

---

# POLITEKNIK NEGERI LHOKSEUMAWE

JURUSAN TEKNOLOGI INFORMASI DAN KOMPUTER

TEKNOLOGI REKAYASA KOMPUTER JARINGAN


---




---


## TRKJ-6435 — CLOUD COMPUTING

---

DOSEN PENGAMPU: Muhammad Davi, S.Kom., M.Cs. 

Gedung Terpadu Jurusan Teknologi Informasi dan Komputer 

muhammad.davi@pnl.ac.id 

2023-2024 Genap 

## **Daftar Isi**

<b>1</b>	<b>Lab 1.1: Docker Image "hello-world"</b>	<b>2</b>
<b>2</b>	<b>Lab 1.2: Shell</b>	<b>3</b>
<b>3</b>	<b>Lab 1.3: Membuat Akun di Docker Hub</b>	<b>4</b>
<b>4</b>	<b>Lab 1.4: Membuat Image baru dan Menyimpan di Docker Hub</b>	<b>6</b>
<b>5</b>	<b>Docker Image "node_project"</b>	<b>2</b>
<b>6</b>	<b>Simpan Image di Dockerhub</b>	<b>3</b>
<b>7</b>	<b>Jalankan docker image teman Anda dari docker hub</b>	<b>4</b>
<b>8</b>	<b>Docker Volume</b>	<b>2</b>
<b>9</b>	<b>Multi kontainer dengan docker-compose</b>	<b>4</b>

---

# LAB 1 — DOCKER

---

## 1 Lab 1.1: Docker Image "hello-world"

Pertama periksa dulu versi docker yang digunakan dengan perintah berikut:

```
1 $ docker version
```

Tuliskan versi docker disini: .....

Dari output diatas, Docker diimplementasikan dengan bahasa pemrograman: .....

Anda ingin menjalankan sebuah Image yang bernama hello-world. Image ini tersedia di Docker Hub. Namun sebelum menjalankan instruksi tersebut, lebih dahulu periksa apakah ada container dan image yang aktif.

```
1 $ docker images
```

Jika ternyata ada images yang sudah terdaftar, maka hapus lebih dahulu dengan instruksi dibawah ini.

```
1 // hentikan proses docker yang masih aktif
2 $ docker stop $(docker ps -a -q)
3
4
5 // hapus semua kontainer yang ada
6 $ docker rm $(docker ps -a -q)
7
8 // hapus semua images
9 $ docker rmi $(docker images -a -q)
10
11
12 // atau hapus semua image yang ada dan kontainer yg stop
13 $ docker system prune -a
```

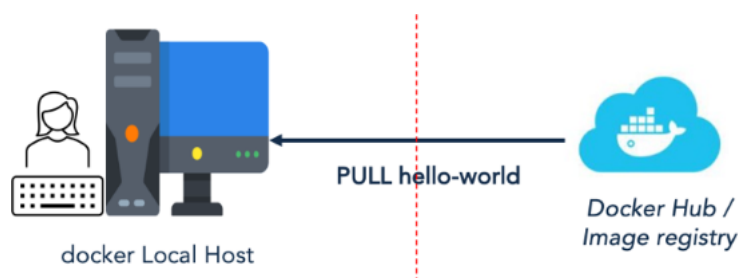
Opsi -a menampilkan semua elemen, sedangkan -q menampilkan ID nya saja. Ketiga instruksi ini dilakukan hanya jika ada proses tersebut yang aktif, urutannya penting, mulai dari proses, kontainer dan image. Selanjutnya lakukan instruksi berikut:

```
1 $ docker run hello-world
```

Pesan pertama yang muncul adalah:

Unable to find image 'hello-world:latest' locally

Docker berusaha untuk menjalankan image yang bernama hello-world, tapi tidak ditemukan di localhost. Oleh karena itu docker secara otomatis mencari hello-world di Docker Hub. Image tersebut ditemukan disana, dan selanjutnya docker mengambil (pull) image tersebut ke localhost, kemudian menjalankannya setelah selesai menarik image tersebut.



