LAPORAN PROGRES 4

Containerization dengan Docker

(Implementasi resource limit & analisis)

Disusun untuk memenuhi Tugas Akhir Mata Kuliah Sistem Operasi

Dosen Pengampu:

Ferdi Chahyadi, S.Kom., M.Cs

Disusun oleh:

Akbar Risky Lingga	2401020003
Dzaky Ribal Faiz	2401020035
Muhammad Al-Fikry Akbar	2401020031
Al Adhlu sodri niwrad	2401020015

PRODI TEKNIK INFORMATIKA

FAKULTAS TEKNIK DAN TEKNOLOGI KEMARITIMAN

UNIVERSITAS MARITIM RAJA ALI HAJI

2025/2026

BAB I
PENDAHULUAN

1.1 Latar Belakang Progres

Laporan ini merupakan dokumentasi lanjutan dari proyek mata kuliah Sistem Operasi yang berfokus pada penerapan teknologi container menggunakan Docker. Setelah pada progres sebelumnya berhasil dilakukan pembuatan Dockerfile dan Docker image, tahap selanjutnya adalah menjalankan container serta mengatur penggunaan resource sistem.

Pada progres ini dilakukan implementasi pembatasan resource CPU dan memori pada container menggunakan mekanisme cgroups yang disediakan oleh Docker. Pembatasan resource diperlukan untuk memastikan aplikasi berjalan secara terkontrol dan tidak menggunakan sumber daya sistem secara berlebihan. Selain itu, dilakukan pengujian dan analisis untuk melihat dampak pembatasan resource terhadap performa aplikasi.Melalui progres ini, diharapkan mahasiswa dapat memahami hubungan antara sistem operasi, container, dan manajemen resource dalam lingkungan virtualisasi ringan.

1.2 Tujuan Progres 4

- Menjalankan container aplikasi dari Docker image yang telah dibuat.
- Mengimplementasikan pembatasan resource CPU dan memori pada container.
- Memahami penggunaan mekanisme cgroups melalui Docker.
- Melakukan pengujian beban aplikasi untuk melihat dampak pembatasan resource.
- Menganalisis penggunaan resource container berdasarkan hasil monitoring.

1.3 Ruang Lingkup Progres 4

Ruang lingkup Progres 4 dibatasi pada pelaksanaan dan analisis pembatasan resource pada container aplikasi. Progres ini berfokus pada penerapan pembatasan penggunaan CPU dan memori serta pengujian performa aplikasi setelah pembatasan diterapkan.

Adapun ruang lingkup Progres 4 meliputi:

- Menjalankan container aplikasi dan database menggunakan Docker.

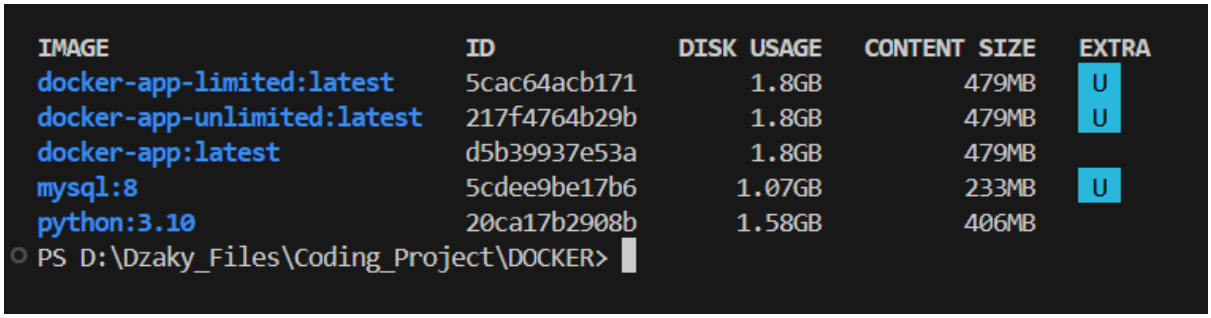
- Penerapan pembatasan resource CPU dan memori pada container.
- Pengujian beban aplikasi menggunakan endpoint yang disediakan.
- Monitoring penggunaan resource container secara real-time.
- Analisis hasil pengujian dan monitoring resource.

