

Docker Container Fundamental & Basic Docker Orchestration Part 2

Detail Materi



Konsep Networking Docker

Indikator :
Memahami konsep Networking Docker,
dapat menggunakan dockerhub dan
melakukan build image dengan Dockerfile



Konfigurasi Docker CLI Networking



Mengenal Dockerhub dan
Manage Image Dockerhub



Push Dockerhub



Build Image dengan Dockerfile

Modul Sekolah DevOps Cilsy

Hak Cipta © 2020 **PT. Cilsy Fiolution Indonesia**

Hak Cipta dilindungi Undang-Undang. Dilarang memperbanyak atau memindahkan sebagian atau seluruh isi buku ini dalam bentuk apapun, baik secara elektronik maupun mekanis, termasuk mecropy, merekam atau dengan sistem penyimpanan lainnya, tanpa izin tertulis dari Penulis dan Penerbit.

Penulis : Adi Saputra, Irfan Herfiandana, Tresna Widiyaman, Muhammad Fakhri
Abdillah

Editor: Taufik Maulana, Rizal Rahman, Tresna Widiyaman & Muhammad Fakhri
Abdillah

Revisi Batch 9

Penerbit : **PT. Cilsy Fiolution Indonesia**

Web Site : <https://cilsyfiolution.com> , <https://devops.cilsy.id>

Sanksi Pelanggaran Pasal 113 Undang-undang Nomor 28 Tahun 2014 tentang Hak Cipta

1. Setiap orang yang dengan tanpa hak melakukan pelanggaran hak ekonomi sebagaimana dimaksud dalam pasal 9 ayat (1) huruf i untuk penggunaan secara komersial dipidana dengan pidana penjara paling lama 1 (satu) tahun dan atau pidana denda paling banyak Rp100.000.000,00 (seratus juta rupiah).
2. Setiap orang yang dengan tanpa hak dan atau tanpa izin pencipta atau pemegang hak cipta melakukan pelanggaran hak ekonomi pencipta sebagaimana dimaksud dalam pasal 9 ayat (1) huruf c, huruf d, huruf f, dan atau huruf h, untuk penggunaan secara komersial dipidana dengan pidana penjara paling lama 3 (tiga) tahun dan atau pidana denda paling banyak Rp500.000.000,00 (lima ratus juta rupiah)
3. Setiap orang yang dengan tanpa hak dan atau tanpa izin pencipta atau pemegang hak melakukan pelanggaran hak ekonomi pencipta sebagaimana dimaksud dalam pasal 9 ayat (1) huruf a, huruf b, huruf e, dan atau huruf g, untuk penggunaan secara komersial dipidana dengan pidana penjara paling lama 4 (empat) tahun dan atau pidana denda paling banyak Rp1.000.000.000,00 (satu miliar rupiah)
4. Setiap orang yang memenuhi unsur sebagaimana dimaksud pada ayat (3) yang dilakukan dalam bentuk pembajakan, dipidana dengan pidana penjara paling lama 10 (sepuluh) tahun dan atau pidana denda paling banyak Rp4.000.000.000,00 (empat miliar rupiah)



Daftar Isi

Cover.....	1
8. Docker Container Fundamental & Basic Docker Orchestration di Server	
Production Part 2.....	5
Learning Outcomes.....	5
Outline Materi.....	5
8.1. Apa yang ada didalam Image (dan apa yang tidak).....	6
8.2. Mengeksplorasi Docker Hub.....	6
8.2.1. Mendaftar Docker Hub.....	7
8.2.2. Bagaimana Cara Melihat Image-Image yang Bagus.....	7
8.2.2.1. Image Official.....	8
8.2.2.2. Image non Official yang memiliki reputasi baik.....	9
8.2.3. Bagaimana Cara Mengunduh Berbagai Macam Image.....	11
8.2.3.1. Melihat List Image Yang Ada di Local.....	11
8.2.3.2. Image ID vs Tag.....	12
8.2.3.3. Efisiensi Penyimpanan Image.....	13
8.2.3.4. Latest Tidaklah Selalu “latest”.....	13
8.3. Image Layer.....	14
8.3.1. Visualisasi Layer pada Image.....	16
8.3.2. Container dan Copy on Write.....	18
8.4. Image Tagging dan Cara Push ke Docker Hub.....	20
8.4.1. Praktek Image Tagging.....	20
8.4.2. Docker Login.....	21
8.4.3. Push Image ke Docker Hub.....	21
8.5. Exercise.....	22
8.6. Building Image : Basic Dockerfile.....	22
8.6.1. Apa itu Dockerfile ? Untuk apa Dockerfile?.....	22
8.6.2. Memahami Dockerfile dari contoh.....	24
8.7. Building image : Running Docker Builds.....	28



8.8. Building Image : Extending Official Image.....	31
8.8.1. Build Official Image yang sudah dimodifikasi.....	31
8.9. Study Case: Writing Dockerfiles.....	34
8.9.1. Menambahkan Landing Page ke Container NGINX.....	34
8.9.2. Membuat Dockerfile untuk Aplikasi Python.....	36
8.9.3. Membuat Dockerfile untuk Aplikasi GoLang.....	39
8.10. Push ke Docker Hub.....	44
8.11. Exercise.....	46
8.12. Summary.....	47

8.

Docker Container Fundamental & Basic Docker Orchestration di Server Production Part 2

Learning Outcomes

Setelah selesai mempelajari bab ini, peserta mampu :

1. Mengenal DockerHub dan Manage Image Dockerhub
2. Melakukan Push ke Dockerhub
3. Melakukan Build Image dengan Dockerfile

Outline Materi

1. Eksplorasi Dockerhub
2. Image Layer
3. Image Tagging dan Push DockerHub
4. Building Image Dockerfile



8.1. Apa yang ada didalam Image (dan apa yang tidak)

Image seperti yang sudah dibahas pada materi sebelumnya adalah aplikasi yang ingin kita jalankan. Tapi sebenarnya terbuat dari apa image ini?

Apa saja yang ada di dalam Image :

1. Binary dan dependensi aplikasi. Seperti file-file konfigurasi `.conf`, file executable `.sh` dari aplikasi.
2. Metadata terkait bagaimana image tersebut akan dijalankan.

Image bukanlah template OS seperti file ISO sehingga Image tidak memiliki :

1. Kernel OS
2. Driver Module
3. Dan modul-modul lainnya yang terkait hardware.

Akibatnya Image dapat berukuran super kecil berupa sebuah library saja, maupun super besar (bergiga-giga) berupa distro lengkap dengan aplikasi-aplikasi didalamnya seperti Ubuntu + Wordpress + Mysql.

Berikutnya kita akan banyak membahas lebih detail tentang hal-hal yang berkaitan dengan image, bagaimana konsep image bekerja, dan manajemen-manajemen yang bisa kita lakukan seputar image.

8.2. Mengeksplorasi Docker Hub

Jika kita berbicara terkait Image maka tidak lepas dari membicarakan Registry, atau tempat penyimpanan Image. Docker Hub merupakan Public Cloud Image Registry atau tempat penyimpanan image yang terdapat di internet. Kita dapat mengunduh berbagai macam image dari Docker Hub maupun menyimpan image kita sendiri ke dalamnya.

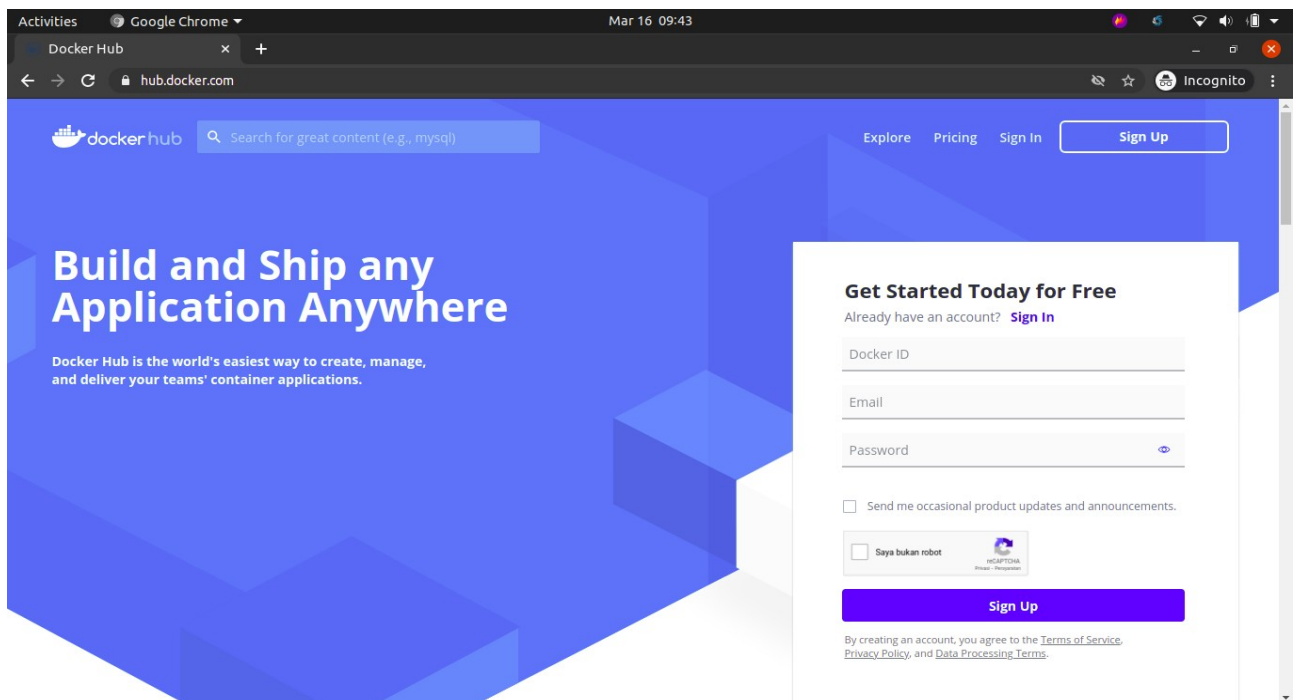


Pada bagian ini kita akan mempelajari apa-apa saja informasi yang bisa kita dapatkan di Docker Hub dan bagaimana cara memanfaatkan Docker Hub itu sendiri selama kita menggunakan Docker kedepannya.

