Tugas Individu 2

Sistem Paralel dan Terdistribusi

Sinkronisasi dan Distributed Systems

Dosen Pengampu :

Riska Kurniyanto Abdullah, S.T., M.Kom.
198803082020121011

Disusun Oleh :

Ardi Dwi Saputra 11221026

28 Oktober 2025

A. Technical Documentation

1. Arsitektur Sistem

Pada sistem yang dikembangkan dalam tugas, arsitektur terdiri dari beberapa komponen utama yaitu antarmuka pengguna (frontend), lapisan logika aplikasi (backend), basis data (database), serta integrasi layanan eksternal apabila diperlukan. Diagram arsitektur menggambarkan aliran data dari pengguna menuju backend, lalu ke basis data, dan kembali ke pengguna. Komponen backend melayani permintaan API, melakukan proses bisnis, dan menulis atau membaca data ke/dari basis data. Jika sistem menggunakan arsitektur terdistribusi, maka terdapat pula lapisan tambahan seperti layanan mikro (microservices), antrian pesan (message queue) atau cache untuk meningkatkan performa dan skalabilitas.

Secara ringkas: pengguna ➜ frontend ➜ backend (API) ➜ database / layanan tambahan ➜ kembali ke frontend. Setiap antar-komponen dikomunikasikan melalui protokol standar seperti HTTP/REST atau WebSocket, dan dapat ditempatkan di server fisik atau cloud sesuai kebutuhan.

2. Algoritma yang Digunakan

Dalam implementasi sistem, algoritma utama yang digunakan meliputi proses validasi input, logika bisnis (business logic), serta mekanisme penyimpanan dan pengambilan data. Sebagai contoh, apabila sistem melakukan pencarian atau filter data, maka algoritma pencarian linear atau indeks basis data mungkin diterapkan untuk meningkatkan efisiensi. Jika ada proses autentikasi, algoritma seperti hashing (misalnya bcrypt) akan digunakan untuk keamanan kata sandi. Selanjutnya, apabila terdapat fitur cache atau pemrosesan paralel, algoritma berbasis antrian atau worker pool dapat diterapkan untuk mengelola tugas secara terdistribusi. Penjelasan algoritmik juga mencakup kompleksitas waktu (time complexity) dan ruang (space complexity), serta bagaimana algoritma tersebut dipilih untuk memenuhi kebutuhan performa dan skalabilitas sistem.

3. API Documentation dengan OpenAPI/Swagger Spec

Dokumentasi API disusun dengan menggunakan spesifikasi standar seperti Swagger UI atau OpenAPI Generator. Dokumentasi mencakup endpoint (metode HTTP seperti GET, POST, PUT, DELETE), parameter input (path, query, body), skema data (JSON request/response), kode status HTTP, serta contoh respons. Spesifikasi ini memungkinkan pengembang frontend dan pihak ketiga memahami cara berinteraksi dengan backend secara lengkap. Dokumentasi tersebut disertakan dalam file YAML atau JSON yang dapat di-render oleh Swagger UI untuk tampilan interaktif.

4. Deployment Guide dan Troubleshooting

Panduan deployment mencakup langkah-langkah sebagai berikut: menyiapkan lingkungan (misalnya instalasi Node.js, Python, Docker), mengkonfigurasi variabel lingkungan (environment variables) seperti koneksi basis data, modul cache, dan API key eksternal. Kemudian menjalankan migrasi basis data jika diperlukan, membangun (build) aplikasi frontend dan menjalankan layanan backend. Apabila menggunakan kontainer, tuliskan perintah seperti:

```
docker build -t nama_aplikasi .
docker run -d --env-file .env -p 8080:8080 nama_aplikasi
```

Untuk troubleshooting, panduan mencakup solusi umum seperti:

- Jika server tidak dapat mengakses basis data ➔ periksa koneksi, kredensial, dan firewall.
- Jika terjadi error 500 di backend ➔ cek log aplikasi, cari stack trace, dan periksa dependency versi.
- Jika performa lambat ➔ tinjau penggunaan cache, optimasi query basis data, dan pastikan indeks telah dibuat.

