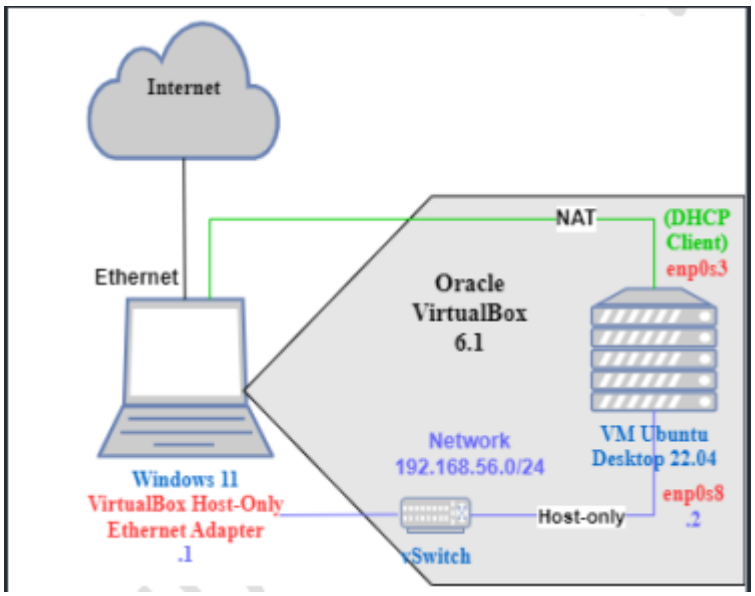
BAB I
INSTALASI DAN KONFIGURASI UBUNTU DESKTOP VERSI 22.04 LTS PADA ORACLE VIRTUALBOX 6.1

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:

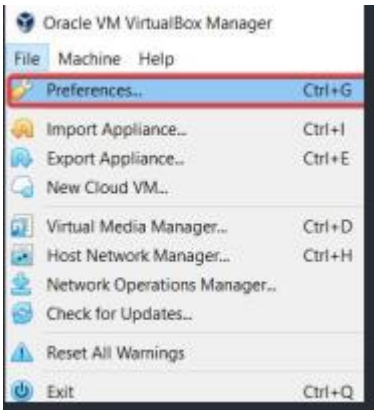


Pada Oracle VirtualBox akan dibuat Guest Virtual Machine (VM) dengan sistem operasi Ubuntu Desktop versi 22.04 LTS. Alamat network yang digunakan untuk komunikasi antara Host Machine Windows 11 dengan VM Ubuntu adalah 192.168.56.0/24. Interface VirtualBox Host-Only Ethernet Adapter pada Windows 11 menggunakan 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.

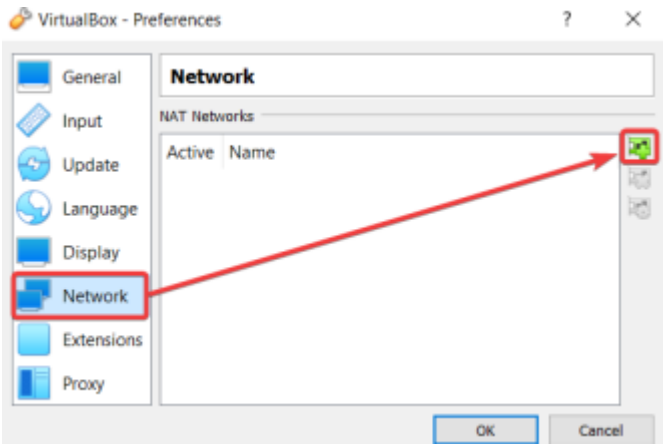
B. Instalasi Ubuntu Desktop Versi 22.04 LTS

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

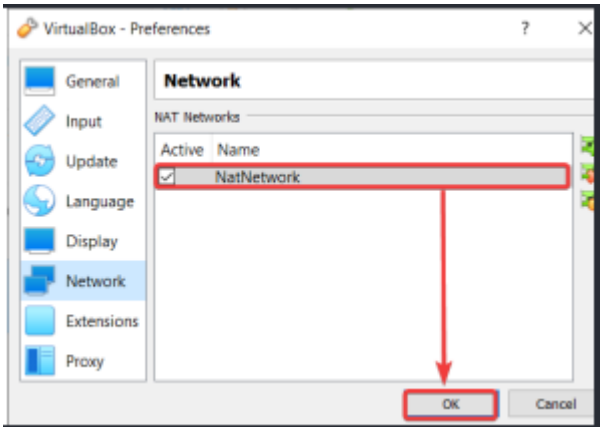
1. Jalankan aplikasi Oracle VirtualBox melalui Start > Oracle VM VirtualBox > Oracle VM VirtualBox.
2. Tampil aplikasi Oracle VM VirtualBox Manager dan lakukan penambahan Network Address Translation (NAT) Network dengan mengakses menu File > Preferences pada Oracle VM VirtualBox Manager, seperti yang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



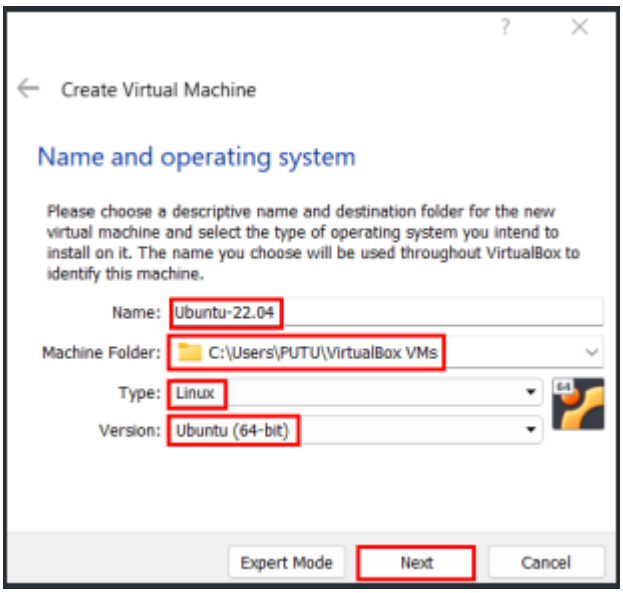
3. Tampil kotak dialog VirtualBox – Preferences dan pada panel menu sebelah kiri pilih Network. Pada panel detail dari Network di sebelah kanan, pilih icon Adds new NAT Network, seperti yang ditunjukkan pada gambar berikut:



4. Terbentuk “NatNetwork” yang telah aktif, seperti terlihat pada gambar berikut:



5. 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:



C. Remote Server

Manfaat SSH

1. Mengontrol server dengan lebih mudah dan bebas — Anda bisa dengan mudah mengakses dan mengontrol file dan pengaturan yang ada pada hosting tanpa harus mendatangi dimana server fisik tersimpan.

2. Menghindari cyber crime — dengan ketiga teknik kriptografi yang ada, SSH akan secara otomatis memutus koneksi jika mendeteksi aktivitas mencurigakan saat proses transfer terjadi. Tentunya hal ini akan melindungi Anda dari hacker dan malware.
3. DNS dan IP spoofing — fitur spoofing yang ada di SSH berfungsi untuk melindungi proses transfer dari hacker atau peretas, untuk lebih jelasnya baca artikel tentang spoofing berikut ini.
4. Keamanan yang terjamin — dengan sistem autentikasi dan enkripsi data, koneksi data yang dilakukan melalui SSH mempunyai keamanan yang ketat dan terjamin

D. Cara Kerja SSH

Cara kerja SSH dimulai dari SSH client yang akan melakukan koneksi ke SSH server. Pada proses koneksi ini, SSH client menggunakan teknik kriptografi untuk melakukan otentikasi dan identifikasi SSH server. Jika kunci yang dipakai SSH client tidak sama dengan kunci yang ada pada server, koneksi akan secara otomatis gagal. Selanjutnya, koneksi yang sukses akan dienkripsi melalui symmetric encryption dan hashing yang berfungsi untuk memastikan privasi dan keutuhan data yang ditukar selama proses transfer.cUntuk membuat Anda memahami lebih jelas cara kerja SSH, berikut adalah beberapa tahapan-tahapannya:

1. SSH client melakukan koneksi ke SSH server menggunakan port default 22.
2. Jika SSH client dan SSH server menggunakan versi yang sama, koneksi akan sukses.
3. Selanjutnya, sSH client meminta public key dan host key dari SSH server dan melakukan verifikasi.
4. Jika kedua kunci yang digunakan sama, SSH client akan membentuk session key dan mengenkripsi kunci tersebut melalui public key milik SSH server.
5. Kemudian, SSH server akan mengenkripsi balik public key milik SSH client untuk verifikasi.
6. User dari SSH client lalu harus melakukan otentikasi diri dengan cara memasukan username dan password yang sudah ada.
7. Setelah verifikasi pengguna berhasil, koneksi sudah terbentuk dengan aman

E. Langkah Kerja

1. Perbarui sistem operasi Debian/ubuntu untuk memastikan semua paket yang ada mutakhir:

   ```
   sudo apt update && sudo apt upgrade -y
   ```

2. Jalankan perintah berikut di terminal Anda untuk memulai instalasi.

   ```
   apt install open-ssh
   ```

   Ketik Y, lalu tekan tombol MASUKKAN KUNCI untuk melanjutkan.

3. Setelah terinstal, aktifkan SSH menggunakan perintah berikut.

   ```
   sudo systemctl enable --now ssh
   ```

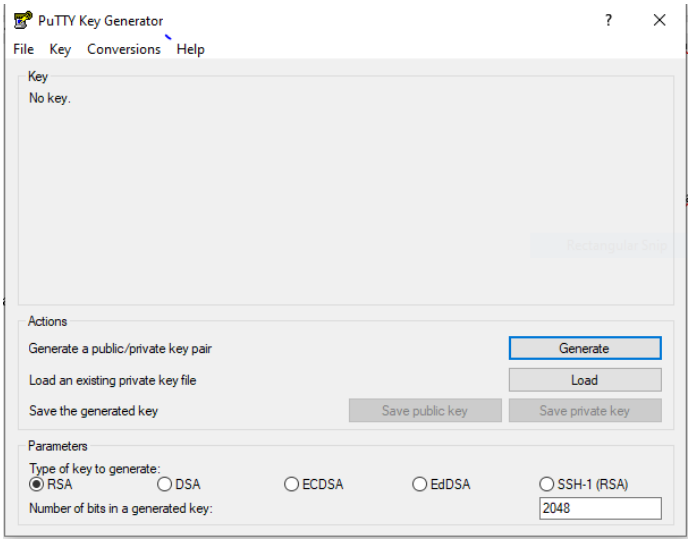
4. Periksa status instance SSH untuk memastikannya berjalan dan tidak ada kesalahan yang terjadi.

   ```
   systemctl status ssh
   ```

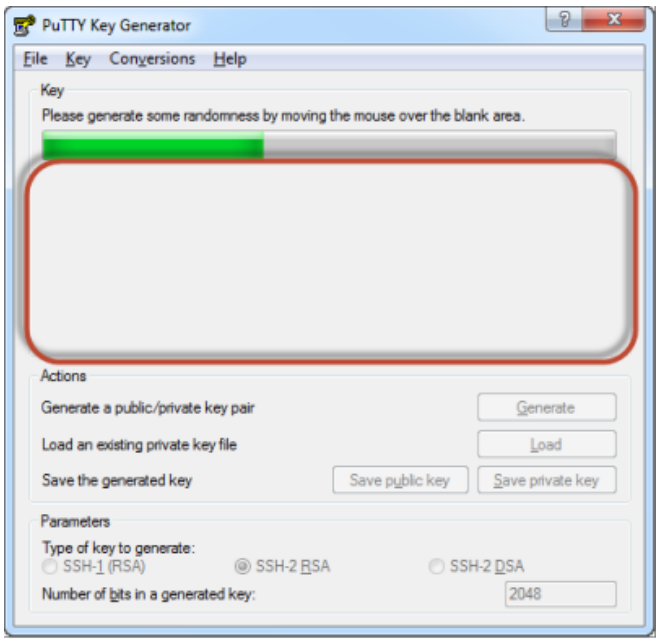
5. Dengan SSH sekarang diinstal dan diaktifkan, Anda dapat terhubung ke system jarak jauh, PC jarak jauh lain, atau server. Ingat, perangkat lunak SSH harus diaktifkan di kedua ujungnya agar ini berfungsi.
6. Pertama, Anda harus memiliki alamat IP internal/eksternal atau nama host dan sebaiknya nama pengguna akun yang terhubung juga, kemudian gunakan perintah ssh untuk memulai koneksi.

F. Langkah Praktikum Membuat Keys SSH Pada Putty

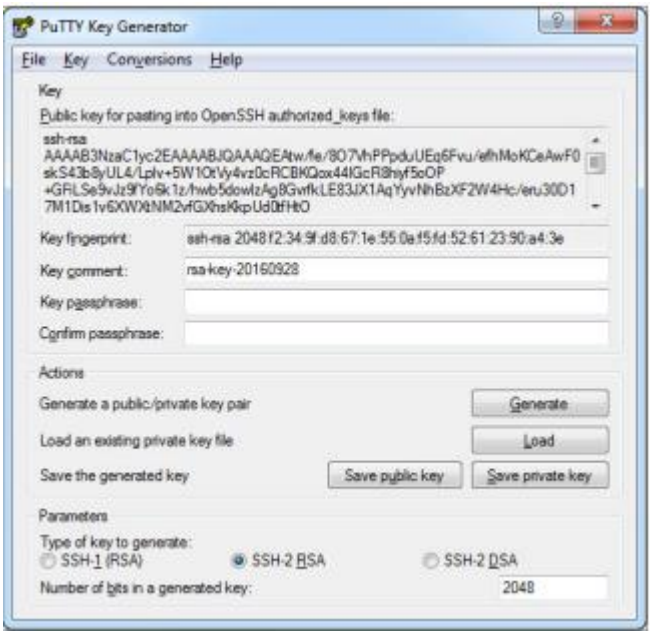
1. Install Putty sebagai SSH Client.
2. Jalankan PuTTYgen untuk membuat key. PuTTYgen adalah aplikasi khusus dari PuTTY untuk membuat public key dan private key
3. Klik tombol Generate untuk membuat public key dan private key.