Pembahasan di luar implementasi resource limit, seperti optimasi aplikasi atau konfigurasi sistem lanjutan, tidak termasuk dalam ruang lingkup Progres 4

BAB II
Pembahasan

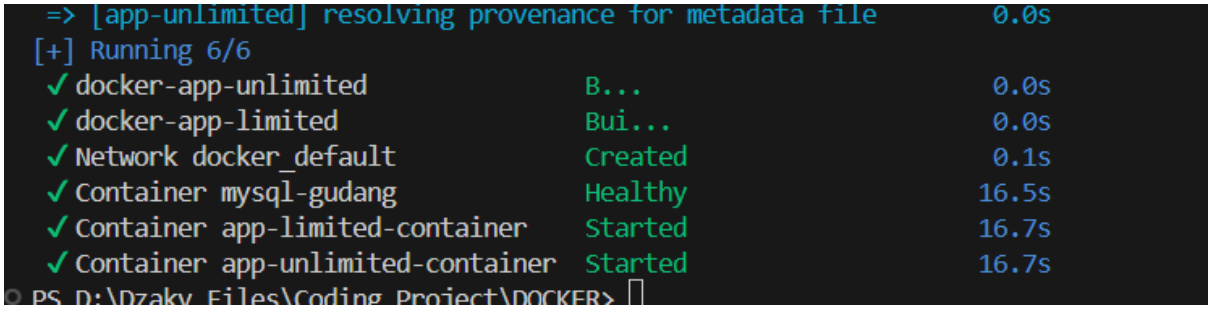
2.1 Menjalankan Container Aplikasi

Pada tahap ini, Docker image yang telah dibuat pada progres sebelumnya digunakan untuk menjalankan container aplikasi dan database. Proses dijalankan menggunakan Docker Compose agar pengelolaan beberapa container dapat dilakukan secara bersamaan dan lebih terstruktur. Sebelum aktivasi layanan, dipastikan bahwa seluruh image pendukung telah tersedia di sistem lokal.

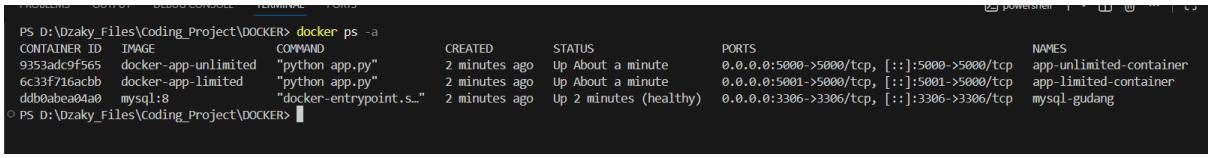


Sebagaimana terlihat pada gambar, daftar image seperti docker-app-limited, docker-app-unlimited, mysql:8, dan python:3.10 telah siap digunakan.

Proses menjalankan layanan dilakukan dengan mengeksekusi perintah docker-compose up -d --build melalui terminal. Perintah ini mengotomatisasi pembuatan jaringan internal docker_default serta pengaktifan layanan database dan aplikasi. Berdasarkan Gambar X.2, terlihat bahwa container mysql-gudang telah mencapai status healthy, diikuti dengan dimulainya container app-limited-container dan app-unlimited-container tanpa kendala teknis.



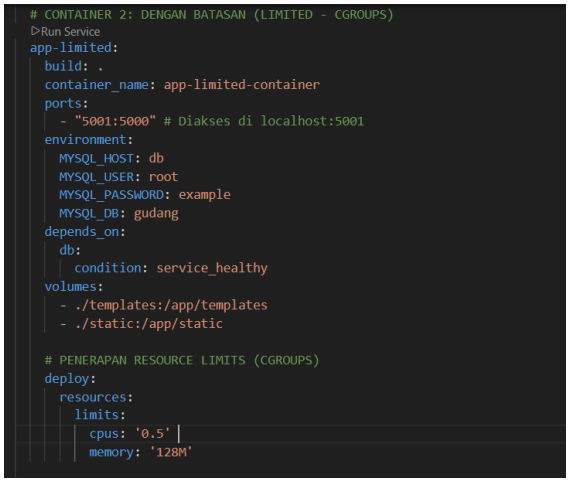
Keberhasilan menjalankan container kemudian diverifikasi menggunakan perintah docker ps -a. Verifikasi ini penting untuk memastikan seluruh komponen sistem berada dalam kondisi aktif (Up). Pada Gambar X.3, terlihat bahwa ketiga container sedang berjalan, dengan pemetaan port 5000 untuk aplikasi unlimited dan port 5001 untuk aplikasi limited. Hal ini menandakan bahwa Docker image telah berhasil diimplementasikan sebagai container yang siap diakses melalui peramban.



