PANDUAN PRAKTIKUM

DOCKER & KUBERNETES

I PUTU HARIYADI

PANDUAN PRAKTIKUM
DOCKER & KUBERNETES

OLEH
I PUTU HARIYADI

www.iputuhariyadi.net

KATA PENGANTAR

Puji syukur penyusun panjatkan kepada Tuhan Yang Maha Esa atas berkat dan rahmatnya sehingga "**Panduan Praktikum Docker & Kubernetes**" ini dapat terselesaikan. Panduan ini dibuat sebagai petunjuk praktik bagi Anda yang tertarik mempelajari teknologi **Cloud Native** menggunakan **Docker** dan **Kubernetes**. Materi yang dibahas pada panduan ini meliputi instalasi dan konfigurasi *Ubuntu Desktop* pada *Oracle VirtualBox*, instalasi dan konfigurasi *Docker Engine* pada *Ubuntu Desktop*, *Docker Fundamental*, membangun *container images* menggunakan *Dockerfile* dan membangun aplikasi *multi-container* menggunakan *Docker Compose*, serta *Kubernetes Fundamental*.

Penyusun menyadari bahwa panduan praktikum ini masih jauh dari sempurna. Untuk itu kritik dan saran demi pengembangan panduan praktikum ini sangat diharapkan. Kritik dan saran dapat dikirimkan melalui email dengan alamat: atau admin@iputuhariyadi.net. Terimakasih.

Mataram, 20 Agustus 2024

Penyusun

DAFTAR ISI

PENDAHULUAN

Adapun kebutuhan perangkat keras (*hardware*) dan lunak (*software*) yang diperlukan untuk dapat mengujicoba materi yang terdapat pada panduan praktikum ini adalah sebagai berikut:

A. Kebutuhan *Hardware*

Satu unit komputer dengan rekomendasi spesifikasi sebagai berikut:

1. CPU: 64 bit.
2. RAM: 8 GB.
3. Hard drive.
4. 1 (satu) *Network Interface Card*.

B. Kebutuhan *Software*

1. *Ubuntu Desktop versi 20.04 LTS* yang dapat diunduh melalui situs *Ubuntu* di alamat https://ubuntu.com/download/alternative-downloads atau melalui https://releases.ubuntu.com/jammy/ubuntu-22.04.4-desktop-amd64.iso
2. *Oracle VirtualBox versi 6.1* atau yang lebih baru dan dapat diunduh melalui situs *VirtualBox* di alamat https://www.virtualbox.org.
3. *Putty SSH Client* yang dapat diunduh pada alamat https://www.putty.org/
4. *Browser Chrome* yang dapat diunduh pada alamat https://www.google.com/chrome/.
5. *WinSCP* untuk melakukan file transfer dari Windows ke Ubuntu Desktop yang dapat diunduh pada alamat https://winscp.net/eng/index.php

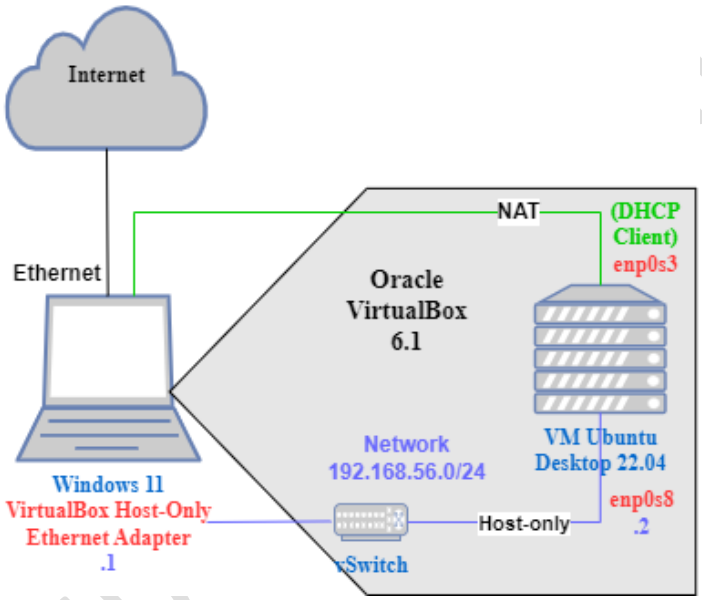
Selain itu juga diperlukan koneksi *Internet* untuk mengunduh perangkat lunak tersebut dan ujicoba materi.

BAB I

INSTALASI DAN KONFIGURASI UBUNTU DESKTOP VERSI 22.04 LTS PADA ORACLE VIRTUALBOX 6.1

A. Rancangan Jaringan Ujicoba

Rancangan jaringan ujicoba terdiri dari 1 unit *notebook* dengan sistem operasi *Windows 11* yang telah diinstalasi *Oracle VirtualBox 6.1* sebagai *hosted hypervisor*, seperti terlihat pada gambar berikut:

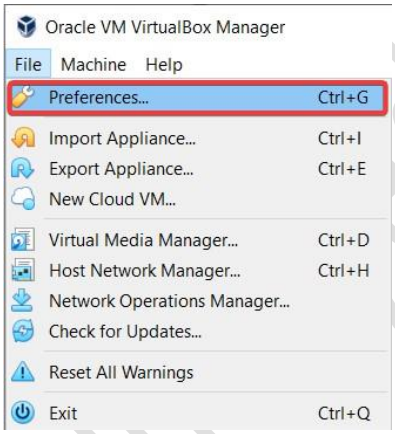


Pada *Oracle VirtualBox* akan dibuat *Guest Virtual Machine* (*VM*) dengan sistem operasi *Ubuntu Desktop* versi *22.04 LTS*. Alamat *network* yang digunakan untuk komunikasi antara *Host Machine Windows* 11 dengan VM *Ubuntu* adalah 192.168.56.0/24. *Interface* ***VirtualBox Host-Only Ethernet Adapter*** pada *Windows* 11 menggunakan pengalamatan IP secara statik yaitu **192.168.56.1/24**. Sedangkan *VM Ubuntu* diatur agar memiliki dua *network adapter*. *Network adapter* pertama **enp0s3** berjenis *Network Address Translation (NAT)* agar VM tersebut dapat terkoneksi ke *Internet* dan pengalamatan IP-nya dilakukan secara dinamis sehingga bertindak sebagai *Dynamic Host Configuration Protocol (DHCP) Client*. *Oracle VirtualBox* akan bertindak sebagai *server* DHCP yang mengalokasikan pengalamatan IP secara dinamis ke DHCP *Client*. Sebaliknya *network adapter* kedua **enp0s8** berjenis *Host-only* untuk koneksi ke *Windows* 11 dan pengalamatan IP-nya dilakukan secara statik yaitu menggunakan **192.168.56.2/24**.

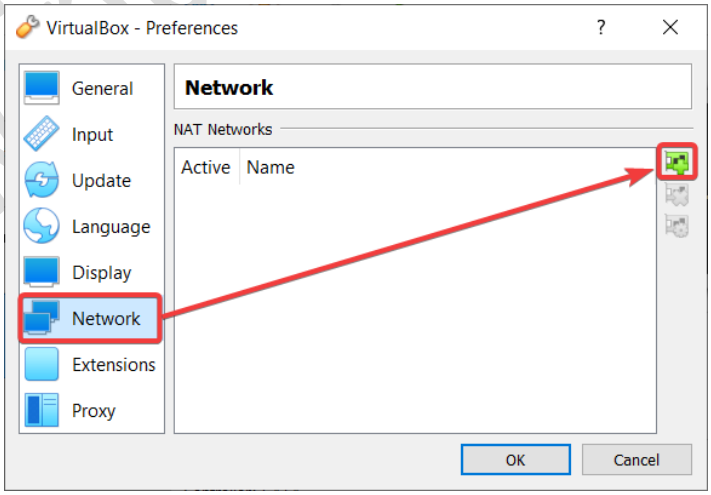
B. Instalasi Ubuntu Desktop Versi 22.04 LTS

Adapun langkah-langkah instalasi dan konfigurasi *Ubuntu Desktop* versi *22.04 LTS* pada *Oracle VirtualBox 6.1* adalah sebagai berikut:

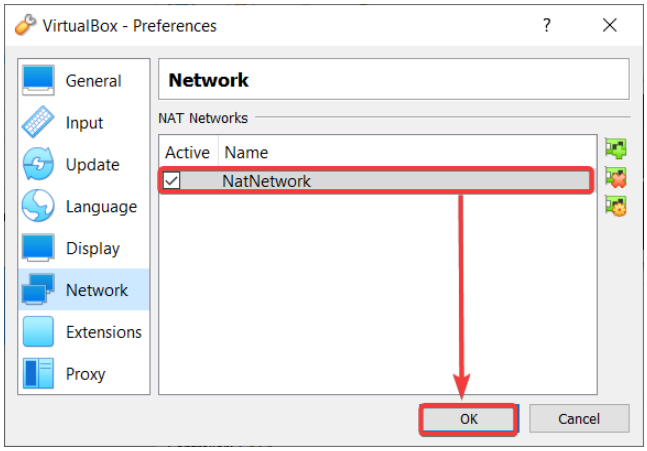
1. Jalankan aplikasi *Oracle VirtualBox* melalui **Start > Oracle VM VirtualBox > Oracle VM VirtualBox**.
2. Tampil aplikasi *Oracle VM VirtualBox Manager* dan lakukan penambahan **Network Address Translation (NAT) Network** dengan mengakses menu **File > Preferences** pada **Oracle VM VirtualBox Manager**, seperti yang ditunjukkan pada gambar berikut:



Tampil kotak dialog **VirtualBox – Preferences** dan pada panel menu sebelah kiri pilih **Network**. Pada panel detail dari *Network* di sebelah kanan, pilih *icon* **Adds new NAT Network**, seperti yang ditunjukkan pada gambar berikut:

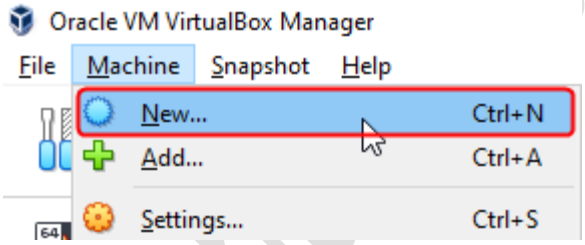


Terbentuk "**NatNetwork**" yang telah aktif, seperti terlihat pada gambar berikut:

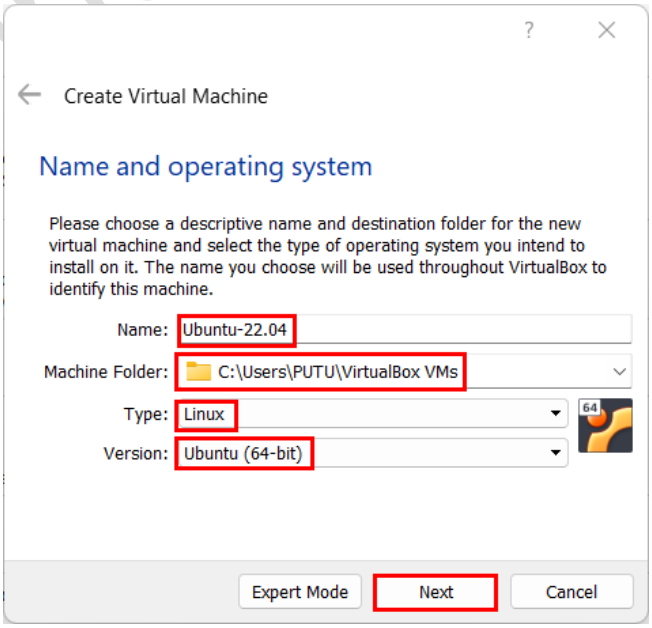


Klik tombok **OK** untuk menutup kotak dialog *VirtualBox - Preferences*.

3. Untuk membuat *virtual machine* baru, maka pada *Oracle VM VirtualBox Manager* pilih menu **Machine > New …**, seperti terlihat pada gambar berikut:



4. Tampil kotak dialog *Create Virtual Machine* untuk menentukan nama pengenal, lokasi penyimpanan dan jenis serta versi dari sistem operasi yang ingin dibuat pada *virtual machine*, seperti terlihat pada gambar berikut:

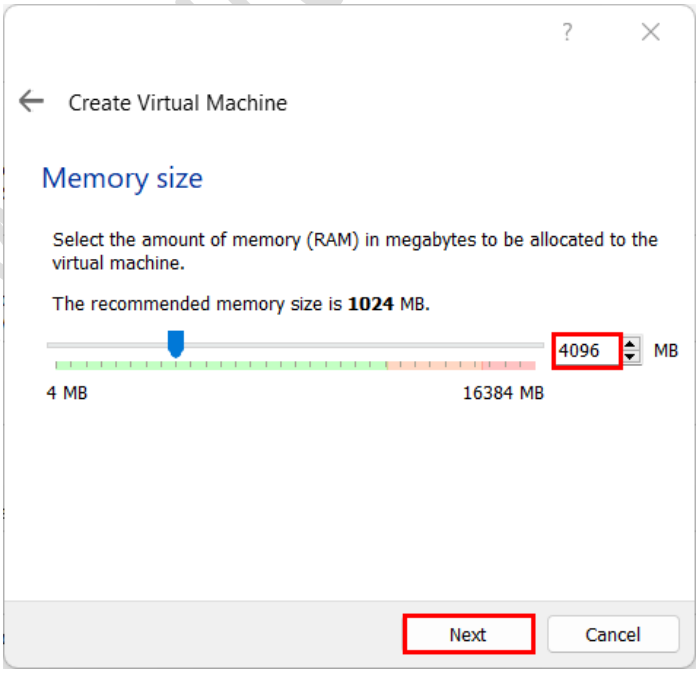


Lengkapi isian beberapa parameter berikut:

a. **Name:** dengan nama pengenal dari *virtual machine*, sebagai contoh **Ubuntu-22.04**.
b. **Machine Folder** digunakan untuk menentukan lokasi penyimpanan *file virtual machine* yang dibuat yaitu secara *default* disimpan di **C:\Users\NamaLogin\VirtualBox VMs. NamaLogin** merupakan nama login pengguna yang digunakan untuk akses ke sistem *Windows*, sebagai contoh **ASUS** sehingga nilai pada *dropdown* **Machine Folder** adalah **C:\Users\PUTU\VirtualBox VMs**. Apabila ingin menyimpan di lokasi lain maka pilih **Other…** pada *dropdown* **Machine Folder** tersebut dan arahkan ke lokasi direktori penyimpanan baru yang diinginkan.
c. **Type** digunakan untuk menentukan jenis sistem operasi yang akan diinstalasi pada *virtual machine* yaitu **Linux**.
d. **Version** digunakan untuk menentukan versi dari sistem operasi yang akan diinstalasi pada *virtual machine* yaitu **Ubuntu (64 bit)**.

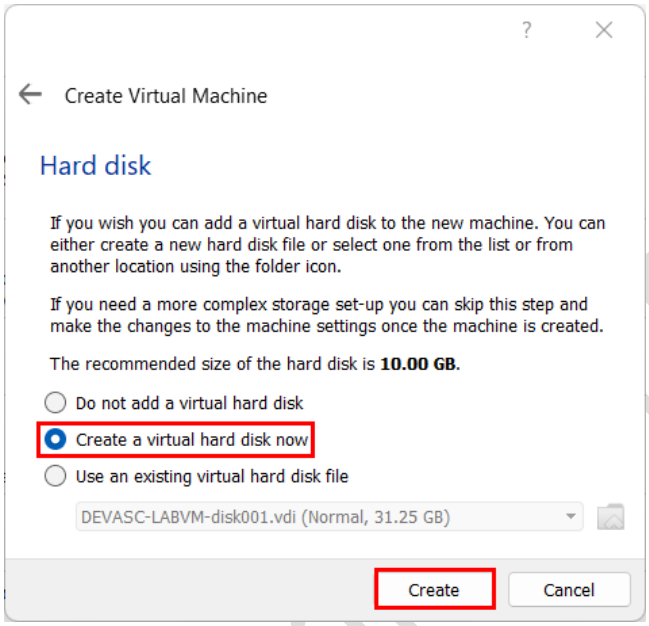
Klik tombol **Next** untuk melanjutkan.

5. Tampil kotak dialog *Memory size* untuk menentukan ukuran memori yang dialokasikan bagi *virtual machine* yang dibuat. Sebagai contoh dialokasikan **4096 MB**, seperti terlihat pada gambar berikut:



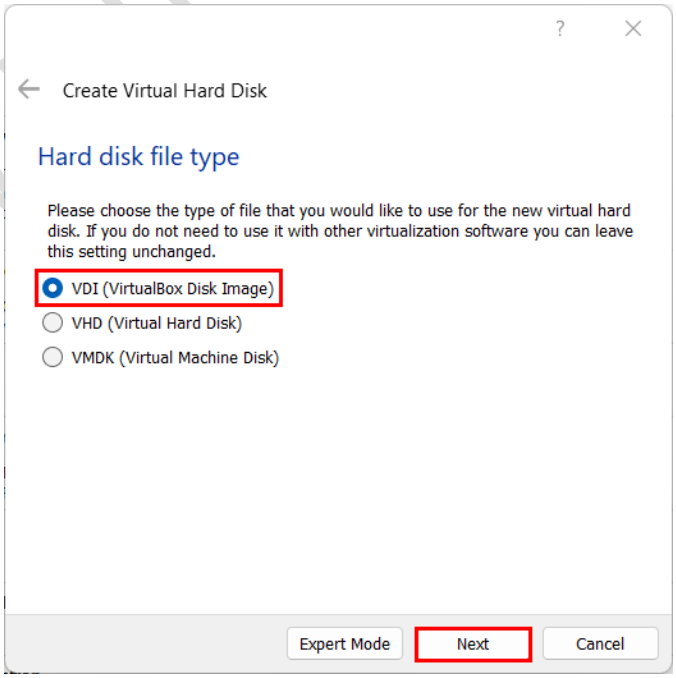
Klik tombol **Next >** untuk melanjutkan.

6. Tampil kotak dialog *Hard disk* untuk menentukan *virtual hardisk* yang digunakan oleh *virtual machine* yang dibuat. Secara default telah terpilih **Create a virtual hard disk now** untuk membuat *virtual hard disk* bagi *virtual machine* baru yang dibuat, seperti terlihat pada gambar berikut:



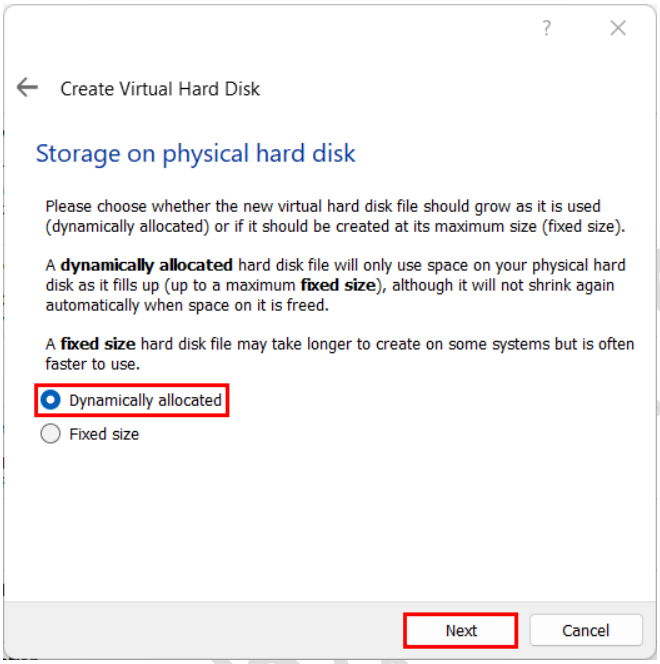
Klik tombol **Create** untuk melanjutkan.

7. Tampil kotak dialog *Hard disk file type* untuk menentukan jenis *file* yang digunakan untuk virtual *hard disk* baru yang dibuat. Secara *default* telah terpilih **VDI (VirtualBox Disk Image)**, seperti terlihat pada gambar berikut:



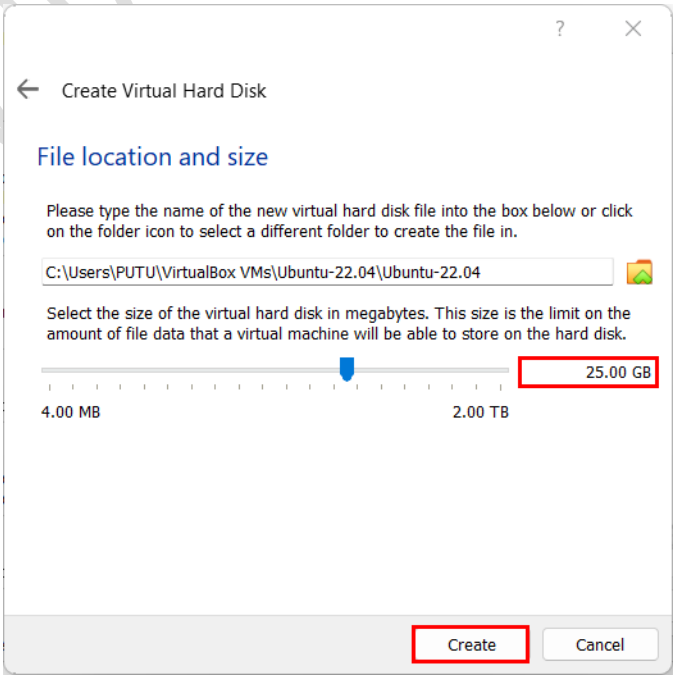
Klik tombol **Next** untuk melanjutkan.

8. Tampil kotak dialog **Storage on physical hard disk** untuk menentukan apakah *file virtual hard disk* yang baru dibuat harus bertumbuh mengikuti penggunaan **(dynamically allocated)** atau dibuat ke ukuran maksimumnya **(fixed size)**. Secara *default* telah terpilih **Dynamically allocated**, seperti terlihat pada gambar berikut:

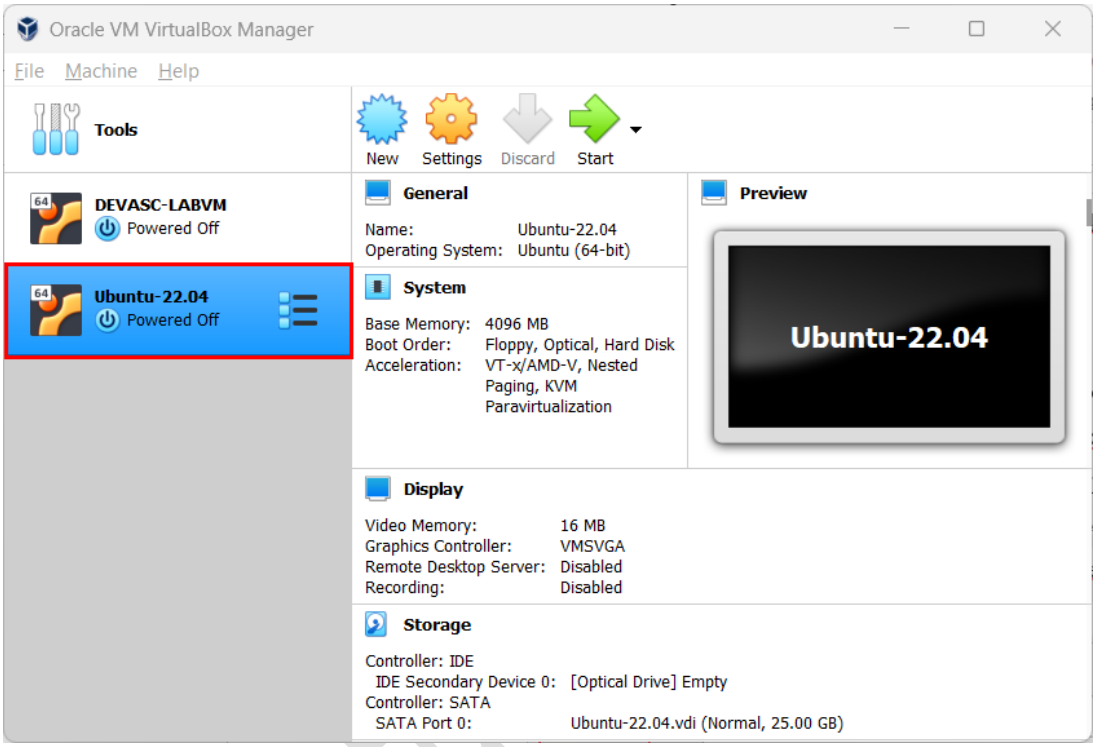


Klik tombol **Next** untuk melanjutkan.

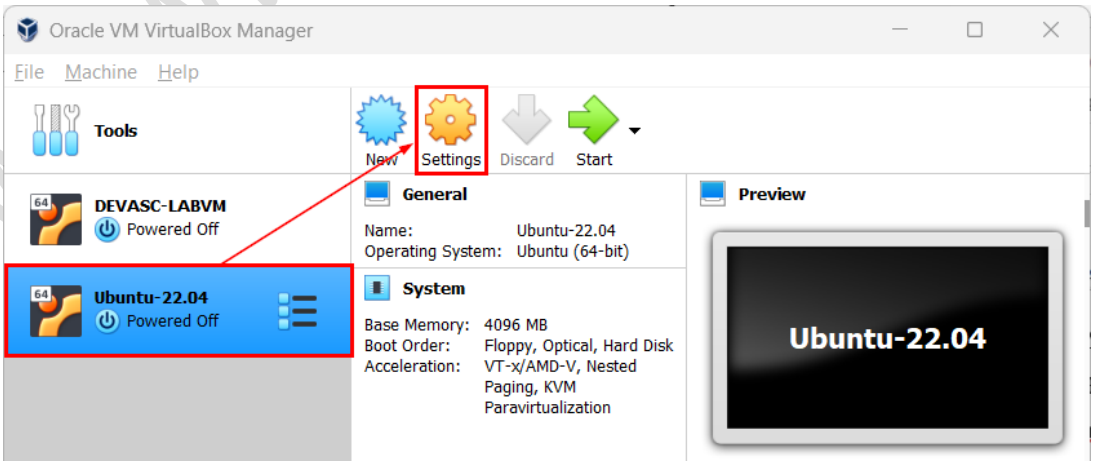
9. Tampil kotak dialog *File location and size* untuk menentukan lokasi dan ukuran *virtual hard disk* yang dapat digunakan oleh *virtual machine* sebagai penyimpanan. Sebagai contoh dialokasikan **25.00 GB**, seperti terlihat pada gambar berikut:



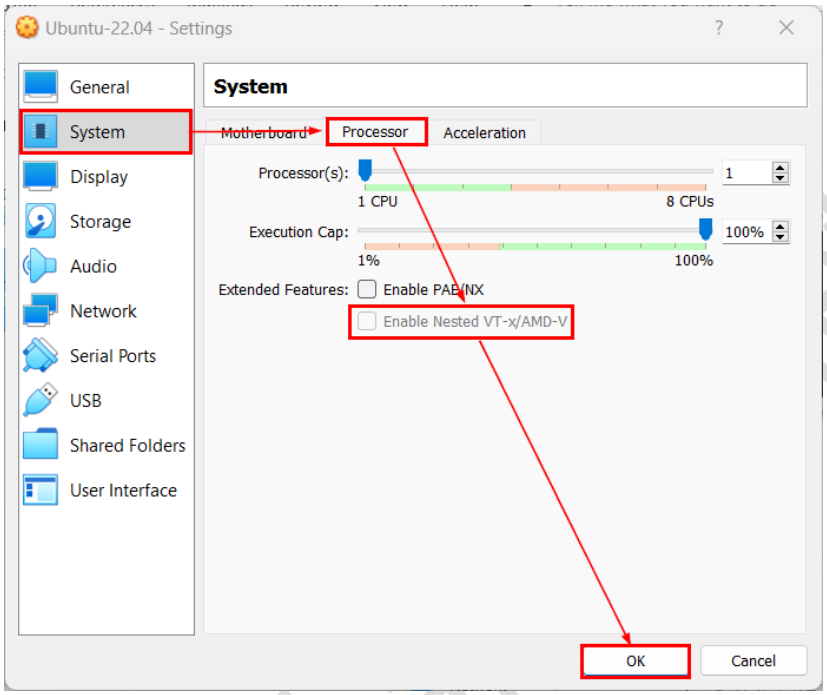
Klik tombol **Create** untuk melanjutkan. Tunggu beberapa saat, proses pembuatan *virtual machine* sedang dilakukan. Apabila proses pembuatan VM berhasil dilakukan maka pada daftar dari **Oracle VM VirtualBox Manager** akan memperlihatkan **VM** dengan nama pengenal **Ubuntu-22.04**.



10. Mengaktifkan **nested VT-X/AMD-V** di *VirtualBox* untuk VM dengan nama **Ubuntu-22.04**. Secara *default* fitur tersebut tidak dapat diaktifkan. Hal ini dapat diverifikasi dengan mengakses **Ubuntu-22.04 - Settings** melalui pemilihan **Settings** pada *toolbar* dari *Oracle VM VirtualBox Manager*, seperti terlihat pada gambar berikut:

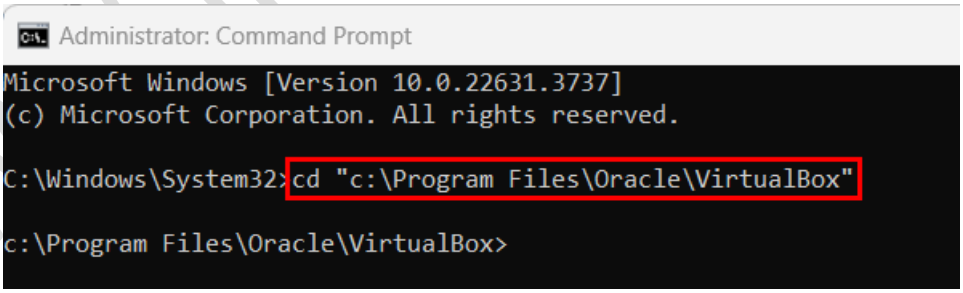


Pada kotak dialog **Ubuntu-22.04 – Settings** yang tampil, pilih **System** pada panel menu sebelah kiri dan pada panel detail sebelah kanan pilih tab **Processor**, seperti yang ditunjukkan pada gambar berikut:



Pengaktifan *nested virtualization* ini dapat dilakukan melalui **command prompt Windows** yang dijalankan sebagai *administrator* (**Run as administrator**).

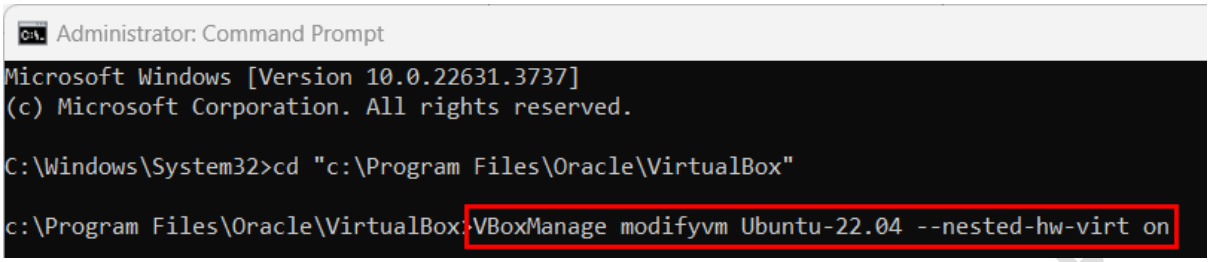
Pada kotak dialog **command prompt** yang tampil, lakukan eksekusi perintah `cd "C:\Program Files\Oracle\VirtualBox"` untuk berpindah ke lokasi direktori yang memuat instalasi *VirtualBox* yaitu di "**C:\Program Files\Oracle\VirtualBox**", seperti yang ditunjukkan oleh gambar berikut:



Silakan menyesuaikan jika lokasi direktori instalasi dari *VirtualBox* yang dimiliki berbeda.

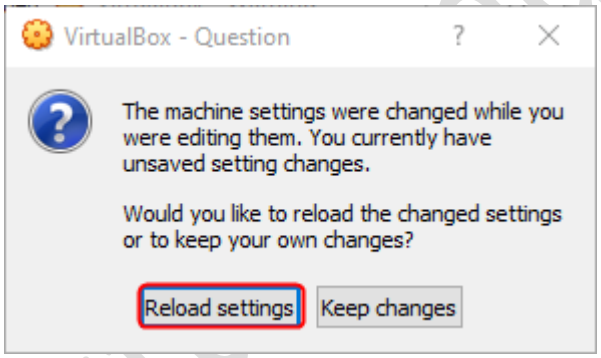
Sintak perintah untuk mengaktifkan *nested VT-X/AMD-V* adalah **VBoxManage modifyvm <virtual_machine_name> --nested-hw-virt on** dimana <virtual_machine_name> merupakan nama *virtual machine* yang akan diubah yaitu **Ubuntu-22.04**. Sehingga perintah pengaktifan virtualisasi yang dieksekusi menjadi

VBoxManage modifyvm Ubuntu-22.04 --nested-hw-virt on, seperti ditunjukkan pada gambar berikut:

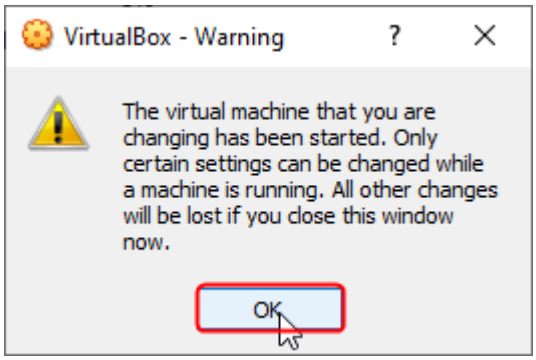


Tutup kotak dialog **Command Prompt**.

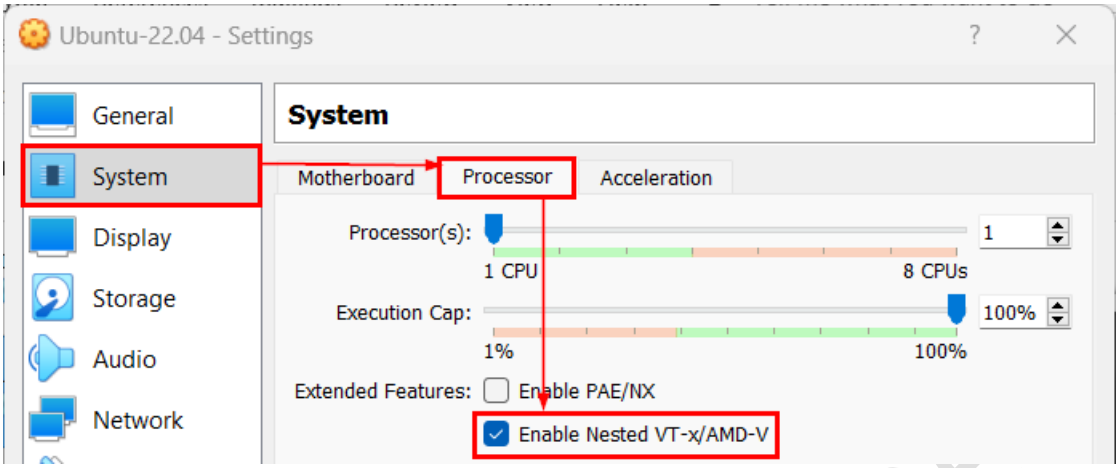
Kembali ke **Oracle VM VirtualBox Manager**. Apabila tampil kotak dialog **VirtualBox – Question** yang menginformasikan bahwa terdapat perubahan pada pengaturan *virtual machine* sebagai dampak pengaktifan *Nested Virtualization* melalui perintah yang dieksekusi pada *command prompt*, seperti terlihat pada gambar berikut:



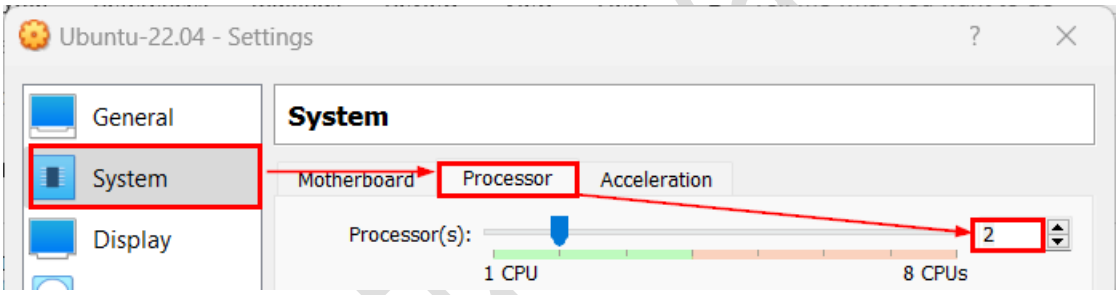
Klik tombol **Reload settings** untuk memuat ulang perubahan pengaturan yang dilakukan pada *virtual machine*. Selanjutnya tampil kotak dialog **VirtualBox – Warning** yang menginformasikan bahwa perubahan pada *virtual machine* telah dijalankan, seperti terlihat pada gambar berikut:



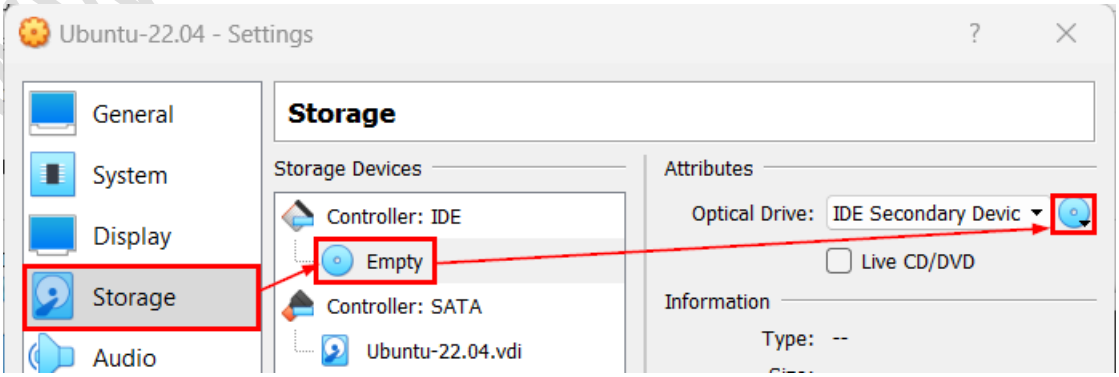
Klik tombok **OK**. Selanjutnya pada **Ubuntu-22.04 Settings** yaitu di bagian **Extended Features** dari **Processor**, pilihan **Enable Nested VT-X/AMD-V** telah tercentang (✓) yang menandakan bahwa fitur *nested virtualization* telah berhasil diaktifkan, seperti terlihat pada gambar berikut:



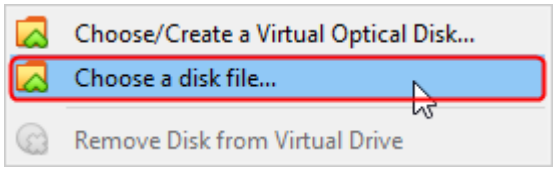
11. Mengubah jumlah **Processor** menjadi minimal **2 CPU** untuk kebutuhan lab **Kubernetes** menggunakan **Minikube**. Pada panel detail sebelah kanan dari System tab Processor, lakukan penyesuaian nilai dari parameter **Processor(s)** dari **1** menjadi **2**, seperti terlihat pada gambar berikut:



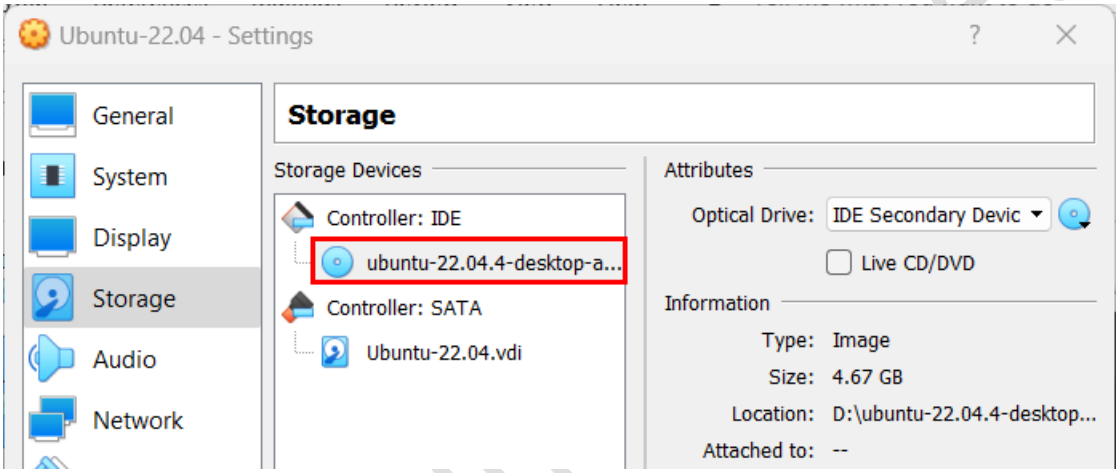
12. Mengubah pengaturan **storage** untuk mengarahkan ke lokasi penyimpanan **file ISO** dari **Ubuntu 22.04**. Pada panel sebelah kiri dari **Ubuntu-22.04 Settings**, pilih **Storage** dan pada panel detail sebelah kanan yaitu di bagian **Storage Devices,** pilih **Empty** yang ditandai dengan **icon CD.** Selanjutnya di bagian **Attributes,** klik pada **icon CD dengan tanda panah bawah** dari pilihan **Optical Drive**, seperti yang ditunjukkan pada gambar berikut:



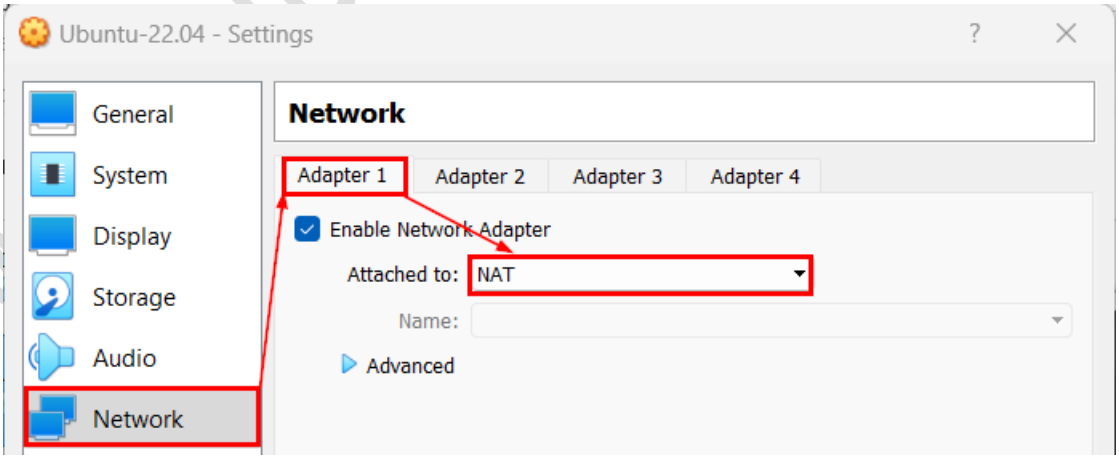
Pada *dropdown* yang tampil, pilih **Choose a disk file**…, seperti terlihat pada gambar berikut:



Arahkan ke lokasi penyimpanan **file ISO** dari **Ubuntu 22.04**, sebagai contoh di **D:\ ubuntu-22.04.4-desktop-amd64.iso**. **Silakan menyesuaikan dengan lokasi file ISO yang tersimpan di komputer masing-masing.** Pilih *file* dengan nama **ubuntu-22.04.4-desktop-amd64.iso** yang terdapat di direktori tersebut dan klik tombol **Open** sehingga hasilnya akan terlihat seperti pada gambar berikut:

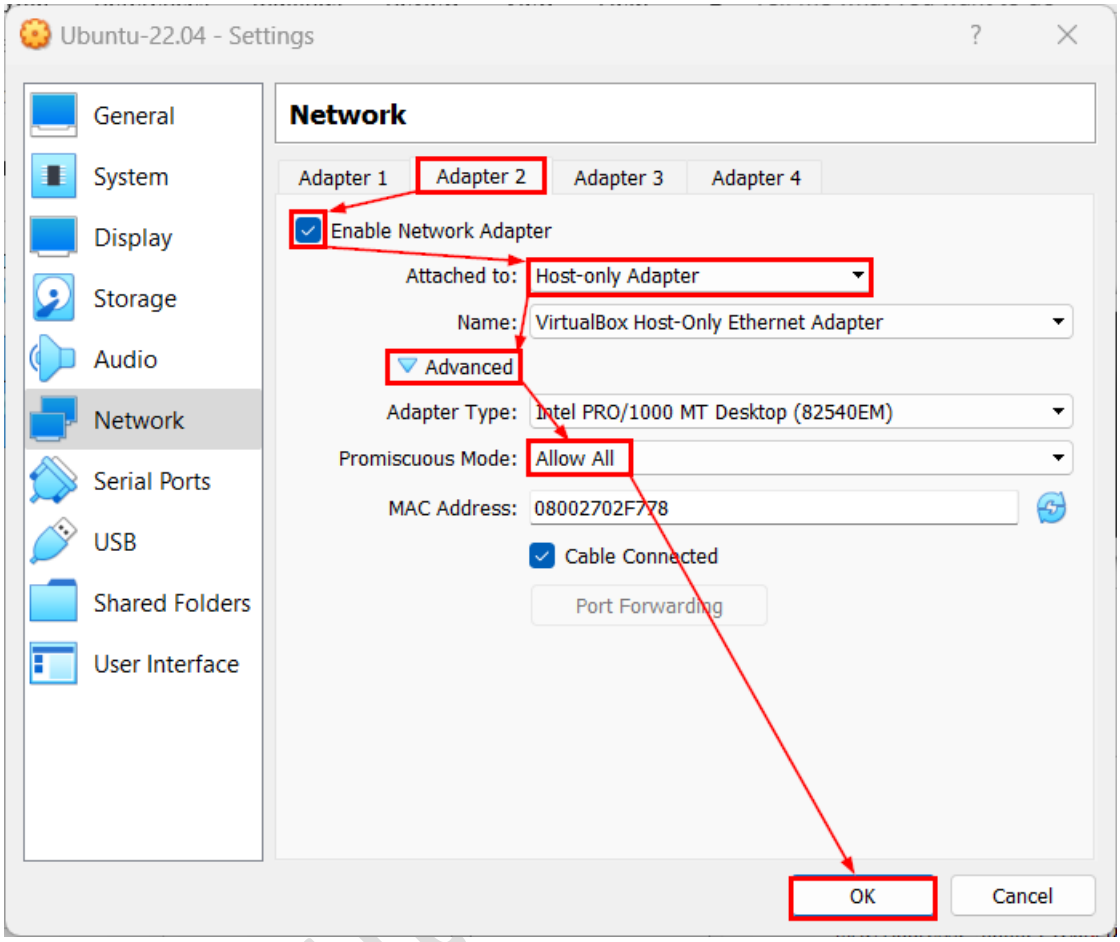


13. Menambah dan mengubah pengaturan **network adapter** terkait jenis dan **promiscuous mode**. Pada panel sebelah kiri dari **Ubuntu-22.04 Settings**, pilih **Network.** Sedangkan pada panel detail sebelah kanan yaitu di tab **Adapter 1,** pastikan pilihan jenis koneksi jaringan pada *dropdown* **Attached to** adalah **NAT**, seperti terlihat pada gambar berikut:



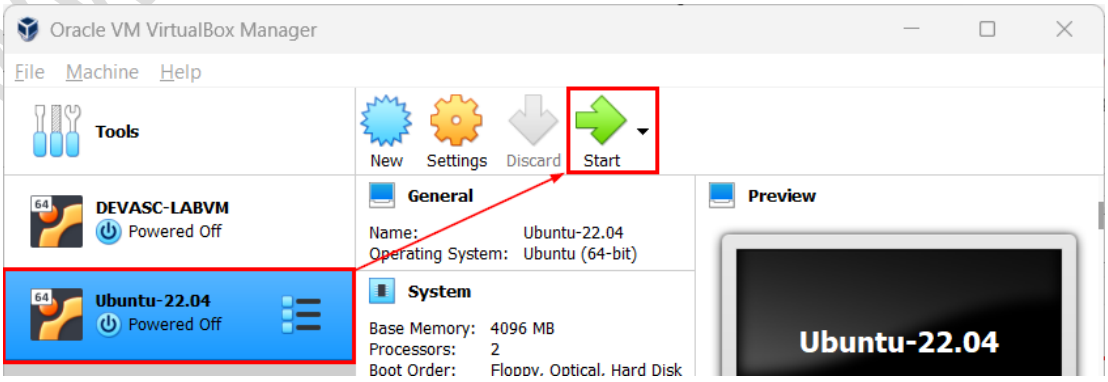
Selanjutnya pindah ke tab **Adapter 2** dan centang (√) pada **Enable Network Adapter** untuk mengaktifkan *network adapter* tersebut serta pilih **Host-only Adapter** sebagai

jenis koneksi jaringan pada *dropdown* **Attached to**, seperti terlihat pada gambar berikut:

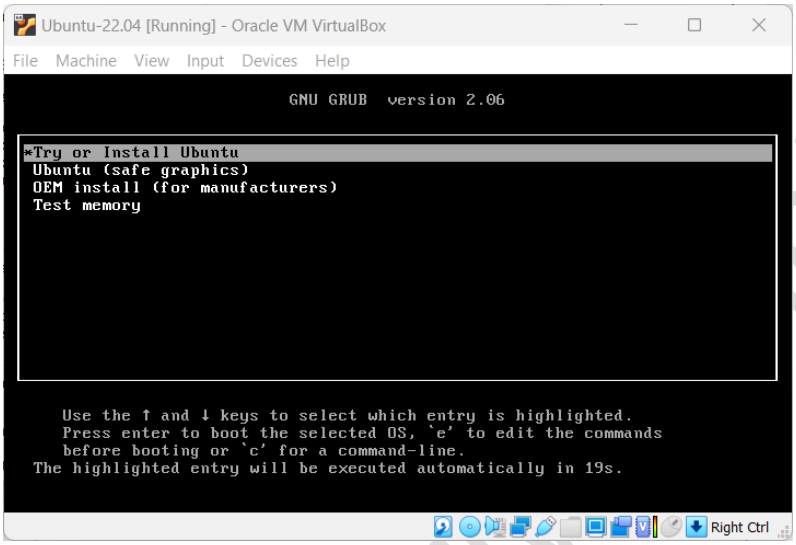


Selanjutnya klik bagian **Advanced** dan pada *dropdown* **Promiscuous Mode**, pilih **Allow All**. Klik tombol **OK** untuk menyimpan pengaturan.

14. Menjalankan **VM** dengan memilih **Ubuntu-22.04** pada daftar dari *Oracle VM VirtualBox Manager* dan memilih **Start** pada *toolbar*, seperti yang ditunjukkan pada gambar berikut:

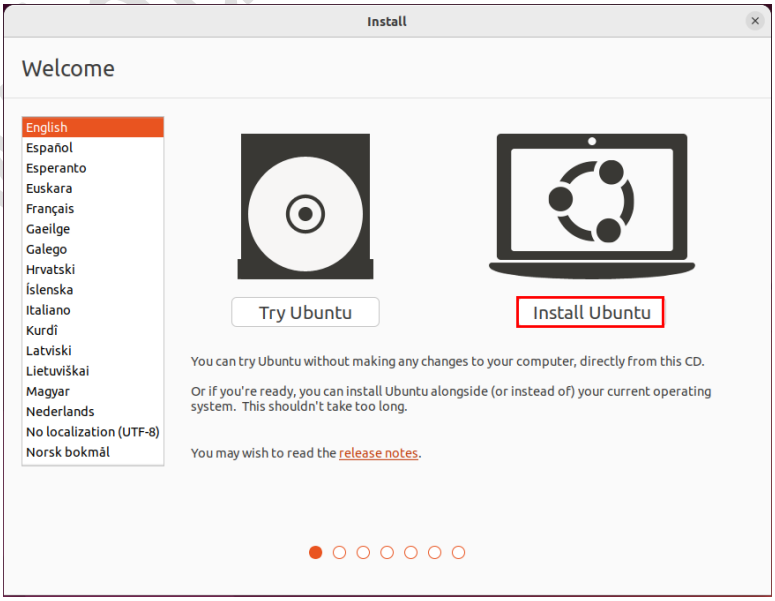


15. Tampil kotak dialog **Ubuntu-8.0 [Running] – Oracle VM VirtualBox** dan didalamnya menampilkan menu awal dengan beberapa pilihan meliputi *Try or Install Ubuntu*, *Ubuntu (safe graphics)* dan *OEM install (for manufacturers)* serta *Test memory*, seperti terlihat pada gambar berikut:



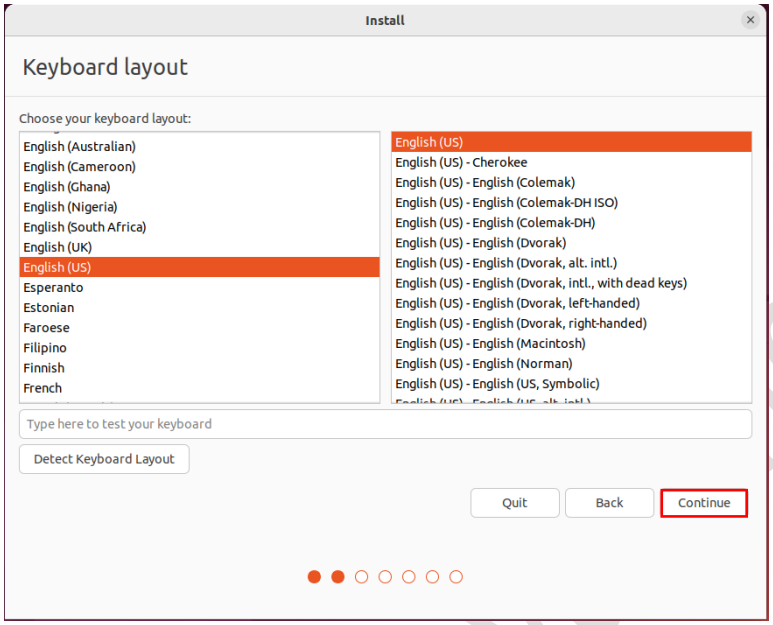
Secara *default* telah terpilih **Try or Install Ubuntu** untuk mencoba *live version* dari *Ubuntu* atau untuk menginstalasi *Ubuntu* pada *hardisk* dari *virtual machine*. Tekan tombol **Enter** untuk melanjutkan.

16. Tampil kotak dialog **Install** yang menampilkan pesan **Welcome** dengan pilihan "**Try Ubuntu**" dan "**Install Ubuntu**", seperti terlihat pada gambar berikut:



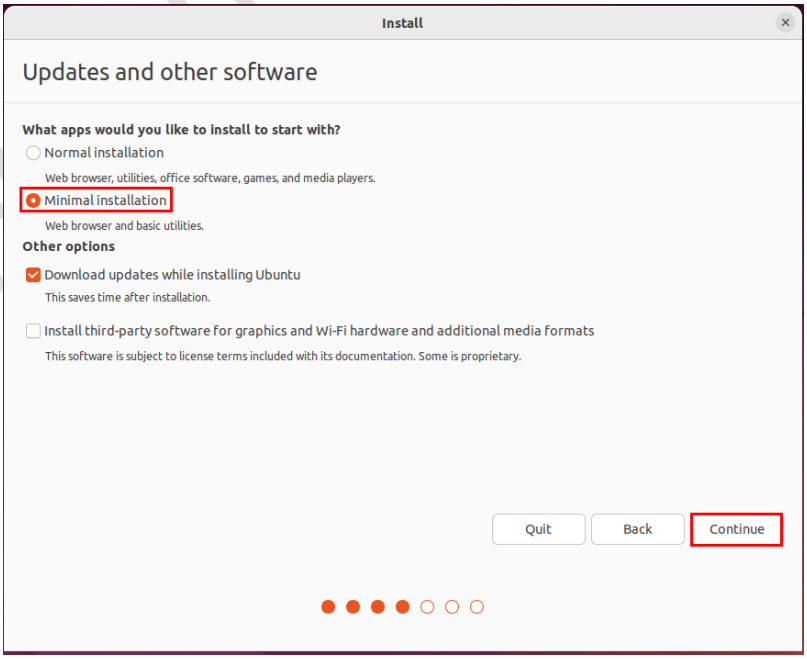
Klik tombol **Install Ubuntu** untuk menginstalasi *Ubuntu* secara permanen pada *hardisk* dari *virtual machine*.

