Bayu Widodo

08 April 2025

## Daftar Isi

**Daftar Gambar**



Sekolah Vokasi
School of Vocational Studies

## 8 Docker Ubuntu 24.04

"In a world of complexity, Docker keeps deployment simple."

Docker adalah sebuah tool open-source yang dirancang untuk menyederhanakan proses deployment aplikasi. Bagi para pengembang, khususnya di bidang Rekayasa Perangkat Lunak dan Teknologi Rekayasa Komputer, Docker merupakan solusi modern yang memudahkan proses pengemasan, pengujian, dan eksekusi aplikasi lintas platform tanpa kekhawatiran terhadap perbedaan lingkungan sistem operasi.

Docker memungkinkan aplikasi dan seluruh dependensinya dikemas dalam satu unit terisolasi yang disebut container. Unit ini dapat dijalankan secara fleksibel di berbagai sistem, seperti VPS hosting, platform cloud, maupun dedicated machine, tanpa memerlukan konfigurasi ulang lingkungan.

Keunggulan Docker:

1. Ringan dan Cepat:
   Docker memiliki overhead yang jauh lebih kecil dibandingkan virtual machine karena berjalan langsung di atas sistem operasi host.
2. Konsistensi Lingkungan:
   Semua dependensi, library, dan konfigurasi disatukan dalam satu container, mengurangi masalah "works on my machine".
3. Skalabilitas:
   Memudahkan pengelolaan aplikasi berskala besar dengan arsitektur microservices.
4. Integrasi CI/CD:
   Sangat kompatibel dengan pipeline CI/CD (Continuous Integration,Continuous Delivery) dalam pengembangan modern. Docker sangat kompatibel dengan pipeline CI/CD dalam pengembangan perangkat lunak modern, karena memungkinkan proses build, test, dan deployment dilakukan secara otomatis, konsisten, dan efisien di berbagai lingkungan.

### 8.1 Peran Docker dalam Pengembangan dan Rekayasa Teknologi Modern

Dalam era transformasi digital yang semakin pesat, kebutuhan akan proses pengembangan perangkat lunak dan rekayasa sistem yang efisien, konsisten, serta skalabel menjadi semakin penting. Docker, sebagai platform berbasis container, hadir sebagai solusi inovatif yang mendukung proses tersebut dengan memungkinkan aplikasi dikemas bersama seluruh dependensinya dalam satu lingkungan yang ringan dan terisolasi.

#### 8.1.1 Peran Docker dalam Rekayasa Perangkat Lunak (RPL)

Dalam bidang Rekayasa Perangkat Lunak (RPL), Docker memberikan kontribusi besar terhadap efisiensi dan konsistensi dalam proses build, test, dan deployment aplikasi. Dengan menggunakan container, pengembang dapat memastikan bahwa aplikasi yang dibangun akan berjalan secara identik di berbagai lingkungan baik di mesin lokal, server pengujian, maupun di lingkungan produksi. Hal ini mengurangi risiko "it works on my machine" yang sering terjadi saat perangkat lunak dipindahkan antar tim atau tahap pengembangan.

Docker juga sangat mendukung pendekatan pengembangan perangkat lunak modern seperti Agile dan DevOps. Dalam konteks Agile, proses iteratif yang cepat membutuhkan infrastruktur yang fleksibel, dan Docker menjawab kebutuhan ini dengan kemampuan untuk menjalankan lingkungan pengembangan dan pengujian secara instan. Sementara dalam praktik DevOps, Docker memfasilitasi CI/CD dengan menyediakan lingkungan yang dapat direplikasi untuk membangun, menguji, dan mendistribusikan perangkat

lunak secara otomatis.

Selain itu, Docker juga memungkinkan pengembangan kolaboratif yang lebih efektif. Setiap anggota tim dapat bekerja dengan konfigurasi lingkungan yang sama, tanpa perlu mengatur ulang dependensi atau sistem operasi. Pengujian juga menjadi lebih efisien karena dapat dilakukan di dalam container yang terisolasi, sehingga menghindari konflik konfigurasi antar proyek atau modul. Dengan kata lain, Docker menjadi jembatan penting untuk mencapai automasi, konsistensi, dan skalabilitas dalam siklus hidup perangkat lunak.

### 8.1.2 Peran Docker dalam Teknologi Rekayasa Komputer (TRK)

Dalam bidang Teknologi Rekayasa Komputer (TRK), Docker memainkan peran yang sangat penting sebagai alat bantu dalam pengembangan, pengujian, dan pengelolaan sistem berbasis teknologi modern. Salah satu penerapan utama Docker adalah dalam simulasi sistem terdistribusi dan layanan mikro (microservices). Dengan menggunakan container, mahasiswa atau praktisi dapat membangun dan menguji arsitektur sistem yang terdiri dari banyak layanan yang berjalan secara terisolasi namun dapat saling berkomunikasi melalui jaringan virtual Docker. Ini sangat berguna dalam memahami konsep komunikasi antar node dalam jaringan, load balancing, maupun service discovery.

Selain itu, Docker juga dapat dimanfaatkan untuk melakukan pengujian performa jaringan dan server melalui skenario kontainer yang dapat dikonfigurasi sesuai kebutuhan. Misalnya, pengujian terhadap bottleneck jaringan, latensi antar layanan, atau kemampuan server dalam menangani permintaan tinggi dapat dilakukan secara fleksibel tanpa harus menggunakan perangkat keras fisik tambahan.

Docker juga mempermudah deployment otomatis aplikasi IoT atau embedded system yang saling berinteraksi. Dengan mengemas masing-masing komponen aplikasi ke dalam container, pengembang dapat menciptakan lingkungan uji coba yang merepresentasikan sistem IoT skala kecil atau besar, lengkap dengan komunikasi antar node (seperti sensor, gateway, dan server cloud) secara real-time.

Lebih jauh lagi, Docker mendukung pendekatan modern dalam pengelolaan infrastruktur, yaitu Infrastructure as Code (IaC). Dengan memanfaatkan file konfigurasi seperti Dockerfile dan Docker Compose, infrastruktur sistem dapat didefinisikan dalam bentuk kode yang mudah dikelola, dilacak, dan direproduksi. Hal ini meningkatkan efisiensi dan akurasi dalam pengelolaan sistem, serta sangat cocok untuk mendukung pendekatan DevOps dalam proyek rekayasa komputer.

## 8.2 Tujuan Modul

Seiring berkembangnya teknologi komputasi dan kebutuhan akan pengembangan sistem yang cepat, ringan, dan konsisten di berbagai platform, keterampilan dalam menggunakan Docker menjadi semakin penting bagi mahasiswa dan profesional di bidang teknologi. Docker tidak hanya mempermudah proses instalasi dan deployment, tetapi juga membuka peluang untuk menerapkan praktik terbaik dalam pengembangan perangkat lunak modern, seperti integrasi berkelanjutan, layanan mikro, dan infrastruktur berbasis kode.

Modul ini dirancang untuk membantu Anda memahami konsep dasar Docker dan cara penerapannya dalam berbagai konteks, baik di bidang Rekayasa Perangkat Lunak (RPL) maupun Teknologi Rekayasa Komputer (TRK). Dengan mempelajari modul ini, Anda akan mampu membangun, mengelola, dan mengoptimalkan container Docker untuk mendukung pengembangan aplikasi yang lebih efisien, terstandarisasi, dan siap digunakan dalam skala industri. Modul ini akan memandu Anda:

1. Mengenal konsep dasar Docker dan container.

2. Membuat dan menjalankan container.
3. Menggunakan Dockerfile untuk otomatisasi proses build.
4. Menyusun arsitektur aplikasi berbasis container.
5. Menerapkan *best practices* dalam pengembangan software dengan Docker.
6. Menghubungkan Docker dengan topik-topik penting di RPL dan TRK, termasuk DevOps, CI/CD, dan sistem cloud-native.

Mengapa Memilih Ubuntu Server 24.0x sebagai Platform Docker ? Ubuntu Server 24.0x merupakan salah satu distribusi Linux yang banyak digunakan dalam lingkungan server profesional maupun pengembangan karena keandalannya dalam hal stabilitas sistem, keamanan tingkat tinggi, dan dukungan jangka panjang (LTS). Kombinasi ini menjadikannya pilihan ideal bagi organisasi dan pengembang yang membutuhkan sistem operasi yang solid dan minim gangguan untuk kebutuhan layanan jangka panjang.

Dalam konteks penerapan Docker, Ubuntu Server 24.0x memberikan lingkungan yang optimal dan ringan untuk menjalankan aplikasi berbasis kontainer. Docker dapat diinstal dan dikonfigurasi dengan cepat di Ubuntu, berkat ekosistem open-source yang matang dan dukungan komunitas yang luas. Dengan memanfaatkan Docker di atas Ubuntu Server, pengguna dapat mengelola aplikasi dengan lebih efisien, mengurangi kompleksitas proses deployment, dan meningkatkan portabilitas antar lingkungan pengembangan, staging, dan produksi.

Lebih dari itu, kombinasi Docker dan Ubuntu memungkinkan pengembangan arsitektur mikroservis yang skalabel, serta integrasi yang mulus dengan layanan cloud maupun server on-premises. Hal ini sangat sejalan dengan kebutuhan modern dalam Rekayasa Perangkat Lunak (RPL) dan Teknologi Rekayasa Komputer (TRK) yang menuntut solusi fleksibel, cepat, dan andal.