Bila Anda menjalankan untuk kedua kali, maka reaksi dari localhost lebih cepat karena image hello-world sudah berada di localhost, sehingga tidak lagi perlu mengunduh (pull) dari docker hub.

```
1 $ docker images
2 $ docker run hello-world
```

Tuliskan Image ID disini: .....  
Kalau punya ..... Image ID-nya: .....

## 2 Lab 1.2: Shell

Anda ingin mengaktifkan Image seperti Lab 1, tapi kali ini Anda ingin masuk ke kontainer tersebut untuk bekerja diatas kontainer seperti bekerja diatas sebuah Sistem Operasi yang dengan menggunakan Shell.

Sebelumnya pilih dulu Sistem Operasi yang akan digunakan, umumnya versi Linux. Ada 3 versi yang populer, yaitu RedHat, Ubuntu, Suse dan turunan lainnya seperti CentOS, Alpine, Linux Mint, Manjaro dan lainnya. Masing-masing punya kelebihan atau konsentrasi di segment tertentu.

Kita akan coba Linux yang relatif paling ringan, yaitu Alpine. Cari dulu di Docker Hub, apakah image alpine tersedia.

```
1 $ docker search alpine
```

Ada banyak image alpine terdaftar disitu, lengkap dengan utilitas lainnya. Yang kita butuhkan adalah alpine.

```
1 $ docker pull alpine
2 $ docker images
```

Perhatikan bahwa Tag adalah versi, dan bila tidak diberikan, nilai default adalah "latest". Sehingga instruksi berikut ekuivalen dengan sebelumnya.

```
1 $ docker pull alpine:latest
```

Jalankan sebagai kontainer dan bekerja dengan shell (bash):

```
1 $ docker run -it alpine
```

Opsi -it artinya jalankan sebagai interactive terminal (sebagai terminal). Jalankan beberapa instruksi shell untuk mencoba:

```
1 # id
2 # date
3 # hostname
4 # netstat -r
5 # exit
```

Anda bisa melihat proses aktif di kontainer dengan ps (proses status)

```
1 $ docker ps
```

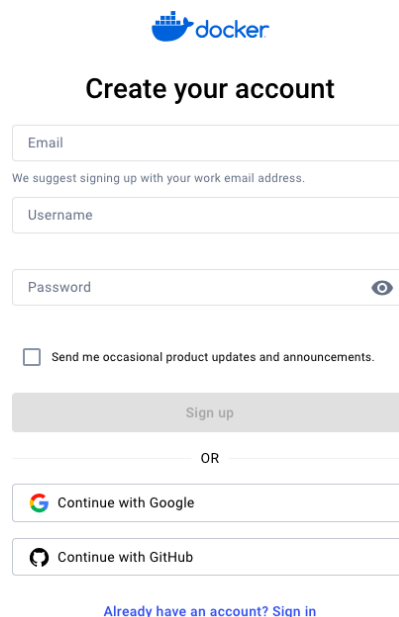
Tapi tidak ada kontainer yang aktif. Untuk melihat jejak proses sebelumnya, gunakan opsi -a (all).

```
1 $ docker ps -a
```

Banyak proses yang sudah berjalan dan selesai dengan status Exited.

### 3 Lab 1.3: Membuat Akun di Docker Hub

Persiapkan email yang valid yang dapat segera diakses. Validasi token akan diberikan melalui akun email tersebut. Arahkan browser ke: <https://hub.docker.com>. Pilih Sign up untuk membuat akun baru.







This site is protected by reCAPTCHA and the Google [Privacy Policy](#) and [Terms of Service](#) apply.

By creating an account I agree to the [Subscription Service Agreement](#), [Privacy Policy](#), [Data Processing Terms](#).