8.2.1. Mendaftar Docker Hub

Hal yang pertama harus kita lakukan adalah mendaftar di Docker Hub. Karena nantinya kita akan cukup banyak mengeksplorasi isi dari web Docker hub.

Cara mendaftar Docker Hub sangat mudah. Anda cukup buka halaman web <https://hub.docker.com> dan lakukan pendaftaran seperti layaknya mendaftar web biasa.

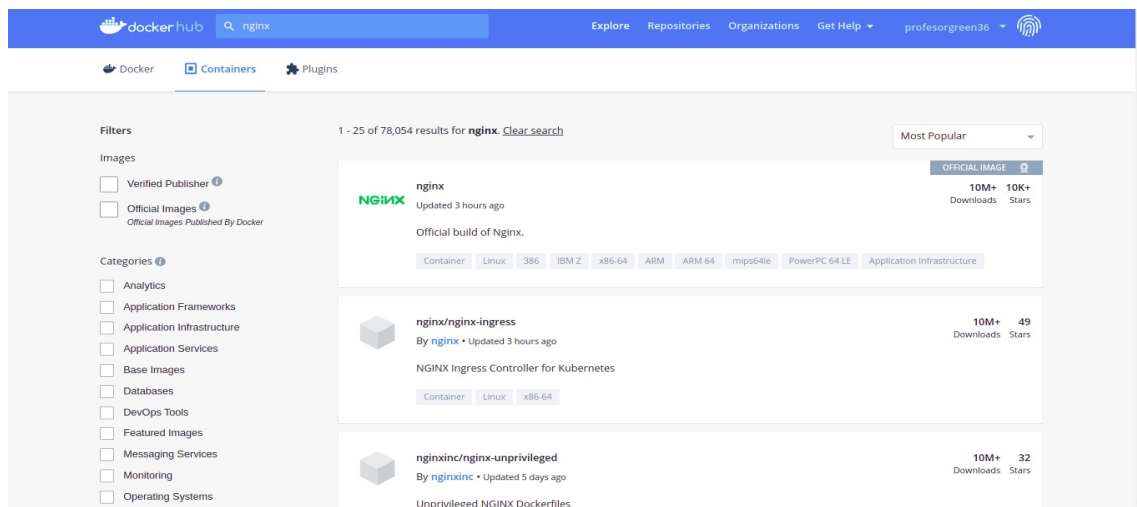


Setelah berhasil mendaftar, simpanlah username dan password akun Docker Hub Anda karena kita akan membutuhkannya sewaktu-waktu.

8.2.2. Bagaimana Cara Melihat Image-Image yang Bagus

Apabila kita mencari suatu image di Docker Hub, misalnya nginx. Maka kita akan mendapatkan 40 ribu lebih image repository terkait nginx.





Hasil pencarian image dockerhub

Lalu bagaimana kita menentukan mana image yang bagus?

8.2.2.1. Image Official

Sangat disarankan untuk menggunakan image official karena memang image ini yang benar-benar dikelola secara resmi oleh perusahaan pengembangnya masing-masing. Sehingga versi-versinya dijamin yang paling baru, memiliki dokumentasi yang sangat lengkap terkait bagaimana penggunaannya, hingga memiliki Dockerfile yang sesuai standar (terkait Dockerfile akan dibahas lebih lanjut pada materi berikutnya)

Ciri image official adalah :

1. Terdapat tulisan Official dibagian samping dari nama image repositorynya.
2. Tidak memiliki awalan nama user Docker hub pada nama imagenya. Jadi hanya murni bertuliskan nginx saja atau wordpress saja atau mysql saja. Bukan seperti kuasai/nginx atau bitnami/wordpress.



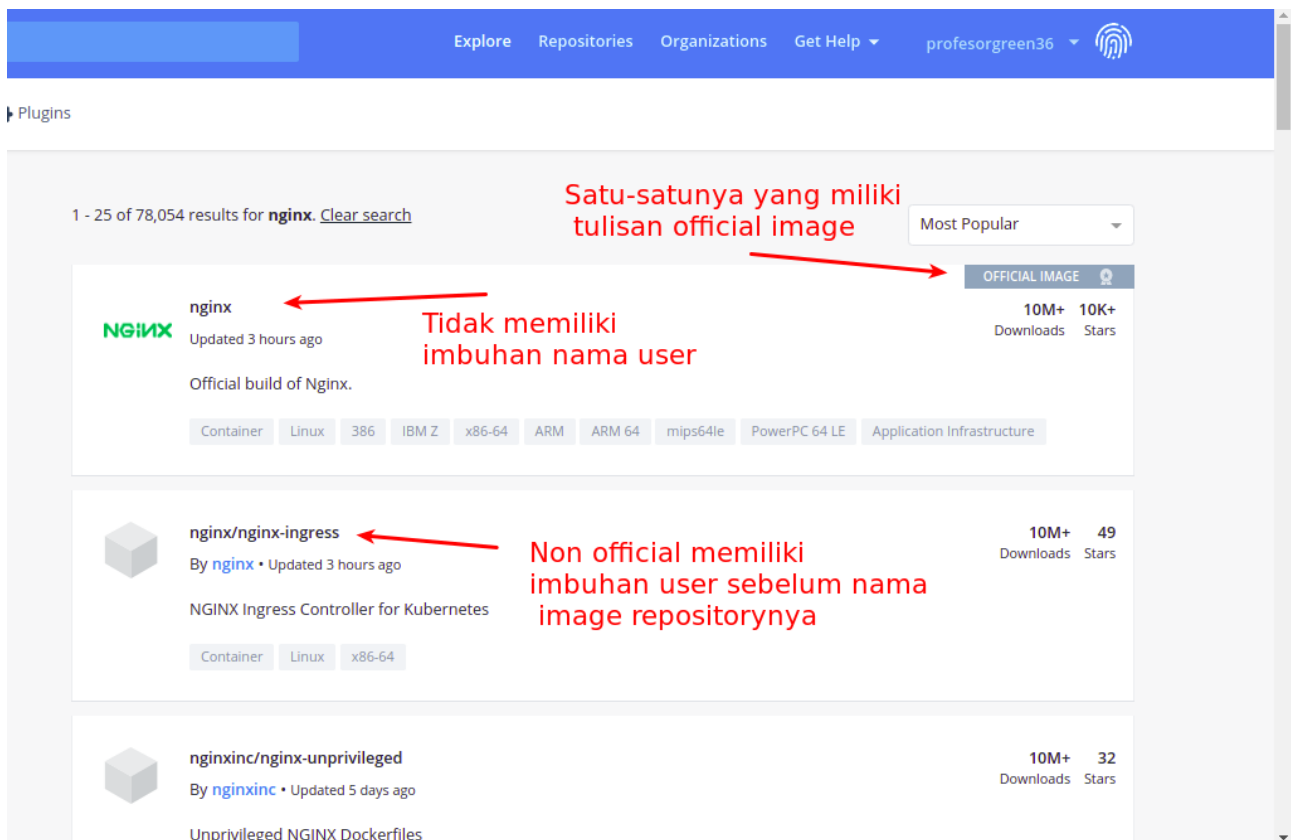


Image official dan non-official

8.2.2.2. Image non Official yang memiliki reputasi baik

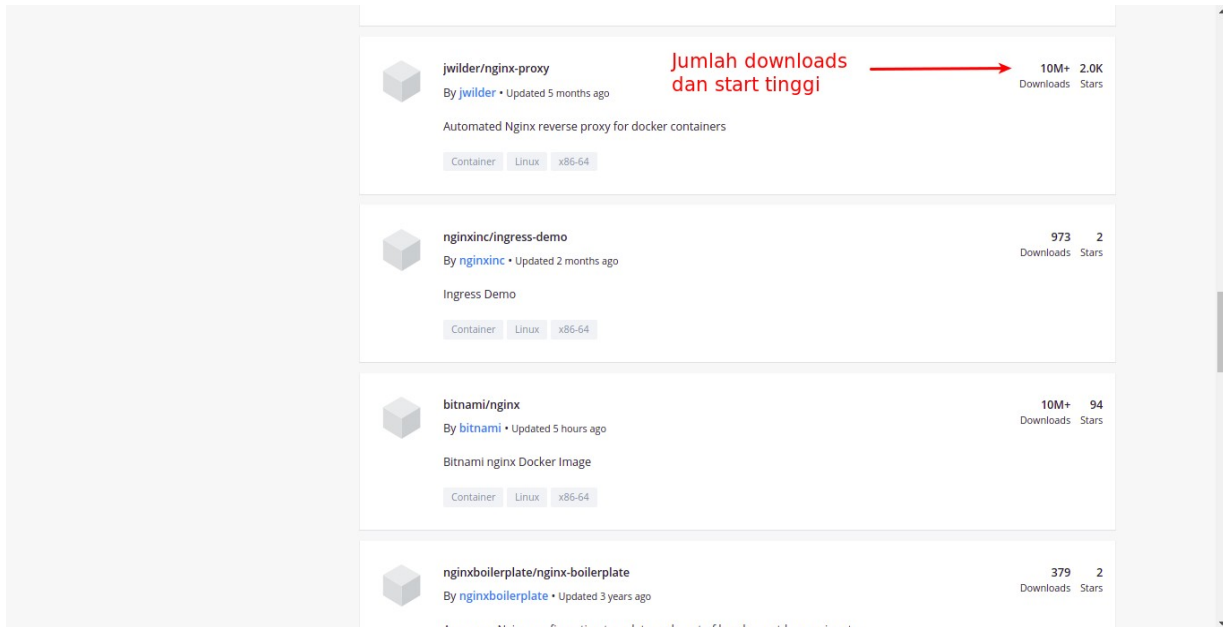
Terkadang ada image-image non official yang lebih cocok kebutuhannya untuk kita dibanding image official. Misalnya karena ada tambahan konfigurasi atau paket lainnya yang lebih cocok untuk kita.