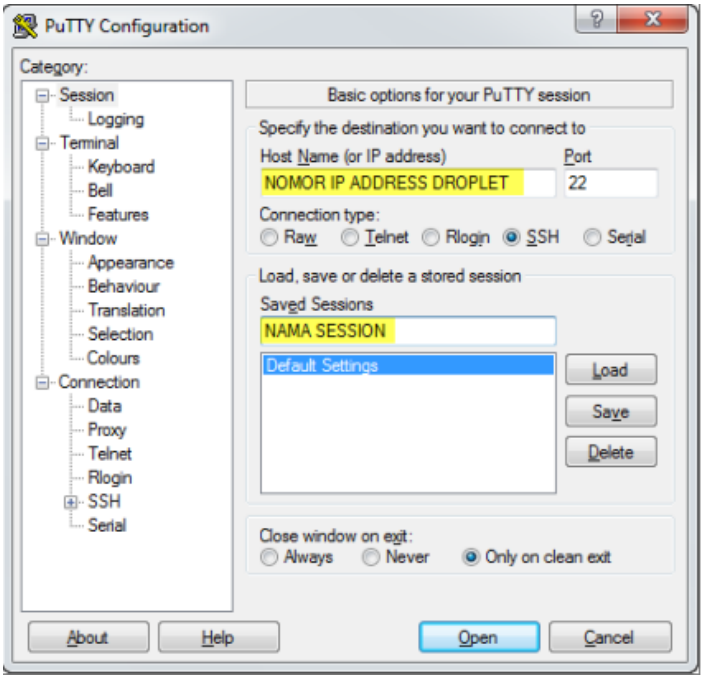
4. Untuk parameter, pilih saja settingan default yang diberikan oleh PuTTY.
5. Gerakkan mouse di bagian kotak merah agar PuTTY bisa membuat key. Gerakkan terus sampai progress bar terisi penuh.



6. Jika sudah selesai maka simpan public key dan private key dengan nama file yang berbeda.

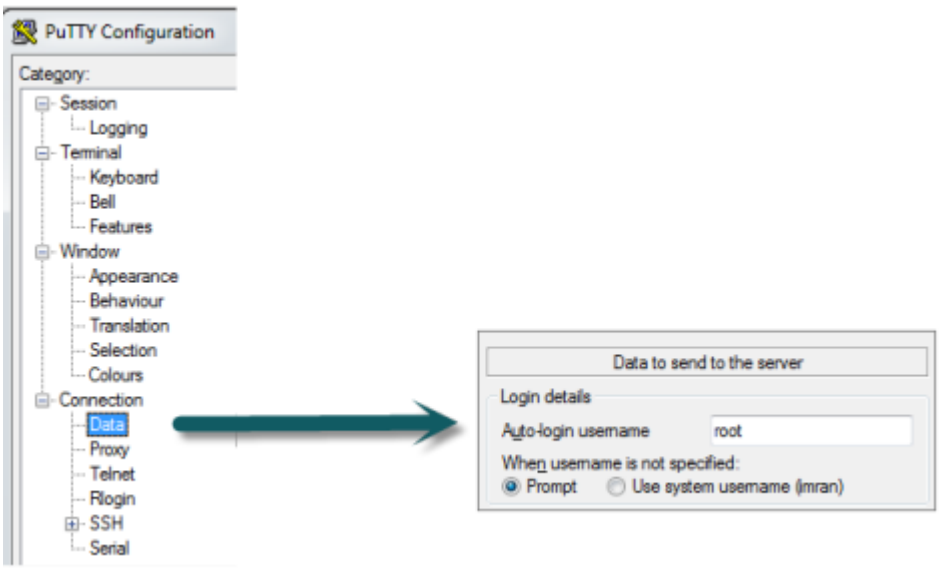


7. Pada bagian yang diberi kotak merah, copy mulai dari ssh-rsa sampai ke bagian bawah.
8. Buka notepad, dan paste ke dalam notepad. Simpan dalam bentuk file text.
9. Kemudian pindahkan file notepad ke komputer yang biasa digunakan untuk mengakses server VPS.
10. Pada komputer yang biasa digunakan untuk akses server VPS, buka file tersebut dengan editor text biasa. Bisa menggunakan notepad. Select All semua isi file tersebut kemudian copy.
11. Paste ke dalam file authorized_keys di server VPS. Tentu saja kamu harus sudah menambah user baru ssh di server VPS.
12. Setelah public key tersimpan di server VPS, maka kamu bisa mencoba melakukan koneksi dengan PuTTY ke server VPS
13. Buka PuTTY. Kamu akan langsung dihadapkan pada PuTTY Configuration

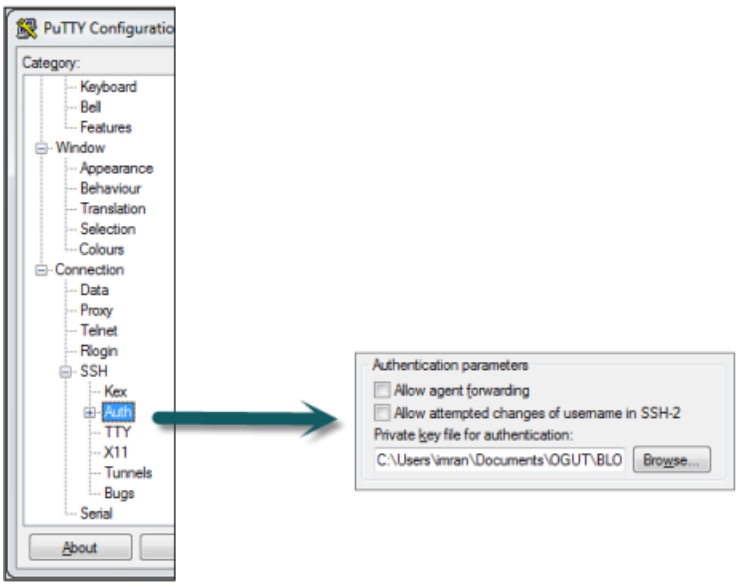


14. Masukkan nomor IP server dan port untuk mengakses server. Secara default port ssh adalah port 22. Tapi port itu bisa dirubah di file /etc/ssh/sshd_config

15. Pada bagian Category > Connection > Data, masukkan auto login username pada Server



16. Kemudian beralih pada Category > Connection > SSH > Auth

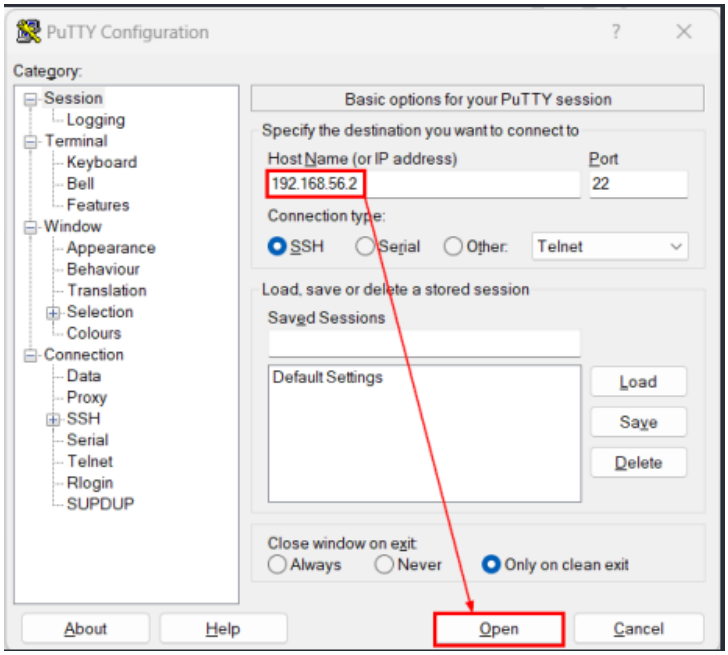


17. Klik tombol Browse dan arahkan pada folder file private key disimpan.
18. Kembali ke Category > Session, masukkan name session pada bagian “Saved Sessions” dan klik Save untuk menyimpan.

G. Instalasi Docker Engine

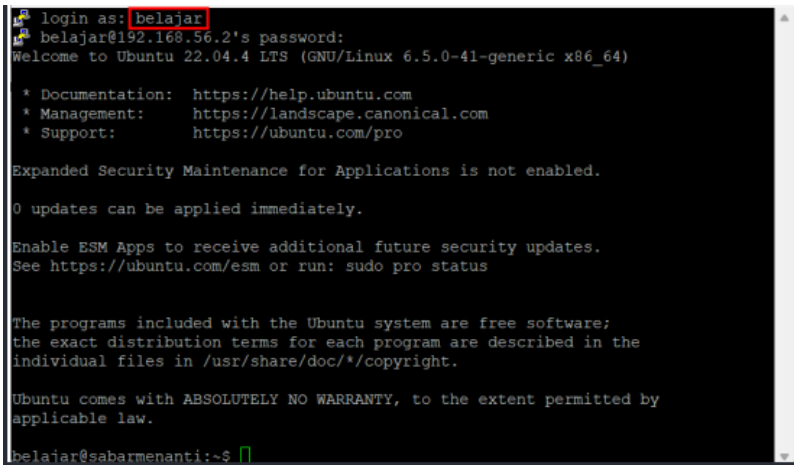
Adapun langkah-langkah menginstalasi dan mengkonfigurasi Docker Engine pada Ubuntu Desktop versi 22.04 LTS menggunakan Docker apt repository adalah sebagai berikut:

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.

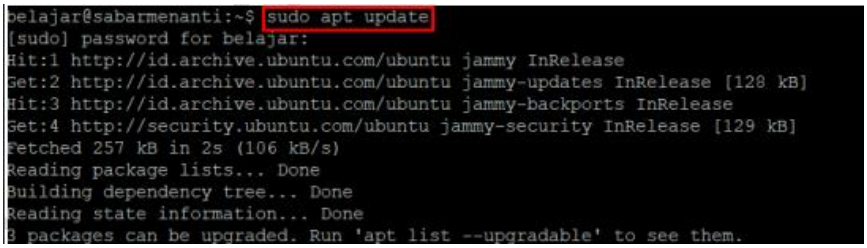


2. Tampil kotak dialog Putty yang meminta pengguna untuk melakukan proses otentikasi login ke Ubuntu Desktop, seperti terlihat pada gambar berikut:

3. 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:



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

   ```
   $ sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg
   -o /etc/apt/k
   ```

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-plug

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

    $ sudo docker run hello-world

```
zeus@zeus:~$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
c1ec31eb5944: Pull complete
Digest: sha256:94323f3e5e09a8b9515d74337010375a456c909543e1ff1538f5116d38ab3989
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.
```

11. Menampilkan informasi versi Docker yang terinstalasi.

    $ sudo docker --version

```
zeus@zeus:~$ docker --version
Docker version 27.0.3, build 7d4bcd8
zeus@zeus:~$
```

    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

    Terlihat user yang 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 “zeus” dengan password “12345”, seperti terlihat pada gambar berikut:

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

    $ id

    Terlihat user “belajar” menjadi anggota dari group Bernama docker dengan group ID 999.



17. Menampilkan informasi versi dari Docker tanpa menggunakan sudo.



BAB 2 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:

H. Manajemen docker container

9. 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
   ```

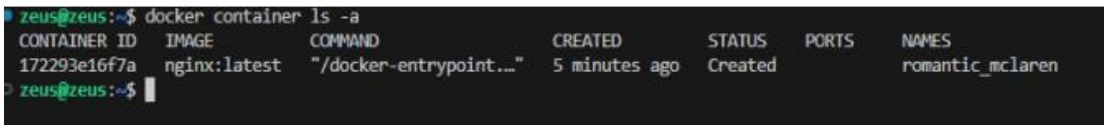
   Sebagai contoh untuk membuat container dari image “nginx” dengan tag “latest” di VM Ubuntu dan tanpa menjalankannya maka dapat mengeksekusi perintah:

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

   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 “172293e16f7a021957491464e2e1b364dbbf7c071a39777b7ada730a9580878a”.

10. Menampilkan daftar docker container yang sedang berjalan.

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



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

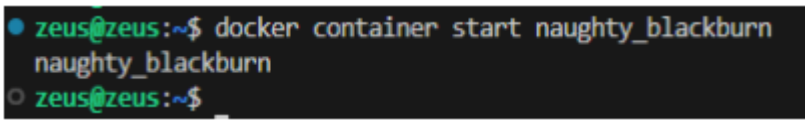
    ```
    $docker container start CONTAINER
    ```

12. Argumen CONTAINER memuat nama container atau container ID yang akan dijalankan. Perintah ini juga memiliki alias:

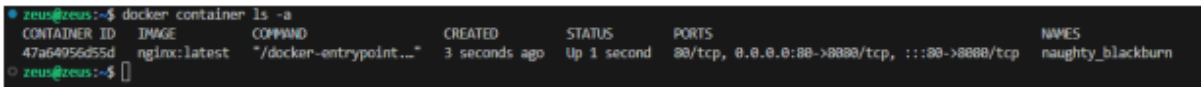
    ```
    docker start
    ```

    Sebagai contoh untuk menjalankan docker container bernama “naughty_blackburn” yang telah dibuat sebelumnya maka dapat mengeksekusi perintah:

    ```
    $ docker container start naughty_blackburn
    ```

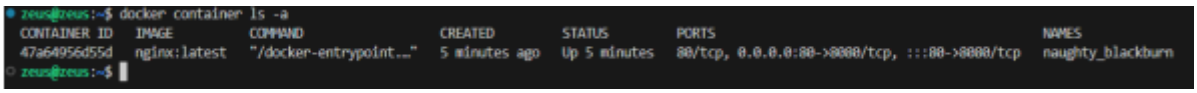


Terlihat informasi nama container yang telah berhasil dijalankan :



13. Memverifikasi docker container bernama “naughty_blackburn” telah berjalan.

    ```
    $ docker container ls
    ```



Terlihat container bernama “naughty_blackburn” 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.

14. 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 “clever_hamilton” yang telah berjalan sebelumnya maka dapat mengeksekusi perintah:

```
$ docker container stop clever_hamilton
```

```
zeus@zeus:~$
zeus@zeus:~$ docker container stop clever_hamilton
clever_hamilton
zeus@zeus:~$
```

Dapat diketahui bahwa status pada container dengan nama clever_hamilton sekarang berubah menjadi **Exited**

15. Memverifikasi container yang berhenti

```
$ docker container ls
```

```
zeus@zeus:~$ docker container ls -a
CONTAINER ID   IMAGE          COMMAND                  CREATED         STATUS                     PORTS     NAMES
429973411cef   nginx:latest   "/docker-entrypoint.…"   4 minutes ago   Created                              affectionate_tesla
6fb75d73c66d   nginx:latest   "/docker-entrypoint.…"   5 minutes ago   Exited (0) 3 seconds ago             clever_hamilton
zeus@zeus:~$
```

16. 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 “clever_hamilton” yang telah dihentikan sebelumnya maka dapat mengeksekusi perintah:

```
$ docker container rm clever_hamilton
```

Atau jika menggunakan container ID “6fb75d73c66d” dari container tersebut maka dapat mengeksekusi perintah:

```
$ docker container rm 6fb75d73c66d
```

```
zeus@zeus:~$
zeus@zeus:~$ docker container rm clever_hamilton
clever_hamilton
```

17. Memverifikasi container clever hamilton yang telah dihapus :

```
zeus@zeus:~$ docker container ls -a
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS    PORTS     NAMES
429973411cef   nginx:latest   "/docker-entrypoint.…"   14 minutes ago   Created             affectionate_tesla
zeus@zeus:~$
```

18. 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.

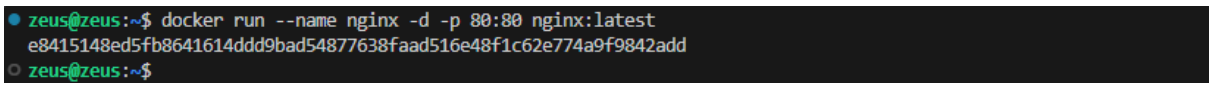
• Container image menggunakan nginx dengan tag “latest”.

• Merutekan trafik dari port 80 di mesin host ke port 80 juga di dalam container.

• Container berjalan di background.

Maka dapat mengeksekusi perintah:

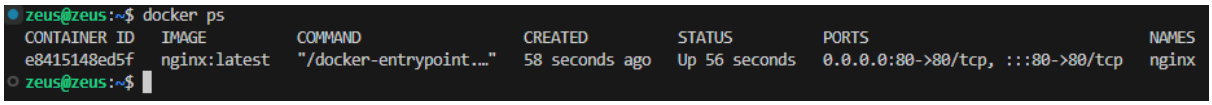
```
$ docker run --name nginx -d -p 80:80 nginx:latest
```



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

19. Memverifikasi docker container bernama “nginx” telah berjalan.

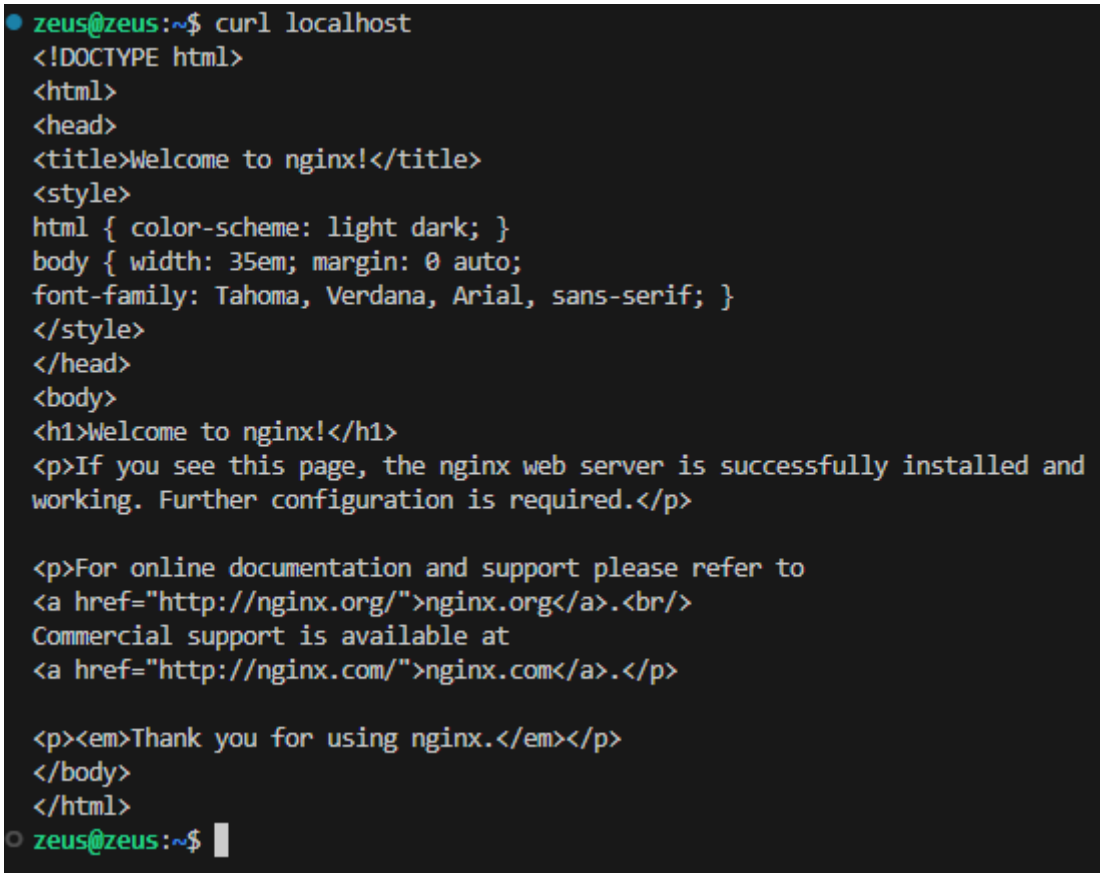
```
$ docker ps
```



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.

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

```
$ curl localhost
```



I. 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
```

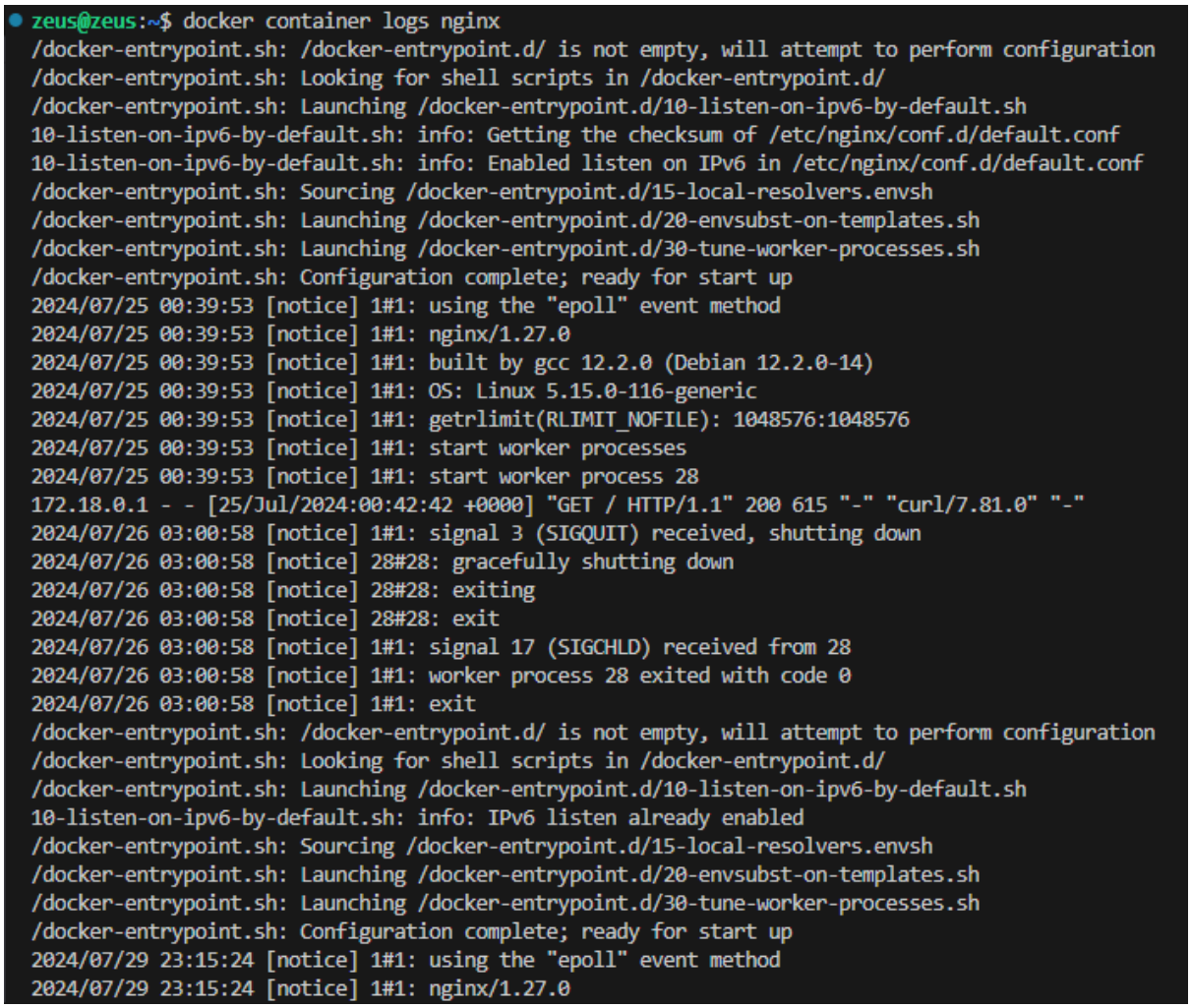
Penjelasan dari salah satu options yang umum digunakan:
-f, --follow digunakan untuk melihat log dari container secara real time.
Argumen CONTAINER memuat nama container atau container ID yang akan diakses atau diambil log-nya.
Perintah ini juga memiliki alias:

```
docker logs
```

Sebagai contoh untuk melihat logs dari container bernama nginx yang sedang berjalan maka

dapat mengeksekusi perintah:

```
$ docker container logs nginx
```

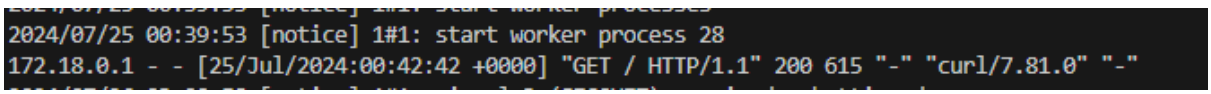


Pada bagian akhir dari output tersebut terlihat log terkait pengaksesan layanan container nginx menggunakan utilitas curl yaitu yang dieksekusi menggunakan perintah curl localhost pada bagian B di langkah 14 sebelumnya.

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

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

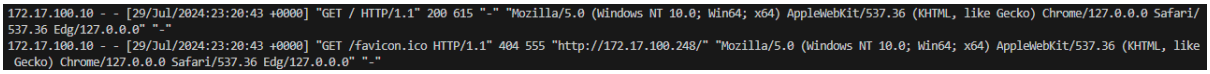
Cuplikan output dari hasil eksekusi perintah tersebut, seperti terlihat pada gambar berikut:



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 yaitu Alamat IP Ubuntu server, 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.



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



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.

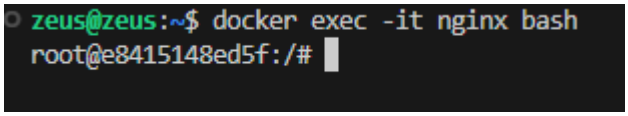
Argumen CONTAINER memuat nama *container* atau *container* ID yang akan diakses sebagai lokasi atau target eksekusi perintah. Sedangkan COMMAND [ARG…] digunakan untuk mencantumkan perintah yang akan dieksekusi dan argumen yang akan dieksekusi pada *container* yang dibuat.