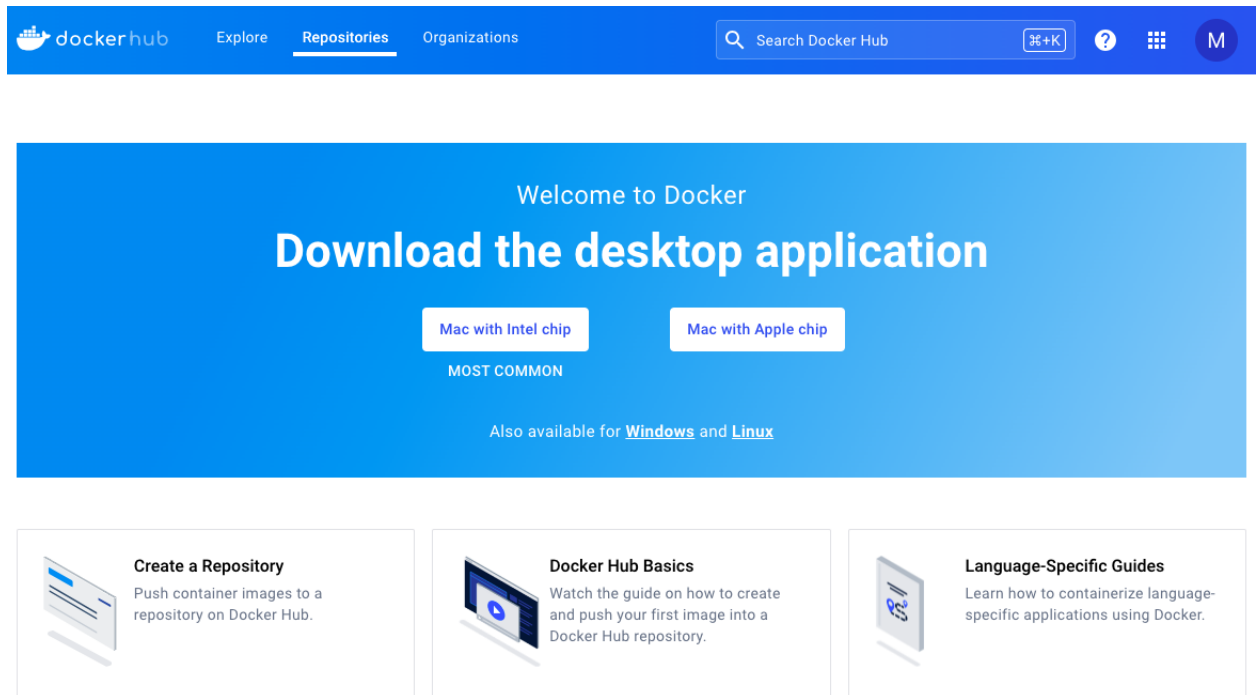
Tulis identitas, alamat email, dan password untuk login ke docker hub. Ikuti instruksi untuk validasi dan verifikasi (biasanya melalui pertanyaan pada sebuah gambar yang diberikan). Pilih Skema yang Free (\$0), "**Continue with Personal**".

#### Choose your Docker Core subscription

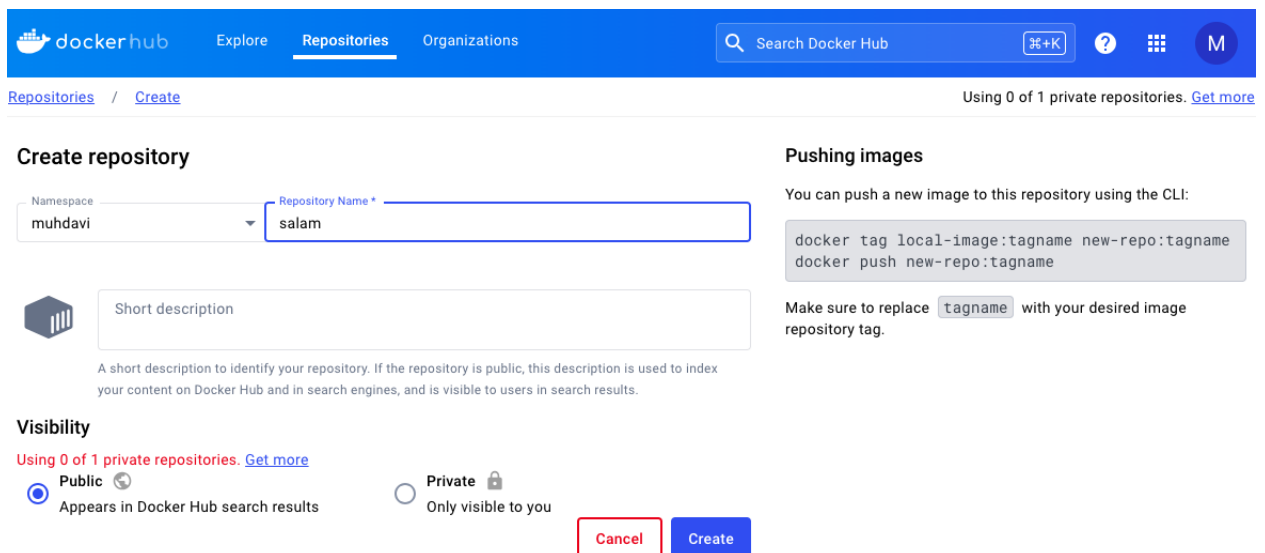
Use of Docker Desktop at a company of more than 250 employees OR more than \$10 million in annual revenue requires a paid subscription (Pro, Team, or Business).