Namun sebaiknya kita tetap melihat bahwa image non-official yang kita gunakan memiliki 3 ciri-ciri berikut :

1. Jumlah pull dan starsnya tinggi. Karena ini membuktikan bahwa banyak user yang mengunduh image ini dan memberikan rating baik.
2. Dokumentasinya lengkap. Cobalah untuk meng-klik salah satu image repository yang ada, disana nanti akan tampil dokumentasi penggunaan dari image tersebut. Jika lengkap dan mudah dimengerti, bisa diartikan bahwa user yang mengupload image ini “sangat niat” dan “bertanggung jawab”.



3. Pengembangnya masih aktif. Dapat Anda cek juga jika Anda meng-klik tab “Most Popular” lalu “Recently Update”. Jika cukup baru, maka dapat diartikan pengembang ini masih aktif mengelola image ini.



Jumlah Downloads & Stars Tinggi



Dokumentasi Lengkap dan jelas



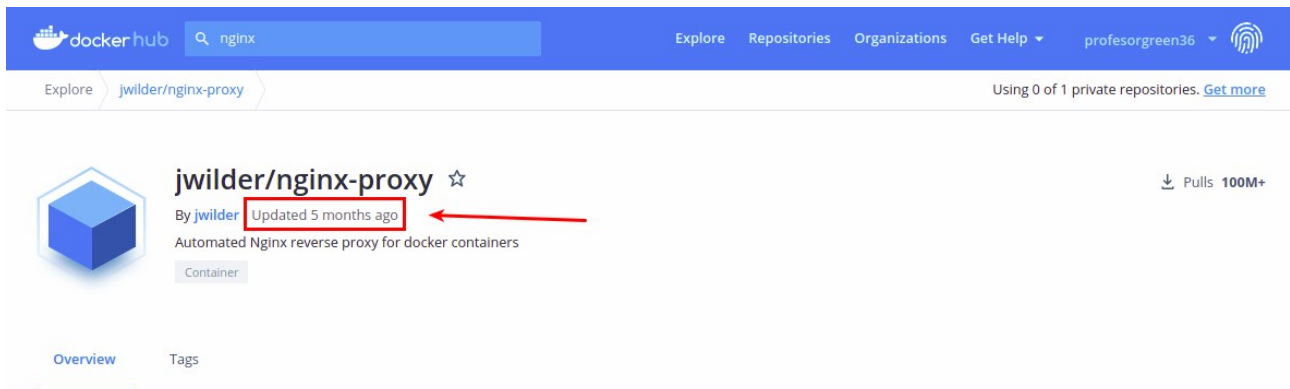


Image yang cukup update

8.2.3. Bagaimana Cara Mengunduh Berbagai Macam Image

Di bagian ini kita akan sedikit mempraktekkan bagaimana cara untuk mengunduh berbagai image berdasarkan hasil pencarian kita di Docker Hub. Karena pada prakteknya nanti, kita akan sering melakukan pencarian di google maupun di Docker Hub untuk mengetahui informasi image yang akan kita gunakan. Nama imagenya, versinya, tagnya apa, sebaiknya gunakan image yang mana dll. Setelah sudah kita tentukan akan menggunakan yang mana, barulah kita aplikasikan ke dalam command-command Docker.

8.2.3.1. Melihat List Image Yang Ada di Local

Image-image yang kita sudah kita unduh/pull dari Docker Hub akan tersimpan di local Host kita. Kita dapat melihatnya dengan perintah berikut :

```
$ docker image ls
```

```
rizal@rizal-Inspiron-5468 ~ $ docker image ls
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
httpd	latest	dabb52744997	38 hours ago	178MB
mongo	latest	c1d6c06b5775	6 days ago	381MB
alpine	latest	196d12cf6ab1	2 weeks ago	4.41MB
nginx	latest	06144b287844	3 weeks ago	109MB
mysql	<none>	563a026a1511	3 weeks ago	372MB
mysql	latest	6a834f03bd02	3 weeks ago	484MB
cilsy/wordpress-thegem	<none>	6202e0e353e7	3 weeks ago	409MB
phpmyadmin/phpmyadmin	<none>	126b8717cebb	5 weeks ago	166MB
traefik	<none>	7df6f82b8839	5 weeks ago	52.5MB

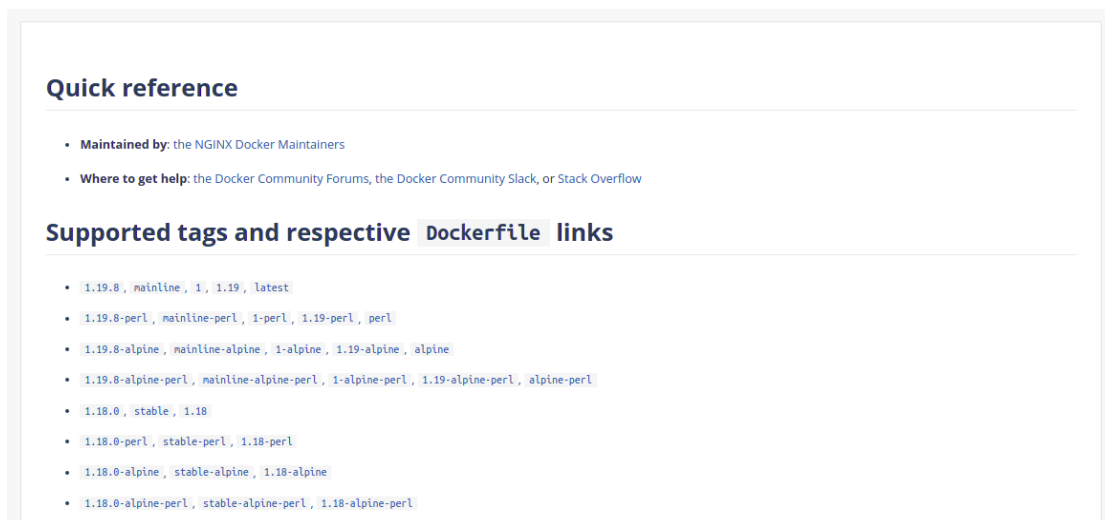
Informasi yang bisa kita dapatkan dari perintah ini diantaranya nama imagenya apa, tag nya apa, image ID nya, serta ukuran dari image ini berapa. Hal ini akan kita gali berikutnya. Sementara disini kita cukup mengetahui bahwa



perintah diatas dapat menampilkan daftar image yang tersimpan di local Host kita.

8.2.3.2. Image ID vs Tag

Untuk memahami hal ini, kita coba praktekan saja. Pertama-tama kita lihat beberapa tag yang dapat digunakan pada image nginx official (Anda dapat akses ini pada bagian Full Description pada Image di Docker Hub) :



Semua tag tersebut dapat digunakan dengan format berikut :

```
$ docker image pull nginx:<tag>
```

Contoh :

```
$ docker image pull nginx:1.15
$ docker image pull nginx:1.15.4
$ docker image pull nginx
```

Sekarang coba lihat lagi apa yang terjadi setelah kita mengunduh 3 image tersebut :

```
$ docker image ls
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
httpd	latest	dabb52744997	38 hours ago	178MB
nginx	1.15	bc26f1ed35cf	46 hours ago	109MB
nginx	1.15.4	bc26f1ed35cf	46 hours ago	109MB
nginx	latest	bc26f1ed35cf	46 hours ago	109MB
mongo	latest	c1d6c06b5775	6 days ago	381MB

Kita bisa melihat bahwa image nginx memiliki 3 buah tag yang berbeda namun memiliki Image ID dan ukuran yang sama. Kenapa hal ini bisa terjadi?

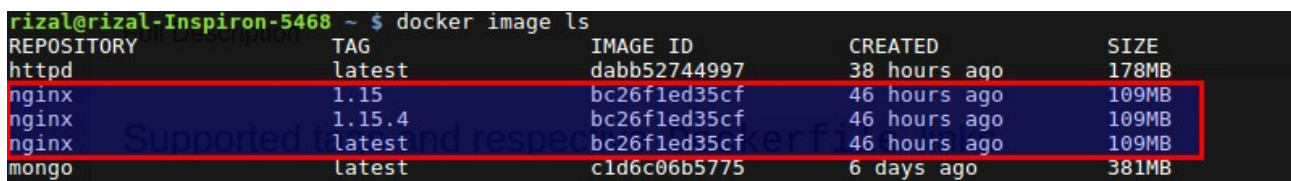


Ini diakibatkan karena ketiga buah tag tersebut sebenarnya merujuk ke image yang sama, sehingga hanya memiliki 1 buah image ID saja. Sehingga bisa disimpulkan bahwa image ID merupakan identitas unik dari suatu image, sedangkan tag hanyalah rujukan kepada suatu image ID.

Tag dapat lebih memudahkan kita untuk menentukan tujuan penggunaan dari suatu versi image. Contohnya pada image nginx terdapat tag stable dan tag latest. Walaupun pada tag stable versi nginx lebih lama dibanding tag latest, namun dari sini kita bisa mengetahui bahwa kita jika kita ingin menggunakan versi nginx yang lebih stable maka kita menggunakan nginx versi 1.14, namun jika ingin versi nginx yang lebih baru maka kita akan mendapatkan nginx versi 1.15.

8.2.3.3. Efisiensi Penyimpanan Image

Kita coba lihat kembali gambar berikut :



REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
httpd	latest	dabb52744997	38 hours ago	178MB
nginx	1.15	bc26f1ed35cf	46 hours ago	109MB
nginx	1.15.4	bc26f1ed35cf	46 hours ago	109MB
nginx	latest	bc26f1ed35cf	46 hours ago	109MB
mongo	latest	c1d6c06b5775	6 days ago	381MB

Sekilas pada gambar tersebut terlihat bahwa kita seperti menyimpan 3 buah image dengan tag yang berbeda dengan masing-masing berukuran 109MB. Sehingga total yang tersimpan di local Host kita adalah 300MB.

Pada kenyatannya tidak. Berhubung ketiga buah image tersebut merujuk pada image ID yang sama, maka sebenarnya yang disimpan pada local hanyalah 1 file saja, yaitu sebesar 109MB.