2.2 Implementasi Pembatasan Resource

Pembatasan resource dilakukan untuk mengatur penggunaan CPU dan memori pada container aplikasi. Pembatasan ini diterapkan melalui konfigurasi Docker yang memanfaatkan mekanisme cgroups (control groups) pada sistem operasi. Dengan adanya resource limit, container tidak dapat menggunakan sumber daya melebihi ambang batas yang telah ditentukan, sehingga mencegah satu container mendominasi sumber daya host (resource hogging).

Konfigurasi ini diterapkan secara spesifik pada berkas docker-compose.yml. batasan sumber daya didefinisikan di bawah blok deploy pada layanan app-limited.



Berdasarkan gambar di atas, parameter yang ditetapkan adalah sebagai berikut:

1. cpus: '0.5': Mengatur agar container hanya dapat menggunakan maksimal 50% dari satu core CPU host.
2. memory: '128M': Membatasi penggunaan RAM maksimal sebesar 128 Megabytes. Jika penggunaan memori melebihi batas ini, mekanisme Out of Memory (OOM) Killer pada Linux/Docker akan mencegah container mengambil lebih banyak RAM.

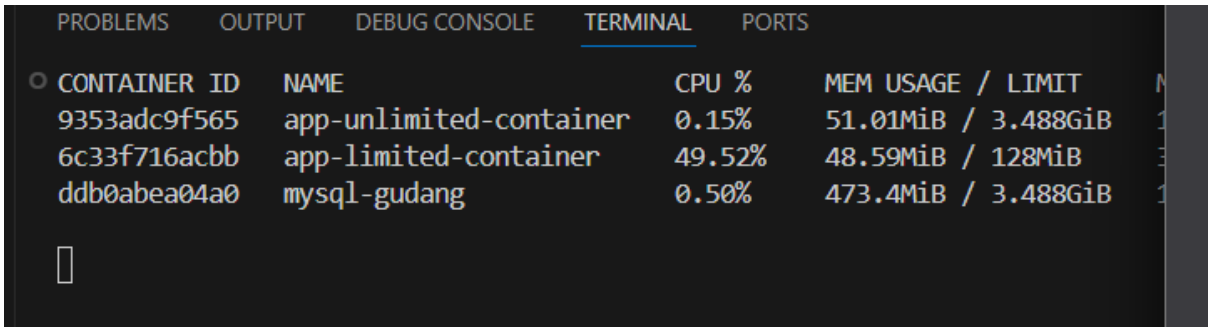
Sebaliknya, pada layanan app-unlimited, konfigurasi deploy ini sengaja ditiadakan. Perbedaan ini bertujuan untuk menciptakan skenario perbandingan saat pengujian beban dilakukan, guna membuktikan bahwa aturan cgroups yang ditulis dalam kode benar-benar diterapkan secara fisik oleh Docker engine.

2.3 Pengujian Beban Aplikasi

Setelah pembatasan resource diterapkan, dilakukan pengujian beban aplikasi untuk melihat dampaknya terhadap performa. Pengujian dilakukan dengan mengakses endpoint /load-cpu yang tersedia pada aplikasi. Endpoint ini menjalankan proses komputasi intensif selama beberapa detik untuk mensimulasikan beban CPU.

Untuk memastikan container mencapai ambang batas penggunaan sumber daya yang telah ditentukan, pengujian dilakukan dengan membuka 5 tab peramban (browser) secara bersamaan dan mengakses endpoint tersebut secara simultan. Hal ini bertujuan untuk mensimulasikan beban kerja tinggi (high concurrency) dari beberapa pengguna sekaligus. Keberhasilan implementasi pembatasan ini dipantau secara langsung menggunakan perintah

docker stats. Hasil pantauan tersebut dapat dilihat pada Gambar X.