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*:

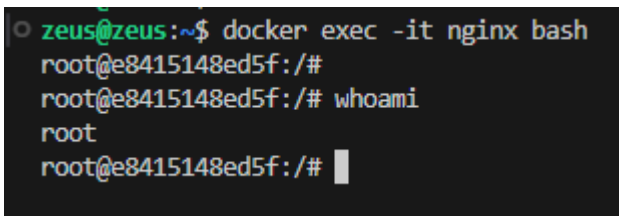
```
$ docker exec -it nginx bash
```



Terlihat telah berhasil masuk ke dalam *container* **nginx** yang ditandai dengan *prompt* **root@e8415148ed5f**

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
```

```
root@e8415148ed5f:/# whoami
root
root@e8415148ed5f:/# exit
exit
zeus@zeus:~$
```

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

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

```
zeus@zeus:~$ docker exec nginx echo Cloud Native
Cloud Native
zeus@zeus:~$
```

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

J. 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
   ```

   ```
   zeus@zeus:~$ docker network ls
   NETWORK ID     NAME      DRIVER    SCOPE
   1cc5adbbb630   bridge    bridge    local
   5e04631b13a2   host      host      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
   ```

   ```
   zeus@zeus:~$ docker network create belajar
   5f96e031b198f0e2493e96a16d6c488b70cd92df5f39dc8558b7cb96215ac80c
   zeus@zeus:~$
   ```

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

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

   ```
   $ docker network ls
   ```

   ```
   zeus@zeus:~$ docker network ls
   NETWORK ID     NAME      DRIVER    SCOPE
   5f96e031b198   belajar   bridge    local
   1cc5adbbb630   bridge    bridge    local
   ```

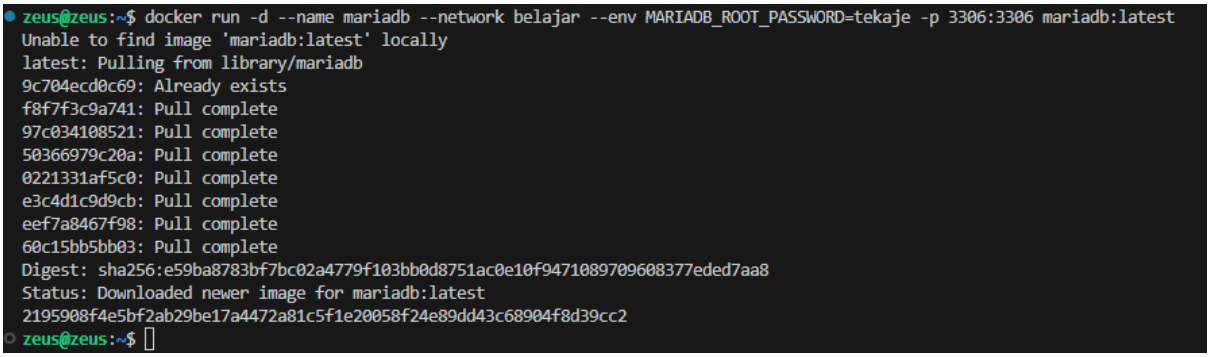
   Terlihat telah terdapat *docker network* bernama **belajar** dengan *network driver* bertipe **bridge**.

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 tekaje
- 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:

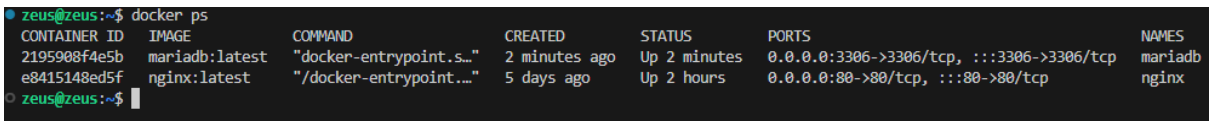
```
$ docker run -d --name mariadb --network belajar \
--env MARIADB_ROOT_PASSWORD=tekaje \
-p 3306:3306 mariadb:latest
```



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 “**2195908f4e5bf2ab29be17a4472a81c5f1e20058f24e89dd43c68904f8d39cc2**”.

5. Memverifikasi *docker container* bernama “**mariadb**” telah berjalan.

```
$ docker ps
```



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
   ```

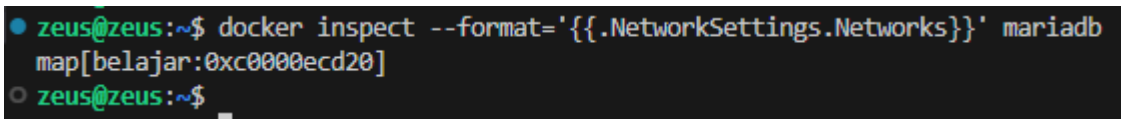
   Penjelasan dari salah satu options yang umum digunakan:

   -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
   ```

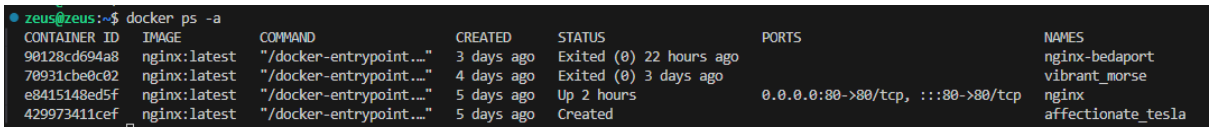
   Terlihat informasi bahwa *docker container* **mariadb** menggunakan *network* **belajar**

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

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

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

   ```
   $ docker ps -a
   ```

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
```

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

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

```
$ docker network ls
```

```
zeus@zeus:~$ docker network ls
NETWORK ID     NAME       DRIVER    SCOPE
1cc5adbbb630   bridge     bridge    local
5e04631b13a2   host       host      local
1dd818597308   net-coba   bridge    local
8843bd61e911   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.

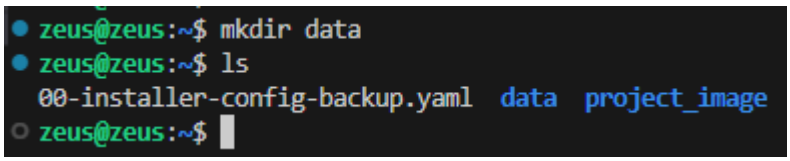
c. destination digunakan untuk menentukan lokasi direktori tujuan *mounting* di *docker container*. *Field* ini memiliki alias dst dan target.

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
   ```

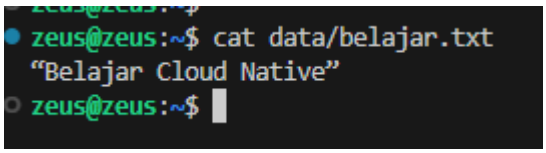
   Terlihat direktori **data** telah berhasil dibuat.

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
   ```

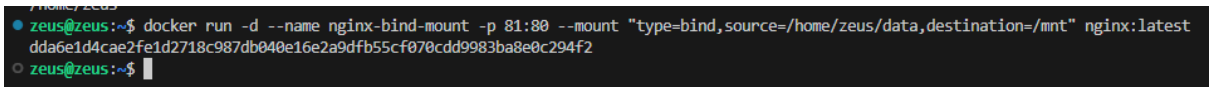
3. Memverifikasi konten dari *file* **belajar.txt** yang terdapat di dalam direktori **data**.

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

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.

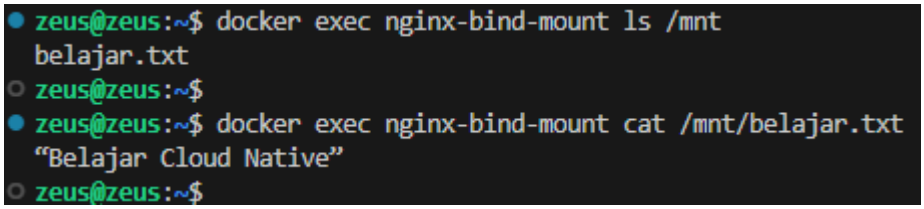
   ```
   $ docker run -d --name nginx-bind-mount -p 81:80 \
   --mount "type=bind,source=/home/belajar/data,destination=/mnt" \
   nginx:latest
   ```

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

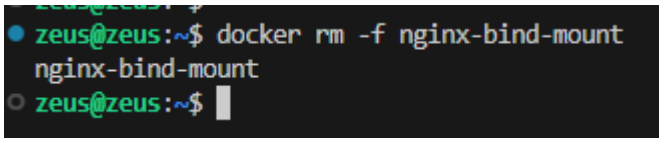
5. Memverifikasi keberhasilan *mounting* direktori dengan mengeksekusi perintah di dalam *container* untuk menampilkan isi dari direktori /**mnt**.

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



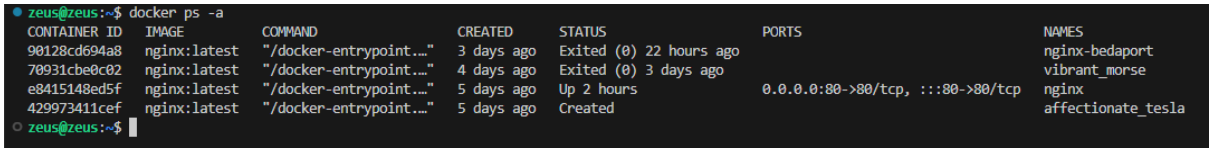
6. Menghentikan dan menghapus *docker container* bernama **nginx-bind-mount**.

```
$ docker rm -f 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**.



8. **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
   ```

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

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

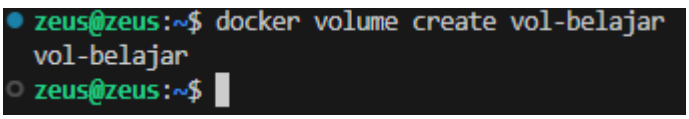
   Penjelasan salah satu *options*:

   -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.

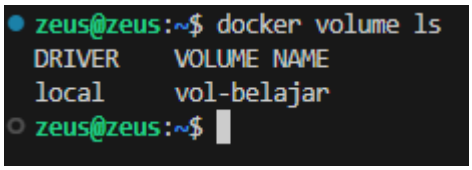
   Sebagai contoh untuk membuat *docker volume* baru bernama **vol-belajar** maka dapat mengeksekusi perintah:

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



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

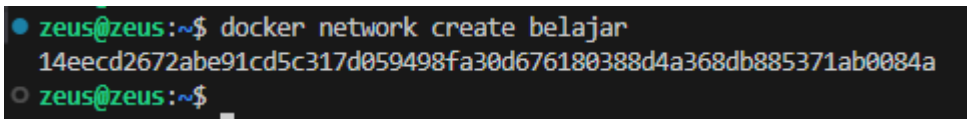
```
$ docker volume ls
```



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 tekaje
    - Merutekan trafik dari port 3306 di mesin host ke port 3306 juga di dalam container.
    - Container image menggunakan mariadb dengan tag “latest”.

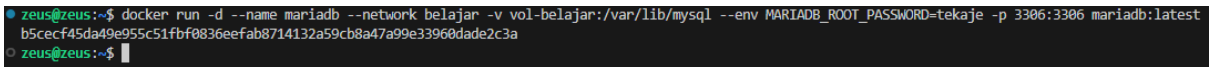
    Sebelum mengeksekusi perintah untuk membuat container dengan ketentuan tersebut maka akan dilakukan pembuatan docker network bernama belajar menggunakan perintah:

    ```
    $ 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=tekaje \
    -p 3306:3306 mariadb:latest
    ```

Terlihat informasi *container* ID dari *container* yang dibuat yaitu "**b5cecf45da49e955c51fbf0836eefab8714132a59cb8a47a99e33960dade2c3a**"

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

```
$ docker ps
```



Terlihat *container* bernama "**mariadb**" telah berhasil dibuat menggunakan *image* **mariadb:latest** pada beberapa menit yang lalu berdasarkan informasi pada kolom **CREATED** dan telah aktif 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
```

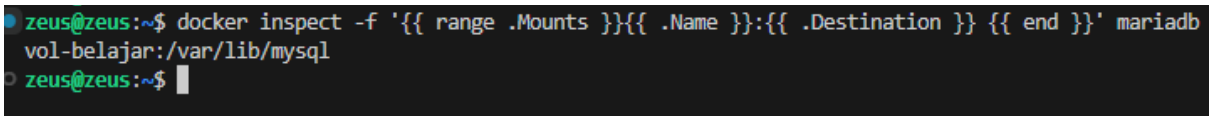
Penjelasan dari salah satu options yang umum digunakan:

-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
```



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	*Kubernetes*	*Nigel Poulton*

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

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

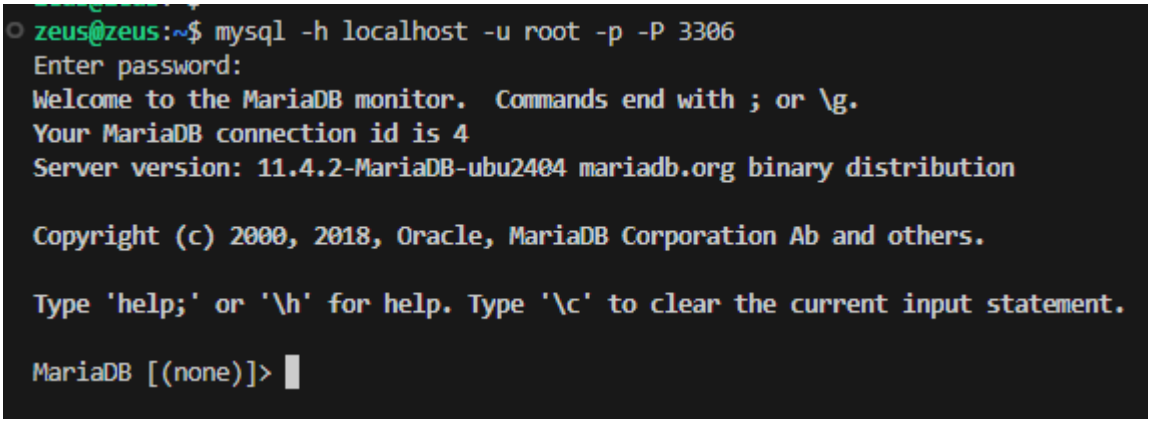
b. Mengakses *server mariadb* pada *docker container* menggunakan utilitas *mysql client* dengan parameter-parameter berikut:

- -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:

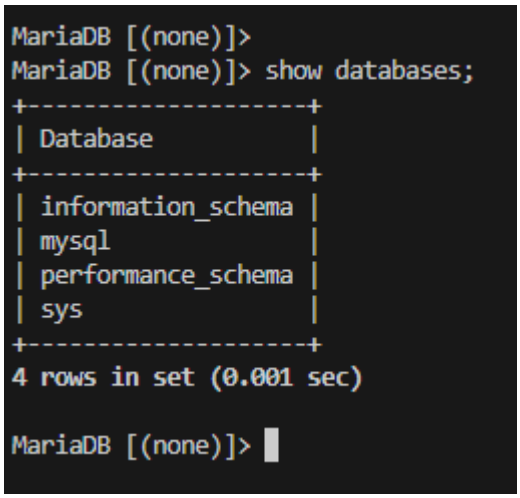
```
$ mysql -h localhost -u root -p -P 3306
```

Tampil inputan **Enter password**, masukkan tekaje & tekan **Enter**.