17. Tampil kotak dialog **Install** yang menampilkan pilihan **Keyboard layout**, seperti terlihat pada gambar berikut:



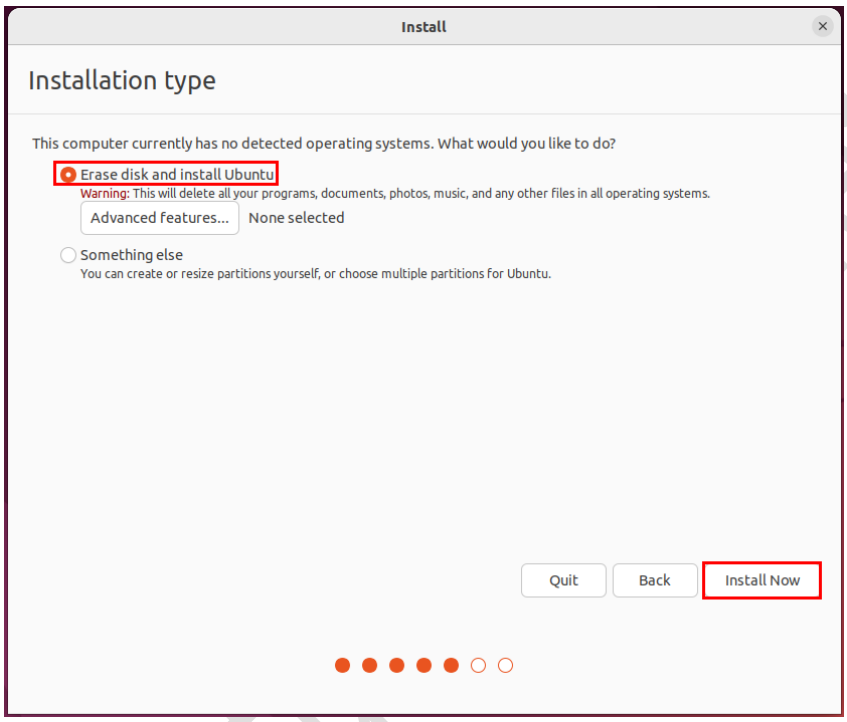
Secara *default* telah terpilih **English (US)**. Klik tombol **Continue** untuk melanjutkan instalasi.

18. Tampil kotak dialog **Install** yang menampilkan pilihan **Updates and other software** untuk memilih aplikasi-aplikasi yang ingin diinstalasi dimana terbagi menjadi dua pilihan yaitu **Normal Installation** atau **Minimal Installation**, seperti terlihat pada gambar berikut:

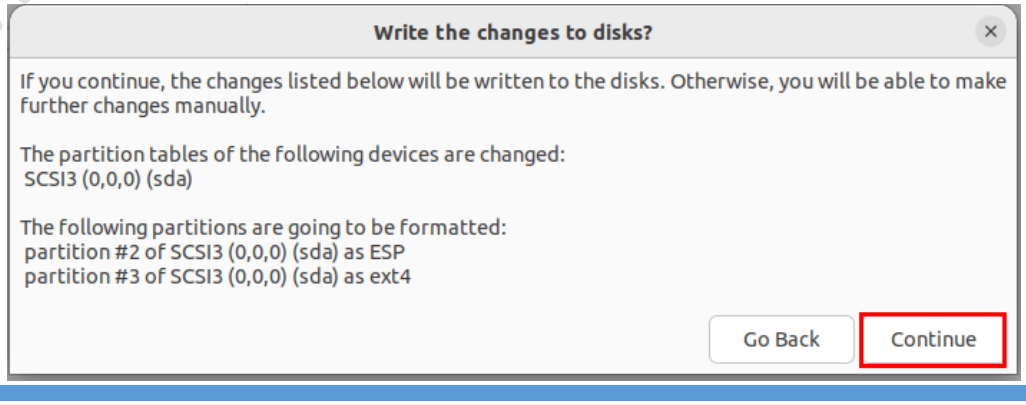


Pada parameter "**What apps would you like to install to start with?**", pilih **Normal Installation** untuk menginstalasi *Ubuntu* dengan paket minimal meliputi *web browser* dan utilitas-utilitas dasar. Klik tombol **Continue** untuk melanjutkan instalasi.

19. Tampil kotak dialog **Install** yang menampilkan pilihan **Installation type**, seperti terlihat pada gambar berikut:

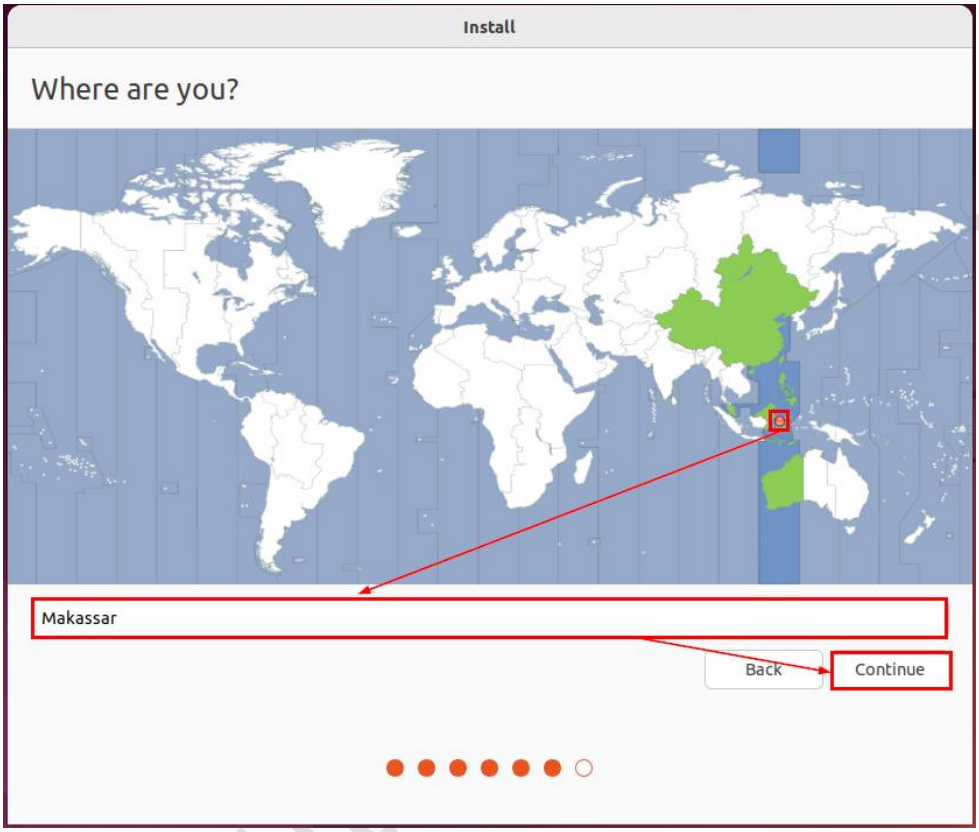


Terdeteksi pula oleh *installer Ubuntu* bahwa VM yang digunakan saat ini tidak memiliki sistem operasi. Terdapat 2 (dua) pilihan yang dapat dilakukan yaitu "**Erase disk and install Ubuntu**" yang akan menghapus seluruh program, dokumen, foto, music dan *file-file* lain di seluruh sistem operasi dan "**Something else**" untuk dapat membuat atau mengubah partisi secara mandiri atau memilih beberapa partisi untuk *Ubuntu*. Secara *default* telah terpilih "**Erase disk and install Ubuntu**". Klik tombol **Continue** untuk melanjutkan instalasi. Tampil kotak dialog "**Write the changes to disks**", seperti terlihat pada gambar berikut:



Klik tombol **Continue** untuk melanjutkan instalasi.

20. Tampil kotak dialog **Install** dengan pesan **Where are you?** yang menampilkan pilihan untuk mengatur lokasi dan secara otomatis akan mengatur *time zone* sesuai dengan lokasi terpilih, seperti terlihat pada gambar berikut:

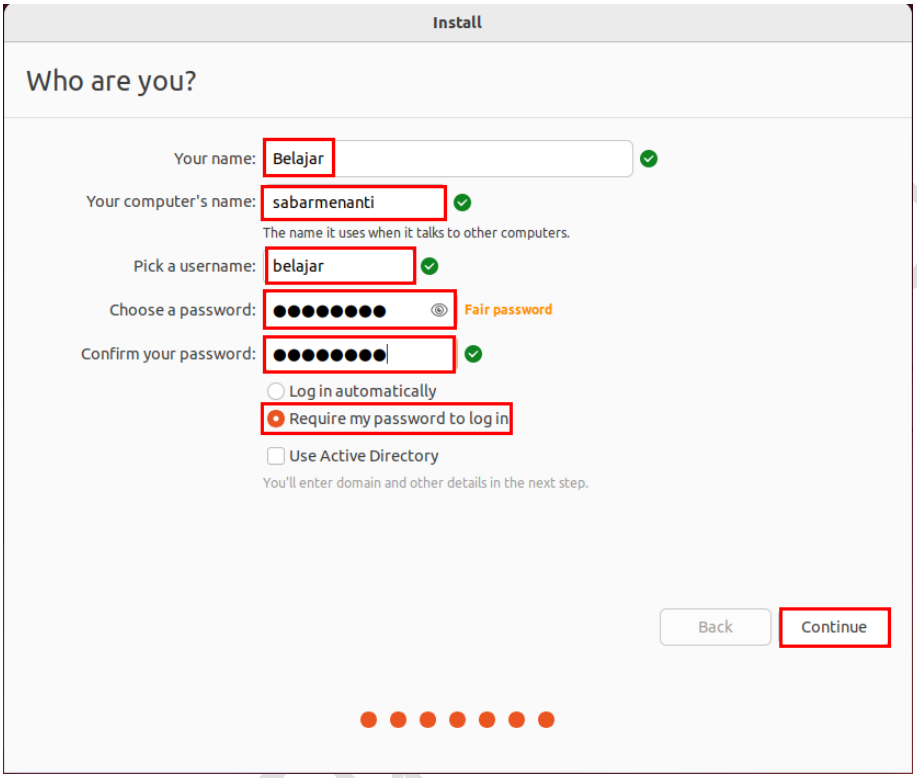


Pada peta pilih pulau **Sulawesi** sehingga *time zone* yang terpilih adalah **Makassar**. Silakan menyesuaikan apabila berada di lokasi lain. Klik tombol **Continue** untuk melanjutkan instalasi.

21. Tampil kotak dialog **Install** dengan pesan **Who are you?** yang menampilkan inputan untuk pembuatan *user* atau pengguna dan pengaturan nama komputer (*hostname*). Lengkapi isian dari masing-masing inputan parameter berikut:
    a. **Your name:**, masukkan nama lengkap. Sebagai contoh "**Belajar**".
    b. **Your computer's name:**, masukkan nama computer. Sebagai contoh "**sabarmenanti**".
    c. **Pick a username:**, masukkan nama *login* pengguna. Sebagai contoh "**belajar**".
    d. **Choose a password:**, masukkan sandi *login* pengguna. Sebagai contoh "**12345678**".
    e. **Confirm your password:** masukkan kembali sandi *login* pengguna untuk konfirmasi. Sebagai contoh "**12345678**".

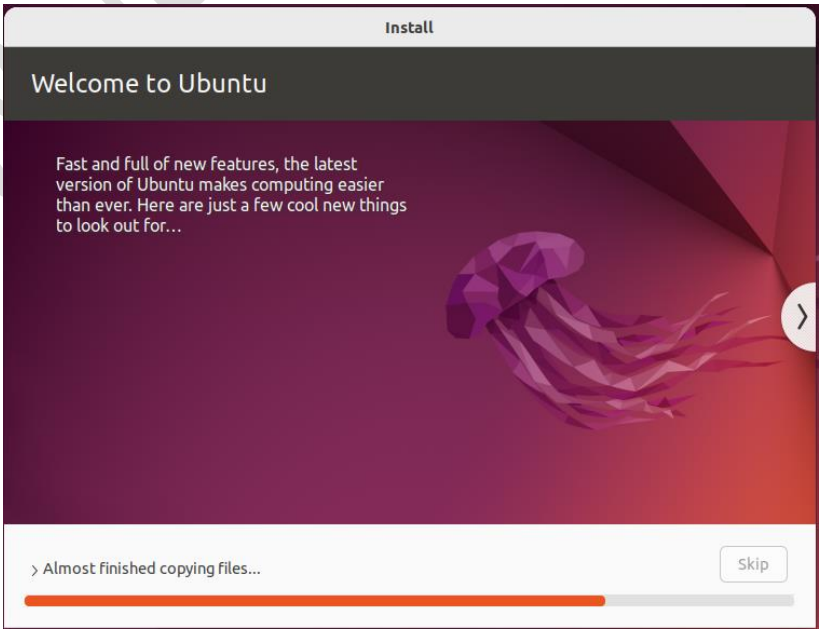
f. Secara *default* telah terpilih "**Require my password to log in**" untuk memasukkan sandi ketika *login*.

Hasil pengisian untuk keseluruhan parameter tersebut, seperti terlihat pada gambar berikut:

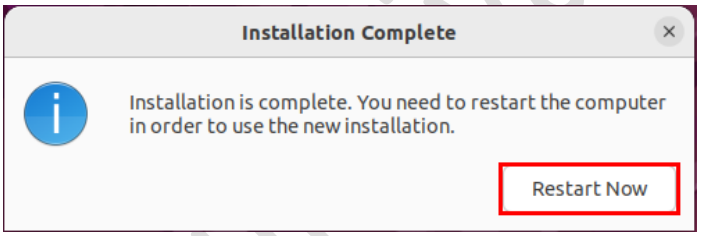


Klik tombol **Continue** untuk melanjutkan instalasi.

22. Tampil kotak dialog yang menampilkan proses instalasi, seperti terlihat pada beberapa gambar berikut:

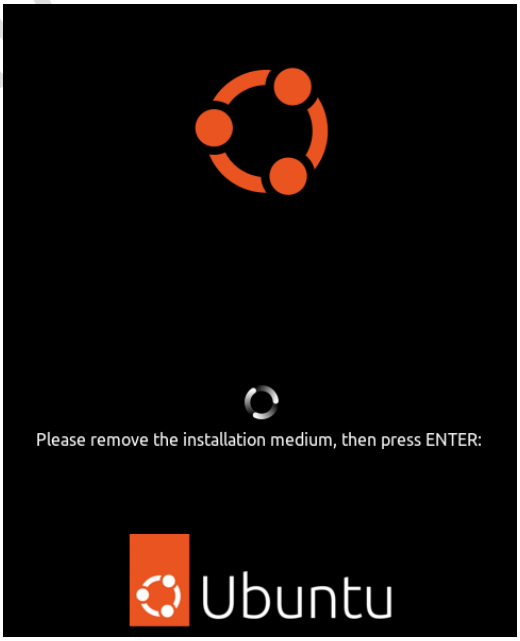


Tunggu hingga proses instalasi selesai dilakukan dan menampilkan kotak dialog **Installation Complete**, seperti terlihat pada gambar berikut:



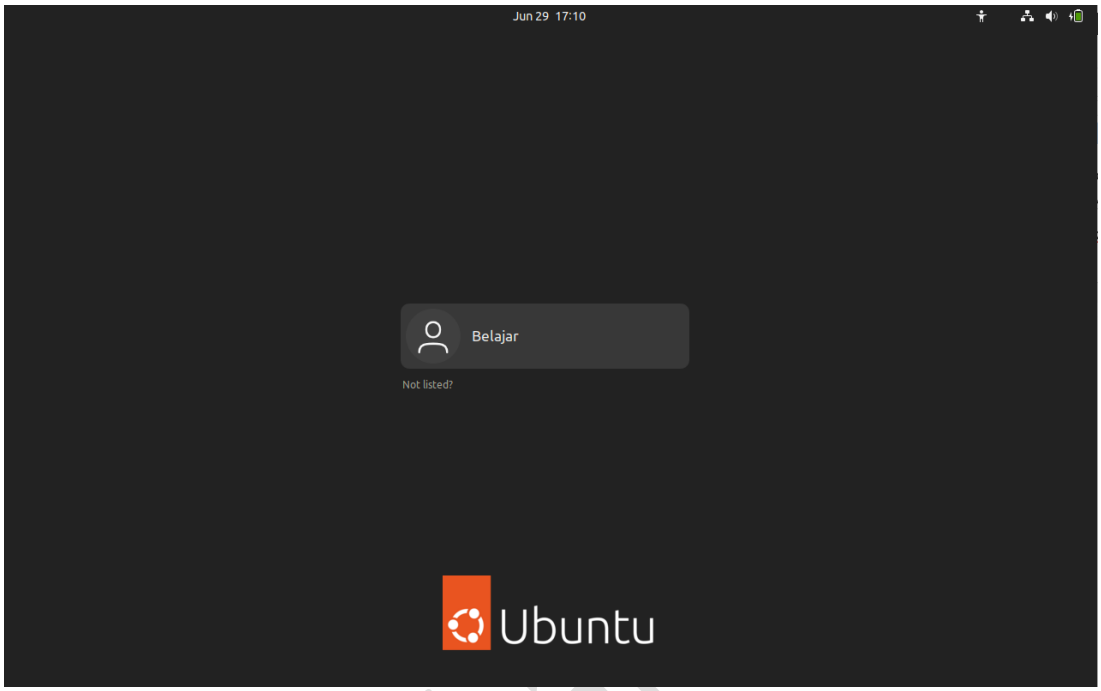
Tekan tombol **Restart Now** untuk melakukan *restart* komputer.

23. Tampil pesan "**Please remove the installation medium, then press ENTER:**", seperti terlihat pada gambar berikut:

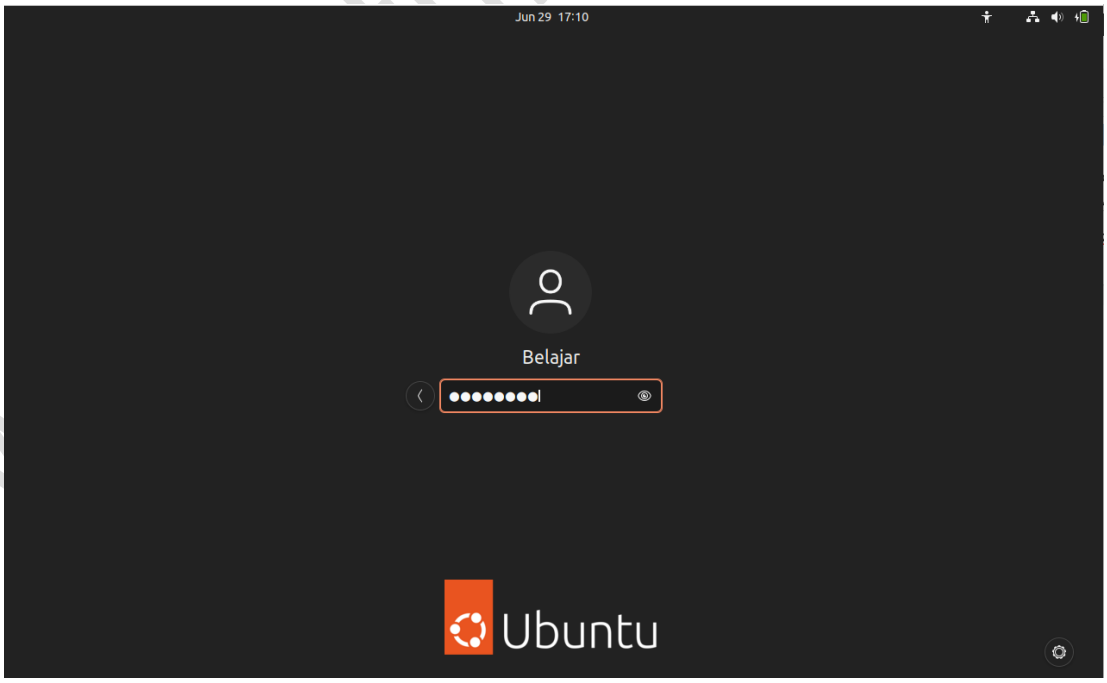


Tekan **Enter** untuk melanjutkan.

24. Tunggu hingga proses *reboot* selesai dilakukan. Setelah proses *reboot* selesai dilakukan maka akan tampak layar *login* untuk otentikasi sebelum dapat mengakses sistem *Ubuntu Desktop*, seperti terlihat pada gambar berikut:



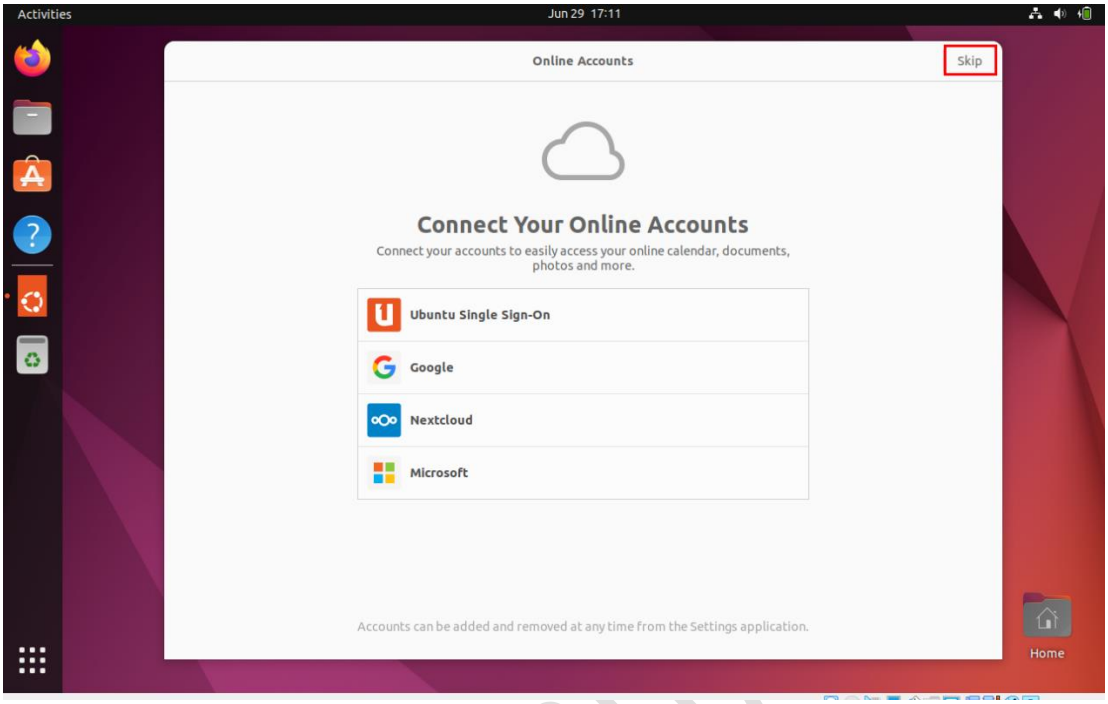
Pilih *user* **Belajar** dan tampil inputan untuk memasukkan sandi *login* dari pengguna tersebut. Masukkan "**12345678**", seperti terlihat pada gambar berikut:



Tekan **Enter** untuk melanjutkan.

25. Tampil *Desktop* dari *Ubuntu* dan kotak dialog **Online Accounts** untuk menghubungkan ke akun *online* yang dimiliki sehingga memudahkan dalam mengakses kalender,

dokumen, foto atau lainnya seperti *Ubuntu Single Sign-On*, *Google*, *Nextcloud* atau *Microsoft*, seperti terlihat pada gambar berikut:

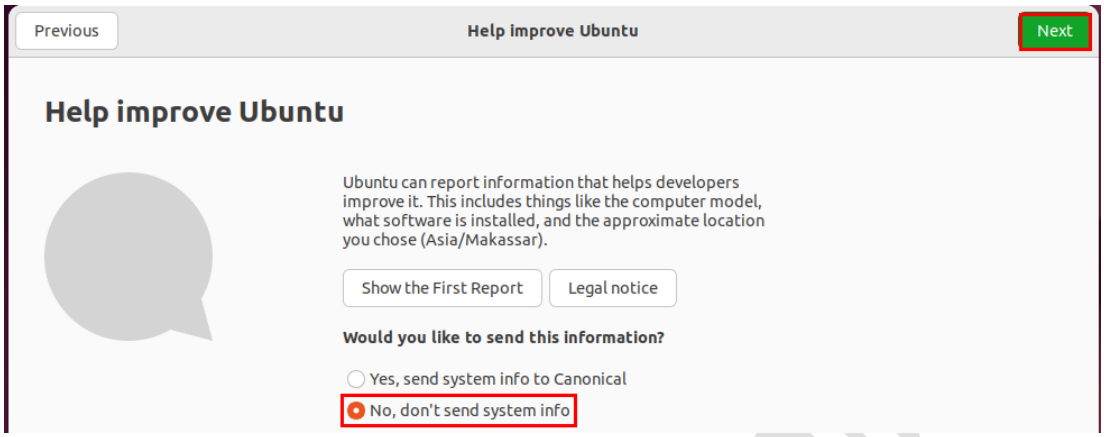


Klik tombol **Skip** untuk melanjutkan.

26. Tampil kotak dialog **Ubuntu Pro** dengan pesan "**Enable Ubuntu Pro**" yang menampilkan pilihan pengaturan untuk mengaktifkan *Ubuntu Pro*, seperti terlihat pada gambar berikut:

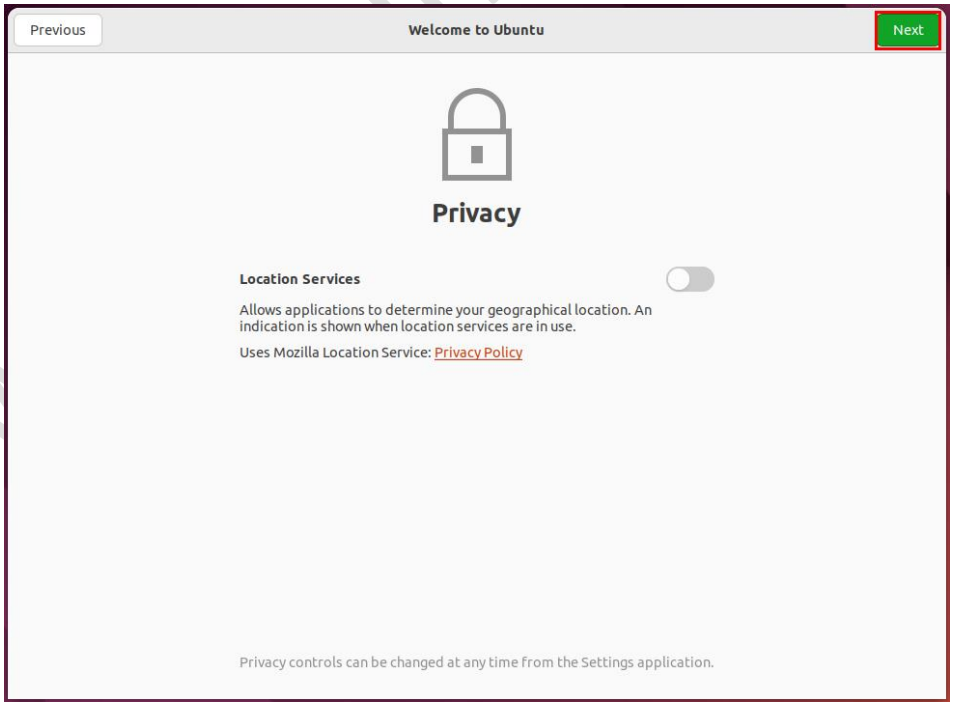
Secara *default* telah terpilih **Skip for now** pada pilihan parameter "**Enable Ubuntu Pro for this installation or skip this step**" untuk mengabaikan pengaktifan fitur tersebut saat ini. Klik tombol **Next** untuk melanjutkan.

27. Tampil kotak dialog **Help improve Ubuntu** untuk mengatur terkait pengiriman informasi dari *Ubuntu* yang diinstalasi ke *Canonical* sehingga membantu pengembang untuk memperbaikinya, seperti terlihat pada gambar berikut:



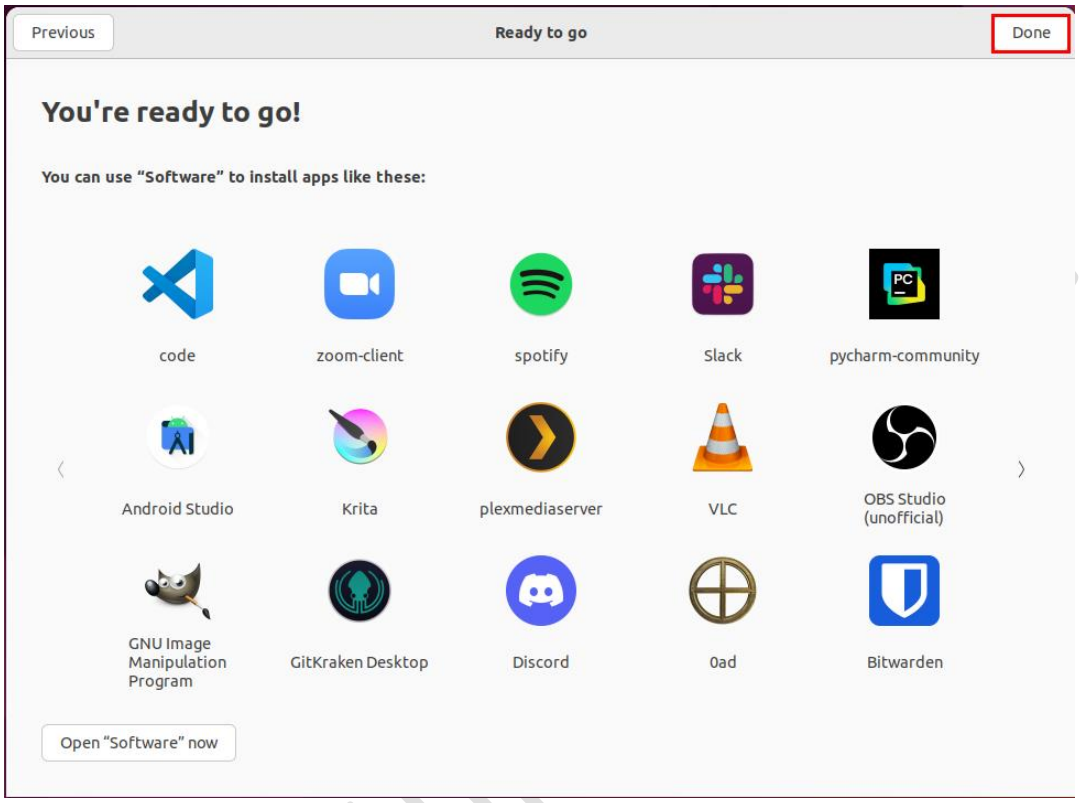
Pilih "**No, don't send system info**" pada pilihan dari parameter "**Would you like to send this information?**" agar tidak mengirimkan informasi terkait sistem Ubuntu yang dimiliki. Klik tombol **Next** untuk melanjutkan.

28. Tampil kotak dialog **Welcome to Ubuntu** dengan pesan **Privacy** yang dapat digunakan untuk mengatur *Location Services* sehingga mengijinkan aplikasi untuk menentukan lokasi geografis Anda, seperti terlihat pada gambar berikut:



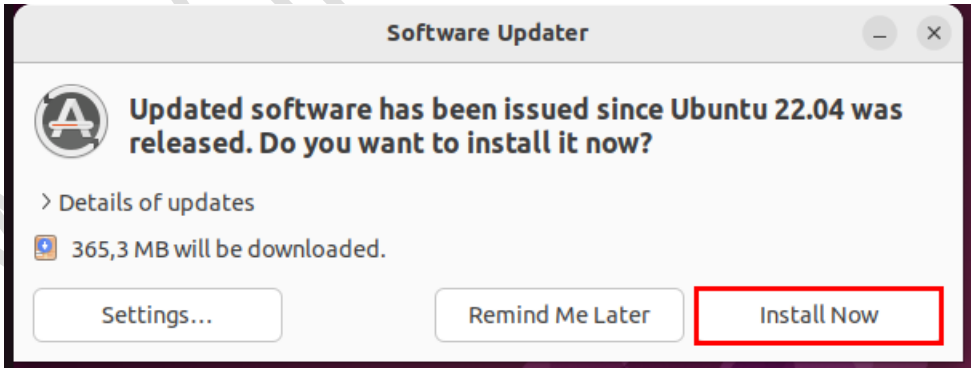
Secara *default* pengaturan *Location Services* masih tidak aktif (*disable*). Klik tombol **Next** untuk melanjutkan.

29. Tampil kotak dialog **Ready to go** yang menampilkan informasi terkait penggunaan fitur **Software** untuk menginstalasi aplikasi di **Ubuntu Desktop**, seperti terlihat pada gambar berikut:



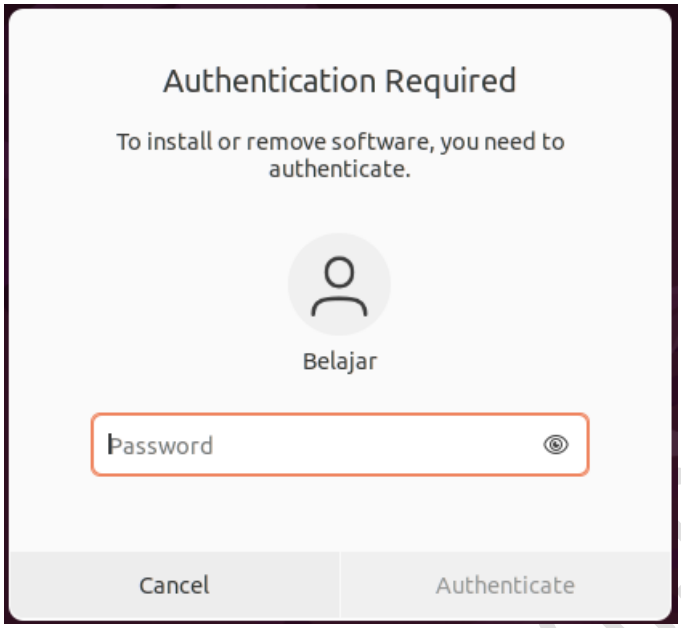
Klik tombol **Done** untuk menutup kotak dialog tersebut.

30. Tampil kotak dialog **Software Updater** yang menginformasikan bahwa telah tersedia pembaharuan perangkat lunak untuk Ubuntu 22.04, seperti terlihat pada gambar berikut:

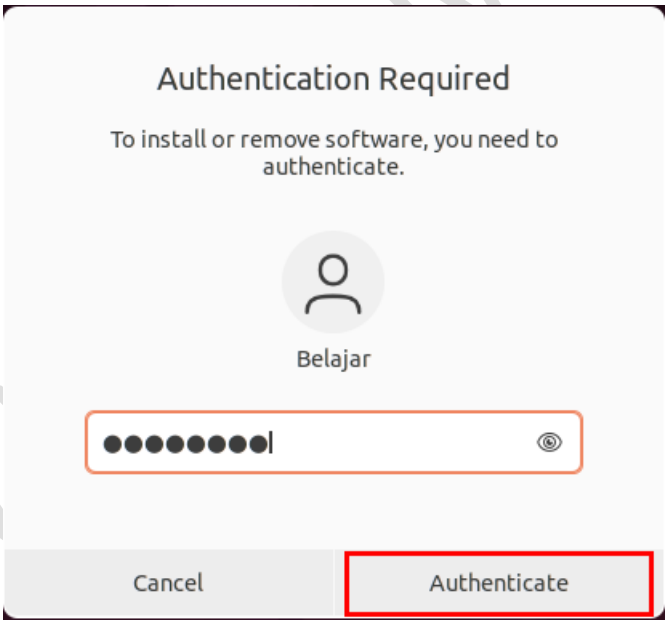


Klik tombol **Install Now** untuk menginstalasi perbaharuan perangkat lunak tersebut sekarang.

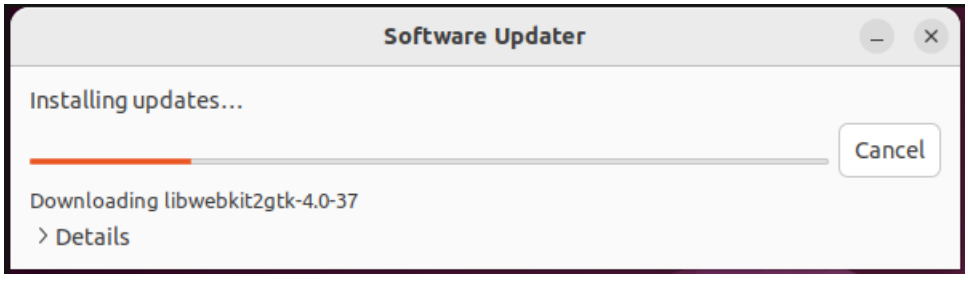
Selanjutnya akan tampil kotak dialog **Authentication Required** yang meminta inputan sandi login dari *user* **Belajar**, seperti terlihat pada gambar berikut:



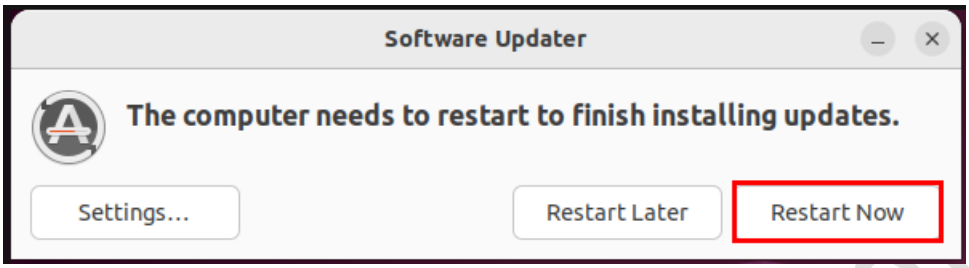
Masukkan "**12345678**" dan tekan tombol **Authenticate** untuk melanjutkan proses, seperti terlihat pada gambar berikut:



Tampil kotak dialog **Software Updater** yang memperlihatkan proses instalasi pembaharuan, seperti terlihat pada gambar berikut:

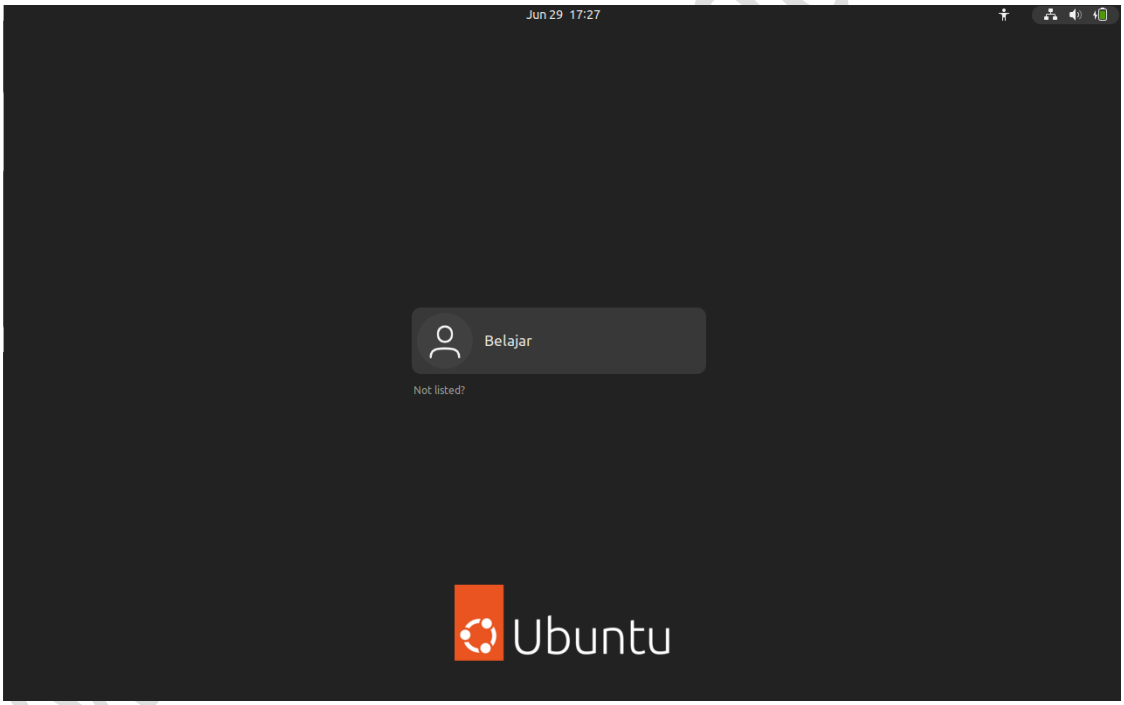


Tunggu hingga proses instalasi pembaharuan selesai dilakukan dan menampilkan pesan "**The computer needs to restart to finish installing updates**", seperti terlihat pada gambar berikut:

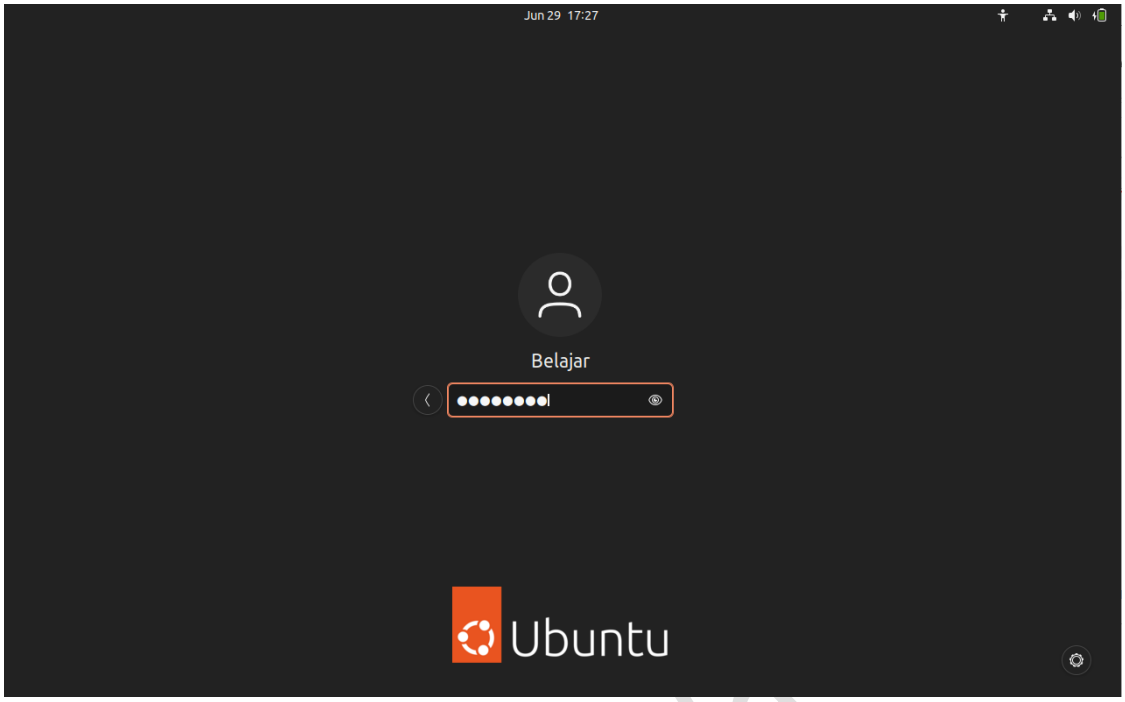


Klik tombol **Restart Now** untuk melakukan *restart* komputer sekarang. Tunggu hingga proses restart selesai dilakukan.

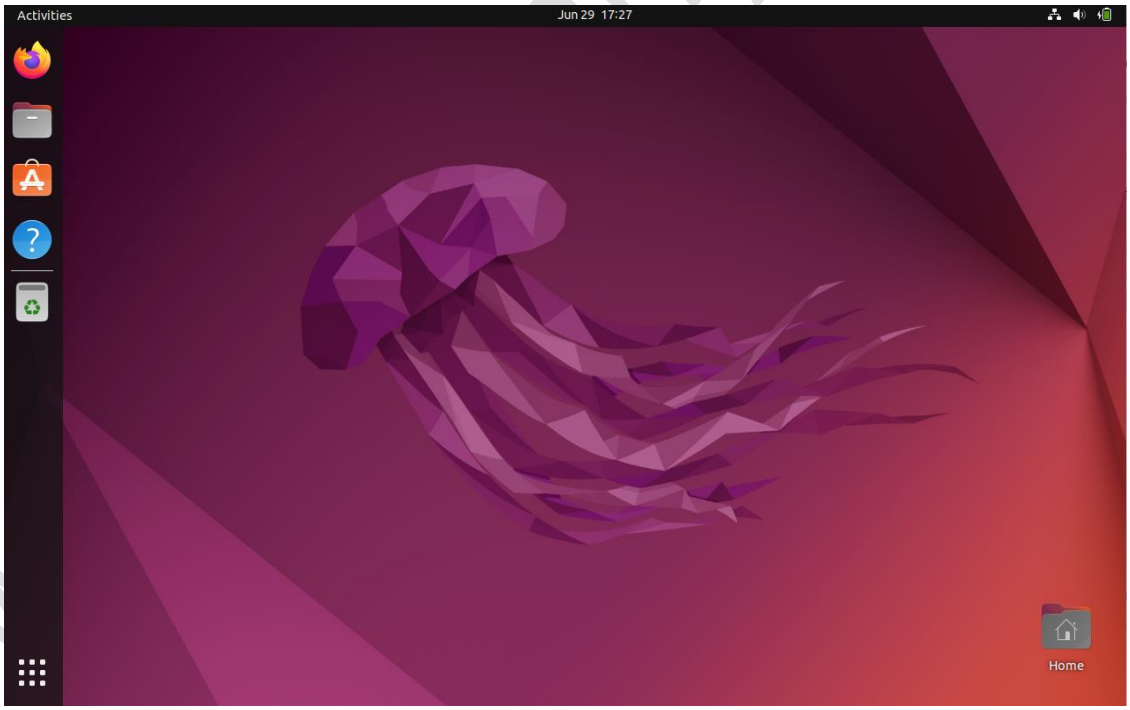
Setelah proses *restart* selesai dilakukan maka akan tampak layar *login* untuk otentikasi sebelum dapat mengakses sistem *Ubuntu Desktop*, seperti terlihat pada gambar berikut:



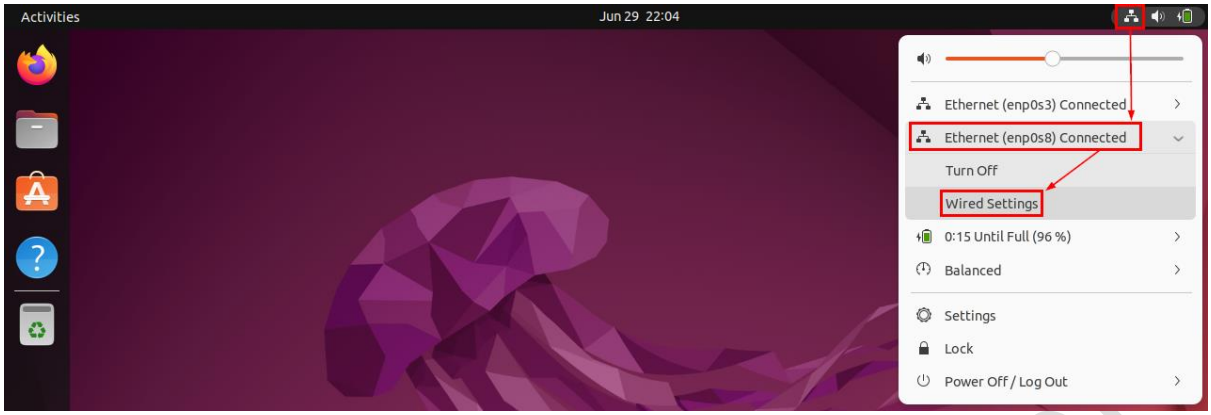
Pilih *user* **Belajar** dan tampil inputan untuk memasukkan sandi *login* dari pengguna tersebut. Masukkan "**12345678**", seperti terlihat pada gambar berikut:



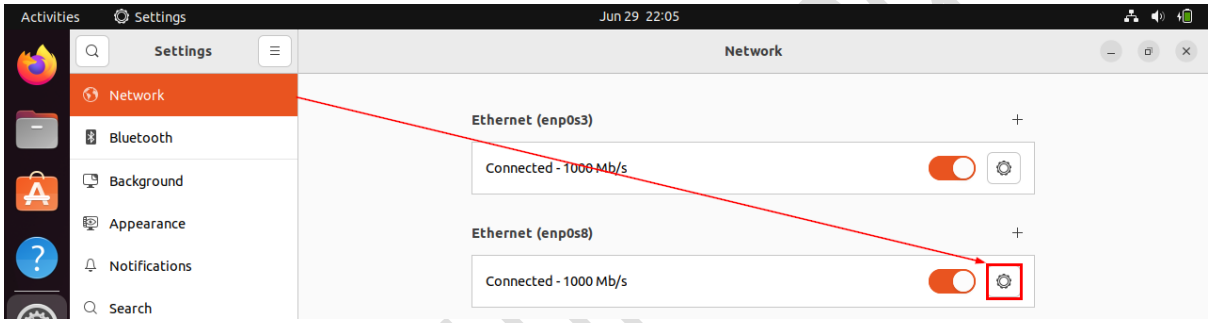
Tekan **Enter** untuk melanjutkan maka selanjutnya akan tampil *Desktop* dari *Ubuntu*, seperti terlihat pada gambar berikut:



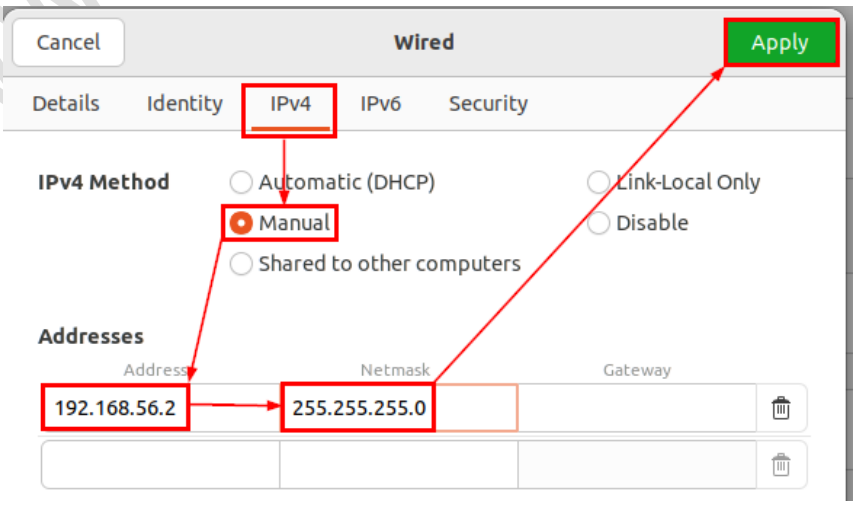
31. Mengatur pengalamatan IP secara statik untuk *network adapter* kedua yaitu **enp0s8** menggunakan **192.168.56.2/24** dengan mengakses *shortcut* dari **Networking** yang terdapat pada menu di pojok kanan atas dari *desktop Ubuntu*, seperti terlihat pada gambar berikut:



Pilih **Ethernet (enp0s8) Connected** dan **Wired Settings** maka akan tampil kotak dialog **Settings** untuk pengaturan **Network**. Selanjutnya pada panel detail sebelah kanan dari pengaturan **Network**, pilih icon **gear** dari **Ethernet (enp0s8)**, seperti terlihat pada gambar berikut:

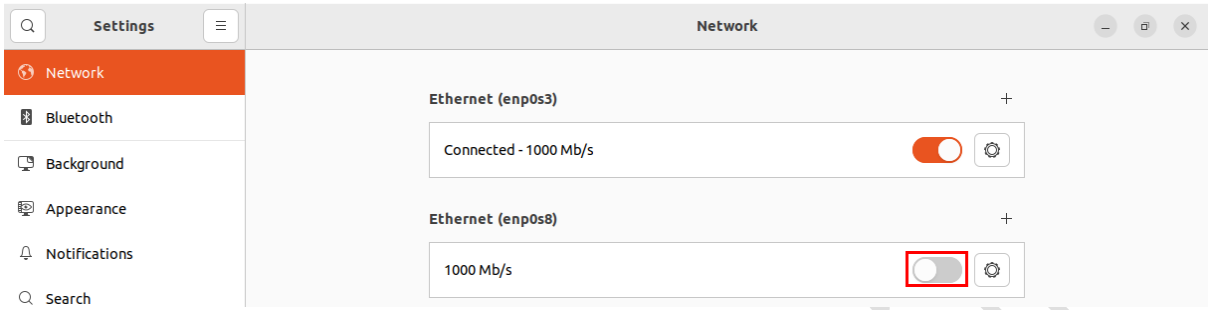


Tampil kotak dialog **Wired** dan pilih tab **IPv4**. Pada parameter **IPv4 Method**, pilih **Manual** untuk mengalokasikan pengalamatan IP secara statik. Kemudian pada parameter **Addresses**, lengkapi inputan parameter **Address** dengan **192.168.56.2**. Sedangkan pada inputan **Netmask** dengan **255.255.255.0**. Hasil akhir dari pengaturan pada parameter-parameter tersebut, seperti terlihat pada gambar berikut:

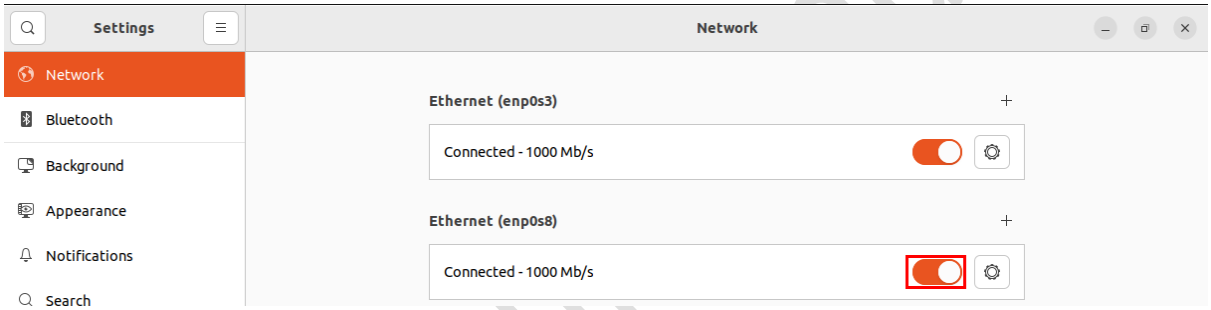


Klik tombol **Apply** untuk menyimpan pengaturan yang dilakukan.

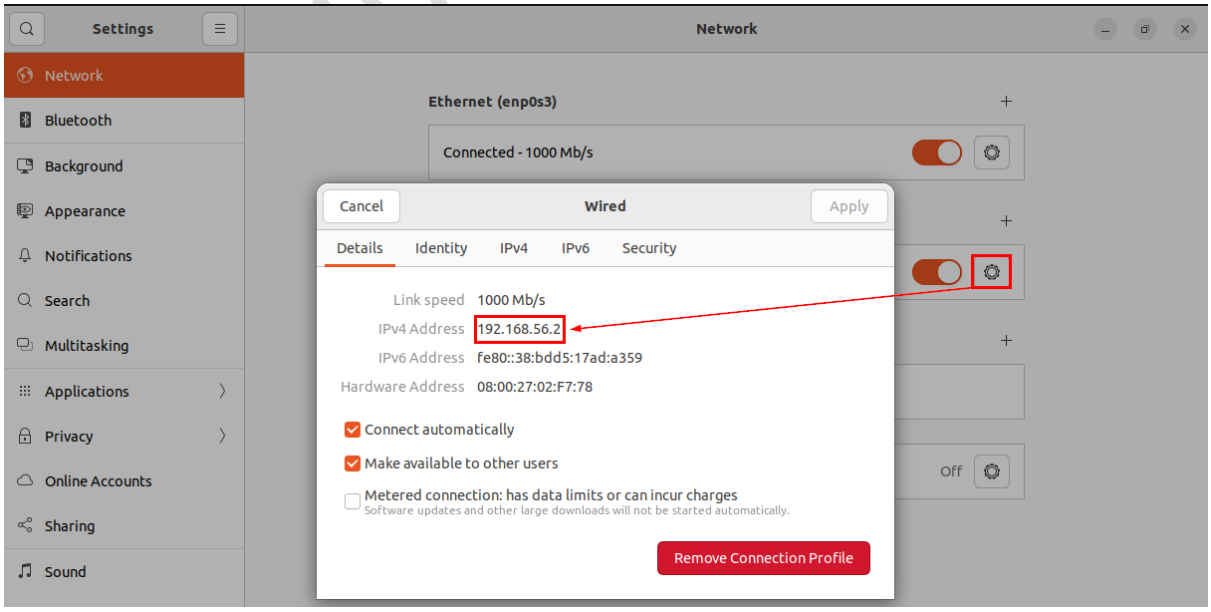
32. Mengaktifkan perubahan pengalamatan IP pada *interface Ethernet enp0s8* dengan cara melakukan **disable** dan **enable** *interface* tersebut. Geser *slider* dari **Ethernet (enp0s8)** ke kiri untuk menonaktifkan (*disable*) *interface* tersebut, seperti terlihat pada gambar berikut:



Selanjutnya geser kembali *slider* dari **Ethernet (enp0s8)** ke kanan untuk mengaktifkan (*enable*) *interface* tersebut, seperti terlihat pada gambar berikut:



33. Memverifikasi hasil penyesuaian pengalamatan IP pada *interface Ethernet enp0s8*. Pilih icon **gear** dari **Ethernet (enp0s8)** maka akan tampil kotak dialog **Wired**, seperti terlihat pada gambar berikut:



Terlihat *interface enp0s8* telah menggunakan alamat IP **192.168.56.2**.