Personal	Pro	Team	Business
For new developers and/or students getting started with containers.	For professional developers who want to accelerate innovation.	For development teams looking to increase collaboration and agility.	For businesses seeking to establish an enterprise-grade development approach.
<b>\$0</b>	<b>\$5</b> / month	<b>\$9</b> / seat / month	<b>\$24</b> / seat / month
<ul style="list-style-type: none"><li>✓ Docker Desktop Personal</li><li>✓ Unlimited public repositories</li><li>✓ Docker Engine + Kubernetes</li><li>✓ 200 image pulls per 6 hours</li><li>✓ 3 Scout-enabled repositories</li><li>✓ Local Scout analysis</li></ul>	<ul style="list-style-type: none"><li>← Everything in Personal, plus:</li><li>✓ Docker Desktop Pro</li><li>✓ Unlimited private repositories</li><li>✓ 5,000 image pulls per day</li><li>✓ 5 concurrent builds</li><li>✓ Synchronized file shares</li><li>✓ Local Scout analysis and remediation</li><li>✓ 5 day support response</li></ul>	<ul style="list-style-type: none"><li>← Everything in Pro, plus:</li><li>✓ Docker Desktop Team</li><li>✓ Unlimited teams</li><li>✓ 15 concurrent builds</li><li>✓ Add users in bulk</li><li>✓ Audit logs</li><li>✓ Up to 100 seats</li><li>✓ Role-based access control</li><li>✓ 2 day support response</li></ul>	<ul style="list-style-type: none"><li>← Everything in Team, plus:</li><li>✓ Docker Desktop Business</li><li>✓ Hardened Docker Desktop</li><li>✓ Single Sign-On (SSO)</li><li>✓ SCIM user provisioning</li><li>✓ VDI support</li><li>✓ Private Extensions Marketplace</li><li>✓ Image and Registry Access Management</li><li>✓ Purchase via invoice</li><li>✓ Priority case routing</li><li>✓ Proactive case monitoring</li><li>✓ 24-hour support response</li></ul>
			
Included minutes: 400 mins / month Included cache: 100 GB	Included minutes: 400 mins / month Included cache: 100 GB	Included minutes: 400 mins / month Included cache: 100 GB	Included minutes: 800 mins / month Included cache: 200 GB
<a href="#">Continue with Personal</a>	<a href="#">Buy Now</a>	<a href="#">Buy Now</a>	<a href="#">Buy Now</a>

Abaikan Download the desktop application. Pilih **Create a Repository**.



Pada Menu **Create repository**, tulis nama repository - pada contoh adalah "salam". Berikan deskripsi terkait. Visibility adalah "public", yang mana repository ini dapat diakses oleh siapa saja di Internet.



Hasil Pembuatan repository. Disebelah kanan adalah informasi untuk melakukan "push" yaitu menyimpan docker image di docker hub. Instruksi tersebut dilakukan dari komputer client setelah membuat image docker baru.