Hal ini bisa terjadi karena prinsip layer pada image yang akan kita bahas lebih detail berikutnya.

8.2.3.4. Latest Tidaklah Selalu “latest”

Sebagai catatan bahwa Tag Latest itu tidak selalu menunjukkan versi terbaru dari suatu image. Melainkan lebih cocok menunjukkan versi “default” dari



image tersebut. Versi default ketika kita tidak memberikan tag pada saat melakukan pull/push images.

Namun pada kebanyakan kasus, khususnya pada image-image yang dikelola secara rutin seperti image-image official, Latest memang merujuk pada versi terbaru pada image tersebut.

8.3. Image Layer

Jika kita memperhatikan lebih rinci, selama kita mempraktekkan mengambil berbagai macam image dengan berbagai tag, ada beberapa proses yang seperti terjadi beberapa proses download sekaligus seperti ini :

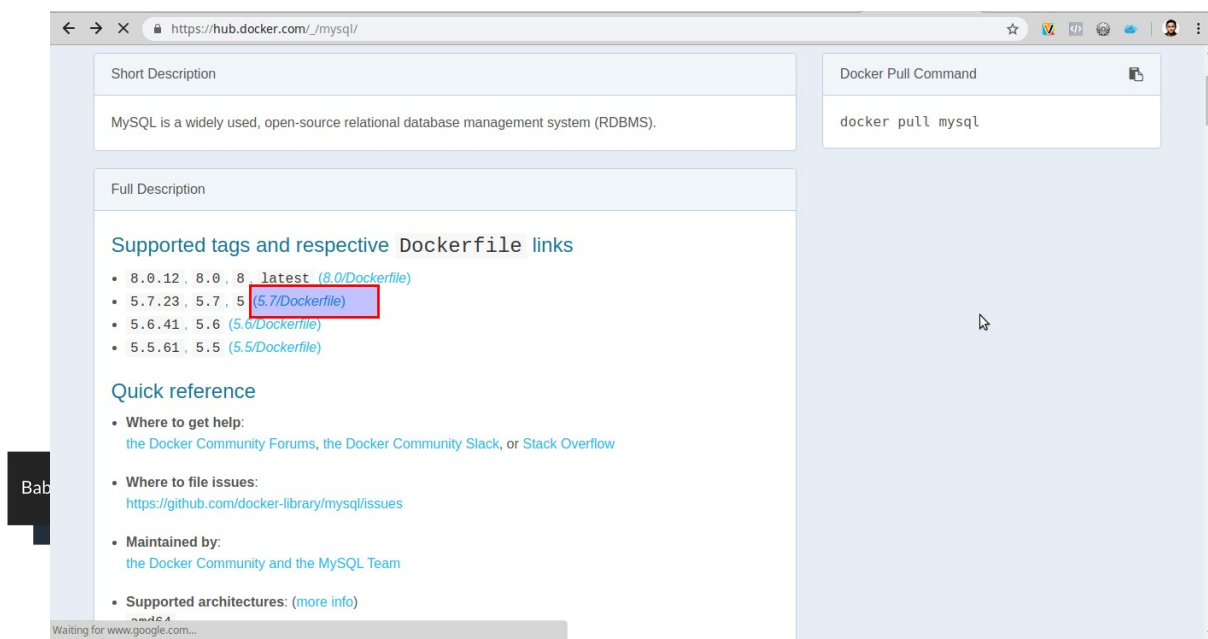
```
rizal@rizal-Inspiron-5468 ~ $ docker image pull nginx:1.14.0-alpine
1.14.0-alpine: Pulling from library/nginx
c67f3896b22c: Downloading [=====] 1.35MB/2.107MB
428de5b8d58a: Downloading [=====] 2.473MB/5.454MB
7efd417f3e28: Download complete
61a56b170416: Waiting
```

Walaupun image yang kita unduh 1, akan tetapi kenapa seperti ada beberapa file yang di download ?

Hal ini karena sebuah Image bukanlah hanya berbentuk 1 file utuh, melainkan dibangun dari tumpukan layer. Persis seperti pancake.

Masing-masing layer ini mewakili sebuah instruksi yang sudah ditentukan pada Dockerfile. Karena perlu Anda diketahui bahwa sebuah image itu sebenarnya dibangun berdasarkan seluruh instruksi yang ada pada Dockerfile.

Sebagai contoh cobalah buka image repository mysql melalui Dockerhub, lalu disana cobalah klik salah satu menu Dockerfile yang ada :



Maka kita akan mendapatkan sebuah rangkaian script yang cukup panjang seperti ini :

```

81 lines (68 sloc) | 3.69 KB
Raw Blame History
1 FROM debian:stretch-slim
2
3 # add our user and group first to make sure their IDs get assigned consistently, regardless of whatever dependencies get added
4 RUN groupadd -r mysql && useradd -r -g mysql mysql
5
6 RUN apt-get update && apt-get install -y --no-install-recommends gnupg dirmngr && rm -rf /var/lib/apt/lists/*
7
8 # add gosu for easy step-down from root
9 ENV GOSU_VERSION 1.7
10 RUN set -x \
11     && apt-get update && apt-get install -y --no-install-recommends ca-certificates wget && rm -rf /var/lib/apt/lists/* \
12     && wget -O /usr/local/bin/gosu "https://github.com/tianon/gosu/releases/download/$GOSU_VERSION/gosu-$(dpkg --print-arch)" \
13     && wget -O /usr/local/bin/gosu.asc "https://github.com/tianon/gosu/releases/download/$GOSU_VERSION/gosu-$(dpkg --print-arch).asc" \
14     && export GNUPGHOME="$(mktemp -d)" \
15     && gpg --keyserver ha.pool.sks-keyservers.net --recv-keys B42F6819007F00F88E364FD4036A9C25BF357DD4 \
16     && gpg --batch --verify /usr/local/bin/gosu.asc /usr/local/bin/gosu \
17     && gpgconf --kill all \
18     && rm -rf "$GNUPGHOME" /usr/local/bin/gosu.asc \
19     && chmod +x /usr/local/bin/gosu \
20     && gosu nobody true \
21     && apt-get purge -y --auto-remove ca-certificates wget
22
23 RUN mkdir /docker-entrypoint-initdb.d
24
25 RUN apt-get update && apt-get install -y --no-install-recommends \
26 # for MYSQL_RANDOM_ROOT_PASSWORD
27     pwgen \
28 # for mysql_ssl_rsa_setup
29     openssl \
30 # FATAL ERROR: please install the following Perl modules before executing /usr/local/mysql/scripts/mysql_install_db:
31 # File::Basename

```

Masing-masing yang ditandai diatas merepresentasikan sebuah instruksi, dan setiap instruksi ini akan menjadi sebuah layer pada image tersebut. Artinya apa? Dalam sebuah image bisa memiliki layer yang bervariasi, bisa 1 layer, bisa juga ratusan layer. Tergantung dari jumlah instruksi yang diberikan.

Tidak masalah jika Anda masih belum memahami apa Dockerfile dan bagaimana bisa memahami script super panjang tersebut (karena materi Dockerfile akan dibahas berikutnya). Yang penting Anda sudah memiliki mindset bahwa sebuah image itu terdiri dari tumpukan layer, dan masing-masing layer ini merupakan instruksi-instruksi yang sudah ditentukan di dalam Dockerfile.

Kita juga dapat coba mengecek bahwa benar image ini terdiri dari layer-layer menggunakan command berikut :

```
$ docker image history <nama image>
```



Contoh :

```
$ docker image history mysql
```

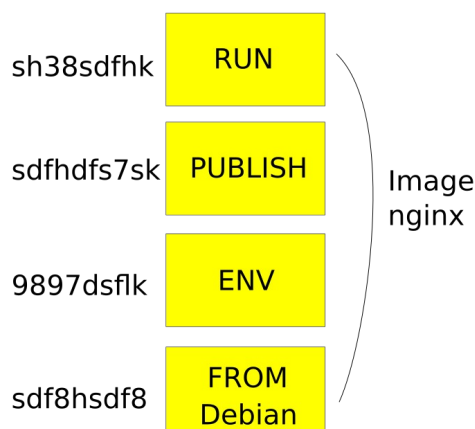
```
rizal@rizal-Inspiron-5468 ~$ docker image history mysql
IMAGE          CREATED          CREATED BY                                      SIZE      COMMENT
6a834f03bd02   3 weeks ago     /bin/sh -c #(nop) CMD ["mysqld"]              0B        Dockerfile
<missing>      3 weeks ago     /bin/sh -c #(nop) EXPOSE 3306/tcp 3306/tcp     0B        Dockerfile
<missing>      3 weeks ago     /bin/sh -c #(nop) ENTRYPOINT ["docker-entryp... 0B        Dockerfile
<missing>      3 weeks ago     /bin/sh -c ln -s usr/local/bin/docker-entryp... 34B        Dockerfile
<missing>      3 weeks ago     /bin/sh -c #(nop) COPY file:59647086b032bcb2... 6.53kB     Dockerfile
<missing>      3 weeks ago     /bin/sh -c #(nop) COPY dir:110dcf1221c1f9432... 1.22kB     Dockerfile
<missing>      3 weeks ago     /bin/sh -c #(nop) VOLUME [/var/lib/mysql]       0B        Dockerfile
<missing>      3 weeks ago     /bin/sh -c { echo mysql-community-server m... 369MB      Dockerfile
<missing>      3 weeks ago     /bin/sh -c echo "deb http://repo.mysql.com/a... 56B        Dockerfile
<missing>      3 weeks ago     /bin/sh -c #(nop) ENV MYSQL_VERSION=8.0.12-... 0B        Dockerfile
<missing>      3 weeks ago     /bin/sh -c #(nop) ENV MYSQL_MAJOR=8.0.12-... 0B        Dockerfile
<missing>      3 weeks ago     /bin/sh -c set -ex; key='A4A9406876FCBD3C45... 25kB       Dockerfile
<missing>      3 weeks ago     /bin/sh -c apt-get update && apt-get install... 44.7MB     Dockerfile
<missing>      3 weeks ago     /bin/sh -c mkdir /docker-entrypoint-initdb.d 0B         Dockerfile
<missing>      3 weeks ago     /bin/sh -c set -x && apt-get update && apt-... 4.44MB     Dockerfile
<missing>      3 weeks ago     /bin/sh -c #(nop) ENV GOSU_VERSION=1.7         0B         Dockerfile
<missing>      3 weeks ago     /bin/sh -c apt-get update && apt-get install... 10.2MB     Dockerfile
<missing>      3 weeks ago     /bin/sh -c groupadd -r mysql && useradd -r -... 329kB      Dockerfile
<missing>      3 weeks ago     /bin/sh -c #(nop) CMD ["bash"]                 0B         Dockerfile
<missing>      3 weeks ago     /bin/sh -c #(nop) ADD file:e6ca98733431f75e9... 55.3MB     Dockerfile
```

Anda bisa melihat bahwa ada kemiripan antara hasil command tersebut dengan isi Dockerfile mysql yang kita lihat sebelumnya, karena memang command ini menunjukkan instruksi-instruksi apa saja yang dilakukan untuk membangun image mysql ini berdasarkan Dockerfile-nya.