Klik tombol **Cancel** untuk menutup kotak dialog **Wired**. Selanjutnya tutup kotak dialog **Settings**.

Selamat Anda telah berhasil menginstalasi *Ubuntu Desktop* versi 22.04 LTS.

BAB II

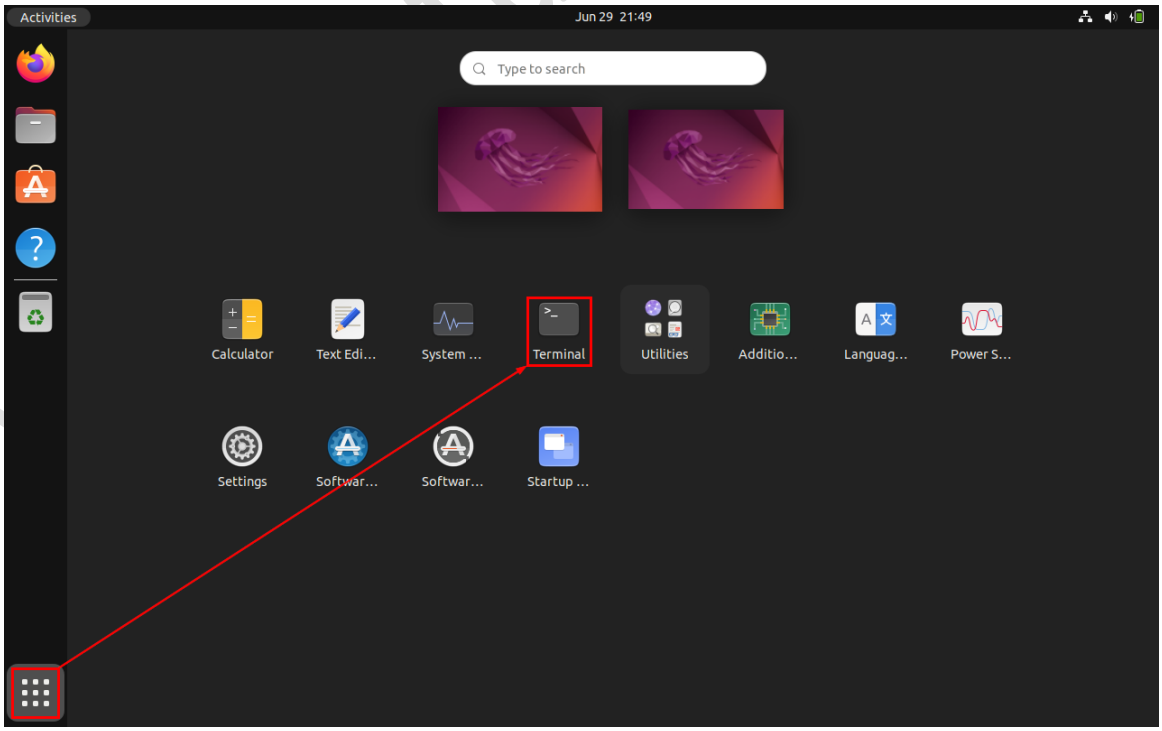
INSTALASI DAN KONFIGURASI SERVER SECURE SHELL (SSH) PADA UBUNTU DESKTOP VERSI 22.04 LTS

Setelah berhasil melakukan "**Instalasi dan Konfigurasi Ubuntu Desktop Versi 22.04 LTS**" di bab sebelumnya maka selanjutnya akan dilakukan instalasi dan konfigurasi *server Secure Shell (SSH)* menggunakan **OpenSSH**. Hal ini diperlukan agar *Ubuntu Desktop* dapat diakses secara jarak jauh (*remote*) melalui SSH menggunakan aplikasi SSH *Client*. Sebagai contoh menggunakan aplikasi SSH *Client* **Putty** dari *host machine Windows* 11.

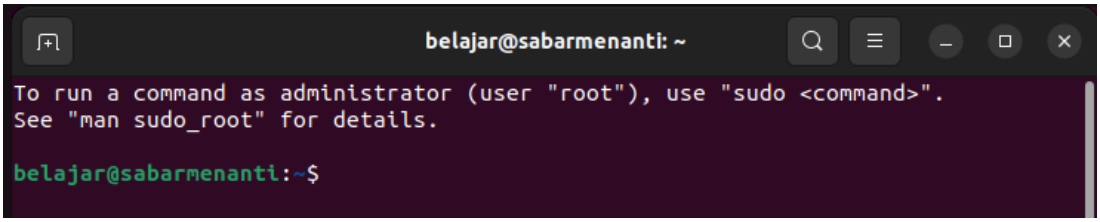
A. Instalasi dan Konfigurasi Server SSH

Adapun langkah-langkah instalasi dan konfigurasi *server SSH* pada *Ubuntu Desktop* versi 22.04 LTS adalah sebagai berikut:

1. Mengakses **terminal** dari *Ubuntu* dengan memilih tombol **Show Applications** yang terdapat di pojok kiri bawah dari *desktop*, seperti terlihat pada gambar berikut:



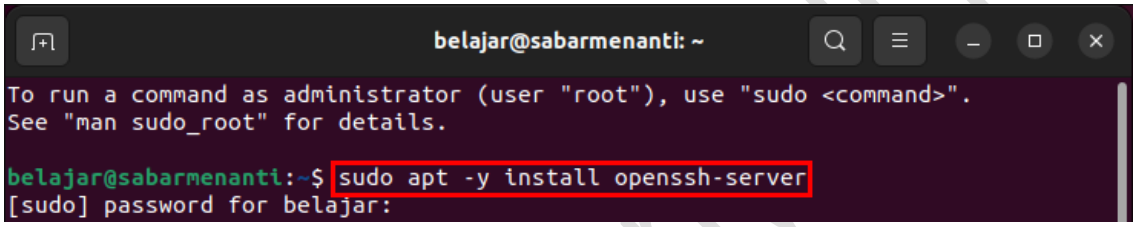
Pilih **Terminal** pada pilihan aplikasi yang tampil. Selanjutnya akan tampil kotak dialog **Terminal**, seperti terlihat pada gambar berikut:



2. Menginstasi paket aplikasi **OpenSSH** dengan mengeksekusi perintah:

```
$ sudo apt -y install openssh-server
```

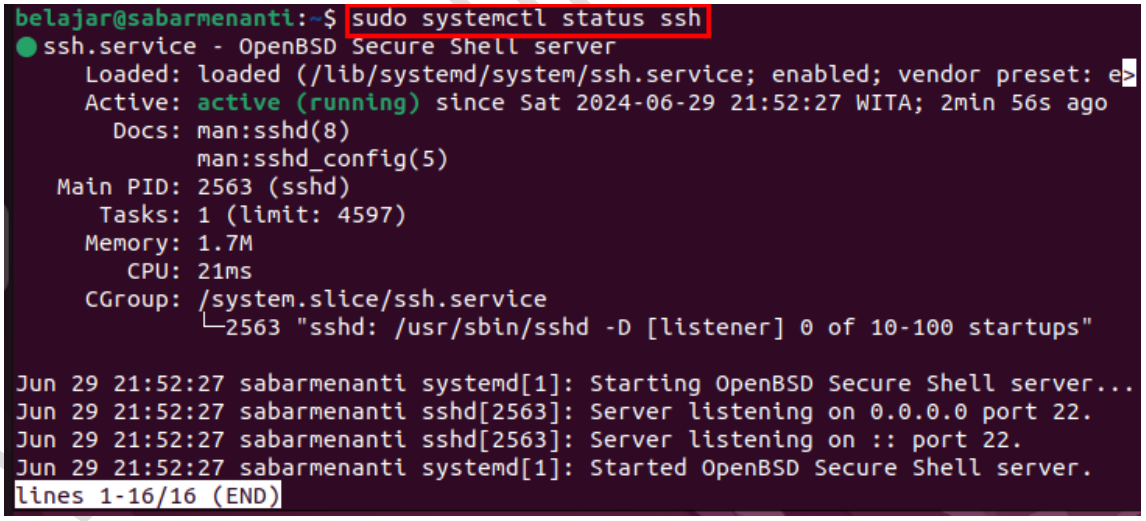
Tekan **Enter** maka akan tampil inputan yang meminta pengguna untuk memasukkan sandi *login* dari *user* **belajar** yaitu "**12345678**", seperti terlihat pada gambar berikut:



Tekan **Enter** untuk melanjutkan. Tunggu hingga proses instalasi paket selesai dilakukan.

3. Memverifikasi status *service* SSH.

```
$ sudo systemctl status ssh
```



Terlihat *service* SSH telah aktif atau berjalan yaitu ditandai dengan pesan **active (running)** berwarna hijau pada parameter **Active**. Tekan tombol **q** untuk keluar dari *output* eksekusi perintah tersebut sehingga memunculkan kembali *prompt* dari terminal.

4. Memverifikasi *service* SSH dijalankan secara otomatis ketika *booting Ubuntu*.

```
$ sudo systemctl is-enabled ssh
```

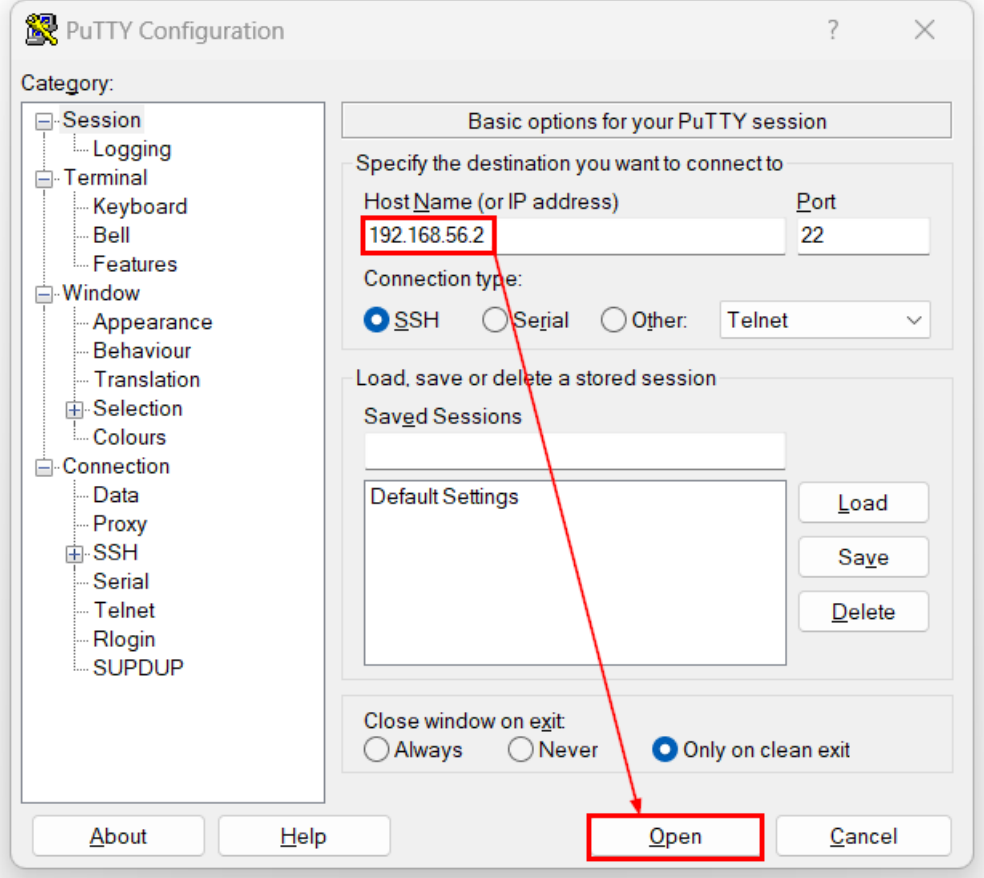
```
belajar@sabarmenanti:~$ sudo systemctl is-enabled ssh
enabled
```

Terlihat *output* **enabled** yang menandakan bahwa *service* SSH akan dijalankan secara otomatis ketika *booting* sistem *Ubuntu*.

B. Verifikasi Koneksi SSH dari Windows 11

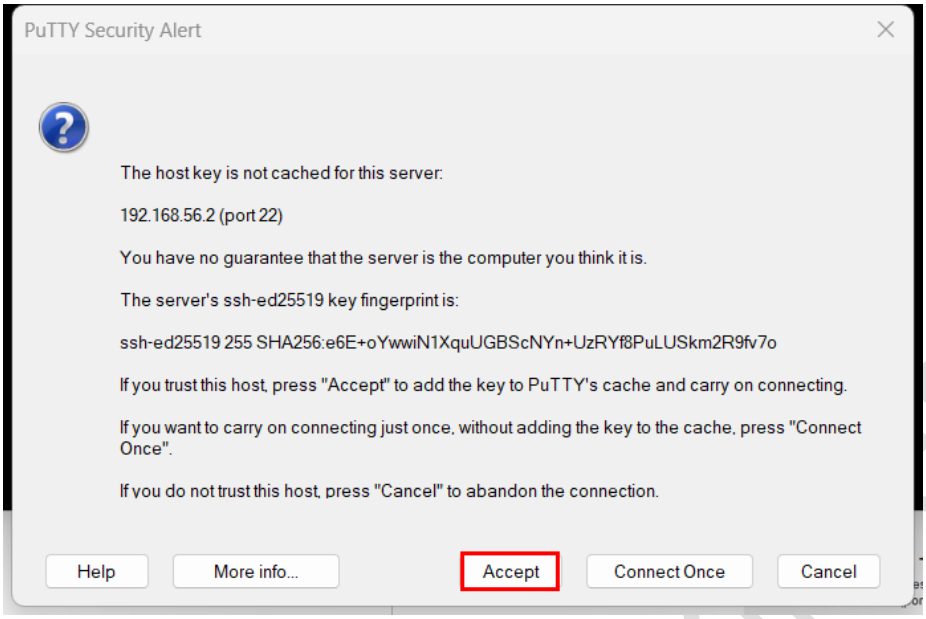
Adapun langkah-langkah memverifikasi koneksi SSH dari *Windows* 11 adalah sebagai berikut:

1. Jalankan aplikasi *SSH Client*, sebagai contoh menggunakan aplikasi **Putty**. Tampil kotak dialog *Putty Configuration*. Pada isian **Host Name (or IP Address)**, masukkan alamat IP dari *Ubuntu Desktop* yaitu **192.168.56.2**, seperti terlihat pada gambar berikut:



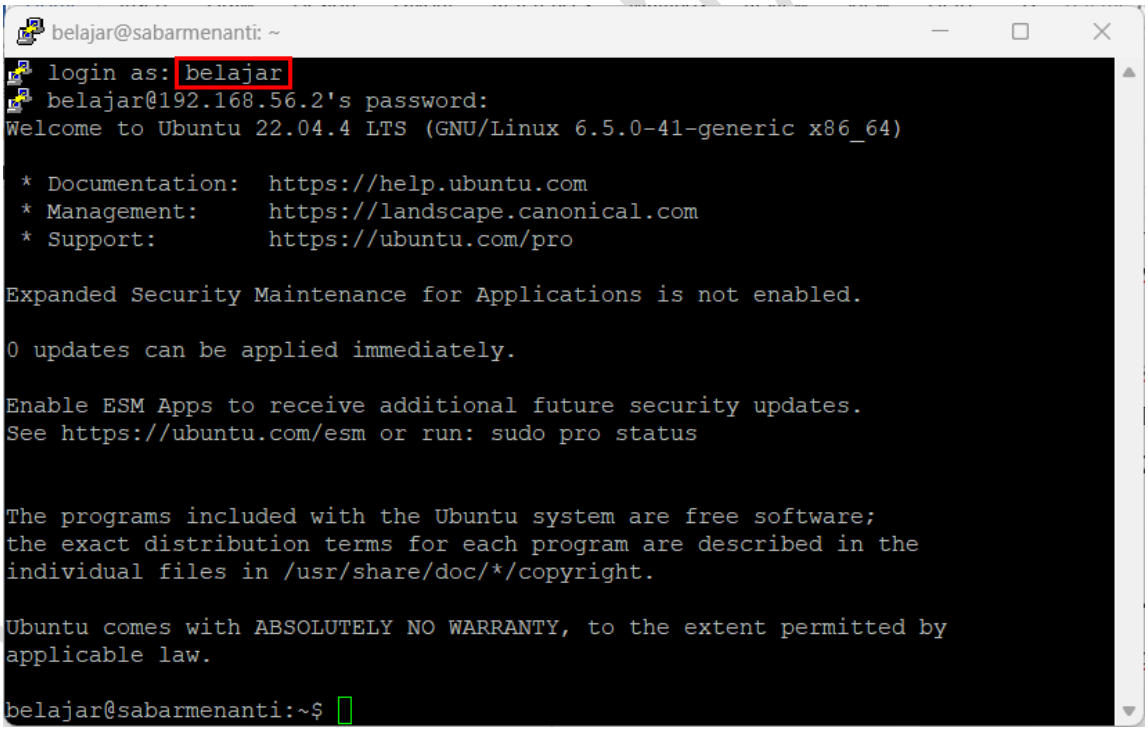
Klik tombol **Open**.

2. Tampil kotak dialog **Putty Security Alert** yang menampilkan pesan peringatan terkait potensi pelanggaran keamanan, seperti terlihat pada gambar berikut:



Klik tombol **Accept** untuk melanjutkan.

3. Tampil kotak dialog *Putty* yang meminta pengguna untuk melakukan proses otentikasi login ke *Ubuntu Desktop*, seperti terlihat pada gambar berikut:



Pada inputan **login as:,** masukkan "**belajar**" dan tekan tombol **Enter**. Selanjutnya tampil inputan **password:**, masukkan "**12345678**" dan tekan tombol **Enter**. Apabila proses otentikasi berhasil dilakukan maka akan tampil *shell prompt* **$**.

4. Keluar dari akses SSH.

```
$ exit
```

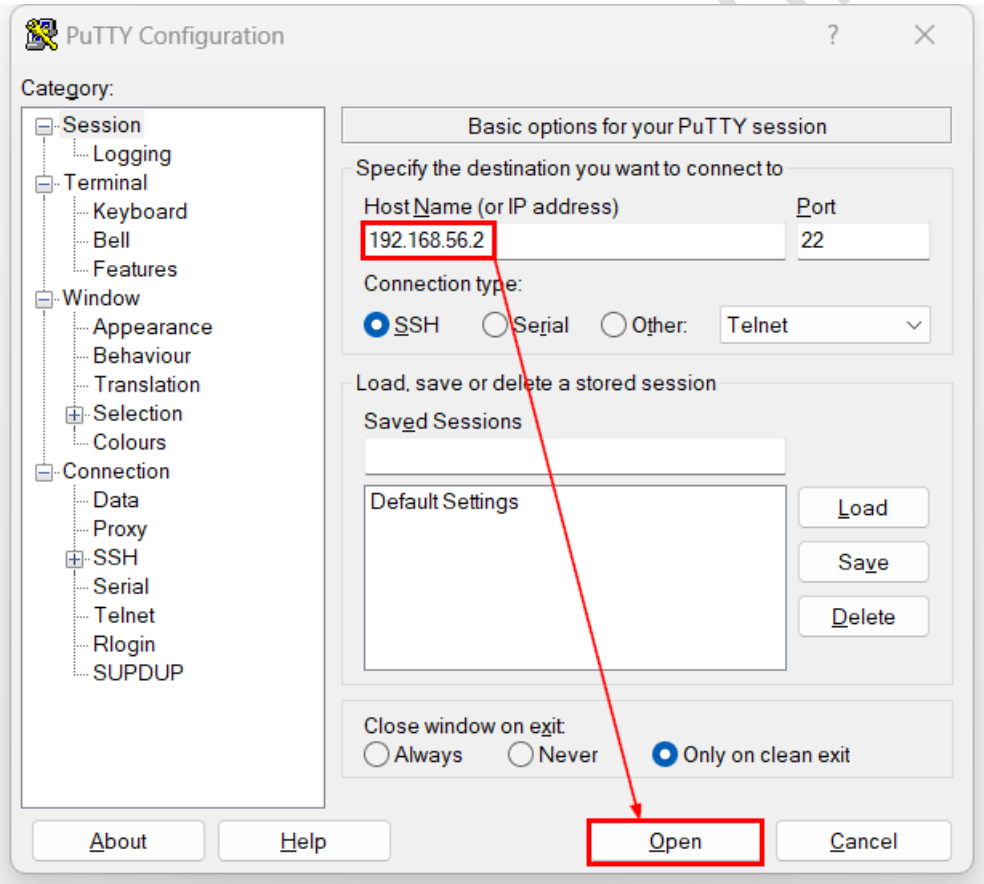
BAB III

INSTALASI DAN KONFIGURASI DOCKER ENGINE

PADA UBUNTU DESKTOP VERSI 22.04 LTS

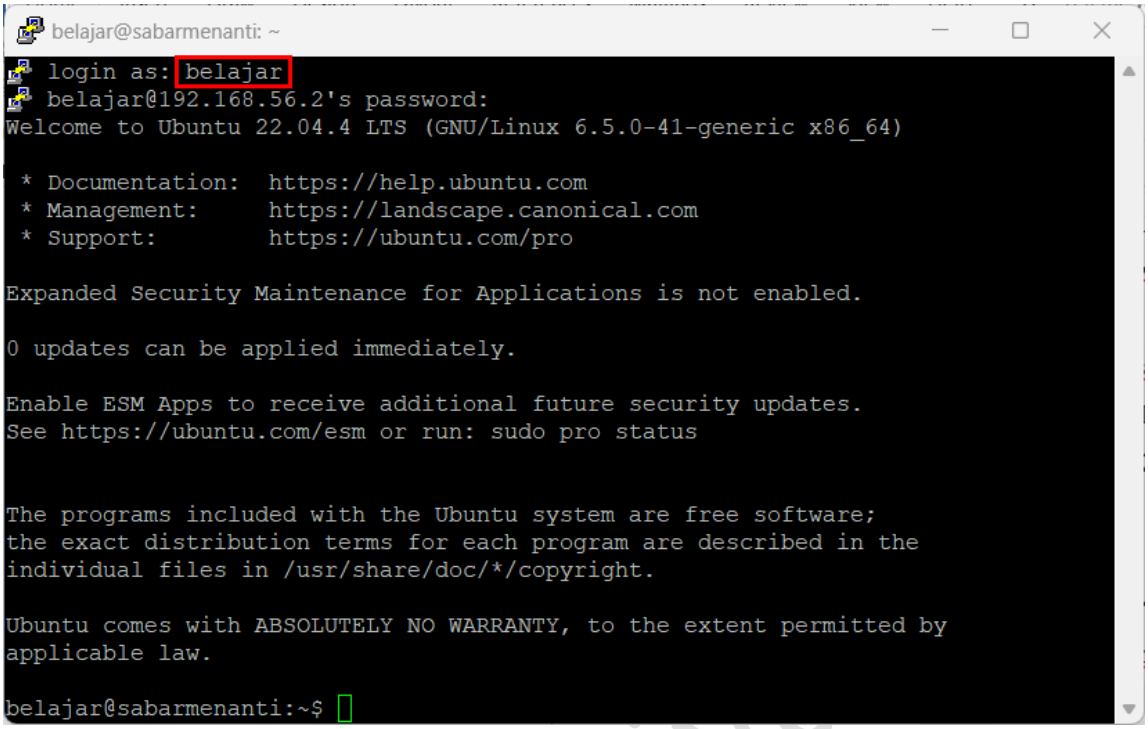
Adapun langkah-langkah menginstalasi dan mengkonfigurasi **Docker Engine** pada *Ubuntu Desktop* versi 22.04 LTS menggunakan **Docker apt repository** adalah sebagai berikut:

1. Jalankan aplikasi *SSH Client*, sebagai contoh menggunakan aplikasi **Putty**. Tampil kotak dialog *Putty Configuration*. Pada isian **Host Name (or IP Address)**, masukkan alamat IP dari *Ubuntu Desktop* yaitu **192.168.56.2**, seperti terlihat pada gambar berikut:



   Klik tombol **Open**.

3. Tampil kotak dialog *Putty* yang meminta pengguna untuk melakukan proses otentikasi login ke *Ubuntu Desktop*, seperti terlihat pada gambar berikut:

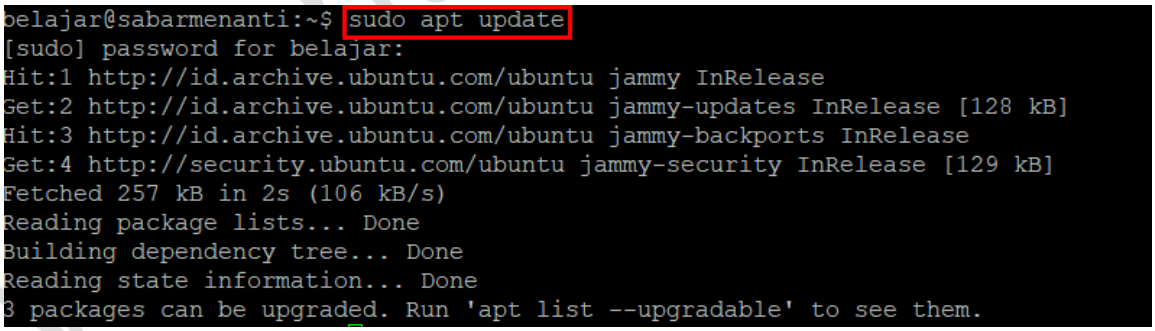


Pada inputan **login as:,** masukkan "**belajar**" dan tekan tombol **Enter**. Selanjutnya tampil inputan **password:**, masukkan "**12345678**" dan tekan tombol **Enter**. Apabila proses otentikasi berhasil dilakukan maka akan tampil *shell prompt* **$**.

4. Memperbaharui *package index* dari **apt**.

```
$ sudo apt update
```

Apabila diminta menginputkan sandi *login user* **belajar** maka masukkan "**12345678**" dan tekan **Enter**, seperti terlihat pada gambar berikut:



Tunggu hingga proses pembaharuan selesai dilakukan.

5. Menginstalasi paket untuk mengijinkan **apt** menggunakan *repository* melalui HTTPS.

```
$ sudo apt-get install ca-certificates curl
```

6. Menambahkan *Docker's Official GPG key* dengan mengeksekusi beberapa perintah berikut:

sudo install -m 0755 -d /etc/apt/keyrings

sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc

sudo chmod a+r /etc/apt/keyrings/docker.asc

7. Menambahkan *repository* ke **apt source**.

```
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] https://download.docker.com/linux/ubuntu \
  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

8. Memperbaharui *package index* dari **apt**.

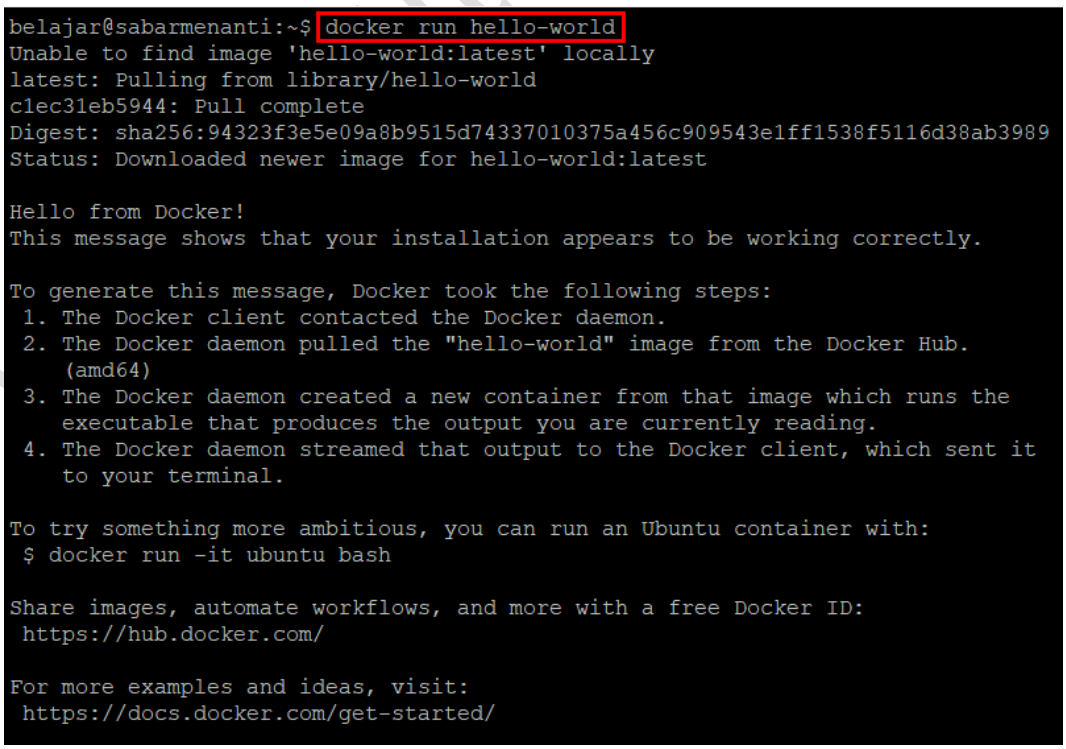
```
$ sudo apt-get update
```

9. Menginstalasi paket-paket **Docker** meliputi **Docker engine**, **containerd** dan **Docker compose**.

```
$ sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
```

10. Memverifikasi hasil instalasi **Docker engine** dengan menjalankan **image hello-world**.

```
$ sudo docker run hello-world
```



11. Menampilkan informasi versi **Docker** yang terinstalasi.

```
$ sudo docker --version
```

```
belajar@sabarmenanti:~$ sudo docker --version
Docker version 27.0.2, build 912c1dd
```

Terlihat versi *docker* yang terinstalasi adalah **27.0.2**.

12. Mengatur agar manajemen **Docker** sebagai user selain **root** (*user* biasa) dengan menambahkan *user* yang digunakan *login* saat ini yaitu "**belajar**" ke dalam **group docker**.

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

13. Memverifikasi hasil penambahan *user* "**belajar**" ke **group docker**.

```
$ grep docker /etc/group
```

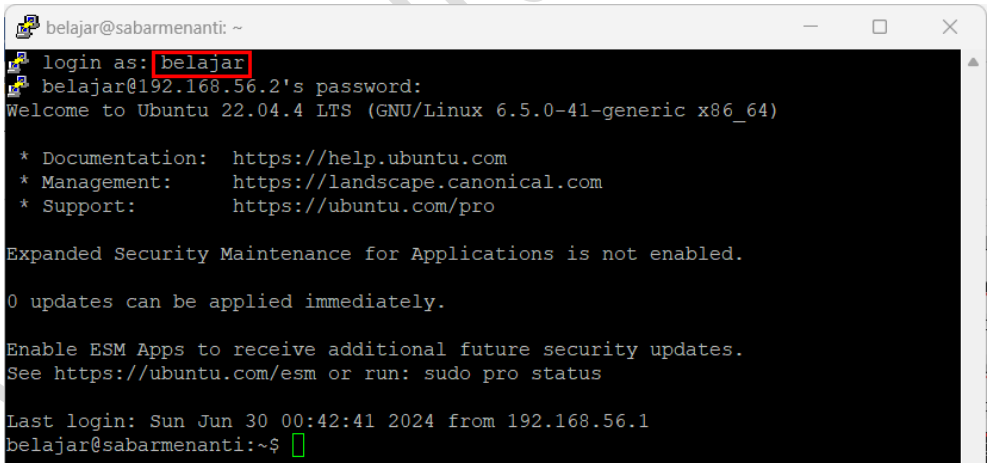
```
belajar@sabarmenanti:~$ grep docker /etc/group
docker:x:999:belajar
belajar@sabarmenanti:~$
```

Terlihat *user* "**belajar**" telah menjadi anggota dari **group docker**.

14. Lakukan **logout** atau keluar dari SSH untuk menguji perubahan terkait manajemen Docker sebagai *user* "**belajar**".

```
$ logout
```

15. Lakukan akses SSH kembali menggunakan aplikasi *Putty SSH Client* ke alamat IP dari *Ubuntu Desktop* yaitu **192.168.56.2** dan *login* menggunakan *user* "**belajar**" dengan *password* "**12345678**", seperti terlihat pada gambar berikut:



16. Menampilkan informasi **User ID** dan **Group ID** dari *user* yang digunakan *login* saat ini.

```
$ id
```

```
belajar@sabarmenanti:~$ id
uid=1000(belajar) gid=1000(belajar) groups=1000(belajar),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),122(lpadmin),135(lxd),136(sambashare),999(docker)
```

Terlihat *user* "**belajar**" menjadi anggota dari group Bernama **docker** dengan **group ID 999**.

17. Menampilkan informasi versi dari **Docker** tanpa menggunakan **sudo**.

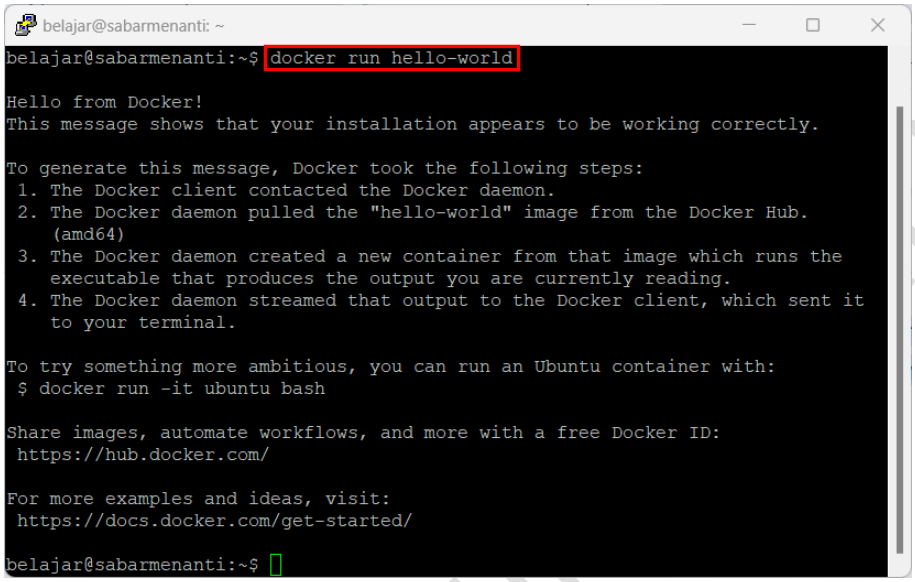
```
$ docker --version
```

```
belajar@sabarmenanti:~$ docker --version
Docker version 27.0.2, build 912c1dd
```

Terlihat versi *docker* yang terinstalasi adalah **27.0.2**.

18. Menjalankan **docker image hello-world** tanpa menggunakan **sudo**.

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



Terlihat *image* tersebut dapat dijalankan.

BAB IV

DOCKER FUNDAMENTAL

A. Manajemen Docker Image

Adapun langkah-langkah dalam memanajemen *docker image* adalah sebagai berikut:

1. Menampilkan daftar *docker image* yang dimiliki dengan sintak penulisan perintah:

   ```
   docker image ls
   ```

   Perintah ini juga memiliki *alias*:

   ```
   docker image list
   ```

   ```
   docker images
   ```

   Sebagai contoh untuk menampilkan daftar *docker image* yang dimiliki di VM *Ubuntu* maka dapat mengeksekusi perintah:

   ```
   $ docker image ls
   ```

   ```
   belajar@sabarmenanti:~$ docker image ls
   REPOSITORY    TAG       IMAGE ID       CREATED          SIZE
   hello-world   latest    d2c94e258dcb   14 months ago    13.3kB
   ```

   Terlihat terdapat satu *image* bernama "**hello-world**" dengan *tag* "**latest**" dan memiliki *image* ID "**d2c94e258dcb**". Selain itu pula terlihat informasi bahwa *image* tersebut dibuat 14 (empat belas) bulan yang lalu (**14 months ago**) dan berukuran "**13.3Kb**".

2. Mengunduh *docker image* dari *Docker Registry* dengan sintak penulisan perintah:

   ```
   docker image pull NAME:TAG
   ```

   Penjelasan argumen:

   a. `NAME` merupakan nama *image* yang akan diunduh.

   b. `TAG` merupakan versi *image* yang akan diunduh.

   Perintah ini juga memiliki *alias*:

   ```
   docker pull NAME:TAG
   ```

   Sebagai contoh untuk mengunduh *docker image* bernama "**alpine**" dengan **tag** "**latest**" dari *Docker Registry* yaitu **Docker Hub** dengan alamat https://hub.docker.com maka dapat mengeksekusi perintah:

   ```
   $ docker image pull alpine:latest
   ```

```
belajar@sabarmenanti:~$ docker image pull alpine:latest
latest: Pulling from library/alpine
ec99f8b99825: Pull complete
Digest: sha256:b89d9c93e9ed3597455c90a0b88a8bbb5cb7188438f70953fede212a0c4394e0
Status: Downloaded newer image for alpine:latest
docker.io/library/alpine:latest
```

Tunggu hingga proses pengunduhan *image* selesai dilakukan.

3. Memverifikasi hasil pengunduhan *docker image* **alpine** dengan mengeksekusi perintah untuk melihat daftar *docker image* yang dimiliki.

   ```
   $ docker image ls
   ```

```
belajar@sabarmenanti:~$ docker image ls
REPOSITORY    TAG       IMAGE ID       CREATED         SIZE
alpine        latest    a606584aa9aa   9 days ago      7.8MB
hello-world   latest    d2c94e258dcb   14 months ago   13.3kB
```

   Terlihat terdapat dua *image* dan salah satunya bernama **alpine** dengan **tag** "**latest"** dan memiliki *image* ID "**a606584aa9aa**". Selain itu pula terlihat informasi bahwa *image* tersebut dibuat 9 (sembilan) hari yang lalu (**9 days ago**) dan berukuran "**7.8 MB**".

4. Menghapus *docker image* tertentu dengan sintak penulisan perintah:

   ```
   docker image rm NAME:TAG
   ```

   Penjelasan argumen:

   a. `NAME` merupakan nama *image* yang akan dihapus.

   b. `TAG` merupakan versi *image* yang akan dihapus.

   Perintah ini juga memiliki *alias*:

   ```
   docker image remove NAME:TAG
   docker rmi NAME:TAG
   ```

   Sebagai contoh untuk menghapus *docker image* bernama "**alpine**" dengan tag "**latest**" maka dapat mengeksekusi perintah:

   ```
   $ docker image rm alpine:latest
   ```

```
belajar@sabarmenanti:~$ docker image rm alpine:latest
Untagged: alpine:latest
Untagged: alpine@sha256:b89d9c93e9ed3597455c90a0b88a8bbb5cb7188438f70953fede212a
0c4394e0
Deleted: sha256:a606584aa9aa875552092ec9e1d62cb98d486f51f389609914039aabd9414687
Deleted: sha256:94e5f06ff8e3d4441dc3cd8b090ff38dc911bfa8ebdb0dc28395bc98f82f983f
```

5. Memverifikasi hasil penghapusan *docker image* **alpine** dengan mengeksekusi perintah untuk melihat daftar *docker image* yang dimiliki.

   ```
   $ docker image ls
   ```

```
belajar@sabarmenanti:~$ docker image ls
REPOSITORY    TAG       IMAGE ID       CREATED         SIZE
hello-world   latest    d2c94e258dcb   14 months ago   13.3kB
```

Image tersebut telah berhasil dihapus.

6. Mengunduh *docker image* bernama "**alpine**" namun tanpa mencantumkan **tag** dari *Docker Registry* yaitu **Docker Hub** dengan alamat https://hub.docker.com.

```
belajar@sabarmenanti:~$ docker image pull alpine
Using default tag: latest
latest: Pulling from library/alpine
ec99f8b99825: Pull complete
Digest: sha256:b89d9c93e9ed3597455c90a0b88a8bbb5cb7188438f70953fede212a0c4394e0
Status: Downloaded newer image for alpine:latest
docker.io/library/alpine:latest
```

Terlihat *docker image* **alpine** yang diunduh menggunakan *default* **tag** yaitu **latest** atau versi terkini. Hal ini karena *tag* tidak dicantumkan ketika mengeksekusi perintah mengunduh *image* tersebut sehingga secara *default* akan menggunakan *tag* "**latest**".

7. Menampilkan atau melihat daftar *docker image* yang dimiliki.

```
$ docker image ls
```

```
belajar@sabarmenanti:~$ docker image ls
REPOSITORY    TAG       IMAGE ID       CREATED          SIZE
alpine        latest    a606584aa9aa   9 days ago       7.8MB
hello-world   latest    d2c94e258dcb   14 months ago    13.3kB
```

Terlihat terdapat dua *image* dan salah satunya bernama **alpine** dengan **tag** "**latest"** dan memiliki *image* ID "**a606584aa9aa**". Selain itu pula terlihat informasi bahwa *image* tersebut dibuat 9 (sembilan) hari yang lalu (**9 days ago**) dan berukuran "**7.8 MB**".

B. Manajemen Docker Container

Adapun langkah-langkah dalam memanajemen *docker container* adalah sebagai berikut:

1. Menampilkan daftar *docker container* dengan sintak penulisan perintah:

```
docker container ls [OPTIONS]
```

Penjelasan salah satu *options*:

`-a, --all` digunakan untuk menampilkan daftar seluruh *container* baik sedang berjalan maupun tidak. Secara *default* hanya menampilkan daftar *container* yang sedang berjalan.

Perintah ini juga memiliki *alias*:

```
docker container list
```

```
docker container ps

docker ps
```

Sebagai contoh untuk menampilkan daftar *docker container* yang sedang berjalan saja di VM *Ubuntu* maka dapat mengeksekusi perintah:

```
$ docker container ls
```

```
belajar@sabarmenanti:~$ docker container ls
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
```

Terlihat tidak ada *container* yang sedang berjalan.

2. Menampilkan daftar keseluruhan *docker container* baik yang sedang berjalan (*running*) atau tidak.

```
$ docker container ls -a
```

```
belajar@sabarmenanti:~$ docker container ls -a
CONTAINER ID   IMAGE         COMMAND     CREATED          STATUS                      PORTS     NAMES
4e89ae4978b6   hello-world   "/hello"    13 hours ago     Exited (0) 13 hours ago               blissful_williamson
5f80c3d17fdd   hello-world   "/hello"    13 hours ago     Exited (0) 13 hours ago               sharp_saha
```

Terlihat terdapat 2 (dua) *container* yang tidak berjalan yaitu ditandai dengan nilai **Exited** pada kolom **STATUS**.

3. Membuat *docker container* dari *image* tertentu tanpa menjalankannya dengan sintak penulisan perintah:

```
docker container create [OPTIONS] IMAGE [COMMAND] [ARG…]
```

Penjelasan salah satu *options*:

`--name` digunakan untuk memberikan nama pada *container* yang dibuat.

Argumen `IMAGE` memuat nama dan *tag* dari *image* yang digunakan untuk membuat *container*. Sedangkan `[COMMAND] [ARG…]` digunakan untuk mencantumkan perintah dan argumen yang akan dieksekusi pada *container* yang dibuat.

Perintah ini juga memiliki *alias*:

```
docker create
```

Sebagai contoh untuk membuat *container* dari *image* "**nginx**" dengan *tag* "**latest**" di VM *Ubuntu* dan tanpa menjalankannya maka dapat mengeksekusi perintah:

```
$ docker container create nginx:latest
```

```
belajar@sabarmenanti:~$ docker container create nginx:latest
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
2cc3ae149d28: Pull complete
1018f2b8dba8: Pull complete
b831e78d8e20: Pull complete
3ab22521e919: Pull complete
5112bf42775b: Pull complete
cbdaf9e4ee2d: Pull complete
a06b6fd631e8: Pull complete
Digest: sha256:9c367186df9a6b18c6735357b8eb7f407347e84aea09beb184961cb83543d46e
Status: Downloaded newer image for nginx:latest
a8b4b009a12f800368a4fc29832b6a0744e5855f4ef8beb8003a20a064d500f0
```

Pada bagian awal dari *output* hasil eksekusi perintah tersebut memperlihatkan pesan "**Unable to find image 'nginx:latest' locally**". Pesan tersebut bermakna bahwa *image* **nginx:latest** tidak ditemukan di lokal sehingga *image* tersebut langsung diunduh dari *Docker Registry* di *Internet* yaitu dari **Docker Hub**. Selain itu juga pada bagian paling akhir terlihat informasi *container* ID dari *container* yang dibuat yaitu "**a8b4b009a12f800368a4fc29832b6a0744e5855f4ef8beb8003a20a064d500f0**".

4. Menampilkan daftar *docker container* yang sedang berjalan.

```
$ docker container ls
```

```
belajar@sabarmenanti:~$ docker container ls
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
```

Terlihat tidak ada *container* yang sedang berjalan.

5. Menampilkan keseluruhan daftar *docker container* baik yang sedang berjalan (*running*) maupun tidak.

```
$ docker container ls -a
```

```
belajar@sabarmenanti:~$ docker container ls -a
CONTAINER ID   IMAGE         COMMAND                  CREATED         STATUS                     PORTS     NAMES
a8b4b009a12f   nginx:latest  "/docker-entrypoint.…"   3 minutes ago   Created                              exciting_vaughan
4e89ae4978b6   hello-world   "/hello"                 14 hours ago    Exited (0) 14 hours ago              blissful_williamson
5f80c3d17fdd   hello-world   "/hello"                 14 hours ago    Exited (0) 14 hours ago              sharp_saha
```

Terlihat informasi terkait *container* yang dibuat pada langkah sebelumnya yaitu menggunakan **IMAGE nginx:latest** yang dibuat 3 (tiga) menit yang lalu (**3 minutes ago**) pada kolom **CREATED** serta memiliki status **Created** atau telah berhasil dibuat pada kolom **STATUS**. *Container* yang dibuat tersebut juga terlihat telah diberikan nama secara acak (*random*) oleh *Docker* pada kolom **NAMES** yaitu "**exciting_vaughan**" dan memiliki **CONTAINER ID** "**a8b4b009a12f**".

Selain itu juga terlihat terdapat 2 (dua) *docker container* menggunakan **IMAGE hello-world** yang tidak berjalan yaitu ditandai dengan nilai **Exited** pada kolom **STATUS**.

6. Menjalankan *docker container* yang telah dibuat dengan sintak penulisan perintah:

```
docker container start CONTAINER
```

Argumen `CONTAINER` memuat nama *container* atau *container* ID yang akan dijalankan.

Perintah ini juga memiliki *alias*:

```
docker start
```

Sebagai contoh untuk menjalankan *docker container* bernama "**exciting_vaughan**" yang telah dibuat sebelumnya maka dapat mengeksekusi perintah:

```
$ docker container start exciting_vaughan
```

```
belajar@sabarmenanti:~$ docker container start exciting_vaughan
exciting_vaughan
```

Terlihat informasi nama *container* yang telah berhasil dijalankan.

7. Memverifikasi *docker container* bernama "**exciting_vaughan**" telah berjalan.

```
$ docker container ls
```

```
belajar@sabarmenanti:~$ docker container ls
CONTAINER ID   IMAGE          COMMAND                  CREATED         STATUS         PORTS     NAMES
a8b4b009a12f   nginx:latest   "/docker-entrypoint.…"   5 minutes ago   Up 5 seconds   80/tcp    exciting_vaughan
```

Terlihat *container* bernama "**exciting_vaughan**" sedang berjalan (*running*) dan telah aktif 5 detik yang lalu (**Up 5 seconds**) berdasarkan informasi pada kolom **STATUS**. Selain itu juga terlihat informasi *port* yang digunakan oleh *container* yaitu **80/tcp** pada kolom **PORTS**.

8. Menghentikan *docker container* tertentu yang sedang berjalan dengan sintak penulisan perintah:

```
docker container stop CONTAINER
```

Argumen `CONTAINER` memuat nama *container* atau *container* ID yang akan dihentikan.

Perintah ini juga memiliki *alias*:

```
docker stop
```

Sebagai contoh untuk menghentikan *docker container* bernama "**exciting_vaughan**" yang telah berjalan sebelumnya maka dapat mengeksekusi perintah:

```
$ docker container stop exciting_vaughan
```

```
belajar@sabarmenanti:~$ docker container stop exciting_vaughan
exciting_vaughan
```

Terlihat informasi nama *container* yang telah berhasil dihentikan.

9. Memverifikasi keberhasilan penghentian *container* bernama "**exciting_vaughan**".

```
$ docker container ls
```

```
belajar@sabarmenanti:~$ docker container ls
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
```

Terlihat sudah tidak ada *container* yang sedang berjalan. Selanjutnya eksekusi perintah untuk melihat keseluruhan daftar *docker container* baik yang sedang berjalan (*running*) maupun tidak.

```
$ docker container ls -a
```

```
belajar@sabarmenanti:~$ docker container ls -a
CONTAINER ID   IMAGE         COMMAND                  CREATED         STATUS                      PORTS     NAMES
a8b4b009a12f   nginx:latest  "/docker-entrypoint.…"   6 minutes ago   Exited (0) 8 seconds ago              exciting_vaughan
4e89ae4978b6   hello-world   "/hello"                 14 hours ago    Exited (0) 14 hours ago               blissful_williamson
5f80c3d17fdd   hello-world   "/hello"                 14 hours ago    Exited (0) 14 hours ago               sharp_saha
```

Terlihat informasi bahwa *container* bernama "**exciting_vaughan**" telah berhasil dihentikan yaitu ditandai dengan nilai **Exited** pada kolom **STATUS** 8 (delapan) detik yang lalu **(8 seconds ago)**.

10. Menghapus *docker container* dengan sintak penulisan perintah:

```
docker container rm [OPTIONS] CONTAINER
```

Penjelasan salah satu *options*:

`-f, --force` digunakan untuk memaksa menghapus *container* yang sedang berjalan.

Argumen `CONTAINER` memuat nama *container* atau *container* ID yang akan dihapus.

Perintah ini juga memiliki *alias*:

```
docker container remove
```

```
docker rm
```

Sebagai contoh untuk menghapus *docker container* bernama "**exciting_vaughan**" yang telah dihentikan sebelumnya maka dapat mengeksekusi perintah:

```
$ docker container rm exciting_vaughan
```

Atau jika menggunakan *container* ID "**a8b4b009a12f**" dari *container* tersebut maka dapat mengeksekusi perintah:

```
$ docker container rm a8b4b009a12f
```

```
belajar@sabarmenanti:~$ docker container rm a8b4b009a12f
a8b4b009a12f
```

Terlihat informasi *container* ID dari *container* yang telah berhasil dihapus.

11. Memverifikasi hasil penghapusan *container* dengan ID ““**a8b4b009a12f**””.

```
$ docker container ls -a
```

```
belajar@sabarmenanti:~$ docker container ls -a
CONTAINER ID   IMAGE         COMMAND    CREATED         STATUS                     PORTS     NAMES
4e89ae4978b6   hello-world   "/hello"   14 hours ago    Exited (0) 14 hours ago              blissful_williamson
5f80c3d17fdd   hello-world   "/hello"   14 hours ago    Exited (0) 14 hours ago              sharp_saha
```

Sudah tidak terlihat *container* ID tersebut sehingga penghapusan *container* telah berhasil dilakukan.

12. Membuat dan menjalankan *docker container* baru dari *image* tertentu dengan sintak penulisan perintah:

```
docker container run [OPTIONS] IMAGE [COMMAND] [ARG…]
```

Penjelasan beberapa *options* yang umum digunakan:

a. `--cpus` digunakan untuk menentukan jumlah CPU yang digunakan oleh *container*.
b. `-d, --detach` digunakan untuk menjalankan *container* di *background* dan menampilkan *container* ID.
c. `-e, --env` digunakan untuk mengatur *environment variable*.
d. `-i, --interactive` digunakan untuk membiarkan STDIN terbuka atau menjaga input tetap aktif.
e. `-m, --memory` digunakan untuk membatasi jumlah memori yang dapat digunakan oleh *container*.
f. `--mount` digunakan untuk menghubungkan sistem file ke *container*.
g. `--name` digunakan untuk memberikan nama pada *container* yang dibuat.
h. `--network` digunakan untuk menghubungkan *container* ke *network* tertentu.
i. `-p, --publish` digunakan untuk mempublikasikan *port container* ke *host*. Argumen dari *option* ini memiliki format penulisan `PORTHOST:PORTCONTAINER`. `PORTHOST` adalah nomor *port* di *host machine*. Sebaliknya `PORTCONTAINER` adalah nomor *port* di dalam *container*.
j. `--rm` digunakan untuk secara otomatis menghapus *container* ketika dihentikan (*exit*).

k. `-t, --tty` digunakan untuk mengalokasikan *pseudo*-TTY.

l. `-v, --volume` digunakan untuk melakukan *bind mount volume* tertentu.

Argumen `IMAGE` memuat nama dan *tag* dari *image* yang digunakan untuk membuat *container*. Sedangkan `[COMMAND] [ARG...]` digunakan untuk mencantumkan perintah dan argumen yang akan dieksekusi pada *container* yang dibuat.

Perintah ini juga memiliki *alias*:

```
docker run
```

Sebagai contoh untuk membuat dan menjalankan container dengan ketentuan sebagai berikut:

- Nama *container* adalah **nginx**.
- *Container image* menggunakan **nginx** dengan **tag "latest"**.
- Merutekan trafik dari **port 80** di mesin **host** ke **port 80** juga di dalam *container*.
- *Container* berjalan di *background*.

Maka dapat mengeksekusi perintah:

```
$ docker run --name nginx -d -p 80:80 nginx:latest
```

```
belajar@sabarmenanti:~$ docker run --name nginx -d -p 80:80 nginx:latest
70d3a045f01d99e074100bab7392cbfe8ec6ffe117d80f5784a38bdeaa234b6a
```

Terlihat informasi *container* ID dari *container* yang telah berhasil dibuat dan dijalankan.

13. Memverifikasi *docker container* bernama "**nginx**" telah berjalan.

```
$ docker ps
```

```
belajar@sabarmenanti:~$ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED         STATUS         PORTS                                   NAMES
70d3a045f01d   nginx:latest   "/docker-entrypoint.…"   3 minutes ago   Up 3 minutes   0.0.0.0:80->80/tcp, :::80->80/tcp   nginx
```

Terlihat *container* bernama "**nginx**" telah berhasil dibuat menggunakan *image* **nginx:latest** pada 3 menit yang lalu berdasarkan informasi pada kolom **CREATED** dan telah aktif juga 3 menit yang lalu (**Up 3 minutes**) berdasarkan informasi pada kolom **STATUS**. Selain itu juga terlihat informasi *port* yang digunakan oleh *container* yaitu **80** pada *host* akan dirutekan ke port **80/tcp** di dalam *container* sesuai dengan nilai pada kolom **PORTS**.

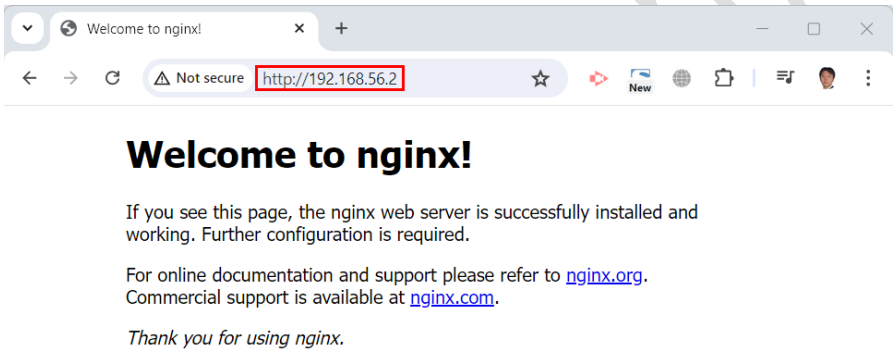
14. Memverifikasi akses ke layanan *web* dari *container* **nginx** pada **port 80** menggunakan aplikasi **curl**.

