PROPOSAL PROJECT SISTEM OPERASI

IMPLEMENTASI KONTAINERISASI MENGGUNAKAN DOCKER: PEMBUATAN DOCKERFILE, MANAJEMEN MULTIKONTAINER DAN PEMBATASAN RESOURCE

Ganjil 2025 – 2026

2401020003	AKBAR RIZKI LINGGA
2401020015	AL ADLHU SODRI NIWRAD
2401020031	MUHAMMAD AL-FIKRY AKBAR
2401020035	DZAKY RIBAL FAIZ

PROGRAM STUDI TEKNIK INFORMATIKA

FAKULTAS TEKNIK DAN TEKNOLOGI KEMARITIMAN

UNIVERSITAS MARITIM RAJA ALI HAJI

2025

DAFTAR ISI

BAB I
PENDAHULUAN

1.1 Latar Belakang

Perkembangan teknologi komputasi modern menuntut aplikasi yang dapat dijalankan secara konsisten pada berbagai lingkungan tanpa perlu konfigurasi ulang. Perbedaan sistem operasi, versi library, dan dependency sering menyebabkan aplikasi gagal berjalan atau menghasilkan perilaku berbeda antara satu mesin dengan mesin lainnya (environment drift).

Untuk mengatasi hal tersebut, digunakan konsep containerization, yaitu teknik virtualisasi ringan yang memungkinkan aplikasi berjalan dalam lingkungan terisolasi (isolated execution environment) tanpa memerlukan sistem operasi lengkap sebagaimana pada virtual machine. Salah satu platform containerization paling banyak digunakan adalah Docker.

Docker menyediakan mekanisme untuk membangun image, yaitu paket berisi aplikasi beserta dependensinya, dan menjalankannya sebagai container, yaitu instance yang terisolasi dan ringan. Dalam konteks Sistem Operasi, Docker memanfaatkan fitur kernel Linux seperti namespaces dan cgroups untuk menciptakan isolasi proses, file system, networking, serta pembatasan penggunaan CPU dan memori.

Proyek ini dirancang sebagai bentuk penerapan langsung konsep-konsep tersebut melalui pembuatan Dockerfile, pembangunan image, menjalankan dua container, serta menerapkan pembatasan resource menggunakan cgroups melalui Docker. Dengan demikian, mahasiswa tidak hanya memahami teori, tetapi juga melakukan implementasi nyata dari virtualisasi tingkat sistem operasi.

1.2 Identifikasi Masalah

2. Aplikasi sering tidak dapat berjalan secara konsisten pada berbagai lingkungan.
3. Virtual machine memiliki overhead yang tinggi dan tidak efisien untuk aplikasi ringan.
4. Mahasiswa belum memahami pemanfaatan namespace dan cgroups untuk isolasi proses.

5. Belum ada pengalaman praktis dalam membangun dan menjalankan container Docker.

1.3 Rumusan Masalah

1. Bagaimana cara membuat Docker image menggunakan Dockerfile?
2. Bagaimana menjalankan dua container independen dari image yang sama?
3. Bagaimana menerapkan pembatasan CPU dan memori pada container?
4. Bagaimana memastikan container berjalan terisolasi secara konsisten?

1.4 Batasan Masalah

1. Aplikasi yang dikontainerisasi adalah program Python sederhana (mesinturing.py).
2. Fokus pada pembatasan resource CPU dan memori saja.
3. Tidak membahas jaringan Docker, volume, atau orchestrator seperti Kubernetes.
4. Tidak membahas image multistage atau optimasi lanjutan.

1.5 Tujuan

1. Memahami konsep containerization menggunakan Docker.
2. Membuat Dockerfile dan membangun Docker image.
3. Menjalankan dua container sekaligus.
4. Menerapkan pembatasan sumber daya CPU & memori.
5. Menghasilkan laporan akhir proyek containerization.

1. 6 Manfaat

1.6.1 Manfaat Teoritis

- Menambah pemahaman mengenai virtualisasi tingkat kernel (namespaces & cgroups).
- Memahami hubungan antara aplikasi, runtime, dan sistem operasi.