Canggihnya lagi, setiap layer ini memiliki id unik tersendiri menggunakan enkripsi SHA. SHA ini digunakan untuk mencocokkan apakah ada perbedaan antar layer, untuk memastikan bahwa layer yang disimpan hanyalah 1. Untuk lebih jelasnya kita akan coba memvisualisasikan seperti apa layer-layer pada image itu.

8.3.1. Visualisasi Layer pada Image

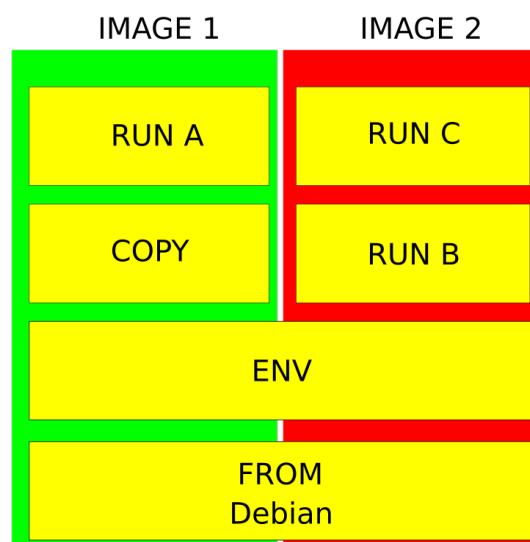
Perhatikan gambar berikut :



Ilustrasi Layer Image

Itu adalah gambar sebuah image nginx yang terdiri dari beberapa layer. Layer ini harus dimulai dari FROM untuk menentukan base-image dari layer ini menggunakan image apa. Contohnya pada nginx menggunakan FROM image debian jessie. Kemudian berikutnya ada layer-layer lainnya seperti penentuan environment variable, membuka port, mengkopi file dll. Tiap layer memiliki id unik berupa enkripsi SHA-nya masing-masing.

Sekarang perhatikan gambar berikut :



Ilustrasi Layer 2 Image

Gambar tersebut menunjukkan bahwa ada 2 buah image yang berbeda, namun beberapa layernya ternyata sama persis. Kenapa bisa ada layer yang sama? Karena instruksinya sama, dan urutan eksekusi instruksinya sama. Disinilah letak kecanggihannya. Docker tidak akan menyimpan 2 buah layer yang sama lebih dari 1 kali. Efeknya tentu penghematan secara storage serta lebih cepat dalam proses pulling dan pushing image ke Docker Hub.

Berikut adalah contoh lain misalnya ada 2 buah image wordpress dengan versi yang sama namun memiliki data website yang berbeda :

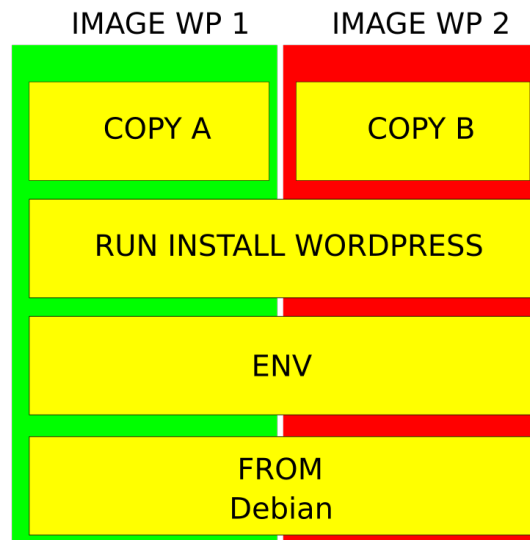


Image berbeda data

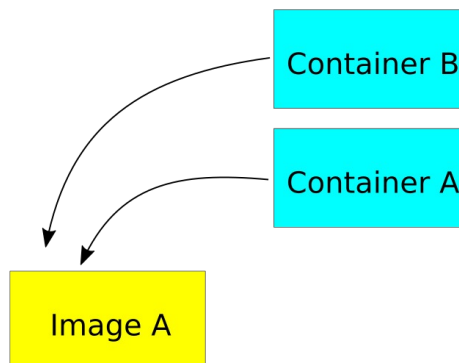
Dari gambar tersebut bisa dilihat bahwa walaupun imagenya berbeda, namun data pada layer 1-3 (dari bawah) hanyalah disimpan 1 kali karena isi instruksinya sama dan urutan instruksinya sama. Sedangkan data yang disimpan 2 kali hanyalah layer paling atasnya saja.

Pada proses pulling dan pushing image pun sama. Ketika kita melakukan pulling image dari Docker Hub, sistem Docker akan mencocokkan SHA dari masing-masing layer di local Host kita dengan yang ada di Docker Hub. Yang didownload oleh Host kita hanyalah layer-layer yang SHA nya berbeda saja. Apabila seluruh SHA telah sama, maka tidak ada yang di download. Sangat menghemat waktu dan resource data bukan?

8.3.2. Container dan Copy on Write

Ketika kita menjalankan sebuah Container, maka sebenarnya Container tersebut hanyalah menjadi sebuah layer baru diatas image. Perhatikan gambar berikut :



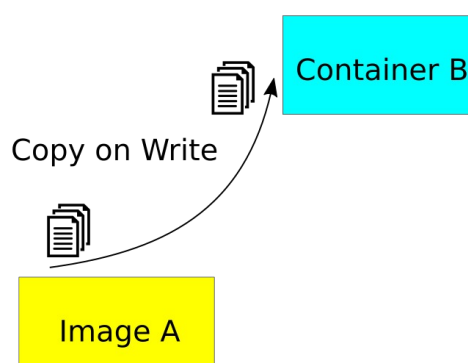


Ilustrasi Container dan layer

Akibatnya proses container ini ukurannya tentunya sangatlah kecil. Paling isinya hanyalah metadata karena proses-proses utamanya sudah ada pada layer-layer image dibawahnya.

Bagaimana kalau kita melakukan perubahan file yang terdapat di dalam image melalui Container tersebut ? Misalnya di dalam image nginx terdapat file nginx.conf, kemudian kita melakukan perubahan file nginx.conf tersebut dari dalam shell Container?

Ketika ini terjadi, proses yang digunakan adalah Copy on Write. Alurnya adalah file dari image tersebut akan di copy terlebih dahulu ke dalam layer Container sehingga menjadi sebuah file yang baru. Yang diubah-ubah oleh Container tersebut adalah file yang didalam layer Container itu sendiri, bukan file yang didalam base-image nginx. File nginx.conf di dalam image nginx tetaplah original seperti semula.



Ilustrasi proses Copy on Write

Metode ini merupakan metode yang cukup brilian. Karena akibatnya ketika ada beberapa Container yang menggunakan image yang sama, tidak akan saling mempengaruhi 1 sama lain.

8.4. Image Tagging dan Cara Push ke Docker Hub

Kita sudah mempelajari konsep tagging dan Docker Hub. Lalu bagaimana kalau kita ingin melakukan tagging pada image kita sendiri dan melakukan upload image tersebut ke Docker Hub?

Karena pada prakteknya kita nantinya akan cukup sering melakukan building image kita sendiri berdasarkan kebutuhan dan kondisi. Kita tidak lagi menggunakan image resmi dari Docker Hub karena image-image tersebut pasti akan kita modifikasi.

Disini kita akan mempelajari bagaimana cara melakukan hal tersebut dan bagaimana cara agar image yang baru tersebut bisa kita upload di Docker Hub agar bisa kita gunakan kapanpun dimanapun.

8.4.1. Praktek Image Tagging

Untuk melakukan tagging dari sebuah image ke image kita sendiri, kita dapat menggunakan format berikut :

Untuk official image

```
$ docker image tag <source_image>:<tag> <username>/<dst_image>:<tag>
```

Contoh :

```
$ docker image tag nginx:1.15.4 cilsy/nginxmyapp:1.15.4
$ docker image tag nginx cilsy/nginxmyapp
```

Untuk non-official

```
$ docker image tag <username>/<source_image>:<tag> <username>/<dst_image>:<tag>
```

Contoh :



```
$ docker image tag bitnami/wordpress:4.9.8 cilsy/wordpresscustom:4.9.8
$ docker image tag bitnami/wordpress cilsy/wordpresscustom
```

Cek hasilnya menggunakan command :

```
$ docker image ls
```

8.4.2. Docker Login

Untuk bisa melakukan Push Image ke Docker Hub pertama-tama kita harus menghubungkan terlebih dahulu Host docker kita ke akun Docker Hub yang sudah kita miliki. Caranya dengan mengetikkan perintah berikut :

```
$ docker login
```

Lalu isikan username dan password Anda masing-masing. Pastikan Anda mendapatkan pesan berhasil login seperti ini :

```
rizal@rizal-Inspiron-5468 ~ $ docker login
Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head over to https://hub.docker.com to create one.
Username: cilsy
Password:
WARNING! Your password will be stored unencrypted in /home/rizal/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store
Login Succeeded
```

8.4.3. Push Image ke Docker Hub

Untuk melakukan push image yang sudah kita tagging sebelumnya caranya sangat mudah. Kita tinggal mengeksekusi perintah berikut :

```
$ docker image push <nama user docker hub>/<nama image>:<tag>
```

Contohnya disini saya ingin memberikan tag latest saja sehingga commandnya menjadi seperti ini :

```
$ docker image push cilsy/nginxmyapp
```

Nantinya kita bisa juga menggunakan tag-tag tambahan seperti :

```
$ docker image push cilsy/nginxmyapp:1.15.1
```

Atau :

```
$ docker image push cilsy/nginxmyapp:staging
```

Bebas bisa disesuaikan dengan kebutuhan kita masing-masing.