```
$ curl localhost
```

```
belajar@sabarmenanti:~$ curl localhost
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

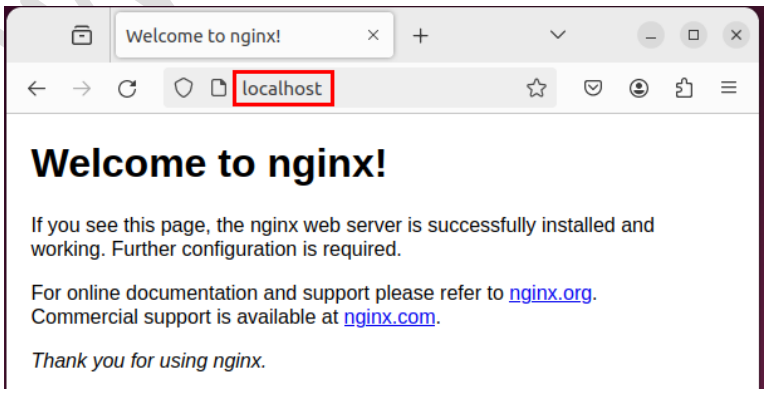
Terlihat konten *homepage* dari *container* **nginx** berhasil diakses.

15. Memverifikasi akses ke layanan web dari *container* **nginx** pada **port 80** menggunakan *browser* **Chrome** atau lainnya di **Windows 11** ke alamat **http://192.168.56.2**.



Terlihat konten *homepage* dari *container* **nginx** berhasil diakses.

16. Memverifikasi akses ke layanan web dari *container* **nginx** pada **port 80** menggunakan *browser* **Firefox** di **VM Ubuntu Desktop** ke alamat **http://localhost**.



Terlihat konten *homepage* dari *container* **nginx** berhasil diakses.

C. Mengakses Logs dari Docker Container

Docker menyediakan *logs* dari *container* terkait layanan yang berjalan di dalam *container* tersebut. *Log* tersebut memuat informasi yang bermanfaat dan dapat digunakan untuk kebutuhan *troubleshooting* ketika terjadi permasalahan pada *container*. Sintak penulisan dari perintah yang digunakan untuk mengakses atau mengambil *log* untuk *docker container* tertentu adalah sebagai berikut:

```
docker container logs [OPTIONS] CONTAINER
```

Penjelasan dari salah satu *options* yang umum digunakan:

`-f, --follow` digunakan untuk melihat *log* dari *container* secara *real time*.

Argumen `CONTAINER` memuat nama *container* atau *container* ID yang akan diakses atau diambil *log*-nya.

Perintah ini juga memiliki *alias*:

```
docker logs
```

Sebagai contoh untuk melihat *logs* dari *container* bernama **nginx** yang sedang berjalan maka dapat mengeksekusi perintah:

```
$ docker container logs nginx
```

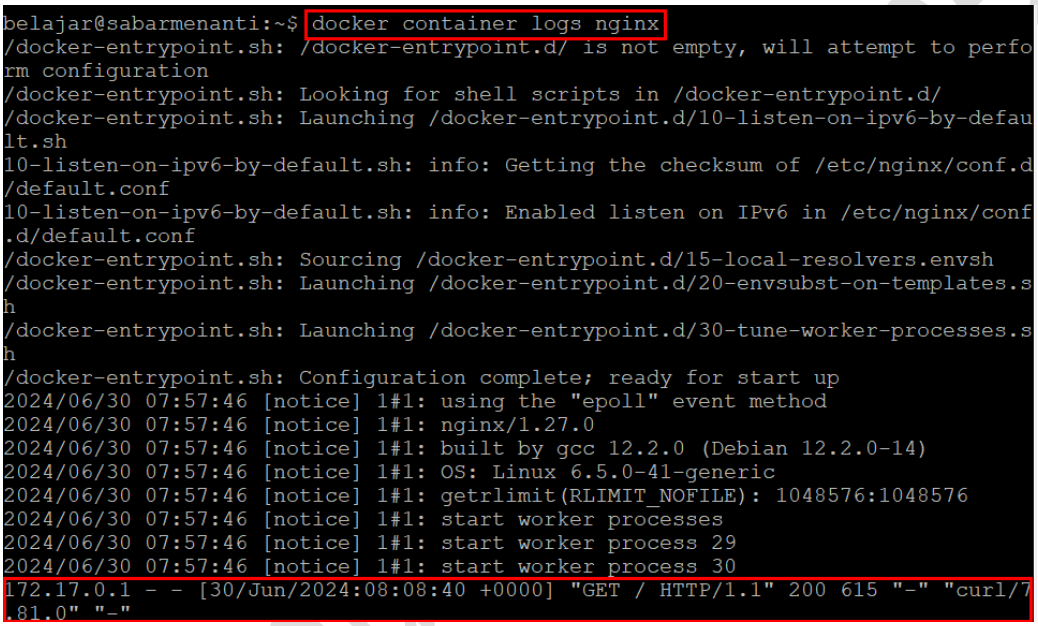
```
belajar@sabarmenanti:~$ docker container logs nginx
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perfo
rm configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-defau
lt.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d
/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf
.d/default.conf
/docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.s
h
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.s
h
/docker-entrypoint.sh: Configuration complete; ready for start up
2024/06/30 07:57:46 [notice] 1#1: using the "epoll" event method
2024/06/30 07:57:46 [notice] 1#1: nginx/1.27.0
2024/06/30 07:57:46 [notice] 1#1: built by gcc 12.2.0 (Debian 12.2.0-14)
2024/06/30 07:57:46 [notice] 1#1: OS: Linux 6.5.0-41-generic
2024/06/30 07:57:46 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2024/06/30 07:57:46 [notice] 1#1: start worker processes
2024/06/30 07:57:46 [notice] 1#1: start worker process 29
2024/06/30 07:57:46 [notice] 1#1: start worker process 30
172.17.0.1 - - [30/Jun/2024:08:08:40 +0000] "GET / HTTP/1.1" 200 615 "-" "curl/7
.81.0" "-"
```

Pada bagian akhir dari *output* tersebut terlihat *log* terkait pengaksesan layanan *container* **nginx** menggunakan utilitas **curl** yaitu yang dieksekusi menggunakan perintah `curl localhost` pada bagian B di langkah 14 sebelumnya.

Sedangkan untuk melihat *logs* dari *container* bernama **nginx** yang sedang berjalan secara *real time* maka dapat mengeksekusi perintah:

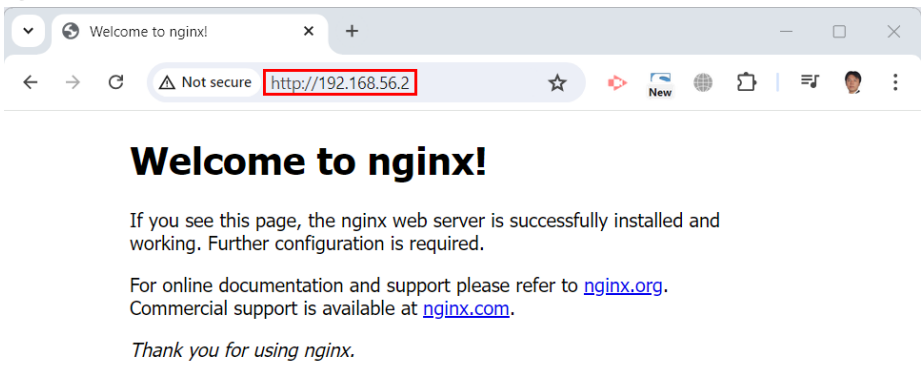
```
$ docker container logs -f nginx
```

Cuplikan *output* dari hasil eksekusi perintah tersebut, seperti terlihat pada gambar berikut:



Prompt (`$`) dari terminal *Ubuntu* tidak akan tampil. Hal ini sebagai dampak dari penggunaan *options* `-f` untuk menampilkan *logs* secara *real time* dari *docker container* **nginx**.

Selanjutnya lakukan pengaksesan kembali ke layanan *web* dari *container* **nginx** pada **port 80** menggunakan *browser* **Chrome** atau lainnya di **Windows 11** ke alamat **http://192.168.56.2**, seperti terlihat pada gambar berikut:



Terlihat konten *homepage* dari *container* **nginx** telah berhasil diakses kembali.

Terlampir cuplikan *logs* pada *terminal* dari aplikasi **Putty** sebagai hasil dari pengaksesan layanan *web* pada *container* **nginx** dari *browser* **Chrome** di **Windows 11**.

```
192.168.56.1 - - [30/Jun/2024:09:17:21 +0000] "GET / HTTP/1.1" 304 0 "-" "Mozill
a/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrom
e/126.0.0.0 Safari/537.36" "-"
```

Untuk keluar dari penampilan *logs* secara *real time* maka lakukan penekanan tombol **CTRL+C** pada *terminal* dari aplikasi *Putty* sehingga akan tampil pesan **^Ccontext canceled**, seperti terlihat pada gambar berikut:

```
192.168.56.1 - - [30/Jun/2024:09:17:21 +0000] "GET / HTTP/1.1" 304 0 "-" "Mozill
a/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrom
e/126.0.0.0 Safari/537.36" "-"
^Ccontext canceled
belajar@sabarmenanti:~$
```

Terlihat *prompt* (`$`) dari terminal di aplikasi *Putty* telah tampil kembali.

D. Mengeksekusi Perintah Di Dalam Docker Container

Sintak penulisan dari perintah yang digunakan untuk mengeksekusi perintah di dalam *docker container* adalah sebagai berikut:

```
docker container exec [OPTIONS] CONTAINER COMMAND [ARG…]
```

Penjelasan beberapa *options* yang umum digunakan:

a. `-i`, `--interactive` digunakan untuk membiarkan STDIN terbuka atau menjaga *input* tetap aktif.
b. `-t`, `--tty` digunakan untuk mengalokasikan *pseudo*-TTY.

Argumen `CONTAINER` memuat nama *container* atau *container* ID yang akan diakses sebagai lokasi atau target eksekusi perintah. Sedangkan `COMMAND [ARG…]` digunakan untuk mencantumkan perintah yang akan dieksekusi dan argumen yang akan dieksekusi pada *container* yang dibuat.

Perintah ini juga memiliki *alias*:

```
docker exec
```

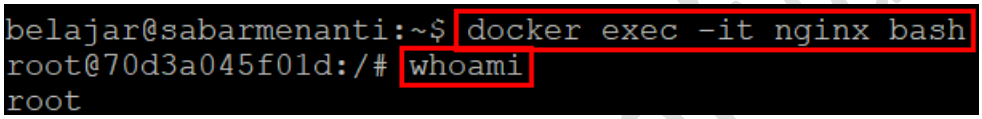
Sebagai contoh untuk masuk ke dalam *docker container* bernama **nginx** yang telah berjalan secara interaktif dan mengalokasikan *pseudo*-TTY serta *shell* **bash** sebagai *command* atau proses utama maka dapat mengeksekusi perintah berikut pada *terminal* dari aplikasi *Putty*:

```
$ docker exec -it nginx bash
```

Terlihat telah berhasil masuk ke dalam *container* **nginx** yang ditandai dengan *prompt* **root@70d3a045f01d**.

Lakukan eksekusi perintah `whoami` untuk menampilkan informasi *username* dari pengguna saat ini di dalam *container* tersebut.

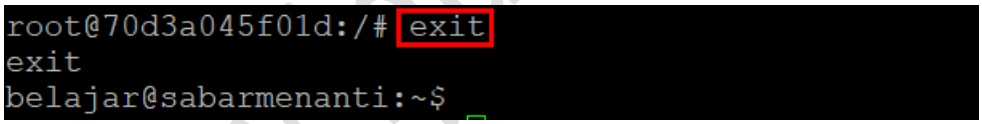
```
# whoami
```



Terlihat informasi *username* yang digunakan di dalam *container* adalah **root**.

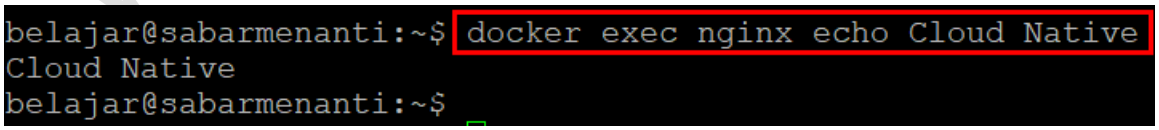
Selanjutnya lakukan eksekusi perintah `exit` untuk keluar dari *container*.

```
# exit
```



Contoh berikutnya adalah mengeksekusi perintah `echo` dengan argumen **Cloud Native** di dalam *container* **nginx**.

```
$ docker exec nginx echo Cloud Native
```



Terlihat *output* **Cloud Native** sebagai hasil dari eksekusi perintah `echo` di dalam *container* tersebut.

E. Manajemen Docker Network

Adapun langkah-langkah dalam memanajemen *docker network* adalah sebagai berikut:

1. Menampilkan daftar *docker network* yang dimiliki dengan sintak penulisan perintah:

   ```
   docker network ls
   ```

   Perintah ini juga memiliki *alias*:

   ```
   docker network list
   ```

   Sebagai contoh untuk menampilkan daftar *docker network* yang dimiliki di VM *Ubuntu* maka dapat mengeksekusi perintah:

   ```
   $ docker network ls
   ```

   ```
   belajar@sabarmenanti:~$ docker network ls
   NETWORK ID     NAME      DRIVER    SCOPE
   5cba8811f3a0   bridge    bridge    local
   5aac6e2179b1   host      host      local
   6ec967d4f726   none      null      local
   ```

   Terlihat terdapat 3 (tiga) *network* yaitu bernama **bridge** dengan *network driver* bertipe "*bridge*", **host** dengan *network driver* bertipe "*host*" dan **none** dengan *network driver* bertipe "*null*" atau "*none*".

   Network driver bertipe "**bridge**" menjadi *network driver default* di *Docker* dan digunakan agar suatu *container* dapat berkomunikasi dengan *container* lain di *host* yang sama.. Sedangkan *network driver* bertipe "**host**" digunakan untuk menghapus isolasi jaringan antara *container* dengan *host Docker* sehingga akan secara langsung menggunakan jaringan dari *host*. Terakhir *network driver* bertipe "**none**" digunakan untuk mengisolasi sepenuhnya suatu *container* dari *host* dan *container* lainnya.

2. Membuat *docker network* baru dengan sintak penulisan perintah:

   ```
   docker network create [OPTIONS] NETWORK
   ```

   Penjelasan salah satu *options*:

   `-d, --driver` digunakan untuk menentukan jenis *network driver* yang digunakan untuk mengelola jaringan yang dibuat. Secara *default* bertipe *bridge*. Pilihan *driver* yang tersedia meliputi **bridge** atau **overlay** yang merupakan *network driver* bawaan. Sedangkan pilihan *network driver* lainnya memerlukan instalasi terlebih dahulu sebelum dapat digunakan.

Argumen `NETWORK` merupakan nama dari jaringan baru yang akan dibuat.

Sebagai contoh untuk membuat *docker network* baru bernama **belajar** dengan *network driver* bertipe **bridge** maka dapat mengeksekusi perintah:

```
$ docker network create belajar
```

```
belajar@sabarmenanti:~$ docker network create belajar
0d75c527773b13cba5f7a1e74d33631af5d7629b877af2c695e1f17011c78703
```

Terlihat *network* ID dari *network* **belajar** yang dibuat.

3. Memverifikasi hasil dari pembuatan *docker network* baru bernama **belajar**.

```
$ docker network ls
```

```
belajar@sabarmenanti:~$ docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
0d75c527773b   belajar   bridge    local
5cba8811f3a0   bridge    bridge    local
5aac6e2179b1   host      host      local
6ec967d4f726   none      null      local
```

Terlihat telah terdapat *docker network* bernama **belajar** dengan *network driver* bertipe **bridge**.

4. Membuat *docker container* baru yang memanfaatkan *docker network* yang telah dibuat dengan ketentuan sebagai berikut:
    - *Container* berjalan di *background*.
    - Nama *container* adalah **mariadb**.
    - *Container* terhubung ke *docker network* **belajar**.
    - *Environment variable* `MARIADB_ROOT_PASSWORD` bernilai `sabarmenanti`
    - Merutekan trafik dari **port 3306** di mesin **host** ke **port 3306** juga di dalam *container*.
    - *Container image* menggunakan **mariadb** dengan **tag "latest"**.

    Maka dapat mengeksekusi perintah:

```
$ docker run -d --name mariadb --network belajar \
--env MARIADB_ROOT_PASSWORD=sabarmenanti \
-p 3306:3306 mariadb:latest
```

```
belajar@sabarmenanti:~$ docker run -d --name mariadb --network belajar \
> --env MARIADB_ROOT_PASSWORD=sabarmenanti \
> -p 3306:3306 mariadb:latest
Unable to find image 'mariadb:latest' locally
latest: Pulling from library/mariadb
9c704ecd0c69: Pull complete
f8f7f3c9a741: Pull complete
97c034108521: Pull complete
50366979c20a: Pull complete
0221331af5c0: Pull complete
e3c4d1c9d9cb: Pull complete
eef7a8467f98: Pull complete
60c15bb5bb03: Pull complete
Digest: sha256:e59ba8783bf7bc02a4779f103bb0d8751ac0e10f9471089709608377eded7aa8
Status: Downloaded newer image for mariadb:latest
fabbec664016ec792d05604874ea4282f44ec4bd4cd58e5c30e6cd959fa68a09
```

Pada bagian awal dari *output* hasil eksekusi perintah tersebut memperlihatkan pesan "**Unable to find image 'mariadb:latest' locally**". Pesan tersebut bermakna bahwa *image* **mariadb:latest** tidak ditemukan di lokal sehingga *image* tersebut langsung diunduh dari *Docker Registry* di *Internet* yaitu dari **Docker Hub**. Selain itu juga pada bagian paling akhir terlihat informasi *container* ID dari *container* yang dibuat yaitu "**fabbec664016ec792d05604874ea4282f44ec4bd4cd58e5c30e6cd959fa68a09**".

5. Memverifikasi *docker container* bernama "**mariadb**" telah berjalan.

   ```
   $ docker ps
   ```

```
belajar@sabarmenanti:~$ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED         STATUS         PORTS                                      NAMES
fabbec664016   mariadb:latest "docker-entrypoint.s…"   6 minutes ago   Up 6 minutes   0.0.0.0:3306->3306/tcp, :::3306->3306/tcp  mariadb
58b043605e66   nginx:latest   "/docker-entrypoint.…"   2 hours ago     Up 2 hours     0.0.0.0:80->80/tcp, :::80->80/tcp          nginx
```

   Terlihat *container* bernama "**mariadb**" telah berhasil dibuat menggunakan *image* **mariadb:latest** pada 6 menit yang lalu berdasarkan informasi pada kolom **CREATED** dan telah aktif juga 6 menit yang lalu (**Up 6 minutes**) berdasarkan informasi pada kolom **STATUS**. Selain itu juga terlihat informasi *port* yang digunakan oleh *container* yaitu **3306** pada *host* akan dirutekan ke port **3306/tcp** di dalam *container* sesuai dengan nilai pada kolom **PORTS**.

6. Menginspeksi *docker container* tertentu terkait *docker network* yang digunakan dengan sintak penulisan perintah:

   ```
   docker inspect [OPTIONS] CONTAINER
   ```

   Penjelasan dari salah satu *options* yang umum digunakan:

   `-f, --format` digunakan untuk memformat *output* menggunakan *Go template*.

   Argumen `CONTAINER` memuat nama *container* atau *container* ID yang akan diinspeksi. Secara *default docker inspect* akan melakukan *render* hasil dalam *JSON array*.

Sebagai contoh untuk menginspeksi *docker container* **mariadb** terkait *network* yang digunakan maka dapat mengeksekusi perintah:

```
$ docker inspect \
--format='{{.NetworkSettings.Networks}}' mariadb
```

```
belajar@sabarmenanti:~$ docker inspect \
> --format='{{.NetworkSettings.Networks}}' mariadb
map[belajar:0xc0000eca80]
```

Terlihat informasi bahwa *docker container* **mariadb** menggunakan *network* **belajar**.

7. Menghentikan dan menghapus *docker container* **mariadb**.

```
$ docker rm -f mariadb
```

8. Memverifikasi penghentian dan penghapusan *docker container* **mariadb**.

```
$ docker ps -a
```

```
belajar@sabarmenanti:~$ docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS                    PORTS                                   NAMES
58b043605e66   nginx:latest   "/docker-entrypoint.…"   3 hours ago    Up 3 hours                0.0.0.0:80->80/tcp, :::80->80/tcp       nginx
4e89ae4978b6   hello-world    "/hello"                 23 hours ago   Exited (0) 23 hours ago                                           blissful_williamson
5f80c3d17fdd   hello-world    "/hello"                 23 hours ago   Exited (0) 23 hours ago                                           sharp_saha
```

Sudah tidak terlihat *docker container* dengan nama tersebut sehingga penghentian dan penghapusan *container* telah berhasil dilakukan.

9. Menghapus *docker network* tertentu dengan sintak penulisan perintah:

```
docker network rm NETWORK
```

Argumen `NETWORK` merupakan nama dari jaringan yang akan dihapus. *Network* tidak akan dapat dihapus apabila masih digunakan oleh suatu *container*. Sebelum menghapus maka diperlukan pemutusan koneksi dari *container* yang terhubung ke *network* tersebut.

Perintah ini juga memiliki *alias*:

```
docker network remove
```

Sebagai contoh untuk menghapus *docker network* bernama "**belajar**" maka dapat mengeksekusi perintah:

```
$ docker network rm belajar
```

```
belajar@sabarmenanti:~$ docker network rm belajar
belajar
```

Terlihat informasi *nama* dari *docker network* yang telah berhasil dihapus.

10. Memverifikasi hasil dari penghapusan *docker network* bernama **belajar**.

```
$ docker network ls
```

```
belajar@sabarmenanti:~$ docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
5cba8811f3a0   bridge    bridge    local
5aac6e2179b1   host      host      local
6ec967d4f726   none      null      local
```

Sudah tidak terlihat *docker network* dengan nama tersebut sehingga penghapusan *network* telah berhasil dilakukan.

F. Bind Mounts

Bind mounts merupakan fitur pada *Docker* yang digunakan untuk memasangkan (*mount*) direktori dari *host machine* ke dalam *container* sehingga konten direktori tersebut dapat diakses dari dalam *container*. *Bind mount* dapat dibuat menggunakan perintah yang sama dengan perintah untuk membuat dan menjalankan *docker container* baru dari *image* tertentu yaitu:

```
docker container run [OPTIONS] IMAGE [COMMAND] [ARG…]
```

Namun *options* yang memerlukan pengaturan lebih lanjut adalah:

`--mount` yang digunakan untuk menghubungkan sistem file ke *container*.

Options ini memiliki beberapa argumen atau *field* dengan format pasangan `key=value` yang dipisahkan menggunakan tanda koma (,). Terdapat 4 (empat) *field* umum yang memerlukan pengaturan meliputi:

a. `type` digunakan untuk menentukan jenis *mount* dengan pilihan berupa **bind**, **volume** dan **tmpfs**.
b. `source` digunakan untuk menentukan lokasi direktori sumber pada *host machine* yang akan dipasangkan ke dalam *container*. *Field* ini memiliki alias `src`.
c. `destination` digunakan untuk menentukan lokasi direktori tujuan *mounting* di *docker container*. *Field* ini memiliki alias `dst` dan `target`.
d. `readonly` digunakan untuk mengatur agar *mounting* bersifat *read only* atau data hanya dapat dibaca saja.

Sebagai contoh akan dilakukan *mount* direktori **/home/belajar/data** dari *host machine* (sumber) ke direktori **/mnt** pada *docker container* (tujuan) bernama **nginx-bind-mount** yang

dibuat menggunakan *image* **nginx:latest**. Adapun langkah-langkah pembuatan *bind mount* dengan ketentuan tersebut adalah sebagai berikut:

1. Membuat direktori bernama **data** dan memverifikasi hasil dari pembuatan direktori tersebut.

   ```
   $ mkdir data

   $ ls
   ```

   ```
   belajar@sabarmenanti:~$ mkdir data
   belajar@sabarmenanti:~$ ls
   data  Desktop  Documents  Downloads  Music  Pictures  Public  snap  Templates  Videos
   ```

   Terlihat direktori **data** telah berhasil dibuat.

2. Membuat *file* dengan nama **belajar.txt** yang disimpan di dalam direktori **data** dan memiliki konten "**Belajar Cloud Native**".

   ```
   $ echo "Belajar Cloud Native" > data/belajar.txt
   ```

3. Memverifikasi konten dari *file* **belajar.txt** yang terdapat di dalam direktori **data**.

   ```
   $ cat data/belajar.txt
   ```

   ```
   belajar@sabarmenanti:~$ cat data/belajar.txt
   Belajar Cloud Native
   ```

   Terlihat *file* **belajar.txt** telah memuat konten seperti yang telah ditentukan.

4. Membuat dan menjalankan *docker container* serta melakukan *bind mount* direktori sumber **/home/belajar/data** di *host machine* ke direktori tujuan di *container* yaitu **/mnt**.

   ```
   $ docker run -d --name nginx-bind-mount -p 81:80 \
   --mount "type=bind,source=/home/belajar/data,destination=/mnt" \
   nginx:latest
   ```

   ```
   belajar@sabarmenanti:~$ docker run -d --name nginx-bind-mount -p 81:80 \
   --mount "type=bind,source=/home/belajar/data,destination=/mnt" \
   nginx:latest
   e383f49348c089baeb74fa5120e55d725bcb842950e8e923ef6c1e2f62310067
   ```

   Terlihat informasi *container ID* dari *container* yang telah berhasil dibuat dan dijalankan.

5. Memverifikasi keberhasilan *mounting* direktori dengan mengeksekusi perintah di dalam *container* untuk menampilkan isi dari direktori **/mnt**.

```
$ docker exec nginx-bind-mount ls /mnt
$ docker exec nginx-bind-mount cat /mnt/belajar.txt
```

```
belajar@sabarmenanti:~$ docker exec nginx-bind-mount ls /mnt
belajar.txt
belajar@sabarmenanti:~$ docker exec nginx-bind-mount cat /mnt/belajar.txt
Belajar Cloud Native
```

Terlihat pada direktori **/mnt** di dalam *container* telah memuat file **belajar.txt** dengan konten "**Belajar Cloud Native**".

6. Menghentikan dan menghapus *docker container* bernama **nginx-bind-mount**.

```
$ docker rm -f nginx-bind-mount
```

```
belajar@sabarmenanti:~$ docker rm -f nginx-bind-mount
nginx-bind-mount
```

Terlihat informasi nama *container* yang telah dihentikan dan dihapus sebagai *output* dari eksekusi perintah tersebut.

7. Memverifikasi keberhasilan penghentian dan penghapusan *docker container* dengan nama **nginx-bind-mount**.

```
belajar@sabarmenanti:~$ docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED             STATUS                  PORTS                                   NAMES
c79d47b2340f   nginx:latest   "/docker-entrypoint.…"   About an hour ago   Up About an hour        0.0.0.0:80->80/tcp, :::80->80/tcp       nginx
4e89ae4978b6   hello-world    "/hello"                 34 hours ago        Exited (0) 34 hours ago                                         blissful_williamson
5f80c3d17fdd   hello-world    "/hello"                 35 hours ago        Exited (0) 35 hours ago                                         sharp_saha
```

Sudah tidak terlihat *docker* container dengan nama tersebut sehingga penghapusan dan penghentian *container* telah berhasil dilakukan.

G. Manajemen Docker Volume

Volume merupakan salah satu mekanisme untuk menyimpan data yang dihasilkan dan digunakan oleh *docker container*. Pengelolaan *volume* dilakukan sepenuhnya oleh *Docker* sehingga berbeda dengan *bind mount* yang bergantung pada struktur direktori dan sistem operasi dari *host machine*. *Volume* memiliki berbagai kelebihan dibandingkan dengan *bind mounts* seperti data tersimpan secara permanen pada *volume* meskipun *container* dihentikan dan dihapus serta kemudahan dalam melakukan *backup* dan *migrasi* daripada *bind mounts*. Selain itu pengelolaan volume dapat dilakukan baik menggunakan perintah-perintah *Docker Command Line Interface (CLI)* maupun *Docker Application Programming Interface (API)*.

Volume dapat diaplikasikan pada *container* menggunakan perintah yang sama dengan perintah untuk membuat dan menjalankan *docker container* baru dari *image* tertentu yaitu:

```
docker container run [OPTIONS] IMAGE [COMMAND] [ARG…]
```

Namun *options* yang memerlukan pengaturan lebih lanjut adalah:

`-v, --volume` yang digunakan untuk menghubungkan *volume* ke *container*.

Options ini memiliki tiga *field* yang dipisahkan oleh karaketer *colon* (:). *Field* harus memiliki urutan yang tepat. *Field* pertama berupa nama *volume*. Sedangkan *field* kedua berupa *path* atau lokasi direktori pada *container* sebagai target *mounting* dari *volume*. Terakhir *field* ketiga bersifat opsional yaitu `ro` digunakan untuk mengatur agar *volume* dipasangkan ke *container* sebagai *read-only* atau hanya dapat dibaca saja.

Adapun langkah-langkah untuk memanajemen *docker volume* adalah sebagai berikut:

1. Menampilkan daftar *docker volume* yang dimiliki dengan sintak penulisan perintah:

   ```
   docker volume ls
   ```

   Perintah ini juga memiliki *alias*:

   ```
   docker volume list
   ```

   Sebagai contoh untuk menampilkan daftar *docker volume* yang dimiliki di VM *Ubuntu* maka dapat mengeksekusi perintah:

   ```
   $ docker volume ls
   ```

   ```
   belajar@sabarmenanti:~$ docker volume ls
   DRIVER    VOLUME NAME
   local     2bd8310f1e313bc0d06f8c75fd8a25fab56c6385d1f6b8417799ef3f63a28c51
   ```

   Terlihat terdapat satu *volume* dengan nama "**2bd8310f1e313bc0d06f8c75fd8a25fab56c6385d1f6b8417799ef3f63a28c51**" dan menggunakan *volume driver* "**local**".

2. Membuat *docker volume* baru yang dapat digunakan untuk menyimpan data dengan sintak penulisan perintah:

   ```
   docker volume create [OPTIONS] VOLUME
   ```

   Penjelasan salah satu *options*:

`-d, --driver` digunakan untuk menentukan jenis *volume driver* yang digunakan untuk mengelola jaringan yang dibuat. Secara *default* bertipe *local*.

Argumen `VOLUME` merupakan nama dari *volume* baru yang akan dibuat.

Sebagai contoh untuk membuat *docker volume* baru bernama **vol-belajar** maka dapat mengeksekusi perintah:

```
$ docker volume create vol-belajar
```

```
belajar@sabarmenanti:~$ docker volume create vol-belajar
vol-belajar
```

Terlihat *output nama volume* yang dibuat sebagai hasil eksekusi perintah tersebut.

3. Memverifikasi hasil dari pembuatan *docker volume* baru bernama **vol-belajar**.

```
$ docker volume ls
```

```
belajar@sabarmenanti:~$ docker volume ls
DRIVER    VOLUME NAME
local     2bd8310f1e313bc0d06f8c75fd8a25fab56c6385d1f6b8417799ef3f63a28c51
local     vol-belajar
```

Terlihat telah terdapat *docker volume* bernama **vol-belajar** dengan *volume driver* bertipe **local**.

4. Membuat *docker container* baru yang memanfaatkan *docker volume* yang telah dibuat dengan ketentuan sebagai berikut:
   - *Container* berjalan di *background*.
   - Nama *container* adalah **mariadb**.
   - *Container* terhubung ke *docker network* **belajar**.
   - *Container* menggunakan *volume* **vol-belajar** dan *volume* tersebut dipasangkan (*mounting*) pada *container* ke direktori `/var/lib/mysql`.
   - *Environment variable* `MARIADB_ROOT_PASSWORD` bernilai `sabarmenanti`
   - Merutekan trafik dari **port 3306** di mesin **host** ke **port 3306** juga di dalam *container*.
   - *Container image* menggunakan **mariadb** dengan **tag "latest"**.

   Sebelum mengeksekusi perintah untuk membuat *container* dengan ketentuan tersebut maka akan dilakukan pembuatan *docker network* bernama **belajar** menggunakan perintah:

```
$ docker network create belajar
```

Selanjutnya dapat mengeksekusi perintah untuk membuat *container*:

```
$ docker run -d --name mariadb --network belajar \
-v vol-belajar:/var/lib/mysql \
--env MARIADB_ROOT_PASSWORD=sabarmenanti \
-p 3306:3306 mariadb:latest
```

```
belajar@sabarmenanti:~$ docker run -d --name mariadb --network belajar \
> -v vol-belajar:/var/lib/mysql \
> --env MARIADB_ROOT_PASSWORD=sabarmenanti \
> -p 3306:3306 mariadb:latest
e7cd4ff10351dc7ce20ad2eba4ae3cc84e2c8aae27acd8a0a0a6743c4c8206c4
```

Terlihat informasi *container* ID dari *container* yang dibuat yaitu "**e7cd4ff10351dc7ce20ad2eba4ae3cc84e2c8aae27acd8a0a0a6743c4c8206c4**".

5. Memverifikasi *docker container* bernama "**mariadb**" telah berjalan.

   ```
   $ docker ps
   ```

```
belajar@sabarmenanti:~$ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS          PORTS                                       NAMES
e7cd4ff10351   mariadb:latest   "docker-entrypoint.s…"   20 seconds ago   Up 19 seconds   0.0.0.0:3306->3306/tcp, :::3306->3306/tcp   mariadb
82c97d4be9e2   nginx:latest     "/docker-entrypoint.…"   45 minutes ago   Up 45 minutes   0.0.0.0:80->80/tcp, :::80->80/tcp           nginx
```

   Terlihat *container* bernama "**mariadb**" telah berhasil dibuat menggunakan *image* **mariadb:latest** pada 20 detik yang lalu berdasarkan informasi pada kolom **CREATED** dan telah aktif 19 detik yang lalu (**Up 19 seconds**) berdasarkan informasi pada kolom **STATUS**. Selain itu juga terlihat informasi *port* yang digunakan oleh *container* yaitu **3306** pada *host* akan dirutekan ke port **3306/tcp** di dalam *container* sesuai dengan nilai pada kolom **PORTS**.

6. Menginspeksi *docker container* tertentu terkait *docker volume* yang digunakan dengan sintak penulisan perintah:

   ```
   docker inspect [OPTIONS] CONTAINER
   ```

   Penjelasan dari salah satu *options* yang umum digunakan:

   `-f, --format` digunakan untuk memformat *output* menggunakan *Go template*.

   Argumen `CONTAINER` memuat nama *container* atau *container* ID yang akan diinspeksi. Secara *default docker inspect* akan melakukan *render* hasil dalam *JSON array*.

   Sebagai contoh untuk menginspeksi *docker container* **mariadb** terkait *volume* yang digunakan dan *path* atau lokasi *mounting* di dalam *container* maka dapat mengeksekusi perintah:

```
$ docker inspect \
-f '{{ range .Mounts }}{{ .Name }}:{{ .Destination }} {{ end }}' \
mariadb
```

```
belajar@sabarmenanti:~$ docker inspect \
> -f '{{ range .Mounts }}{{ .Name }}:{{ .Destination }} {{ end }}' \
> mariadb
vol-belajar:/var/lib/mysql
```

Terlihat informasi bahwa *docker container* **mariadb** menggunakan *volume* **vol-belajar** yang di *mounting* ke direktori **/var/lib/mysql** di dalam *container*.

7. Memverifikasi koneksi ke *docker container* **mariadb** dari *host machine* menggunakan utilitas *mysql client* dan ujicoba membuat *database* bernama "**perpustakaan**" di *container* tersebut. Selain itu juga akan dilakukan ujicoba pembuatan sebuah **tabel** bernama "**buku**" di dalam *database* "**perpustakaan**" tersebut dengan struktur dan konten tabel seperti terlihat pada tabel berikut:

kode	judul	pengarang
001	Docker Deep Dive	Nigel Poulton
002	The Kubernetes Book	Nigel Poulton

a. Menginstalasi paket aplikasi *mariadb-client* agar tersedia utilitas *mysql client* di *host machine*.

```
$ sudo apt -y install mariadb-client
```

b. Mengakses *server mariadb* pada *docker container* menggunakan utilitas *mysql client* dengan parameter-parameter berikut:

- `-h` untuk mengatur hostname dari server mariadb yang akan diakses yaitu **localhost**.
- `-u` untuk mengatur nama login pengguna yang digunakan untuk mengakses ke *server mariadb* yaitu **root**.
- `-p` digunakan untuk mengatur inputan sandi untuk nama login pengguna.
- `-P` digunakan untuk mengatur nomor *port* dari *server mariadb* yaitu **3306**.

Sehingga perintahnya terlihat seperti berikut:

```
$ mysql -h localhost -u root -p -P 3306
```

Tampil inputan **Enter password**, masukkan `sabarmenanti` & tekan **Enter**.

```
belajar@sabarmenanti:~$ mysql -h localhost -u root -p -P 3306
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 6
Server version: 11.4.2-MariaDB-ubu2404 mariadb.org binary distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]>
```

Terlihat koneksi ke *server mariadb* berhasil dilakukan ditandai dengan penampilan *prompt* `MariaDB [(none)]>`

Perhatian: Apabila muncul pesan kesalahan "`ERROR 1045 (28000): Access denied for user 'root'@'localhost' (using password: YES)`" ketika mengakses *container* **mariadb** menggunakan utilitas *mysql* maka lakukan penghentian dan penghapusan mariadb terlebih dahulu dengan mengeksekusi perintah:

```
$ docker rm -f mariadb
```

Selanjutnya lakukan pembuatan ulang *container* **mariadb** dengan mengeksekusi perintah berikut:

```
$ docker run -d --name mariadb --network belajar \
-v vol-belajar:/var/lib/mysql \
--env MARIADB_ROOT_PASSWORD=sabarmenanti \
-p 127.0.0.1:3306:3306 mariadb:latest
```

Terlihat terdapat tambahan nilai **127.0.0.1** saat melakukan *publish* atau mengekspos port dengan option `-p`.

c. Menampilkan informasi *database* yang telah ada di *server mariadb*.

```
show databases;
```

```
MariaDB [(none)]> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
4 rows in set (0,001 sec)
```

Terlihat terdapat 4 (empat) *database* yaitu *information_schema*, *mysql*, *performance_schema* dan *sys*.

d. Membuat *database* bernama **perpustakaan**.

```
create database perpustakaan;
```

```
MariaDB [(none)]> create database perpustakaan;
Query OK, 1 row affected (0,001 sec)
```

e. Mengatur agar *database* **perpustakaan** sebagai *database* aktif dan melihat tabel-tabel yang terdapat pada *database* aktif tersebut.

```
use perpustakaan;
show tables;
```

```
MariaDB [(none)]> use perpustakaan;
Database changed
MariaDB [perpustakaan]> show tables;
Empty set (0,001 sec)
```

Terlihat belum terdapat tabel pada *database* **perpustakaan**.

f. Membuat tabel **buku** dengan struktur yang telah ditentukan sebelumnya.

```
create table buku (
kode char(3) not null,
judul varchar(100) not null,
pengarang varchar(50) not null,
primary key (kode));
```

```
MariaDB [perpustakaan]> create table buku (
    -> kode char(3) not null,
    -> judul varchar(100) not null,
    -> pengarang varchar(50) not null,
    -> primary key (kode));
Query OK, 0 rows affected (0,025 sec)
```

g. Menampilkan informasi tabel-tabel yang terdapat di database **perpustakaan**.

```
show tables;
```

```
MariaDB [perpustakaan]> show tables;
+------------------------+
| Tables_in_perpustakaan |
+------------------------+
| buku                   |
+------------------------+
1 row in set (0,001 sec)
```

Terlihat telah terdapat tabel **buku**.

h. Menampilkan struktur dari tabel **buku**.

```
desc buku;
```

```
MariaDB [perpustakaan]> desc buku;
+-----------+--------------+------+-----+---------+-------+
| Field     | Type         | Null | Key | Default | Extra |
+-----------+--------------+------+-----+---------+-------+
| kode      | char(3)      | NO   | PRI | NULL    |       |
| judul     | varchar(100) | NO   |     | NULL    |       |
| pengarang | varchar(50)  | NO   |     | NULL    |       |
+-----------+--------------+------+-----+---------+-------+
3 rows in set (0,002 sec)
```

i. Menambah 2 (dua) data buku ke dalam tabel **buku**.

```
insert into buku values ('001','Docker Deep Dive','Nigel Poulton');
insert into buku values ('002','The Kubernetes Book','Nigel Poulton');
```

```
MariaDB [perpustakaan]> insert into buku
    -> values ('001','Docker Deep Dive','Nigel Poulton');
Query OK, 1 row affected (0,011 sec)

MariaDB [perpustakaan]> insert into buku
    -> values ('002','The Kubernetes Book','Nigel Poulton');
Query OK, 1 row affected (0,011 sec)
```

j. Menampilkan isi dari tabel **buku**.

```
select * from buku;
```

```
MariaDB [perpustakaan]> select * from buku;
+------+---------------------+---------------+
| kode | judul               | pengarang     |
+------+---------------------+---------------+
| 001  | Docker Deep Dive    | Nigel Poulton |
| 002  | The Kubernetes Book | Nigel Poulton |
+------+---------------------+---------------+
2 rows in set (0,001 sec)
```

k. Keluar dari *server mariadb*.

```
quit;
```

```
MariaDB [perpustakaan]> quit;
Bye
```

8. Menghentikan dan menghapus *docker container* **mariadb**.

```
$ docker rm -f mariadb
```

9. Memverifikasi penghentian dan penghapusan *docker container* **mariadb**.

```
$ docker ps -a
```

```
belajar@sabarmenanti:~$ docker ps -a
CONTAINER ID   IMAGE         COMMAND                  CREATED        STATUS                    PORTS                                   NAMES
82c97d4be9e2   nginx:latest  "/docker-entrypoint.…"   3 hours ago    Up 3 hours                0.0.0.0:80->80/tcp, :::80->80/tcp       nginx
4e89ae4978b6   hello-world   "/hello"                 45 hours ago   Exited (0) 45 hours ago                                           blissful_williamson
5f80c3d17fdd   hello-world   "/hello"                 46 hours ago   Exited (0) 46 hours ago                                           sharp_saha
```

Sudah tidak terlihat *docker container* dengan nama tersebut sehingga penghentian dan penghapusan *container* telah berhasil dilakukan.

10. Membuat kembali *docker container* **mariadb** yang memanfaatkan *docker volume* **vol-belajar** sebelumnya. Hal ini untuk memverifikasi bahwa *database* **perpustakaan** dan tabel **buku** serta konten di dalam tabel tersimpan secara permanen pada *volume* **vol-belajar** meskipun *container* telah dihentikan dan dihapus sebelumnya.

```
$ docker run -d --name mariadb --network belajar \
-v vol-belajar:/var/lib/mysql \
--env MARIADB_ROOT_PASSWORD=sabarmenanti \
-p 127.0.0.1:3306:3306 mariadb:latest
```

Perhatian: Apabila muncul pesan kesalahan "`ERROR 1045 (28000): Access denied for user 'root'@'localhost' (using password: YES)`" ketika mengakses *container* **mariadb** menggunakan utilitas *mysql* maka lakukan penghentian dan penghapusan mariadb terlebih dahulu dengan mengeksekusi perintah:

```
$ docker rm -f mariadb
```

Selanjutnya lakukan pembuatan ulang *container* **mariadb** dengan mengeksekusi perintah berikut:

```
$ docker run -d --name mariadb --network belajar \
-v vol-belajar:/var/lib/mysql \
--env MARIADB_ROOT_PASSWORD=sabarmenanti \
-p 127.0.0.1:3306:3306 mariadb:latest
```

Terlihat terdapat tambahan nilai **127.0.0.1** saat melakukan *publish* atau mengekspos port dengan option `-p`.

11. Memverifikasi *docker container* bernama "**mariadb**" telah berjalan.

```
$ docker ps
```

```
belajar@sabarmenanti:~$ docker ps
CONTAINER ID   IMAGE           COMMAND                  CREATED         STATUS         PORTS                                      NAMES
17abca2f2ce3   mariadb:latest  "docker-entrypoint.s…"   3 minutes ago   Up 3 minutes   0.0.0.0:3306->3306/tcp, :::3306->3306/tcp  mariadb
82c97d4be9e2   nginx:latest    "/docker-entrypoint.…"   3 hours ago     Up 3 hours     0.0.0.0:80->80/tcp, :::80->80/tcp          nginx
```

12. Mengakses *server mariadb* pada *docker container* menggunakan utilitas *mysql client*.

```
$ mysql -h localhost -u root -p -P 3306
```

Tampil inputan **Enter password**, masukkan `sabarmenanti` & tekan **Enter**.

```
belajar@sabarmenanti:~$ mysql -h localhost -u root -p -P 3306
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 6
Server version: 11.4.2-MariaDB-ubu2404 mariadb.org binary distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]>
```

Terlihat koneksi ke *server mariadb* berhasil dilakukan ditandai dengan penampilan *prompt* `MariaDB [(none)]>`

13. Menampilkan informasi *database* yang telah ada di *server mariadb*.

    ```
    show databases;
    ```

```
MariaDB [(none)]> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| perpustakaan       |
| sys                |
+--------------------+
5 rows in set (0,001 sec)
```

Terlihat masih terdapat *database* **perpustakaan**.

14. Mengatur agar *database* **perpustakaan** sebagai *database* aktif dan melihat tabel-tabel yang terdapat pada *database* aktif tersebut.

    ```
    use perpustakaan;
    show tables;
    ```

```
MariaDB [(none)]> use perpustakaan;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
MariaDB [perpustakaan]> show tables;
+------------------------+
| Tables_in_perpustakaan |
+------------------------+
| buku                   |
+------------------------+
1 row in set (0,001 sec)
```

15. Menampilkan isi dari tabel **buku**.

    ```
    select * from buku;
    ```

```
MariaDB [perpustakaan]> select * from buku;
+------+---------------------+---------------+
| kode | judul               | pengarang     |
+------+---------------------+---------------+
| 001  | Docker Deep Dive    | Nigel Poulton |
| 002  | The Kubernetes Book | Nigel Poulton |
+------+---------------------+---------------+
2 rows in set (0,001 sec)
```

16. Keluar dari *server mariadb*.

    ```
    quit;
    ```

    ```
    MariaDB [perpustakaan]> quit;
    Bye
    ```

17. Menghentikan dan menghapus *docker container* **mariadb**.

    ```
    $ docker rm -f mariadb
    ```

18. Memverifikasi penghentian dan penghapusan *docker container* **mariadb**.

    ```
    $ docker ps -a
    ```

    ```
    belajar@sabarmenanti:~$ docker ps -a
    CONTAINER ID   IMAGE         COMMAND                  CREATED        STATUS                     PORTS                                   NAMES
    82c97d4be9e2   nginx:latest  "/docker-entrypoint.…"   3 hours ago    Up 3 hours                 0.0.0.0:80->80/tcp, :::80->80/tcp       nginx
    4e89ae4978b6   hello-world   "/hello"                 46 hours ago   Exited (0) 46 hours ago                                            blissful_williams
    5f80c3d17fdd   hello-world   "/hello"                 46 hours ago   Exited (0) 46 hours ago                                            sharp_saha
    ```

    Sudah tidak terlihat *docker container* dengan nama tersebut sehingga penghentian dan penghapusan *container* telah berhasil dilakukan.

19. Menghapus *docker volume* tertentu dengan sintak penulisan perintah:

    ```
    docker volume rm VOLUME
    ```

    Argumen `VOLUME` merupakan nama dari *volume* yang akan dihapus. *Volume* tidak akan dapat dihapus apabila masih digunakan oleh suatu *container*. Sebelum menghapus maka diperlukan penghentian dan penghapusan *container* yang menggunakan ke *volume* tersebut.

    Perintah ini juga memiliki *alias*:

    ```
    docker volume remove
    ```

    Sebagai contoh untuk menghapus *docker volume* bernama "**vol-belajar**" maka dapat mengeksekusi perintah:

    ```
    $ docker volume rm vol-belajar
    ```

    ```
    belajar@sabarmenanti:~$ docker volume rm vol-belajar
    vol-belajar
    ```

    Terlihat informasi *nama* dari *docker volume* yang telah berhasil dihapus.

20. Memverifikasi hasil dari penghapusan *docker volume* bernama **vol-belajar**.

```
$ docker volume ls
```

```
belajar@sabarmenanti:~$ docker volume ls
DRIVER    VOLUME NAME
local     2bd8310f1e313bc0d06f8c75fd8a25fab56c6385d1f6b8417799ef3f63a28c51
```

Sudah tidak terlihat *docker volume* dengan nama tersebut sehingga penghapusan *volume* telah berhasil dilakukan.

H. Docker Prune

Docker menyediakan mekanisme untuk menghapus keseluruhan *container*, *network*, *images* dan *volume* yang sudah tidak digunakan yaitu dengan mengeksekusi perintah `docker system prune`. Namun apabila ingin menghapus objek tertentu dari *docker* maka dapat mengeksekusi perintah-perintah berikut:

a. Menghapus seluruh *docker container* yang telah dihentikan:
   `docker container prune [OPTIONS]`
b. Menghapus seluruh *docker network* yang tidak digunakan:
   `docker network prune [OPTIONS]`
c. Menghapus seluruh *docker images* yang tidak diberi *tag* dan tidak terkait dengan *container* manapun (*dangling images*). *Dangling images* dibuat ketika ditimpa oleh *image* baru dengan nama dan *tag* yang sama.
   `docker image prune [OPTIONS]`
   Jika *options* `-a` dicantumkan maka akan menghapus seluruh *images* yang tidak dirujuk oleh *container* manapun.
d. Menghapus seluruh *docker volume* local yang tidak digunakan:
   `docker volume prune [OPTIONS]`
   Jika *options* `-a` dicantumkan maka akan menghapus tidak hanya *anonymous volume* namun juga seluruh *volume* yang tidak digunakan atau dirujuk oleh *container* manapun. *Volume* berjenis *anonymous* karena tidak memiliki sumber atau nama spesifik.

Perintah-perintah penghapusan setiap obyek tersebut memiliki satu *options* yang umum digunakan yaitu:

`-f, --force` digunakan agar tidak memunculkan *prompt* konfirmasi terkait operasi penghapusan yang akan dilakukan.

Menghapus seluruh *docker container* yang telah dihentikan

Sebagai contoh sebelum menghapus seluruh *docker container* yang telah dihentikan maka dilakukan verifikasi terlebih dahulu dengan melihat baik *container* yang sedang berjalan atau berhenti dengan mengeksekusi perintah:

```
docker ps -a
```

```
belajar@sabarmenanti:~$ docker ps -a
CONTAINER ID   IMAGE         COMMAND                  CREATED        STATUS                    PORTS                                   NAMES
82c97d4be9e2   nginx:latest  "/docker-entrypoint.…"   3 hours ago    Up 3 hours                0.0.0.0:80->80/tcp, :::80->80/tcp       nginx
4e89ae4978b6   hello-world   "/hello"                 46 hours ago   Exited (0) 46 hours ago                                           blissful_williamson
5f80c3d17fdd   hello-world   "/hello"                 46 hours ago   Exited (0) 46 hours ago                                           sharp_saha
```

Terlihat terdapat satu *container* bernama **"nginx"** yang masih berjalan (*running*) dan 2 (dua) *container* bernama "**blissful_williamson**" dan "**sharp_saha**" yang telah berhenti. Selanjutnya untuk menghapus seluruh *container* yang telah dihentikan tanpa konfirmasi maka lakukan eksekusi perintah berikut:

```
docker container prune -f
```

```
belajar@sabarmenanti:~$ docker container prune -f
Deleted Containers:
4e89ae4978b65ed9fe4bad48cbaf7765806bb889480a47e189151baf4b88895e
5f80c3d17fdd2dd62cc8a6c85cacceec80604db028f0fb8985d278b087fc9280

Total reclaimed space: 0B
```

Terlihat informasi 2 (dua) *container ID* dari *docker container* yang dihapus. *Container ID* tersebut merupakan milik dari *container* bernama "**blissful_williamson**" dan "**sharp_saha**" yang telah diidentifikasi sebelumnya.

Terakhir lakukan verifikasi kembali keberhasilan penghapusan *docker container* yang terhenti tersebut dengan mengeksekusi perintah:

```
docker ps -a
```

```
belajar@sabarmenanti:~$ docker ps -a
CONTAINER ID   IMAGE         COMMAND                  CREATED        STATUS        PORTS                                   NAMES
82c97d4be9e2   nginx:latest  "/docker-entrypoint.…"   4 hours ago    Up 4 hours    0.0.0.0:80->80/tcp, :::80->80/tcp       nginx
```

Terlihat informasi hanya menyisakan satu *container* bernama **nginx** yang sedang berjalan (*running*). Dua *container* terhenti lainnya telah berhasil dihapus.

Menghapus seluruh *docker network* yang tidak digunakan

Sebagai contoh sebelum menghapus seluruh *docker network* yang tidak digunakan maka dilakukan verifikasi terlebih dahulu dengan melihat *docker network* yang dimiliki dengan mengeksekusi perintah:

```
docker network ls
```

```
belajar@sabarmenanti:~$ docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
0d75c527773b   belajar   bridge    local
d7e0bdcd9a10   bridge    bridge    local
5aac6e2179b1   host      host      local
6ec967d4f726   none      null      local
```

Terlihat masih terdapat 4 (empat) *docker network*. Selanjutnya untuk menghapus seluruh *docker network* yang tidak digunakan tanpa konfirmasi maka lakukan eksekusi perintah berikut:

```
docker network prune -f
```

```
belajar@sabarmenanti:~$ docker network prune -f
Deleted Networks:
belajar
```

Terlihat informasi satu *docker network* bernama **belajar** yang dihapus.

Terakhir lakukan verifikasi kembali keberhasilan penghapusan *docker network* yang tidak digunakan tersebut dengan mengeksekusi perintah:

```
docker network ls
```

```
belajar@sabarmenanti:~$ docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
d7e0bdcd9a10   bridge    bridge    local
5aac6e2179b1   host      host      local
6ec967d4f726   none      null      local
```

Terlihat informasi hanya menyisakan 3 (tiga) *docker network*. Satu *docker network* terhenti lainnya telah berhasil dihapus.

Menghapus seluruh *docker images* yang *dangling* dan tidak digunakan

Sebagai contoh sebelum menghapus seluruh *docker images* yang *dangling* maka dilakukan verifikasi terlebih dahulu dengan melihat *images* yang dimiliki dengan mengeksekusi perintah:

```
docker image ls
```

```
belajar@sabarmenanti:~$ docker image ls
REPOSITORY    TAG       IMAGE ID       CREATED          SIZE
nginx         latest    e0c9858e10ed   10 days ago      188MB
alpine        latest    a606584aa9aa   10 days ago      7.8MB
mariadb       latest    4486d64c9c3b   2 weeks ago      406MB
hello-world   latest    d2c94e258dcb   14 months ago    13.3kB
```

Terlihat terdapat 4 (empat) *docker images*. Selanjutnya untuk menghapus seluruh *dangling images* tanpa konfirmasi maka lakukan eksekusi perintah berikut:

```
docker image prune -f
```

```
belajar@sabarmenanti:~$ docker image prune -f
Total reclaimed space: 0B
```

Terlihat *output* "**Total reclaimed space: 0B**" yang bermakna tidak ada *dangling image* yang dihapus. Sedangkan untuk menghapus seluruh *images* yang tidak dirujuk oleh container lainnya maka lakukan eksekusi perintah berikut:

```
docker image prune -a -f
```

```
belajar@sabarmenanti:~$ docker image prune -a -f
Deleted Images:
untagged: mariadb:latest
untagged: mariadb@sha256:e59ba8783bf7bc02a4779f103bb0d8751ac0e10f9471089709608377eded7aa8
deleted: sha256:4486d64c9c3b538b6dee6bcd5ea0ac5f74ea5e3cafc71181a54ec678ae0370aa
deleted: sha256:cb7bf164331273be5cf25fba961039469981c740cf400fd329f4b2c3b89eaec7
deleted: sha256:f4facb5595424cf6d659f3694f942bae69465c0057291a758be53db13088e272
deleted: sha256:ea6b69c01bfabfe3407970d9484f8433262963f2297a59fab36a5ef6f462a2b6
deleted: sha256:f7dcbe3243635dab2018bdfd5e129205e664d4d1d9144374c397d06be5eedcb6
deleted: sha256:20983f7bf487687cfda9e1d196ea71169ad5548b7505ca718398bb52adcd16ba
deleted: sha256:b4e7d1e2addfb8586fd055eef9596123ea590f4c23fd04fb6b2690bb028fb184
deleted: sha256:ba0ce632a4a21ac60bf66c3313a25452e5b781119bf4daeec0cdd281467c07c0
deleted: sha256:a30a5965a4f7d9d5ff76a46eb8939f58e95be844de1ac4a4b452d5d31158fdea
untagged: alpine:latest
untagged: alpine@sha256:b89d9c93e9ed3597455c90a0b88a8bbb5cb7188438f70953fede212a0c4394e0
deleted: sha256:a606584aa9aa875552092ec9e1d62cb98d486f51f389609914039aabd9414687
deleted: sha256:94e5f06ff8e3d4441dc3cd8b090ff38dc911bfa8ebdb0dc28395bc98f82f983f
untagged: hello-world:latest
untagged: hello-world@sha256:94323f3e5e09a8b9515d74337010375a456c909543e1ff1538f5116d38ab3989
deleted: sha256:d2c94e258dcb3c5ac2798d32e1249e42ef01cba4841c2234249495f87264ac5a
deleted: sha256:ac28800ec8bb38d5c35b49d45a6ac4777544941199075dff8c4eb63e093aa81e

Total reclaimed space: 413.6MB
```

Terlihat informasi 3 (tiga) *images* yang dihapus meliputi **mariadb:latest**, **alpine:latest** dan **hello-world:latest**. Selain itu juga terlihat *output* "**Total reclaimed space: 413.6MB**" yang bermakna terdapat sebesar **413.6 MB** ruang penyimpanan yang dapat dimanfaatkan kembali.