The screenshot shows the Docker Hub interface for a repository named 'muhdavi/salam'. The repository is newly created (less than a minute ago) and is currently incomplete, as indicated by the 'INCOMPLETE' status and the absence of a description and category. The 'General' tab is selected, showing the repository name, creation time, and status. The 'Tags' tab is also visible, showing that the repository is empty. The 'Docker commands' section provides a command to push a new tag: `docker push muhdavi/salam:tagname`. The 'Automated Builds' section explains how to connect the repository to GitHub or Bitbucket for automatic builds. A 'Public View' button is located in the top right corner.

## 4 Lab 1.4: Membuat Image baru dan Menyimpan di Docker Hub

1. Buat folder baru untuk image baru

```
1 $ mkdir salam
2 $ cd salam
```

2. Buat sebuah file Golang sebagai berikut dengan nama salam.go:

```
1 package main
2 import "fmt"
3 func main() {
4     fmt.Println("Salam dari Faiza Kelompok 6")
5 }
```

Ganti Faiza dengan nama anda sendiri dan kelompok dengan kelompok Anda.

3. Buat Dockerfile seperti berikut:

```
1 FROM golang:latest
2 # Faiza - faiza@gmail.com
3 WORKDIR /app
4 COPY . .
5 RUN go build salam.go
6 CMD ["/app/salam"]
```

4. Bentuk Image baru dengan docker build

Perhatikan, ganti nama muhdavi dengan nama akun Anda di Docker Hub. Jangan lupa ada titik pada akhir instruksi, yang berarti mengolah atau membaca Dockerfile dari Folder Aktual (current directory).

```
1 $ docker build -t muhdavi/salam .
```

5. Periksa hasilnya dengan:



```
1 $ docker images
```

6. Simpan image tersebut di Docker Hub. Sebelumnya login terlebih dahulu ke repositori.

```
1 $ docker login -u muhdavi
2 password:
3
4 $ docker push muhdavi/salam
```

7. Minta partner ada untuk mengambil dan menjalankan program tersebut dengan perintah berikut:

```
1 $ docker pull muhdavi/salam
```

Atau langsung dengan run (otomatis menjalankan pull)

```
1 $ docker run muhdavi/salam
```

Perhatikan: ganti nama muhdavi dengan user name Anda di docker hub.

---

## LAB 2 — DOCKER

---

## 5 Docker Image "node\_project"

Buat direktori tempat kerja Anda di direktori lokal dengan **mkdir** dan kemudian pindah ke direktori tersebut dengan **cd** (change directory).

```
1 $ mkdir node_project
2 $ cd node_project
```

Cek instalasi npm dengan perintah berikut:

```
1 $ npm -v
```

Bila belum ada, download dan install melalui link <https://nodejs.org/en/download/>. Selesai instalasi npm, lakukan `npm init` dan hasilnya sebuah file teks.

```
1 $ npm init
```

Periksa file yang terbentuk, dengan:

```
1 $ cat package.json
```

Hasilnya akan mirip tampilan disini:

```
1 {
2   "name": "node_project",
3   "version": "1.0.0",
4   "description": "Cloud Computing",
5   "main": "index.js",
6   "scripts": {
7     "test": "echo \"Error: no test specified\" && exit 1"
8   },
9   "author": "Muhammad Davi",
10  "license": "ISC",
11  "dependencies": {
12    "express": "^4.19.2"
13  }
14 }
```

File `package.json` akan ditambahkan modul "express" dengan perintah berikut:

```
1 $ npm install express
```

Hasil dari penambahan modul di file `package.json` sebagai berikut:

```
1   "author": "Muhammad Davi",
2   "license": "ISC",
3   "dependencies": {
4     "express": "^4.19.2"
5   }
6 }
```

Selanjutnya buat sebuah file dengan nama `index.js` dan isi dengan kode berikut ini:

```
1 const express = require("express")
2 const app = express()
3 const port = 7000
4
5 app.get("/", (req, resp) => {
6   // identitas
```

```

7     resp.send("Nama saya Faiza dari Kelompok 6");
8 })
9
10 app.listen(port, () => console.log('listening on port ${port}'))

```

Perhatikan bahwa tanda petik di string listening adalah tanda kutip terbalik ('), bukan tandan petik (.). Tanda ini akan melakukan substitusi \$port menjadi 7000.

```

1 // Terminal 1
2 $ node index.js
3 Listening on port 7000

```

Kemudian aktifkan terminal 2 dan jalankan curl:

```

1 // Terminal 2
2 $ curl localhost:7000
3 Nama saya Faiza dari Kelompok 6

```

Dan coba buka di web browser dengan url <http://localhost:7000> apakah menghasilkan output yang sama?