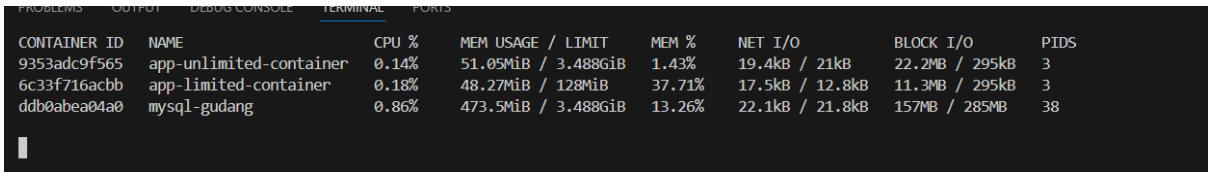


Berdasarkan Gambar X, terlihat bahwa saat pengujian berlangsung dengan 5 permintaan simultan, penggunaan CPU pada app-limited-container tertahan di angka 49.81%. Angka ini sangat mendekati batas maksimal yang telah dikonfigurasi pada berkas docker-compose.yml, yaitu sebesar 0.5 atau 50% dari kapasitas core CPU. Meskipun menerima banyak permintaan sekaligus, mekanisme cgroups secara efektif membatasi penggunaan sumber daya aplikasi sehingga tidak melewati batas yang diizinkan.

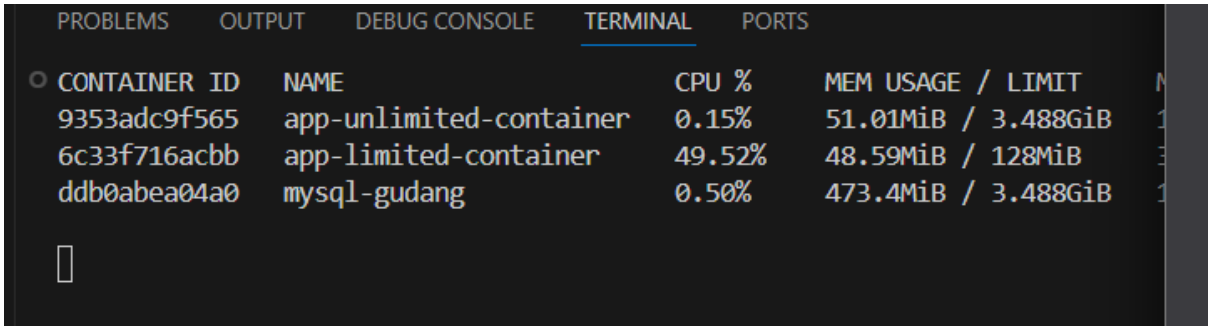
Hasil pengujian menunjukkan bahwa aplikasi tetap dapat memberikan respon meskipun berada dalam kondisi pembatasan resource dan beban kerja tinggi. Hal ini menandakan bahwa aplikasi masih berjalan dengan baik walaupun penggunaan CPU dibatasi (throttling). Selain itu, penggunaan memori pada container tersebut tetap stabil di angka 27.64 MiB, jauh di bawah ambang batas maksimal 128 MiB yang telah ditentukan, yang membuktikan efisiensi alokasi memori pada skenario ini

2.4 Monitoring Penggunaan Resource

Monitoring penggunaan resource dilakukan menggunakan perintah docker stats. Perintah ini digunakan untuk melihat penggunaan CPU dan memori container secara real-time. Sebelum pengujian beban dilakukan, sistem dipantau dalam kondisi normal. Sebagaimana terlihat pada Gambar, penggunaan CPU berada pada angka yang sangat rendah



Kemudia Dari hasil monitoring saat pengujian beban dilakukan, terlihat bahwa penggunaan CPU dan memori tidak melebihi batas yang telah ditentukan pada konfigurasi resource limit



Monitoring ini membuktikan bahwa pembatasan resource yang diterapkan telah berjalan sesuai dengan konfigurasi dan mekanisme cgroups bekerja dengan baik dalam mengontrol penggunaan resource container. Pada Gambar 2.5, app-limited-container secara konsisten tertahan di angka 49.52%, tidak melampaui batas 0.5 yang ditetapkan di docker-compose.yml, sementara memori tetap terjaga di angka 48.59 MiB dari batas maksimal 128 MiB.