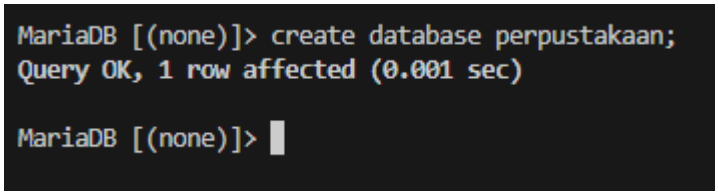
c. Menampilkan informasi *database* yang telah ada di *server mariadb*.

```
show databases;
```



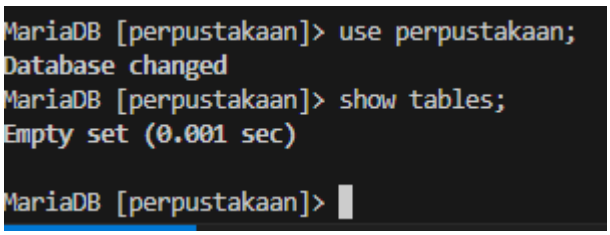
d. Membuat *database* bernama **perpustakaan**.

```
create database perpustakaan;
```



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

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



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 (0.198 sec)

MariaDB [perpustakaan]>
```

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)

MariaDB [perpustakaan]>
```

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)

MariaDB [perpustakaan]>
```

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 (0.057 sec)

MariaDB [perpustakaan]> insert into buku values ('002','The Kubernetes Book','Nigel Poulton');
Query OK, 1 row affected (0.028 sec)

MariaDB [perpustakaan]>
```

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*.

```
MariaDB [perpustakaan]> quit
Bye
zeus@zeus:~$
```

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

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

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

```
$ docker ps -a
```

10. Membuat kembali *docker container* **mariadb** yang memanfaatkan *docker volume* **volbelajar** 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=tekaje \
-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:

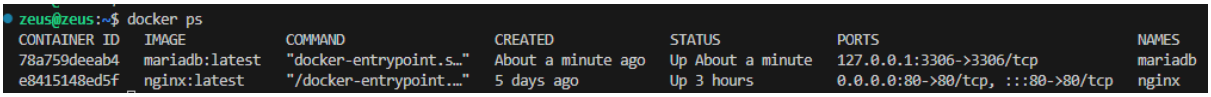
```
$ docker run -d --name mariadb --network belajar \
```

```
-v vol-belajar:/var/lib/mysql \
--env MARIADB_ROOT_PASSWORD=tekaje \
-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
```



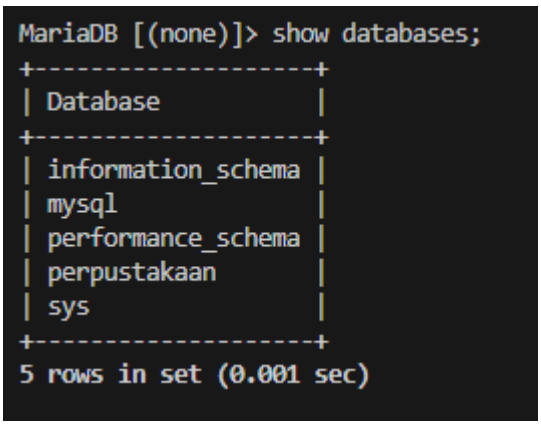
12. Mengakses server mariadb pada docker container menggunakan utilitas mysql client.

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

Tampil inputan Enter password, masukkan tekaje & tekan **Enter**.

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

```
show databases;
```



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)]> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| perpustakaan       |
| sys                |
+--------------------+
5 rows in set (0.001 sec)

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 maridb

    ```
    quit;
    ```

17. Menghentikan dan menghapus docker container mariadb.

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

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

    ```
    $ docker ps -a
    ```

    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
    ```

Docker Image

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

Komentar

INSTRUKSI argument

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:

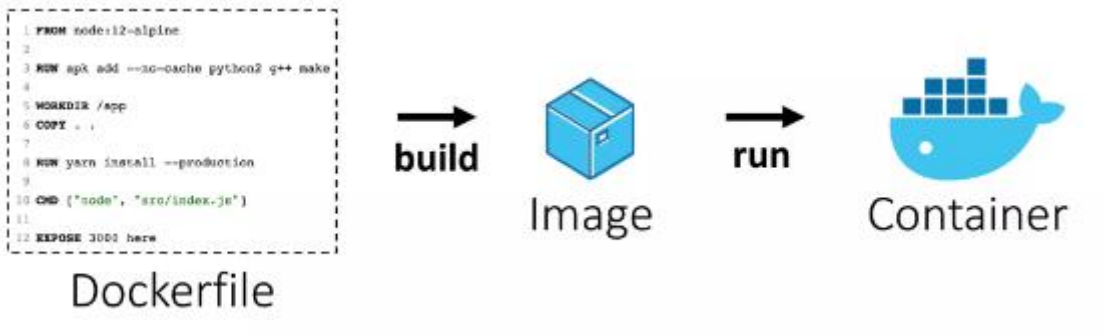
Perintah / Instruksi	**Penjelasan**
FROM	Digunakan untuk mengatur base image dari container image yang akan dihasilkan. Instruksi ini akan membuat build stage dari base image yang dicantumkan sebagai argumen dengan format penulisan image:tag
RUN	Digunakan untuk mengeksekusi perintah di dalam container 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:



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
```

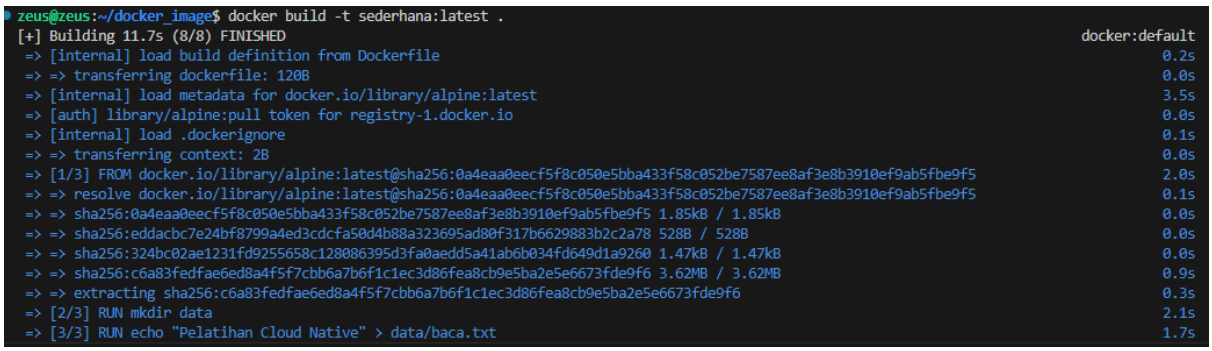
```
RUN echo "Pelatihan Cloud Native" > data/baca.txt
```

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.
   `$ docker image ls -a`

```
zeus@zeus:~/docker_image$ docker image ls -a
REPOSITORY              TAG       IMAGE ID       CREATED          SIZE
sederhana               latest    66ec8335a525   44 seconds ago   7.8MB
```

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
```

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

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:

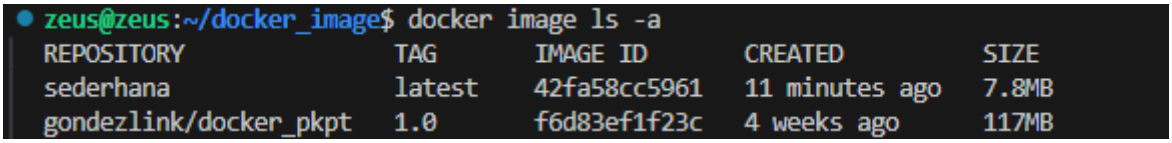
   ```
   FROM alpine:latest
   RUN mkdir data
   ```

```
RUN echo "Pelatihan Cloud Native" > data/baca.txt
CMD ["cat", "data/baca.txt"]
```

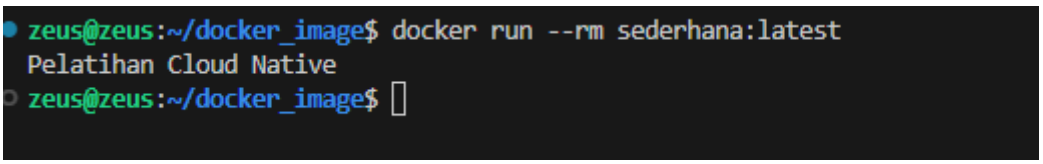
Simpan perubahan dengan menekan tombol CTRL+O dan tekan Enter. Keluar dari editor nano dengan menekan tombol CTRL+X.

2. Membuat docker container images dengan mengeksekusi perintah:
   ```
   $ docker image build -t sederhana:latest .
   ```

3. Memverifikasi hasil pembuatan container images.



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



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* "**ekonugroho@smkn22jakarta.sch.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
   ```

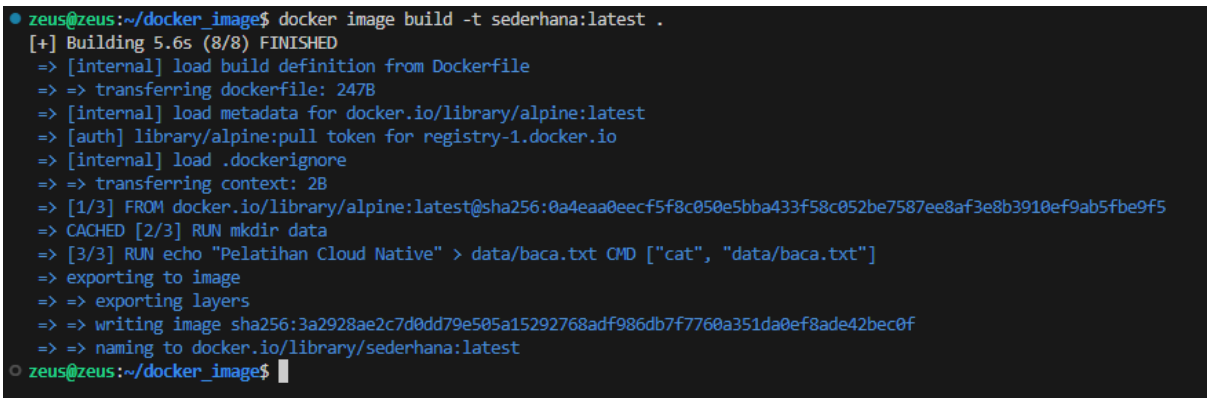
   Sebagai contoh penambahan metadata dilakukan setelah instruksi FROM sehingga hasilnya terlihat seperti berikut:

   ```
   FROM alpine:latest
   LABEL description="Belajar membuat Docker Image"
   LABEL email="ekonugroho@smkn22jakarta.shc.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 .
   ```

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

   ```
   $ docker image inspect sederhana:latest
   ```

```
zeus@zeus:~/docker_image$ docker image inspect sederhana:latest
[
    {
        "Id": "sha256:3a2928ae2c7d0dd79e505a15292768adf986db7f7760a351da0ef8ade42bec0f",
        "RepoTags": [
            "sederhana:latest"
        ],
        "RepoDigests": [],
        "Parent": "",
        "Comment": "buildkit.dockerfile.v0",
        "Created": "2024-08-08T00:42:12.635796018Z",
        "DockerVersion": "",
        "Author": "",
        "Config": {
            "Hostname": "",
            "Domainname": "",
            "User": "",
            "AttachStdin": false,
            "AttachStdout": false,
            "AttachStderr": false,
            "Tty": false,
            "OpenStdin": false,
            "StdinOnce": false,
```

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

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