Selanjutnya coba buat Dockerfile menggunakan kode berikut ini:

```

1 FROM node:latest
2 WORKDIR /app
3 COPY package.json .
4 RUN npm install
5 COPY . .
6 EXPOSE 7000
7 CMD ["node", "index.js"]

```

Buat docker image dengan perintah `docker build .` ← jangan lupa tanda titik di belakang.

Kemudian periksa apakah image tersebut sudah ada?

Tuliskan Image ID disini: .....

Kalau punya ..... Image ID-nya: .....

Perbaiki image tersebut dari <none> dengan nama image menggunakan opsi -t dan cantumkan nomor versi 1.0 (node\_project:1.0), jangan menggunakan latest.

```

1 $ docker build -t muhdavi/node_project:1.0 .

```

Lihat hasilnya dengan `docker images` dan periksa status dengan `docker ps`.

Jalankan docker images tersebut sebagai proses daemon dengan menggunakan opsi -d (detach). Berikan nama kontainer dengan menggunakan opsi -name dan lakukan port mapping dengan opsi -p agar aplikasi dapat diakses. Coba akses menggunakan web browser.

```

1 $ docker run -d --name node_project -p 80:7000 muhdavi/node_project:1.0

```

## 6 Simpan Image di Dockerhub

Siapkan image yang sudah dibuat untuk dikirim ke docker hub. Langkah pertama adalah menggunakan login ke dockerhub lengkap dengan password. Langkah kedua adalah melakukan docker push.

## 7 Jalankan docker image teman Anda dari docker hub

Untuk lab ini dibutuhkan nama akun teman Anda dan nama repositori beserta nomor versi nya.

```
1 $ docker pull nama_akun/nama_repositori:nomor_versi
```

Atau dengan `docker run` untuk menjalankan secara otomatis.

```
1 $ docker run nama_akun/nama_repositori:nomor_versi
```

**Perhatikan:** kemungkinan Anda harus membuat port-mapper dan sebaiknya juga memberi nama pada kontainer tersebut.

---

## LAB 3 — DOCKER

---

## 8 Docker Volume

Docker volume dibutuhkan oleh kontainer karena kontainer tidak dapat menyimpan data secara persisten. Docker volume menawarkan fisik disk yang ada di Host sebagai media penyimpanan untuk kontainer. Dengan demikian jika kontainer selesai bekerja dan hilang dari Host, maka data tersebut tetap ada.

Keuntungan lain dari docker volume adalah menghemat media penyimpanan yang mana Multiple Container dapat mengakses folder dan data yang sama. Hal ini dapat direalisasikan dengan membuat "named volume" artinya volume yang secara eksplisit diberi nama.

Terdapat beberapa instruksi shell primitif yang akan digunakan.

- **touch file.txt**  
Membuat file kosong, hanya menciptakan nama file.
- **echo "string" > file.txt**  
Mengisi konten file.txt, bila sebelumnya sudah ada isi, maka isi tersebut akan dihapus. Bila belum pernah ada, file.txt akan diciptakan.
- **echo "string juga" » file.txt**  
Mengisi konten file.txt pada akhir file, menyambung dengan isi sebelumnya.