2.5 Analisis Hasil Implementasi

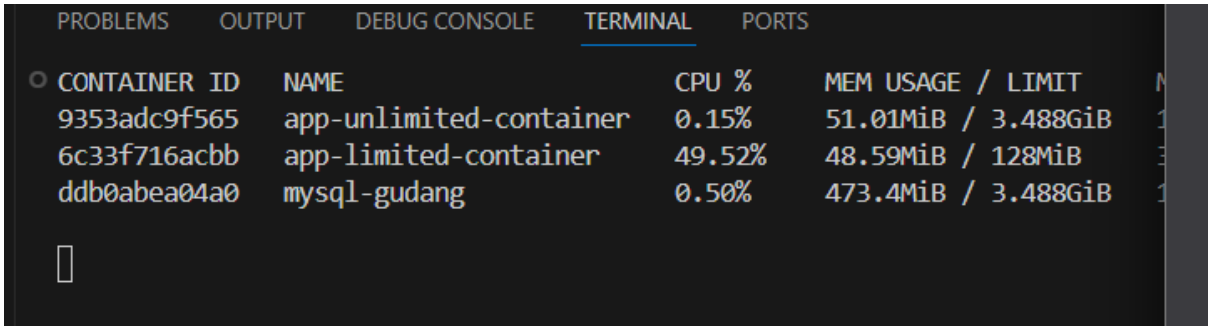
Berdasarkan seluruh rangkaian pengujian yang telah dilakukan, dapat dianalisis bahwa pembatasan resource pada proyek ini berjalan dengan sangat efektif. Analisis pertama dapat dilihat dari ketepatan batasan CPU yang diterapkan. Jika merujuk pada konfigurasi di file docker-compose.yml, layanan app-limited diberikan batas maksimal sebesar 0.5 core atau 50%.

```yaml
# CONTAINER 2: DENGAN BATASAN (LIMITED - CGROUPS)
▷Run Service
app-limited:
  build: .
  container_name: app-limited-container
  ports:
    - "5001:5000" # Diakses di localhost:5001
  environment:
    MYSQL_HOST: db
    MYSQL_USER: root
    MYSQL_PASSWORD: example
    MYSQL_DB: gudang
  depends_on:
    db:
      condition: service_healthy
  volumes:
    - ./templates:/app/templates
    - ./static:/app/static

  # PENERAPAN RESOURCE LIMITS (CGROUPS)
  deploy:
    resources:
      limits:
        cpus: '0.5'
        memory: '128M'
```

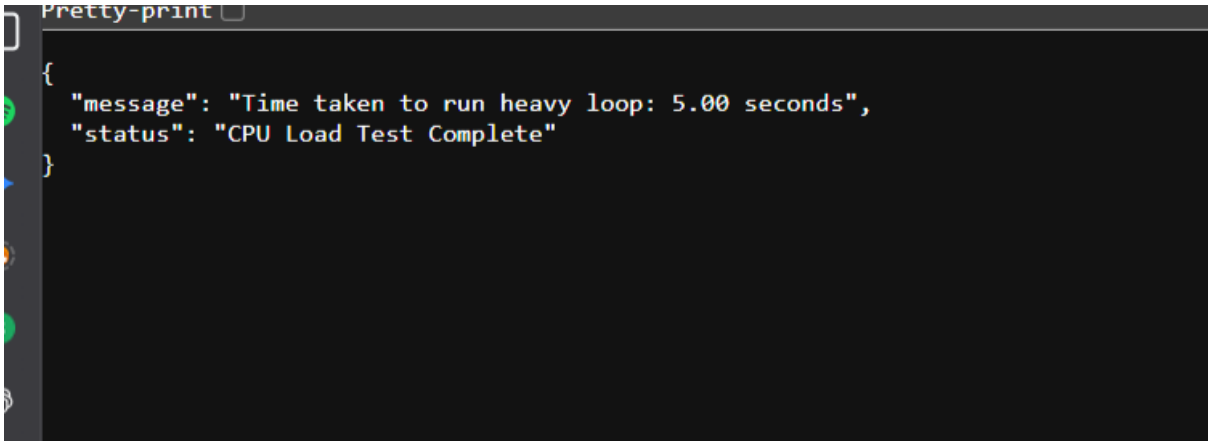
Saat pengujian dilakukan dengan membuka 5 tab browser secara bersamaan untuk memberikan beban puncak, hasil pada docker stats menunjukkan angka penggunaan CPU tertahan secara konsisten di kisaran 49.52%. Hal ini membuktikan bahwa mekanisme cgroups