1.6.2 Manfaat Praktis

- Menambah keterampilan DevOps dasar.
- Melatih kemampuan deployment terpadu yang relevan dengan industri.
- Memahami kelebihan container dibanding virtual machine

BAB II

LANDASAN TEORITIS

2.1 Konsep Virtualisasi

Virtualisasi adalah proses menciptakan versi virtual dari perangkat fisik atau sumber daya komputasi seperti sistem operasi, server, atau aplikasi. Virtualisasi memungkinkan beberapa lingkungan berjalan pada satu sistem fisik yang sama tanpa saling mengganggu.

Dalam sistem operasi modern, virtualisasi bertujuan untuk:

- meningkatkan pemanfaatan hardware,
- isolasi proses,
- efisiensi pengelolaan resource,
- serta peningkatan keamanan.

Terdapat dua tipe utama virtualisasi:

1. **Virtualisasi penuh (Full Virtualization)**

   Menggunakan hypervisor seperti VMware, VirtualBox, atau Hyper-V. Setiap mesin virtual (VM) menjalankan sistem operasi lengkap dengan kernel masing-masing. Kelebihannya memberikan isolasi penuh, namun memiliki overhead besar.

2. **Virtualisasi tingkat OS (Operating System-Level Virtualization)**

   Digunakan pada Docker dan LXC. Metode ini tidak membuat sistem operasi baru, tetapi menjalankan proses yang diisolasi menggunakan kernel host. Karena tidak membuat OS baru, metode ini jauh lebih ringan dan cepat.

2.2 Container vs Virtual Machine

Virtual machine menjalankan OS lengkap di atas hypervisor. Container menjalankan aplikasi dalam isolasi tetapi berbagi kernel dengan host.

Aspek	Kontainer	Virual Machine
Karnel	Berbagi karnel host	Karnel Terpisah

Boot Time	Kurang dari 1 detik	Puluha detik
Resource Usage	Sangat rendah	tinggi
Use case	Deploy cepat,scalable	Menjalan OS berbeda

Karena lebih ringan, container ideal untuk aplikasi kecil, microservices, deployment cepat, dan CI/CD

2.3 Docker

Docker adalah platform containerization yang memudahkan pengembang untuk membangun, mengirim, dan menjalankan aplikasi dalam container. Docker memecahkan masalah klasik "*works on my machine*" dengan memberikan lingkungan yang konsisten di setiap perangkat.

Docker terdiri dari beberapa komponen penting:

1. **Docker Engine**
   Sebuah daemon (dockerd) yang mengelola image, container, jaringan, dan volume.
2. **Docker CLI (Command Line Interface)**
   Alat yang digunakan pengguna untuk menjalankan perintah seperti:
   docker build, docker run, docker images, dll.
3. **Docker Hub / Registry**
   Repository online tempat penyimpanan image.
4. **Dockerfile**
   File instruksi untuk membangun image.

Docker menggunakan dua teknologi inti Linux:

- **Namespaces** untuk isolasi proses.
- **cgroups** untuk pembatasan resource.

  Kombinasi keduanya memungkinkan Docker menjalankan container secara ringan namun aman.

2.4 Docker Image

Docker image adalah blueprint berisi:

- aplikasi,
- library,

- konfigurasi,
- environment,
- file system yang diperlukan aplikasi.

Image bersifat **read-only** dan disusun berlapis (*layered file system*). Setiap instruksi Dockerfile akan menambah satu layer baru.

Keuntungan utama Docker image:

- *Portability*: dapat dipindah antar perangkat tanpa konfigurasi ulang.
- *Reproducibility*: lingkungan selalu konsisten meskipun dijalankan di OS berbeda.
- *Layered caching*: mempercepat build image.

2.5 Docker Container

Container adalah instance berjalan dari image. Container bekerja seperti proses terisolasi yang memiliki:

- file system sendiri,
- environment variable,
- konfigurasi runtime,
- batas resource,
- namespace terpisah.