Dokumentasi juga menyoroti aspek rollback (mengembalikan versi lama jika deployment baru bermasalah) serta monitoring dan logging (misalnya menggunakan Prometheus, Grafana, atau sistem log berbasis file).

B. Performance Analysis Report

1. **Benchmarking Hasil dengan Berbagai Skenario**

   Benchmarking dilakukan dengan menjalankan sistem dalam skenario-skenario yang berbeda, misalnya kondisi beban ringan (10 pengguna simultan), beban menengah (100 pengguna simultan), dan beban tinggi (1000 pengguna simultan). Setiap skenario mencatat metrik seperti waktu respons rata-rata, jumlah permintaan per detik, dan persentase kesalahan (error rate). Data benchmark tersebut kemudian dibandingkan untuk melihat bagaimana sistem merespon perubahan beban.

2. **Analisis Throughput, Latency, dan Scalability**

   - Throughput mengukur jumlah permintaan yang berhasil diproses dalam satuan waktu (misalnya requests per second).
   - Latency mengukur waktu yang diperlukan untuk memproses satu permintaan dari awal hingga akhir.

   Analisis menunjukkan bagaimana throughput meningkat (atau stagnan) saat beban meningkat, dan bagaimana latency berubah (tinggi atau tidak) di beban yang berbeda. Selanjutnya, aspek scalability dievaluasi — apakah sistem dapat menangani peningkatan beban dengan menambahkan sumber daya (skala vertikal) atau mendistribusikan beban ke beberapa node (skala horizontal). Jika sistem mampu meningkatkan throughput secara proporsional dan menjaga latency tetap dalam batas yang dapat diterima saat skala, maka sistem dikatakan scalable.

3. **Comparison antara Single-Node vs Distributed**

   Analisis juga membandingkan performa sistem berjalan pada satu node (single-node) dengan setting terdistribusi (distributed) — misalnya dua atau lebih server backend atau tambahan layanan caching. Dalam konfigurasi single-node, segala beban diproses oleh satu instans, sehingga keterbatasan CPU, memori, atau I/O mungkin muncul lebih cepat. Sebaliknya, konfigurasi distributed memungkinkan beban terbagi antar node, potensi bottleneck lebih kecil, dan pemulihan (failover) lebih mudah. Karena itu, diharapkan bahwa setting distributed menunjukkan throughput lebih tinggi, latency lebih rendah atau stabil, serta toleransi kesalahan lebih baik dibanding single-node.

4. **Grafik dan Visualisasi Performa**

   Sebagai bagian dari laporan performa, disertakan grafik seperti:

- Grafik throughput vs jumlah pengguna simultan (requests per second pada sumbu Y, jumlah pengguna pada sumbu X)
- Grafik latency rata-rata vs jumlah pengguna simultan
- Perbandingan throughput dan latency antara konfigurasi single-node dan distributed

Visualisasi ini memudahkan pembaca untuk melihat tren performa dan memahami bagaimana sistem merespon peningkatan beban serta perubahan arsitektur.