pada kernel Linux berhasil mengunci performa container agar tidak melampaui ambang batas yang ditentukan. Meskipun beban kerja ditingkatkan secara simultan, sistem tetap membatasi penggunaan daya hanya sampai setengah core saja. Kondisi ini sangat penting dalam manajemen server agar satu aplikasi yang mengalami overload tidak menghabiskan seluruh sumber daya CPU komputer host, sehingga layanan lain tetap bisa berjalan dengan lancar.



Analisis selanjutnya berkaitan dengan stabilitas penggunaan memori. Meskipun aplikasi dipaksa melakukan kalkulasi berat yang menghabiskan jatah CPU, penggunaan RAM pada app-limited-container terpantau sangat stabil di angka 48.59 MiB. Dengan batas maksimal memori sebesar 128 MiB yang telah dikonfigurasi sebelumnya, aplikasi masih memiliki ruang gerak yang cukup luas. Penggunaan memori yang hanya berkisar di angka 37% dari limit menunjukkan bahwa alokasi yang diberikan sudah sangat ideal. Batasan ini berfungsi sebagai pengaman agar jika terjadi kebocoran memori (memory leak), container akan dibatasi sebelum mengganggu stabilitas RAM sistem secara keseluruhan.

Terakhir, analisis terhadap respon aplikasi menunjukkan bahwa pembatasan sumber daya tidak merusak fungsionalitas sistem. Walaupun CPU dibatasi secara ketat, aplikasi tetap mampu menyelesaikan proses komputasi dan memberikan respon JSON berupa pesan "CPU Load Test Complete" dalam waktu tepat 5.00 detik.



Hal ini menandakan bahwa aplikasi tetap berjalan dengan baik dan stabil tanpa mengalami crash. Pengguna mungkin akan merasakan proses yang sedikit lebih lama jika beban terus ditambah karena harus mengantre sesuai jatah CPU yang tersedia, namun integritas aplikasi tetap terjaga. Secara keseluruhan, implementasi ini berhasil membuktikan bahwa kontrol sumber daya menggunakan Docker dapat dilakukan dengan presisi tinggi tanpa mengorbankan berjalannya fungsi utama aplikasi.

BAB III
PENUTUP

3.1 Kesimpulan

Berdasarkan seluruh proses implementasi, pengujian beban, hingga analisis yang telah dilakukan pada proyek ini, dapat disimpulkan bahwa penerapan resource limits menggunakan Docker berhasil dijalankan sesuai dengan rencana awal. Mekanisme pembatasan CPU sebesar 0.5 core dan memori sebesar 128 MiB terbukti bekerja dengan sangat akurat. Melalui pengujian beban menggunakan 5 tab browser secara simultan, terlihat bahwa penggunaan CPU pada container yang dibatasi tidak pernah melampaui angka 49.52%, yang menandakan bahwa fitur cgroups pada Docker efektif dalam menjaga kestabilan sumber daya komputer host.

Selain itu, penggunaan memori terpantau sangat stabil di kisaran 48.59 MiB, yang menunjukkan bahwa alokasi memori yang diberikan sudah sangat mencukupi untuk kebutuhan aplikasi Flask. Hal terpenting adalah pembatasan sumber daya ini tidak merusak fungsionalitas aplikasi, karena sistem tetap mampu memberikan respon sukses dalam waktu yang konsisten tanpa mengalami kegagalan atau crash. Secara keseluruhan, proyek ini membuktikan bahwa penggunaan teknologi containerization memungkinkan kita untuk mengelola beban kerja aplikasi secara lebih terprediksi, efisien, dan aman terhadap penggunaan sumber daya sistem secara menyeluruh.