Container memiliki sifat:

- **Efisien** (berbagi kernel host)
- **Isolatif** (tidak saling mengganggu)
- **Ringan** (startup cepat)
- **Portabel**
- **Scalable**

Inilah alasan Docker menjadi fondasi utama microservices, DevOps, dan cloud computing.

2.6 Dockerfile

Dockerfile adalah file teks deklaratif berisi instruksi untuk membuat image. Instruksi umum:

- **FROM** → menentukan base image

- **WORKDIR** → direktori kerja
- **COPY** → menyalin file ke dalam image
- **RUN** → menjalankan perintah saat build
- **CMD** → perintah default ketika container dijalankan
- **EXPOSE** → membuka port

  Contoh Sederhana :

```
FROM python:3.10-slim
WORKDIR /app
COPY mesinturing.py .
CMD ["python3", "mesinturing.py"]
```

Dengan Dockerfile ini, image dapat dibangun secara otomatis dan konsisten.

2.7 cgroups dan Resource Limiting

Control Groups (cgroups) adalah fitur kernel Linux untuk:

- membatasi (*limit*)
- memonitor (*monitor*)
- memprioritaskan (*prioritize*)
- mengisolasi penggunaan resource (*isolate*)

Proses dapat dibatasi dalam hal:

- CPU
- Memori
- Disk I/O
- Network I/O

Dalam Docker digunakan untuk:

- *--cpus* → menentukan jumlah core yang boleh digunakan container
- *--memory* → menetapkan batas RAM

Contoh Sederhana :

```
docker run --cpus="1.0" --memory="256m" image-name
```

Dengan cgroups, Docker dapat mengendalikan resource tanpa mengganggu host.

2.8 Containerization dalam Pengembanggan Modern

Container saat ini menjadi standar industri karena:

- deployment cepat
- kemudahan scaling
- kompatibilitas tinggi
- mudah di-CI/CD
- mendukung microservices
- dapat dijalankan di cloud, server, maupun laptop

Docker menjadi fondasi untuk teknologi lebih besar seperti:

- Kubernetes
- Docker Swarm
- OpenShift
- AWS ECS

Oleh sebab itu, pemahaman Docker penting untuk mahasiswa IT modern

BAB III

METODOLOGI PROYEK

3.1 Gambaran Umum Proyek

Proyek ini berfokus pada pembuatan lingkungan terisolasi berbasis container. Mahasiswa akan membuat Dockerfile, membangun image, menjalankan dua container, serta menerapkan batas resource CPU dan memori. Seluruh proses dilakukan berdasarkan konsep virtualisasi tingkat sistem operasi.

3.2 Arsitektur Sistem

Arsitektur yang direncanakan mencakup komponen berikut:

1. **Host Machine** — perangkat fisik yang menjalankan Docker Engine.
2. **Docker Engine** — sistem utama yang mengelola image dan container.
3. **Docker Image** — blueprint aplikasi Python.
4. **Container 1** — dibatasi CPU 0.5 core & memori 128 MB.
5. **Container 2** — dibatasi CPU 1 core & memori 256 MB.

Arsitektur ini bersifat konseptual dan digunakan sebagai acuan implementasi pada laporan akhir.

3.3 Analisis Kebutuhan

1.3.1 Kebutuhan Perangkat Keras

- CPU minimal dual-core
- RAM minimal 4 GB
- Ruang penyimpanan minimal 2 GB

1.3.2 Kebutuhan Perangkat Lunak

- Docker Desktop / Docker Engine
- Python 3
- Terminal / Command Prompt
- Text Editor (misalnya VS Code)