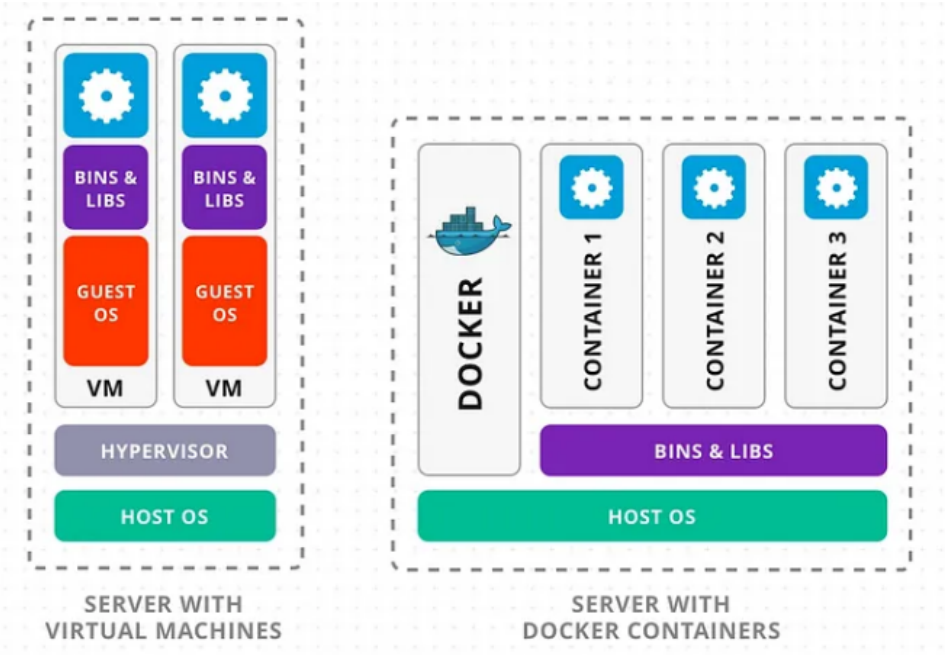
### 8.3 Container vs Virtual Machine

Berbeda dengan mesin virtual (VM), yang menjalankan sistem operasi lengkap dengan kernel dan virtualisasi perangkat kerasnya sendiri, *container berbagi kernel dengan sistem operasi host*, menjadikannya lebih ringan dan lebih cepat untuk dijalankan. VM membutuhkan hypervisor[1] untuk mengelola beberapa instance sistem operasi yang berjalan secara terisolasi, sedangkan Docker menggunakan fitur isolasi dari kernel Linux seperti cgroups dan namespaces untuk menjalankan banyak kontainer dalam satu sistem operasi host.

Karena tidak perlu mem-boot sistem operasi secara terpisah, kontainer dapat dimulai dalam hitungan detik, sementara VM biasanya membutuhkan lebih banyak waktu untuk booting. Selain itu, kontainer menggunakan lebih sedikit sumber daya dibandingkan VM, karena tidak ada redundansi dalam sistem operasi yang berjalan di setiap instance. Hal ini membuat Docker lebih efisien dalam hal penggunaan memori dan CPU, serta lebih fleksibel dalam deployment aplikasi di berbagai lingkungan, baik lokal, cloud, maupun hybrid.

Untuk lebih memahami perbedaan fundamental antara VM dan Docker dalam hal arsitektur, penggunaan sumber daya, serta mekanisme isolasi aplikasi, berikut adalah ilustrasi sederhana yang menggambarkan bagaimana masing-masing teknologi bekerja, mulai dari bagaimana mereka memanfaatkan perangkat keras, pengelolaan sistem operasi, hingga cara aplikasi dijalankan di dalamnya.

[1]Hypervisor (VM Manager) adalah perangkat lunak, firmware, atau perangkat keras yang memungkinkan pembuatan dan pengelolaan mesin virtual (VM). Hypervisor bertindak sebagai lapisan abstraksi antara perangkat keras fisik dan VM, memungkinkan beberapa sistem operasi berjalan secara independen pada satu mesin fisik dengan membagi sumber daya seperti CPU, memori, dan penyimpanan.




Gambar 1: VM vs Kontainer

Dari gambar tersebut, dapat dilihat bahwa VM membutuhkan sistem operasi tersendiri di setiap instansnya, sementara kontainer berbagi kernel dengan sistem operasi host. Hal ini membuat kontainer lebih ringan dan efisien dibandingkan dengan VM, terutama dalam hal konsumsi sumber daya dan waktu startup.

### 8.4 Konsep Dasar Docker

Docker adalah platform yang memungkinkan developer untuk membuat, mengemas, dan menjalankan aplikasi di dalam sebuah container. Kontainer mencakup aplikasi beserta dependensi, pustaka, dan konfigurasinya, sehingga memastikan konsistensi dalam eksekusi tanpa bergantung pada sistem operasi tertentu.

Teknologi ini memungkinkan efisiensi dalam penggunaan sumber daya serta kemudahan dalam deployment di lingkungan cloud maupun on-premises. Lihat kembali Gambar 1 untuk memahami bagaimana Docker bekerja dibandingkan dengan mesin virtual.

Docker tidak membutuhkan OS tambahan di dalam container, karena container berjalan langsung di atas kernel sistem host, inilah yang membuat Docker ringan dan cepat.

#### 8.4.1 Arsitektur dan Komponen Docker

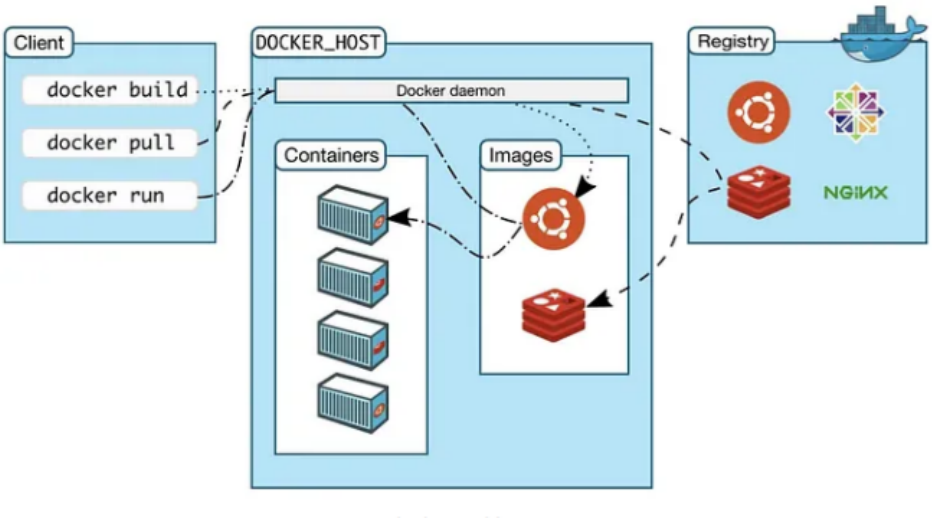
Untuk memahami cara kerja Docker secara menyeluruh, penting untuk mengenal komponen-komponen utama dalam arsitekturnya. Komponen sentral dari Docker adalah **Docker Engine**, yaitu layanan inti yang bertanggung jawab untuk membangun, menjalankan, dan mengelola container. Docker Engine bekerja di latar belakang sebagai daemon (sering disebut *Dockerd*) dan menyediakan antarmuka bagi pengguna melalui perintah CLI (Docker).

Selanjutnya, terdapat **Docker Image**, yaitu blueprint atau cetakan dari container. Image ini bersifat read-only dan berisi semua hal yang dibutuhkan untuk menjalankan sebuah aplikasi, seperti kode pro-

gram, dependensi, library, konfigurasi, dan sebagainya. Dari image inilah kemudian Docker membentuk **Docker Container**, yaitu instansi (runtime) yang aktif dari sebuah image. Container adalah unit terkecil dalam Docker yang dapat dieksekusi dan bersifat ringan karena berjalan langsung di atas sistem operasi host tanpa perlu hypervisor seperti pada virtual machine.

Untuk membuat sebuah Docker image, digunakan **Dockerfile**, yaitu file teks sederhana yang berisi instruksi-instruksi tahap demi tahap mengenai bagaimana sebuah image dibangun. Instruksi dalam Dockerfile mencakup perintah untuk menginstall paket, menyalin file, menjalankan konfigurasi tertentu, dan lain-lain.

Untuk memberikan gambaran yang lebih jelas tentang hubungan antar komponen tersebut, silakan lihat Gambar 2. Gambar tersebut menunjukkan bagaimana Docker Engine berinteraksi dengan Dockerfile, Image, dan Container dalam alur kerja containerisasi.




Gambar 2: Arsitektur Docker

1. Docker Engine:
   - Komponen inti yang memungkinkan pengelolaan kontainer. Terdiri dari Docker daemon (Dockerd) yang berjalan di latar belakang dan Docker CLI (Docker) sebagai antarmuka baris perintah untuk berinteraksi dengan Docker.
     - Docker Client (CLI): Docker Client adalah antarmuka baris perintah (CLI) yang memungkinkan pengguna berinteraksi dengan Docker. CLI ini mengirimkan perintah ke Docker Daemon untuk mengelola container, image, jaringan, dan sumber daya Docker lainnya. Pengguna menggunakan perintah seperti Docker run, Docker build, dan lainnya.
     - Docker Daemon (Dockerd): Docker Daemon adalah inti dari Docker yang berjalan sebagai layanan latar belakang di mesin host. Ia bertanggung jawab untuk mengelola container, image, dan objek Docker lainnya. Docker Daemon menerima perintah dari Docker Client melalui REST API dan mengeksekusinya sesuai instruksi.
2. Docker Containers:
   - Container adalah instance runtime yang terisolasi, ringan, dan portabel dari sebuah Docker image. Setiap container menjalankan aplikasi beserta dependensinya dalam lingkungan yang terisolasi. Container dibuat dari Docker images.
3. Docker Images:

- Docker Image adalah template hanya-baca yang berisi kode aplikasi, pustaka, dependensi, dan konfigurasi yang diperlukan untuk menjalankan sebuah container. Image merupakan dasar dalam pembuatan container.
- Image berisi semua yang diperlukan untuk menjalankan aplikasi, termasuk sistem operasi, dependensi, dan kode aplikasi.
  - Dockerfile adalah skrip yang berisi instruksi untuk membangun sebuah image.
  - Image dapat disimpan dan diambil dari Docker Hub atau registry lain.

4. Docker Registry (Docker Hub):
   - Docker Registry adalah repositori yang menyimpan Docker images. Docker Hub adalah registry publik bawaan yang dikelola oleh Docker, tempat pengembang dapat mengakses dan berbagi image Docker. Registry privat juga dapat digunakan untuk penyimpanan image yang lebih aman.