```
zeus@zeus:~/docker_image$  docker image inspect sederhana:latest -f “{{.Config.Labels}}”
“map[description:Belajar membuat Docker Image email:ekonugrohc@smkn22jakarta.shc.id]”
zeus@zeus:~/docker_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
   ```

   ```
   zeus@zeus:~/project_image$ ls
    Dockerfile   dokumen  'project6 yaml'
   zeus@zeus:~/project_image$
   ```

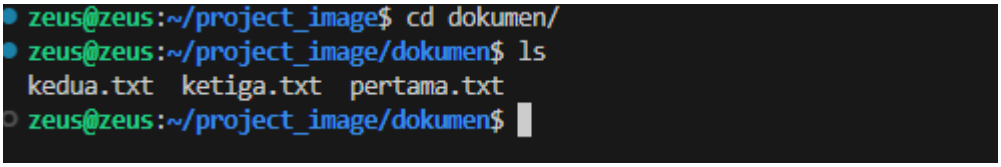
2. Membuat tiga *file* yaitu "**pertama.txt**", "**kedua.txt**", dan "**ketiga.txt**" dengan konten yang telah ditentukan sebelumnya.

   ```
   $ 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
```



4. Memverifikasi konten dari tiga *file* di dalam direktori **dokumen**.

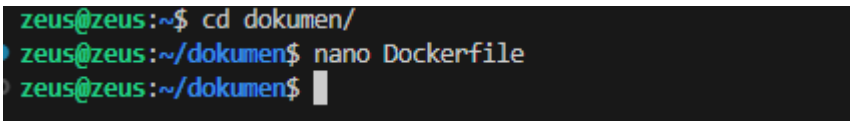
```
$ cat dokumen/pertama.txt
$ cat dokumen/kedua.txt
$ cat dokumen/ketiga.txt
```

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="ekonugroho@smkn22jakarta.sch.id"
RUN mkdir data
RUN echo "Pelatihan Cloud Native" > data/baca.txt
COPY dokumen/*.txt data
CMD ["ls", "data"]
```



```
  GNU nano 6.2                                                    Dockerfile
FROM alpine:latest
LABEL description="Belajar membuat Docker Image"
LABEL email="ekonugrohc@smkn22jakarta.sch.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 .
   ```

```
zeus@zeus:~/project_image$ docker image build -t sederhana:latest .
[+] Building 4.3s (9/9) FINISHED                                                docker:default
 => [internal] load build definition from Dockerfile                                      0.1s
 => => transferring dockerfile: 264B                                                      0.0s
 => [internal] load metadata for docker.io/library/alpine:latest                          1.0s
 => [internal] load .dockerignore                                                         0.2s
 => => transferring context: 2B                                                           0.0s
 => [1/4] FROM docker.io/library/alpine:latest@sha256:0a4eaa0eecf5f8c050e5bba433f58c052be7587ee8af3e8b3910ef9ab5fbe9f5   0.0s
 => [internal] load build context                                                         0.4s
 => => transferring context: 236B                                                         0.0s
 => CACHED [2/4] RUN mkdir data                                                           0.0s
 => CACHED [3/4] RUN echo "Pelatihan Cloud Native" > data/baca.txt                        0.0s
 => [4/4] COPY dokumen/*.txt data                                                         0.7s
 => exporting to image                                                                    0.8s
 => => exporting layers                                                                   0.5s
 => => writing image sha256:549c605cb7cee298cc05fd4c7cc568b33642e63a66118bbabd857829ae79467b   0.0s
 => => naming to docker.io/library/sederhana:latest                                       0.1s
zeus@zeus:~/project_image$
```

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
   ```

```
zeus@zeus:~/project_image$ docker run --rm sederhana:latest
baca.txt
kedua.txt
ketiga.txt
pertama.txt
zeus@zeus:~/project_image$
```

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
```

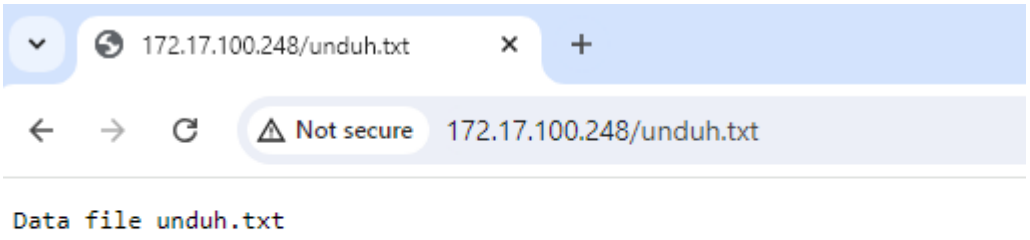
```
zeus@zeus:~/project_image$ docker container start 56c1d036b4cb
56c1d036b4cb
zeus@zeus:~/project_image$ docker container ls -a
CONTAINER ID   IMAGE           COMMAND                  CREATED       STATUS            PORTS
                  NAMES
707db7e37d4f   mariadb:latest   "docker-entrypoint.s…"   3 days ago    Up About an hour   0.0.0.0:3306->3306/tcp,
 :::3306->3306/tcp   mariadb
56c1d036b4cb   nginx:latest     "/docker-entrypoint.…"   10 days ago   Up 3 seconds       0.0.0.0:80->80/tcp, :::
80->80/tcp          nginx
zeus@zeus:~/project_image$
```

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"
   ```

```
zeus@zeus:~/project_image$ docker exec nginx bash -c "echo 'Data file unduh.txt' > /usr/share/nginx/html/unduh.tx
t"
zeus@zeus:~/project_image$
```

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



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:

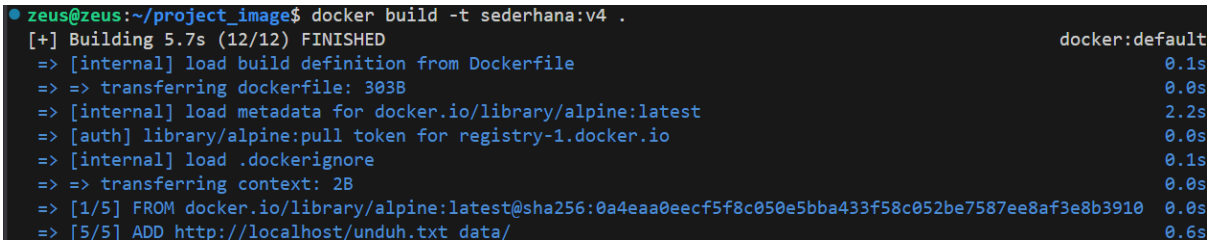
```
FROM alpine:latest
LABEL description="Belajar membuat Docker Image"
LABEL email=ekonugroho@smkn22jakarta.sch.id
RUN mkdir data
RUN echo "Pelatihan Cloud Native" > data/baca.txt
COPY dokumen/*.txt data
```

```
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:

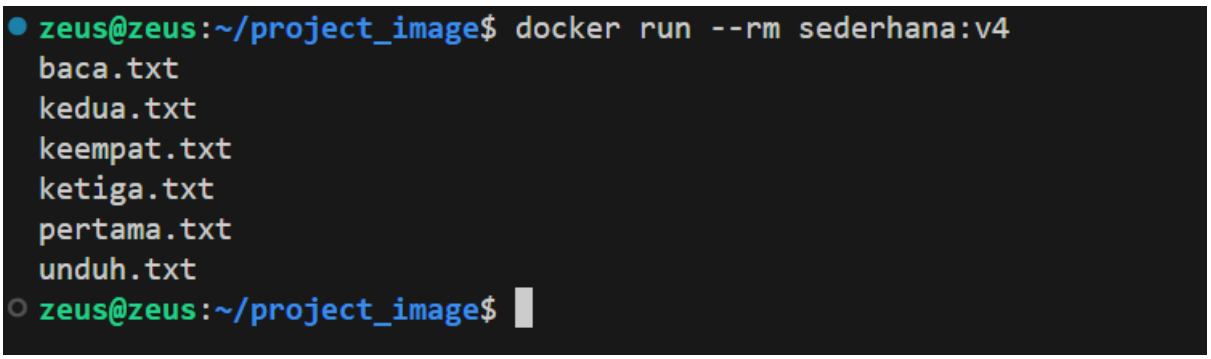
```
$ docker image build -t sederhana:latest .
```



6. Memverifikasi hasil pembuatan *container images*.

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
```



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:

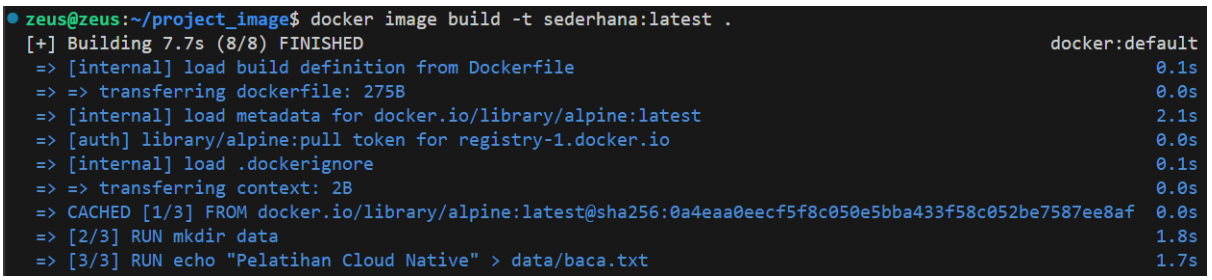
```
FROM alpine:latest
LABEL description="Belajar membuat Docker Image"
LABEL email="ekonugroho@smkn22jakarta.sch.id"
ARG konten="Pelatihan Cloud Native"
RUN mkdir data
RUN echo "$konten" > data/baca.txt
CMD ["cat", "data/baca.txt"]
```

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.

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 .
   ```

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



3. Memverifikasi hasil pembuatan *container images*.

```
zeus@zeus:~/project_image$ docker image ls -a
REPOSITORY    TAG       IMAGE ID       CREATED          SIZE
sederhana     latest    d0aefcef481c   38 seconds ago   7.8MB
nginx-web     v1        4a5a3ad78a22   12 days ago      199MB
nginx         latest    a72860cb95fd   2 months ago     188MB
mariadb       latest    4486d64c9c3b   2 months ago     406MB
```

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
   ```

   ```
   zeus@zeus:~/project_image$ 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:1.0 .
   ```

   ```
   zeus@zeus:~/project_image$ docker build --build-arg konten="Selamat Belajar Cloud Native" -t sederhana:1.0 .
   [+] Building 7.0s (8/8) FINISHED                                                                    docker:default
    => [internal] load build definition from Dockerfile                                                        0.1s
    => => transferring dockerfile: 275B                                                                        0.0s
    => [internal] load metadata for docker.io/library/alpine:latest                                            2.1s
    => [auth] library/alpine:pull token for registry-1.docker.io                                               0.0s
    => [internal] load .dockerignore                                                                           0.1s
    => => transferring context: 2B                                                                             0.0s
    => CACHED [1/3] FROM docker.io/library/alpine:latest@sha256:0a4eaa0eecf5f8c050e5bba433f58c052be7587ee8af3  0.0s
    => [2/3] RUN mkdir data                                                                                    1.4s
    => [3/3] RUN echo "Selamat Belajar Cloud Native" > data/baca.txt                                           1.5s
   ```

   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*

   ```
   zeus@zeus:~/project_image$ docker image ls -a
   REPOSITORY    TAG       IMAGE ID       CREATED              SIZE
   sederhana     1.0       0a6a639710e5   About a minute ago   7.8MB
   ```

7. Ujicoba membuat dan menjalankan *container* menggunakan *images* **sederhana:1.0**.

   Selain itu apabila *container* dihentikan maka akan langsung menghapus *container* dengan memanfaatkan option `--rm` (*remove*) dari perintah `docker run`.

   ```
   $ docker run --rm sederhana:1.0
   ```

   ```
   zeus@zeus:~/project_image$ docker run --rm sederhana:1.0
   Selamat Belajar Cloud Native
   zeus@zeus:~/project_image$
   ```

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=" ekonugroho@smkn22jakarta.sch.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**.

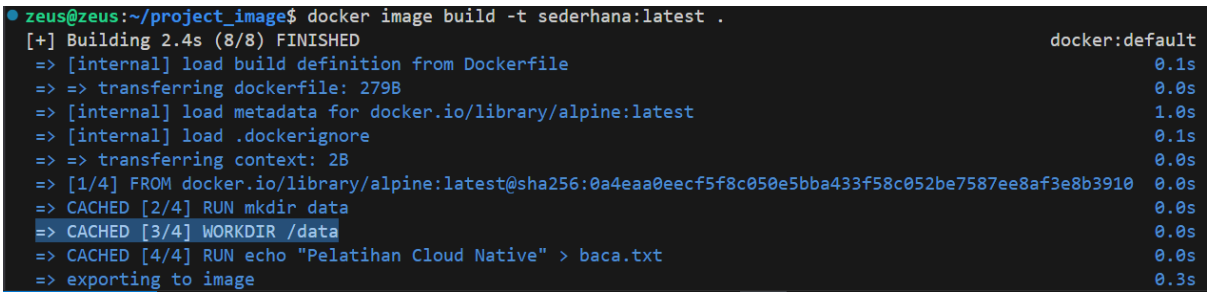
   ```
   $ nano Dockerfile
   ```

   ```
   FROM alpine:latest
   LABEL description="Belajar membuat Docker Image"
   LABEL email="ekonugroho@smkn22jakarta.sch.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 .
   ```

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

3. Memverifikasi hasil pembuatan *container images*.

   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
   ```

   ```
   zeus@zeus:~/project_image$ docker run --rm sederhana:latest
   Pelatihan Cloud Native
   zeus@zeus:~/project_image$
   ```

   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* "**Fery**" maka deklarasinya akan terlihat seperti berikut:

```
ENV NAMA=”Fery”
```

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
   ```

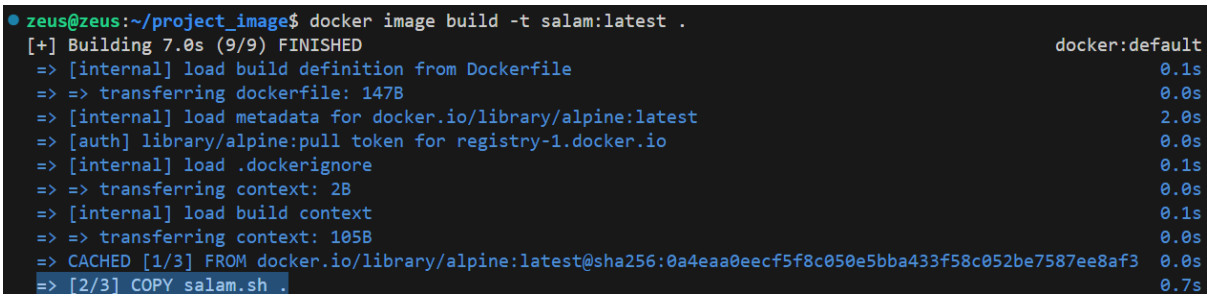
   Konten dari *file* akan terlihat seperti berikut:

   ```
   FROM alpine:latest
   ENV NAMA="Fernandez eko"
   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* "**Fery**". 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.