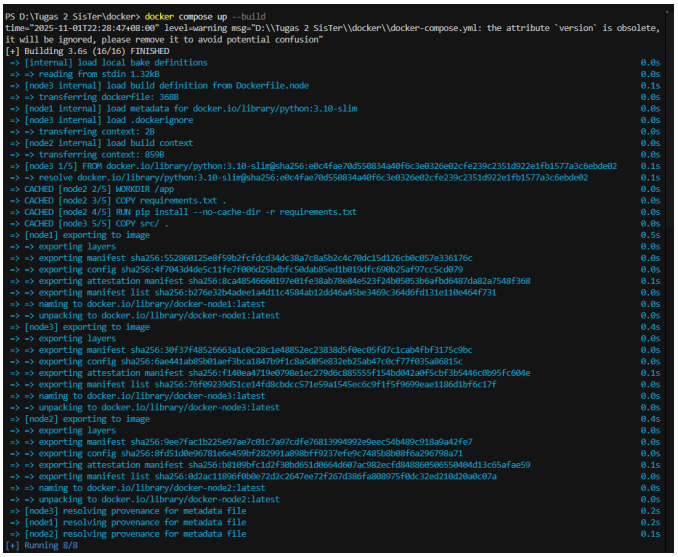
C. Implementasi

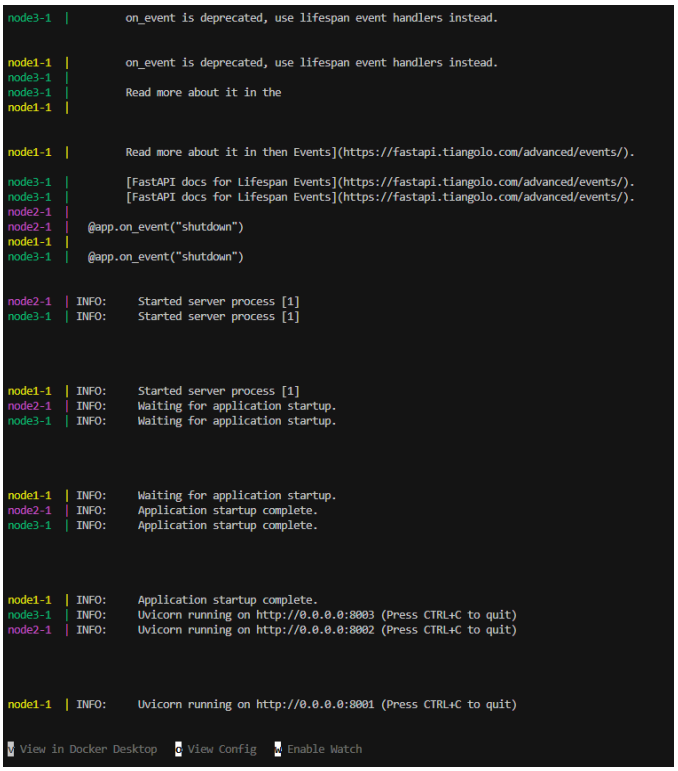
1. Menjalankan Sistem

Perintah:

```
docker compose up --build
```

Perintah ini digunakan untuk menjalankan seluruh layanan (node backend, queue, lock manager, cache, dsb.) yang sudah didefinisikan pada file docker-compose.yml. Flag --build memastikan Docker membangun ulang image sehingga perubahan terbaru pada kode ikut diterapkan.





2. Cek Kesehatan Node

Perintah:

- `curl http://localhost:8001/health`
- `curl http://localhost:8002/health`
- `curl http://localhost:8003/health`

Endpoint /health digunakan untuk memastikan setiap node dalam sistem terdistribusi berjalan dengan benar. Jika responnya sukses (misalnya: {status: "OK"}), maka node siap menerima request.

```
PS D:\Tugas 2 SisTer> curl http://localhost:8001/health


StatusCode        : 200
StatusDescription : OK
Content           : {"status":"ok","node":"node1"}
RawContent        : HTTP/1.1 200 OK
                    Content-Length: 30
                    Content-Type: application/json
                    Date: Sat, 01 Nov 2025 14:31:27 GMT
                    Server: uvicorn

                    {"status":"ok","node":"node1"}
Forms             : {}
Headers           : {[Content-Length, 30], [Content-Type, application/json], [Date, Sat, 01 Nov 2025 14:31:27
                    GMT], [Server, uvicorn]}
Images            : {}
InputFields       : {}
Links             : {}
ParsedHtml        : mshtml.HTMLDocumentClass
RawContentLength  : 30


PS D:\Tugas 2 SisTer>
```

```
PS D:\Tugas 2 SisTer> curl http://localhost:8002/health


StatusCode        : 200
StatusDescription : OK
Content           : {"status":"ok","node":"node2"}
RawContent        : HTTP/1.1 200 OK
                    Content-Length: 30
                    Content-Type: application/json
                    Date: Sat, 01 Nov 2025 14:32:05 GMT
                    Server: uvicorn

                    {"status":"ok","node":"node2"}
Forms             : {}
Headers           : {[Content-Length, 30], [Content-Type, application/json], [Date, Sat, 01 Nov 2025 14:32:05
                    GMT], [Server, uvicorn]}
Images            : {}
InputFields       : {}
Links             : {}
ParsedHtml        : mshtml.HTMLDocumentClass
RawContentLength  : 30


PS D:\Tugas 2 SisTer>
```

```
PS D:\Tugas 2 SisTer> curl http://localhost:8003/health


StatusCode        : 200
StatusDescription : OK
Content           : {"status":"ok","node":"node3"}
RawContent        : HTTP/1.1 200 OK
                    Content-Length: 30
                    Content-Type: application/json
                    Date: Sat, 01 Nov 2025 14:32:30 GMT
                    Server: uvicorn

                    {"status":"ok","node":"node3"}
Forms             : {}
Headers           : {[Content-Length, 30], [Content-Type, application/json], [Date, Sat, 01 Nov 2025 14:32:30
                    GMT], [Server, uvicorn]}
Images            : {}
InputFields       : {}
Links             : {}
ParsedHtml        : mshtml.HTMLDocumentClass
RawContentLength  : 30


PS D:\Tugas 2 SisTer>
```