### 8.4.2 Interaksi Antar Komponen Docker

Docker menggunakan arsitektur client-server di mana setiap komponen memiliki peran spesifik dan berinteraksi satu sama lain melalui mekanisme tertentu. Dengan memahami hubungan antar komponen ini ⇒ Docker Engine, Image, Container, dan Dockerfile, pengguna akan lebih mudah dalam merancang, mengembangkan, dan mengelola lingkungan aplikasi yang efisien dan portabel menggunakan Docker. Berikut adalah alur interaksi antara komponen utama Docker:

1. Pengguna berinteraksi dengan Docker melalui Docker Client (CLI).
   - Pengguna menjalankan perintah seperti Docker run, Docker build, atau Docker pull.
   - Docker Client menerjemahkan perintah ini dan mengirimkannya ke Docker Daemon melalui REST API atau soket UNIX.
2. Docker Daemon menerima dan memproses perintah dari Docker Client.
   - Jika perintah terkait dengan image (misalnya Docker pull), Docker Daemon akan mencari image tersebut di Docker Registry (Docker Hub atau registry privat).
   - Jika perintah terkait dengan container (misalnya Docker run), Docker Daemon akan membuat container baru berdasarkan image yang ada atau yang diunduh dari Docker Registry.
3. Docker Registry digunakan untuk menyimpan dan mengambil Docker Images.
   - Jika image yang dibutuhkan tidak ada di lokal, Docker Daemon akan mengunduhnya dari Docker Hub atau registry yang dikonfigurasi.
   - Jika pengguna ingin menyimpan image yang telah dibuat, mereka dapat mengunggahnya (Docker push) ke registry untuk dibagikan atau digunakan kembali.
4. Docker Images digunakan untuk membuat Docker Containers.
   - Setiap container dibuat berdasarkan image tertentu yang berisi aplikasi dan dependensinya.
   - Docker Daemon mengelola lifecycle container, termasuk menjalankan (Docker start), menghentikan (Docker stop), dan menghapusnya (Docker rm).
5. Docker Containers berjalan sebagai proses yang terisolasi di atas host machine.
   - Setiap container memiliki sistem file sendiri, jaringan, dan lingkungan runtime yang terisolasi.
   - Pengguna dapat mengakses container melalui Docker Client untuk melakukan debugging, logging, atau menjalankan aplikasi di dalamnya.

## 8.5 Instalasi dan Menjalankan Docker

Sebelum mulai menjalankan perintah Docker, pastikan sistem Anda telah siap untuk instalasi. Langkah-langkah persiapan ini mencakup memperbarui indeks paket, menginstal Docker jika belum terpasang,

serta mengonfigurasi Docker agar berjalan secara otomatis saat sistem dinyalakan. Selain itu, menambahkan pengguna ke dalam grup Docker akan memungkinkan penggunaan perintah Docker tanpa harus memasukkan sudo setiap kali.

1. Memperbarui Indeks Paket
   Perbaruhi (update) daftar paket pada sistem Anda untuk memastikan akses ke versi terbaru dari perangkat lunak yang tersedia.

```
sudo apt update
sudo apt dist-upgrade
```

2. Menginstal Dependensi Paket
   Sebelum menginstal Docker, Anda perlu menginstal beberapa paket pendukung yang diperlukan. Paket-paket ini memungkinkan Ubuntu 24.04 untuk mengakses repositori Docker secara aman melalui protokol HTTPS. Instalasi dependensi ini memastikan sistem dapat mengunduh dan memperbarui paket Docker dari sumber resminya tanpa kendala.

```
sudo apt install apt-transport-https ca-certificates curl
sudo apt install software-properties-common
```

3. Menambahkan Kunci GPG
   Untuk memastikan keaslian dan keamanan paket yang diunduh dari repositori resmi Docker, Anda perlu menambahkan kunci GPG. Kunci ini memungkinkan sistem Ubuntu 24.04 memverifikasi integritas paket sebelum menginstalnya. Gunakan perintah berikut untuk menambahkan kunci GPG Docker ke sistem Anda.

```
sudo curl -fsSL https://download.Docker.com/linux/ubuntu/gpg | /
sudo apt-key add -
```

4. Menambahkan Repositori Docker
   Agar dapat menginstal Docker di Ubuntu 24.04, Anda perlu menambahkan repositori resmi Docker ke dalam sumber APT. Dengan menambahkan repositori ini, sistem akan selalu mendapatkan versi terbaru dan stabil dari Docker langsung dari sumber resminya. Gunakan perintah berikut untuk menambahkan repositori Docker ke sistem Anda.

```
sudo  add-apt-repository "deb [arch=amd64] /
https://download.Docker.com/linux/ubuntu $(lsb_release -cs) stable"
```

5. Menginstal Docker
   Setelah repositori Docker berhasil ditambahkan ke sistem, kini Anda siap untuk menginstal Docker di Ubuntu 24.04. Proses instalasi ini akan mengunduh dan memasang versi terbaru Docker dari repositori resmi, memastikan Anda mendapatkan fitur dan pembaruan terbaru. Gunakan perintah berikut untuk memulai instalasi Docker.

```
sudo apt install docker-ce docker-ce-cli containerd.io
```

6. Memverifikasi Paket
   Setelah proses instalasi selesai, penting untuk memverifikasi bahwa Docker telah terinstal dengan benar dan berjalan di sistem. Anda dapat memeriksa versi Docker yang terpasang dengan menggunakan perintah berikut. Hal ini memastikan bahwa Anda telah menginstal versi terbaru dan siap untuk menggunakan Docker.

```
    docker info
    #atau
    docker --version
    #atau
    docker version
    #memverivikasi apakah docker compose terinstall
    docker-compose version
```

7. Memeriksa Status Layanan
   Setelah Docker terinstal, Anda perlu memastikan bahwa layanan Docker berjalan dengan benar di sistem. Gunakan perintah berikut untuk memeriksa status layanan Docker dan memastikan bahwa Docker telah aktif dan siap digunakan.

```
    sudo systemctl status docker
```

   Contoh hasil:

```
 docker.service - Docker Application Container Engine
   Loaded: loaded (/lib/systemd/system/Docker.service; enabled; vendor preset>
   Active: active (running) since Sun 2025-03-23 10:17:56 WIB; 56min ago
TriggeredBy:  docker.socket
     Docs: https://docs.Docker.com
 Main PID: 2011 (dockerd)
    Tasks: 14
   Memory: 95.1M
      CPU: 1.957s
   CGroup: /system.slice/docker.service
           2011 /usr/bin/Dockerd -H fd:// --containerd=/run/containerd/cont>
```

8. Mengaktifkan Docker saat Booting
   Secara default, Docker seharusnya berjalan secara otomatis setiap kali sistem dinyalakan. Namun, jika Docker tidak dimulai secara otomatis, Anda dapat mengaktifkannya agar berjalan saat booting berikutnya dengan perintah berikut.

```
    sudo systemctl enable docker
```

9. Menguji Docker
   Jalankan sebuah kontainer uji untuk memastikan bahwa Docker telah berfungsi dengan baik di sistem Anda.

```
    docker run hello-world
```

   Perintah ini akan mengunduh image uji dari repositori Docker Hub dan menjalankannya dalam sebuah kontainer. Jika Docker telah terinstal dan dikonfigurasi dengan benar, Anda akan melihat pesan output yang mengonfirmasi bahwa Docker berfungsi sebagaimana mestinya.

### 8.5.1 Perintah Dasar Docker

Setelah Docker terinstal, berikut beberapa perintah dasar yang perlu dikuasai:

| Perintah | Penjelasan |
|---|---|
| docker –version | Cek versi Docker |
| systemctl status Docker | Untuk memastikan bahwa daemon Docker (Dockerd) berjalan. Output-nya akan menunjukkan apakah layanan Docker active (running) atau tidak. |
| ps aux \| grep dockerd | Untuk melihat apakah proses Docker jalan. Kalau ada proses bernama Dockerd, berarti daemon-nya aktif. |
| docker ps | Melihat daftar container yang sedang berjalan. Output-nya akan menampilkan: CONTAINER ID: ID unik dari container, NAMES: nama container, dan informasi lain seperti image, status, port, dll. |
| docker ps -a | Melihat semua container, baik yang sedang berjalan maupun yang sudah exited/stopped. |
| docker pull nginx | Perintah untuk mengambil (men-download) image dari registry (secara default, Docker Hub). nginx: nama image yang ingin diambil. Ini adalah web server populer dan open-source. |
| docker pull nginx:1.25 | Manarik nginx versi 1.25 |
| docker stop <container_id atau container_name> | menghentikan container |
| docker rm <container_id/nama> | Menghapus Container. Jika image atau container sedang digunakan, Anda harus stop dulu containernya dengan Docker stop <container_id> |
| docker stop $(docker ps -q) | Stop semua container yang aktif |
| docker container prune | Hapus semua container yang berhenti. |
| docker rmi <image_id/nama> | Menghapus images tertentu. |
| docker image prune -a | Hapus semua image yang tidak digunakan. |

Tabel 1: Perintah Dasar Docker

Dalam pengembangan aplikasi modern, workflow Docker yang paling umum dan praktis terdiri dari tiga tahap utama, yaitu: menarik image, menjalankan container, dan memantau container.

### 8.5.2 Menjalankan Docker tanpa Sudo

Secara default, penggunaan Docker memerlukan hak akses root atau perintah sudo. Namun, jika Anda ingin menjalankan Docker tanpa harus menggunakan sudo setiap kali, Anda perlu menambahkan pengguna ke dalam grup Docker. Untuk melakukannya, jalankan perintah berikut:

```
usermod -aG docker $USER
```

$USER pada perintah di atas adalah placeholder yang secara otomatis akan mengambil nama pengguna yang sedang aktif. Jika Anda ingin menetapkan pengguna tertentu, ganti $USER dengan nama pengguna yang sesuai. Perintah **usermod** adalah perintah di Linux untuk memodifikasi informasi akun pengguna.