Beberapa perintah untuk melihat environment kerja

```
1 $ docker info
2 $ docker images
3 $ docker ps
4 $ docker volume ls
```

Agar bisa dimulai dengan data kosong, lakukan hal berikut. Pastikan tidak ada container yang "running" saat Lab dimulai.

```
1 $ docker system prune -a
2 $ docker volume prune
```

Docker Image "alpine" adalah sistem operasi Linux yang sangat ringan, tidak seberat Ubuntu atau lainnya. Alpine digunakan sebagai basis OS untuk Lab berikut. Opsi -i menandakan bahwa input akan diberikan dari keyboard secara interaktif dan opsi -t menandakan modus terminal. Opsi -i -t atau -it mengaktifkan OS sebagai terminal shell. Instruksi shell dapat diberikan langsung. Opsi -v digunakan sebagai opsi untuk volume, -v nama folder, folder tersebut digunakan sebagai volume.

```
1 $ mkdir -p /home/muhdavi/localdata
2 $ docker run -it -v /home/muhdavi/localdata:/mydata --name k1 alpine
3 $ cd /mydata
4 $ touch f1 f2 f3
5 $ ls -al
6 $ ctrl+d (^d)
7 $ docker ps
8 $ ls /home/muhdavi/localdata
```

Kembali aktifkan kontainer k1 dengan interactive mode.

```
1 $ docker start -i k1
2 $ ls mydata
3 $ docker volume ls
```

Volume anonymous tidak menggunakan nama untuk mengakses volume tersebut. Docker otomatis memberi hash-number sebagai nama folder. Sebelum memasuki Lab, ada baiknya hapus semua kontainer dan image agar tidak tercampur image, kontainer dan obyek lain hasil dari lab sebelumnya.

```
1 $ docker system prune -a
2 $ docker volume prune
```

Intruksi untuk menghasilkan volume anonymous adalah sebagai berikut:

```
1 $ docker run -it -v /myfolder --name k2 alpine
2 $ cd myfolder
3 $ touch f4 f5 f6
```

Aktifkan Terminal 2, lakukan instruksi berikut:

```
1 $ docker ps
2 $ docker volume ls
```

Nama folder local yang diberikan berupa random hash number. Docker melakukan random hash number ini agar tidak terjadi duplikasi dengan nama folder yang sudah diciptakan sebelumnya. Berikut ini inspeksi kontainer k2 untuk melihat informasi details, dimana folder tersebut berada.

```
1 $ docker inspect k2
2 $ docker volume ls
3 $ docker volume inspect
```

Asynchronous volume ini dapat diakses bersama dengan kontainer lain. Buka dari terminal 2, buat kontainer baru dengan nama k3 menggunakan volume dari kontainer k2 tersebut.

```
1 $ docker run -it --volumes-from k2 --name k3 alpine
2 $ cd myfolder
3 $ echo "salam dari kontainer 3" > pesan
```

Dari terminal 1 lakukan hal berikut.

```
1 $ cat pesan
2 $ echo "salam kembali untuk kontainer 3 dari kontainer 2" >> pesan
```

Kembali ke Terminal 2 untuk melihat hasilnya. Berdasarkan hasil yang ditampilkan dua kontainer ini melakukan shared file.

```
1 $ cat pesan
```

Kali ini volume diberikan nama, sehingga kontainer dapat menggunakan nama tersebut sebagai referensi, tidak lagi menggunakan random-hash. Pemberian nama akan memudahkan multiple kontainer untuk mengakses volume tersebut. Sebelum memasuki lab, sebaiknya dihapus semua kontainer dan image agar tidak tercampur image, kontainer dan obyek lain hasil dari lab sebelumnya.

```
1 $ docker system prune -a
2 $ docker volume prune
```

Lakukan instruksi berikut untuk menghasilkan volume bernama. Namakan secara logical "ourdata" karena volume tersebut akan diakses oleh banyak kontainer.



```

1 $ docker volume create ourdata
2 $ docker volume ls
3 $ docker run -it -v ourdata:/projectdata --name k4 alpine
4 $ ls
5 $ cd projectdata
6 $ echo "Ini pesan dari k4 alpine" > pesan
7 $ ls

```

Dari terminal 2

```

1 $ docker run -it -v ourdata:/projectdata --name k5 alpine
2 $ cd projectdata
3 $ echo "Ini pesan dari k5 alpine" >> pesan
4 $ cat pesan

```

Dari terminal 3 lihat status kontainer tersebut

```

1 $ docker ps
2 $ docker volume ls
3 $ docker images

```

## 9 Multi kontainer dengan docker-compose

Docker compose merupakan alat (tools) untuk menjalankan kontainer docker secara bersamaan. File compose digunakan untuk melakukan konfigurasi layanan aplikasi. Docker compose sering digunakan untuk development dan testing workflows karena seluruh kontainer dapat berjalan menggunakan single host secara default. Kali ini akan membuat multi services menggunakan docker compose dengan wordpress dan database.