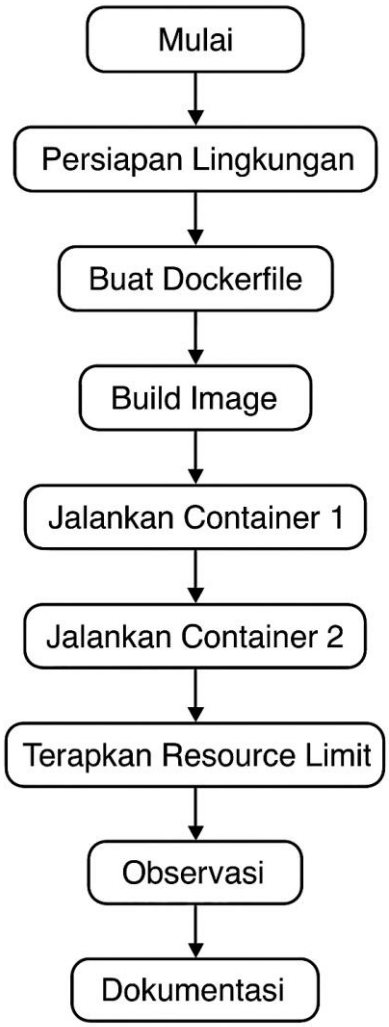
1.3.3 Kebutuhan Fungsional

- Sistem dapat membangun image dari Dockerfile.
- Sistem dapat menjalankan dua container secara bersamaan.
- Sistem dapat membatasi penggunaan CPU/memori container.

3.4 Tahapan Pelaksanaan

1. Persiapan lingkungan dan instalasi Docker.
2. Pembuatan script Python sederhana.
3. Penyusunan Dockerfile.
4. Pembentukan image menggunakan docker build.
5. Menjalankan container pertama dan kedua.
6. Menerapkan pembatasan resource.
7. Melakukan uji coba dan analisis.
8. Menyusun laporan akhir proyek.

3.5 Rancangan Proses



3.6 Analisis Resiko Proyek

Resiko	Dampak	Mitigasi
Docker gagal terinstal	Proyek tidak berjalan	Gunakan Docker Decsktop versi stabil
Konflik port atau proses	Container tidak berjalan	Stop container/daemon lain
Image tidak dapat dibangun	Proyek tertunda	Validasi Dockerfile sebelum build
Pembatasan resource membuat aplikasi crash	Performa tergangu	Sesuaikan batas memori $ CPU
Sistem kurang kuat saat menjalankan dua container	Lag atau crash	Gunakan resource host yang cukup

BAB 4
PENUTUP

4.1 Kesimpulan

Proposal ini disusun sebagai rencana pelaksanaan proyek Containerization menggunakan Docker. Proyek ini berfokus pada pembuatan Dockerfile, pembangunan Docker image, menjalankan dua container, serta menerapkan pembatasan resource CPU dan memori menggunakan cgroups. Melalui proyek ini mahasiswa akan memperoleh pemahaman teknis mengenai isolasi proses, virtualisasi tingkat OS, dan pengelolaan resource.

4.2 Harapan

Semoga proyek ini dapat meningkatkan kompetensi mahasiswa dalam bidang DevOps, sistem operasi, dan deployment modern berbasis container.

DAFTAR PUSTAKA

1. Docker Inc. (2023). *Docker Documentation*. Diakses dari https://docs.docker.com
2. Merkel, Dirk. (2014). “Docker: Lightweight Linux Containers for Consistent Development and Deployment.” *Linux Journal*, 2014(239).
3. Turnbull, James. (2014). *The Docker Book: Containerization Is the New Virtualization*. James Turnbull Publishing.
4. Kerrisk, Michael. (2010). *The Linux Programming Interface: A Linux and UNIX System Programming Handbook*. No Starch Press.
5. Red Hat Inc. (2020). *Understanding Linux Control Groups (cgroups)*. Red Hat Documentation.
6. IBM Cloud. (2022). *Containers and Virtualization Concepts*. IBM Knowledge Center.
7. Mell, P. & Grance, T. (2011). *The NIST Definition of Cloud Computing*. NIST Special Publication 800-145.
8. Python Software Foundation. (2023). *Python 3 Documentation*.
9. Soltesz, S., et al. (2007). “Container-based Operating System Virtualization: A Scalable, High-performance Alternative to Hypervisors.” *EuroSys Conference*.