- Opsi -a (append): menambahkan user ke grup tanpa menghapus grup lainnya.
- Opai -G docker: menambahkan user ke grup secondary bernama Docker.
- $USER: adalah variabel lingkungan yang menunjuk ke nama user yang sedang login

Setelah menjalankan perintah ini, Anda perlu me-restart sistem agar perubahan dapat diterapkan. Setelah sistem dinyalakan kembali, Anda dapat menjalankan Docker tanpa menggunakan sudo.

## 8.6 Best Practice: Nginx-Docker

Nginx adalah salah satu web server yang ringan, cepat, dan andal, yang banyak digunakan untuk melayani situs web statis maupun sebagai reverse proxy untuk aplikasi berbasis web. Web server ini dikenal karena kemampuannya dalam menangani banyak koneksi secara efisien dengan konsumsi sumber daya yang rendah, menjadikannya pilihan utama bagi banyak pengembang dan administrator sistem. Selain sebagai server untuk menyajikan file statis seperti HTML, CSS, dan JavaScript, Nginx juga sering digunakan sebagai load balancer untuk mendistribusikan lalu lintas ke beberapa server backend.

Dengan menggunakan Docker, kita dapat menjalankan Nginx dalam sebuah container tanpa perlu menginstalnya langsung di sistem host. Docker memungkinkan kita untuk mengisolasi aplikasi dalam lingkungan yang terpisah, sehingga instalasi dan konfigurasi menjadi lebih sederhana dan dapat dilakukan dengan cepat. Selain itu, dengan Docker, kita dapat dengan mudah mengelola dan menjalankan berbagai versi Nginx tanpa khawatir tentang konflik dengan aplikasi lain di sistem host.

### 8.6.1 Prasyarat

Sebelum memulai, pastikan Anda telah menginstal Docker di sistem Anda. Docker adalah platform containerisasi yang memungkinkan Anda menjalankan aplikasi dalam lingkungan yang terisolasi dari sistem utama, sehingga mempermudah proses deployment dan manajemen aplikasi. Jika Anda belum menginstalnya, Anda dapat mengikuti langkah-langkah instalasi Docker di Ubuntu Server 24.04 pada bagian sebelumnya.

Pastikan sistem Anda telah diperbarui sebelum melakukan instalasi, dan setelah proses instalasi selesai, verifikasi bahwa Docker telah terinstal dengan baik menggunakan perintah yang sesuai. Dengan Docker yang sudah berjalan, Anda siap untuk menjalankan Nginx dalam container.

```
Docker --version
groups
#atau
id
#atau
getent group Docker
```

Jika pengguna belum menjadi anggota grup Docker, Anda dapat menambahkannya dengan perintah:

```
sudo usermod -aG docker $USER
```

Kemudian, logout dan login kembali atau jalankan:

```
newgrp docker
```

agar perubahan diterapkan tanpa harus restart sistem. Setelah itu, ulangi langkah-langkah di atas untuk memastikan bahwa pengguna telah menjadi anggota grup docker.

Salah satu keunggulan utama Docker adalah kemampuannya untuk menjalankan aplikasi dalam container yang bersifat portabel dan ringan. Dengan Docker, kita dapat menjalankan Nginx tanpa harus melakukan instalasi dan konfigurasi yang rumit. Proses ini memungkinkan kita untuk mengaktifkan web server dengan cepat dan menghindari masalah kompatibilitas pada sistem operasi yang berbeda. Pada bagian ini, kita akan menjalankan Nginx dalam container, melakukan pemetaan port agar dapat diakses dari luar, serta memastikan bahwa container berjalan dengan benar.

### 8.6.2 Menarik (Pull) Image Nginx

Docker Hub adalah repositori daring yang menyediakan berbagai macam image container siap pakai, termasuk image resmi Nginx. Image ini dikembangkan dan dipelihara oleh komunitas serta vendor resmi, sehingga memudahkan pengguna dalam mendapatkan perangkat lunak yang sudah dikemas dalam format container.

Dengan menggunakan Docker Hub, kita dapat mengunduh dan menjalankan aplikasi tanpa perlu melakukan instalasi manual yang kompleks.

```
Docker pull nginx:latest
```

Perintah ini akan mengunduh image Nginx versi terbaru (latest) ke sistem lokal sehingga dapat digunakan untuk menjalankan container. Jika image tersebut sudah tersedia di sistem, Docker akan langsung menggunakannya tanpa perlu mengunduh ulang. Setelah proses ini selesai, kita dapat **langsung menjalankan container** berbasis image tersebut.

Setelah menjalankan Docker pull nginx, kita dapat memastikan bahwa image telah diunduh dengan menjalankan:

```
docker images
#atau
docker inspect nginx:latest
```

### 8.6.3 Menjalankan Kontainer Nginx

Setelah berhasil mengunduh image Nginx, langkah selanjutnya adalah menjalankan container berdasarkan image tersebut. Dalam Docker, image adalah blueprint berisi semua yang dibutuhkan untuk menjalankan aplikasi, seperti kode, dependensi, dan konfigurasi. Container adalah instansi aktif dari image tersebut yang dapat dijalankan. Ibaratnya, image adalah resep, sedangkan container adalah masakan yang dibuat dari resep itu. Satu image bisa digunakan untuk membuat banyak container secara konsisten.

Gunakan perintah berikut untuk menjalankan container Nginx:

```
docker run --name nginx-server -d -p 8080:80 nginx
```

Penjelasan parameter yang digunakan:

- docker run : Perintah untuk menjalankan container baru.
- --name nginx-server : Memberikan nama "nginx-server" pada container agar mudah dikelola.
- -d : Menjalankan container dalam mode detached (di latar belakang).
- -p 8080:80 : Memetakan port 8080 di host ke port 80 di dalam container[2] sehingga server dapat

[2] Jika ingin mengetahui port yang belum digunakan, jalankan: sudo netstat -tulnp | grep LISTEN, sudo netstat -tuln, atau sudo ss -tuln. Kalau ingin tahu aplikasi apa yang menggunakan port tertentu, misalnya port 8080 gunakan perintah:

diakses melalui http://localhost:8080.
- nginx : Nama image yang digunakan untuk menjalankan container.

### 8.6.4 Mengecek Status Container

Dengan perintah ini, Docker akan membuat container baru berdasarkan image Nginx dan menjalankannya di latar belakang. Setelah container berjalan, Nginx siap untuk menerima permintaan HTTP melalui port yang telah dipetakan.

```
docker ps
```

Perintah ini akan menampilkan daftar container yang sedang berjalan, termasuk informasi seperti ID container, nama, status, serta port yang digunakan. Jika **nama container "nginx-server"** muncul dalam daftar dengan status "Up", maka Nginx sudah berjalan dan dapat diakses melalui browser.

### 8.6.5 Mengakses Web Server Nginx

Buka browser dan akses:

```
http://localhost:8080
# Jika dijalankan di server, gunakan alamat IP server:
http://<IP-Server>:8080
```

Anda akan melihat halaman default Nginx yang menampilkan pesan "Welcome to Nginx!" beserta beberapa informasi dasar. Halaman ini menandakan bahwa web server Nginx telah berhasil dijalankan di dalam container dan siap digunakan. Jika halaman ini muncul, berarti konfigurasi dasar sudah berjalan dengan baik, dan Anda dapat mulai menyesuaikan pengaturan sesuai kebutuhan, seperti mengganti halaman utama atau menambahkan konfigurasi tambahan.

### 8.6.6 Menghentikan dan menghapus Kontainer nginx-server

Setelah menjalankan container, penting juga untuk mengetahui bagaimana cara mengelola dan membersihkannya. Untuk menghentikan container yang sedang berjalan, misalnya container dengan nama nginx-server, kita dapat menggunakan perintah Docker stop nginx-server.

```
docker stop nginx-server
```

Perintah ini akan menghentikan proses container tanpa menghapusnya dari sistem. Jika container sudah tidak dibutuhkan lagi, maka dapat dihapus sepenuhnya dengan perintah Docker rm nginx-server. Penghapusan container ini akan membebaskan resource sistem yang digunakan oleh container tersebut.

```
docker rm nginx-server
```

Perintah ini hanya dapat dijalankan jika container dalam keadaan berhenti. Jika container masih berjalan, Anda perlu menghentikannya terlebih dahulu dengan Docker stop nginx-server sebelum menjalankan perintah ini.

### 8.6.7 Menghapus Images nginx

Sementara itu, untuk melihat daftar semua image yang sudah tersimpan secara lokal di mesin Docker, kita dapat menggunakan perintah Docker images. Perintah ini akan menampilkan informasi seperti

sudo lsof -i :8080

nama image, tag, ID, dan ukuran masing-masing image, yang sangat berguna untuk manajemen image yang efisien di lingkungan pengembangan.

```
docker images
```

Jika Anda tidak lagi memerlukan image Nginx yang telah diunduh dari Docker Hub dan ingin menghemat ruang penyimpanan, Anda dapat menghapusnya dengan perintah berikut:

```
docker rmi nginx
```

Namun, sebelum menghapus image, pastikan tidak ada container yang menggunakan image tersebut. Jika masih ada container yang menggunakan image, Anda harus menghapus container terlebih dahulu dengan Docker rm sebelum menjalankan perintah ini. Jika Anda ingin menghapus semua container dan image yang tidak lagi digunakan secara otomatis, Anda bisa menggunakan:

```
docker system prune -a
```

Perintah ini akan menghapus semua container yang sudah tidak aktif serta image yang tidak memiliki container yang menggunakannya, sehingga membantu membebaskan ruang penyimpanan pada sistem.

### 8.6.8 Direktori Kustom untuk Konten Web

Secara default, Nginx menyimpan file HTML dan konten web lainnya di dalam container pada direktori /usr/share/nginx/html. Jika ingin masuk ke shell atau terminal dalam container yang sedang berjalan, gunakan perintah:

```
docker ps
docker exec -it <nama-container> bash
root@9920623cd3d5:/# ls
```

Direktori ini berisi halaman default yang ditampilkan ketika Nginx pertama kali dijalankan. Namun, dalam banyak kasus, kita ingin menggunakan direktori di sistem host agar lebih mudah dalam mengelola dan memperbarui file tanpa harus masuk ke dalam container.