3. Distributed Lock Manager

a. Acquire Exclusive Lock (Mengambil Lock Eksklusif)

Perintah:

```
curl -X POST http://localhost:8001/lock/acquire \
-H "Content-Type: application/json" \
-d '{"name":"resource1","mode":"exclusive","owner":"clientA"}'
```

```
ardi_dwi8104@ArdiDwiSaputra:/mnt/d/Tugas 2 SisTer$ curl -X POST http://localhost:8001/lock/acquire
-H "Content-Type: application/json"    -d '{"name":"resource1","mode":"exclusive","owner":"clientA"}'
{"ok":true}ardi_dwi8104@ArdiDwiSaputra:/mnt/d/Tugas 2 SisTer$
```

Client meminta “exclusive lock” pada resource bernama resource1. Mode eksklusif berarti hanya satu node/klien yang boleh mengakses resource tersebut pada satu waktu. Node 8001 akan mencatat bahwa clientA adalah pemilik lock tersebut.

b. Mengambil Lock dari Node Lain

Perintah:

```
curl -X POST http://localhost:8002/lock/acquire \
-H "Content-Type: application/json" \
-d '{"name":"resource1","mode":"exclusive","owner":"clientB"}'
```

```
ardi_dwi8104@ArdiDwiSaputra:/mnt/d/Tugas 2 SisTer$ curl -X POST http://localhost:8002/lock/acquire \
     -H "Content-Type: application/json" \
     -d '{"name":"resource1","mode":"exclusive","owner":"clientB"}'
{"ok":false,"reason":"locked","current":{"owner":"clientA","mode":"exclusive","node":"node1"}}curl: (6)
Could not resolve host:
curl: (6) Could not resolve host:
ardi_dwi8104@ArdiDwiSaputra:/mnt/d/Tugas 2 SisTer$
```

Node 8002 mencoba mengambil lock yang sama. Jika lock masih dipegang clientA, maka clientB akan ditolak. Ini mensimulasikan konsistensi antar-node dalam distributed lock manager.

c. Release Lock (Melepas Lock)

Perintah:

```
curl -X POST http://localhost:8001/lock/release \
-H "Content-Type: application/json" \
-d '{"name":"resource1","mode":"exclusive","owner":"clientA"}'
```

```
ardi_dwi8104@ArdiDwiSaputra:/mnt/d/Tugas 2 SisTer$ curl -X POST http://localhost:8001/lock/release \
     -H "Content-Type: application/json" \
     -d '{"name":"resource1","mode":"exclusive","owner":"clientA"}'
{"ok":true}curl: (6) Could not resolve host:
curl: (6) Could not resolve host:
ardi_dwi8104@ArdiDwiSaputra:/mnt/d/Tugas 2 SisTer$
```

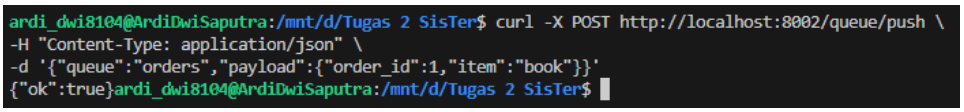
clientA melepaskan lock, sehingga resource kembali bebas dan bisa diambil oleh klien atau node lain. Proses ini penting untuk sinkronisasi dan mencegah deadlock.