8.5. Exercise

Teori

1. Bolehkah kita hanya melakukan tagging nama image tanpa nama user Docker Hub? Jika boleh, jelaskan kenapa dan kapan bisa menggunakan cara ini. Jika tidak boleh, jelaskan juga alasan Anda.

Praktek

1. Carilah nama image untuk aplikasi apache webserver di Docker Hub.
2. Lakukan image tagging pada image apache tersebut, lalu push ke Docker Hub. Nama image bebas.
3. Hapus semua image terkait apache dari lokal.
4. Jalankan Container webserver apache menggunakan image yang sudah anda push ke Docker Hub tersebut.

8.6. Building Image : Basic Dockerfile

8.6.1. Apa itu Dockerfile ? Untuk apa Dockerfile?

Dockerfile merupakan resep untuk membangun sebuah image. Semua image pasti memiliki sebuah Dockerfile didalamnya. Anda bisa coba lihat-lihat contoh Dockerfile dari image-image yang ada di Docker Hub, seperti contohnya Dockerfile pada salah satu image Postgres berikut :



Full Description

Supported tags and respective Dockerfile links

- 11-beta4 , 11 ([11/Dockerfile](#))
- 11-beta4-alpine , 11-alpine ([11/alpine/Dockerfile](#))
- 10.5 , 10 , latest ([10/Dockerfile](#))
- 10.5-alpine , 10-alpine , alpine ([10/alpine/Dockerfile](#))
- 9.6.10 , 9.6 , 9 ([9.6/Dockerfile](#))
- 9.6.10-alpine , 9.6-alpine , 9-alpine ([9.6/alpine/Dockerfile](#))
- 9.5.14 , 9.5 ([9.5/Dockerfile](#))
- 9.5.14-alpine , 9.5-alpine ([9.5/alpine/Dockerfile](#))
- 9.4.19 , 9.4 ([9.4/Dockerfile](#))
- 9.4.19-alpine , 9.4-alpine ([9.4/alpine/Dockerfile](#))
- 9.3.24 , 9.3 ([9.3/Dockerfile](#))
- 9.3.24-alpine , 9.3-alpine ([9.3/alpine/Dockerfile](#))

Quick reference

- **Where to get help:**
[the Docker Community Forums](#), [the Docker Community Slack](#), or [Stack Overflow](#)

Docker file pada image postgres

```

173 lines (158 sloc) | 7.53 KB
1  # vim:set ft=dockerfile:
2  FROM debian:stretch-slim
3
4  RUN set -ex; \
5      if ! command -v gpg > /dev/null; then \
6          apt-get update; \
7          apt-get install -y --no-install-recommends \
8              gnupg \
9              dirmngr \
10             ; \
11             rm -rf /var/lib/apt/lists/*; \
12         fi
13
14 # explicitly set user/group IDs
15 RUN set -eux; \
16     groupadd -r postgres --gid=999; \
17     https://salsa.debian.org/postgresql/postgresql-common/blob/997d842ee744687d99a2b2d95c1083a2615c79e8/debian/postgresql-common
18     useradd -r -g postgres --uid=999 --home-dir=/var/lib/postgresql --shell=/bin/bash postgres; \
19     # also create the postgres user's home directory with appropriate permissions
20     # see https://github.com/docker-library/postgres/issues/274
21     mkdir -p /var/lib/postgresql; \
22     chown -R postgres:postgres /var/lib/postgresql
23
24 # grab gosu for easy step-down from root
25 ENV GOSU_VERSION 1.10
26 RUN set -x \
27     && apt-get update && apt-get install -y --no-install-recommends ca-certificates wget && rm -rf /var/lib/apt/lists/* \
28     && wget -O /usr/local/bin/gosu "https://github.com/tianon/gosu/releases/download/$GOSU_VERSION/gosu-$(dpkg --print-arch)" \
29     && wget -O /usr/local/bin/gosu.asc "https://github.com/tianon/gosu/releases/download/$GOSU_VERSION/gosu-$(dpkg --print-arch).asc" \
30     && export GNUPGHOME="$(mktemp -d)" \
31     && gpg --keyserver ha.pool.sks-keyservers.net --recv-keys B42F6819007F00F88E364FD4036A9C25BF357DD4 \
32     && gpg --batch --verify /usr/local/bin/gosu.asc /usr/local/bin/gosu \
33     && { command -v gpgconf > /dev/null && gpgconf --kill all || ; } \

```

Isi dari Dockerfile

Dockerfile benar-benar bahasa yang khusus hanya untuk docker, bukanlah :

1. Shell Script
2. Bahasa programming

Perlu dipahami juga bahwa setiap command yang dijalankan di Dockerfile (yang berwarna merah pada screenshot diatas) dibaca dari atas kebawah, dan masing-masing command tersebut mewakili 1 buah layer image.

Apa maksud dari masing-masing command tersebut serta bagaimana cara membacanya? Kita akan coba jelaskan langsung dari file contoh di materi berikut.

8.6.2. Memahami Dockerfile dari contoh

Kita coba buka Dockerfile dari folder dockerfile-contoh-1 yang merupakan Dockerfile dari official image nginx :

```
# NOTE: this example is taken from the default Dockerfile for the official nginx
# Docker Hub Repo
# https://hub.docker.com/_/nginx/

FROM debian:stretch-slim
# all images must have a FROM
# usually from a minimal Linux distribution like debian or (even better) alpine
# if you truly want to start with an empty container, use FROM scratch

ENV NGINX_VERSION 1.13.6-1~stretch
ENV NJS_VERSION 1.13.6.0.1.14-1~stretch
# optional environment variable that's used in later lines and set as envvar
# when container is running

RUN apt-get update \
    && apt-get install --no-install-recommends --no-install-suggests -y gnupg1
\
    && \
```




```

NGINX_GPGKEY=573BFD6B3D8FBC641079A6ABABF5BD827BD9BF62; \
found=''; \
for server in \
    ha.pool.sks-keyservers.net \
    hkp://keyserver.ubuntu.com:80 \
    hkp://p80.pool.sks-keyservers.net:80 \
    pgp.mit.edu \
; do \
    echo "Fetching GPG key $NGINX_GPGKEY from $server"; \
    apt-key adv --keyserver "$server" --keyserver-options timeout=10 --\
recv-keys "$NGINX_GPGKEY" && found=yes && break; \
done; \
test -z "$found" && echo >&2 "error: failed to fetch GPG key\n$NGINX_GPGKEY" && exit 1; \
apt-get remove --purge -y gnupg1 && apt-get -y --purge autoremove && rm -\
rf /var/lib/apt/lists/* \
&& echo "deb http://nginx.org/packages/mainline/debian/ stretch nginx"\
>> /etc/apt/sources.list \
&& apt-get update \
&& apt-get install --no-install-recommends --no-install-suggests -y \
    nginx=${NGINX_VERSION} \
    nginx-module-xslt=${NGINX_VERSION} \
    nginx-module-geoip=${NGINX_VERSION} \
    nginx-module-image-filter=${NGINX_VERSION} \
    nginx-module-njs=${NJS_VERSION} \
    gettext-base \
    && rm -rf /var/lib/apt/lists/*
# optional commands to run at shell inside container at build time
# this one adds package repo for nginx from nginx.org and installs it

RUN ln -sf /dev/stdout /var/log/nginx/access.log \
    && ln -sf /dev/stderr /var/log/nginx/error.log
# forward request and error logs to docker log collector

EXPOSE 80 443
# expose these ports on the docker virtual network

```



```
# you still need to use -p or -P to open/forward these ports on host
```

```
CMD ["nginx", "-g", "daemon off;"]
```

```
# required: run this command when container is launched
```

```
# only one CMD allowed, so if there are multiple, last one wins
```

Penjelasan :

- FROM merupakan command untuk menentukan basis image ini menggunakan image apa. Contohnya untuk image nginx ini berbasis image salah satu distro Linux yaitu Debian Jessie. Setiap Dockerfile harus memiliki FROM, dan biasanya FROM ini merujuk pada distro Linux yang super kecil seperti alpine.
- ENV untuk menentukan variable tertentu yang dapat terus dijalankan baik saat mem-build image ini maupun saat sudah menjalankan container. Seperti disini sudah ditentukan variable NGINX_VERSION adalah 1.13.6-1~stretch. Sehingga setiap ada kebutuhan untuk menyebutkan versi NGINX, dibanding untuk menulis 1.13.6-1~stretch, kita bisa langsung tulis NGINX_VERSION saja. Contohnya di Dockerfile ini ada tertulis :

```
apt-get install --no-install-recommends --no-install-suggests -y nginx=${NGINX_VERSION} \
```

Artinya perintah itu sama saja dengan :

```
apt-get install --no-install-recommends --no-install-suggests -y nginx=1.13.6-1~stretch \
```

- RUN merupakan command-command tambahan yang akan dieksekusi saat mem-build image ini. Command-command ini biasanya merupakan command-command Linux pada umumnya.
- EXPOSE ini untuk membuka suatu port di image ini. Tapi tetap harus menggunakan opsi -p pada saat menjalankan container.



- CMD merupakan command default yang akan dijalankan oleh container jika menggunakan image ini. Biasanya berisi command untuk menjalankan layanan/aplikasi tersebut. Contohnya disini command yang dijalankan : `nginx -g daemon off`

Secara garis besar konsep urutan isi Dockerfile ini mirip seperti urutan langkah-langkah instalasi dan konfigurasi suatu layanan/program di Linux pada umumnya, yaitu :

1. Instalasi dependensi-dependensi aplikasi yang dibutuhkan
2. Instalasi aplikasi/layanannya itu sendiri
3. Lakukan konfigurasi-konfigurasi tambahan agar aplikasi/layanan itu dapat berjalan
4. Jalankan aplikasi/layanan tersebut.