Untuk itu, kita bisa menggantinya dengan direktori di sistem host menggunakan **fitur volume** di Docker. Dengan volume, perubahan yang dilakukan pada file di sistem host akan langsung tercermin dalam container, memungkinkan pengelolaan konten web yang lebih fleksibel dan efisien.

1. Membuat direktori untuk konten web:

```
   mkdir -p ~/nginx-html
```

2. Membuat file index.html di dalamnya:
   Sebelum menjalankan Nginx dengan Docker, kita perlu membuat direktori khusus di sistem host yang akan digunakan untuk menyimpan file HTML dan konten web lainnya. Dengan cara ini, kita dapat dengan mudah mengelola dan memperbarui halaman web tanpa harus masuk ke dalam container. Gunakan perintah berikut di terminal untuk membuat direktori bernama nginx-html di dalam home directory (~/):

```
   echo '<h1>Halo, ini adalah server Nginx dengan Docker!</h1>'  \
   ~/nginx-html/index.html
```

   Opsi -p pada perintah mkdir memastikan bahwa jika direktori belum ada, maka akan dibuat tanpa menampilkan pesan error. Direktori ini akan digunakan sebagai volume yang dipasang ke dalam

container Nginx sehingga kontennya dapat langsung diakses oleh web server.

Perintah echo digunakan untuk menulis teks HTML ke dalam file index.html. Jika Anda ingin mengedit file lebih lanjut, gunakan editor teks seperti nano

```
nano ~/nginx-html/index.html
```

Setelah selesai mengedit, tekan CTRL + X, lalu tekan Y dan Enter untuk menyimpan perubahan.

### 8.6.9 Menjalankan Nginx dengan Volume

Agar Nginx dapat menggunakan file HTML yang disimpan di sistem host, kita perlu menjalankan container dengan opsi volume. Volume memungkinkan kita untuk menyimpan dan mengelola file di luar container, sehingga perubahan yang dilakukan pada file HTML di sistem host akan langsung tercermin di dalam container tanpa perlu membangun ulang image atau menyalin file secara manual.

Gunakan perintah berikut di terminal untuk menjalankan container Nginx dengan volume yang terhubung ke direktori ~/nginx-html yang telah kita buat sebelumnya:

```
docker ps -a --filter name=nginx-server
docker stop nginx-server
docker rm nginx-server
docker run -d --name nginx-server -p 8080:80 -v  \
       ~/nginx-html:/usr/share/nginx/html nginx
```

Penjelasan parameter pada perintah ini:

- docker run ⇒ Perintah untuk menjalankan container.
- -d ⇒ Menjalankan container dalam mode detached, sehingga berjalan di latar belakang.
- --name nginx-server ⇒ Memberikan nama nginx-server untuk container agar lebih mudah dikelola.
- -p 8080:80 ⇒ Memetakan port 8080 di host ke port 80 di dalam container, sehingga Nginx dapat diakses melalui http://localhost:8080.
- -v ~/nginx-html:/usr/share/nginx/html Menggunakan volume untuk menghubungkan direktori ~/nginx-html di sistem host ke dalam container di lokasi /usr/share/nginx/html, yang merupakan direktori default tempat Nginx mencari file HTML.
- nginx ⇒ Menentukan image yang digunakan, dalam hal ini adalah Nginx terbaru dari Docker Hub.

Setelah menjalankan perintah ini, container akan berjalan dengan direktori yang telah kita buat sebelumnya sebagai penyimpanan kontennya. Anda dapat mengakses Nginx dengan membuka browser dan mengetikkan http://localhost:8080, di mana halaman index.html yang telah dibuat akan ditampilkan.

## 8.7 Dockerfile

Dockerfile adalah sebuah file teks yang berisi serangkaian instruksi yang dituliskan secara berurutan untuk memberi tahu Docker *bagaimana cara membangun sebuah Docker image*. File ini menjadi blueprint atau cetak biru dari image yang akan dibuat. Dengan kata lain, Dockerfile adalah resep lengkap untuk membungkus sebuah aplikasi ke dalam container, lengkap dengan seluruh dependensinya, konfigurasi yang diperlukan, serta perintah-perintah yang harus dijalankan saat container dibuat atau dijalankan.

Penggunaan Dockerfile memungkinkan proses pembuatan image menjadi otomatis, berulang, dan konsisten, tanpa perlu melakukan setup manual yang rawan kesalahan. Misalnya, jika sebuah aplikasi

membutuhkan pustaka tertentu, konfigurasi jaringan khusus, atau file konfigurasi yang spesifik, semuanya bisa diatur hanya satu kali dalam Dockerfile. Setelah itu, image bisa dibangun kapan pun dengan hasil yang identik.

Dockerfile sangat penting, terutama dalam konteks DevOps, CI/CD (Continuous Integration/Continuous Deployment), dan pengembangan perangkat lunak modern berbasis container. Dalam proyek rekayasa perangkat lunak, Dockerfile membantu tim developer membagikan lingkungan pengembangan yang sama, tanpa perlu mengatur ulang dependensi di setiap komputer. Sementara itu, dalam bidang Teknologi Rekayasa Komputer, Dockerfile dapat digunakan untuk menyusun sistem yang kompleks, seperti server simulasi, sistem terdistribusi, maupun container khusus untuk aplikasi embedded dan IoT.

Dengan menggunakan Dockerfile, kita dapat memastikan bahwa aplikasi dapat dijalankan secara portabel di berbagai platform, dari laptop pengembang, server lokal, hingga layanan cloud — semua dengan konfigurasi yang seragam dan dapat dikontrol penuh.

### 8.7.1 Struktur Dasar Dockerfile

Sebuah Dockerfile dibangun dengan kumpulan instruksi yang dituliskan secara berurutan, dari atas ke bawah. Setiap instruksi mewakili satu langkah dalam proses membangun image. Urutan penulisan sangat penting, karena Docker akan menjalankan instruksi-instruksi ini secara berurutan dan menyimpannya sebagai layer (lapisan) dalam image yang dihasilkan. Berikut adalah beberapa instruksi umum yang paling sering digunakan dalam Dockerfile, beserta fungsinya:

Dengan memahami struktur dasar ini, kita bisa mulai membangun Dockerfile sesuai kebutuhan proyek. Instruksi-instruksi tersebut dapat dikombinasikan untuk membuat lingkungan yang lengkap dan siap digunakan dalam pengembangan, pengujian, maupun produksi.

Dalam praktiknya, penggunaan Dockerfile juga dapat diintegrasikan dengan sistem manajemen versi (seperti Git), pipeline CI/CD, hingga container registry (seperti Docker Hub atau GitHub Container Registry), menjadikannya alat yang sangat esensial dalam workflow modern pengembangan perangkat lunak maupun rekayasa sistem komputer.

### 8.7.2 Ilustrasi Struktur Dockerfile Sederhana

Agar lebih mudah memahami cara kerja Dockerfile, berikut ini adalah sebuah contoh struktur Dockerfile sederhana yang digunakan untuk membuat image aplikasi web berbasis Python menggunakan framework Flask. Contoh ini mencerminkan praktik yang umum dilakukan dalam pengembangan perangkat lunak modern, terutama dalam konteks containerization dan otomatisasi deployment.

Dockerfile merupakan jantung dari proses pembuatan image, karena berisi kumpulan instruksi yang disusun secara berurutan, dimulai dari pemilihan base image hingga perintah yang dijalankan saat container aktif. Dalam contoh ini, kita akan membuat aplikasi Flask yang berjalan di dalam container secara mandiri, dengan lingkungan yang konsisten dan terisolasi.

Untuk membuat Dockerfile, Anda cukup menggunakan teks editor favoritmu (seperti nano, vim, atau editor GUI seperti VS Code) dan menyimpannya dengan nama Dockerfile (tanpa ekstensi apa pun).

1. Masuk ke direktori project dengan mengetikkan:

```
cd my-flask-app
```

2. Membuat file dengan nama Dockerfile (tanpa ekstensi apa pun) dengan editor teks, misalnya nano:

| Perintah | Penjelasan |
|---|---|
| FROM | Menentukan base image yang akan digunakan sebagai fondasi. Contohnya: FROM ubuntu:24.04 atau FROM python:3.10. Ini adalah langkah pertama yang wajib ada dalam setiap Dockerfile. |
| RUN | Menjalankan perintah dalam image saat proses build. Umumnya digunakan untuk instalasi dependensi atau update sistem. Contoh: RUN apt update && apt install -y nginx. |
| COPY | Menyalin file atau direktori dari komputer lokal (host) ke dalam image. Contoh: COPY . /app akan menyalin seluruh isi folder ke direktori /app di dalam container. |
| ADD | Mirip seperti COPY, tapi dengan kemampuan tambahan seperti mendownload dari URL atau mengekstrak file arsip .tar. |
| WORKDIR | Menentukan direktori kerja di dalam container. Semua perintah setelah WORKDIR akan dieksekusi dalam direktori tersebut. Contoh: WORKDIR /app. |
| CMD | Menentukan perintah default yang dijalankan saat container dijalankan. Contoh: CMD ["python", "app.py"]. |
| ENTRYPOINT | Mirip seperti CMD, tetapi lebih "kaku" perintah ini akan selalu dijalankan dan tidak bisa ditimpa saat menjalankan container (kecuali dengan opsi –entrypoint). |
| ENV | Mendefinisikan variabel lingkungan (environment variable) di dalam image. Contoh: ENV PORT=8080. |
| EXPOSE | Memberitahu Docker bahwa container akan mendengarkan pada port tertentu, misalnya: EXPOSE 5000. |
| VOLUME | Mendeklarasikan direktori sebagai volume yang bisa di-mount dari luar container. Cocok untuk data yang harus disimpan secara persisten. |

Tabel 2: Instruksi Umum Dockerfile

```
nano Dockerfile
```

3. Ketikkan isi berikut ke dalam file Dockerfile:

```
# Menggunakan image dasar resmi dari Python versi 3.10
FROM python:3.10
# Menentukan direktori kerja di dalam container
WORKDIR /app
# Menyalin seluruh file dari direktori proyek lokal
# ke direktori kerja di container
COPY . /app
# Menginstal dependensi yang terdaftar di file requirements.txt
RUN pip install --no-cache-dir -r requirements.txt
# Menentukan port yang akan diekspos oleh container
```

```
EXPOSE 5000
# Menentukan perintah yang akan dijalankan saat container aktif
CMD ["python", "app.py"]
```

Dalam contoh di atas, Dockerfile akan membangun image yang siap menjalankan aplikasi Flask secara otomatis. Semua proses seperti instalasi, konfigurasi, dan penyalinan file telah didefinisikan di dalam Dockerfile, sehingga siapa pun yang membangun image ini akan mendapatkan lingkungan yang identik.