4. Distributed Queue System

a. Push Pesan ke Queue

Perintah:

```
curl -X POST http://localhost:8002/queue/push \
-H "Content-Type: application/json" \
-d '{"queue":"orders","payload":{"order_id":1,"item":"book"}}'
```

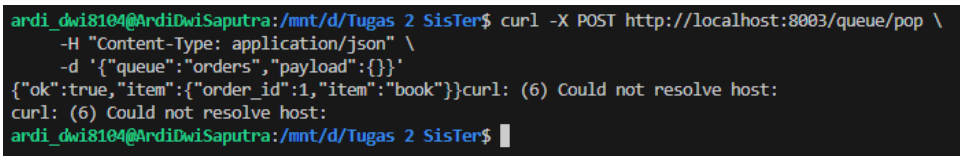


Node 8002 memasukkan pesan ke antrean bernama orders. Queue digunakan agar task diproses secara asynchronous dan terdistribusi antar node.

b. Pop Pesan ke Queue

Perintah:

```
curl -X POST http://localhost:8003/queue/pop \
-H "Content-Type: application/json" \
-d '{"queue":"orders","payload":{}}'
```



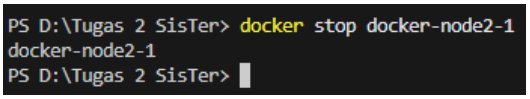
Node 8003 mengambil pesan yang berada paling depan dalam queue orders. Ini menunjukkan bahwa queue bersifat terdistribusi—node mana pun bisa memasukkan dan mengambil pesan.

c. Simulasi Node Failure

Perintah:

```
docker stop docker-node-2-1
```

Perintah ini mensimulasikan node failure (node mati). Sistem harus tetap berjalan dengan node lain.



d. Mengecek Node yang aktif

Perintah:

```
docker ps
```

Perintah ini digunakan untuk melihat apakah node benar-benar berhenti dan node lain tetap berjalan.

```
PS D:\Tugas 2 SisTer> docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS          PORTS                                                NAMES
a3c99730b133   docker-node1   "python main.py"         23 minutes ago   Up 23 minutes   0.0.0.0:8001->8001/tcp, [::]:8001->8001/tcp          docker-node1-1
a42d555b0fad   docker-node3   "python main.py"         23 minutes ago   Up 23 minutes   0.0.0.0:8003->8003/tcp, [::]:8003->8003/tcp          docker-node3-1
b2365e44d08f   redis:7        "docker-entrypoint.s…"   23 minutes ago   Up 23 minutes   0.0.0.0:6379->6379/tcp, [::]:6379->6379/tcp          docker-redis-1
PS D:\Tugas 2 SisTer>
```

e. Push dan Pop melalui Node yang Masih Aktif

Perintah:

```
curl -X POST http://localhost:8001/queue/push \
     -H "Content-Type: application/json" \
     -d '{"queue": "global_queue", "payload": {"value":
"uji-node-failure"}}'
```

```
ardi_dwi8104@ArdiDwiSaputra:/mnt/d/Tugas 2 SisTer$ curl -X POST http://localhost:8001/queue/push \
     -H "Content-Type: application/json" \
     -d '{"queue": "global_queue", "payload": {"value": "uji-node-failure"}}'
{"ok":true}ardi_dwi8104@ArdiDwiSaputra:/mnt/d/Tugas 2 SisTer$
```

```
curl -X POST http://localhost:8001/queue/pop \
     -H "Content-Type: application/json" \
     -d '{"queue": "global_queue"}'
```

```
ardi_dwi8104@ArdiDwiSaputra:/mnt/d/Tugas 2 SisTer$ curl -X POST http://localhost:8001/queue/pop \
     -H "Content-Type: application/json" \
     -d '{"queue": "global_queue"}'
{"ok":true,"item":{"value":"uji-node-failure"}}ardi_dwi8104@ArdiDwiSaputra:/mnt/d/Tugas 2 SisTer$
```