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

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



4. Memverifikasi hasil pembuatan *container images*.

```
zeus@zeus:~/project_image$ docker image ls -a
REPOSITORY    TAG       IMAGE ID       CREATED          SIZE
salam         latest    7173db902168   47 seconds ago   7.8MB
sederhana     latest    a6338fc9c425   35 minutes ago   7.8MB
nginx-web     v1        4a5a3ad78a22   12 days ago      199MB
nginx         latest    a72860cb95fd   2 months ago     188MB
mariadb       latest    4486d64c9c3b   2 months ago     406MB
zeus@zeus:~/project_image$
```

5. 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
   ```

   ```
   zeus@zeus:~/project_image$ docker run --rm salam:latest
   Selamat belajar Cloud Native, rekan Fernandez eko. Semangat!
   zeus@zeus:~/project_image$
   ```

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

6. 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:

   ```
           "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=Fernandez eko"
           ],
           "Cmd": [
               "/salam.sh"
   ```

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

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

```
zeus@zeus:~/project_image$ docker image inspect salam:latest -f "{{.Config.Env}}"
"[PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin NAMA=Fernandez eko]"
zeus@zeus:~/project_image$
```

Terlihat *output* environment variable **NAMA** dengan nilai default "**Fernandez eko**" pada *container image*.

7. 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
   ```

   ```
   zeus@zeus:~/project_image$ docker run --rm --env NAMA='Linux Torvalds' salam:latest
   Selamat belajar Cloud Native, rekan Linux Torvalds. Semangat!
   zeus@zeus:~/project_image$
   ```

   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 .
```



3. Memverifikasi hasil pembuatan *container images*.

```
zeus@zeus:~/project_image$ docker image ls -a
REPOSITORY          TAG       IMAGE ID       CREATED          SIZE
ubuntu-with-ping    latest    24d8325cfabe   11 seconds ago   119MB
salam               latest    7173db902168   16 minutes ago   7.8MB
sederhana           latest    a6338fc9c425   51 minutes ago   7.8MB
nginx-web           v1        4a5a3ad78a22   12 days ago      199MB
nginx               latest    a72860cb95fd   2 months ago     188MB
mariadb             latest    4486d64c9c3b   2 months ago     406MB
```

4. Ujicoba membuat dan menjalankan *container* menggunakan *images* **ubuntu-withping: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
```

```
zeus@zeus:~/project_image$ docker run --rm ubuntu-with-ping:latest
PING google.com (142.250.4.100) 56(84) bytes of data.
64 bytes from sm-in-f100.1e100.net (142.250.4.100): icmp_seq=1 ttl=107 time=16.8 ms
64 bytes from sm-in-f100.1e100.net (142.250.4.100): icmp_seq=2 ttl=107 time=16.8 ms
64 bytes from sm-in-f100.1e100.net (142.250.4.100): icmp_seq=3 ttl=107 time=16.9 ms
64 bytes from sm-in-f100.1e100.net (142.250.4.100): icmp_seq=4 ttl=107 time=16.8 ms

--- google.com ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3005ms
rtt min/avg/max/mdev = 16.751/16.827/16.904/0.054 ms
zeus@zeus:~/project_image$
```

5. Ujicoba membuat dan menjalankan *container* kembali menggunakan *images* **ubuntu-withping: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
```

```
zeus@zeus:~/project_image$ docker run --rm ubuntu-with-ping:latest -c 2 detik.com
PING detik.com (103.49.221.211) 56(84) bytes of data.
64 bytes from s211-cast-211-221-49-103.detik.com (103.49.221.211): icmp_seq=1 ttl=57 time=2.54 ms
64 bytes from s211-cast-211-221-49-103.detik.com (103.49.221.211): icmp_seq=2 ttl=57 time=2.37 ms

--- detik.com ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms
rtt min/avg/max/mdev = 2.372/2.454/2.536/0.082 ms
zeus@zeus:~/project_image$
```

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
```



```
        ],
        "Cmd": [
            "-c",
            "4",
            "google.com"
        ],
        "ArgsEscaped": true,
        "Image": "",
        "Volumes": null,
        "WorkingDir": "",
        "Entrypoint": [
            "ping"
```

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}}"
```

```
zeus@zeus:~/project_image$ docker inspect ubuntu-with-ping:latest -f "{{.Config.Entrypoint}} {{.Config.Cmd}}"
[ping] [-c 4 google.com]
zeus@zeus:~/project_image$
```

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 ada **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
   ```

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
   ```

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

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

   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 .
   ```

6. Memverifikasi hasil pembuatan *container images*.

7. Terlihat *container image* tersebut telah berhasil dibuat.

8. 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:

   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}}"
   ```

   Terlihat *output* nilai dari **ExposedPorts** berupa **8080/tcp** pada *container image*.

9. Ujicoba membuat dan menjalankan *container* menggunakan *images* **nginx-custom:latest**.

```
$ docker run --name nginx-custom -d -p 8080:8080 \
nginxcustom:latest
```

10. Memverifikasi *docker container* bernama "**nginx-custom**" telah berjalan.
    ```
    $ docker ps
    ```
    Terlihat *container* tersebut telah berjalan berdasarkan informasi pada kolom **STATUS** bernilai **Up 4 seconds.**

11. Memverifikasi akses ke layanan *web* dari *container* **nginx-custom** pada **port 8080** menggunakan aplikasi **curl**.
    ```
    $ curl localhost:8080
    ```

K. Volume Dockerfile

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 .
   ```
3. Memverifikasi hasil pembuatan *container images*.
   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:
   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}}"
   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
   ```
   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
   ```

   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
8. Memverifikasi hasil pembuatan *file* “baru.txt”

   ```
   $ ls
   $ cat baru.txt
   ```
9. Keluar dari penggunaan container berjalan dengan mengeksekusi perintah **exit**.

   ```
   # exit
   ```
10. Ujicoba membuat dan menjalankan kembali container menggunakan images alpinevolume: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
    ```

    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
    ```

    Terlihat masih terdapat *file* **baru.txt** yang dibuat.
12. Menampilkan konten dari *file* **baru.txt**.

    ```
    $ cat baru.txt
    ```
13. Keluar dari penggunaan container berjalan dengan mengeksekusi perintah **exit**.

    ```
    # exit
    ```

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:

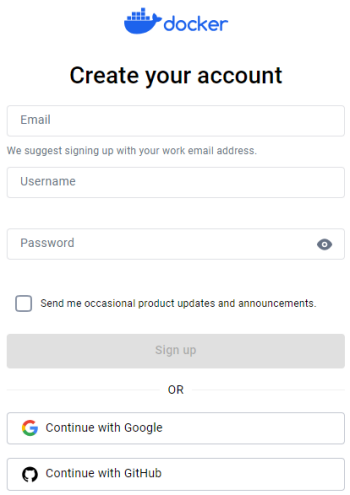
a. Repositories untuk mengunggah dan mengunduh *container images*,

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*

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/



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.

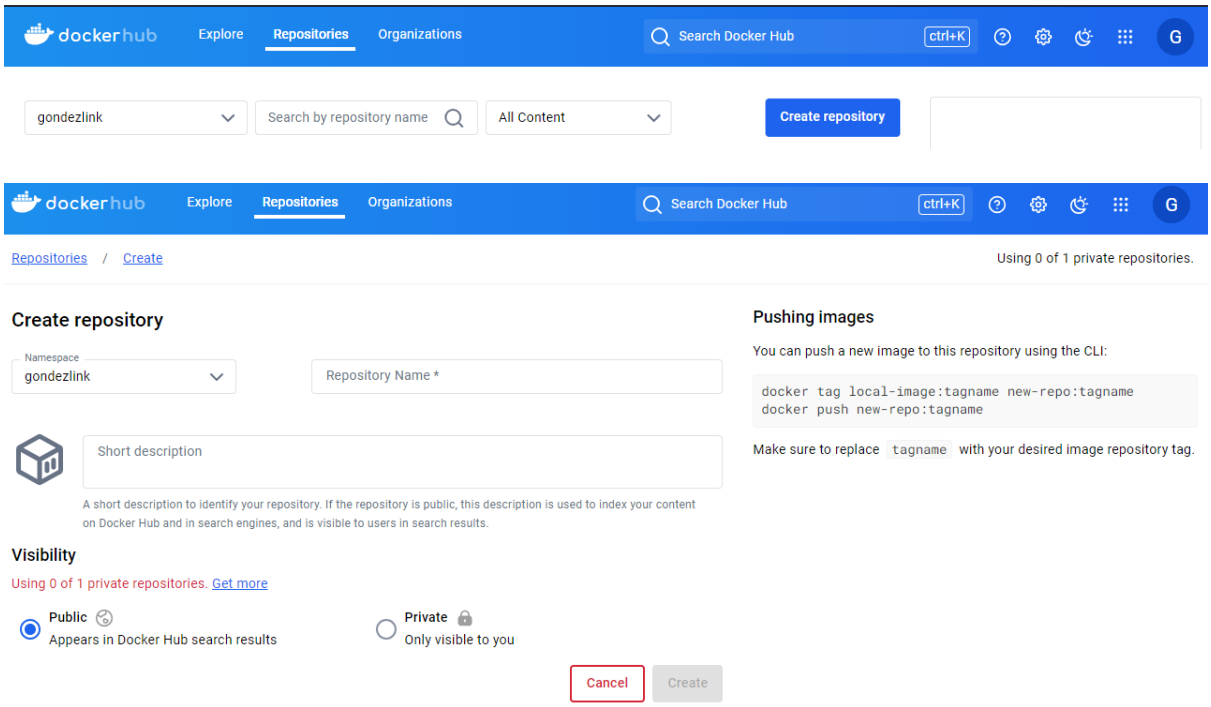
Untuk melakukan *login* ke **Docker Hub** maka dapat mengakses alamat https://hub.docker.com/login atau mengakses https://hub.docker.com dan menekan tombol **Sign In**, seperti terlihat pada gambar berikut:

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:

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**".

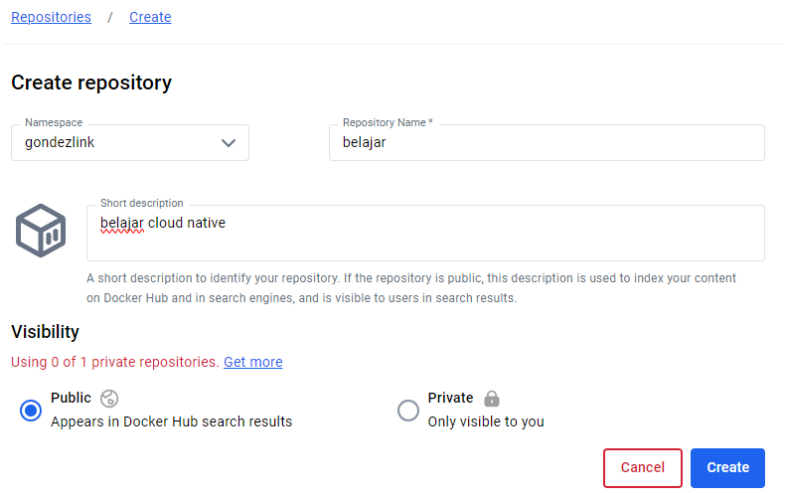
b) **Short Description** digunakan untuk memberikan keterangan singkat terkait repository yang dibuat. Sebagai contoh "**belajar cloud native**".

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:

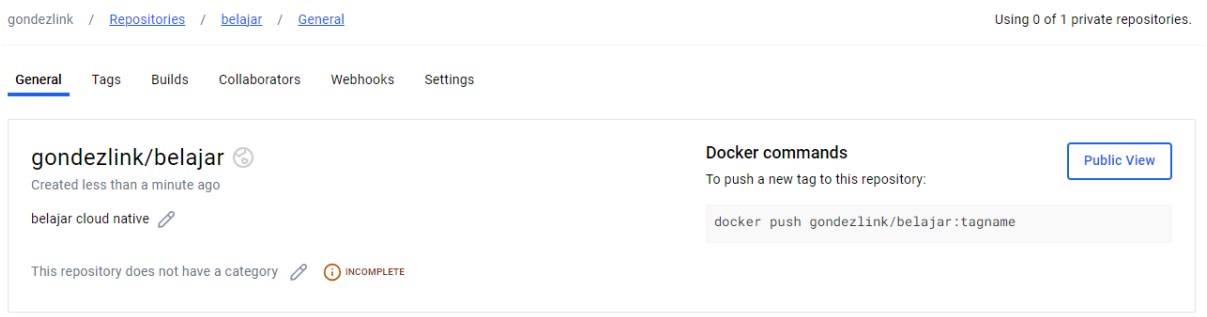
```
docker tag local-image:tagname new-repo:tagname
```

```
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
```

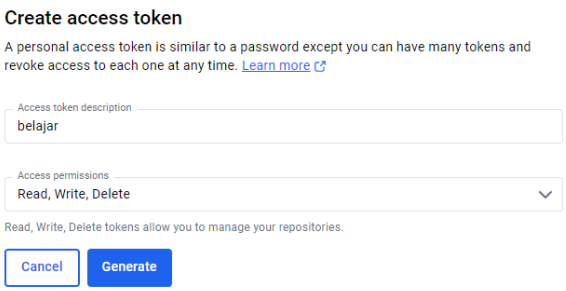
Membuat Access Token di Docker Hub

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:



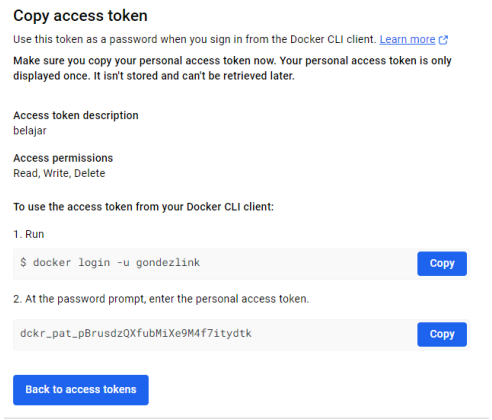
Selanjutnya akan tampil halaman yang menampilkan daftar **Access Tokens**. Tekan tombol **New Access Token** untuk membuat *access token* baru, seperti terlihat pada gambar berikut:

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**.