Terakhir lakukan verifikasi kembali keberhasilan penghapusan *docker images* tersebut dengan mengeksekusi perintah:

```
docker image ls
```

```
belajar@sabarmenanti:~$ docker image ls
REPOSITORY   TAG       IMAGE ID       CREATED       SIZE
nginx        latest    e0c9858e10ed   10 days ago   188MB
```

Terlihat informasi hanya menyisakan satu *images* **nginx:latest** yang tidak dihapus karena masih digunakan oleh *container* **nginx** yang sedang berjalan. Sedangkan 3 (tiga) *images* yang tidak digunakan lainnya telah berhasil dihapus.

Menghapus seluruh *volume* local yang tidak digunakan

Sebagai contoh sebelum menghapus seluruh *docker volume* yang telah dihentikan maka dilakukan verifikasi terlebih dahulu dengan melihat *volume* yang dimiliki dengan mengeksekusi perintah:

```
docker volume ls
```

```
belajar@sabarmenanti:~$ docker volume ls
DRIVER    VOLUME NAME
local     2bd8310f1e313bc0d06f8c75fd8a25fab56c6385d1f6b8417799ef3f63a28c51
```

Terlihat terdapat satu *volume* bernama **"2bd8310f1e313bc0d06f8c75fd8a25fab56c6385d1f6b8417799ef3f63a28c51"** dengan *volume driver* bertipe "**local**".

Untuk mendukung kebutuhan ujicoba simulasi penghapusan *volume* yang tidak digunakan maka terlebih dahulu akan dilakukan pembuatan 2 (dua) *volume* baru yaitu masing-masing bernama "**vol-dokumen**" dan "**vol-database**" dengan mengeksekusi perintah:

```
docker volume create vol-dokumen
```

```
docker volume create vol-database
```

```
belajar@sabarmenanti:~$ docker volume create vol-dokumen
vol-dokumen
belajar@sabarmenanti:~$ docker volume create vol-database
vol-database
```

Lakukan verifikasi pembuatan dua *volume* tersebut dengan mengeksekusi perintah:

```
docker volume ls
```

```
belajar@sabarmenanti:~$ docker volume ls
DRIVER    VOLUME NAME
local     2bd8310f1e313bc0d06f8c75fd8a25fab56c6385d1f6b8417799ef3f63a28c51
local     vol-database
local     vol-dokumen
```

Terlihat telah terdapat *volume* bernama **vol-dokumen** dan **vol-database**.

Selanjutnya untuk menghapus seluruh *volume* yang tidak digunakan tanpa menggunakan *option* `-f` sehingga akan menampilkan konfirmasi dengan mengeksekusi perintah berikut:

```
docker volume prune
```

```
belajar@sabarmenanti:~$ docker volume prune
WARNING! This will remove anonymous local volumes not used by at least one container.
Are you sure you want to continue? [y/N] y
Deleted Volumes:
2bd8310f1e313bc0d06f8c75fd8a25fab56c6385d1f6b8417799ef3f63a28c51

Total reclaimed space: 168.6MB
```

Tampil *prompt* yang mengkonfirmasi penghapusan *anonymous local volumes*. Ketik **y** dan tekan **Enter** untuk melanjutnya operasi penghapusan *volume* tersebut. Terlihat informasi *volume* yang berhasil dihapus memiliki nama "**2bd8310f1e313bc0d06f8c75fd8a25fab56c6385d1f6b8417799ef3f63a28c51**" yang merupakan *volume* dengan *driver* bertipe **local** dan merupakan *anonymous volume*.

Lakukan verifikasi kembali keberhasilan penghapusan *anonymous volume* yang tidak digunakan tersebut dengan mengeksekusi perintah:

```
docker volume ls
```

```
belajar@sabarmenanti:~$ docker volume ls
DRIVER    VOLUME NAME
local     vol-database
local     vol-dokumen
```

Terlihat informasi hanya tersisa 2 (dua) *volume* yang belum dihapus.

Selanjutnya lakukan penghapusan seluruh volume yang tidak digunakan dengan mengeksekusi perintah:

```
docker volume prune -a
```

```
belajar@sabarmenanti:~$ docker volume prune -a
WARNING! This will remove all local volumes not used by at least one container.
Are you sure you want to continue? [y/N] y
Deleted Volumes:
vol-database
vol-dokumen

Total reclaimed space: 0B
```

Tampil *prompt* yang mengkonfirmasi penghapusan seluruh *local volumes* yang tidak digunakan oleh *container* mana pun. Ketik **y** dan tekan **Enter** untuk melanjutnya operasi penghapusan *volume* tersebut. Terlihat informasi *volume* yang berhasil dihapus memiliki Bernama "**vol-database**" dan "**vol-dokumen**".

Terakhir lakukan verifikasi kembali keberhasilan penghapusan *volume* yang tidak digunakan tersebut dengan mengeksekusi perintah:

```
docker volume ls
```

```
belajar@sabarmenanti:~$ docker volume ls
DRIVER    VOLUME NAME
```

Keseluruhan *volume* telah terhapus.

BAB V

MEMBANGUN CONTAINER IMAGES MENGGUNAKAN DOCKERFILE

Dockerfile merupakan *file text* yang didalamnya memuat sekumpulan instruksi untuk membangun *docker container images*. Format instruksi di dalam *Dockerfile* adalah:

```
# Komentar

INSTRUKSI argumen
```

Penjelasan:

- `#` digunakan untuk menambahkan komentar
- `INSTRUKSI` tidak bersifat *case-sensitive* namun disarankan menuliskan menggunakan huruf kapital untuk kemudahan dalam membedakan dengan argumen.

Instruksi-instruksi yang didukung pada *Dockerfile*, seperti terlihat pada tabel berikut:

Instruksi	Deskripsi
`FROM`	Digunakan untuk mengatur *base image* dari *container image* yang akan dihasilkan. Instruksi ini akan membuat *build stage* dari *base image* yang dicantumkan sebagai argumen dengan format penulisan **image:tag**.
`RUN`	Digunakan untuk mengeksekusi perintah di dalam *container* ketika *build stage*.
`CMD`	Digunakan untuk mengatur perintah yang akan dieksekusi ketika *container* dijalankan dan tidak akan dijalankan ketika proses *build*.
`LABEL`	Digunakan untuk menambahkan metadata pada *image* yang dituliskan dalam format pasangan **key=value**. Metadata dapat berupa deskripsi terkait *image* dan lain-lain.
`ADD`	Digunakan untuk menambahkan *file* dari sumber tertentu ke dalam *docker image* dengan fungsionalitas tambahan seperti

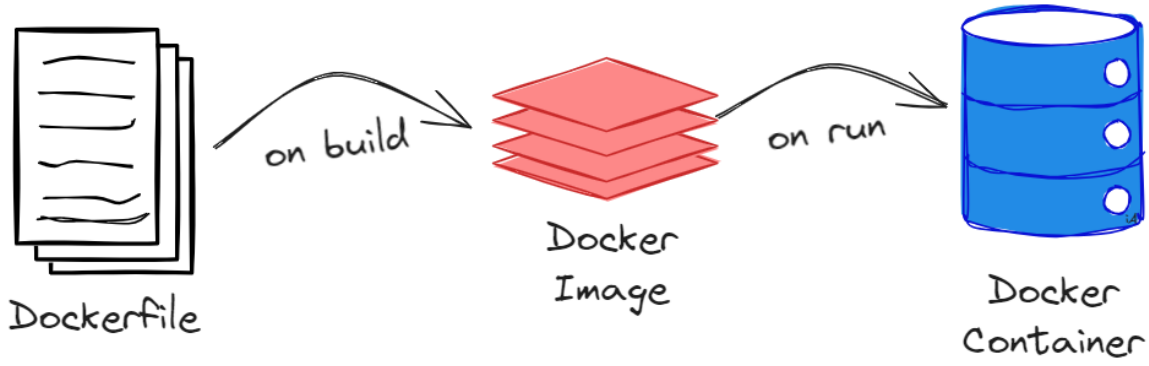
	mengekstrak arsip berformat *tar* dan menyalinkan file yang bersumber dari *Uniform Resource Locator (URL)* tertentu.
`COPY`	Digunakan untuk menyalinkan *file* dan direktori dari *host* ke dalam *container image* yang dihasilkan.
`EXPOSE`	Digunakan untuk menjelaskan bahwa container akan melakukan *listen* pada nomor *port* dan *protocol* tertentu yang digunakan hanya sebagai informasi atau dokumentasi.
`ENV`	Digunakan untuk mengatur *environment variable*.
`VOLUME`	Digunakan untuk membuat titik akses (*mount point*) dengan nama yang ditentukan pada bagian argumen dari instruksi tersebut dan menandainya sebagai tempat menampung *volume* yang dipasang secara eksternal di *host machine* atau *container* lain.
`WORKDIR`	Digunakan untuk mengatur *current working directory* di dalam *container* sebagai lokasi perintah-perintah dieksekusi.
`USER`	Digunakan untuk mengubah user aktif di dalam *container*.
`ARG`	Digunakan untuk mengatur *variable* saat *build*.
`HEALTHCHECK`	Digunakan untuk mengecek kesehatan dari *container* saat dijalankan (*startup*).
`ENTRYPOINT`	Digunakan untuk mengatur *default executable*.
`MAINTAINER`	Digunakan untuk menentukan pembuat *image*.

Contoh konten dari *Dockerfile* adalah sebagai berikut:

```
FROM alpine:latest

RUN mkdir data

RUN echo "Pelatihan Cloud Native" > data/baca.txt

CMD ["cat", "data/baca.txt"]
```

Setiap baris yang terdapat pada *Dockerfile* direpresentasikan sebagai perintah. *Docker* mengeksekusi instruksi-instruksi di dalam *Dockerfile* secara berurutan. Sebuah *Dockerfile* harus diawali dengan instruksi `FROM` yang mengatur *image* induk sebagai titik awal *build stage*. Pembangunan *docker image* dilakukan dengan mengeksekusi perintah *docker build* yang

akan membaca konten pada *Dockerfile* dan menghasilkan sebuah *docker image*. *Docker image* yang telah dihasilkan tersebut selanjutnya dapat digunakan untuk membuat dan menjalankankan *docker container*, seperti terlihat pada gambar berikut:



Sumber gambar: geeksforgeeks.org

A. Instruksi FROM dan RUN pada Dockerfile

Sebagai contoh akan dibuat *container image* menggunakan *base image* "**alpine**" dengan versi terkini atau tag "**latest**". Saat *build stage* dari *image* tersebut akan dibuat direktori bernama "**data**" dan sebuah *file* bernama "**baca.txt**" di dalam direktori tersebut. Konten dari *file* "**baca.txt**" adalah "**Pelatihan Cloud Native**". Nama dari *container image* yang dibuat adalah **sederhana:latest**.

Adapun langkah-langkah pembuatan *container image* dengan ketentuan tersebut dan mengujicoba hasil pembuatannya adalah sebagai berikut:

1. Membuat direktori baru bernama "**images**" sebagai lokasi pembuatan **Dockerfile**.

   ```
   $ mkdir images
   ```

2. Berpindah ke dalam direktori "**images**".

   ```
   $ cd images
   ```

3. Membuat *file* bernama **Dockerfile** menggunakan *editor* **nano**.

   ```
   $ nano Dockerfile
   ```

   Instruksi `FROM` memiliki sintak penulisan:

   ```
   FROM <image>
   ```

   `<image>` merupakan nama dan tag image yang akan digunakan sebagai *base image*.

   Sedangkan instruksi `RUN` memiliki dua bentuk penulisan yaitu:

- `RUN [OPTIONS] <command> …` merupakan bentuk *shell*.
- `RUN [OPTIONS] [ "<command>", … ]` merupakan bentuk *exec*.

Sebagai contoh untuk instruksi `RUN` untuk mengeksekusi perintah yang ditentukan menggunakan bentuk *shell* maka konten dari *file* akan terlihat seperti berikut:

```
FROM alpine:latest
RUN mkdir data
RUN echo "Pelatihan Cloud Native" > data/baca.txt
```

Simpan perubahan dengan menekan tombol **CTRL+O** dan tekan **Enter**.
Keluar dari *editor nano* dengan menekan tombol **CTRL+X**.

4. Membuat *docker container images* dengan sintak penulisan perintah:
   `docker image build [OPTIONS] PATH`
   Penjelasan *options* yang umum digunakan:
   `-f, --file` digunakan untuk mengatur nama dari *file Dockerfile*. Secara *default* adalah **PATH/Dockerfile**.
   `-t, --tag` bersifat opsional dan digunakan untuk memberikan nama dan *tag* dengan format **name:tag**.

   `PATH` adalah argumen berupa lokasi direktori yang menampung *file Dockerfile*. Sebagai contoh jika *Dockerfile* berada di lokasi direktori dimana saat ini berada maka dapat menggunakan tanda titik (.) yang berarti *current working* direktori.

   Perintah ini memiliki alias:
   ```
   docker build
   docker buildx build
   docker builder build
   ```
   Mengeksekusi pembuatan *container images* dengan nama dan *tag* **sederhana:latest** dengan perintah:
   ```
   $ docker build -t sederhana:latest .
   ```
5. Memverifikasi hasil pembuatan *container images*.

```
belajar@sabarmenanti:~/images$ docker image ls
REPOSITORY    TAG       IMAGE ID       CREATED          SIZE
sederhana     latest    4935ebc9280c   10 hours ago     7.8MB
httpd         latest    c0c20df5e7be   23 hours ago     148MB
nginx         latest    fffffc90d343   13 days ago      188MB
hello-world   latest    d2c94e258dcb   14 months ago    13.3kB
```

Terlihat *container image* tersebut telah berhasil dibuat.

6. Ujicoba membuat dan menjalankan *container* menggunakan *images* **sederhana:latest** serta mengeksekusi perintah untuk menampilkan isi dari *file* **baca.txt** yang terdapat di dalam direktori data. Selain itu apabila *container* dihentikan maka akan langsung menghapus *container* dengan memanfaatkan option `--rm` (*remove*) dari perintah `docker run`.

   ```
   $ docker run --rm sederhana:latest cat data/baca.txt
   ```

```
belajar@sabarmenanti:~/images$ docker run --rm sederhana:latest cat data/baca.txt
Pelatihan Cloud Native
```

Terlihat *output* "**Pelatihan Cloud Native**" yang merupakan isi dari *file* "**baca.txt**".

B. Instruksi CMD pada Dockerfile

Menindaklanjuti hasil penyelesaian *container image* di bagian A maka pada bagian ini akan dilakukan penyesuaian pada *file* **Dockerfile** yaitu agar ketika *container* dijalankan dari *image* tersebut akan menampilkan konten dari *file* "**baca.txt**" yang terdapat di direktori "**data**" melalui eksekusi instruksi **CMD**.

Adapun langkah-langkah pembuatan *container image* untuk mengakomodir ketentuan tersebut dan mengujicoba hasil pembuatannya adalah sebagai berikut:

1. Mengubah konten *file* **Dockerfile** menggunakan *editor* **nano**.

   ```
   $ nano Dockerfile
   ```

   Menambahkan instruksi **CMD** di baris terakhir yang digunakan untuk mengeksekusi perintah "**cat data/baca.txt**" sehingga konten dari *file* "**baca.txt"** di dalam direktori "**data**" dapat ditampilkan ke layar. Instruksi CMD dapat dinyatakan menggunakan bentuk *shell* atau *exec*, seperti berikut:

   - `CMD ["executable", "parameter1", "parameter2"]` merupakan bentuk *exec*.
   - `CMD ["parameter1", "parameter2"]` merupakan bentuk *exec* sebagai parameter default ke instruksi `ENTRYPOINT`.

- `CMD command parameter1 parameter2` merupakan bentuk *shell*.

Sebagai contoh digunakan bentuk *exec* sehingga hasilnya terlihat seperti berikut:

```
FROM alpine:latest
RUN mkdir data
RUN echo "Pelatihan Cloud Native" > data/baca.txt
CMD ["cat", "data/baca.txt"]
```

Simpan perubahan dengan menekan tombol **CTRL+O** dan tekan **Enter**.

Keluar dari *editor nano* dengan menekan tombol **CTRL+X**.

2. Membuat *docker container images* dengan mengeksekusi perintah:

   ```
   $ docker image build -t sederhana:latest .
   ```

3. Memverifikasi hasil pembuatan *container images*.

```
belajar@sabarmenanti:~/images$ docker image ls
REPOSITORY    TAG       IMAGE ID       CREATED          SIZE
sederhana     latest    21eleeb9e43e   14 hours ago     7.8MB
<none>        <none>    4935ebc9280c   14 hours ago     7.8MB
httpd         latest    c0c20df5e7be   27 hours ago     148MB
nginx         latest    fffffc90d343   13 days ago      188MB
hello-world   latest    d2c94e258dcb   14 months ago    13.3kB
```

Terlihat *container image* tersebut telah berhasil dibuat.

4. Ujicoba membuat dan menjalankan *container* menggunakan *images* **sederhana:latest**. Selain itu apabila *container* dihentikan maka akan langsung menghapus *container* dengan memanfaatkan option `--rm` (*remove*) dari perintah `docker run`.

   ```
   $ docker run --rm sederhana:latest
   ```

```
belajar@sabarmenanti:~/images$ docker run --rm sederhana:latest
Pelatihan Cloud Native
```

Terlihat *output* "**Pelatihan Cloud Native**" yang merupakan isi dari *file* "**baca.txt**".

C. Instruksi LABEL pada Dockerfile

Menindaklanjuti hasil penyelesaian *container image* di bagian B maka pada bagian ini akan dilakukan penyesuaian pada *file* **Dockerfile** untuk menambahkan metadata ke dalam image yang dibuat dengan menggunakan instruksi `LABEL`. Instruksi LABEL memiliki format penulisan:

```
LABEL <key>=<value> <key>=<value> <key>=<value> ...
```

Sebagai contoh metadata yang akan ditambahkan ke dalam *image* yaitu berupa *key* **description** dengan *value* "**Belajar membuat Docker Image**" dan *key* **email** *value* "**putu.hariyadi@universitasbumigora.ac.id**". Silakan menyesuaikan *key* dan *value* yang akan ditambahkan pada *image* sesuai dengan kebutuhan.

Adapun langkah-langkah pembuatan *container image* untuk mengakomodir ketentuan tersebut dan mengujicoba hasil pembuatannya adalah sebagai berikut:

1. Mengubah konten *file* **Dockerfile** menggunakan *editor* **nano**.

   ```
   $ nano Dockerfile
   ```

   Sebagai contoh penambahan metadata dilakukan setelah instruksi `FROM` sehingga hasilnya terlihat seperti berikut:

   ```
   FROM alpine:latest
   LABEL description="Belajar membuat Docker Image"
   LABEL email="putu.hariyadi@universitasbumigora.ac.id"
   RUN mkdir data
   RUN echo "Pelatihan Cloud Native" > data/baca.txt
   CMD ["cat", "data/baca.txt"]
   ```

   Simpan perubahan dengan menekan tombol **CTRL+O** dan tekan **Enter**.

   Keluar dari *editor nano* dengan menekan tombol **CTRL+X**.

2. Membuat *docker container images* dengan mengeksekusi perintah:

   ```
   $ docker image build -t sederhana:latest .
   ```

3. Memverifikasi hasil pembuatan *container images*.

```
belajar@sabarmenanti:~/images$ docker image ls
REPOSITORY    TAG       IMAGE ID       CREATED          SIZE
sederhana     latest    b997caf140ab   14 hours ago     7.8MB
<none>        <none>    4935ebc9280c   14 hours ago     7.8MB
<none>        <none>    21e1eeb9e43e   14 hours ago     7.8MB
httpd         latest    c0c20df5e7be   27 hours ago     148MB
nginx         latest    fffffc90d343   13 days ago      188MB
hello-world   latest    d2c94e258dcb   14 months ago    13.3kB
```

Terlihat *container image* tersebut telah berhasil dibuat.

4. Memverifikasi hasil penambahan metadata pada *container image* tersebut dengan mengeksekusi perintah:

   $ `docker image inspect sederhana:latest`

   Cuplikan output dari hasil eksekusi perintah ini adalah sebagai berikut:

```
            "Labels": {
                "description": "Belajar membuat Docker Image",
                "email": "putu.hariyadi@universitasbumigora.ac.id"
            }
```

   Atau dengan menggunakan pemfilteran sehingga hanya menampilkan metadata maka dapat mengeksekusi perintah:

   `$ docker image inspect sederhana:latest -f "{{.Config.Labels}}"`

```
belajar@sabarmenanti:~/images$ docker image inspect sederhana:latest -f "{{.Config.Labels}}"
map[description:Belajar membuat Docker Image email:putu.hariyadi@universitasbumigora.ac.id]
```

   Terlihat *output* metadata berupa **description** dan **email** pada *container image*.

D. Instruksi COPY pada Dockerfile

Menindaklanjuti hasil penyelesaian *container image* di bagian C maka pada bagian ini akan dilakukan penyesuaian pada *file* **Dockerfile** untuk menyalinkan *file* atau direktori dari *build context* ke dalam *image* yang dibuat dengan menggunakan instruksi `COPY`. Instruksi `COPY` memiliki format penulisan:

```
COPY [OPTIONS] <src> ... <dest>

COPY [OPTIONS] ["<src>", ... "<dest>"]
```

Perintah ini memiliki beberapa *options* yaitu:

- `--from` digunakan untuk menyalin *file* dari sebuah *image*, *build stage* atau *named context*.
- `--chown` digunakan untuk mengubah kepemilikan dari *file* yang disalin baik *username* atau *groupname*.
- `--chmod` digunakan untuk mengubah ijin akses (*permission*) dari *file* yang disalin menggunakan notasi oktal.
- `--exclude` digunakan untuk mengatur *path* atau lokasi ekspresi dari *file* yang tidak disalin.

Argumen `<src>` merupakan lokasi *file* atau direktori sumber yang akan disalin. Sedangkan argumen `<dest>` merupakan lokasi tujuan penyalinan di dalam *file system container image*.

Sebagai contoh pada *host machine* **Ubuntu Desktop** terdapat direktori "**images**" di *home* direktori dari *user* yang digunakan *login* yaitu *user* "**belajar**" sehingga memiliki *path* "**/home/belajar/images**". Pada direktori tersebut akan dibuat direktori baru bernama "**dokumen**" dan didalamnya akan memuat 3 (tiga) *file* yaitu:

- **pertama.txt** dengan konten "**Data pada file pertama**".
- **kedua.txt** dengan konten "**Data pada file kedua**".
- **ketiga.txt** dengan konten "**Data pada file ketiga**".

Ketiga *file* dengan ekstensi ***.txt** tersebut akan disalinkan ke direktori "**data**" di dalam *image* menggunakan instruksi `COPY`. Selain itu ketika *container* dijalankan akan mengeksekusi perintah "`ls data`" untuk menampilkan isi dari direktori data di dalam *container* tersebut menggunakan instruksi `CMD`.

Adapun langkah-langkah pembuatan *container image* untuk mengakomodir ketentuan tersebut dan mengujicoba hasil pembuatannya adalah sebagai berikut:

1. Membuat direktori dokumen.
   ```
   $ mkdir dokumen
   ```
2. Membuat tiga *file* yaitu "**pertama.txt**", "**kedua.txt**", dan "**ketiga.txt**" dengan konten yang telah ditentukan sebelumnya.
   ```
   $ echo Data pada file pertama > dokumen/pertama.txt
   $ echo Data pada file kedua > dokumen/kedua.txt
   $ echo Data pada file ketiga > dokumen/ketiga.txt
   ```
3. Memverifikasi hasil pembuatan tiga *file* di dalam direktori **dokumen**.
   ```
   $ ls dokumen
   ```
   ```
   belajar@sabarmenanti:~/images$ ls dokumen
   kedua.txt  ketiga.txt  pertama.txt
   ```
4. Memverifikasi konten dari tiga *file* di dalam direktori **dokumen**.
   ```
   $ cat dokumen/pertama.txt
   $ cat dokumen/kedua.txt
   $ cat dokumen/ketiga.txt
   ```

```
belajar@sabarmenanti:~/images$ cat dokumen/pertama.txt
Data pada file pertama
belajar@sabarmenanti:~/images$ cat dokumen/kedua.txt
Data pada file kedua
belajar@sabarmenanti:~/images$ cat dokumen/ketiga.txt
Data pada file ketiga
```

5. Mengubah konten *file* **Dockerfile** menggunakan *editor* **nano**.

   ```
   $ nano Dockerfile
   ```

   Sebagai contoh instruksi `COPY` untuk menyalin *file* ke dalam *image* guna memenuhi ketentuan ditambahkan pada baris sebelum instruksi `CMD`. Selain itu parameter dari instruksi `CMD` diubah agar menampilkan isi dari direktori **data** ketika *container* tersebut dijalankan. Hasil akhir penyesuaian pada *file* **Dockerfile** akan terlihat seperti berikut:

   ```
   FROM alpine:latest
   LABEL description="Belajar membuat Docker Image"
   LABEL email="putu.hariyadi@universitasbumigora.ac.id"
   RUN mkdir data
   RUN echo "Pelatihan Cloud Native" > data/baca.txt
   COPY dokumen/*.txt data
   CMD ["ls", "data"]
   ```

   Simpan perubahan dengan menekan tombol **CTRL+O** dan tekan **Enter**.

   Keluar dari *editor nano* dengan menekan tombol **CTRL+X**.

6. Membuat *docker container images* dengan mengeksekusi perintah:

   ```
   $ docker image build -t sederhana:latest .
   ```

5. Memverifikasi hasil pembuatan *container images*.

```
belajar@sabarmenanti:~/images$ docker image ls
REPOSITORY    TAG       IMAGE ID       CREATED          SIZE
sederhana     latest    8f6e0c38113b   4 seconds ago    7.8MB
<none>        <none>    21e1eeb9e43e   15 hours ago     7.8MB
<none>        <none>    b997caf140ab   15 hours ago     7.8MB
<none>        <none>    4935ebc9280c   15 hours ago     7.8MB
httpd         latest    c0c20df5e7be   29 hours ago     148MB
nginx         latest    fffffc90d343   13 days ago      188MB
hello-world   latest    d2c94e258dcb   14 months ago    13.3kB
```

Terlihat *container image* tersebut telah berhasil dibuat.

6. Ujicoba membuat dan menjalankan *container* menggunakan *images* **sederhana:latest**. Selain itu apabila *container* dihentikan maka akan langsung menghapus *container* dengan memanfaatkan option `--rm` (*remove*) dari perintah `docker run`.

   ```
   $ docker run --rm sederhana:latest
   ```

```
belajar@sabarmenanti:~/images$ docker run --rm sederhana:latest
baca.txt
kedua.txt
ketiga.txt
pertama.txt
```

   Terlihat *output* isi dari direktori data yang memuat tiga file yaitu masing-masing dengan nama "**pertama.txt**", "**kedua.txt**", dan "**ketiga.txt**".

E. Instruksi ADD pada Dockerfile

Menindaklanjuti hasil penyelesaian *container image* di bagian D maka pada bagian ini akan dilakukan penyesuaian pada *file* **Dockerfile** untuk menyalinkan *file* atau direktori yang ada di lokal (*host machine*) atau *remote file Uniform Resource Locator (URL)* ke dalam *image* yang dibuat dengan menggunakan instruksi `ADD`. Instruksi ini mirip dengan instruksi `COPY` namun memiliki fungsionalitas tambahan seperti mengekstrak arsip berformat *tar* dan menyalinkan *file* yang bersumber dari URL tertentu. Instruksi `ADD` memiliki format penulisan:

```
ADD [OPTIONS] <src> ... <dest>

ADD [OPTIONS] ["<src>", ... "<dest>"]
```

Perintah ini memiliki beberapa *options* yaitu:

- `--chown` digunakan untuk mengubah kepemilikan dari *file* yang disalin baik *username* atau *groupname*.
- `--chmod` digunakan untuk mengubah ijin akses (*permission*) dari *file* yang disalin menggunakan notasi oktal.
- `--exclude` digunakan untuk mengatur *path* atau lokasi ekspresi dari *file* yang tidak disalin.

Argumen `<src>` merupakan lokasi *file* atau direktori sumber yang akan disalin. Sedangkan argumen `<dest>` merupakan lokasi tujuan penyalinan di dalam *file system container image*.

Sebagai contoh pada *host machine* **Ubuntu Desktop** telah berjalan *container* bernama **nginx** yang menggunakan *image* **nginx:latest** dan didalamnya memuat *file* "**unduh.txt**" yang dapat diunduh dari host machine dengan mengakses alamat **http://localhost/unduh.txt**. *Remote file* "**unduh.txt**" dengan URL tersebut akan disalinkan ke direktori "**data**" di dalam *image* menggunakan instruksi `ADD`. Selain itu ketika *container* dijalankan akan mengeksekusi perintah "`ls data`" untuk menampilkan isi dari direktori data di dalam *container* tersebut menggunakan instruksi `CMD`.

Adapun langkah-langkah pembuatan *container image* untuk mengakomodir ketentuan tersebut dan mengujicoba hasil pembuatannya adalah sebagai berikut:

1. Menampilkan informasi container yang sedang berjalan.

   ```
   $ docker ps
   ```

   ```
   belajar@sabarmenanti:~/images$ docker ps
   CONTAINER ID   IMAGE          COMMAND                  CREATED              STATUS              PORTS                                   NAMES
   89397ladl19d   nginx:latest   "/docker-entrypoint.…"   About a minute ago   Up About a minute   0.0.0.0:80->80/tcp, :::80->80/tcp   nginx
   ```

   Terlihat terdapat *container* bernama **nginx** yang sedang berjalan.

2. Membuat *file* bernama "**unduh.txt**" dengan konten "**Data file unduh**" di dalam *container* **nginx** yaitu dengan lokasi penyimpanan di direktori **/usr/share/nginx/html**.

   ```
   $ docker exec nginx bash -c "echo 'Data file unduh.txt' > /usr/share/nginx/html/unduh.txt"
   ```

3. Memverifikasi isi dari file "**unduh.txt**" yang terdapat di dalam *container* **nginx**.

   ```
   belajar@sabarmenanti:~/images$ docker exec nginx cat /usr/share/nginx/html/unduh.txt
   Data file unduh.txt
   ```

4. Mengubah konten *file* **Dockerfile** menggunakan *editor* **nano**.

   ```
   $ nano Dockerfile
   ```

   Sebagai contoh instruksi `ADD` untuk menyalin *file* ke dalam *image* guna memenuhi ketentuan ditambahkan pada baris sebelum instruksi `CMD`. Hasil akhir penyesuaian pada *file* **Dockerfile** akan terlihat seperti berikut:

   ```
   FROM alpine:latest
   LABEL description="Belajar membuat Docker Image"
   LABEL email="putu.hariyadi@universitasbumigora.ac.id"
   RUN mkdir data
   RUN echo "Pelatihan Cloud Native" > data/baca.txt
   COPY dokumen/*.txt data
   ```

```
ADD http://localhost/unduh.txt data/
CMD ["ls", "data"]
```

Simpan perubahan dengan menekan tombol **CTRL+O** dan tekan **Enter**.

Keluar dari *editor nano* dengan menekan tombol **CTRL+X**.

5. Membuat *docker container images* dengan mengeksekusi perintah:

   ```
   $ docker image build -t sederhana:latest .
   ```

6. Memverifikasi hasil pembuatan *container images*.

```
belajar@sabarmenanti:~/images$ docker image ls
REPOSITORY    TAG       IMAGE ID       CREATED              SIZE
sederhana     latest    c7e1d45cfd57   6 seconds ago        7.8MB
<none>        <none>    8f6e0c38113b   49 minutes ago       7.8MB
<none>        <none>    b997caf140ab   16 hours ago         7.8MB
<none>        <none>    4935ebc9280c   16 hours ago         7.8MB
<none>        <none>    21e1eeb9e43e   16 hours ago         7.8MB
httpd         latest    c0c20df5e7be   29 hours ago         148MB
nginx         latest    fffffc90d343   13 days ago          188MB
hello-world   latest    d2c94e258dcb   14 months ago        13.3kB
```

Terlihat *container image* tersebut telah berhasil dibuat.

7. Ujicoba membuat dan menjalankan *container* menggunakan *images* **sederhana:latest**. Selain itu apabila *container* dihentikan maka akan langsung menghapus *container* dengan memanfaatkan option `--rm` (*remove*) dari perintah `docker run`.

   ```
   $ docker run --rm sederhana:latest
   ```

```
belajar@sabarmenanti:~/images$ docker run --rm sederhana:latest
baca.txt
kedua.txt
ketiga.txt
pertama.txt
unduh.txt
```

Terlihat *output* isi dari direktori **data** yang telah memuat *file* "**unduh.txt**" yang diunduh dari *container* **nginx** ketika pembuatan *image container* **sederhana:latest**.

F. Instruksi ARG pada Dockerfile

Menindaklanjuti hasil penyelesaian *container image* di bagian E maka pada bagian ini akan dilakukan penyesuaian pada *file* **Dockerfile** untuk mengatur *variable* saat *build*. Sebagai contoh konten dari *file* "**baca.txt**" yang terdapat di direktori "**data**" diambil dari nilai *variable*

bernama "**konten**" yang dideklarasikan menggunakan instruksi `ARG`. *Variable* konten tersebut diatur agar memiliki nilai *default* "**Pelatihan Cloud Native**".

Instruksi `ARG` memiliki format penulisan:

```
ARG <name>[=<default value>]
```

Argumen `<name>` merupakan nama *variable*. *Variable* dapat dideklarasikan tanpa memiliki nilai *default*. Sebagai contoh *variable* bernama **konten** dideklarasikan tanpa nilai *default* maka penulisannya akan terlihat seperti berikut:

`ARG` **konten**

Sebaliknya jika *variable* tersebut diatur agar memiliki nilai *default* "**Pelatihan Cloud Native**" maka deklarasinya akan terlihat seperti berikut:

`ARG` **konten="Pelatihan Cloud Native"**

Apabila pengguna ingin mengubah nilai dari *variable* "**konten**" tersebut maka dapat menambahkan *option* `--build-arg <varname>=<value>` sewaktu mengeksekusi perintah `docker build`. Sehingga sintak penulisan perintahnya akan terlihat seperti berikut:

```
docker image build --build-arg <varname>=<value> PATH
```

Argumen `<varname>` merupakan nama *variable*. Sedangkan `<value>` adalah nilai dari *variable* yang dideklarasikan. Apabila pengguna tidak memasukkan nilai untuk *variable* "**konten**" sewaktu *build* maka nilai *default* tersebut yang akan digunakan.

Adapun langkah-langkah pembuatan *container image* untuk mengakomodir ketentuan tersebut dan mengujicoba hasil pembuatannya adalah sebagai berikut:

1. Mengubah konten *file* **Dockerfile** menggunakan *editor* **nano**.

   ```
   $ nano Dockerfile
   ```

   Sebagai contoh instruksi `ARG` dengan nama *variable* **konten** bernilai "**Pelatihan Cloud Native**" ditambahkan pada baris setelah instruksi `LABEL`. Hasil akhir penyesuaian pada *file* **Dockerfile** akan terlihat seperti berikut:

   ```
   FROM alpine:latest
   LABEL description="Belajar membuat Docker Image"
   ```

```
LABEL email="putu.hariyadi@universitasbumigora.ac.id"
ARG konten="Pelatihan Cloud Native"
RUN mkdir data
RUN echo "$konten" > data/baca.txt
CMD ["cat", "data/baca.txt"]
```

Terlihat penyesuaian dilakukan pada instruksi `RUN` yang terdapat di baris kedua dari baris terakhir konten *file* **Dockerfile**. Argumen dari perintah `echo` diambil dari nilai *variable* **konten** dengan sintak penulisan **`$konten`**. Selain itu instruksi **CMD** di baris terakhir juga dilakukan penyesuaian agar konten dari *file* "**baca.txt"** di dalam direktori "**data**" dapat ditampilkan ke layar saat *container* dijalankan.

Simpan perubahan dengan menekan tombol **CTRL+O** dan tekan **Enter**.
Keluar dari *editor nano* dengan menekan tombol **CTRL+X**.

2. Membuat *docker container images* tanpa mengubah nilai dari *variable* **konten** yang terdapat pada instruksi `ARG` dengan mengeksekusi perintah:

```
$ docker image build -t sederhana:latest .
```

```
belajar@sabarmenanti:~/images$ docker build -t sederhana:latest .
[+] Building 14.8s (8/8) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 276B
 => [internal] load metadata for docker.io/library/alpine:latest
 => [auth] library/alpine:pull token for registry-1.docker.io
 => [internal] load .dockerignore
 => => transferring context: 2B
 => CACHED [1/3] FROM docker.io/library/alpine:latest@sha256:b89d9c93e9ec
 => [2/3] RUN mkdir data
 => [3/3] RUN echo "Pelatihan Cloud Native" > data/baca.txt
 => exporting to image
 => => exporting layers
 => => writing image sha256:e565fa45b2a3f9911ba41a427d607a082620af51e2601
 => => naming to docker.io/library/sederhana:latest
```

Terlihat nilai dari argumen perintah `echo` pada instruksi `RUN` menggunakan nilai dari *variable* **konten** pada instruksi `ARG`.

3. Memverifikasi hasil pembuatan *container images*.

```
belajar@sabarmenanti:~/images$ docker image ls
REPOSITORY    TAG       IMAGE ID       CREATED          SIZE
sederhana     latest    e565fa45b2a3   2 hours ago      7.8MB
<none>        <none>    38d99562a9c9   4 hours ago      7.8MB
<none>        <none>    45cc751d62ea   4 hours ago      7.8MB
<none>        <none>    ec85034aa87a   4 hours ago      7.8MB
<none>        <none>    4b6219f77335   5 hours ago      188MB
<none>        <none>    f366735f05d6   10 hours ago     7.8MB
nginx         latest    fffffc90d343   2 weeks ago      188MB
```

Terlihat *container image* tersebut telah berhasil dibuat.

4. Ujicoba membuat dan menjalankan *container* menggunakan *images* **sederhana:latest**. Selain itu apabila *container* dihentikan maka akan langsung menghapus *container* dengan memanfaatkan option `--rm` (*remove*) dari perintah `docker run`.

   ```
   $ docker run --rm sederhana:latest
   ```

```
belajar@sabarmenanti:~/images$ docker run --rm sederhana:latest
Pelatihan Cloud Native
```

   Terlihat *output* "**Pelatihan Cloud Native**" yang merupakan nilai *default variable* **konten** pada instruksi **ARG** sewaktu pembuatan *image container* **sederhana:latest**.

5. Membuat *docker container images* dengan mengubah nilai dari *variable* **konten** yang terdapat pada instruksi `ARG` menggunakan "**Selamat Belajar Cloud Native**" dengan mengeksekusi perintah:

   ```
   $ docker build --build-arg \
   konten="Selamat Belajar Cloud Native" \
   -t sederhana:latest .
   ```

```
belajar@sabarmenanti:~/images$ docker build --build-arg \
> konten="Selamat Belajar Cloud Native" \
> -t sederhana:latest .
[+] Building 7.8s (8/8) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 276B
 => [internal] load metadata for docker.io/library/alpine:latest
 => [auth] library/alpine:pull token for registry-1.docker.io
 => [internal] load .dockerignore
 => => transferring context: 2B
 => CACHED [1/3] FROM docker.io/library/alpine:latest@sha256:b89d9c93e9ed3597
 => [2/3] RUN mkdir data
 => [3/3] RUN echo "Selamat Belajar Cloud Native" > data/baca.txt
 => exporting to image
 => => exporting layers
 => => writing image sha256:62f35d0abb550dc9e2539418e388a4565a271bed861d34ade
 => => naming to docker.io/library/sederhana:latest
```

   Terlihat nilai dari argumen perintah `echo` pada instruksi `RUN` menggunakan nilai dari *variable* **konten** pada instruksi `ARG` yang dimasukkan melalui pengaturan *option* `--build-arg` sewaktu mengeksekusi perintah `docker build`.

6. Memverifikasi hasil pembuatan *container images*.

```
belajar@sabarmenanti:~/images$ docker image ls
REPOSITORY    TAG       IMAGE ID       CREATED          SIZE
sederhana     latest    62f35d0abb55   10 minutes ago   7.8MB
<none>        <none>    e565fa45b2a3   2 hours ago      7.8MB
<none>        <none>    38d99562a9c9   4 hours ago      7.8MB
<none>        <none>    45cc751d62ea   4 hours ago      7.8MB
<none>        <none>    ec85034aa87a   4 hours ago      7.8MB
<none>        <none>    4b6219f77335   6 hours ago      188MB
<none>        <none>    f366735f05d6   10 hours ago     7.8MB
nginx         latest    fffffc90d343   2 weeks ago      188MB
```

Terlihat *container image* tersebut telah berhasil dibuat.

7. Ujicoba membuat dan menjalankan *container* menggunakan *images* **sederhana:latest**. Selain itu apabila *container* dihentikan maka akan langsung menghapus *container* dengan memanfaatkan option `--rm` (*remove*) dari perintah `docker run`.

```
$ docker run --rm sederhana:latest
```

```
belajar@sabarmenanti:~/images$ docker run --rm sederhana:latest
Selamat Belajar Cloud Native
```

Terlihat *output* "**Selamat Belajar Cloud Native**" yang merupakan nilai *variable* **konten** pada instruksi **ARG** yang dimasukkan melalui pengaturan *option* `--build-arg` dari perintah `docker build` sewaktu pembuatan *image container* **sederhana:latest**.

G. Instruksi WORKDIR pada Dockerfile

Menindaklanjuti hasil penyelesaian *container image* di bagian F maka pada bagian ini akan dilakukan penyesuaian pada *file* **Dockerfile** untuk mengatur *current working directory* di dalam *container* sebagai lokasi perintah-perintah dieksekusi. Konten *file Dockerfile* saat ini terlihat seperti berikut:

```
FROM alpine:latest
LABEL description="Belajar membuat Docker Image"
LABEL email="putu.hariyadi@universitasbumigora.ac.id"
ARG konten="Pelatihan Cloud Native"
RUN mkdir data
RUN echo "$konten" > data/baca.txt
CMD ["cat", "data/baca.txt"]
```

Instruksi `RUN` dan `CMD` pada dua baris terakhir di *file* tersebut memiliki perintah dengan argumen yang merujuk ke lokasi direktori yang sama yaitu "**data**". Hal ini dapat disederhanakan dengan memanfaatkan instruksi `WORKDIR`. Instruksi `WORKDIR` dengan memiliki format penulisan:

```
WORKDIR /path/to/workdir
```

Argumen `/path/to/workdir` dari instruksi tersebut merupakan lokasi direktori yang dapat ditulis secara absolut atau *relative*. Untuk merujuk ke *file* "**baca.txt**" dengan lokasi di direktori "**data**" maka instruksi berikut jika ditulis secara *relative*:

```
WORKDIR data
```

Sebaliknya jika ditulis secara absolut maka dapat ditulis sebagai berikut:

```
WORKDIR /data
```

Adapun langkah-langkah pembuatan *container image* untuk mengakomodir ketentuan tersebut dan mengujicoba hasil pembuatannya adalah sebagai berikut:

1. Mengubah konten *file* **Dockerfile** menggunakan *editor* **nano**.

```
$ nano Dockerfile
```

```
FROM alpine:latest
LABEL description="Belajar membuat Docker Image"
LABEL email="putu.hariyadi@universitasbumigora.ac.id"
ARG konten="Pelatihan Cloud Native"
RUN mkdir data
WORKDIR /data
RUN echo "$konten" > baca.txt
CMD ["cat", "baca.txt"]
```

Terlihat dilakukan penambahan instruksi `WORKDIR` di baris ketiga dari baris terakhir konten *file* **Dockerfile** dengan argumen berupa lokasi atau *path* direktori yang ditulis secara absolut yaitu **/data**. Penyesuaian juga dilakukan pada instruksi `RUN` yang terdapat di baris kedua dari baris terakhir. Cukup dituliskan nama *file* saja setelah simbol redirection yaitu "**baca.txt**" karena *current working directory* saat ini sudah berada di direktori **/data**.

Demikian pula untuk instruksi **CMD** di baris terakhir juga dilakukan penyesuaian nilai pada parameter kedua dari instruksi `CMD` yaitu menjadi "**baca.txt".**

Simpan perubahan dengan menekan tombol **CTRL+O** dan tekan **Enter**.

Keluar dari *editor nano* dengan menekan tombol **CTRL+X**.

2. Membuat *docker container images* dengan mengeksekusi perintah:

```
$ docker image build -t sederhana:latest .
```

```
belajar@sabarmenanti:~/images$ docker build -t sederhana:latest .
[+] Building 1.2s (8/8) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 280B
 => [internal] load metadata for docker.io/library/alpine:latest
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [1/4] FROM docker.io/library/alpine:latest@sha256:b89d9c93e9ed359
 => CACHED [2/4] RUN mkdir data
 => CACHED [3/4] WORKDIR /data
 => CACHED [4/4] RUN echo "Pelatihan Cloud Native" > baca.txt
 => exporting to image
 => => exporting layers
 => => writing image sha256:167cda67099367301b03af6d8da149e1391f3ad31
 => => naming to docker.io/library/sederhana:latest
```

Terlihat nilai argumen dari instruksi `WORKDIR` telah menggunakan **/data**.

3. Memverifikasi hasil pembuatan *container images*.

```
belajar@sabarmenanti:~/images$ docker image ls
REPOSITORY   TAG       IMAGE ID       CREATED          SIZE
sederhana    latest    167cda670993   50 seconds ago   7.8MB
<none>       <none>    62f35d0abb55   57 minutes ago   7.8MB
<none>       <none>    e565fa45b2a3   3 hours ago      7.8MB
<none>       <none>    38d99562a9c9   5 hours ago      7.8MB
<none>       <none>    45cc751d62ea   5 hours ago      7.8MB
<none>       <none>    ec85034aa87a   5 hours ago      7.8MB
<none>       <none>    4b6219f77335   6 hours ago      188MB
<none>       <none>    f366735f05d6   11 hours ago     7.8MB
nginx        latest    fffffc90d343   2 weeks ago      188MB
```

Terlihat *container image* tersebut telah berhasil dibuat.

4. Ujicoba membuat dan menjalankan *container* menggunakan *images* **sederhana:latest**. Selain itu apabila *container* dihentikan maka akan langsung menghapus *container* dengan memanfaatkan option `--rm` (*remove*) dari perintah `docker run`.

```
$ docker run --rm sederhana:latest
```

```
belajar@sabarmenanti:~/images$ docker run --rm sederhana:latest
Pelatihan Cloud Native
```

Terlihat *output* "**Pelatihan Cloud Native**" yang merupakan konten dari *file* "**baca.txt**" di dalam direktori **/data** tetap dapat ditampilkan.

H. Instruksi ENV pada Dockerfile

Sebagai contoh akan dibuat *container image* menggunakan *base image* "**alpine**" dengan versi terkini atau tag "**latest**". Nama dari *container image* yang dibuat adalah **salam:latest**. Saat *build stage* dari *image* tersebut akan menyalinkan *shell script* bernama **salam.sh**. *Shell script* tersebut ketika dipicu akan menampilkan pesan "`Selamat belajar Cloud Native, rekan` **`$NAMA`**`. Semangat!`". **`$NAMA`** digunakan untuk mengambil nilai **environment variable** yang dimasukkan pengguna ketika membuat dan menjalankan *container image* **salam:latest**.

Environment variable dideklarasikan menggunakan instruksi `ENV`. Instruksi `ENV` memiliki format penulisan:

```
ENV <key>=<value> ...
```

Argumen `<key>` merupakan nama *variable*. *Variable* dapat dideklarasikan tanpa memiliki nilai *default*. Sebagai contoh *variable* bernama **NAMA** dideklarasikan tanpa nilai *default* maka penulisannya akan terlihat seperti berikut:

`ENV` **NAMA**

Sebaliknya jika *variable* tersebut diatur agar memiliki nilai *default* "**I Putu Hariyadi**" maka deklarasinya akan terlihat seperti berikut:

`ENV` **NAMA="I Putu Hariyadi"**

Apabila pengguna ingin mengubah nilai dari *variable* "**NAMA**" tersebut maka dapat menambahkan *option* `--env <key>=<name>` sewaktu mengeksekusi perintah untuk membuat dan menjalankan *container*. Sebagai contoh jika menggunakan `docker run`. Maka sintak penulisan perintahnya akan terlihat seperti berikut:

```
docker run --env <key>=<name>
```

Argumen `<key>` merupakan nama *variable*. Sedangkan `<name>` adalah nilai dari *variable* yang dideklarasikan. Apabila pengguna tidak memasukkan nilai untuk *variable* "**NAME**" sewaktu membuat dan menjalankan *container* maka nilai *default* tersebut yang akan digunakan.

Adapun langkah-langkah pembuatan *container image* dengan ketentuan tersebut dan mengujicoba hasil pembuatannya adalah sebagai berikut:

1. Membuat *shell script* bernama **salam.sh**.

   ```
   $ nano salam.sh
   ```

   Konten dari *file shell script* akan terlihat seperti berikut:

   ```
   #!/bin/sh
   echo "Selamat belajar Cloud Native, rekan $NAMA. Semangat!"
   ```

   Terlihat pada baris pertama terdapat *shebang* **#!** untuk menginformasikan ke sistem terkait *interpreter* yang digunakan ketika mengeksekusi skrip yaitu **/bin/sh**. Sedangkan pada baris kedua terdapat perintah `echo` yang digunakan untuk menampilkan pesan ke layar dan memuat `$NAMA` untuk mengambil nilai dari *environment variable* yang dideklarasikan menggunakan instruksi `ENV`.

   Simpan perubahan dengan menekan tombol **CTRL+O** dan tekan **Enter**.
   Keluar dari *editor nano* dengan menekan tombol **CTRL+X**.

2. Membuat *file* bernama **Dockerfile** menggunakan *editor* **nano**.

   ```
   $ nano Dockerfile
   ```

   Konten dari *file* akan terlihat seperti berikut:

   ```
   FROM alpine:latest
   ENV NAMA="I Putu Hariyadi"
   COPY salam.sh .
   RUN chmod a+x /salam.sh
   CMD ["/salam.sh"]
   ```

   Terlihat pada baris pertama terdapat instruksi `FROM` untuk menggunakan *base image* **alpine:latest** ketika *build stage*. Pada baris kedua terdapat deklarasi *environment variable* "`NAMA`" dengan nilai *default* "**I Putu Hariyadi**". Instruksi `COPY` pada baris ketiga digunakan untuk menyalinkan *file shell script* **salam.sh** ke *container image*. Sedangkan pada baris ke empat terdapat instruksi `RUN` untuk mengubah *permission* pada *file shell*

script **salam.sh** menggunakan **chmod** agar seluruh pengguna memiliki hak **executable (x)**. Instruksi `CMD` pada baris terakhir digunakan untuk mengeksekusi *shell script* **/salam.sh** ketika *container* dijalankan.

Simpan perubahan dengan menekan tombol **CTRL+O** dan tekan **Enter**.

Keluar dari *editor nano* dengan menekan tombol **CTRL+X**.

3. Membuat *docker container images* dengan mengeksekusi perintah:

```
$ docker image build -t salam:latest .
```

```
belajar@sabarmenanti:~/images$ docker build -t salam:latest .
[+] Building 2.0s (8/8) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 141B
 => [internal] load metadata for docker.io/library/alpine:latest
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load build context
 => => transferring context: 105B
 => [1/3] FROM docker.io/library/alpine:latest@sha256:b89d9c93e9ed359745
 => CACHED [2/3] COPY salam.sh .
 => [3/3] RUN chmod a+x /salam.sh
 => exporting to image
 => => exporting layers
 => => writing image sha256:fcdb4bb5adae22a9bfeeeca9eada6c458b24c9ca7ec2
 => => naming to docker.io/library/salam:latest
```

4. Memverifikasi hasil pembuatan *container images*.

```
belajar@sabarmenanti:~/images$ docker image ls
REPOSITORY   TAG       IMAGE ID       CREATED         SIZE
salam        latest    808267be1e0c   7 hours ago     7.8MB
sederhana    latest    167cda670993   10 hours ago    7.8MB
<none>       <none>    62f35d0abb55   11 hours ago    7.8MB
<none>       <none>    e565fa45b2a3   12 hours ago    7.8MB
<none>       <none>    38d99562a9c9   14 hours ago    7.8MB
<none>       <none>    45cc751d62ea   14 hours ago    7.8MB
<none>       <none>    ec85034aa87a   14 hours ago    7.8MB
<none>       <none>    4b6219f77335   16 hours ago    188MB
<none>       <none>    f366735f05d6   20 hours ago    7.8MB
nginx        latest    fffffc90d343   2 weeks ago     188MB
```

Terlihat *container image* tersebut telah berhasil dibuat.

8. Ujicoba membuat dan menjalankan *container* menggunakan *images* **salam:latest** tanpa mengubah nilai dari *environment variable* **NAMA .** Selain itu apabila *container* dihentikan maka akan langsung menghapus *container* dengan memanfaatkan option `--rm` (*remove*) dari perintah `docker run`.

```
$ docker run --rm salam:latest
```

```
belajar@sabarmenanti:~/images$ docker run --rm salam:latest
Selamat belajar Cloud Native, rekan I Putu Hariyadi. Semangat!
```

Terlihat *output* "Selamat belajar Cloud Native, rekan **I Putu Hariyadi**. Semangat!". Nilai "**I Putu Hariyadi**" pada *output* tersebut merupakan nilai *default environment variable* **NAMA** dari instruksi **ENV** yang dideklarasikan sewaktu pembuatan *image container* **salam:latest**.

9. Memverifikasi hasil penambahan instruksi ENV pada *container image* tersebut dengan mengeksekusi perintah:

   $ `docker image inspect salam:latest`

   Cuplikan output dari hasil eksekusi perintah ini adalah sebagai berikut:

```
        "Config": {
            "Hostname": "",
            "Domainname": "",
            "User": "",
            "AttachStdin": false,
            "AttachStdout": false,
            "AttachStderr": false,
            "Tty": false,
            "OpenStdin": false,
            "StdinOnce": false,
            "Env": [
                "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
                "NAMA=I Putu Hariyadi"
            ],
```

Atau dengan menggunakan pemfilteran sehingga hanya menampilkan *environment variable (Env)* maka dapat mengeksekusi perintah:

`$ docker image inspect salam:latest -f` "{{.Config.Env}}"

```
belajar@sabarmenanti:~/images$ docker image inspect salam:latest -f "{{.Config.Env}}"
[PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin NAMA=I Putu Hariyadi]
```

Terlihat *output* environment variable **NAMA** dengan nilai default "**I Putu Hariyadi**" pada *container image*.

10. Ujicoba membuat dan menjalankan *container* kembali menggunakan *images* **salam:latest** dengan mengubah nilai dari *environment variable* **NAMA** menjadi "**Linus Torvalds**". Selain itu apabila *container* dihentikan maka akan langsung menghapus *container* dengan memanfaatkan option `--rm` (*remove*) dari perintah `docker run`.