Penjelasan setiap baris:

- FROM python:3.10, Menentukan base image yang akan digunakan. Dalam hal ini, kita memilih Python versi 3.10 yang sudah dikonfigurasi oleh komunitas Docker.
- WORKDIR /app, Mengatur direktori kerja di dalam container menjadi /app. Semua perintah berikutnya dijalankan dari direktori ini.
- COPY . /app, Menyalin seluruh file dari folder proyek lokal ke folder /app dalam container.
- RUN pip install –no-cache-dir -r requirements.txt, Menginstal semua dependensi Python yang dibutuhkan aplikasi, berdasarkan file requirements.txt.
- EXPOSE 5000, Mendeklarasikan bahwa aplikasi akan menggunakan port 5000port default untuk aplikasi Flask.
- CMD ["python", "app.py"], Mendefinisikan perintah yang akan dieksekusi ketika container dijalankan, yaitu menjalankan aplikasi utama app.py.

4. Simpan file:
   Jika menggunakan nano, tekan Ctrl + O, lalu Enter untuk menyimpan. Tekan Ctrl + X untuk keluar dari nano.

### 8.8 Docker Compose

Docker Compose adalah sebuah alat bantu (tool) dari Docker yang memungkinkan kita untuk secara otomatis membuat dan menjalankan aplikasi multi-container hanya dengan satu perintah. Dalam dunia nyata, banyak aplikasi modern tidak hanya terdiri dari satu komponen (misalnya: backend saja), melainkan terdiri dari beberapa layanan (service) yang berjalan secara bersamaan, seperti:

1. Database (contoh: MySQL, MariaDB)
2. Cache & message broker (contoh: Redis, RabbitMQ)
3. Backend Application (contoh: ERPNext)
4. Frontend atau Web Server (contoh: Nginx)
5. Worker/background jobs

Tanpa Docker Compose, kita perlu menjalankan masing-masing layanan tersebut secara manual dengan perintah Docker run, yang bisa menjadi kompleks dan membosankan jika jumlah layanan banyak. Dengan Docker Compose, kita cukup menuliskan semua konfigurasi layanan dalam satu file: Docker-compose.yml. File ini menggunakan format YAML untuk mendefinisikan:

1. Gambar container (Docker image) yang akan digunakan
2. Variabel lingkungan (environment variables)
3. Port yang perlu diekspose
4. Volume untuk penyimpanan data
5. Ketergantungan antar-layanan (dependency)
6. Jaringan yang digunakan

Setelah konfigurasi selesai, semua layanan bisa langsung dijalankan sekaligus hanya dengan:

```
1 docker-compose up
```

## 8.9 Studi Kasus 1: Flask + Docker Compose

> Di era pengembangan perangkat lunak modern, pemanfaatan container menjadi sangat penting untuk mendukung proses deployment yang cepat, konsisten, dan portabel. Salah satu alat yang paling banyak digunakan dalam praktik DevOps adalah Docker, sebuah platform yang memungkinkan aplikasi dan seluruh dependensinya dikemas ke dalam sebuah kontainer ringan dan terisolasi.

Untuk membantu memahami konsep Docker dan Docker Compose secara langsung, disajikan sebuah studi kasus pembangunan aplikasi sederhana berbasis Flask, yaitu salah satu micro-framework populer pada bahasa pemrograman Python. Aplikasi ini kemudian akan dilakukan containerisasi menggunakan Docker, serta dikelola layanannya melalui Docker Compose.

Melalui contoh ini, diharapkan Anda dapat memahami peran Docker dalam proses pengembangan dan deployment aplikasi modern, serta mampu menerapkannya dalam berbagai proyek, baik di lingkungan kampus maupun dunia industri.

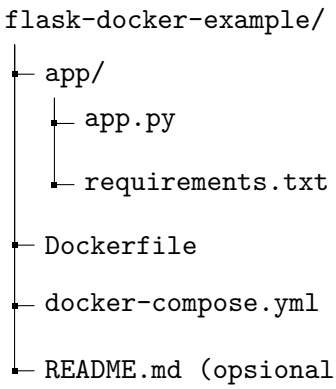
### 8.9.1 Tujuan

Tujuan dari studi kasus ini adalah untuk memberikan pemahaman praktis mengenai konsep dasar containerization dan orkestrasi layanan menggunakan Docker dan Docker Compose, melalui penerapan langsung pada aplikasi sederhana berbasis Flask. Secara lebih spesifik, studi kasus ini dirancang untuk:

1. Menunjukkan bagaimana sebuah aplikasi web dapat dibungkus (dikemas) ke dalam sebuah container Docker lengkap dengan seluruh dependensinya.
2. Memperkenalkan penggunaan Docker Compose untuk mengelola dan mengorkestrasi layanan agar lebih terstruktur dan mudah dijalankan.
3. Memberikan pengalaman belajar yang mudah dipahami oleh mahasiswa, terutama bagi mereka yang baru memulai pembelajaran di bidang DevOps, container-based deployment.

### 8.9.2 Struktur Folder

Sebelum mulai membangun dan menjalankan aplikasi Flask menggunakan Docker, penting untuk memahami struktur folder proyek secara keseluruhan. Struktur yang terorganisir dengan baik akan memudahkan proses pembangunan image, pengelolaan dependensi, serta orkestrasi layanan menggunakan Docker Compose. Pada studi kasus ini, kita akan menggunakan struktur folder sederhana namun representatif, yang mencerminkan praktik umum dalam pengembangan aplikasi berbasis container. Penjelasan berikut akan membantu Anda memahami peran masing-masing file dan folder dalam proyek ini.




File app/app.py merupakan inti dari aplikasi Flask yang akan dijalankan. Di dalam file ini, kita mendefinisikan sebuah aplikasi web sederhana menggunakan Flask, dengan satu rute utama (/) yang akan menampilkan pesan sebagai output ketika diakses melalui browser. File ini menjadi titik awal untuk memahami bagaimana aplikasi Python dapat dijalankan di dalam container Docker.

Skrip app.py inilah yang akan dibungkus ke dalam container menggunakan Docker, dan nantinya dijalankan sebagai layanan utama dalam proyek ini. Oleh karena itu, pemahaman terhadap struktur dan isi dari skrip app.py sangat penting sebelum melangkah ke proses containerization. Berikut adalah isi dari skrip app.py yang digunakan dalam studi kasus ini:

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def hello():
    return "Hello, Docker Compose!"

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

Untuk memastikan aplikasi dapat berjalan dengan lancar di dalam container, kita perlu mendefinisikan semua dependensi Python yang dibutuhkan. Dependensi ini dituliskan dalam file app/requirements.txt, yang akan digunakan oleh Docker saat proses pembangunan image.

Dalam kasus ini, kita hanya memerlukan Flask sebagai framework utama untuk menjalankan aplikasi web. Oleh karena itu, isi dari file requirements.txt cukup sederhana, yaitu hanya menyertakan baris flask.

Selain itu, pada tahap selanjutnya, kita juga akan menambahkan Nginx sebagai reverse proxy untuk mengarahkan lalu lintas HTTP ke layanan Flask. Meskipun Nginx tidak didefinisikan di dalam requirements.txt karena bukan bagian dari ekosistem Python, konfigurasinya akan disiapkan secara terpisah dan dikelola melalui Docker Compose sebagai layanan mandiri.

Berikut adalah isi dari file app/requirements.txt, yang berfungsi untuk mendeklarasikan dependensi Python yang diperlukan oleh aplikasi, dalam hal ini hanya framework Flask:

```
flask
```

Untuk mengatur dan menjalankan beberapa layanan sekaligus dalam satu lingkungan terisolasi, kita

menggunakan file Docker-compose.yml. File ini memungkinkan kita untuk mendefinisikan layanan aplikasi Flask serta layanan pendukung lainnya, seperti Nginx, dalam satu konfigurasi yang mudah dikelola.

Berikut adalah isi dari file Docker-compose.yml yang digunakan dalam studi kasus ini:

```
version: '3.8'

services:
  web:
    build: ./app
    ports:
      - "5000:5000"
    volumes:
      - ./app:/app
    networks:
      - flask-net

  nginx:
    image: nginx:alpine
    ports:
      - "80:80"
    volumes:
      - ./nginx/default.conf:/etc/nginx/conf.d/default.conf
    depends_on:
      - web
    networks:
      - flask-net

networks:
  flask-net:
    driver: bridge
```

1. version: '3.8'. Baris ini menunjukkan versi schema Docker Compose yang digunakan. Versi 3.8 umum digunakan dan mendukung berbagai fitur terbaru, termasuk pengelolaan jaringan dan dependency layanan.
2. Bagian services. Semua container yang akan dijalankan didefinisikan dalam bagian services. Dalam kasus ini, terdapat dua layanan: web dan nginx.
3. Service: web
   - build: ./app. Menunjukkan bahwa Docker akan membangun image dari folder ./app, yang seharusnya berisi Dockerfile dan kode aplikasi Flask.
   - ports:
     - "5000:5000". Mengekspose port 5000 dari container ke port 5000 di host, sehingga aplikasi Flask bisa diakses melalui localhost:5000.
   - volumes: - ./app:/app. Menyambungkan folder ./app di host ke /app di dalam container. Ini berguna saat ingin melakukan perubahan kode secara langsung (live reload).
   - networks: - flask-net. Menghubungkan service web ke jaringan internal flask-net agar bisa berkomunikasi dengan service lain seperti nginx.