Walaupun salah satu node mati, node lainnya tetap bisa memproses antrian. Ini menunjukkan fault tolerance pada distributed queue.

5. Distributed Cache Coherence (MESI)

a. Write ke Cache

Perintah:

```
curl -X POST http://localhost:8001/cache/write \
     -H "Content-Type: application/json" \
     -d '{"key":"temperature","value":25}'
```

```
ardi_dwi8104@ArdiDwiSaputra:/mnt/d/Tugas 2 SisTer$ curl -X POST http://localhost:8001/cache/write \
     -H "Content-Type: application/json" \
     -d '{"key":"temperature","value":25}'
{"ok":true}ardi_dwi8104@ArdiDwiSaputra:/mnt/d/Tugas 2 SisTer$
```

Node 8001 menulis data ke cache. Protokol MESI memastikan node lain diberi tahu bahwa cache mereka harus diperbarui atau invalid.

b. Read dari Node Lain

Perintah:

```
curl "http://localhost:8003/cache/read?key=temperature"
```

```
ardi_dwi8104@ArdiDwiSaputra:/mnt/d/Tugas 2 SisTer$ curl "http://localhost:8003/cache/read?key=temperature"
{"ok":true,"value":{"state":"S","value":25}}ardi_dwi8104@ArdiDwiSaputra:/mnt/d/Tugas 2 SisTer$
```

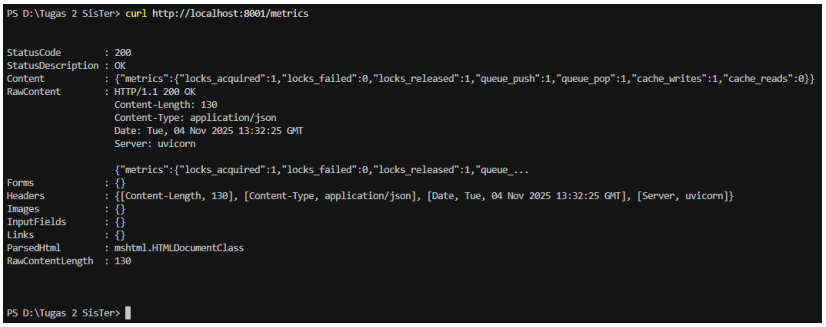
Node 8003 membaca nilai yang sama. Jika data di node 3 invalid atau outdated, node 3 akan mengambil versi terbaru dari node lain, sesuai aturan MESI.

6. **Monitoring & Metrics**

   Perintah:

   `curl http://localhost:8001/metrics`

   Perintah ini Digunakan untuk memonitor performa node, seperti jumlah request, waktu respon, cache hit/miss, dan statistik proses lainnya. Cocok untuk integrasi dengan Prometheus/Grafana.

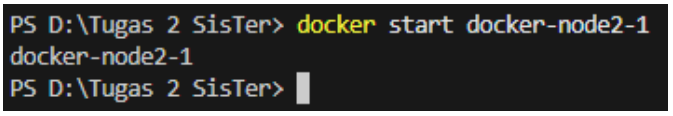


7. **Simulasi Recovery**

   a. Menyalakan Kembali Node

   Perintah:

   `docker start docker-node2-1`



   Node 2 dihidupkan kembali. Node ini akan melakukan sinkronisasi ulang dengan cluster.

   b. Menyalakan Kembali Node

   Perintah:

   `docker ps`

   Perintah ini digunakan untuk memastikan node sudah kembali online dan ikut beroperasi lagi dalam sistem.