    `$ docker run --rm --env` NAMA="Linux Torvalds" `salam:latest`

```
belajar@sabarmenanti:~/images$ docker run --rm --env NAMA="Linus Torvalds" salam:latest
Selamat belajar Cloud Native, rekan Linus Torvalds. Semangat!
```

Terlihat *output* "Selamat belajar Cloud Native, rekan **Linus Torvalds**. Semangat!". Nilai "**Linus Torvalds**" pada *output* tersebut merupakan nilai *environment variable* yang dimasukkan melalui pengaturan *option* `--env` dari perintah `docker run` sewaktu membuat dan menjalankan *container* **sederhana:latest**

I. Instruksi ENTRYPOINT pada Dockerfile

Sebagai contoh akan dibuat *container image* menggunakan *base image* "**ubuntu**" dengan versi terkini atau tag "**latest**". Saat *build stage* dari *image* tersebut akan melakukan pembaharuan *package index* dari *Ubuntu* dan menginstalasi paket **iputils-ping** agar pada *container* tersedia utilitas **ping** yang digunakan untuk menguji keterjangkauan *host* di jaringan. Selain itu ketika *container* tersebut dijalankan akan mengeksekusi perintah **ping** menggunakan instruksi `ENTRYPOINT`. Argumen *default* dari perintah **ping** yaitu menguji keterjangkauan ke *host* **google.com** dengan mengirimkan 4 (empat) **packet Internet Control Message Protocol (ICMP) echo request** menggunakan *option* **-c 4 google.com** yang disimpan sebagai parameter pada instruksi `CMD`. Nama dari *container image* yang dibuat adalah **ubuntu-with-ping:latest**.

Instruksi `ENTRYPOINT` memiliki 2 (dua) format penulisan yaitu bentuk **exec** dengan sintak:

```
ENTRYPOINT ["executable", "param1", "param2"]
```

Dan bentuk **shell** dengan sintak:

```
ENTRYPOINT command param1 param2
```

Argumen `executable` merupakan perintah yang akan dieksekusi saat *container* dijalankan. Sedangkan `param1` dan `param2` adalah parameter dari perintah yang dieksekusi saat *container* dijalankan.

Adapun langkah-langkah pembuatan *container image* dengan ketentuan tersebut dan mengujicoba hasil pembuatannya adalah sebagai berikut:

1. Membuat *file* bernama **Dockerfile** menggunakan *editor* **nano**.

   ```
   $ nano Dockerfile
   ```

   Konten dari *file* akan terlihat seperti berikut:

   ```
   FROM ubuntu:latest
   RUN apt update
   ```

```
RUN apt -y install iputils-ping
ENTRYPOINT ["ping"]
CMD ["-c","4","google.com"]
```

Terlihat pada baris pertama terdapat instruksi `FROM` untuk menggunakan *base image* **ubuntu:latest** ketika *build stage*. Instruksi `RUN` pada baris kedua akan mengeksekusi perintah **apt update** untuk melakukan pembaharuan *package index* dari *Ubuntu*. Sedangkan pada baris ketiga juga terdapat instruksi `RUN` yang akan mengeksekusi perintah **apt install -y iputils-ping** untuk menginstalasi paket **iputils-ping** agar pada *container* tersedia utilitas **ping** yang digunakan untuk menguji keterjangkauan *host* di jaringan. Baris ke empat terdapat instruksi `ENTRYPOINT` yang ditulis dalam bentuk **exec** yang akan mengeksekusi perintah **ping** ketika *container* dijalankan. Instruksi `CMD` pada baris terakhir memuat 3 (tiga) parameter dari perintah **ping** yang ditulis dalam bentuk **exec** yaitu **-c**, **4** dan **google.com**.

Simpan perubahan dengan menekan tombol **CTRL+O** dan tekan **Enter**.

Keluar dari *editor nano* dengan menekan tombol **CTRL+X**.

2. Membuat *docker container images* dengan mengeksekusi perintah:

```
$ docker image build -t ubuntu-with-ping:latest .
```

```
belajar@sabarmenanti:~/images$ docker build -t ubuntu-with-ping:latest .
[+] Building 252.2s (8/8) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 152B
 => [internal] load metadata for docker.io/library/ubuntu:latest
 => [auth] library/ubuntu:pull token for registry-1.docker.io
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [1/3] FROM docker.io/library/ubuntu:latest@sha256:2e863c44b718727c860746568e1d54afd13b2fa71b160f5c
 => => resolve docker.io/library/ubuntu:latest@sha256:2e863c44b718727c860746568e1d54afd13b2fa71b160f5c
 => => sha256:2e863c44b718727c860746568e1d54afd13b2fa71b160f5cd9058fc436217b30 1.13kB / 1.13kB
 => => sha256:c920ba4cfca05503764b785c16b76d43c83a6df8d1ab107e7e6610000d94315c 424B / 424B
 => => sha256:35a88802559dd2077e584394471ddaa1a2c5bfd16893b829ea57619301eb3908 2.30kB / 2.30kB
 => => sha256:9c704ecd0c694c4cbdd85e589ac8d1fc3fd8f890b7f3731769a5b169eb495809 29.71MB / 29.71MB
 => => extracting sha256:9c704ecd0c694c4cbdd85e589ac8d1fc3fd8f890b7f3731769a5b169eb495809
 => [2/3] RUN apt update
 => [3/3] RUN apt -y install iputils-ping
 => exporting to image
 => => exporting layers
 => => writing image sha256:31dee90bda7d6391603b02186b86601119c68f0a412d0c6a1fe01038885fc79c
 => => naming to docker.io/library/ubuntu-with-ping:latest
```

3. Memverifikasi hasil pembuatan *container images*.

```
belajar@sabarmenanti:~/images$ docker image ls
REPOSITORY          TAG       IMAGE ID       CREATED              SIZE
ubuntu-with-ping    latest    31dee90bda7d   About a minute ago   117MB
salam               latest    fcdb4bb5adae   About an hour ago    7.8MB
sederhana           latest    167cda670993   11 hours ago         7.8MB
<none>              <none>    62f35d0abb55   12 hours ago         7.8MB
<none>              <none>    e565fa45b2a3   14 hours ago         7.8MB
<none>              <none>    38d99562a9c9   16 hours ago         7.8MB
<none>              <none>    45cc751d62ea   16 hours ago         7.8MB
<none>              <none>    ec85034aa87a   16 hours ago         7.8MB
<none>              <none>    4b6219f77335   17 hours ago         188MB
<none>              <none>    f366735f05d6   22 hours ago         7.8MB
nginx               latest    fffffc90d343   2 weeks ago          188MB
```

Terlihat *container image* tersebut telah berhasil dibuat.

4. Ujicoba membuat dan menjalankan *container* menggunakan *images* **ubuntu-with-ping:latest** tanpa mengubah parameter dari instruksi `CMD` yang secara *default* mengirim 4 (empat) paket **ICMP echo request** ke **google.com**. Selain itu apabila *container* dihentikan maka akan langsung menghapus *container* dengan memanfaatkan option `--rm` (*remove*) dari perintah `docker run`.

   ```
   $ docker run --rm ubuntu-with-ping:latest
   ```

```
belajar@sabarmenanti:~/images$ docker run --rm ubuntu-with-ping:latest
PING forcesafesearch.google.com (216.239.38.120) 56(84) bytes of data.
64 bytes from any-in-2678.1e100.net (216.239.38.120): icmp_seq=1 ttl=53 time=42.0 ms
64 bytes from any-in-2678.1e100.net (216.239.38.120): icmp_seq=2 ttl=53 time=45.1 ms
64 bytes from any-in-2678.1e100.net (216.239.38.120): icmp_seq=3 ttl=53 time=42.0 ms
64 bytes from any-in-2678.1e100.net (216.239.38.120): icmp_seq=4 ttl=53 time=41.3 ms

--- forcesafesearch.google.com ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3007ms
rtt min/avg/max/mdev = 41.253/42.582/45.092/1.479 ms
```

Terlihat verifikasi koneksi ke **google.com** berhasil dilakukan.

5. Ujicoba membuat dan menjalankan *container* kembali menggunakan *images* **ubuntu-with-ping:latest** dengan mengubah parameter dari instruksi `CMD` yaitu mengirim 2 (dua) paket **ICMP echo request** ke **detik.com**. Selain itu apabila *container* dihentikan maka akan langsung menghapus *container* dengan memanfaatkan option `--rm` (*remove*) dari perintah `docker run`.

   ```
   $ docker run --rm ubuntu-with-ping:latest -c 2 detik.com
   ```

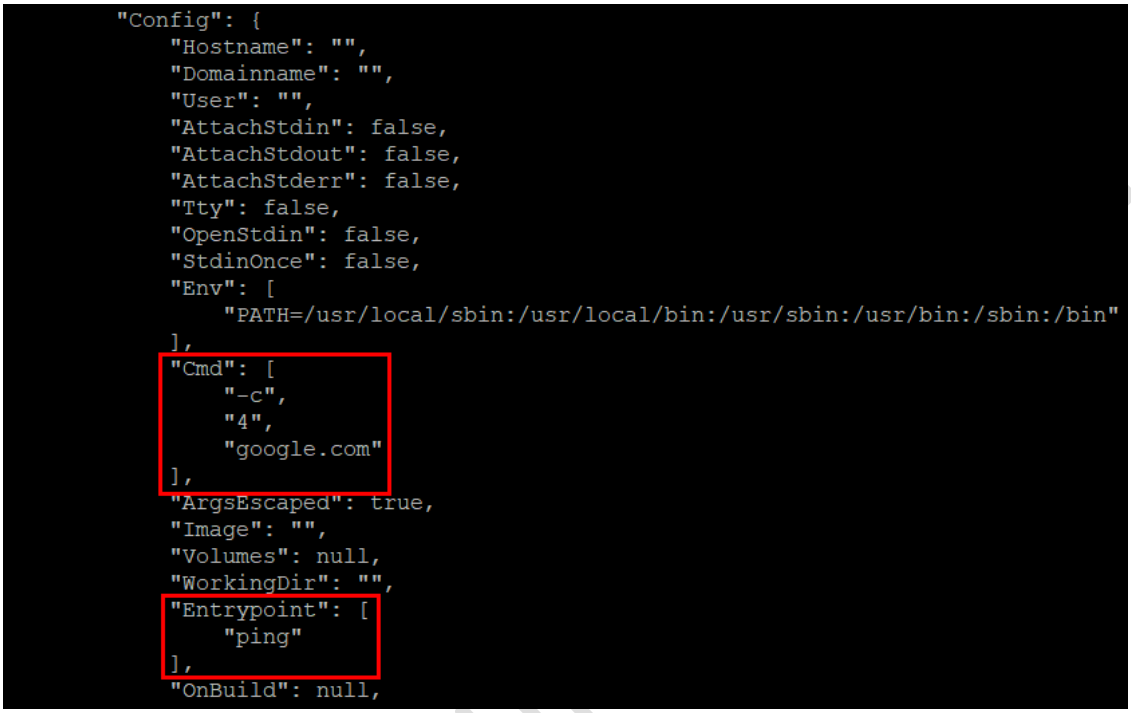
```
belajar@sabarmenanti:~/images$ docker run --rm ubuntu-with-ping:latest -c 2 detik.com
PING detik.com (203.190.242.211) 56(84) bytes of data.
64 bytes from s2-211-242.190.203.detik.com (203.190.242.211): icmp_seq=1 ttl=54 time=30.3 ms
64 bytes from s2-211-242.190.203.detik.com (203.190.242.211): icmp_seq=2 ttl=54 time=31.1 ms

--- detik.com ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms
rtt min/avg/max/mdev = 30.256/30.678/31.101/0.422 ms
```

Terlihat verifikasi koneksi ke **detik.com** berhasil dilakukan.

6. Memverifikasi hasil pengaturan instruksi `ENTRYPOINT` dan `CMD` pada *container image* tersebut dengan mengeksekusi perintah:

   $ `docker image inspect ubuntu-with-ping:latest`

   Cuplikan output dari hasil eksekusi perintah ini adalah sebagai berikut:



   Atau dengan menggunakan pemfilteran sehingga hanya menampilkan nilai dari hasil instruksi `ENTRYPOINT` dan `CMD` maka dapat mengeksekusi perintah:

```
$ docker inspect ubuntu-with-ping:latest \
-f "{{.Config.Entrypoint}} {{.Config.Cmd}}"
```

```
belajar@sabarmenanti:~/images$ docker inspect ubuntu-with-ping:latest \
> -f "{{.Config.Entrypoint}} {{.Config.Cmd}}"
[ping] [-c 4 google.com]
```

   Terlihat *output* nilai dari **Entrypoint** berupa **ping** dan nilai *default* dari **Cmd** yaitu `-c 4 google.com` pada *container image*.

J. Instruksi EXPOSE pada Dockerfile

Sebagai contoh akan dibuat *container image* menggunakan *base image* "**nginx**" dengan versi terkini atau tag "**latest**". Saat *build stage* dari *image* tersebut akan menyalinkan file **default.conf** yang telah disesuaikan nilai untuk *directive* `listen` agar layanan *nginx* berjalan

pada **port 8080**. Selain itu juga dilakukan penambahan instruksi `EXPOSE` dengan argumen **8080/tcp** untuk menjelaskan bahwa *container image* ketika dijalankan akan melakukan *listen* pada nomor *port* dan *protocol* tersebut. Instruksi `EXPOSE` hanya digunakan sebagai informasi atau dokumentasi. Nama dari *container image* yang dibuat adalah **nginx-custom:latest**.

Instruksi `EXPOSE` memiliki format penulisan:

```
EXPOSE <port> [<port>/<protocol>...]
```

Argumen `<port>` merupakan nomor *port* yang *listen* pada *container* ketika dijalankan. Sedangkan argument `<protocol>` digunakan untuk menentukan apakah *port* tersebut *listen* pada protocol *transport* **TCP** atau **UDP**. Secara default jika protokol tidak ditentukan maka akan digunakan **TCP**.

Adapun langkah-langkah pembuatan *container image* dengan ketentuan tersebut dan mengujicoba hasil pembuatannya adalah sebagai berikut:

1. Mengambil *file* **default.conf** dari dalam *container* **nginx** yang berjalan dan disimpan ke lokasi direktori saat ini berada.

   $ docker run --rm --entrypoint=cat nginx /etc/nginx/conf.d/default.conf > default.conf

```
belajar@sabarmenanti:~/images$ docker run --rm --entrypoint=cat nginx /etc/nginx/conf.d/default.conf > default.conf
```

2. Mengubah nilai *directive* **listen** pada *file* **default.conf** dari nomor *port* **80** menjadi **8080** menggunakan utilitas **sed**.

   ```
   $ sed -i 's/80/8080/g' default.conf
   ```

   ```
   belajar@sabarmenanti:~/images$ sed -i 's/80/8080/g' default.conf
   ```

3. Memverifikasi pengubahan *port* pada *directive* **listen** di *file* **default.conf**.

   ```
   $ grep -w listen default.conf
   ```

   ```
   belajar@sabarmenanti:~/images$ grep -w listen default.conf
       listen       8080;
   ```

   Terlihat *directive* **listen** telah bernilai **8080**.

4. Membuat *file* bernama **Dockerfile** menggunakan *editor* **nano**.

   ```
   $ nano Dockerfile
   ```

   Konten dari *file* akan terlihat seperti berikut:

   ```
   FROM nginx:latest
   WORKDIR /etc/nginx/conf.d
   ```

```
COPY default.conf .
EXPOSE 8080
```

Terlihat pada baris pertama terdapat instruksi `FROM` untuk menggunakan *base image* **nginx:latest** ketika *build stage*. Instruksi `WORKDIR` pada baris kedua digunakan untuk mengatur *current working directory* menjadi **/etc/nginx/conf.d**. Sedangkan pada baris ketiga terdapat instruksi `COPY` yang digunakan untuk menyalinkan *file* **default.conf** di *host machine* ke dalam *container image*. Instruksi `EXPOSE` pada baris terakhir memuat nomor *port* **8080** sebagai informasi atau dokumentasi ke pengguna bahwa layanan *nginx* akan berjalan pada *port* tersebut.

Simpan perubahan dengan menekan tombol **CTRL+O** dan tekan **Enter**.

Keluar dari *editor nano* dengan menekan tombol **CTRL+X**.

5. Membuat *docker container images* dengan mengeksekusi perintah:

```
$ docker image build -t nginx-custom:latest .
```

```
belajar@sabarmenanti:~/images$ docker build -t nginx-custom:latest .
[+] Building 0.1s (8/8) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 113B
 => [internal] load metadata for docker.io/library/nginx:latest
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [1/3] FROM docker.io/library/nginx:latest
 => [internal] load build context
 => => transferring context: 1.12kB
 => CACHED [2/3] WORKDIR /etc/nginx/conf.d
 => CACHED [3/3] COPY default.conf .
 => exporting to image
 => => exporting layers
 => => writing image sha256:60a34ac700be42ac65858850721b52a2cc9ab5147e08233b86b743a55bf72eed
 => => naming to docker.io/library/nginx-custom:latest
```

6. Memverifikasi hasil pembuatan *container images*.

```
belajar@sabarmenanti:~/images$ docker image ls
REPOSITORY          TAG       IMAGE ID       CREATED          SIZE
nginx-custom        latest    60a34ac700be   43 minutes ago   188MB
ubuntu-with-ping    latest    31dee90bda7d   3 hours ago      117MB
salam               latest    fcdb4bb5adae   5 hours ago      7.8MB
sederhana           latest    167cda670993   14 hours ago     7.8MB
<none>              <none>    62f35d0abb55   15 hours ago     7.8MB
<none>              <none>    e565fa45b2a3   17 hours ago     7.8MB
<none>              <none>    38d99562a9c9   19 hours ago     7.8MB
<none>              <none>    45cc751d62ea   19 hours ago     7.8MB
<none>              <none>    ec85034aa87a   19 hours ago     7.8MB
<none>              <none>    4b6219f77335   21 hours ago     188MB
<none>              <none>    f366735f05d6   25 hours ago     7.8MB
nginx               latest    fffffc90d343   2 weeks ago      188MB
```

Terlihat *container image* tersebut telah berhasil dibuat.

7. Memverifikasi hasil pengaturan instruksi `EXPOSE` pada *container image* tersebut dengan mengeksekusi perintah:

   $ `docker image inspect nginx-custom:latest`

   Cuplikan *output* dari hasil eksekusi perintah ini adalah sebagai berikut:

```
        "Config": {
            "Hostname": "",
            "Domainname": "",
            "User": "",
            "AttachStdin": false,
            "AttachStdout": false,
            "AttachStderr": false,
            "ExposedPorts": {
                "80/tcp": {},
                "8080/tcp": {}
            },
            "Tty": false,
```

   Atau dengan menggunakan pemfilteran sehingga hanya menampilkan nilai hasil dari instruksi `EXPOSE` maka dapat mengeksekusi perintah:

   `$ docker image inspect nginx-custom:latest \`

   `-f "{{.Config.ExposedPorts}}"`

```
belajar@sabarmenanti:~/images$ docker image inspect nginx-custom:latest \
> -f "{{.Config.ExposedPorts}}"
map[80/tcp:{} 8080/tcp:{}]
```

   Terlihat *output* nilai dari **ExposedPorts** berupa **8080/tcp** pada *container image*.

8. Ujicoba membuat dan menjalankan *container* menggunakan *images* **nginx-custom:latest**.

   `$ docker run --name nginx-custom -d -p 8080:8080 nginx-custom:latest`

```
belajar@sabarmenanti:~/images$ docker run -d --name nginx-custom -p 8080:8080 nginx-custom:latest
f98dbaad01af7f9336840946fbd296568661297a8e105d4707a0e859411ae1a4
```

9. Memverifikasi *docker container* bernama "**nginx-custom**" telah berjalan.

   `$ docker ps`

```
belajar@sabarmenanti:~/images$ docker ps
CONTAINER ID   IMAGE                 COMMAND                  CREATED         STATUS         PORTS                                                  NAMES
f98dbaad01af   nginx-custom:latest   "/docker-entrypoint.…"   5 seconds ago   Up 4 seconds   80/tcp, 0.0.0.0:8080->8080/tcp, :::8080->8080/tcp   nginx-custom
```

   Terlihat *container* tersebut telah berjalan berdasarkan informasi pada kolom **STATUS** bernilai **Up 4 seconds.**

10. Memverifikasi akses ke layanan *web* dari *container* **nginx-custom** pada **port 8080** menggunakan aplikasi **curl**.

```
$ curl localhost:8080
```

```
belajar@sabarmenanti:~/images$ curl localhost:8080
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

Terlihat konten *homepage* dari *container* **nginx-custom** berhasil diakses.

K. Instruksi VOLUME pada Dockerfile

Sebagai contoh akan dibuat *container image* menggunakan *base image* "**alpine**" dengan versi terkini atau tag "**latest**". Saat *build stage* dari *image* tersebut akan membuat direktori bernama "**data**" menggunakan instruksi `RUN`. Selain itu juga menyalinkan seluruh *file* yang terdapat pada direktori "**dokumen"** di *host machine* ke dalam direktori "**data**" pada *container image* menggunakan instruksi `COPY`. Terakhir membuat *volume mount* menggunakan instruksi `VOLUME` sebagai titik akses (*mount point*) bagi *volume* yang dipasang secara eksternal di *host machine* atau *container* lain ke direktori "**/data**". Nama dari *container image* yang dibuat adalah **alpine-volume:latest**.

Instruksi `VOLUME` memiliki format penulisan:

```
VOLUME /lokasi/direktori
```

Atau

```
VOLUME /lokasi/direktori1 /lokasi/direktori2
```

```
VOLUME ["/lokasi/direktori1", "/lokasi/direktori2", …]
```

Argumen `/lokasi/direktori` merupakan *path* atau lokasi direktori *mounting* dari *volume*.

Adapun langkah-langkah pembuatan *container image* dengan ketentuan tersebut dan mengujicoba hasil pembuatannya adalah sebagai berikut:

1. Membuat *file* bernama **Dockerfile** menggunakan *editor* **nano**.

   ```
   $ nano Dockerfile
   ```

   Konten dari *file* akan terlihat seperti berikut:

   ```
   FROM alpine:latest
   RUN mkdir data
   COPY dokumen/* /data
   VOLUME /data
   ```

   Terlihat pada baris pertama terdapat instruksi `FROM` untuk menggunakan *base image* **alpine:latest** ketika *build stage*. Instruksi `RUN` pada baris kedua digunakan untuk direktori bernama "**data**". Sedangkan pada baris ketiga terdapat instruksi `COPY` yang digunakan untuk menyalinkan seluruh *file* yang terdapat di direktori "**dokumen**" pada *host machine* ke dalam direktori "**/data**" pada *container image*. Instruksi `VOLUME` pada baris terakhir digunakan untuk membuat *volume mount* ke direktori "**/data**" di dalam *container* yang dijalankan.

   Simpan perubahan dengan menekan tombol **CTRL+O** dan tekan **Enter**.

   Keluar dari *editor nano* dengan menekan tombol **CTRL+X**.

2. Membuat *docker container images* dengan mengeksekusi perintah:

   ```
   $ docker image build -t alpine-volume:latest .
   ```

```
belajar@sabarmenanti:~/images$ docker build -t alpine-volume:latest .
[+] Building 13.5s (9/9) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 105B
 => [internal] load metadata for docker.io/library/alpine:latest
 => [auth] library/alpine:pull token for registry-1.docker.io
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [1/3] FROM docker.io/library/alpine:latest@sha256:b89d9c93e9ed3597455c90a0b88a8bbb5cb7188438f70953fede212a0c4394e0
 => [internal] load build context
 => => transferring context: 140B
 => CACHED [2/3] RUN mkdir data
 => CACHED [3/3] COPY dokumen/* /data
 => exporting to image
 => => exporting layers
 => => writing image sha256:11cf9412590a875a4fb703cb9a7789a0768fb0cebada7cf295857b88ade85b4d
 => => naming to docker.io/library/alpine-volume:latest
```

3. Memverifikasi hasil pembuatan *container images*.

```
belajar@sabarmenanti:~/images$ docker image ls
REPOSITORY          TAG       IMAGE ID       CREATED          SIZE
alpine-volume       latest    11cf9412590a   29 minutes ago   7.8MB
nginx-custom        latest    60a34ac700be   3 hours ago      188MB
ubuntu-with-ping    latest    31dee90bda7d   5 hours ago      117MB
salam               latest    fcdb4bb5adae   7 hours ago      7.8MB
sederhana           latest    167cda670993   16 hours ago     7.8MB
<none>              <none>    62f35d0abb55   17 hours ago     7.8MB
<none>              <none>    e565fa45b2a3   19 hours ago     7.8MB
<none>              <none>    38d99562a9c9   21 hours ago     7.8MB
<none>              <none>    45cc751d62ea   21 hours ago     7.8MB
<none>              <none>    ec85034aa87a   21 hours ago     7.8MB
<none>              <none>    4b6219f77335   23 hours ago     188MB
<none>              <none>    f366735f05d6   27 hours ago     7.8MB
nginx               latest    fffffc90d343   2 weeks ago      188MB
```

Terlihat *container image* tersebut telah berhasil dibuat.

4. Memverifikasi hasil pengaturan instruksi `VOLUME` pada *container image* tersebut dengan mengeksekusi perintah:

   $ `docker image inspect alpine-volume:latest`

   Cuplikan *output* dari hasil eksekusi perintah ini adalah sebagai berikut:

```
        "Config": {
            "Hostname": "",
            "Domainname": "",
            "User": "",
            "AttachStdin": false,
            "AttachStdout": false,
            "AttachStderr": false,
            "Tty": false,
            "OpenStdin": false,
            "StdinOnce": false,
            "Env": [
                "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
            ],
            "Cmd": [
                "/bin/sh"
            ],
            "Image": "",
            "Volumes": {
                "/data": {}
            },
            "WorkingDir": "",
```

   Atau dengan menggunakan pemfilteran sehingga hanya menampilkan nilai hasil dari instruksi `VOLUME` maka dapat mengeksekusi perintah:

```
$ docker image inspect alpine-volume:latest \
-f "{{.Config.Volumes}}"
```

```
belajar@sabarmenanti:~/images$ docker image inspect alpine-volume:latest \
> -f "{{.Config.Volumes}}"
map[/data:{}]
```

Terlihat *output* nilai dari **Volumes** berupa **/data** pada *container image*.

5. Ujicoba membuat dan menjalankan *container* menggunakan *images* **alpine-volume:latest** serta mengakses terminal *container* berjalan secara interaktif. Selain itu juga menggunakan *volume* **vol-data** yang dipasangkan atau di *mounting* ke direktori **/data** pada *container* berjalan.

   ```
   $ docker run --rm -it -v vol-data:/data alpine-volume:latest
   ```

```
belajar@sabarmenanti:~/images$ docker run --rm -it -v vol-data:/data alpine-volume:latest
/ #
/ #
```

Terlihat *prompt* **#** sebagai penanda telah berada di dalam terminal dari *container* berjalan.

6. Berpindah ke direktori /data dan melihat isi direktori tersebut.

   ```
   $ cd /data
   $ ls
   ```

```
/ # cd /data
/data # ls
kedua.txt    ketiga.txt   pertama.txt
```

Terlihat terdapat 3 (tiga) *file* di dalam direktori **data** tersebut yaitu **pertama.txt**, **kedua.txt** dan **ketiga.txt**.

7. Membuat *file* bernama “baru.txt” di dalam direktori **data** tersebut dengan konten “**Data pada file baru**” untuk menguji penyimpanan data secara permanen di *volume* **vol-data**.

   ```
   # echo "Data pada file baru" > baru.txt
   ```

```
/data # echo "Data pada file baru" > baru.txt
```

8. Memverifikasi hasil pembuatan *file* “baru.txt”

   ```
   $ ls
   $ cat baru.txt
   ```

```
/data # ls
baru.txt     kedua.txt    ketiga.txt   pertama.txt
/data # cat baru.txt
Data pada file baru
```

Terlihat *file* tersebut berhasil dibuat dengan konten yang telah ditentukan.

9. Keluar dari penggunaan container berjalan dengan mengeksekusi perintah **exit**.

   ```
   # exit
   ```

```
/data # exit
```

10. Ujicoba membuat dan menjalankan kembali *container* menggunakan *images* **alpine-volume:latest** serta mengakses terminal *container* berjalan secara interaktif untuk memverifikasi data pada *volume* **vol-data** masih dapat diakses. Selain itu juga menggunakan *volume* **vol-data** yang dipasangkan atau di *mounting* ke direktori **/data** pada *container* berjalan.

```
$ docker run --rm -it -v vol-data:/data alpine-volume:latest
```

```
belajar@sabarmenanti:~/images$ docker run --rm -it -v vol-data:/data alpine-volume:latest
/ #
/ #
```

Terlihat *prompt* **#** sebagai penanda telah berada di dalam terminal dari *container* berjalan.

11. Berpindah ke direktori /data dan melihat isi direktori tersebut.

```
$ cd /data
$ ls
```

```
/ # cd /data
/data # ls
baru.txt      kedua.txt     ketiga.txt    pertama.txt
```

Terlihat masih terdapat *file* **baru.txt** yang dibuat.

12. Menampilkan konten dari *file* **baru.txt**.

```
$ cat baru.txt
```

```
/data # cat baru.txt
Data pada file baru
```

13. Keluar dari penggunaan container berjalan dengan mengeksekusi perintah **exit**.

```
# exit
```

```
/data # exit
```

L. Docker Hub Registry

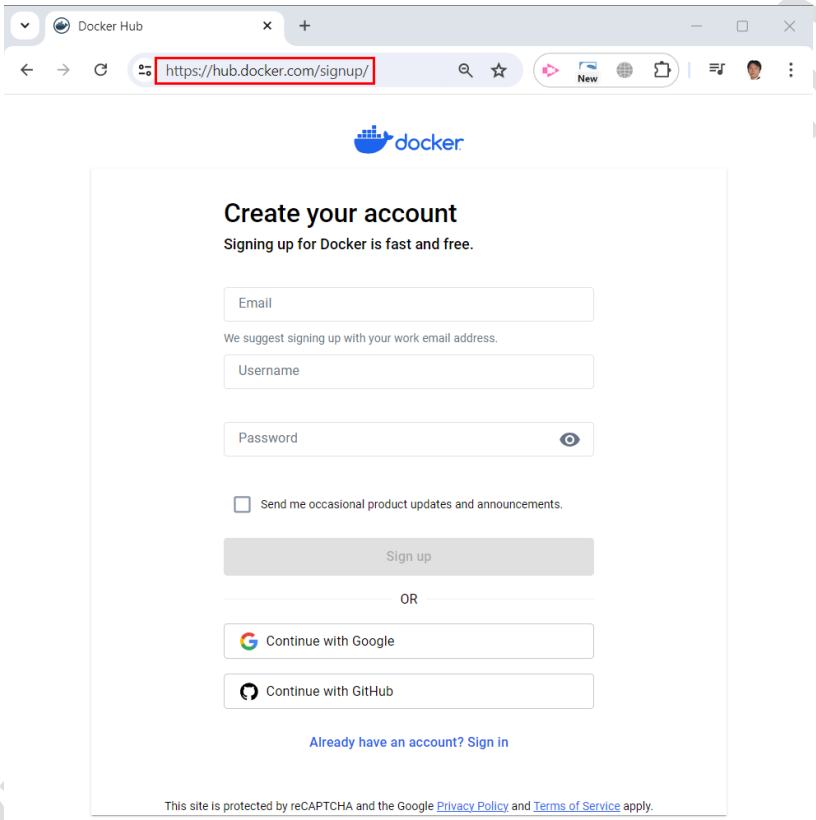
Docker hub dengan alamat https://hub.docker.com merupakan layanan yang disediakan oleh *Docker* untuk menemukan dan berbagi pakai *images* dari *container*. *Docker Hub* memiliki berbagai fitur berdasarkan informasi pada situsnya yaitu meliputi:

a. **Repositories** untuk mengunggah dan mengunduh *container images*,
b. **Builds** untuk secara otomatis melakukan pembuatan *container images* dari GitHub dan Bitbucket dan mengunggahnya ke *Docker Hub*,
c. **Webhooks** untuk memicu tindakan atau aksi setelah keberhasilan mengunggah ke repository maka dapat mengintegrasikan *Docker Hub* ke layanan lain,

d. **Docker Hub Command Line Interface (CLI)** dan **Application Programming Interface (API)** yang digunakan untuk berinteraksi dengan *Docker Hub*.

Registrasi di Docker Hub

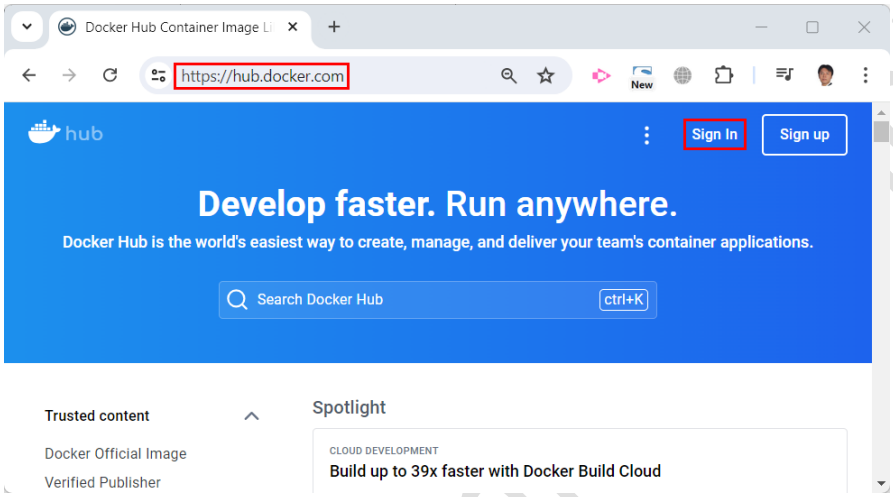
Sebelum dapat membuat *repository* di *Docker Hub* maka diperlukan registrasi terlebih dahulu menggunakan alamat *email* atau akun **Google** atau akun **GitHub** yang dimiliki pada alamat https://hub.docker.com/signup/, seperti terlihat pada gambar berikut:



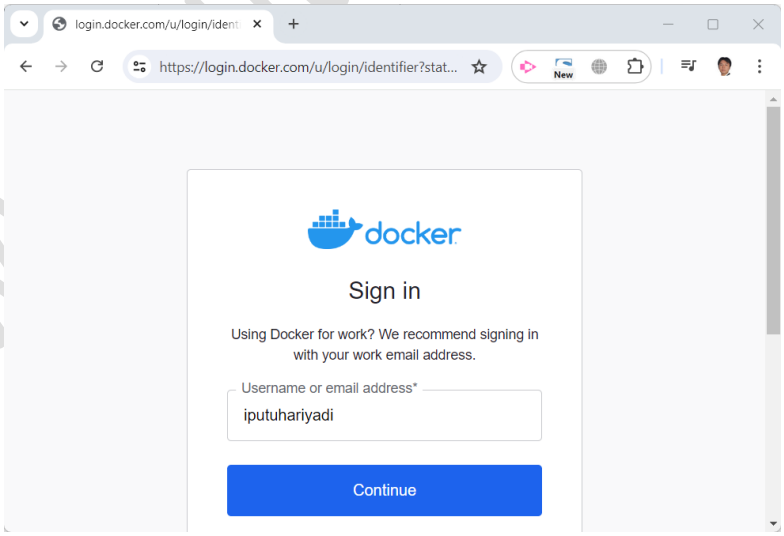
Terlihat tampil form dengan beberapa inputan untuk kelengkapan registrasi. Apabila registrasi menggunakan alamat email maka silakan melengkapi inputan **Email** dengan alamat *email* yang akan digunakan, inputan **Username** yang digunakan sebagai **Docker ID** dengan panjang antara 4 sampai dengan 30 karakter dan hanya dapat mengandung angka dan huruf kecil. Selain itu juga diminta melengkapi inputan **Password** untuk sandi login dengan panjang minimal 9 karakter. Setelah itu tekan tombol **Sign Up** maka *Docker* akan mengirimkan *email* verifikasi ke alamat *email* yang dicantumkan sebelumnya. Silakan memverifikasi alamat *email* yang telah didaftarkan tersebut untuk menyelesaikan proses registrasi.

Login ke Docker Hub

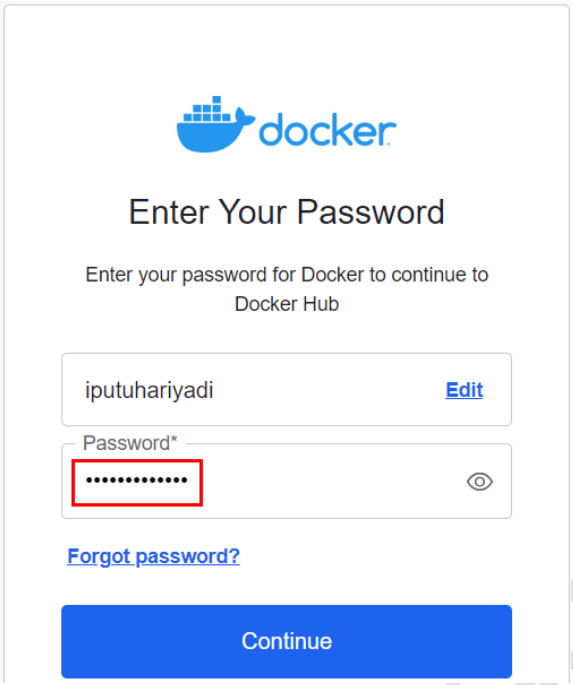
Untuk melakukan *login* ke **Docker Hub** maka dapat mengakses alamat https://hub.docker.com/login atau mengakses https://hub.docker.com dan menekan tombol **Sign In**, seperti terlihat pada gambar berikut:



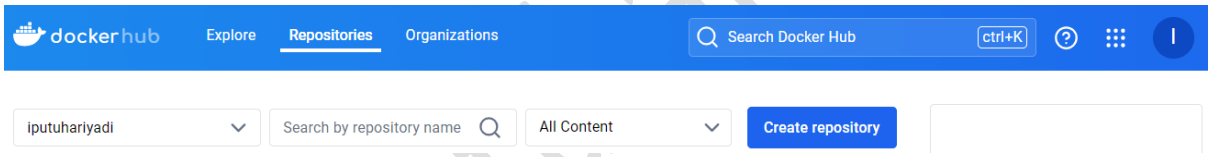
Tampil *form* **Sign in**. Masukkan *username* atau alamat *email* yang telah didaftarkan sebelumnya pada inputan **Username or email address** dan tekan tombol **Continue**, seperti terlihat pada gambar berikut:



Selanjutnya akan tampil *form* **Enter Your Password** yang meminta pengguna untuk memasukkan sandi *login* pada inputan **Password**, seperti terlihat pada gambar berikut:

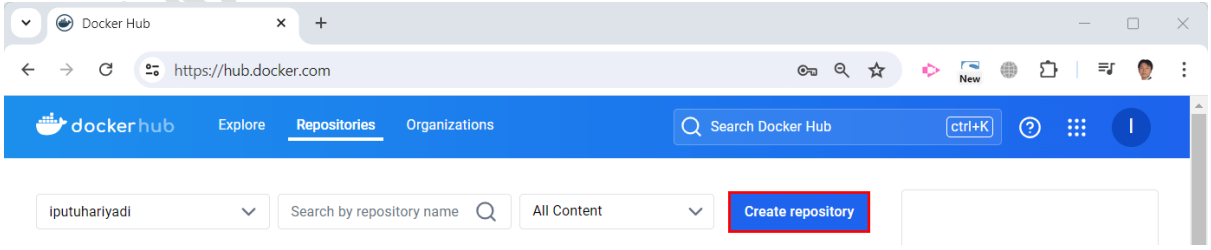


Tekan tombol **Continue** untuk melanjutkan proses *Sign In*. Apabila proses otentikasi berhasil dilakukan maka akan tampil halaman **Repositories**, seperti terlihat pada gambar berikut:



Membuat Repository Public Baru di Docker Hub

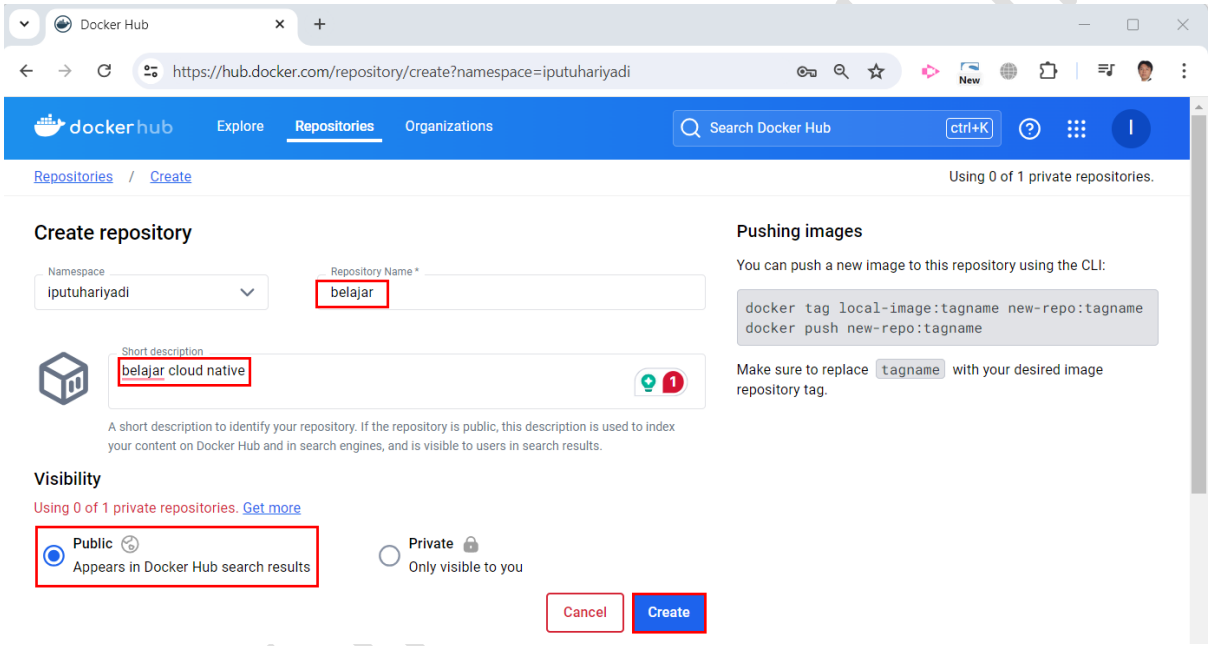
Untuk membuat *repository* baru maka tekan tombol **Create Repository** yang terdapat pada halaman **Repositories**, seperti terlihat pada gambar berikut:



Pada halaman **Create Repository** yang tampil, terdapat form dengan 3 (tiga) inputan yang perlu dilengkapi meliputi:

a) **Repository Name** digunakan untuk mengatur nama repository yang akan dibuat, sebagai contoh bernama "**belajar**".
b) **Short Description** digunakan untuk memberikan keterangan singkat terkait repository yang dibuat. Sebagai contoh "**belajar cloud native**".
c) **Visibility** dengan 2 (dua) pilihan yaitu **Public** agar *repository* dapat diakses oleh publik dan muncul pada pencarian dari *Docker Hub*, dan **Private** agar *repository* hanya dapat diakses oleh diri sendiri (pribadi). Hanya satu *private repository* yang dapat dibuat. Sebagai contoh dipilih **Public**.

Hasil pengaturan akan terlihat seperti pada gambar berikut:



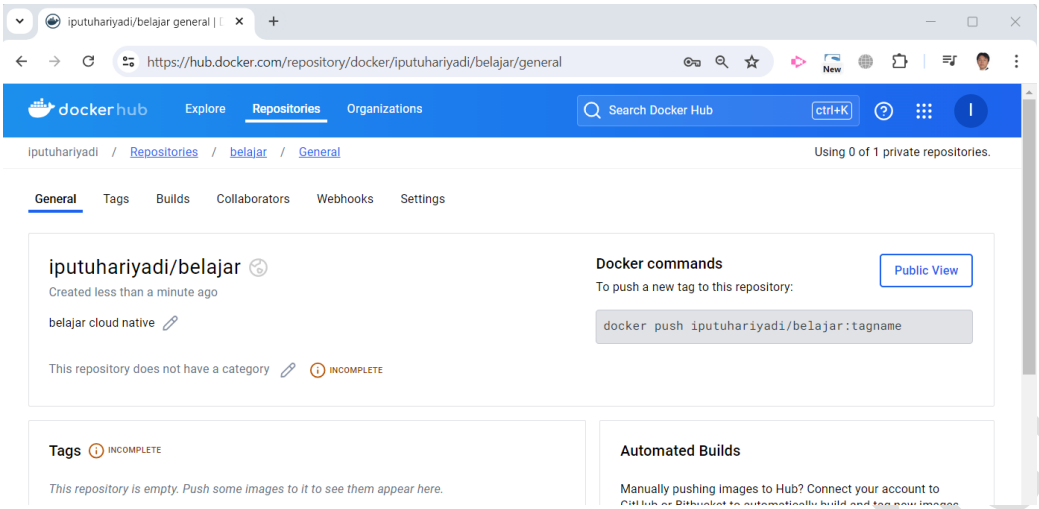
Terlihat pula informasi untuk melakukan mengunggah (*pushing*) *images* ke *repository* yang dibuat melalui CLI dapat mengeksekusi perintah:

```
docker tag local-image:tagname new-repo:tagname
```

```
docker push new-repo:tagname
```

Pastikan mengganti `tagname` dengan *image repository tag* yang diinginkan.

Tekan tombol **Create** untuk memproses pembuatan *repository*. Apabila proses pembuatan *repository* tersebut berhasil dilakukan maka akan tampil halaman dengan informasi dari bagian tab **General** untuk **Repository** "**belajar**", seperti terlihat pada gambar berikut:

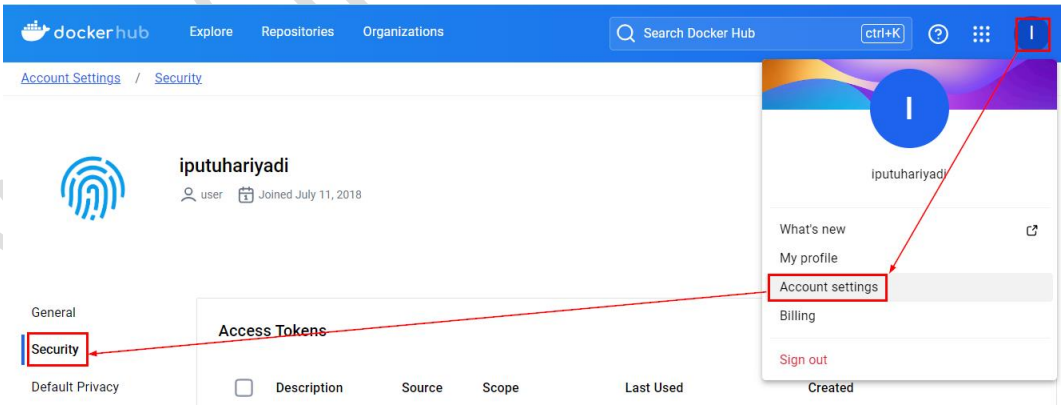


Pada halaman tersebut terdapat informasi perintah yang dapat digunakan untuk mengunggah *container images* ke *repository* tersebut yaitu:

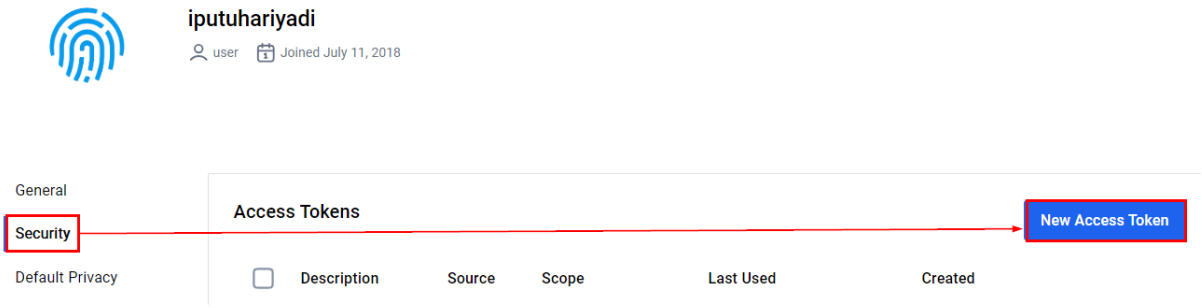
```
docker push iputuhariyadi/belajar:tagname
```

Membuat Access Token di Docker Hub

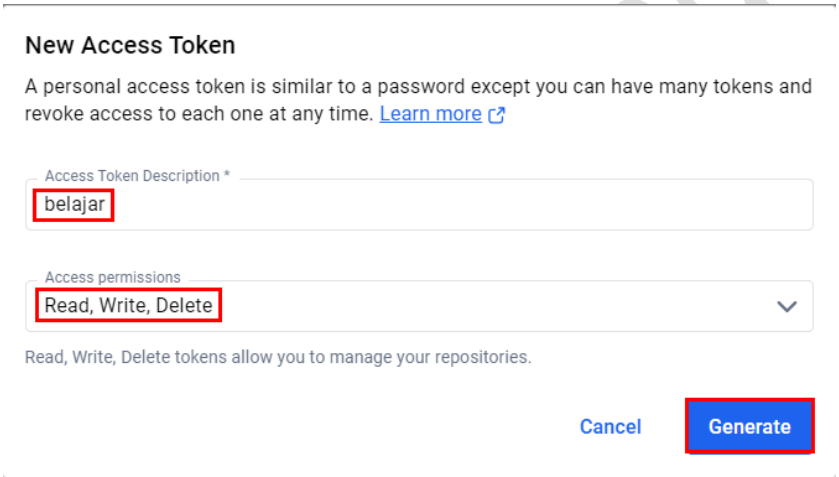
Pengunggahan *container images* ke *Docker Hub* memerlukan otentikasi *login* berupa **username** yaitu **Docker ID** dan **password** berupa **Access Token**. *Access Token* dapat dibuat dengan mengakses menu *Profile* pada halaman *Docker Hub* dan memilih **Account settings**. Kemudian pada halaman **Account Settings** yang tampil, pilih menu **Security** di panel sebelah kiri, seperti terlihat pada gambar berikut:



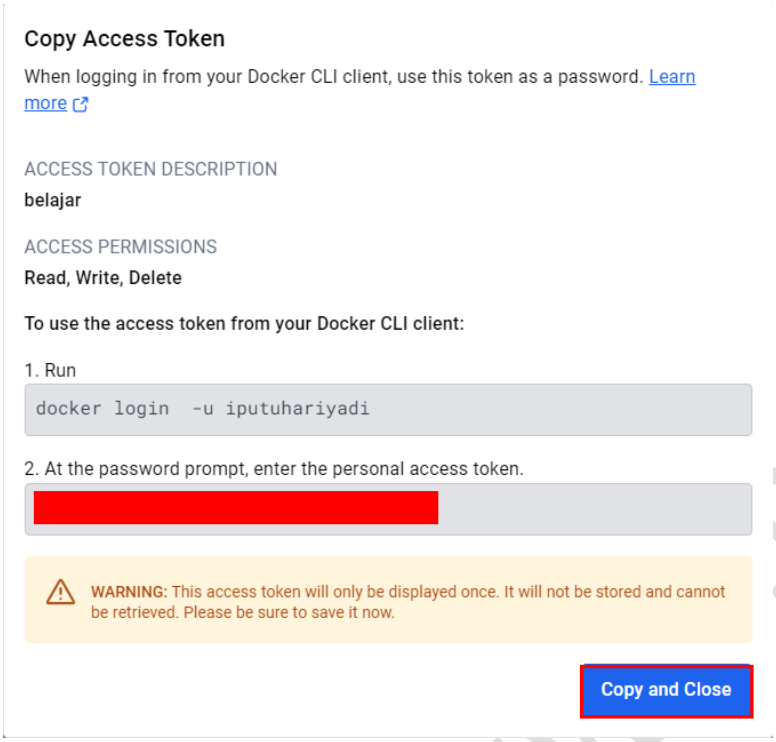
Selanjutnya akan tampil halaman yang menampilkan daftar **Access Tokens**. Tekan tombol **New Access Token** untuk membuat *access token* baru, seperti terlihat pada gambar berikut:



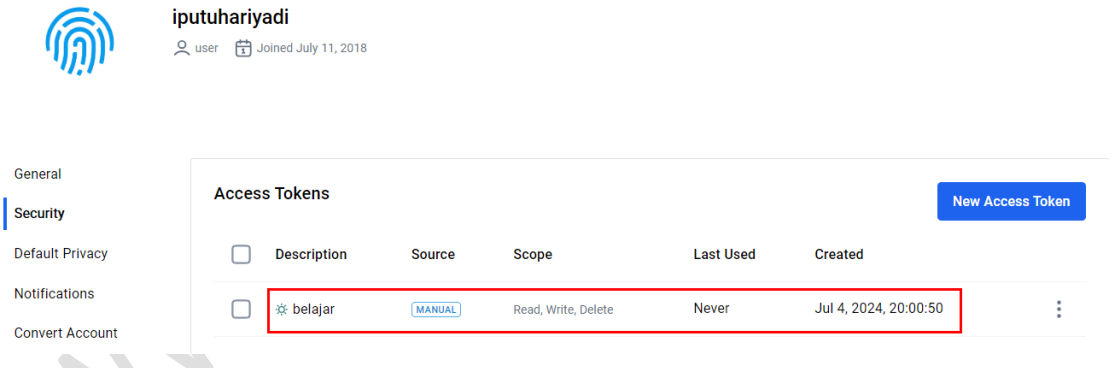
Pada kotak dialog **New Access Token** yang tampil dan lengkapi isian parameter **Access Token Description** dengan deskripsi terkait access token yang dibuat, sebagai contoh "**belajar**". Sedangkan pada *dropdown* **Access permissions**, pilih "**Read, Write, Delete**" agar dapat melakukan operasi *read* (baca), *write* (mengubah) dan *delete* (menghapus) pada *repository*, seperti terlihat pada gambar berikut:



Tekan tombol **Generate** untuk memproses pembuatan *access token*. Apabila proses pembuatan berhasil dilakukan maka akan tampil kotak dialog **Copy Access Token**, seperti terlihat pada gambar berikut:



Tekan tombol **Copy and Close** untuk menyalin *access token* yang telah dihasilkan dan menutup kotak dialog tersebut. Pastikan menyimpan *access token* tersebut dengan baik karena hanya ditampilkan sekali. Selanjutnya akan tampil halaman yang memuat daftar **Access Tokens**, seperti terlihat pada gambar berikut:



Terlihat terdapat *access token* dengan nama "**belajar**" yang telah dibuat sebelumnya.

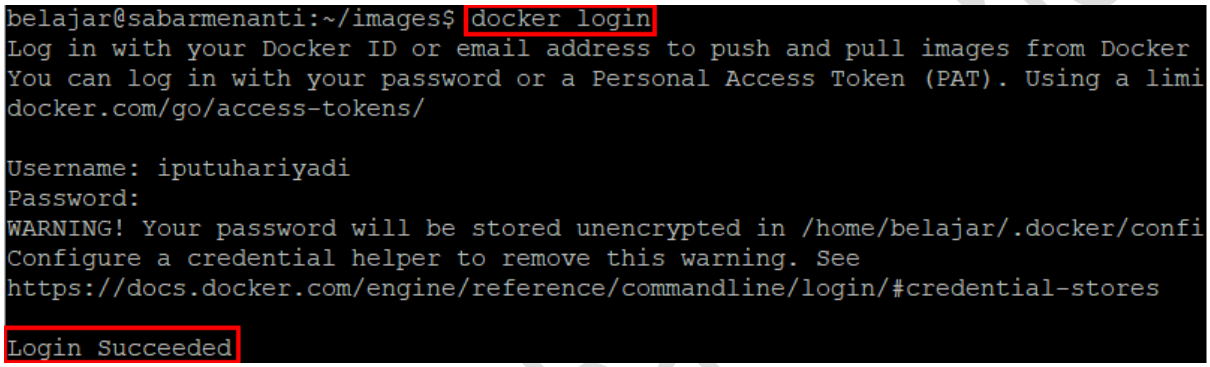
Mengunggah Container Images ke Docker Hub

Sebagai contoh akan dilakukan pengunggahan *container images* yang telah dibuat sebelumnya yaitu dengan nama **sederhana:latest**. Adapun langkah-langkah untuk mengunggah *container images* tersebut ke **Docker Hub** adalah sebagai berikut:

1. *Login* ke **Docker Hub** melalui terminal dengan mengeksekusi perintah berikut:

   ```
   $ docker login
   ```

   Tampil inputan yang meminta pengguna untuk memasukkan **Username** berupa *Docker ID* yang dimiliki. Sebagai contoh milik penulis yaitu "**iputuhariyadi**" dan tekan **Enter**. **Silakan menyesuaikan nilai *username* tersebut dengan yang dimiliki.** Selanjutnya tampil inputan yang meminta pengguna memasukkan **Password** berupa **access token** yang telah dibuat sebelumnya dan tekan **Enter.** Hasilnya akan terlihat seperti pada gambar berikut:



   Terlihat proses otentikasi *login* berhasil dilakukan yang ditandai dengan pesan **"Login Succeeded"**.

2. Membuat *tag* untuk target *container images* yang mengacu ke *images* sumber dengan sintak penulisan perintah:

   ```
   docker image tag SOURCE_IMAGE[:TAG] TARGET_IMAGE[:TAG]
   ```

   Argumen `SOURCE_IMAGE[:TAG]` merupakan nama dan *tag* dari sumber *images*. Sedangkan argumen `TARGET_IMAGE[:TAG]` merupa nama dan *tag* dari target *images* yang akan dibuat.

   Perintah ini memiliki *alias*:

   ```
   docker tag
   ```

   Sesuai dengan contoh maka akan dibuat tag untuk *container images* sumber bernama **sederhana:latest** sebagai **iputuhariyadi/belajar:v1** dengan mengeksekusi perintah:

   ```
   $ docker image tag sederhana:latest iputuhariyadi/belajar:v1
   ```

   atau

   ```
   $ docker tag sederhana:latest iputuhariyadi/belajar:v1
   ```

Silakan menyesuaikan nilai *username* "**iputuhariyadi**" dengan *username* berupa **Docker ID** dan menyesuaikan nama *repository* "**belajar**" dengan nama *repository* yang telah dibuat di **Docker Hub**.

3. Memverifikasi hasil pembuatan *tag* untuk *target container image*.

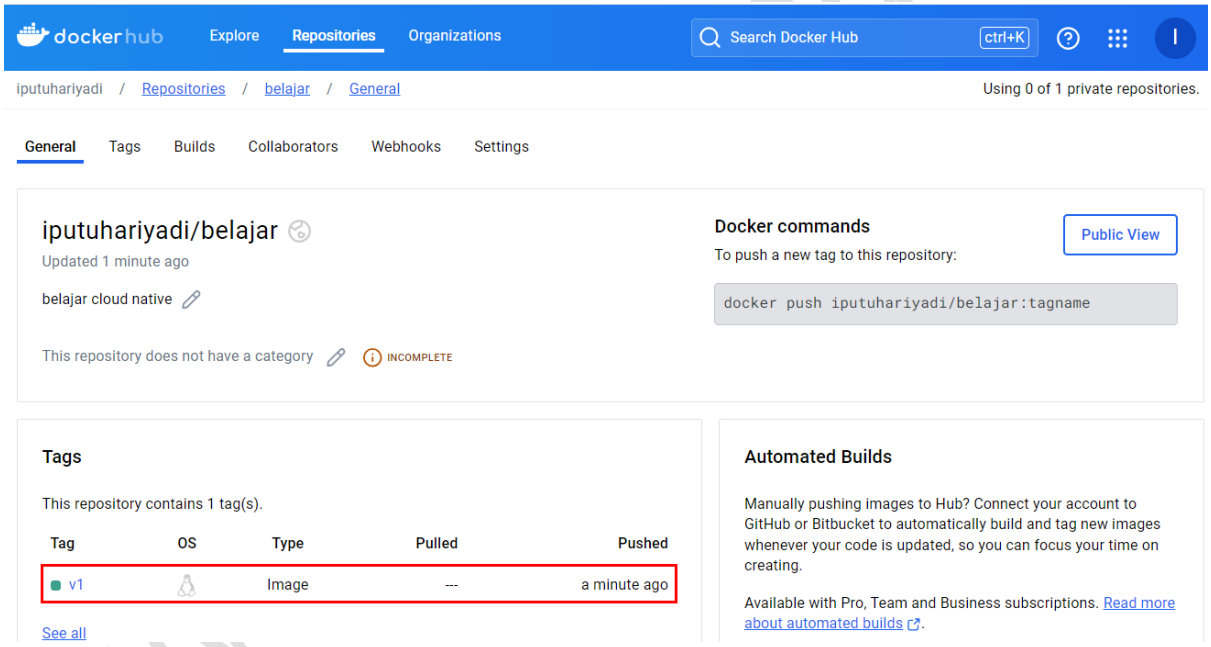
```
$ docker image ls
```

Terlihat telah terdapat *target container images* bernama **iputuhariyadi/belajar:v1**.

4. Mengunggah *container images* bernama **iputuhariyadi/belajar:v1** ke *Docker Hub Repository*.