Setelah seluruh struktur proyek selesai disiapkan, langkah berikutnya adalah menjalankan aplikasi meng-

gunakan Docker Compose. Proses ini akan membangun image dari service yang didefinisikan dan menjalankannya dalam container sesuai dengan konfigurasi yang telah dibuat. Pertama, pastikan Anda berada di direktori utama proyek (misalnya bernama flask-Docker-example). Kemudian jalankan perintah berikut di terminal:

```
# Masuk ke folder flask-Docker-example
cd flask-docker-example

# Build dan jalankan service dengan docker compose
docker-compose up --build
```

Perintah docker-compose up –build akan:

- Membangun ulang image berdasarkan perubahan terbaru di Dockerfile.
- Menjalankan semua service (web dan nginx) sesuai definisi dalam Docker-compose.yml.
- Menampilkan log dari kedua service secara real-time di terminal.

Setelah proses ini selesai, aplikasi Flask dapat diakses melalui browser di alamat http://localhost:port, karena lalu lintas HTTP akan diarahkan oleh Nginx ke service Flask yang berjalan di dalam container.

```
http://localhost:5000
```

### 8.9.3 Pengembangan Lanjutan

Setelah berhasil menjalankan aplikasi Flask dasar menggunakan Docker dan Docker Compose, terdapat beberapa pengembangan yang bisa dilakukan untuk memperkaya studi kasus ini dan lebih mendekatkannya dengan praktik DevOps yang umum di dunia industri maupun kampus.

**8.9.3.1 Menambahkan Database (PostgreSQL atau MySQL)** Dalam pengembangan aplikasi nyata, kebutuhan akan penyimpanan data menjadi sangat penting. Anda dapat menambahkan layanan database seperti PostgreSQL atau MySQL ke dalam file Docker-compose.yml. Contoh service untuk PostgreSQL:

```
db:
  image: postgres:alpine
  environment:
    POSTGRES_DB: flaskdb
    POSTGRES_USER: user
    POSTGRES_PASSWORD: pass
  volumes:
    - pgdata:/var/lib/postgresql/data
  networks:
    - flask-net
```

Kemudian di aplikasi Flask (app.py), Anda dapat menggunakan SQLAlchemy untuk menghubungkan ke database dengan environment variable seperti:

```
DATABASE_URL = "postgresql://user:pass@db:5432/flaskdb"
```

**8.9.3.2 Mengaktifkan Auto-Reload Flask** Selama tahap pengembangan, sangat membantu jika perubahan kode bisa langsung dijalankan ulang tanpa harus membangun ulang container. Untuk itu,

Anda bisa menggunakan flask run –reload dan memastikan FLASK_ENV=development diatur di environment Dockerfile atau .env. Contoh penyesuaian Dockerfile:

```
ENV FLASK_ENV=development
CMD ["flask", "run", "--host=0.0.0.0", "--reload"]
```

Hal ini akan membuat aplikasi Flask otomatis memuat ulang ketika ada perubahan file sumber, sangat memudahkan saat coding.

#### 8.9.3.3 Deployment ke DockerHub atau Server Kampus

Setelah aplikasi stabil, Anda bisa mendistribusikannya melalui DockerHub atau langsung dideploy ke server kampus.

1. Push ke DockerHub:
   - Build image:

```
1  docker build -t username/flask-app .
```

   - Login dan push:

```
docker login
docker push username/flask-app
```

2. Deploy ke Server:
   - Di sisi server

```
docker pull username/flask-app
docker run -d -p 80:5000 username/flask-app
```

Dengan pendekatan ini, aplikasi Anda menjadi portabel dan mudah dijalankan di mana pun, baik di lokal, server kampus, cloud, maupun edge device.

### 8.10 Studi Kasus 2: Instalasi Laravel dengan Docker & Docker Compose

Studi kasus ini bertujuan untuk membangun sebuah environment Laravel yang berjalan secara terisolasi menggunakan Docker dan Docker Compose. Dalam implementasinya, environment ini akan terdiri dari beberapa komponen utama, yaitu Nginx sebagai web server, PHP yang telah dilengkapi dengan ekstensi-ekstensi yang dibutuhkan oleh Laravel, serta MySQL sebagai sistem manajemen basis data.

Laravel sendiri akan bertindak sebagai aplikasi utama yang dikembangkan. Selain itu, untuk memudahkan pengelolaan database secara visual, phpMyAdmin juga dapat ditambahkan sebagai komponen opsional.

Sebelum melanjutkan ke langkah instalasi dan konfigurasi lebih lanjut, pastikan bahwa Docker dan Docker Compose telah terinstal dengan benar pada sistem. Pemeriksaan ini dapat dilakukan dengan menjalankan perintah Docker –version dan Docker-compose –version pada terminal. Jika kedua perintah tersebut mengembalikan versi yang valid, maka lingkungan siap digunakan untuk membangun project Laravel berbasis Docker.

{.bash numberLine} docker --version docker-compose --version

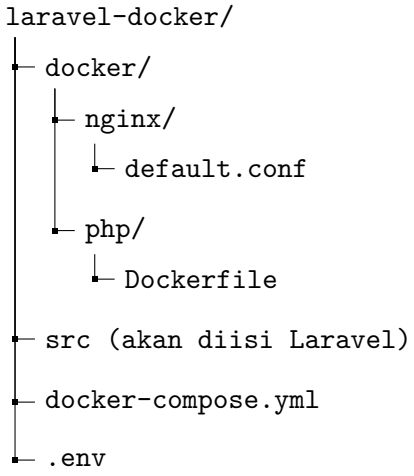
#### 8.10.1 Struktur Fodler Project

Sebelum memulai proses instalasi dan konfigurasi Laravel menggunakan Docker, penting untuk memahami struktur folder proyek yang akan digunakan. Struktur ini dirancang agar setiap komponen

memiliki peran dan tempat yang jelas, sehingga memudahkan dalam pengelolaan, pengembangan, dan pemeliharaan proyek.

Folder utama akan memuat direktori khusus untuk konfigurasi Docker, seperti pengaturan Nginx dan PHP, serta folder src yang berfungsi sebagai lokasi utama source code Laravel. Selain itu, file Docker-compose.yml dan .env akan berada di root direktori sebagai pusat konfigurasi dan pengaturan environment.

Berikut adalah struktur folder proyek yang digunakan dalam studi kasus instalasi Laravel menggunakan Docker dan Docker Compose. Struktur ini dirancang agar setiap komponen dapat terorganisasi dengan baik dan mudah untuk dikembangkan maupun dipelihara:




Struktur ini memungkinkan pemisahan antara konfigurasi sistem dan kode aplikasi, sehingga pengelolaan proyek menjadi lebih modular dan profesional. Selain itu, dengan pendekatan ini, seluruh environment dapat dibangun dan dijalankan secara otomatis menggunakan Docker Compose, tanpa perlu konfigurasi manual yang kompleks.

### 8.10.2 File docker-compose.yml

File Docker-compose.yml merupakan inti dari konfigurasi Docker Compose yang digunakan untuk mendefinisikan dan menjalankan seluruh layanan yang dibutuhkan dalam proyek Laravel ini. Di dalam file ini, terdapat beberapa service utama yang didefinisikan, antara lain: app, webserver, db, dan phpmyadmin.

```
1  version: '3.8'
2
3  services:
4    app:
5      build:
6        context: ./docker/php
7      container_name: laravel-app
8      restart: unless-stopped
9      working_dir: /var/www
10     volumes:
11       - ./src:/var/www
12     networks:
13       - laravel
```

```
14
15    webserver:
16      image: nginx:alpine
17      container_name: nginx-server
18      restart: unless-stopped
19      ports:
20        - "8000:80"
21      volumes:
22        - ./src:/var/www
23        - ./docker/nginx/default.conf:/etc/nginx/conf.d/default.conf
24      depends_on:
25        - app
26      networks:
27        - laravel
28
29    db:
30      image: mysql:5.7
31      container_name: mysql-db
32      restart: unless-stopped
33      environment:
34        MYSQL_ROOT_PASSWORD: root
35        MYSQL_DATABASE: laravel
36        MYSQL_USER: laravel
37        MYSQL_PASSWORD: secret
38      ports:
39        - "3307:3306"
40      volumes:
41        - dbdata:/var/lib/mysql
42      networks:
43        - laravel
44
45    phpmyadmin:
46      image: phpmyadmin/phpmyadmin
47      restart: always
48      ports:
49        - 8081:80
50      environment:
51        PMA_HOST: db
52      depends_on:
53        - db
54      networks:
55        - laravel
56
57  volumes:
58    dbdata:
59
60  networks:
```

```
61   laravel:
```

Service app digunakan untuk menjalankan aplikasi Laravel dengan image PHP yang dikustomisasi, lengkap dengan ekstensi-ekstensi yang dibutuhkan oleh Laravel. Service ini juga memetakan folder src sebagai working directory ke dalam container, sehingga seluruh source code Laravel dapat diakses dan dijalankan dengan baik.

Selanjutnya, service webserver menggunakan image Nginx untuk melayani permintaan HTTP. File konfigurasi Nginx (default.conf) yang disediakan secara khusus akan mengatur routing permintaan ke service Laravel. Port 8000 di-host machine dipetakan ke port 80 di dalam container agar aplikasi dapat diakses melalui localhost:8000.

Service db menggunakan image MySQL versi 5.7 sebagai basis data aplikasi. Di dalamnya didefinisikan variabel environment seperti nama database, username, dan password. Service ini juga dilengkapi volume agar data tersimpan secara persisten.

Sebagai tambahan, service phpmyadmin bersifat opsional dan digunakan untuk memberikan antarmuka visual dalam mengelola database MySQL. Port 8081 di-host machine akan diarahkan ke service ini, sehingga phpMyAdmin dapat diakses melalui localhost:8081.

Seluruh service tersebut terhubung dalam satu jaringan internal yang didefinisikan dengan nama laravel, sehingga mereka dapat saling berkomunikasi secara efisien dalam satu lingkungan terisolasi.

