub
3 from config.spawner import configure_spawner
4 from config.proxy import configure_proxy
5 from config.auth import configure_auth
6 from config.hooks import attach_hooks
7
8 load_environment(c)
9 configure_hub(c)
10 configure_spawner(c)
11 configure_proxy(c)
12 configure_auth(c)
13 attach_hooks()
```

Kode Sumber 2.21: `jupyterhub_config.py`

2.2.3.1 Integrasi Multi-Node dan Spawner Khusus

Untuk mendukung eksekusi JupyterLab pada beberapa node sekaligus, sistem ini mengimplementasikan spawner kustom berbasis `DockerSpawner`, yang disebut `MultiNodeSpawner`. Seluruh kode terkait ditempatkan dalam direktori **`spawner/`**.

A. Implementasi `MultiNodeSpawner`

`MultiNodeSpawner` diimplementasikan sebagai subclass dari `DockerSpawner` dan bertugas menjalankan container JupyterLab pada node yang dipilih berdasarkan informasi dari Discovery Service. Spawner ini mengakses endpoint `/select-nodes` dengan parameter berupa `profile_id` dan `user_id`, lalu menerima daftar node dengan skor beban terbaik.

Jika profil pengguna membutuhkan lebih dari satu node, maka Spawner akan menyimpan daftar node tersebut dalam atribut `selected_nodes`, dan menggunakan node pertama sebagai tempat menjalankan JupyterLab utama, sementara node lainnya digunakan untuk Ray Worker.

```
1 payload = {
2     "profile_id": self.profile_id,
3     "user_id": self.user.name,
4     "num_nodes": self.num_nodes
5 }
6 response = requests.post(f"{self.discovery_api_url}/select-nodes", json=payload)
7 self.selected_nodes = response.json().get("selected_nodes", [])
```

Kode Sumber 2.22: Pemanggilan API Seleksi Node

Setiap node yang dipilih akan dihubungi melalui koneksi Docker remote. Untuk itu, spawner membuat klien Docker dinamis berdasarkan ip node dengan format `tcp://<ip>:2375`.

Fungsi ini di-override melalui metode `_docker()`.

```
1 def _docker(self, node_ip=None):
2     if not node_ip:
3         return super()._docker()
4     if node_ip not in self._docker_clients:
5         self._docker_clients[node_ip] = docker.DockerClient(base_url=f"↵
6         tcp://{node_ip}:2375")
7     return self._docker_clients[node_ip]
```

Kode Sumber 2.23: Inisialisasi Docker Client Berdasarkan IP Node

Setelah node terpilih, proses pembuatan container dibagi dua:

1. **Container JupyterLab primary:** dijalankan di node pertama, berisi environment inter-aktif untuk pengguna.
2. **Container JupyterLab worker(opsional):** dijalankan di node-node sisanya jika profil pengguna mendukung komputasi paralel.

Fungsi `create_worker_container()` akan mengatur nama, image, environment, dan volume binding untuk container Ray Worker. Informasi ID container disimpan di `worker_containers[user.name]` untuk keperluan manajemen.

```
1 worker_container = docker_client.containers.run(
2     image=image,
3     name=container_name,
4     command="ray start --address={ip}:{port}",
5     environment=worker_env,
6     detach=True
7 )
```

Kode Sumber 2.24: Pembuatan Container Ray Worker di Node Tambahan

Semua proses `create_user_container()` dan `create_worker_container()` dijalankan secara paralel dengan `asyncio.gather()` agar proses spawning lebih cepat dan efisien.

```
1 await asyncio.gather(
2     self.create_user_container(primary_node, image),
3     *(self.create_worker_container(n, image) for n in self.selected_nodes↵
4     [1:])
5 )
```

Kode Sumber 2.25: Eksekusi Paralel untuk Container Jupyter dan Worker

B.Implementasi PatchedMultiNodeSpawner

File `multinode.py` berisi kelas `PatchedMultiNodeSpawner` yang mewarisi `MultiNodeSpawner` dan menambahkan perbaikan berikut:

- Perbaikan properti `server_url` agar selalu valid
- Sinkronisasi nilai IP dan port container ke variabel internal
- Penyesuaian konfigurasi URL dan argumen server Jupyter

Listing 2.26 menunjukkan implementasi override URL pada PatchedMultiNodeSpawner:

```
1 @property
2 def url(self):
3     base_url = self.server_url
4     if hasattr(self, 'default_url') and self.default_url:
5         base_url += self.default_url.rstrip("/")
6     return base_url
7
8 @property
9 def server_url(self):
10    if self.ip and self.port:
11        return f"http://{self.ip}:{self.port}"
12    return ""
```

Kode Sumber 2.26: Override URL pada PatchedMultiNodeSpawner

2.2.4 Ray Cluster

Ray digunakan untuk mengatur workload komputasi paralel. Dalam perancangannya, setiap container JupyterLab pengguna akan menjalankan Ray Worker secara otomatis dan terhubung ke Ray Head Node. Dengan cara ini, pengguna dapat langsung menggunakan fitur komputasi terdistribusi seperti `ray.remote()` tanpa konfigurasi manual. Ray menjembatani antar-node agar task berat bisa dijalankan dengan efisien di GPU atau CPU sesuai kapasitas.

Bagian ini akan menjelaskan cara integrasi Ray ke dalam sistem, termasuk konfigurasi head dan worker node, serta bagaimana komputasi paralel dapat dijalankan langsung dari dalam JupyterLab.

2.3 Peralatan Pendukung

Perangkat yang digunakan untuk pengerjaan tugas akhir ini merupakan sebuah komputer dengan spesifikasi sebagai berikut.

Tabel 2.8: Spesifikasi Peralatan Pendukung

No.	Komponen	Spesifikasi
1	Laptop	
	<i>Brand</i>	Asus
	<i>Processor</i>	AMD Ryzen 3
	<i>Operating System</i>	Ubuntu 22.04 LTS
	<i>GPU</i>	AMD Radeon vega 3 graphics
	<i>Memory</i>	18 GB
	<i>Storage</i>	512 GB
2	Komputer	
	<i>Brand</i>	Asus
	<i>Processor</i>	12th Gen Intel i9-12900K
	<i>Operating System</i>	Ubuntu 24.04 LTS
	<i>GPU</i>	NVIDIA GeForce RTX 3080 Ti
	<i>Memory</i>	64 GB
	<i>Storage</i>	723 GB
3	Virtual Machine 1	
	<i>Brand</i>	Asus
	<i>Processor</i>	12th Gen Intel i9-12900K
	<i>Operating System</i>	Ubuntu 24.04 LTS
	<i>GPU</i>	NVIDIA GeForce RTX 3080 Ti
	<i>Memory</i>	64 GB
	<i>Storage</i>	723 GB
4	Virtual Machine 2	
	<i>Brand</i>	Asus
	<i>Processor</i>	12th Gen Intel i9-12900K
	<i>Operating System</i>	Ubuntu 24.04 LTS
	<i>GPU</i>	NVIDIA GeForce RTX 3080 Ti
	<i>Memory</i>	64 GB
	<i>Storage</i>	723 GB

Selain perangkat keras, terdapat juga perangkat lunak pendukung seperti berikut.

Tabel 2.9: Daftar Perangkat Lunak Pendukung

Nama Perangkat Lunak	Versi	Keterangan
Docker Engine	28.2.2	Digunakan untuk menjalankan container JupyterLab secara terisolasi. Memungkinkan lingkungan komputasi tiap pengguna berjalan secara independen dan mudah didistribusikan ke berbagai node.
Docker Compose	2.36.2	Membantu mendefinisikan dan mengatur layanan multi-container JupyterHub dan Service Discovery dalam satu berkas konfigurasi. Memudahkan manajemen dan replikasi layanan.
JupyterHub	5.3.0	Menangani autentikasi pengguna serta spawn container JupyterLab ke node terpilih berdasarkan data dari service discovery.
Ray	2.46	Framework komputasi paralel dan terdistribusi. Setiap pengguna dapat langsung menjalankan task terdistribusi secara otomatis dari dalam JupyterLab.
Redis	7.0	Database key-value in-memory yang digunakan untuk menyimpan status sistem (CPU, RAM, GPU) dan log aktivitas pengguna secara real-time.
Flask	3.1.0	Framework web Python yang digunakan untuk membangun service discovery berupa REST API yang menerima dan menyediakan data status node.
Python	3.11	Bahasa pemrograman utama yang digunakan untuk seluruh komponen sistem, seperti konfigurasi JupyterHub, pengembangan REST API, Ray, serta skrip monitoring.
PostgreSQL	14	Basis data relasional yang digunakan untuk menyimpan data historis seperti riwayat pemilihan node, profil pengguna, dan metrik performa dari setiap node.

BAB III

PENGUJIAN DAN ANALISIS

Bab ini membahas proses pengujian dan hasil analisis terhadap sistem yang telah dibangun. Tujuan utama dari pengujian ini adalah untuk mengevaluasi kinerja Service Discovery dalam memilih node yang optimal untuk menjalankan container JupyterLab, serta memastikan bahwa integrasi antar komponen (JupyterHub, Discovery API, Agent, dan Docker) berjalan sesuai ekspektasi.

3.1 Hasil dan Pengujian

3.1.1 Uji Akses dan Proses Spawn Server