1. Membuat multi kontainer secara otomatis dengan menggunakan tools docker compose dengan image wordpress dan mariadb. Buatlah direktori untuk menyimpan data mariadb dan wordpress persistent dengan nama \$HOME/wp-compose/database dan \$HOME/wp-compose/html dan masuk ke direktori \$HOME/wp-compose.

```

1 $ mkdir -p $HOME/wp-compose/{database, html}
2 $ cd $HOME/wp-compose

```

2. Verifikasi versi docker compose yang digunakan saat ini

```

1 $ docker-compose --version

```

3. Membuat multi kontainer aplikasi Docker menggunakan docker-compose. Buatlah file docker-compose.yml dengan menggunakan teks editor apapun yang Anda sukai, file ini digunakan untuk melakukan konfigurasi seluruh layanan aplikasi (wordpress dan mariadb). Tambahkan baris code berikut ke dalam file docker-compose.yml.

```

1 $ vi docker-compose.yml

```

```

1 version: '3.3'
2
3 services:
4   mariadb:

```

```

5  image: mariadb
6  volumes:
7    - ./database:/var/lib/mysql
8  restart: always
9  environment:
10   MYSQL_ROOT_PASSWORD: rootpassword
11   MYSQL_DATABASE: wordpress
12   MYSQL_USER: wordpress
13   MYSQL_PASSWORD: wordpress
14
15  wordpress:
16    depends_on:
17      - mariadb
18    images: wordpress:latest
19    ports:
20      - "8001:80"
21    restart: always
22    environment:
23      WORDPRESS_DB_HOST: mariadb:3306
24      WORDPRESS_DB_USER: wordpress
25      WORDPRESS_DB_PASSWORD: wordpress
26      WORDPRESS_DB_NAME: wordpress
27    volumes:
28      - ./html:/var/www/html

```

### Catatan

Service wordpress tidak akan dijalankan sebelum service mariadb berjalan (depends\_on: mariadb). Image untuk mendefinisikan image yang akan digunakan pada kontainer. Ports untuk mendefinisikan port mana yang akan di expose sebagai host. Sama halnya dengan parameter -p pada perintah docker run. Setelah memiliki images yang dibutuhkan maka tidak perlu menambahkan images baru.

4. Menjalankan multi service kontainer dengan menggunakan docker-compose. Untuk menjalankannya masuk ke dalam direktori dimana lokasi dari docker-compose.yml berada dengan perintah docker-compose up -d.

```
1 $ docker-compose up -d
```

Untuk melihat log aplikasi, dapat menggunakan perintah docker-compose logs.

```
1 $ docker-compose logs -f wordpress
```

Untuk melihat informasi aplikasi yang dibuat dengan docker-compose, dapat menggunakan perintah docker-compose ps.

```
1 $ docker-compose ps
```

5. Setelah proses *docker-compose* berhasil maka, web dapat diakses melalui browser ke <http://localhost:8001>, sehingga muncul tampilan instalasi wordpress.
6. Untuk stop dan menghapus kontainer dari service saat ini, dapat menggunakan perintah docker-compose down.

```
1 $ docker-compose down
```

### Catatan

Instalasi docker-compose di ubuntu

1. Melakukan instalasi docker-compose dengan mengunduh dan menyimpan file di **/usr/local/bin/docker-compose** dengan menjalankan perintah berikut:

```
1 $ sudo curl -L "https://github.com/docker/compose/releases/download/v2.28.1/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
```

2. Ubah izin akses agar file /usr/local/bin/docker-compose dapat di eksekusi dengan perintah berikut ini:

```
1 $ sudo chmod +x /usr/local/bin/docker-compose
```

3. Verifikasi instalasi berhasil dengan menjalankan perintah berikut ini:

```
1 $ docker-compose --version
```