```
$ docker push iputuhariyadi/belajar:v1
```

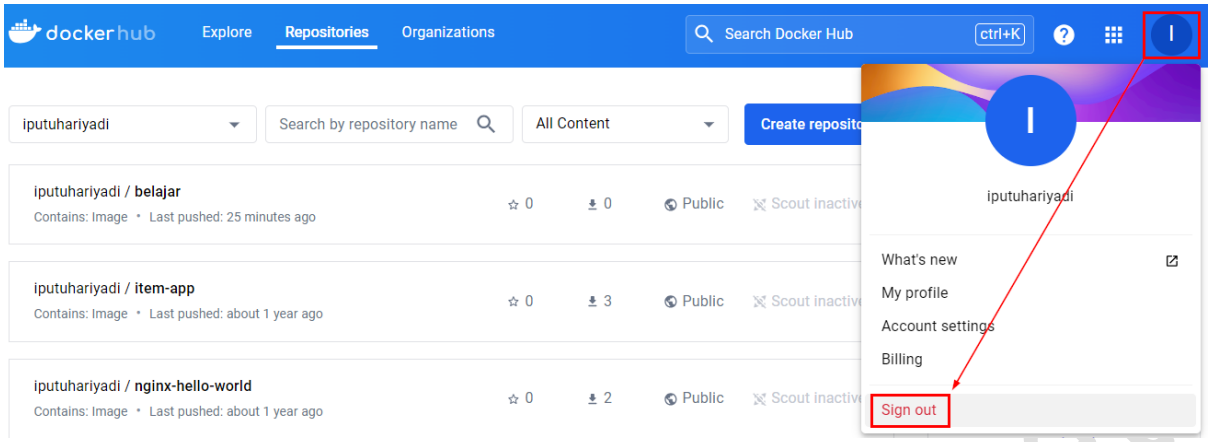
5. Memverifikasi hasil pengunggahan *container images* tersebut di **Docker Hub Repository** "**belajar**", seperti terlihat pada gambar berikut:



Terlihat *container images* dengan *tag* v1 tersebut telah berhasil diunggah.

Keluar dari penggunaan Docker Hub

Untuk keluar dari *Docker Hub*, pilih menu *Profile* pada pojok kanan atas dari halaman **Repositories** dan pilih **Sign Out**, seperti terlihat pada gambar berikut:



BAB VI

MEMBANGUN APLIKASI MULTI-CONTAINER MENGGUNAKAN DOCKER COMPOSE

Menurut dokumentasi dari Docker, **Docker compose** merupakan *tool* yang digunakan untuk mendefinisikan dan menjalankan aplikasi *multi-container*. *Docker compose* memudahkan pengelolaan **service**, **network** dan **volume** dalam sebuah *file* konfigurasi berformat **YAML** sehingga dapat menyederhanakan pengontrolan keseluruhan aplikasi. **YAML Ain't Markup Language (YAML)** merupakan bahasa serialisasi data yang ramah manusia (*human-friendly*) untuk seluruh bahasa pemrograman. *File* konfigurasi YAML dari *Compose* umumnya bernama **compose.yaml** dapat digunakan untuk membuat (**create**) dan menjalankan (**start**) seluruh **service** dengan sebuah perintah. Sebagai gambaran berikut adalah contoh konten dari *file* **compose.yaml** yang digunakan untuk membuat 2 (dua) *container* yaitu **mariadb** dan **phpmyadmin** pada **project** bernama **myapp**:

```
name: myapp

services:
  mariadb:
    image: mariadb:latest
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: sabarmenanti
      MYSQL_DATABASE: perpustakaan
      MYSQL_USER: sabar
      MYSQL_PASSWORD: menanti
    ports:
      - "3306:3306"
    volumes:
      - ./data:/var/lib/mysql
```

```
  phpmyadmin:
    image: phpmyadmin/phpmyadmin
    restart: always
    environment:
      PMA_HOST: mariadb
      PMA_USER: sabar
      PMA_PASSWORD: menanti
    ports:
      - "8080:80"
    depends_on:
      - mariadb
```

File compose.yaml tersebut dibuat dengan mengikuti aturan yang disediakan oleh **Compose Specification** tentang bagaimana mendefinisikan aplikasi *multi-container*. Informasi lebih lanjut tentang *Compose Specification* dapat diakses di alamat https://docs.docker.com/compose/compose-file/.

Compose memiliki perintah-perintah yang digunakan untuk mengelola siklus hidup dari aplikasi meliputi menjalankan, menghentikan dan membangun ulang *service* serta melihat status dari *service* yang berjalan. Selain itu juga dapat melakukan *stream* luaran *log* yang dihasilkan oleh *service* berjalan. *Compose* dapat bekerja di seluruh lingkungan baik *production*, *staging*, *development*, *testing* dan juga alur kerja (*workflow*) **Continuous Integration (CI)**.

Compose telah tersedia bersama *Docker* yang terinstalasi pada sistem yang digunakan. Untuk memverifikasi versi **docker compose** yang terinstalasi pada *Ubuntu Desktop* maka dapat mengeksekusi perintah:

```
$ docker compose version
```

```
belajar@sabarmenanti:~$ docker compose version
Docker Compose version v2.28.1
```

Terlihat versi *Docker Compose* yang terinstalasi adalah **2.28.1**.

Model Aplikasi Compose

Berdasarkan dokumentasi dari Docker Compose, komponen komputasi dari aplikasi didefinisikan sebaga **services**. *Service* merupakan konsep abstrak yang diterapkan pada platform dengan menjalankan *container image* dan konfigurasi yang sama sekali atau lebih.

Services berkomunikasi satu dengan lainnya melalui **networks**. Pada *Compose Specification*, sebuah *network* merupakan kemampuan abstraksi dari platform untuk membentuk perutean IP di antara *container* di dalam *services* yang dihubungkan bersama.

Services menyimpan dan berbagi data secara permanen ke dalam **volumes**.

A. Manajemen Container Dengan Docker Compose

Sebagai contoh akan dibuat *file* **compose.yaml** yang digunakan untuk membuat satu *container* yaitu bernama **nginx-compose** dengan langkah-langkah penyelesaian sebagai berikut:

1. Membuat direktori baru dengan nama "**container**" di dalam direktori "**compose**".
   ```
   $ mkdir -p compose/container
   ```
2. Berpindah ke dalam direktori "**compose/container**".
   ```
   $ cd compose/container
   ```
3. Menampilkan informasi di direktori mana saat ini berada
   ```
   $ pwd
   ```
4. Membuat *file* **compose.yaml** menggunakan editor **nano**.
   ```
   $ nano compose.yaml
   ```
   Konten dari *file* akan terlihat seperti berikut:

```
name: container

services:
  # Mendefinisikan service untuk nginx-contoh
  nginx-contoh:
    container_name: nginx-contoh
    image: nginx:latest
    ports:
```

```
    - "80:80"
```

Elemen *top-level* `name` digunakan untuk mendefinisikan nama *project* yaitu **container**. Sedangkan elemen *top-level* **services** pada baris berikutnya digunakan untuk mendefinisikan secara abstrak sumber daya komputasi di dalam komputasi yang dapat diskalakan atau diganti secara independen dari komponen lainnya. Selanjutnya terdapat baris komentar yang diawali dengan tanda #. **Services** didukung oleh satu *container* yaitu **nginx-contoh** dengan atribut `container_name` yang digunakan untuk mengatur nama *container* yang dibuat yaitu **nginx-contoh** dan menggunakan *image* **nginx:latest** berdasarkan nilai dari atribut `image`. Selain itu atribut `ports` digunakan untuk mendefinisikan pemetaan *port* antara *host machine* yaitu pada *port* 80 ke *port* 80 di dalam *container*.

Simpan perubahan dengan menekan tombol **CTRL+O** dan tekan **Enter**.

Keluar dari *editor nano* dengan menekan tombol **CTRL+X**.

5. Membuat *container* untuk sebuah **service**.

   ```
   $ docker compose create
   ```

```
belajar@sabarmenanti:~/compose/container$ docker compose create
[+] Creating 2/2
 ✔ Network container_default  Created
 ✔ Container nginx-contoh     Created
```

6. Memverifikasi *container* telah berhasil dibuat.

```
belajar@sabarmenanti:~/compose/container$ docker compose ps -a
NAME           IMAGE          COMMAND                  SERVICE        CREATED              STATUS    PORTS
nginx-contoh   nginx:latest   "/docker-entrypoint.…"   nginx-contoh   About a minute ago   Created
```

   Terlihat terdapat *container* **nginx-contoh** dengan status **Created**.

7. Menjalankan *service*.

   ```
   $ docker compose start
   ```

```
belajar@sabarmenanti:~/compose/container$ docker compose start
[+] Running 1/1
 ✔ Container nginx-contoh  Started
```

8. Memverifikasi *container* telah berhasil berjalan.

```
belajar@sabarmenanti:~/compose/container$ docker compose ps
NAME           IMAGE          COMMAND                  SERVICE        CREATED         STATUS         PORTS
nginx-contoh   nginx:latest   "/docker-entrypoint.…"   nginx-contoh   7 minutes ago   Up 5 seconds   80/tcp
```

   Terlihat terdapat *container* **nginx-contoh** dengan status telah aktif 5 detik yang lalu (**Up 5 seconds**).

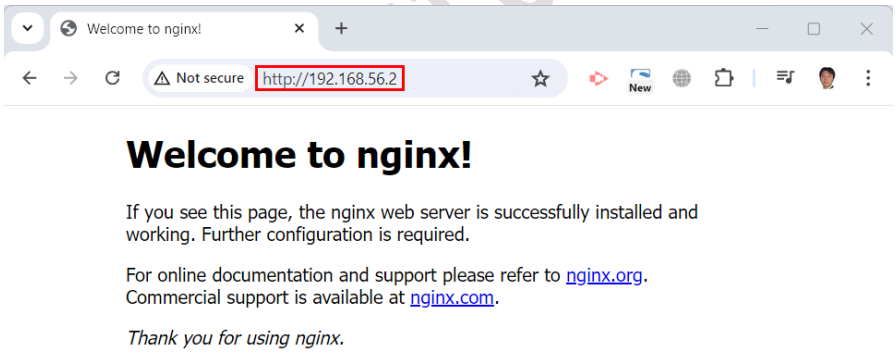
9. Memverifikasi akses ke layanan *web* dari *container* **nginx-contoh** pada **port 80** menggunakan aplikasi **curl**.

```
$ curl localhost
```

```
belajar@sabarmenanti:~/compose/container$ curl localhost
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

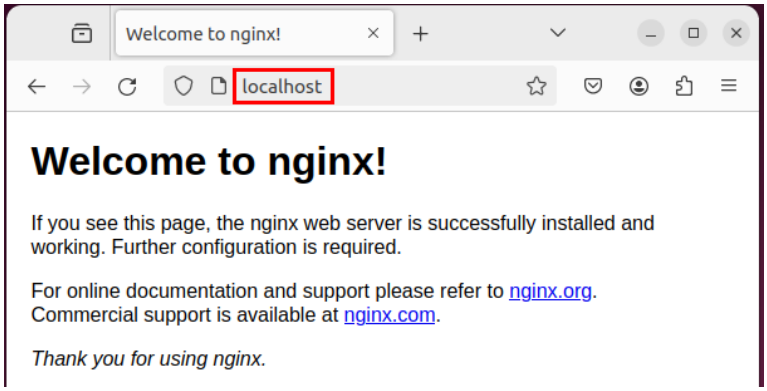
Terlihat konten *homepage* dari *container* **nginx-contoh** berhasil diakses.

10. Memverifikasi akses ke layanan web dari *container* **nginx-contoh** pada **port 80** menggunakan *browser* **Chrome** atau lainnya di **Windows 11** ke alamat **http://192.168.56.2**.



Terlihat konten *homepage* dari *container* **nginx-contoh** berhasil diakses.

11. Memverifikasi akses ke layanan web dari *container* **nginx-contoh** pada **port 80** menggunakan *browser* **Firefox** di **VM Ubuntu Desktop** ke alamat **http://localhost**.



Terlihat konten *homepage* dari *container* **nginx-contoh** berhasil diakses.

12. Menghentikan *service* sehingga *container* yang berjalan akan dihentikan namun tanpa menghapusnya.

```
$ docker compose stop
```

```
belajar@sabarmenanti:~/compose/container$ docker compose stop
[+] Stopping 1/1
 ✔ Container nginx-contoh  Stopped
```

13. Memverifikasi *container* telah berhasil dihentikan.

```
$ docker compose ps -a
```

```
belajar@sabarmenanti:~/compose/container$ docker compose ps -a
NAME           IMAGE          COMMAND                  SERVICE        CREATED          STATUS                      PORTS
nginx-contoh   nginx:latest   "/docker-entrypoint.…"   nginx-contoh   17 minutes ago   Exited (0) 8 seconds ago
```

Terlihat terdapat *container* **nginx-contoh** dengan status telah berhenti 8 detik yang lalu (**Exited (0) 8 seconds ago**).

14. Menghapus *service container* yang telah dihentikan.

```
$ docker compose rm
```

```
belajar@sabarmenanti:~/compose/container$ docker compose rm
? Going to remove nginx-contoh Yes
[+] Removing 1/0
 ✔ Container nginx-contoh  Removed
```

Ketik **Y** pada inputan konfirmasi "**? Going to remove nginx-contoh**" untuk memproses penghapusan *service container*.

15. Memverifikasi container telah berhasil dihapus.

```
$ docker compose ps -a
```

```
belajar@sabarmenanti:~/compose/container$ docker compose ps -a
NAME      IMAGE     COMMAND   SERVICE   CREATED   STATUS    PORTS
```

B. Docker Compose Untuk Manajemen Container MariaDB dan phpMydmin

Sebagai contoh akan dibuat *file* **compose.yaml** yang digunakan untuk membuat 2 (dua) *container* yaitu bernama **mariadb** menggunakan *image* **mariadb:latest** dan **phpMyAdmin** menggunakan *image* **phpmyadmin/phpmyadmin** dengan langkah-langkah penyelesaian sebagai berikut:

1. Berpindah ke satu direktori sebelumnya.
   ```
   $ cd ..
   ```
2. Membuat direktori baru dengan nama "**myapp**".
   ```
   $ mkdir myapp
   ```
3. Berpindah ke dalam direktori "**myapp**".
   ```
   $ cd myapp
   ```
4. Membuat *file* **compose.yaml** menggunakan editor **nano**.
   ```
   $ nano compose.yaml
   ```
   Konten dari *file* akan terlihat seperti berikut:

```
name: myapp

services:
  mariadb:
    image: mariadb:latest
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: sabarmenanti
      MYSQL_DATABASE: perpustakaan
      MYSQL_USER: sabar
      MYSQL_PASSWORD: menanti
    ports:
      - "3306:3306"
    volumes:
      - ./data:/var/lib/mysql

  phpmyadmin:
```

```
image: phpmyadmin/phpmyadmin
restart: always
environment:
  PMA_HOST: mariadb
  PMA_USER: sabar
  PMA_PASSWORD: menanti
ports:
  - "8080:80"
depends_on:
    - mariadb
```

Terlihat terdapat 2 (dua) nama *service* yang digunakan untuk membuat dua container yaitu **mariadb** dan **phpmyadmin**. Setiap *service* memiliki atribut `environment` yang digunakan untuk mengatur *environment variable* di masing-masing *container*. *Service* **mariadb** memiliki empat *environment variable* yaitu `MYSQL_ROOT_PASSWORD` bernilai `sabarmenanti` digunakan untuk mengatur sandi dari *user* **root** *MariaDB*, `MYSQL_DATABASE` bernilai `perpustakaan` digunakan untuk membuat *database* **perpustakaan**, `MYSQL_USER` bernilai `sabar` digunakan untuk membuat user bernama **sabar** di *MariaDB* dan `MYSQL_PASSWORD` bernilai `menanti` digunakan untuk mengatur sandi dari *user* **sabar .** Sedangkan atribut `volumes` digunakan untuk melakukan *bind mount* dengan *short syntax* yaitu sumber "**./data**" di *host machine* VM *Ubuntu Desktop* ke target di dalam container "**/var/lib/mysql**".

Service **phpmyadmin** memiliki tiga *environment variable* yaitu `PMA_HOST` bernilai `mariadb` digunakan untuk mengatur *hostname* dari *server MariaDB* yaitu **mariadb** sesuai dengan nama containernya. *Environment variable* `PMA_USER` bernilai `sabar` digunakan untuk mengatur *user* yang digunakan oleh *phpMyAdmin* ketika mengakses *server MariaDB* yaitu `sabar`. Sedangkan *environment variable* `PMA_PASSWORD` bernilai `menanti` digunakan untuk mengatur sandi dari *user* **sabar .** *User* dan *password* ini sesuai dengan nilai yang ditetapkan pada *environment variable service container* **mariadb** yaitu `MYSQL_USER` dan `MYSQL_PASSWORD`.

Atribut `restart` pada setiap *service* digunakan untuk mendefinisikan kebijakan pada platform yang diterapkan ketika terminasi dari *container* yaitu **always**. Kebijakan ini selalu memulai ulang *container* hingga *container* dihapus.

Selanjutnya atribut `depends_on` pada *service* **phpmyadmin** digunakan untuk mengontrol urutan *startup* dan *shutdown* dari *service*. *Service* **phpmyadmin** bergantung pada *service* **mariadb** sehingga urutan *startup* akan mempengaruhi fungsionalitas dari aplikasi. Dalam hal ini *service* **mariadb** harus berjalan terlebih dahulu sebelum *service* **phpmyadmin** sehingga layanan dari *phpMyAdmin* dapat digunakan oleh pengguna.

Simpan perubahan dengan menekan tombol **CTRL+O** dan tekan **Enter**.

Keluar dari *editor nano* dengan menekan tombol **CTRL+X**.

5. Membuat dan menjalankan *containers services* di *background*.

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

```
belajar@sabarmenanti:~/compose/myapp$ docker compose up -d
[+] Running 28/28
 ✔ mariadb Pulled
   ✔ 9c704ecd0c69 Already exists
   ✔ f8f7f3c9a741 Pull complete
   ✔ 97c034108521 Pull complete
   ✔ 50366979c20a Pull complete
   ✔ 0221331af5c0 Pull complete
   ✔ e3c4d1c9d9cb Pull complete
   ✔ eef7a8467f98 Pull complete
   ✔ 60c15bb5bb03 Pull complete
 ✔ phpmyadmin Pulled
   ✔ faef57eae888 Pull complete
   ✔ 989a1d6c052e Pull complete
   ✔ 0705c9c2f22d Pull complete
   ✔ 621478e043ce Pull complete
   ✔ 98246dcca987 Pull complete
   ✔ bfed8c155cb6 Pull complete
   ✔ 7a7c2e908867 Pull complete
   ✔ d176994b625c Pull complete
   ✔ 2d8ace6a2716 Pull complete
   ✔ c70df516383c Pull complete
   ✔ 15e1b44fe4c7 Pull complete
   ✔ 65e50d44e95a Pull complete
   ✔ 77f68910bc0a Pull complete
   ✔ 605dd3a6e332 Pull complete
   ✔ 99ce27188f07 Pull complete
   ✔ 74d64e32c5d5 Pull complete
   ✔ ef5fc9928b9f Pull complete
   ✔ 163f3256e112 Pull complete
[+] Running 3/3
 ✔ Network myapp_default          Created
 ✔ Container myapp-mariadb-1      Started
 ✔ Container myapp-phpmyadmin-1   Started
```

6. Memverifikasi hasil *containers* telah berjalan.

```
$ docker compose ps
```

```
belajar@sabarmenanti:~/compose/myapp$ docker compose ps
NAME                IMAGE                  COMMAND                  SERVICE      CREATED         STATUS         PORTS
myapp-mariadb-1     mariadb:latest         "docker-entrypoint.s…"   mariadb      9 seconds ago   Up 7 seconds   0.0.0.0:3306->3306/tcp, :::3306->3306/tcp
myapp-phpmyadmin-1  phpmyadmin/phpmyadmin  "/docker-entrypoint.…"   phpmyadmin   8 seconds ago   Up 6 seconds   0.0.0.0:8080->80/tcp, :::8080->80/tcp
```

7. Memverifikasi koneksi ke *docker container* **mariadb** dari *host machine* menggunakan utilitas *mysql client* dengan parameter-parameter berikut:
   - `-h` untuk mengatur hostname dari server mariadb yang akan diakses yaitu **localhost**.
   - `-u` untuk mengatur nama login pengguna yang digunakan untuk mengakses ke *server mariadb* yaitu **root**.
   - `-p` digunakan untuk mengatur inputan sandi untuk nama login pengguna.
   - `-P` digunakan untuk mengatur nomor *port* dari *server mariadb* yaitu **3306**.

   Sehingga perintahnya terlihat seperti berikut:

   ```
   $ mysql -h localhost -u root -p -P 3306
   ```

   Tampil inputan **Enter password**, masukkan `sabarmenanti` & tekan **Enter**.

```
belajar@sabarmenanti:~/compose/myapp$ mysql -h localhost -u root -p -P 3306
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 24
Server version: 11.4.2-MariaDB-ubu2404 mariadb.org binary distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]>
```

   Terlihat koneksi ke *server mariadb* berhasil dilakukan ditandai dengan penampilan *prompt* `MariaDB [(none)]>`

   a. Menampilkan informasi *database* yang telah ada di *server mariadb*.

      ```
      show databases;
      ```

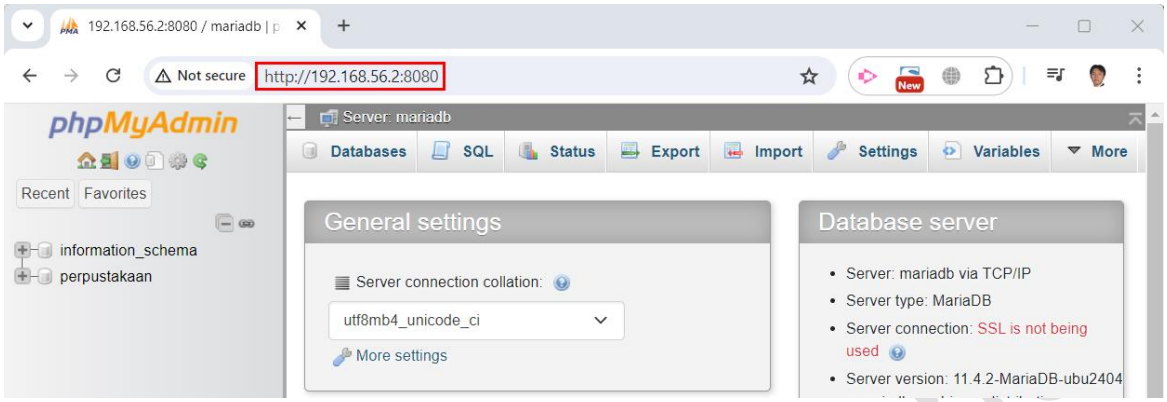
```
MariaDB [(none)]> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| perpustakaan       |
| sys                |
+--------------------+
5 rows in set (0,001 sec)
```

      Terlihat terdapat 5 (lima) *database* yaitu *information_schema*, *mysql*, *performance_schema* dan **perpustakaan** serta *sys*. *Database* **perpustakaan** berhasil dibuat ketika membuat dan menjalankan *container* di tahap sebelumnya.

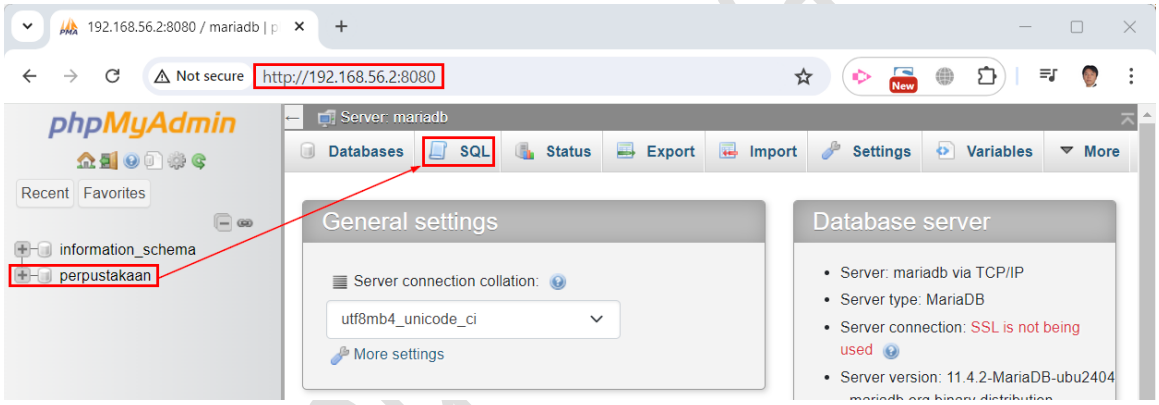
   b. Keluar dari *server mariadb*.

      ```
      quit;
      ```

8. Mengakses layanan **phpMyAdmin** melalui *browser* di *Windows 11* pada alamat http://192.168.56.2:8080, seperti terlihat pada gambar berikut:



9. Membuat tabel **buku** pada *database* **perpustakaan** dengan cara memilih **perpustakaan** pada panel sebelah kiri dari **dashboard phpMyAdmin** yang tampil. Selanjutnya pada panel detail sebelah kanan pilih **SQL**, seperti terlihat pada gambar berikut:



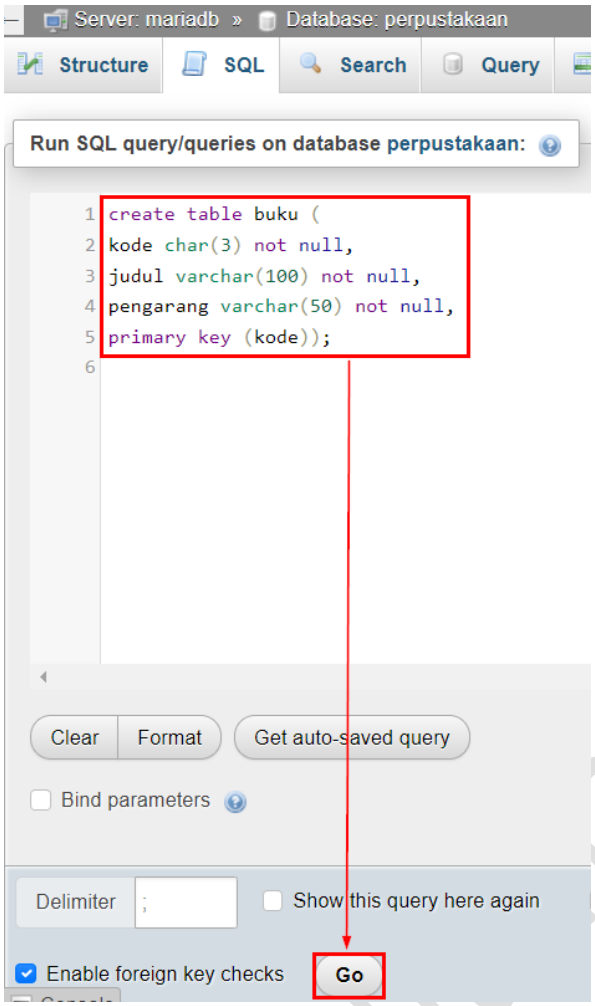
Tabel **buku** yang dibuat di dalam *database* **perpustakaan** memiliki struktur dan konten seperti terlihat pada tabel berikut:

kode	judul	pengarang
001	Docker Deep Dive	Nigel Poulton
002	The Kubernetes Book	Nigel Poulton

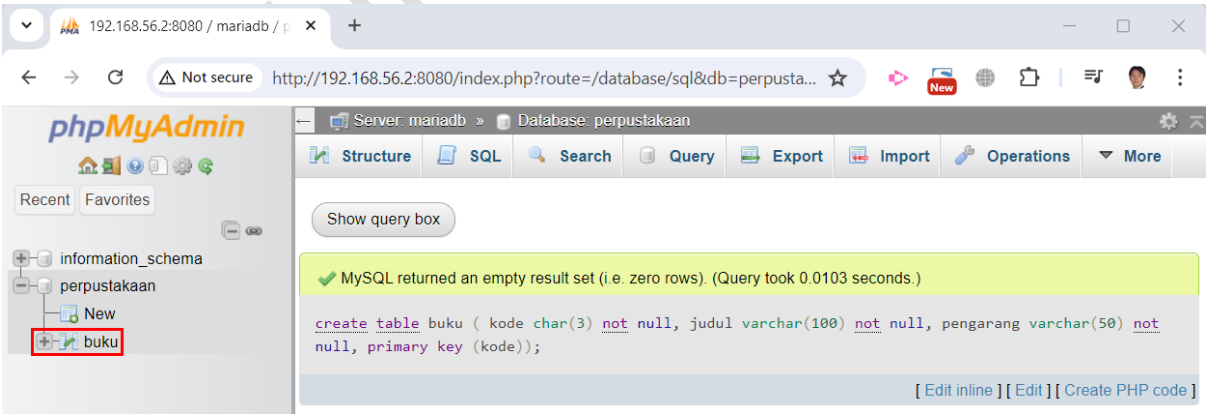
Tabel dengan struktur tersebut dapat dibuat menggunakan **Data Definition Languange (DDL)** berikut yang diinputkan pada *textarea* **Run SQL query/queries on database perpustakaan:**

```
create table buku (
kode char(3) not null,
judul varchar(100) not null,
pengarang varchar(50) not null,
primary key (kode));
```

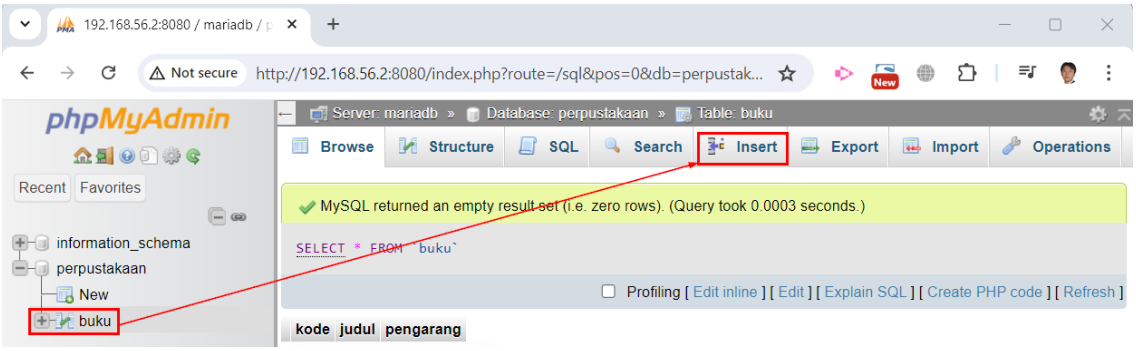
Seperti terlihat pada gambar berikut:



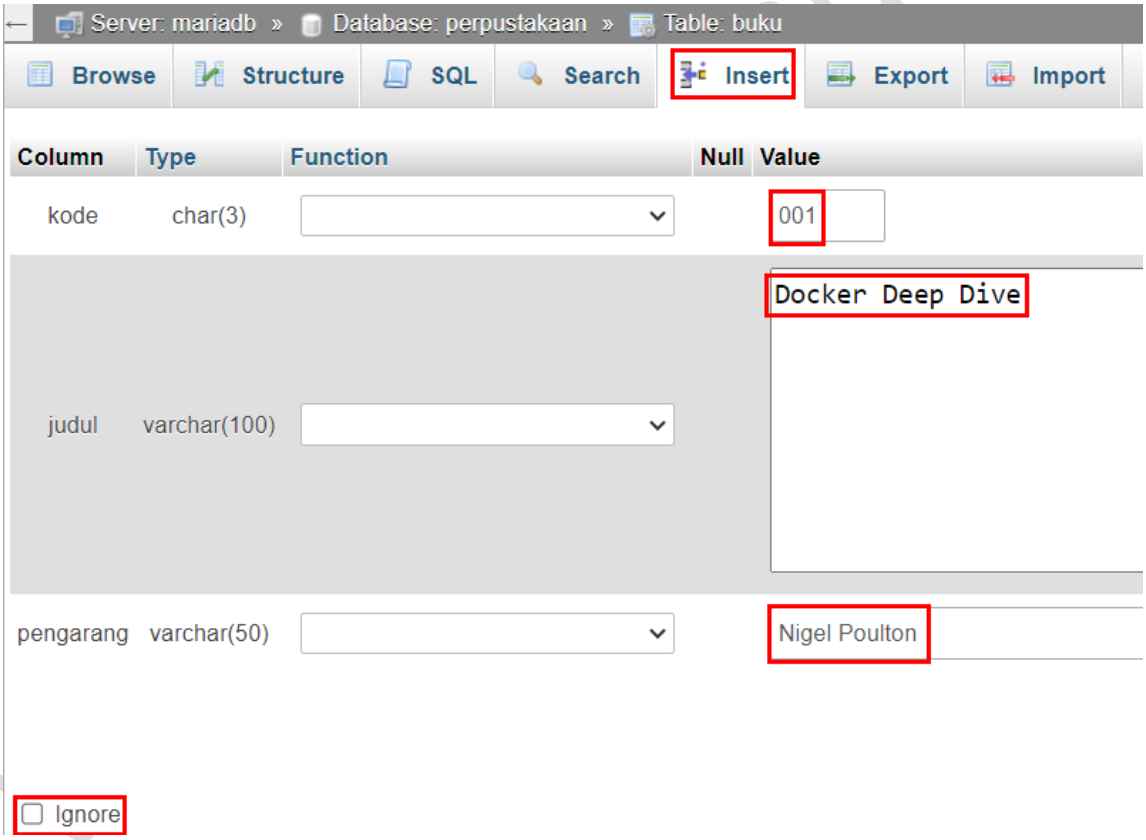
Tekan tombol **Go** untuk memproses pembuatan tabel menggunakan DDL tersebut sehingga hasilnya, seperti terlihat pada gambar berikut:



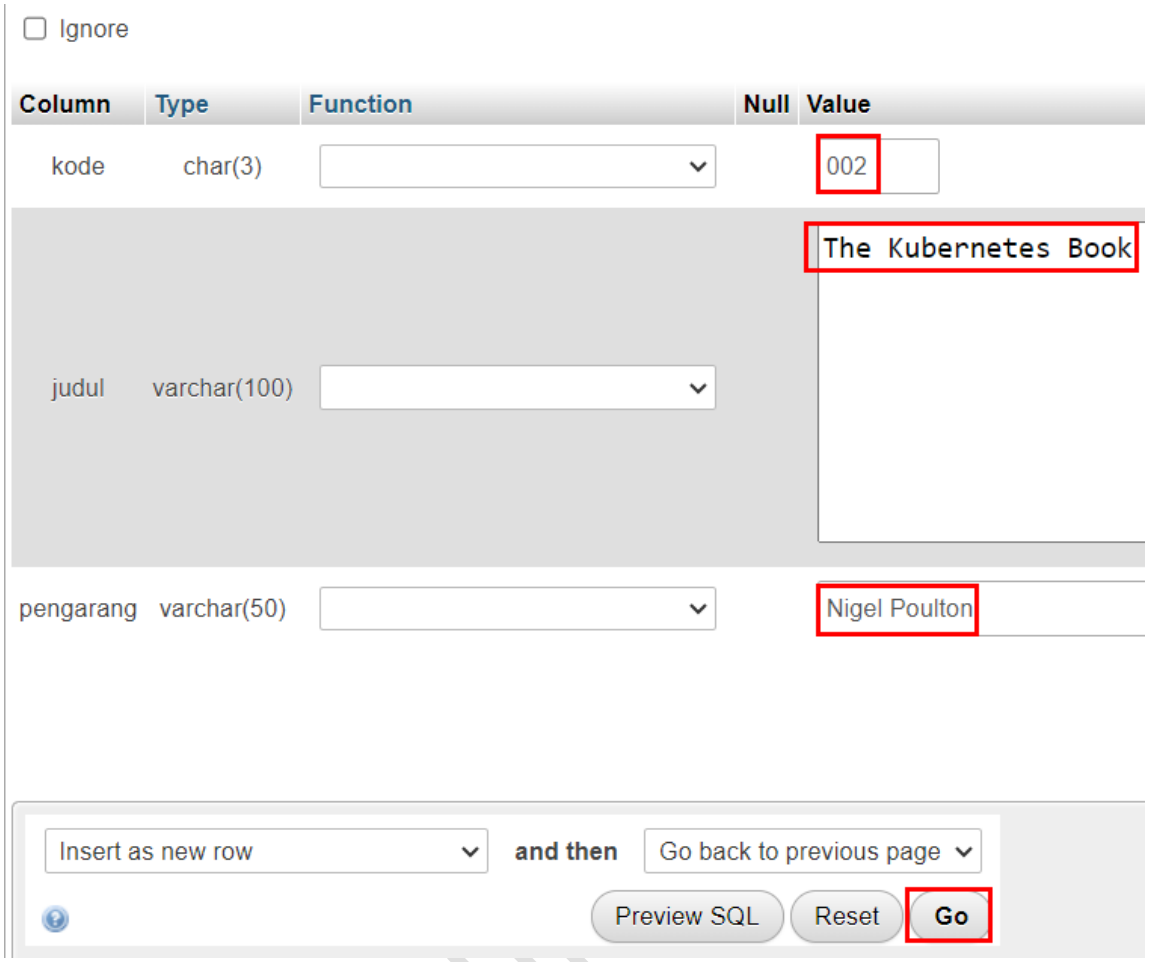
Selanjutnya akan dilakukan ujicoba menambahkan data buku ke tabel tersebut dengan memilih **buku** pada panel sebelah kiri dan pada panel sebelah kanan pilih **Insert**, seperti terlihat pada gambar berikut:



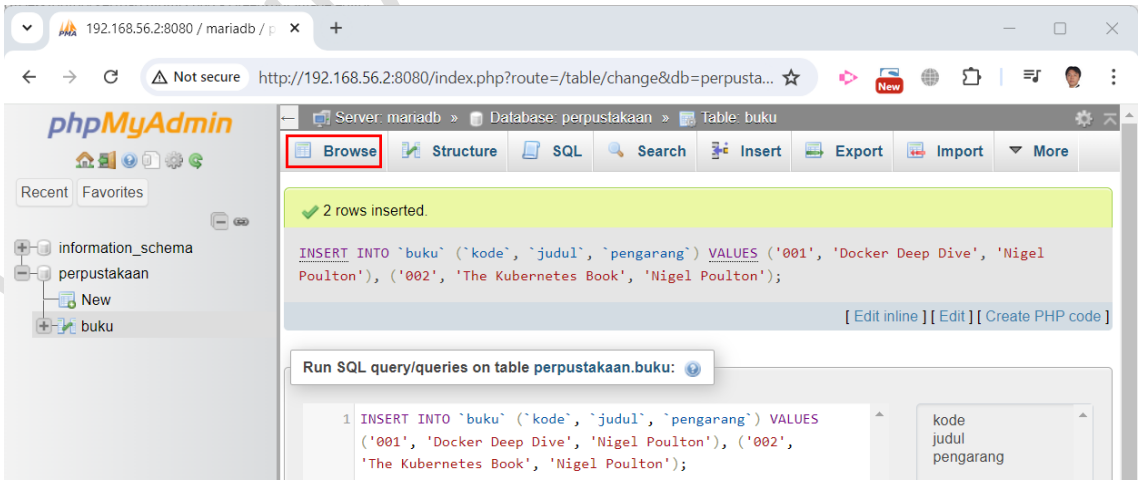
Lengkapi isian **Value** dari *form* pertama untuk kolom **kode** dengan nilai "**001**" dan kolom **judul** dengan nilai "**Docker Deep Dive**" serta kolom **pengarang** dengan nilai "**Nigel Poulton**". Hilangkan centang (√) pada *checkbox* **Ignore** yang terdapat pada bagian paling bawah *form* pertama. Hal tersebut dilakukan agar dapat melakukan penambahan data buku kedua di *form* berikutnya, seperti terlihat pada gambar berikut:



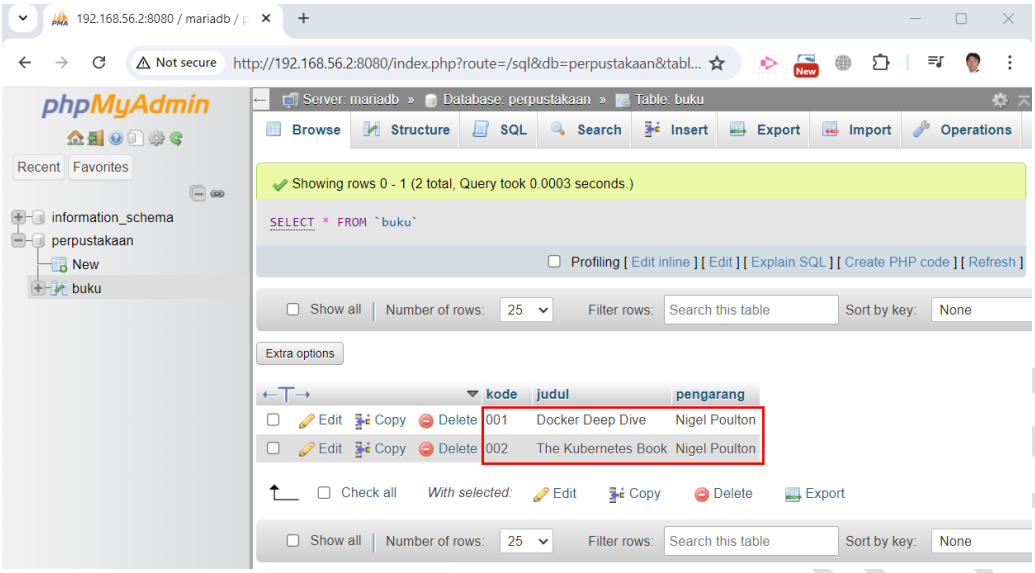
Selanjutnya pada form kedua, lengkapi isian **Value** dari form kedua untuk kolom **kode** dengan nilai "**002**" dan kolom **judul** dengan nilai "**The Kubernetes Book**" serta kolom **pengarang** dengan nilai "**Nigel Poulton**". Tekan tombol **Go** yang terdapat pada bagian paling bawah untuk memproses penambahan kedua data buku tersebut, seperti terlihat pada gambar berikut:



Tampil halaman dengan pesan yang menginformasikan **2 rows inserted** dan perintah SQL yang digunakan untuk menambahkan data ke tabel **buku**. Pilih **Browse** untuk menampilkan data yang telah berhasil ditambahkan ke tabel buku, seperti terlihat pada gambar berikut:



Terlihat terdapat dua data buku yang telah dimasukkan sebelumnya, seperti terlihat pada gambar berikut:



Tutup *browser* yang digunakan mengakses **phpMyAdmin**.

10. Menghentikan *containers*.

```
$ docker compose down
```

```
belajar@sabarmenanti:~/compose/myapp$ docker compose down
[+] Running 3/3
 ✔ Container myapp-phpmyadmin-1  Removed
 ✔ Container myapp-mariadb-1     Removed
 ✔ Network myapp_default         Removed
```

11. Memverifikasi hasil penghentian *containers*.

```
$ docker compose ps -a
```

```
belajar@sabarmenanti:~/compose/myapp$ docker compose ps -a
NAME      IMAGE     COMMAND   SERVICE   CREATED   STATUS    PORTS
belajar@sabarmenanti:~/compose/myapp$
```

Terlihat keseluruhan *container* tersebut telah berhasil dihentikan dan dihapus.

12. Membuat dan menjalankan kembali *containers services* di *background*.

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

```
belajar@sabarmenanti:~/compose/myapp$ docker compose up -d
[+] Running 3/3
 ✔ Network myapp_default         Created
 ✔ Container myapp-mariadb-1     Started
 ✔ Container myapp-phpmyadmin-1  Started
```

13. Memverifikasi hasil *containers* telah berjalan.

```
$ docker compose ps
```

```
belajar@sabarmenanti:~/compose/myapp$ docker compose ps
NAME              IMAGE                  COMMAND                  SERVICE     CREATED          STATUS          PORTS
myapp-mariadb-1   mariadb:latest         "docker-entrypoint.s…"   mariadb     9 seconds ago    Up 7 seconds    0.0.0.0:3306->3306/tcp, :::3306->3306/tcp
myapp-phpmyadmin-1   phpmyadmin/phpmyadmin   "/docker-entrypoint.…"   phpmyadmin  8 seconds ago    Up 6 seconds    0.0.0.0:8080->80/tcp, :::8080->80/tcp
```

14. Memverifikasi kembali koneksi ke *docker container* **mariadb** dari *host machine* menggunakan utilitas *mysql client* dengan parameter-parameter berikut:
    - `-h` untuk mengatur hostname dari server mariadb yang akan diakses yaitu **localhost**.
    - `-u` untuk mengatur nama login pengguna yang digunakan untuk mengakses ke *server mariadb* yaitu **root**.
    - `-p` digunakan untuk mengatur inputan sandi untuk nama login pengguna.
    - `-P` digunakan untuk mengatur nomor *port* dari *server mariadb* yaitu **3306**.

    Sehingga perintahnya terlihat seperti berikut:

    ```
    $ mysql -h localhost -u root -p -P 3306
    ```

    Tampil inputan **Enter password**, masukkan `sabarmenanti` & tekan **Enter**.

```
belajar@sabarmenanti:~/compose/myapp$ mysql -h localhost -u root -p -P 3306
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 24
Server version: 11.4.2-MariaDB-ubu2404 mariadb.org binary distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]>
```

    Terlihat koneksi ke *server mariadb* berhasil dilakukan ditandai dengan penampilan *prompt* `MariaDB [(none)]>`

    a. Menampilkan informasi *database* yang telah ada di *server mariadb*.

    ```
    show databases;
    ```

```
MariaDB [(none)]> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| perpustakaan       |
| sys                |
+--------------------+
5 rows in set (0,001 sec)
```

    Terlihat terdapat 5 (lima) *database* yaitu *information_schema*, *mysql*, *performance_schema* dan **perpustakaan** serta *sys*. *Database* **perpustakaan** masih terlihat sebagai luaran sehingga hal tersebut menguatkan bahwa penyimpanan data

permanen melalui pemanfaatan *bind mount* dengan atribut **volumes** di **service mariadb**.

b. Mengatur agar *database* **perpustakaan** sebagai *database* aktif dan melihat tabel-tabel yang terdapat pada *database* aktif tersebut.

```
use perpustakaan;
show tables;
```

```
MariaDB [(none)]> use perpustakaan;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
MariaDB [perpustakaan]> show tables;
+------------------------+
| Tables_in_perpustakaan |
+------------------------+
| buku                   |
+------------------------+
1 row in set (0,001 sec)
```

c. Menampilkan isi dari tabel **buku**.

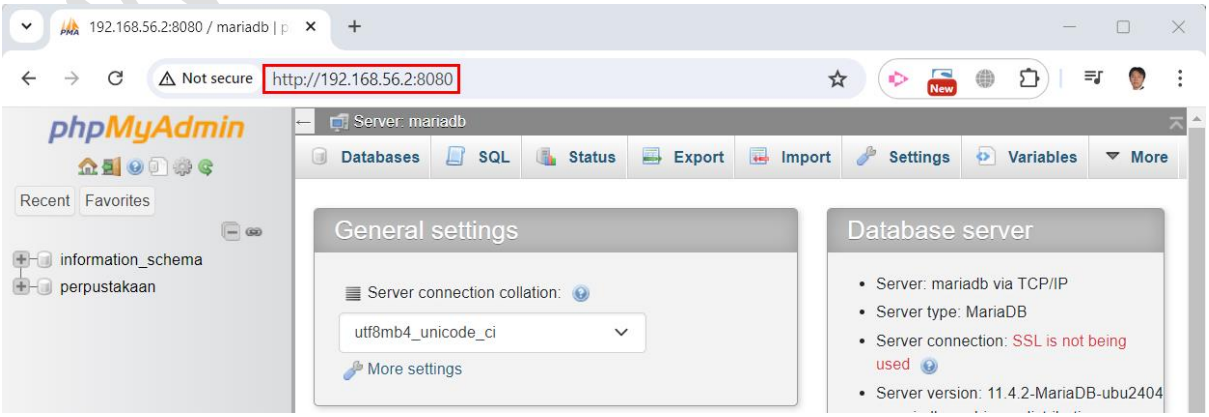
```
select * from buku;
```

```
MariaDB [perpustakaan]> select * from buku;
+------+---------------------+---------------+
| kode | judul               | pengarang     |
+------+---------------------+---------------+
| 001  | Docker Deep Dive    | Nigel Poulton |
| 002  | The Kubernetes Book | Nigel Poulton |
+------+---------------------+---------------+
2 rows in set (0,001 sec)
```

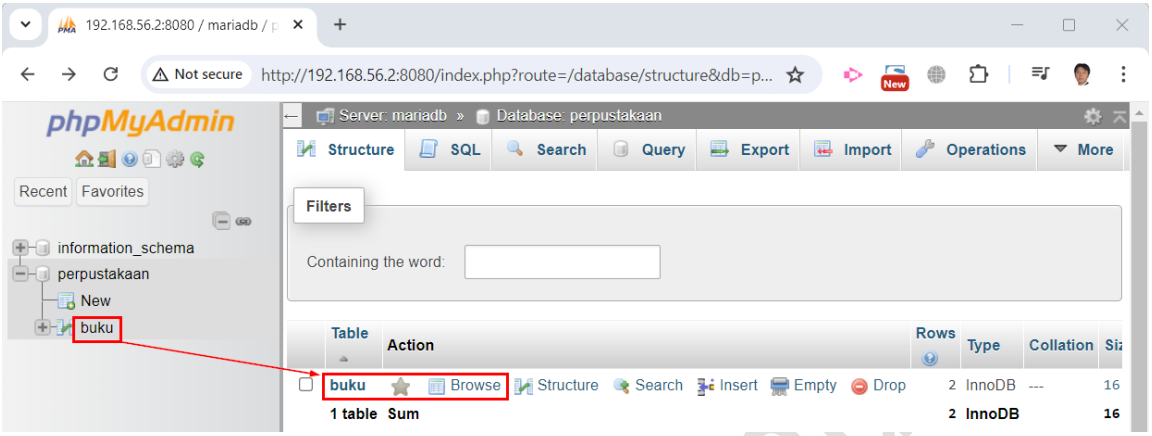
d. Keluar dari *server mariadb*.

```
quit;
```

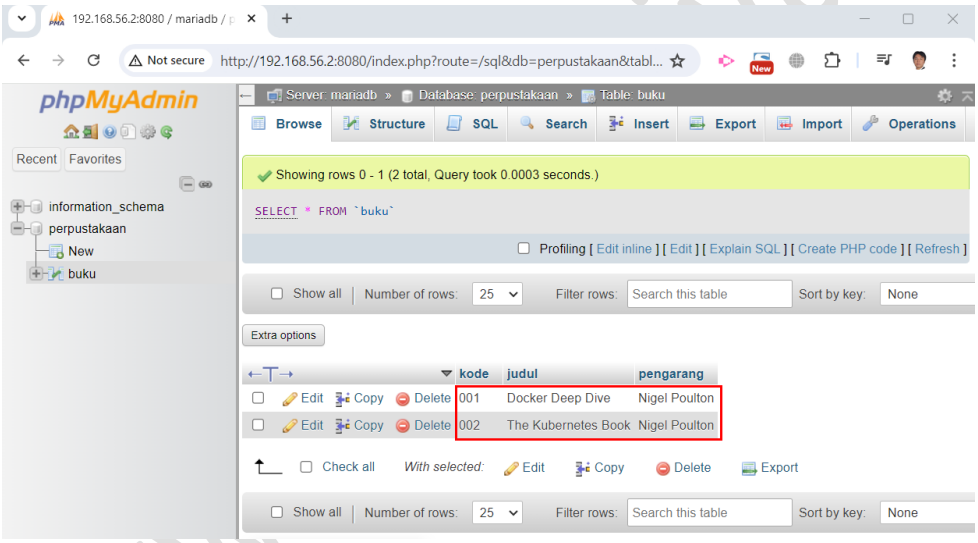
15. Mengakses kembali layanan **phpMyAdmin** melalui *browser* di *Windows 11* pada alamat http://192.168.56.2:8080/, seperti terlihat pada gambar berikut:



16. Mengakses database perpustakaan dengan memilih **perpustakaan** pada panel sebelah kiri dari **dashboard phpMyAdmin** yang tampil. Selanjutnya pada panel sebelah kiri pilih **buku** yang terdapat di dalam *database* **perpustakaan** dan pada panel detail sebelah kanan pilih tautan **Browse** dari tabel **buku**, seperti terlihat pada gambar berikut:



Terlihat terdapat dua data buku yang telah dimasukkan sebelumnya, seperti gambar berikut:



Tutup *browser* yang digunakan mengakses **phpMyAdmin**.