Jika Anda sebelumnya sudah sering berkutat dengan dunia server dan Linux pasti sudah tidak asing dengan pola seperti ini. Dockerfile official image nginx diatas pun jika diperhatikan memiliki pola yang mirip-mirip :

1. Gunakan distro Debian Jessie (FROM)
2. Tentukan versi nginx yang ingin diinstall (ENV)
3. Install dependensi-dependensi dari nginx dan install nginxnya (RUN `apt-get update, apt-get install`)
4. Lakukan konfigurasi tambahan (expose port, RUN `ln -sf`)
5. Jalankan nginx nya (CMD `["nginx", "-g", "daemon off;"]`)

Masih sangat banyak contoh-contoh penggunaan Dockerfile yang lain yang tidak akan mungkin di hafal dan cepat dipahami. Kuncinya adalah sering-sering dicoba dan sering-sering berlatih. Anda bisa coba akses link berikut untuk referensi best practice dan petunjuk penggunaan Dockerfile :

1. <https://docs.docker.com/engine/reference/builder/>



2. https://docs.docker.com/develop/develop-images/dockerfile_best-practices/

8.7. Building image : Running Docker Builds

Perintah untuk build image formatnya adalah sebagai berikut :

```
$ docker image build -t <nama image yang diinginkan>:<tag> .
```

Nb : perintah diatas hanya bisa dilakukan jika kita berada di dalam folder yang sudah terdapat Dockerfile dan nama Dockerfilenya benar-benar bernama "Dockerfile".

Contoh :

```
$ docker image build -t nginxbebas .
```

Apabila nama Dockerfilenya selain dari defaultnya, maka bisa gunakan perintah ini :

```
$ docker image build -f <nama Dockerfile> -t <nama image yang diinginkan>:<tag> .
```

Contoh :

```
$ docker image build -f dockerfile-nginx -t nginxbebas .
```

Misalnya kita coba build image nginx tersebut :

```
$ docker image build -t nginxbebas .
```

Maka kita bisa lihat bahwa akan terjadi banyak proses yang dijalankan sesuai command-command yang sudah ditentukan di dalam Dockerfile, seperti perintah RUN, perintah EXPOSE, perintah CMD.

```
Get:1 http://cdn-fastly.deb.debian.org/debian stretch/main amd64 readline-common all 7.0-3 [70.4 kB]
Get:2 http://cdn-fastly.deb.debian.org/debian stretch/main amd64 libreadline7 amd64 7.0-3 [151 kB]
Get:3 http://cdn-fastly.deb.debian.org/debian stretch/main amd64 gnupg1 amd64 1.4.21-4+deb9u1 [601 kB]
debconf: delaying package configuration, since apt-utils is not installed
Fetched 822 kB in 2s (307 kB/s)
Selecting previously unselected package readline-common.
(Reading database ... 6315 files and directories currently installed.)
Preparing to unpack .../readline-common-7.0-3_all.deb ...
Unpacking readline-common (7.0-3) ...
Selecting previously unselected package libreadline7:amd64.
Preparing to unpack .../libreadline7-7.0-3_amd64.deb ...
Unpacking libreadline7:amd64 (7.0-3) ...
Selecting previously unselected package gnupg1.
Preparing to unpack .../gnupg1-1.4.21-4+deb9u1_amd64.deb ...
Unpacking gnupg1 (1.4.21-4+deb9u1) ...
Setting up readline-common (7.0-3) ...
Setting up libreadline7:amd64 (7.0-3) ...
Setting up gnupg1 (1.4.21-4+deb9u1) ...
Processing triggers for libc-bin (2.24-11+deb9u3) ...
Fetching GPG key 573BF6D63D8FBC641079A6ABAF5B0827BD98F62 from ha.pool.sks-keyservers.net
Warning: apt-key output should not be parsed (stdout is not a terminal)
Executing: /tmp/apt-key-gpghome.BIX5vPVWz/gpg.1.sh --keyserver ha.pool.sks-keyservers.net --keyserver-options timeout=10 --recv-keys 573BF6D63D8FBC641079A6ABAF5B0827BD98F62
gpg: requesting key 7BD98F62 from hkp server ha.pool.sks-keyservers.net
gpg: key 7BD98F62: public key "nginx signing key <signing-key@nginx.com>" imported
gpg: Total number processed: 1
gpg: imported: 1 (RSA: 1)
Reading package lists...
```

Hasil menjalankan docker build

```
The image filter dynamic module for nginx has been installed.
To enable this module, add the following to /etc/nginx/nginx.conf
and reload nginx:

    load_module modules/nginx_http_image_filter_module.so;

Please refer to the module documentation for further details:
http://nginx.org/en/docs/http/nginx_http_image_filter_module.html

Processing triggers for libc-bin (2.24-11+deb9u3) ...
Removing intermediate container 9b5bd2904324
--> 4f6fea024490
Step 5/7 : RUN ln -sf /dev/stdout /var/log/nginx/access.log && ln -sf /dev/stderr /var/log/nginx/error.log
--> Running in 7e8c42091bcc
Removing intermediate container 7e8c42091bcc
--> lead3842f5d6
Step 6/7 : EXPOSE 80 443
--> Running in ba1765c1e428
Removing intermediate container ba1765c1e428
--> a7b7d6e64ca1
Step 7/7 : CMD ["nginx", "-g", "daemon off;"]
--> Running in 8c3cae0c6845
Removing intermediate container 8c3cae0c6845
--> a9d19e4df2f4
Successfully built a9d19e4df2f4
Successfully tagged nginxbebas:latest
rizal@rizal-Inspiron-5460 ~/Documents/DokumenCILSYsupport/Draft Proposal/Kuasai.id/Kuasai Docker/modul/Pertemuan 6/dockerfile-contoh-1 $
```

Hasil menjalankan docker build

Setelah proses build selesai, cobalah ulangi perintah build yang sama dan perhatikan apa yang terjadi :

```
$ docker image build -t nginxbebas .
```

Proses yang tadinya cukup lama, sekarang hanya berjalan tidak sampai 1 detik. Kenapa hal ini bisa terjadi?

Sekarang coba perhatikan gambar dibawah ini :

```
Step 1/7 : FROM debian:stretch-slim
--> 44e19a16bde1
Step 2/7 : ENV NGINX_VERSION 1.13.6-1-stretch Window Help
--> Using cache
--> 59aa2677912a
Step 3/7 : ENV NJS_VERSION 1.13.6.0.1.14-1-stretch
--> Using cache
--> 3ccced8c32f3
Step 4/7 : RUN apt-get update && apt-get install --no-install-recommends --no-install-suggests -y gnupg1 && NGINX_GPGKEY=573BFD6B3D8FBC641079A6ABAF5BD827BD
9BF62; found=""
for server in
ha.pool.sks-keyservers.net
hkp://keyserver.ubuntu.com:80
p
gp.mit.edu
do
echo "Fetching GPG key $NGINX_GPGKEY from $server";
apt-key adv --keyserver "$server" --keyserver-options timeout=10 --recv-
keys "$NGINX_GPGKEY" && found=yes && break;
done;
test -z "$found" && echo >62 "error: failed to fetch GPG key $NGINX_GPGKEY" && exit 1;
apt-get remove --purge -
y gnupg1 && apt-get -y --purge autoremove && rm -rf /var/lib/apt/lists/* && echo "deb http://nginx.org/packages/mainline/debian/ stretch nginx" >> /etc/apt/sourc
es.list
&& apt-get update
&& apt-get install --no-install-recommends --no-install-suggests -y
perhatikan apa
nginx=${NGINX_VE
RSION)
nginx-module-image-filter=${NGINX_VERSION}
nginx-module-njs=${NJS_VERSION}
nginx-module-geoip=${NGINX_VE
n
g
ettext-base && rm -rf /var/lib/apt/lists/*
--> Using cache
--> 4f6fea024490
Step 5/7 : RUN ln -sf /dev/stdout /var/log/nginx/access.log && ln -sf /dev/stderr /var/log/nginx/error.log
--> Using cache
--> lead3842f5d6
Step 6/7 : EXPOSE 80 443
--> Using cache
--> a7b7d6e64ca1
Step 7/7 : CMD ["nginx", "-g", "daemon off;"]
--> Using cache
--> a9d19e4df2f4
```

Terlihat bahwa pada setiap command diatas muncul tulisan "Using Cache". Ini artinya command tersebut masih merupakan layer image yang sama dengan yang tersimpan di local sehingga tidak perlu dieksekusi kembali.