Langkah pertama yang diuji adalah memastikan sistem dapat diakses melalui antarmuka JupyterHub oleh pengguna biasa. Setelah pengguna berhasil login, sistem akan menampilkan form pemilihan profil dan node. Form ini mengirimkan data ke spawner untuk memulai proses alokasi sumber daya dan peluncuran container.

A. Langkah Pengujian

1. Akses halaman JupyterHub melalui `http://<ip_jupyterhub>:18000`.
2. Registrasi dan login menggunakan akun pengguna (misal: `demo`).
3. Pilih profil komputasi dan jumlah node pada form yang tersedia, lalu klik tombol `Launch Server`.

Warning: JupyterHub seems to be served over an unsecured HTTP connection. We strongly recommend enabling HTTPS for JupyterHub.

Username:
demo

Password:
demo@123

Confirm password:
demo@123

Create User

[Login](#) with an existing user.

(a) Melakukan Registrasi User

Warning: JupyterHub seems to be served over an unsecured HTTP connection. We strongly recommend enabling HTTPS for JupyterHub.

Username:
demo

Password:
demo@123

Sign In

[Sign up](#) to create a new user.

(b) Login dengan User yang telah diregistrasi

Gambar 3.1: Proses Registrasi dan Login pada JupyterHub

Server Options

Discovery Service Connected

Select Profile

Single Cpu
Single node with CPU only
2 CPU cores 2 GB RAM 1 node

Single Gpu
Single node with GPU acceleration
2 CPU cores 2 GB RAM 1 node GPU enabled

Multi Gpu
Multiple nodes with GPU acceleration
2 CPU cores 2 GB RAM 2-4 nodes GPU enabled

Multi Cpu
Multiple nodes with CPU only
2 CPU cores 2 GB RAM 2-4 nodes

(a) Memilih profil

Environment Configuration

Docker Image
CPU Environment (danielcrish0/jupyterlab.cpu)

Choose CPU for general computing or GPU for machine learning tasks

Node Configuration
Single Node Multi Node

3 Nodes - Medium cluster

Additional nodes will be used for distributed computing

(b) Memilih environment

Gambar 3.2: User memilih profil dan environment sesuai kebutuhan pada halaman konfigurasi JupyterHub

menunjukkan tampilan halaman awal untuk registrasi dan login, sementara Gambar ?? menunjukkan tampilan JupyterLab yang berhasil dijalankan.

3.1.2 Skenario 2: Multi-User Concurrent

BAB IV

PENUTUP

4.1 Kesimpulan

Berdasarkan hasil pengujian yang Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. sebagai berikut:

1. Mekanisme penjadwalan sumber daya dinamis berhasil diterapkan melalui arsitektur Service Discovery dan Service Agent. Service Agent yang berjalan di setiap node secara periodik mengumpulkan dan melaporkan metrik vital—seperti utilisasi CPU, RAM, GPU, dan jumlah kontainer aktif—ke Service Discovery. Data ini memungkinkan sistem untuk menghitung skor beban dan memilih node dengan beban paling optimal untuk setiap permintaan pengguna baru.
2. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna.

4.2 Saran

Untuk pengembangan lebih lanjut pada Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. antara lain:

1. Memperbaiki Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus.
2. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa.
3. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna.

[Halaman ini sengaja dikosongkan]

DAFTAR PUSTAKA

[Halaman ini sengaja dikosongkan]

BIOGRAFI PENULIS



Gloriyano Cristho Daniel Pepuho, lahir di Nabire pada 19 Agustus 2002. Penulis menempuh pendidikan formal di SD YPK Sion Nabire, SMP YPPK St. Antonius Nabire, dan SMKN 1 Sentani. Pada tahun 2020, penulis diterima sebagai mahasiswa di Departemen Teknik Informatika, FTEIC-ITS.

Selama menempuh studi, penulis aktif dalam berbagai kegiatan akademik dan pengembangan proyek. Penulis menjadi Administrator Laboratorium Networking Technology and Intelligent Cybersecurity (NETICS) dari tahun 2022 hingga 2024, di mana penulis turut membantu kegiatan pembelajaran sebagai asisten dosen pada mata kuliah Sistem Operasi dan Jaringan Komputer. Selain itu, penulis juga berpartisipasi sebagai staf Divisi IT dalam kegiatan Schematics serta terlibat dalam proyek pengembangan sistem Penerimaan Peserta Didik Baru (PPDB) Online untuk SMA/SMK se-Jawa Timur selama periode 2022–2025.

[Halaman ini sengaja dikosongkan]