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

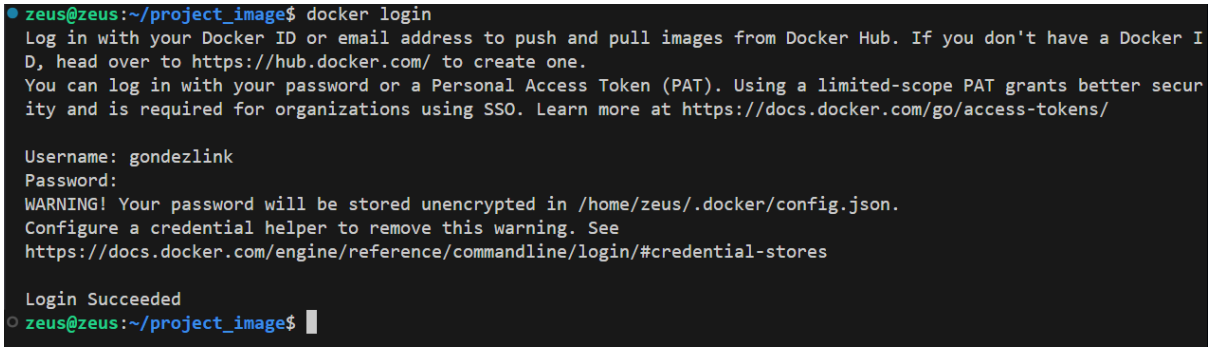
Sebagai contoh akan dilakukan pengunggahan *container images* yang telah dibuat sebelumnya yaitu dengan nama **sederhana:latest**. Adapun langkah-langkah untuk mengunggah *container images* tersebut ke **Docker Hub** adalah sebagai berikut:

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.. **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*:

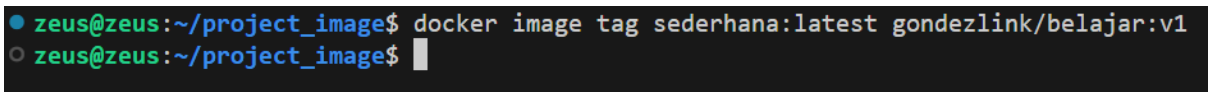
```
docker tag
```

Sesuai dengan contoh maka akan dibuat tag untuk *container images* sumber Bernama **sederhana:latest** sebagai gondezlink/**belajar:v1** dengan mengeksekusi perintah:

```
$ docker image tag sederhana:latest gondezlink/belajar:v1
```

atau

```
$ docker tag sederhana:latest gondezlink/belajar:v1
```



Silakan menyesuaikan nilai *username* "**gondezlink**" 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
   ```

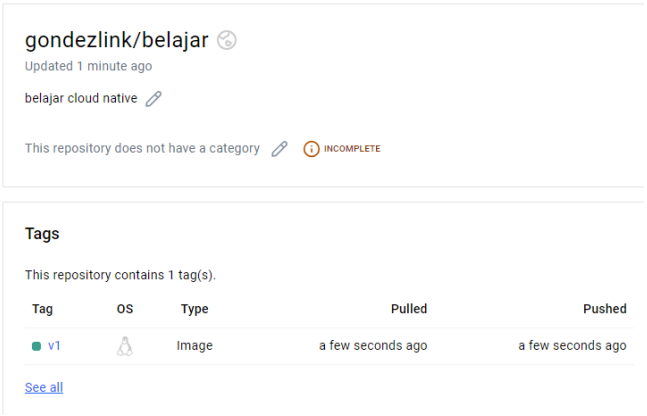
Terlihat telah terdapat *target container images* bernama gondezlink/**belajar:v1**.

4. Mengunggah *container images* bernama **gondezlink/belajar:v1** ke *Docker Hub Repository*.

   ```
   $ docker push gondezlink/belajar:v1
   ```

```
zeus@zeus:~/project_image$ docker push gondezlink/belajar:v1
The push refers to repository [docker.io/gondezlink/belajar]
f357048c08d7: Pushed
6ecd72494f11: Pushed
35b870a65292: Pushed
c40a96680a02: Pushed
78561cef0761: Mounted from library/postgres
v1: digest: sha256:8da2563a697d07c49c7f15c459878c70d217cb7ad69358b7b9692774b275283e size: 1356
zeus@zeus:~/project_image$
```

5. Memverifikasi hasil pengunggahan *container images* tersebut di **Docker Hub Repository** "**belajar**", seperti terlihat pada gambar berikut:



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:

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* maka dapat mengeksekusi perintah:

```
$ docker compose version
```

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
   ```

6. Memverifikasi *container* telah berhasil dibuat.
7. Menjalankan *service*.

   ```
   $ docker compose start
   ```

8. Memverifikasi *container* telah berhasil berjalan.
9. Memverifikasi akses ke layanan *web* dari *container* **nginx-contoh** pada **port 80** menggunakan aplikasi **curl**.

   ```
   $ curl localhost
   ```

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**
11. Menghentikan *service* sehingga *container* yang berjalan akan dihentikan namun tanpa menghapusnya.

    ```
    $ docker compose stop
    ```

12. Memverifikasi *container* telah berhasil dihentikan.

    ```
    $ docker compose ps -a
    ```

    Terlihat terdapat *container* **nginx-contoh** dengan status telah berhenti 8 detik yang lalu (**Exited (0) 8 seconds ago**).
13. Menghapus *service container* yang telah dihentikan.

    ```
    $ docker compose rm
    ```

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
   ```

6. Memverifikasi hasil *containers* telah berjalan.
   ```
   $ docker compose ps
   ```
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**

8. Menghentikan *containers*.
   ```
   $ docker compose down
   ```
9. Memverifikasi hasil penghentian *containers*.
   ```
   $ docker compose ps -a
   ```

   Terlihat keseluruhan *container* tersebut telah berhasil dihentikan dan dihapus.

10. Membuat dan menjalankan kembali *containers services* di *background*.

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

11. Memverifikasi hasil *containers* telah berjalan.

    ```
    $ docker compose ps
    ```

12. 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**.

Menjalankan aplikasi Nodejs menggunakan docker

1. Menyiapkan directory aplikasi dan aplikasi pendukung nodejs, buat sebuah folder node_project

   ```
   $ mkdir node_project
   ```

2. Berpindah atau masuk ke directory node_project yang telah dibuat sebelumnya

   ```
   $ cd node_project
   ```

3. Membuat sebuah file package.json Dimana file ini akan digunakan untuk memberikan informasi dari aplikasi nodejs yang akan digunakan

   ```
   $ nano package.json
   ```

   Isikan pada file package.json sebagai berikut :

   ```
   {
     "name": "nodejs-image-demo",
     "version": "1.0.0",
     "description": "nodejs project docker",
     "author": "gondezlink <gondezlink@gmail.com>",
     "license": "MIT",
     "main": "app.js",
     "keywords": [
       "nodejs",
       "bootstrap",
       "express"
     ],
   ```

```
  "dependencies": {
    "express": "^4.16.4"
  }
}
```

Pada file konfigurasi di atas bagian main merupakan file dari aplikasi nodejs yang akan dijalakan, sedangan pada bagan dependencies adalah versi dari framework nodejs yang digunakan.

4. Kemudian install dependencies dari nodejs yang digunakan dengan perintah

   ```
   $ npm install
   ```

5. Buatlah sebuah file app.js yang akan digunakan sebagai entrypoint aplikasi nodejs

   ```
   $ nano app.js
   ```

   Isikan dengan kode program di bawah ini :

   ```
   const express = require('express');
   const app = express();
   const router = express.Router();
   const path = __dirname + '/views/';
   const port = 8080;
   router.use(function (req,res,next) {
     console.log('/' + req.method);
     next();
   });
   router.get('/', function(req,res){
     res.sendFile(path + 'index.html');
   });
   router.get('/sharks', function(req,res){
     res.sendFile(path + 'sharks.html');
   });
   app.use(express.static(path));
   app.use('/', router);
   app.listen(port, function () {
     console.log('Example app listening on port 8080!')
   })
   ```

6. Membuat folder views dan membuat file index.html, yang digunakan untuk menampilkan aplikasi di browser.

```
$ mkdir views
```

Membuat file index.html di dalam folder views

```
$ nano views/index.html
<!DOCTYPE html>
<html lang="en">

<head>
    <title>About Sharks</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1">
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/b
ootstrap.min.css" integrity="sha384-
MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLP
MO" crossorigin="anonymous">
    <link href="css/styles.css" rel="stylesheet">
    <link
href="https://fonts.googleapis.com/css?family=Merriweather:400
,700" rel="stylesheet" type="text/css">
</head>

<body>
    <nav class="navbar navbar-dark bg-dark navbar-static-top
navbar-expand-md">
        <div class="container">
            <button type="button" class="navbar-toggler
collapsed" data-toggle="collapse" data-target="#bs-example-
navbar-collapse-1" aria-expanded="false"> <span class="sr-
only">Toggle navigation</span>
```

```
            </button> <a class="navbar-brand"
href="#">Everything Sharks</a>
            <div class="collapse navbar-collapse" id="bs-
example-navbar-collapse-1">
                <ul class="nav navbar-nav mr-auto">
                    <li class="active nav-item"><a href="/"
class="nav-link">Home</a>
                    </li>
                    <li class="nav-item"><a href="/sharks"
class="nav-link">Sharks</a>
                    </li>
                </ul>
            </div>
        </div>
    </nav>
    <div class="jumbotron">
        <div class="container">
            <h1>Want to Learn About Sharks?</h1>
            <p>Are you ready to learn about sharks?</p>
            <br>
            <p><a class="btn btn-primary btn-lg"
href="/sharks" role="button">Get Shark Info</a>
            </p>
        </div>
    </div>
    <div class="container">
        <div class="row">
            <div class="col-lg-6">
                <h3>Not all sharks are alike</h3>
                <p>Though some are dangerous, sharks generally
do not attack humans. Out of the 500 species known to
researchers, only 30 have been known to attack humans.
                </p>
```

```
            </div>
            <div class="col-lg-6">
                <h3>Sharks are ancient</h3>
                <p>There is evidence to suggest that sharks
lived up to 400 million years ago.
                </p>
            </div>
        </div>
    </div>
</body>

</html>
```

7. Membuat file shark.html pada folder views sebagai halaman informasi website

   ```
   $ nano views/sharks.html
   ```

   Isikan file sharks.html dengan kode program di bawah ini :

   ```
   <!DOCTYPE html>
   <html lang="en">

   <head>
       <title>About Sharks</title>
       <meta charset="utf-8">
       <meta name="viewport" content="width=device-width,
   initial-scale=1">
       <link rel="stylesheet"
   href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/b
   ootstrap.min.css" integrity="sha384-
   MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLP
   MO" crossorigin="anonymous">
       <link href="css/styles.css" rel="stylesheet">
       <link
   href="https://fonts.googleapis.com/css?family=Merriweather:400
   ,700" rel="stylesheet" type="text/css">
   ```

```
</head>
<nav class="navbar navbar-dark bg-dark navbar-static-top
navbar-expand-md">
    <div class="container">
        <button type="button" class="navbar-toggler collapsed"
data-toggle="collapse" data-target="#bs-example-navbar-
collapse-1" aria-expanded="false"> <span class="sr-
only">Toggle navigation</span>
        </button> <a class="navbar-brand" href="/">Everything
Sharks</a>
        <div class="collapse navbar-collapse" id="bs-example-
navbar-collapse-1">
            <ul class="nav navbar-nav mr-auto">
                <li class="nav-item"><a href="/" class="nav-
link">Home</a>
                </li>
                <li class="active nav-item"><a href="/sharks"
class="nav-link">Sharks</a>
                </li>
            </ul>
        </div>
    </div>
</nav>
<div class="jumbotron text-center">
    <h1>Shark Info</h1>
</div>
<div class="container">
    <div class="row">
        <div class="col-lg-6">
            <p>
                <div class="caption">Some sharks are known to
be dangerous to humans, though many more are not. The
sawshark, for example, is not considered a threat to humans.
```

```
                </div>
                <img
src="https://assets.digitalocean.com/articles/docker_node_imag
e/sawshark.jpg" alt="Sawshark">
            </p>
        </div>
        <div class="col-lg-6">
            <p>
                <div class="caption">Other sharks are known to
be friendly and welcoming!</div>
                <img
src="https://assets.digitalocean.com/articles/docker_node_imag
e/sammy.png" alt="Sammy the Shark">
            </p>
        </div>
    </div>
</div>
</html>
```

8. Membuat folder css untuk halaman website

   ```
   $ mkdir views/css
   ```

9. Membuat file styles.css pada folder views/css

   ```
   $ nano views/css/styles.css
   ```

   Isikan dengan program di bawah ini :

   ```
   .navbar {
       margin-bottom: 0;
   }

   body {
       background: #020A1B;
       color: #ffffff;
       font-family: 'Merriweather', sans-serif;
   }
   ```

```
h1,
h2 {
    font-weight: bold;
}

p {
    font-size: 16px;
    color: #ffffff;
}

.jumbotron {
    background: #0048CD;
    color: white;
    text-align: center;
}

.jumbotron p {
    color: white;
    font-size: 26px;
}

.btn-primary {
    color: #fff;
    text-color: #000000;
    border-color: white;
    margin-bottom: 5px;
}

img,
video,
audio {
    margin-top: 20px;
    max-width: 80%;
```

```
}

div.caption: {
    float: left;
    clear: both;
}
```

10. Pindah ke root directory aplikasi yaitu node_project

    ```
    $ cd ~/node_project
    ```

11. Jalankan aplikasi nodejs dengan perintah :

    ```
    $ node app.js
    ```

12. Lakukan uji coba aplikasi yang terlah dibuat melalui browser :

    ```
    http://your_server_ip:8080
    ```