17. Menghentikan *containers*.

```
$ docker compose down
```

```
belajar@sabarmenanti:~/compose/myapp$ docker compose down
[+] Running 3/3
 ✔ Container myapp-phpmyadmin-1  Removed
 ✔ Container myapp-mariadb-1     Removed
 ✔ Network myapp_default         Removed
```

18. Memverifikasi hasil penghentian *containers*.

```
$ docker compose ps -a
```

```
belajar@sabarmenanti:~/compose/myapp$ docker compose ps -a
NAME      IMAGE     COMMAND   SERVICE   CREATED   STATUS    PORTS
belajar@sabarmenanti:~/compose/myapp$
```

Terlihat keseluruhan *container* tersebut telah berhasil dihentikan dan dihapus.

BAB VII

KUBERNETES FUNDAMENTAL

Menurut dokumentasi dari Kubernetes, **Kubernetes** merupakan platform sumber terbuka (*open source*) yang portabel dan dapat diperluas untuk mengelola beban kerja dan layanan dalam container serta memfasilitasi konfigurasi deklaratif dan otomatisasi. Fitur-fitur yang ditawarkan oleh *Kubernetes* meliputi:

1. *Kubernetes* dapat mengekspos sebuah *container* menggunakan nama *Domain Name System (DNS)* atau menggunakan alamat IP, dikenal dengan istilah *Service Discovery*.
2. *Kubernetes* dapat menyeimbangkan beban dan mendistribusikan lalu lintas jaringan sehingga penerapannya stabil, dikenal dengan istilah *Load Balancing*.
3. Kubernetes memungkinkan memasangkan (mount) sistem penyimpanan yang dipilih secara otomatis seperti lokal, *public cloud provider* dan lainnya yang dikenal dengan istilah *Storage orchestration*.
4. *Automated rollouts dan rollbacks* yang memungkinkan pengguna mendeskripsikan status yang diinginkan (*desired state*) untuk container yang diterapkan menggunakan *Kubernetes*. *Kubernetes* dapat mengubah status sebenarnya (*actual state*) ke status yang diinginkan (*desired state*) dengan kecepatan yang terkendali.
5. Automatic bin packing yang memungkinkan *Kubernetes* memberikan sekelompok node yang dapat digunakan untuk menjalankan tugas-tugas dalam *container*. Pengguna dapat menginformasikan ke Kubernetes berapa banyak CPU dan memori (RAM) yang dibutuhkan oleh setiap *container*. *Kubernetes* dapat memasukkan *container* ke dalam *node* untuk memanfaatkan sumber daya sebaik-baiknya.
6. *Self-healing* yang memungkinkan *Kubernetes* melakukan penyembuhan mandiri meliputi memulai ulang *container* yang gagal, mengganti *container*, mematikan *container* yang tidak merespon *health check* yang ditentukan pengguna dan tidak melakukan *advertise* ke client hingga siap melayani.
7. *Kubernetes* menyimpan dan mengelola informasi sensitif seperti *password, OAuth token* dan *SSH keys*. Pengguna Kubernetes dapat mendeploy dan memperbaharui *secret* tanpa membangun ulang *container images* dan tanpa mengekspos *secret* di *stack configuration*.

8. Kubernetes dapat mengelola beban kerja *batch* dan *Continuous Integration (CI)*, mengganti container yang gagal jika diinginkan.
9. Penskalaan aplikasi secara horizontal dengan perintah sederhana, dengan *User Interface (UI)* atau secara otomatis berdasarkan penggunakan *Central Processing Unit (CPU)*.
10. Alokasi alamat IPv4/IPv6 *dual stack* ke *Pods* dan *Services*.
11. Dirancang dengan kemampuan yang dapat diperluas sehingga penambahan fitur ke *cluster Kubernetes* tanpa mengubah kode sumber *upstream*.

A. Instalasi dan Konfigurasi Minikube

Minikube merupakan *tool* yang digunakan untuk menjalankan **Kubernetes cluster** secara lokal sehingga membantu pengembang aplikasi dan pengguna *Kubernetes* baru. *Minikube* dapat diinstalasi pada sistem operasi *MacOS*, *Windows* dan *Linux*. Tujuan utama pemanfaatan *Minikube* adalah untuk pengembangan dan pengujian secara lokal

Adapun langkah-langkah menginstalasi dan mengkonfigurasi *Minikube* di *Ubuntu Desktop* adalah sebagai berikut:

1. Menginstalasi *dependencies minikube*.

   ```
   $ sudo apt install -y curl wget apt-transport-https
   ```

2. Mengunduh *binary* terkini dari *minikube*.

   $ curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64

```
belajar@sabarmenanti:~$ curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
 95 91.1M   95 86.9M    0     0   441k      0  0:03:31  0:03:21  0:00:10  651
 96 91.1M   96 87.6M    0     0   443k      0  0:03:30  0:03:22  0:00:08  750
 96 91.1M   96 88.3M    0     0   444k      0  0:03:30  0:03:23  0:00:07  830
 97 91.1M   97 89.0M    0     0   445k      0  0:03:29  0:03:24  0:00:05  805
 98 91.1M   98 89.5M    0     0   446k      0  0:03:29  0:03:25  0:00:04  724
 98 91.1M   98 90.1M    0     0   446k      0  0:03:29  0:03:26  0:00:03  652
 99 91.1M   99 90.7M    0     0   447k      0  0:03:28  0:03:27  0:00:01  630
100 91.1M  100 91.1M    0     0   447k      0  0:03:28  0:03:28 --:--:--  595k
```

3. Menginstalasi *minikube* menggunakan *file binary* yang telah diunduh sebelumnya.

   ```
   $ sudo install minikube-linux-amd64 /usr/local/bin/minikube
   ```

```
belajar@sabarmenanti:~$ sudo install minikube-linux-amd64 /usr/local/bin/minikube
[sudo] password for belajar:
```

4. Memverifikasi versi *minikube* yang telah terinstalasi.

   ```
   $ minikube version
   ```

```
belajar@sabarmenanti:~$ minikube version
minikube version: v1.33.1
commit: 5883c09216182566a63dff4c326a6fc9ed2982ff
```

Terlihat versi Minikube yang terinstalasi adalah **1.33.1**.

5. Menginstalasi utilitas **kubectl** yang merupakan *tool Command Line Interface (CLI)* untuk bekerja dengan *Kubernetes cluster*.

$ curl -LO https://storage.googleapis.com/kubernetes-release/release/`curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt`/bin/linux/amd64/kubectl

```
belajar@sabarmenanti:~$ curl -LO https://storage.googleapis.com/kubernetes-release/release/`curl -s https://storage.googleapis.com/kubernetes-release/release/
stable.txt`/bin/linux/amd64/kubectl
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  4 49.0M    4 2080k    0     0   415k      0  0:02:00  0:00:05  0
  4 49.0M    4 2416k    0     0   401k      0  0:02:05  0:00:06  0
  5 49.0M    5 2688k    0     0   375k      0  0:02:13  0:00:07  0
100 49.0M  100 49.0M    0     0   304k      0  0:02:44  0:02:44 --:--:--  301k
```

6. Mengatur *permission executable* untuk *binary* dari *kubectl* dan memindahkannya ke direktori **/usr/local/bin/**.

```
$ chmod +x kubectl
$ sudo mv kubectl /usr/local/bin/
```

```
belajar@sabarmenanti:~$ chmod +x kubectl
belajar@sabarmenanti:~$ sudo mv kubectl /usr/local/bin/
```

7. Memverifikasi versi dari *kubectl*.

```
$ kubectl version
```

```
belajar@sabarmenanti:~$ kubectl version
Client Version: v1.30.2
Kustomize Version: v5.0.4-0.20230601165947-6ce0bf390ce3
The connection to the server localhost:8080 was refused - did you specify the ri
ght host or port?
```

Atau dengan mengeksekusi perintah:

```
$ kubectl version --client
```

```
belajar@sabarmenanti:~$ kubectl version --client
Client Version: v1.30.2
Kustomize Version: v5.0.4-0.20230601165947-6ce0bf390ce3
```

Terlihat versi yang terinstalasi adalah **1.30.2**.

8. Menjalankan *minikube* dengan **docker driver**.

```
$ minikube start --driver=docker
```

```
belajar@sabarmenanti:~$ minikube start --driver=docker
* minikube v1.33.1 on Ubuntu 22.04
* Using the docker driver based on user configuration
* Using Docker driver with root privileges
* Starting "minikube" primary control-plane node in "minikube" cluster
* Pulling base image v0.0.44 ...
* Downloading Kubernetes v1.30.0 preload ...
    > preloaded-images-k8s-v18-v1...:  342.90 MiB / 342.90 MiB  100.00% 219.35
    > gcr.io/k8s-minikube/kicbase...:  481.58 MiB / 481.58 MiB  100.00% 220.51
* Creating docker container (CPUs=2, Memory=2200MB) ...
* Preparing Kubernetes v1.30.0 on Docker 26.1.1 ...
  - Generating certificates and keys ...
  - Booting up control plane ...
  - Configuring RBAC rules ...
* Configuring bridge CNI (Container Networking Interface) ...
* Verifying Kubernetes components...
  - Using image gcr.io/k8s-minikube/storage-provisioner:v5
* Enabled addons: storage-provisioner, default-storageclass
* Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
```

9. Mengecek status dari *minikube*.

   ```
   $ minikube status
   ```

   ```
   belajar@sabarmenanti:~$ minikube status
   minikube
   type: Control Plane
   host: Running
   kubelet: Running
   apiserver: Running
   kubeconfig: Configured
   ```

   Terlihat informasi *profil cluster* yang digunakan adalah **minikube** dengan jenis **Control Plane**. Selain itu juga terlihat status dari *host*, *kubelet* dan *apiserver* yaitu **running**.

10. Memverifikasi *Kubernetes version*, *node status* dan *cluster info*.

    ```
    $ kubectl cluster-info
    ```

    ```
    belajar@sabarmenanti:~$ kubectl cluster-info
    Kubernetes control plane is running at https://192.168.49.2:8443
    CoreDNS is running at https://192.168.49.2:8443/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy

    To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
    ```

    Terlihat informasi *Kubernetes control plane* dan *CoreDNS* telah berjalan (*running*).

    ```
    $ kubectl get nodes
    ```

    ```
    belajar@sabarmenanti:~$ kubectl get nodes
    NAME       STATUS   ROLES           AGE     VERSION
    minikube   Ready    control-plane   4m36s   v1.30.0
    ```

    Terlihat terdapat satu *node* bernama "**minikube**" dengan status **Ready** dan memiliki **Roles** "**control-plane**".

11. Menghentikan *minikube*.

    ```
    $ minikube stop
    ```

```
belajar@sabarmenanti:~$ minikube stop
* Stopping node "minikube"  ...
* Powering off "minikube" via SSH ...
* 1 node stopped.
```

12. Mengecek status dari *minikube*.

    ```
    $ minikube status
    ```

    ```
    belajar@sabarmenanti:~$ minikube status
    minikube
    type: Control Plane
    host: Stopped
    kubelet: Stopped
    apiserver: Stopped
    kubeconfig: Stopped
    ```

    Terlihat *minikube* telah berhasil dihentikan ditandai dengan status *Stopped* pada *kubelet*, *apiserver* dan *kubeconfig*.

B. Mendeploy Aplikasi/Services di Kubernetes

Sebagai contoh akan dilakukan *deployment* **nginx** versi terkini pada *Kubernetes*. Adapun langkah-langkah melakukan *deployment* aplikasi atau *service* tersebut ke *Kubernetes* adalah sebagai berikut:

1. Menjalankan *minikube*.

   ```
   $ minikube start
   ```

```
belajar@sabarmenanti:~$ minikube start
* minikube v1.33.1 on Ubuntu 22.04
* Using the docker driver based on existing profile
* Starting "minikube" primary control-plane node in "minikube" cluster
* Pulling base image v0.0.44 ...
* Restarting existing docker container for "minikube" ...
* Preparing Kubernetes v1.30.0 on Docker 26.1.1 ...
* Verifying Kubernetes components...
  - Using image gcr.io/k8s-minikube/storage-provisioner:v5
* Enabled addons: storage-provisioner, default-storageclass
* Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
```

2. Mengecek status *minikube*.

   ```
   $ minikube status
   ```

   ```
   belajar@sabarmenanti:~$ minikube status
   minikube
   type: Control Plane
   host: Running
   kubelet: Running
   apiserver: Running
   kubeconfig: Configured
   ```

3. Membuat *deployment* **nginx** versi terkini. Sintak penulisan perintah *deployment* adalah:

```
kubectl create deployment NAME --image=image -- [COMMAND] [args...]
```

Argumen `NAME` merupakan nama *deployment*. Sedangkan *option* `--image` digunakan untuk menentukan nama *image* yang akan dijalankan. Perintah `kubectl create` tersebut akan membuat sebuah *deployment* yang mengelola sebuah **Pod**. **Pod** menjalankan *container* berdasarkan pada *docker image* yang diberikan. Sebagai contoh nama *deployment* yang akan digunakan adalah **nginx-deploy** dengan nama *image* **nginx** sehingga perintah yang dieksekusi:

```
$ kubectl create deployment nginx-deploy --image=nginx
```

```
belajar@sabarmenanti:~$ kubectl create deployment nginx-deploy --image nginx
deployment.apps/nginx-deploy created
```

4. Memverifikasi hasil *deployment*.

```
$ kubectl get deployment
```

```
belajar@sabarmenanti:~$ kubectl get deployment
NAME           READY   UP-TO-DATE   AVAILABLE   AGE
nginx-deploy   1/1     1            1           51s
```

Terlihat *deployment* berhasil dilakukan. Apabila nilai pada kolom **AVAILABLE** belum bernilai 1 (masih 0) maka *deployment* masih berlangsung sehingga perlu menunggu beberapa saat. Jika ingin melihat perubahan nilai pada kolom tersebut secara *real time* maka eksekusi perintah:

```
$ kubectl get deployment -w
```

```
belajar@sabarmenanti:~$ kubectl get deployment -w
NAME           READY   UP-TO-DATE   AVAILABLE   AGE
nginx-deploy   0/1     1            0           9s
nginx-deploy   1/1     1            1           34s
```

Tekan **CTRL+C** untuk keluar dari pemantauan secara *real time* tersebut jika nilai pada kolom **AVAILABLE** telah bernilai 1.

5. Menampilkan detail informasi atau deskripsi terkait *deployment* **nginx-deploy** yang telah dilakukan. Sintak penulisan perintah untuk menampilkan detail dari sumber daya atau kelompok sumber daya adalah:

```
kubectl describe TYPE NAME_PREFIX
```

Argumen `TYPE` menentukan jenis sumber daya yang akan ditampilkan informasi atau deskripsi detailnya. Sedangkan argumen `NAME_PREFIX` merupakan nama sumber daya yang akan ditampilkan informasi atau deskripsi detailnya. Sehingga perintah yang

dieksekusi untuk menampilkan jenis sumber daya **deployment** dengan nama **nginx-deploy** adalah:

```
$ kubectl describe deployment nginx-deploy
```

```
belajar@sabarmenanti:~$ kubectl describe deployment nginx-deploy
Name:                   nginx-deploy
Namespace:              default
CreationTimestamp:      Wed, 10 Jul 2024 03:25:40 +0800
Labels:                 app=nginx-deploy
Annotations:            deployment.kubernetes.io/revision: 1
Selector:               app=nginx-deploy
Replicas:               1 desired | 1 updated | 1 total | 1 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  app=nginx-deploy
  Containers:
   nginx:
    Image:         nginx
    Port:          <none>
    Host Port:     <none>
    Environment:   <none>
    Mounts:        <none>
  Volumes:         <none>
  Node-Selectors:  <none>
  Tolerations:     <none>
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Available      True    MinimumReplicasAvailable
  Progressing    True    NewReplicaSetAvailable
OldReplicaSets:  <none>
NewReplicaSet:   nginx-deploy-854b9d5bdb (1/1 replicas created)
Events:
  Type    Reason             Age    From                   Message
  ----    ------             ----   ----                   -------
  Normal  ScalingReplicaSet  3m30s  deployment-controller  Scaled up replica set nginx-deploy-854b9d5bdb to 1
```

Terlihat *image* yang digunakan untuk *deployment* tersebut adalah *nginx*.

6. Melihat informasi *pods*.

```
$ kubectl get pods
```

```
belajar@sabarmenanti:~$ kubectl get pods
NAME                            READY   STATUS    RESTARTS   AGE
nginx-deploy-854b9d5bdb-gqvmc   1/1     Running   0          63m
```

Terlihat terdapat satu *pod*.

7. Menampilkan informasi *container* yang berjalan pada **pod** dan *images* yang digunakan untuk membangun *container* didalamnya.

```
$ kubectl describe pods
```

```
belajar@sabarmenanti:~$ kubectl describe pods
Name:             nginx-deploy-854b9d5bdb-bhwbs
Namespace:        default
Priority:         0
Service Account:  default
Node:             minikube/192.168.49.2
Start Time:       Wed, 10 Jul 2024 14:05:51 +0800
Labels:           app=nginx-deploy
                  pod-template-hash=854b9d5bdb
Annotations:      <none>
Status:           Running
IP:               10.244.0.18
IPs:
  IP:           10.244.0.18
Controlled By:  ReplicaSet/nginx-deploy-854b9d5bdb
Containers:
  nginx:
    Container ID:   docker://28dfd24318213b9c8d1394fda770e2bef0319ed30e114e5e519284efa354c2c4
    Image:          nginx
    Image ID:       docker-pullable://nginx@sha256:67682bda769fae1ccf5183192b8daf37b64cae99c6c3302650f6f8bf5f0f95df
```

```
    Port:           <none>
    Host Port:      <none>
    State:          Running
      Started:      Wed, 10 Jul 2024 14:06:25 +0800
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-zw7cg (ro)
Conditions:
  Type                        Status
  PodReadyToStartContainers   True
  Initialized                 True
  Ready                       True
  ContainersReady             True
  PodScheduled                True
Volumes:
  kube-api-access-zw7cg:
    Type:                    Projected (a volume that contains injected data from multiple sources)
    TokenExpirationSeconds:  3607
    ConfigMapName:           kube-root-ca.crt
    ConfigMapOptional:       <nil>
    DownwardAPI:             true
QoS Class:                   BestEffort
Node-Selectors:              <none>
Tolerations:                 node.kubernetes.io/not-ready:NoExecute op=Exists for 300s
                             node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
Events:
  Type     Reason     Age                From               Message
  ----     ------     ----               ----               -------
  Normal   Scheduled  11m                default-scheduler  Successfully assigned default/nginx-deploy-854b9d5bdb-bhwbs to minikube
  Warning  Failed     11m                kubelet            Failed to pull image "nginx": Error response from daemon: Head "https://registry-1.docker.io/v2/li
brary/nginx/manifests/latest": net/http: TLS handshake timeout
  Warning  Failed     11m                kubelet            Error: ErrImagePull
  Normal   BackOff    11m                kubelet            Back-off pulling image "nginx"
  Warning  Failed     11m                kubelet            Error: ImagePullBackOff
  Normal   Pulling    11m (x2 over 11m)  kubelet            Pulling image "nginx"
  Normal   Pulled     11m                kubelet            Successfully pulled image "nginx" in 3.539s (3.539s including waiting). Image size: 187599276 byte
s.
  Normal   Created    11m                kubelet            Created container nginx
  Normal   Started    11m                kubelet            Started container nginx
```

8. Menampilkan log aplikasi untuk *pod*. Ganti nama *pod* dengan yang dimiliki sesuai dengan luaran dari eksekusi perintah `kubectl get pods`.

   ```
   $ kubectl logs nginx-deploy-854b9d5bdb-gqvmc
   ```

```
belajar@sabarmenanti:~$ kubectl logs nginx-deploy-854b9d5bdb-gqvmc
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2024/07/09 19:26:23 [notice] 1#1: using the "epoll" event method
2024/07/09 19:26:23 [notice] 1#1: nginx/1.27.0
2024/07/09 19:26:23 [notice] 1#1: built by gcc 12.2.0 (Debian 12.2.0-14)
2024/07/09 19:26:23 [notice] 1#1: OS: Linux 6.5.0-41-generic
2024/07/09 19:26:23 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2024/07/09 19:26:23 [notice] 1#1: start worker processes
2024/07/09 19:26:23 [notice] 1#1: start worker process 29
2024/07/09 19:26:23 [notice] 1#1: start worker process 30
10.244.0.1 - - [09/Jul/2024:19:39:17 +0000] "GET / HTTP/1.1" 200 615 "-" "curl/7.81.0" "-"
```

9. Membuat *service* agar *pod* **nginx-deploy** dapat diakses dari luar *Kubernetes virtual network*. Secara *default pod* hanya dapat di akses menggunakan alamat IP di dalam *Kubernetes cluster*. Sintak perintah untuk membuat *service* adalah:

   ```
   kubectl expose TYPE NAME [--type=type] [--port=port]
   ```

   Argumen `TYPE` menentukan jenis sumber daya yang akan dibuat *service*nya. Sedangkan argumen `NAME` merupakan nama sumber daya akan dibuat *service*nya. *Option* `--type` digunakan untuk menentukan jenis *service* yang dapat berupa **ClusterIP**, **NodePort** dan **LoadBalancer**.

   ClusterIP mengekspos service pada IP internal *cluster*. Memilih nilai ini membuat service hanya dapat dijangkau dari dalam *cluster*. *NodePort* mengekspos *service* pada setiap IP

Node di *port* statis (*NodePort*). Untuk membuat *port node* tersedia, Kubernetes menyiapkan alamat IP *cluster*, sama seperti ketika meminta *service* bertipe *ClusterIP*. *LoadBalancer* mengekspos *service* secara eksternal menggunakan penyeimbang beban eksternal. Kubernetes tidak secara langsung menawarkan komponen penyeimbang beban. Pengguna harus menyediakannya atau dapat mengintegrasikan *cluster Kubernetes* dengan penyedia *cloud*.

Sedangkan *option* `--port` digunakan untuk mengatur nomor *port* yang dibuka untuk memberikan pelayanan. Sehingga perintah yang dieksekusi untuk membuat *service* dengan jenis sumber daya **deployment** bernama **nginx-deploy** dan menggunakan **type NodePort** serta **port 80** adalah:

```
$ kubectl expose deployment nginx-deploy --type="NodePort" --port 80
```

```
belajar@sabarmenanti:~$ kubectl expose deployment nginx-deploy --type="NodePort" --port 80
service/nginx-deploy exposed
```

10. Memverifikasi hasil pembuatan *service*.

    ```
    $ kubectl get services
    ```

```
belajar@sabarmenanti:~$ kubectl get services
NAME           TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
kubernetes     ClusterIP   10.96.0.1       <none>        443/TCP        11h
nginx-deploy   NodePort    10.101.92.223   <none>        80:30767/TCP   64s
```

    Terlihat *service* **nginx-deploy** telah berhasil dibuat dan menggunakan **port 30767**.

11. Menampilkan informasi alamat IP dari *minikube* yang diperlukan ketika ujicoba pengaksesan layanan pada *pod* **nginx-deploy**.

    ```
    $ minikube ip
    ```

```
belajar@sabarmenanti:~$ minikube ip
192.168.49.2
```

    Terlihat alamat IP yang digunakan oleh *minikube* adalah **192.168.49.2**.

12. Mengakses layanan pada *pod* **nginx-deploy** menggunakan utilitas **curl** pada **port 30767**. Ganti nomor *port* dengan yang dimiliki sesuai dengan luaran dari eksekusi perintah `kubectl get services`. Alamat IP yang digunakan adalah IP dari *minikube* yang telah diidentifikasi pada langkah sebelumnya yaitu **192.168.49.2**.

    ```
    $ curl 192.168.49.2:30767
    ```

```
belajar@sabarmenanti:~$ curl 192.168.49.2:30767
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

Terlihat *homepage* dari *pod* **nginx-deploy** berhasil diakses.

13. Mengambil nama *pod* dan menyimpannya pada *environment variable* bernama `POD_NAME`.

```
export POD_NAME=$(kubectl get pods -o go-template --template '{{range
.items}}{{.metadata.name}}{{"\n"}}{{end}}')
```

```
export POD_NAME="$(kubectl get pods -o go-template --template '{{range .items}}{{.metadata.name}}{{"\n"}}{{end}}')"
```

14. Memverifikasi nilai dari *environment variable* yang telah dideklarasikan sebelumnya.

```
$ echo Name of the Pod: $POD_NAME
```

```
belajar@sabarmenanti:~$ echo Name of the Pod: $POD_NAME
Name of the Pod: nginx-deploy-854b9d5bdb-gqvmc
```

15. Mengubah halaman *homepage* dari *pod* **nginx-deploy** dengan menjalankan sesi **bash**.

```
$ kubectl exec -it $POD_NAME -- bash
```

a. Berpindah ke direktori **/usr/share/nginx/html**.

```
# cd /usr/share/nginx/html
```

b. Mengubah isi dari *file* **index.html**.

```
# echo \
“Pelatihan Cloud Native” > index.html
```

```
root@nginx-deploy-854b9d5bdb-gqvmc:/usr/share/nginx/html# echo \
> "Pelatihan Cloud Native" > index.html
```

c. Keluar dari sesi *bash pod*.

```
# exit
```

16. Memverifikasi perubahan konten *homepage* dari *pod* **nginx-deploy** menggunakan utilitas **curl** pada **port 30767**. Ganti nomor *port* dengan yang dimiliki sesuai dengan luaran dari eksekusi perintah `kubectl get services`. Alamat IP yang digunakan adalah IP dari *minikube* yang telah diidentifikasi pada langkah sebelumnya yaitu **192.168.49.2**.

    ```
    $ curl 192.168.49.2:30767
    ```

    ```
    belajar@sabarmenanti:~$ curl 192.168.49.2:30767
    Pelatihan Cloud Native
    ```

    Terlihat konten *homepage* berhasil diperbaharui.

17. Menskalakan aplikasi agar ketika trafik meningkat maka dapat memenuhi permintaan pengguna sehingga ketersediaan layanan tetap terjaga. Penskalaan dilakukan dengan mengubah jumlah replika (**replicas**) dalam suatu *deployment*. Aplikasi yang telah di *deploy* sebelumnya hanya memiliki satu *pod* di dalamnya yang berjalan. Sebagai contoh akan dilakukan penskalaan menjadi 4 (empat) *pod* dengan mengeksekusi perintah berikut:

    ```
    $ kubectl scale deployment nginx-deploy --replicas=4
    ```

    ```
    belajar@sabarmenanti:~$ kubectl scale deployment nginx-deploy --replicas=4
    deployment.apps/nginx-deploy scaled
    ```

18. Memverifikasi hasil penskalaan menjadi 4 (empat) *pod*.

    ```
    $ kubectl get deployment
    ```

    ```
    belajar@sabarmenanti:~$ kubectl get deployment
    NAME           READY   UP-TO-DATE   AVAILABLE   AGE
    nginx-deploy   1/4     4            1           40m
    ```

    Terlihat *deployment* masih berlangsung yaitu dapat diketahui dari nilai pada kolom **AVAILABLE** bernilai 1 sehingga perlu menunggu beberapa saat. Jika ingin melihat perubahan nilai pada kolom tersebut secara *real time* maka eksekusi perintah:

    ```
    $ kubectl get deployment -w
    ```

    ```
    belajar@sabarmenanti:~$ kubectl get deployment -w
    NAME           READY   UP-TO-DATE   AVAILABLE   AGE
    nginx-deploy   1/4     4            1           40m
    nginx-deploy   2/4     4            2           40m
    nginx-deploy   3/4     4            3           40m
    nginx-deploy   4/4     4            4           40m
    ```

    Tekan **CTRL+C** untuk keluar dari pemantauan secara *real time* tersebut jika nilai pada kolom **AVAILABLE** telah bernilai 4 atau nilai kolom **READY** bernilai 4/4 yang bermakna proses penskalaan *pod* telah selesai dilakukan.

19. Melakukan pembaruan berkelanjutan (*rolling update*) yang memungkinkan pembaruan *deployment* dilakukan tanpa waktu henti (*zero downtime*). Hal ini dilakukan dengan

mengganti *Pod* yang ada dengan yang baru secara bertahap. *Pod* baru dijadwalkan pada *Node* dengan sumber daya yang tersedia dan *Kubernetes* menunggu *Pod* baru tersebut dimulai sebelum menghapus *Pod* lama. Sebagai contoh akan dilakukan pembaharuan pada *image* dari **nginx** menjadi **nginx:mainline** dengan mengeksekusi perintah:

```
$ kubectl set image deploy nginx-deploy nginx=nginx:mainline
```

```
belajar@sabarmenanti:~$ kubectl set image deploy nginx-deploy nginx=nginx:mainline
deployment.apps/nginx-deploy image updated
```

20. Memverifikasi keteraksesan layanan pada *pod* **nginx-deploy** menggunakan utilitas **curl** pada **port 30767**. Ganti nomor *port* dengan yang dimiliki sesuai dengan luaran dari eksekusi perintah `kubectl get services`. Alamat IP yang digunakan adalah IP dari *minikube* yang telah diidentifikasi pada langkah sebelumnya yaitu **192.168.49.2**.

    ```
    $ curl 192.168.49.2:30767
    ```

    Setiap kali menjalankan perintah **curl**, pengguna akan mendapatkan *Pod* yang berbeda.

21. Memverifikasi bahwa semua pod sekarang menjalankan versi terbaru yaitu **nginx:mainline**.

    ```
    $ kubectl describe pods
    ```

    Terlampir cuplikan hasil dari eksekusi perintah tersebut untuk salah satu *pod*:

```
belajar@sabarmenanti:~$ kubectl describe pods
Name:             nginx-deploy-86b5dfb4c9-7dqxx
Namespace:        default
Priority:         0
Service Account:  default
Node:             minikube/192.168.49.2
Start Time:       Wed, 10 Jul 2024 15:41:09 +0800
Labels:           app=nginx-deploy
                  pod-template-hash=86b5dfb4c9
Annotations:      <none>
Status:           Running
IP:               10.244.0.24
IPs:
  IP:           10.244.0.24
Controlled By:  ReplicaSet/nginx-deploy-86b5dfb4c9
Containers:
  nginx:
    Container ID:   docker://92ee45e1208118de030c72c65ec051a956559882316e1a49761956a7232cb51e
    Image:          nginx:mainline
```

    Terlihat *pod* tersebut telah menggunakan *image* **nginx:mainline** sehingga menandakan bahwa pembaharuan versi image berhasil dilakukan.

22. Menghapus *service* **nginx-deploy**.

    ```
    $ kubectl delete service nginx-deploy
    ```

    ```
    belajar@sabarmenanti:~$ kubectl delete service nginx-deploy
    service "nginx-deploy" deleted
    ```

23. Memverifikasi penghapusan *service*.

    ```
    $ kubectl get services
    ```

```
belajar@sabarmenanti:~$ kubectl get services
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   14h
```

Terlihat tidak terdapat *service* dengan nama **nginx-deploy** sehingga penghapusan *service* dengan nama tersebut berhasil dilakukan.

24. Menghapus *deployment* **nginx-deploy**.

    `$ kubectl delete deployment nginx-deploy`

```
belajar@sabarmenanti:~$ kubectl delete deployment nginx-deploy
deployment.apps "nginx-deploy" deleted
```

25. Memverifikasi penghapusan *deployment*.

    `$ kubectl get deployment`

```
belajar@sabarmenanti:~$ kubectl get deployment
No resources found in default namespace.
```

Terlihat tidak ada *resource* yang ditemukan.

26. Menampilkan informasi terkait *pod*.

```
belajar@sabarmenanti:~$ kubectl get pods
No resources found in default namespace.
```

Terlihat tidak ada *resource* yang ditemukan.

27. Menghentikan *Minikube*.

    `$ minikube stop`

```
belajar@sabarmenanti:~$ minikube stop
* Stopping node "minikube"  ...
* Powering off "minikube" via SSH ...
* 1 node stopped.
```

28. Menampilkan status *Minikube*.

    `$ minikube status`

```
belajar@sabarmenanti:~$ minikube status
minikube
type: Control Plane
host: Stopped
kubelet: Stopped
apiserver: Stopped
kubeconfig: Stopped
```

Terlihat *minikube* telah berhasil dihentikan ditandai dengan status *Stopped* pada *kubelet*, *apiserver* dan *kubeconfig*.

DAFTAR REFERENSI

Canonical, Install Ubuntu Desktop, 2024, https://ubuntu.com/tutorials/install-ubuntu-desktop#1-overview

Docker, Reference Documentation, 2024, https://docs.docker.com/reference/

Docker, Docker Hub Container Image Library, 2024, https://hub.docker.com/

Kubernetes, Kubernetes Documentation, 2024, https://kubernetes.io/docs/home/

Minikube, Minikube Documentation, 2024, https://minikube.sigs.k8s.io/docs/

TENTANG PENULIS

I Putu Hariyadi

adalah dosen di program studi Ilmu Komputer, Universitas Bumigora, Mataram, Nusa Tenggara Barat (NTB). Penulis sangat antusias untuk mendalami dunia Teknologi Informasi & Komunikasi (TIK). Memiliki ketertarikan pada bidang Jaringan Komputer, *Network Programmability*, *Cloud Computing*, *Pemrograman Web* dan Keamanan Sistem Informasi serta Sistem Temu Kembali Informasi (*Information Retrieval*). Sebagian besar pengalaman penulis ketika mengeksplorasi bidang tersebut dituangkan pada situs pribadi yang beralamat di https://www.iputuhariyadi.net. Untuk korespondensi dapat menghubungi penulis melalui email di alamat: admin@iputuhariyadi.net atau putu.hariyadi@universitasbumigora.ac.id.

LAMPIRAN

A. Latihan Pembuatan Container Image nginx Virtual Host

Container image yang dibuat menggunakan *base image* **nginx:latest** dan dilakukan pengaturan 3 (tiga) *virtual host* dengan ketentuan sebagai berikut:

- **default.conf** agar menggunakan **port 9090** dan **Document Root** yang menampung *homepage* berupa *file* "**index.html**" berada di direktori **/usr/share/nginx/html**. Konten dari *file* "**index.html**" adalah "**Selamat datang di Situs Utama [9090]**".
- **makassar.conf**, agar menggunakan **port 8080** dan **Document Root** yang menampung *homepage* berupa *file* "**index.html**" berada di direktori **/makassar**. Konten dari *file* "**index.html**" adalah "**Selamat datang di Situs Makassar [8080]**".
- **jambi.conf** agar menggunakan **port 7070** dan **Document Root** yang menampung *homepage* berupa *file* "**index.html**" berada di direktori **/jambi**. Konten dari *file* "**index.html**" adalah "**Selamat datang di Situs Jambi [7070]**".

Nama *container image* yang dibuat adalah **nginx-vhost:latest.**

Jawaban:

1. Membuat dan menjalankan *container* di *background* bernama **nginx** menggunakan *image* **nginx:latest** dengan *port* yang dipublikasikan pada *host machine* adalah **80** dan port 80 juga di dalam *container*.

```
$ docker run -d --name nginx -p 80:80 nginx:latest
```

2. Mengambil *file* **default.conf** dari dalam *container* **nginx** yang berjalan dan disimpan ke lokasi direktori saat ini berada.

```
$ docker cp nginx:/etc/nginx/conf.d/default.conf .
```

```
belajar@sabarmenanti:~$ docker cp nginx:/etc/nginx/conf.d/default.conf .
Successfully copied 3.07kB to /home/belajar/.
```

3. Menyalin *file* **default.conf** menjadi **makassar.conf** dan **jambi.conf**.

```
$ cp default.conf makassar.conf
$ cp default.conf jambi.conf
```

4. Mengubah nilai *directive* **listen** pada *file* **default.conf** dari nomor *port* **80** menjadi **9090** menggunakan utilitas **sed**.

```
$ sed -i 's/80/9090/g' default.conf
```

5. Memverifikasi pengubahan *port* pada *directive* **listen** di *file* **default.conf**.

```
$ grep -w listen default.conf
```

```
belajar@sabarmenanti:~$ grep -w listen default.conf
    listen       9090;
    listen  [::]:9090;
```

Terlihat *directive* **listen** telah bernilai **9090**.

6. Mengubah nilai *directive* **listen** pada *file* **makassar.conf** dari nomor *port* **80** menjadi **8080** menggunakan utilitas **sed**.

```
$ sed -i 's/80/8080/g' makassar.conf
```

7. Memverifikasi pengubahan *port* pada *directive* **listen** di *file* **makassar.conf**.

```
$ grep -w listen makassar.conf
```

```
belajar@sabarmenanti:~$ grep -w listen makassar.conf
    listen       8080;
    listen  [::]:8080;
```

Terlihat *directive* **listen** telah bernilai **8080**.

8. Mengubah nilai *directive* **root** yang mengatur *document root* pada *file virtual host* **makassar.conf** dari **/usr/share/nginx/html** menjadi **/makassar** menggunakan utilitas *sed*.

```
$ sed -i 's/\/usr\/share\/nginx\/html/\/makassar/' makassar.conf
```

9. Mengubah nilai *directive* **listen** pada *file* **jambi.conf** dari nomor *port* **80** menjadi **7070** menggunakan utilitas **sed**.

```
$ sed -i 's/80/7070/g' jambi.conf
```

10. Memverifikasi pengubahan *port* pada *directive* **listen** di *file* **jambi.conf**.

```
$ grep -w listen jambi.conf
```

```
belajar@sabarmenanti:~$ grep -w listen jambi.conf
    listen       7070;
    listen  [::]:7070;
```

Terlihat *directive* **listen** telah bernilai **7070**.

11. Mengubah nilai *directive* **root** yang mengatur *document root* pada *file virtual host* **jambi.conf** dari **/usr/share/nginx/html** menjadi **/jambi** menggunakan utilitas *sed*.

```
$ sed -i 's/\/usr\/share\/nginx\/html/\/jambi/' jambi.conf
```

12. Membuat *template file homepage* untuk *virtual host* **default.conf** dengan nama *file* "**index.html**" dan memverifikasi hasil pembuatan filenya.

```
$ echo Selamat datang di Situs Utama [9090] > index.html
$ cat index.html
```

```
belajar@sabarmenanti:~$ echo Selamat datang di Situs Utama [9090] > index.html
belajar@sabarmenanti:~$ cat index.html
Selamat datang di Situs Utama [9090]
```

13. Membuat *template file homepage* untuk *virtual host* **makassar.conf** dengan nama *file* "**makassar.html**" dan memverifikasi hasil pembuatan filenya.

    ```
    $ echo Selamat datang di Situs Makassar [8080] > makassar.html
    $ cat makassar.html
    ```

```
belajar@sabarmenanti:~$ echo Selamat datang di Situs Makassar [8080] > makassar.html
belajar@sabarmenanti:~$ cat makassar.html
Selamat datang di Situs Makassar [8080]
```

14. Membuat *template file homepage* untuk *virtual host* **jambi.conf** dengan nama *file* "**jambi.html**" dan memverifikasi hasil pembuatan filenya.

    ```
    $ echo Selamat datang di Situs Jambi [9090] > jambi.html
    $ cat jambi.html
    ```

```
belajar@sabarmenanti:~$ echo Selamat datang di Situs Jambi [7070] > jambi.html
belajar@sabarmenanti:~$ cat jambi.html
Selamat datang di Situs Jambi [7070]
```

15. Membuat direktori **makassar** dan **jambi** di dalam *container* **nginx** yang sedang berjalan serta memverifikasi hasil pembuatan direktori tersebut.

    ```
    $ docker exec nginx mkdir /makassar
    $ docker exec nginx mkdir /jambi
    $ docker exec nginx ls -l
    ```

```
belajar@sabarmenanti:~$ docker exec nginx mkdir /makassar
belajar@sabarmenanti:~$ docker exec nginx mkdir /jambi
belajar@sabarmenanti:~$ docker exec nginx ls -l
total 72
lrwxrwxrwx   1 root root    7 Jul  1 00:00 bin -> usr/bin
drwxr-xr-x   2 root root 4096 Mar 29 17:20 boot
drwxr-xr-x   5 root root  340 Jul 10 09:02 dev
drwxr-xr-x   1 root root 4096 Jul  2 03:20 docker-entrypoint.d
-rwxr-xr-x   1 root root 1620 Jul  2 03:19 docker-entrypoint.sh
drwxr-xr-x   1 root root 4096 Jul 10 09:02 etc
drwxr-xr-x   2 root root 4096 Mar 29 17:20 home
drwxr-xr-x   2 root root 4096 Jul 10 09:19 jambi
lrwxrwxrwx   1 root root    7 Jul  1 00:00 lib -> usr/lib
lrwxrwxrwx   1 root root    9 Jul  1 00:00 lib64 -> usr/lib64
drwxr-xr-x   2 root root 4096 Jul 10 09:18 makassar
drwxr-xr-x   2 root root 4096 Jul  1 00:00 media
drwxr-xr-x   2 root root 4096 Jul  1 00:00 mnt
drwxr-xr-x   2 root root 4096 Jul  1 00:00 opt
dr-xr-xr-x 233 root root    0 Jul 10 09:02 proc
drwx------   2 root root 4096 Jul  1 00:00 root
drwxr-xr-x   1 root root 4096 Jul 10 09:02 run
lrwxrwxrwx   1 root root    8 Jul  1 00:00 sbin -> usr/sbin
drwxr-xr-x   2 root root 4096 Jul  1 00:00 srv
dr-xr-xr-x  13 root root    0 Jul 10 09:02 sys
drwxrwxrwt   2 root root 4096 Jul  1 00:00 tmp
drwxr-xr-x   1 root root 4096 Jul  1 00:00 usr
drwxr-xr-x   1 root root 4096 Jul  1 00:00 var
```

Terlihat telah berhasil dibuat direktori **makassar** dan **jambi** di dalam *container* **nginx** yang sedang berjalan.

16. Menyalinkan *file* **index.html** dari *host machine* ke direktori **/usr/share/nginx/html** di dalam *container* **nginx**.

```
$ docker cp index.html nginx:/usr/share/nginx/html/index.html
```

```
belajar@sabarmenanti:~$ docker cp index.html nginx:/usr/share/nginx/html/index.html
Successfully copied 2.05kB to nginx:/usr/share/nginx/html/index.html
```

17. Menyalinkan *file* **makassar.html** dari *host machine* ke direktori **/makassar** di dalam *container* **nginx**.

```
$ docker cp makassar.html nginx:/makassar/index.html
```

```
belajar@sabarmenanti:~$ docker cp makassar.html nginx:/makassar/index.html
Successfully copied 2.05kB to nginx:/makassar/index.html
```

18. Menyalinkan *file* **jambi.html** dari *host machine* ke direktori **/jambi** di dalam *container* **nginx**.

```
$ docker cp jambi.html nginx:/jambi/index.html
```

```
belajar@sabarmenanti:~$ docker cp jambi.html nginx:/jambi/index.html
Successfully copied 2.05kB to nginx:/jambi/index.html
```

19. Menyalinkan *file* konfigurasi *virtual host* masing-masing dengan nama **default.conf**, **makassar.conf** dan **jambi.conf** ke direktori **/etc/nginx/conf.d** di dalam *container* **nginx**.

```
$ docker cp default.conf nginx:/etc/nginx/conf.d/default.conf
$ docker cp makassar.conf nginx:/etc/nginx/conf.d/makassar.conf
$ docker cp jambi.conf nginx:/etc/nginx/conf.d/jambi.conf
```

```
belajar@sabarmenanti:~$ docker cp default.conf nginx:/etc/nginx/conf.d/default.conf
Successfully copied 3.07kB to nginx:/etc/nginx/conf.d/default.conf
belajar@sabarmenanti:~$ docker cp makassar.conf nginx:/etc/nginx/conf.d/makassar.conf
Successfully copied 3.07kB to nginx:/etc/nginx/conf.d/makassar.conf
belajar@sabarmenanti:~$ docker cp jambi.conf nginx:/etc/nginx/conf.d/jambi.conf
Successfully copied 3.07kB to nginx:/etc/nginx/conf.d/jambi.conf
```

20. Memverifikasi hasil penyalinan *file-file virtual host* ke dalam *container* **nginx**.

```
$ docker exec nginx ls /etc/nginx/conf.d
```

```
belajar@sabarmenanti:~$ docker exec nginx ls /etc/nginx/conf.d
default.conf
jambi.conf
makassar.conf
```

Terlihat seluruh *file* telah berhasil disalinkan ke dalam direktori tersebut.

21. Memverifikasi sintak konfigurasi *nginx* di dalam *container* **nginx**.

```
$ docker exec nginx nginx -t
```

```
belajar@sabarmenanti:~$ docker exec nginx nginx -t
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
```

22. Melakukan *reload service* dari *nginx* di dalam *container* **nginx** agar perubahan konfigurasi diterapkan.

```
$ docker exec nginx nginx -s reload
```

```
belajar@sabarmenanti:~$ docker exec nginx nginx -s reload
2024/07/10 09:30:54 [notice] 60#60: signal process started
```

23. Membuat *container image* baru dari *container nginx* yang dilakukan perubahan.

```
$ docker commit nginx nginx-vhost:latest
```

```
belajar@sabarmenanti:~$ docker commit nginx nginx-vhost:latest
sha256:ff3152a7a79f2464a260301cf31e3eef61f625b45d62ca9cc7472f7900ce6aab
```

24. Memverifikasi hasil pembuatan *container image* bernama **nginx-vhost:latest**.

```
belajar@sabarmenanti:~$ docker image ls
REPOSITORY                    TAG       IMAGE ID       CREATED          SIZE
nginx-vhost                   latest    ff3152a7a79f   4 seconds ago    188MB
httpd                         latest    c0c20df5e7be   6 days ago       148MB
nginx                         latest    fffffc90d343   2 weeks ago      188MB
mariadb                       latest    4486d64c9c3b   4 weeks ago      406MB
gcr.io/k8s-minikube/kicbase   v0.0.44   5a6e59a9bdc0   2 months ago     1.26GB
phpmyadmin/phpmyadmin         latest    933569f3a9f6   11 months ago    562MB
```

Terlihat terdapat *image* dengan nama **nginx-vhost:latest**.

25. Mengujicoba membuat dan menjalankan *container* baru bernama **nginx-vhost** di *background* menggunakan *image* **nginx-vhost:latest** yang mempublikasikan *port* 7070 pada *host machine* ke *port* 7070 di dalam *container* dan mempublikasikan *port* 8080 pada *host machine* ke *port* 8080 di dalam *container* serta mempublikasikan *port* 9090 pada *host machine* ke *port* 9090 di dalam *container*.

```
$ docker run -d --name nginx-vhost -p 7070:7070 \
 -p 8080:8080 -p 9090:9090 nginx-vhost:latest
```

```
belajar@sabarmenanti:~$ docker run -d --name nginx-vhost -p 7070:7070 \
> -p 8080:8080 -p 9090:9090 nginx-vhost:latest
d4f62c56e6d623a58e8a91ed5bd42ef3af86d4dd850df7d6a5f6c83c7142d7af
```

26. Memverifikasi apakah *container* **nginx-vhost** telah berhasil dibuat dan berjalan.

```
$ docker ps
```

```
belajar@sabarmenanti:~$ docker ps
CONTAINER ID   IMAGE                COMMAND                  CREATED          STATUS              PORTS
                                                                             NAMES
d4f62c56e6d6   nginx-vhost:latest   "/docker-entrypoint.…"   4 seconds ago    Up 3 seconds        0.0.0.0:7070->7070/tcp, :::7070->7070/tcp, 0.0.0.0:8080->8
080/tcp, :::8080->8080/tcp, 80/tcp, 0.0.0.0:9090->9090/tcp, :::9090->9090/tcp   nginx-vhost
d67bc06218d6   nginx:latest         "/docker-entrypoint.…"   About an hour ago   Up About an hour   0.0.0.0:80->80/tcp, :::80->80/tcp
                                                                             nginx
```

27. Memverifikasi mengakses layanan pada *container* **nginx-vhost** menggunakan **curl** ke *port* 7070, 8080 dan 9090.

```
$ curl localhost:7070
$ curl localhost:8080
$ curl localhost:9090
```

```
belajar@sabarmenanti:~$ curl localhost:7070
Selamat datang di Situs Jambi [7070]
belajar@sabarmenanti:~$ curl localhost:8080
Selamat datang di Situs Makassar [8080]
belajar@sabarmenanti:~$ curl localhost:9090
Selamat datang di Situs Utama [9090]
```

28. Menghentikan dan menghapus *container* **nginx-vhost** dan **nginx**.

```
$ docker rm -f nginx-vhost
$ docker rm -f nginx
```

```
belajar@sabarmenanti:~$ docker rm -f nginx-vhost
nginx-vhost
belajar@sabarmenanti:~$ docker rm -f nginx
nginx
```

B. Latihan Docker Fundamental

1. Mengunduh *docker container image* bernama **httpd** dengan **tag "latest"**.

   Jawaban:

```
$ docker pull httpd:latest
```

```
belajar@sabarmenanti:~$ docker pull httpd:latest
latest: Pulling from library/httpd
f11c1adaa26e: Already exists
3bce494dbe9a: Pull complete
4f4fb700ef54: Pull complete
0ec6a44b37fe: Pull complete
e4822864e326: Pull complete
65dfcadc56f2: Pull complete
Digest: sha256:fad0c8311b35c689cf1b94fc4783e735a8e3086aebfe318c4e8e0fa224e9ce
1b
Status: Downloaded newer image for httpd:latest
docker.io/library/httpd:latest
```

2. Memverifikasi *image* **httpd** dengan **tag "latest"** telah berhasil diunduh.

   Jawaban:

```
$ docker image ls
```

```
belajar@sabarmenanti:~$ docker image ls
REPOSITORY                   TAG       IMAGE ID       CREATED         SIZE
httpd                        latest    c0c20df5e7be   6 days ago      148MB
nginx                        latest    fffffc90d343   2 weeks ago     188MB
mariadb                      latest    4486d64c9c3b   4 weeks ago     406MB
gcr.io/k8s-minikube/kicbase  v0.0.44   5a6e59a9bdc0   2 months ago    1.26GB
phpmyadmin/phpmyadmin        latest    933569f3a9f6   11 months ago   562MB
```

3. Membuat dan menjalankan *container* dengan ketentuan sebagai berikut:
   - Nama *container* adalah **httpd-namapanggilan**. Ganti kata "**namapanggilan**" dengan nama panggilan Anda. Sebagai contoh pada solusi ini menggunakan nama panggilan "**putu**".
   - *Container image* menggunakan **httpd** dengan **tag "latest"**.
   - Merutekan trafik dari **port 1234 di mesin host ke port 80 di dalam container**.
   - *Container* berjalan di *background*.

   Jawaban:

   ```
   $ docker run -d --name httpd-putu -p 1234:80 \
    httpd:latest
   ```

```
belajar@sabarmenanti:~$ docker run -d --name httpd-putu -p 1234:80 \
> httpd:latest
927b5ce9d02adcabac73a2d25d1531d0a85833ecfff5462a9360859264ee07c1
```

4. Memverifikasi *container* yang sedang berjalan.

   Jawaban:

   ```
   $ docker ps
   ```

```
belajar@sabarmenanti:~$ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED              STATUS              PORTS                                   NAMES
927b5ce9d02a   httpd:latest   "httpd-foreground"   About a minute ago   Up About a minute   0.0.0.0:1234->80/tcp, :::1234->80/tcp   httpd-putu
```

5. Mengakses layanan yang berjalan pada *container* **httpd-namapanggilan** menggunakan **curl**. Ganti kata "**namapanggilan**" dengan nama panggilan Anda.

   Jawaban:

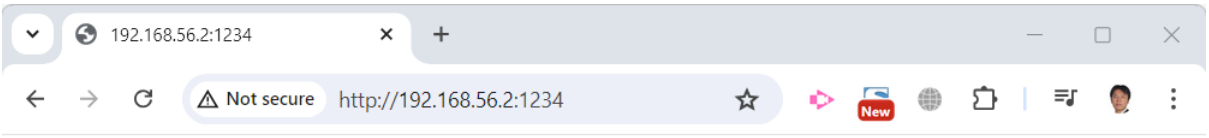
   ```
   $ curl localhost:1234
   ```

```
belajar@sabarmenanti:~$ curl localhost:1234
<html><body><h1>It works!</h1></body></html>
```

6. Mengakses layanan yang berjalan pada *container* **httpd-namapanggilan** menggunakan *browser* di *Windows*. Ganti kata "**namapanggilan**" dengan nama panggilan Anda.

   Jawaban:

   Buka *browser* yang ada di *Windows* dan pada *address bar* masukkan alamat http://192.168.56.2:1234, seperti terlihat pada gambar berikut:



It works!

7. Mengeksekusi perintah di dalam *container* **httpd-namapanggilan** untuk menampilkan isi dari *file index.html* pada direktori **/usr/local/apache2/htdocs**. Ganti kata "**namapanggilan**" dengan nama panggilan Anda.

   Jawaban:

   ```
   $ docker exec -it httpd-putu bash
   # cd htdocs
   # cat index.html
   ```

   ```
   belajar@sabarmenanti:~$ docker exec -it httpd-putu bash
   root@927b5ce9d02a:/usr/local/apache2# cd htdocs
   root@927b5ce9d02a:/usr/local/apache2/htdocs# cat index.html
   <html><body><h1>It works!</h1></body></html>
   ```

8. Mengubah halaman *homepage* dari *container* **httpd-namapanggilan** agar menampilkan konten sebagai berikut:

   Pelatihan Cloud Native - [NAMA LENGKAP]

   Ganti kata **[NAMA LENGKAP]** dengan nama lengkap Anda.

   Jawaban:

   ```
   # echo \
   > "Pelatihan Cloud Native - I Putu Hariyadi" > index.html
   # exit
   ```

   ```
   root@927b5ce9d02a:/usr/local/apache2/htdocs# echo \
   > "Pelatihan Cloud Native - I Putu Hariyadi" > index.html
   root@927b5ce9d02a:/usr/local/apache2/htdocs# exit
   exit
   ```

9. Memverifikasi hasil perubahan konten *homepage* pada *container* **httpd-namapanggilan** menggunakan **curl**. Ganti kata "**namapanggilan**" dengan nama panggilan Anda.

   Jawaban:

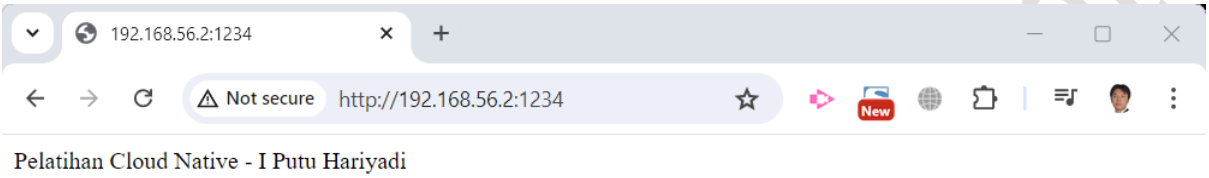
   ```
   $ curl localhost:1234
   ```

```
belajar@sabarmenanti:~$ curl localhost:1234
Pelatihan Cloud Native - I Putu Hariyadi
```

10. Memverifikasi hasil perubahan konten *homepage* pada *container* **httpd-namapanggilan** menggunakan *browser* di *Windows*. Ganti kata "**namapanggilan**" dengan nama panggilan Anda.

    Jawaban:

    Buka *browser* yang ada di *Windows* dan pada *address bar* masukkan alamat http://192.168.56.2:1234, seperti terlihat pada gambar berikut:



11. Menghentikan dan menghapus *container* **httpd-namapanggilan** hanya dengan menggunakan satu perintah. Ganti kata "**namapanggilan**" dengan nama panggilan Anda.

    Jawaban:

    ```
    $ docker rm -f httpd-putu
    ```

    ```
    belajar@sabarmenanti:~$ docker rm -f httpd-putu
    httpd-putu
    ```

12. Memverifikasi hasil penghentian dan penghapusan *container*.

    Jawaban:

    ```
    $ docker ps -a
    ```

```
belajar@sabarmenanti:~$ docker ps -a
CONTAINER ID   IMAGE                                COMMAND                  CREATED         STATUS
```