### 8.10.3 File docker/php/Dockerfile

File Dockerfile yang berada dalam direktori docker/php/ digunakan untuk membangun custom image PHP yang akan menjalankan aplikasi Laravel. File ini berfungsi sebagai instruksi otomatis untuk membentuk environment PHP sesuai dengan kebutuhan Laravel, khususnya dalam hal ekstensi dan konfigurasi tambahan.

```
1  FROM php:8.1-fpm
2
3  # Install dependencies
4  RUN apt-get update && apt-get install -y \
5  libjpeg-dev \
6  libpng-dev \
7  libwebp-dev \
8  libfreetype6-dev \
9  zip \
10 unzip \
11 git \
12 curl \
13 libonig-dev \
14 libxml2-dev \
15 libzip-dev \
16 libmcrypt-dev \
17 && docker-php-ext-configure gd --with-freetype --with-jpeg --with-webp \
18 && docker-php-ext-install pdo_mysql mbstring exif pcntl bcmath gd zip
19
20 # Install Composer
```

```
COPY --from=composer:latest /usr/bin/composer /usr/bin/composer

WORKDIR /var/www
```

Dalam praktiknya, Dockerfile ini biasanya dimulai dari image dasar resmi PHP, misalnya php:8.1-fpm, yang telah dioptimalkan untuk bekerja dengan FastCGI Process Manager (FPM). Selanjutnya, ditambahkan berbagai ekstensi PHP yang umum dibutuhkan oleh Laravel, seperti pdo, mbstring, openssl, bcmath, tokenizer, xml, dan curl. Instalasi ekstensi dilakukan melalui tool seperti docker-php-ext-install agar prosesnya terintegrasi langsung dengan environment container.

Selain instalasi ekstensi, Dockerfile juga dapat menyertakan konfigurasi tambahan seperti pengaturan timezone, instalasi Composer (dependency manager Laravel), serta penyesuaian permission atau folder kerja (WORKDIR) yang diarahkan ke direktori aplikasi Laravel, yaitu /var/www.

Dengan penggunaan Dockerfile ini, setiap kali container app dibangun, image PHP yang digunakan akan selalu konsisten dengan dependensi dan konfigurasi yang telah ditentukan. Hal ini memudahkan pengembangan tim, menghindari perbedaan environment antar mesin, dan menjamin stabilitas aplikasi Laravel dalam jangka panjang.

### 8.10.4 File: docker/nginx/default.conf

File default.conf yang berada di dalam direktori docker/nginx/ merupakan file konfigurasi virtual host untuk web server Nginx yang akan digunakan dalam menjalankan aplikasi Laravel. File ini berfungsi untuk mengatur bagaimana Nginx menangani permintaan HTTP yang masuk dan mengarahkan permintaan tersebut ke service aplikasi Laravel yang dijalankan pada container PHP.

```
server {
    listen 80;
    index index.php index.html;
    server_name localhost;

    root /var/www/public;

    location / {
        try_files $uri $uri/ /index.php?$query_string;
    }

    location ~ \.php$ {
        fastcgi_pass app:9000;
        fastcgi_index index.php;
        fastcgi_param SCRIPT_FILENAME $realpath_root$fastcgi_script_name;
        include fastcgi_params;
    }

    location ~ /\.ht {
        deny all;
    }
}
```

Dalam konfigurasi ini, Nginx disetting untuk mendengarkan pada port 80 dan melayani domain default

(server_name _;). Root direktori diarahkan ke /var/www/public, yang merupakan folder public milik Laravel dan menjadi entry point utama aplikasi. Selain itu, konfigurasi ini juga mencakup pengaturan index.php sebagai file utama yang dieksekusi oleh Nginx.

Bagian penting lainnya adalah konfigurasi untuk menangani permintaan file statis seperti gambar, CSS, dan JavaScript secara langsung, serta pengaturan fallback untuk semua permintaan lain agar diarahkan ke index.php—yang merupakan karakteristik utama dari aplikasi berbasis Laravel (routing berbasis controller). Nginx juga dikonfigurasi untuk meneruskan permintaan ke PHP-FPM melalui socket atau IP internal container, sehingga dapat mengeksekusi skrip PHP secara efisien.

Secara keseluruhan, file default.conf memastikan integrasi yang optimal antara Nginx dan Laravel dalam konteks container Docker, dengan konfigurasi yang sesuai untuk kebutuhan produksi maupun pengembangan.

### 8.10.5 Langkah Instal Laravel

Setelah seluruh struktur folder dan konfigurasi Docker Compose serta Nginx disiapkan, langkah selanjutnya adalah melakukan instalasi Laravel ke dalam direktori src/ yang telah disediakan. Proses instalasi Laravel dilakukan di dalam container PHP, agar seluruh dependensi dan perintah dijalankan dalam environment yang sesuai dengan kebutuhan framework Laravel.

Pastikan apakah docker sudah running dengan mengetikkan perintah:

```
sudo systemctl status docker
```

Langkah berikutnya memastikan semua service Docker telah berjalan dengan perintah docker-compose up -d. Setelah itu, masuk ke dalam container app menggunakan perintah Docker-compose exec app bash. Di dalam container ini, kita dapat menjalankan perintah instalasi Laravel menggunakan Composer, seperti composer create-project laravel/laravel, yang akan mengunduh dan menyiapkan seluruh file Laravel di direktori kerja container, yang terhubung langsung ke folder src/ di host machine.

```
#build and start container
docker-compose up -d --build

#masuk ke container app:
docker exec -it laravel-app bash
#instalasi Laravel di working direktori
composer create-project laravel/laravel .
exit
```

Setelah proses instalasi selesai, aplikasi Laravel akan siap dijalankan. Untuk memastikan aplikasi dapat diakses melalui browser, pastikan file konfigurasi .env telah disesuaikan dengan pengaturan koneksi database yang didefinisikan dalam file docker-compose.yml.

```
cd src/
sudo nano .env
# set .env Laravel untuk koneksi database
DB_CONNECTION=mysql
DB_HOST=db
DB_PORT=3306
DB_DATABASE=laravel
```

```
DB_USERNAME=laravel
DB_PASSWORD=secret
```

Setelah proses instalasi Laravel selesai dan koneksi ke database telah dikonfigurasi dengan benar melalui file .env, langkah berikutnya adalah menjalankan migrasi database. Proses ini bertujuan untuk membuat struktur tabel awal yang dibutuhkan oleh Laravel secara otomatis berdasarkan file migrasi yang tersedia dalam folder database/migrations.

Setelah mengubah .env, jalankan ulang kontainernya atau jalankan perintah migrate:

```
docker-compose down
docker-compose up -d
docker exec -it laravel-app php artisan migrate
```

Aplikasi Laravel dapat diakses melalui alamat http://localhost:8000, yang merupakan port yang telah dipetakan dari service webserver (Nginx) ke host. Akses ini memungkinkan pengembang untuk melihat tampilan awal Laravel serta mulai mengembangkan fitur-fitur selanjutnya.

Selain itu, jika service phpmyadmin diaktifkan dalam konfigurasi Docker Compose, antarmuka phpMyAdmin dapat diakses melalui http://localhost:8081. Melalui tampilan ini, pengguna dapat melakukan manajemen basis data MySQL secara grafis, seperti membuat tabel, menjalankan query SQL, dan memeriksa struktur data, tanpa perlu menggunakan command line.

```
http://localhost:8000
http://localhost:8081
```

## 8.11 Proyek Mini Docker

### 8.11.1 Deskripsi Umum

Dalam tugas ini, mahasiswa diminta untuk menerapkan konsep containerization menggunakan Docker dan Docker Compose melalui dua studi kasus berbeda sesuai dengan program studi masing-masing:

1. Mahasiswa TPL (Teknologi Rekayasa Perangkat Lunak) akan membangun aplikasi web sederhana berbasis Flask.
2. Mahasiswa TRK (Teknologi Rekayasa Komputer) akan melakukan instalasi dan pengelolaan platform ThingsBoard, sebuah platform IoT berbasis web yang umum digunakan dalam sistem monitoring dan automasi.

### 8.11.2 Tujuan Pembelajaran

1. Mahasiswa memahami cara membangun dan menjalankan container menggunakan Docker.
2. Mahasiswa memahami orkestrasi layanan multi-container menggunakan Docker Compose.
3. Mahasiswa mampu membuat image, menjalankan container, serta menjelaskan arsitektur deployment aplikasi berbasis container.

#### 8.11.2.1 Tugas Mahasiswa TPL – Docker + Flask

1. Deskripsi Tugas:
   Bangun sebuah aplikasi web sederhana menggunakan Flask, lalu lakukan containerisasi dengan Docker dan orkestrasi dengan Docker Compose.

2. Kriteria Teknis:
   - Buat aplikasi Flask minimal menampilkan halaman "Hello, Docker!".
   - Struktur folder harus mengikuti praktik yang rapi (app, requirements.txt, Dockerfile, Docker-compose.yml).
   - Gunakan Docker-compose up untuk menjalankan aplikasi.
   - Gunakan volume agar kode bisa direload otomatis saat diubah.
   - Opsional: Tambahkan PostgreSQL untuk penyimpanan data.

#### 8.11.2.2 Tugas Mahasiswa TRK – Docker + ThingsBoard

1. Deskripsi Tugas
   Instal dan jalankan platform ThingsBoard Community Edition menggunakan Docker Compose. Pastikan layanan dapat berjalan dan diakses melalui browser.
2. Kriteria Teknis:
   - Jalankan ThingsBoard menggunakan Compose dengan minimal:
     - PostgreSQL
     - ThingsBoard service
   - Akses ThingsBoard melalui http://localhost:8080.
   - Buat user admin dan lakukan login.
   - Screenshot halaman dashboard ThingsBoard.
3. Referensi Struktur Compose:
   Gunakan Compose file yang sesuai dari dokumentasi resmi ThingsBoard:

```
git clone https://github.com/thingsboard/thingsboard.git
cd thingsboard/docker
```

4. Pengumpulan. Tugas dikumpulkan dalam bentuk:
   - File .zip berisi seluruh folder proyek.
   - File .md atau .pdf dokumentasi singkat (deskripsi tugas, cara menjalankan, kendala).
   - Screenshot aplikasi berjalan di browser.
5. Deadline:
6. Penilaian:
   - Struktur folder & file, 20%
   - Fungsi aplikasi berjalan, 30%
   - Konfigurasi Compose benar, 30%
   - Dokumentasi, 20%