Sebuah image layer dinyatakan sama ketika :

1. Isi commandnya persis sama
2. Urutan commandnya sama



Misalnya saja kita coba ubah isi dari Dockerfile yang bagian EXPOSE menjadi 80 443 8080.

```
EXPOSE 80 443 8080
# expose these ports on the docker virtual network
# you still need to use -p or -P to open/forward these ports on host
```

Maka ketika kita coba lakukan build ulang imagenya, bagian command tersebut tidak akan menggunakan cache lagi. Tapi benar-benar di eksekusi ulang. Terlihat dari tidak adanya tanda “Using Cache” disana.

```
Step 3/7 : ENV NJS_VERSION 1.13.6.0.1.14-1-stretch
--> Using cache
--> 3cce08a2f2
Step 4/7 : RUN apt-get update && apt-get install --no-install-recommends --no-install-suggests -y gnupg1 && NGINX_GPGKEY=5738FD6B3D8F8C641079A6ABAF580827BD
9BF62; found=''; for server in ha.pool.sks-keyservers.net hkp://keyserver.ubuntu.com:80 hkp://p80.pool.sks-keyservers.net:80 p
gp.mit.edu ; do echo "Fetching GPG key $NGINX_GPGKEY from $server"; apt-key adv --keyserver "$server" --keyserver-options timeout=10 --recv-
keys "$NGINX_GPGKEY" && found=yes && break; done; test -z "$found" && echo >&2 "error: failed to fetch GPG key $NGINX_GPGKEY" && exit 1; apt-get remove --purge -
y gnupg1 && apt-get -y --purge autoremove && rm -rf /var/lib/apt/lists/* && echo "deb http://nginx.org/packages/mainline/debian/ stretch nginx" >> /etc/apt/sourc
es.list && apt-get update && apt-get install --no-install-recommends --no-install-suggests -y nginx=${NGINX_VE
RSION} nginx-module-xslt=${NGINX_VERSION} nginx-module-njs=${NJS_VERSION} nginx-module-geoip=${NGINX_VERSION} n
ginx-module-image-filter=${NGINX_VERSION}
--> Using cache
--> 4f6fea024490
Step 5/7 : RUN ln -sf /dev/stdout /var/log/nginx/access.log && ln -sf /dev/stderr /var/log/nginx/error.log
--> Using cache
--> lead3842f5d6
Step 6/7 : EXPOSE 80 443 8080
--> Running in 1a4d2b4dac8e
Removing intermediate container 1a4d2b4dac8e
--> 969309364009
Step 7/7 : CMD ["nginx", "-g", "daemon off;"]
--> Running in 123304aebb0
Removing intermediate container 123304aebb0
--> 2cc7c83c7d01
Successfully built 2cc7c83c7d01
Successfully tagged nginxbebas:latest
```

Hasil build setelah diubah expose

Uniknya lagi, ketika kita mengubah sebuah command, maka command yang berada dibawahnya semuanya akan dieksekusi ulang walaupun kita tidak merubahnya. Bisa kita lihat bahwa command CMD juga sudah tidak menggunakan Cache lagi.

Hal ini merupakan mekanisme keamanan default dari Docker untuk memastikan bahwa command yang baru saja diubah tidak mempengaruhi command-command dibawahnya.

Kesimpulannya sebaiknya pastikan command-command yang bagian teratas dari Dockerfile merupakan command-command yang paling sedikit diubah, dan yang bagian bawah merupakan command-command yang paling sering berubah. Hal ini perlu dipahami dengan baik agar proses build image dapat tetap dijalankan dengan cepat dan tidak memakan resource data yang tidak perlu.




```

Step 1/7 : FROM debian:stretch-slim
--> 44e19a16bde1
Step 2/7 : ENV NGINX_VERSION 1.13.6-1-stretch
--> Using cache
--> 59aa2677912a
Step 3/7 : ENV NJS_VERSION 1.13.6.0.1.14-1-stretch
--> Using cache
--> 3cceed8e32ff3
Step 4/7 : RUN apt-get update && apt-get install --no-install-recommends --no-install-suggests -y gnupg1 && NGINX_GPGKEY=5738FD6B3D8F8C641079A6ABAF58D827BD
9BF62; found=""; for server in ha.pool.sks-keyservers.net hkp://keyserver.ubuntu.com:80 hkp://p80.pool.sks-keyservers.net:80 p
gp.mit.edu ; do echo "Fetching GPG key $NGINX_GPGKEY from $server"; apt-key adv --keyserver "$server" --keyserver-options timeout=10 --recv-
keys "$NGINX_GPGKEY" && found=yes && break; done; test -z "$found" && echo >&2 "error: failed to fetch GPG key $NGINX_GPGKEY" && exit 1; apt-get remove --purge -
y gnupg1 && apt-get -y --purge autoremove && rm -rf /var/lib/apt/lists/* && echo "deb http://nginx.org/packages/mainline/debian/ stretch nginx" >> /etc/apt/sourc
es.list && apt-get update && apt-get install --no-install-recommends --no-install-suggests -y perhatikan apa nginx=${NGINX_VE
RSION} nginx-module-image-filter=${NGINX_VERSION} nginx-module-xslt=${NGINX_VERSION} nginx-module-geoip=${NGINX_VERSION} n
ettxt-base && rm -rf /var/lib/apt/lists/*
--> Using cache
--> 4f6fea024490
Step 5/7 : RUN ln -sf /dev/stdout /var/log/nginx/access.log && ln -sf /dev/stderr /var/log/nginx/error.log
--> Using cache
--> lead3842f5d6
Step 6/7 : EXPOSE 80 443
--> Using cache
--> a7bf0de64ca1
Step 7/7 : CMD ["nginx", "-g", "daemon off;"]
--> Using cache
--> a9d19e4df24

```

Command Dockerfile

8.8. Building Image : Extending Official Image

Sering kali official image pada akhirnya tidak bisa memenuhi kebutuhan kita. Misalnya kita selain membutuhkan webserver nginx, tapi kita juga membutuhkan nginx ini ditambahkan dengan file html dari web perusahaan kita. Atau misalnya image mysql perlu ditambah dengan database dari aplikasi kita.

Oleh karena itu biasanya kita menggunakan Official Image sebagai basis dari image custom yang akan kita buat sendiri.

8.8.1. Build Official Image yang sudah dimodifikasi

Disini kita akan coba untuk menggunakan official image dari nginx kemudian menambah sebuah file index.html kedalamnya. Hal ini sebagai simulasi dasar ketika kita ingin membangun sebuah webserver + file-file web didalamnya.

Bukalah folder **dockerfile-contoh-2**, disana tersedia 2 buah file yaitu Dockerfile dan **index.html**. Kita tidak perlu tahu isi dari file html tersebut, kita juga tidak perlu tahu bahasa html, tugas kita sebagai orang infrastruktur/devops adalah mengotak-atik Dockerfile nya.

Ini lumrah terjadi karena nanti biasanya para Developer hanya akan memberikan Anda folder semacam ini. Tugas kita adalah memastikan folder aplikasi ini bisa dijadikan image dan dijalankan kedalam container dengan baik.



Sekarang kita coba lihat isi dari Dockerfilenya :

```
# this shows how we can extend/change an existing official image from Docker Hub

FROM nginx:latest
# highly recommend you always pin versions for anything beyond dev/learn

WORKDIR /usr/share/nginx/html
# change working directory to root of nginx webhost
# using WORKDIR is preferred to using 'RUN cd /some/path'

COPY index.html index.html
# replace file index.html original with index.html custom
```

Penjelasan :

Pada bagian FROM kita menggunakan official image nginx:latest. Sehingga kita sudah tidak perlu mengurus bagian instalasi dependensi nginx, expose port, dan lain-lainnya seperti yang sudah ada pada Dockerfile official image nginx.

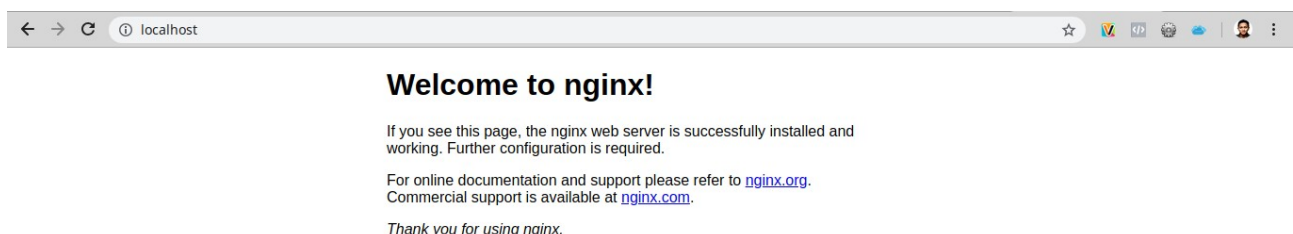
Seperti yang sudah dijelaskan pada bagian komentar (#) diatas, bagian WORKDIR seperti menjalankan perintah pindah direktori “cd”. Kenapa pindah ke direktori /usr/share/nginx/html ? Karena itu adalah folder default tempat meletakkan file-file web untuk nginx.

Setelah kita masuk ke direktori /usr/share/nginx/html, maka kita kopikan file index.html custom yang kita miliki ke dalam folder tersebut.

Jika sudah cukup mengerti, keluarlah dari file Dockerfile tersebut. Sekarang sebelum kita mencoba untuk mem-build image ini dan menjalankannya, kita coba jalankan ulang sebuah container nginx yang default untuk nanti melihat perbedaannya :

```
# docker container run -p 80:80 --rm nginx
```

Lihat hasilnya di browser :



Hasil nginx

Tekan CTRL + C untuk menghentikan container tersebut. Baru berikutnya kita coba untuk build image yang baru dengan nama nginx-custom :

```
# docker image build -t nginx-custom .
```

Nb : pastikan saat melakukan build, Anda sudah berada didalam folder **dockerfile-contoh-2**

Coba lihat, pada proses build image nginx-custom ini, tidak ada lagi proses download dan instalasi yang panjang seperti sebelumnya. Karena kita sudah memiliki image nginx:latest di local dengan image layer yang sama, dan disini yagn terjadi adalah hanya proses mengkopi file index.htmlnya saja.

```
rizal@rizal-Inspiron-5468 ~/Documents/DokumenCILSYSupport/Draft Proposal/Kuasai.id/Kuasai Docker/modul/Pertemuan 6/dockerfile-contoh-2 $ docker image build -t nginx-custom .
Sending build context to Docker daemon 3.072kB
Step 1/3 : FROM nginx:latest
--> bc26f1ed35cf
Step 2/3 : WORKDIR /usr/share/nginx/html
--> Running in 3158d9136c92
Removing intermediate container 3158d9136c92
--> 489c16ca280b
Step 3/3 : COPY index.html index.html
--> 51219c82d206
Successfully built 51219c82d206
Successfully tagged nginx-custom:latest
```

Jika sudah berhasil di build, kita coba untuk menjalankan container ini :

```
# docker container run -p 80:80 --rm nginx-custom
```

Lalu lihat hasilnya di browser :



Hasil custom dockerfile nginx

8.9. Study Case: Writing Dockerfiles

Kalau Anda melihat docker file diatas, pasti pusing, bukan? Apalagi kita masih dalam tahap belajar. Pada bagian ini, kita akan mempelajari beberapa studi kasus membuat Dockerfile yang sederhana.



8.9.1. Menambahkan Landing Page ke Container NGINX

Masih ingat landing page yang ada di materi Linux sebelumnya? Untuk membuat NGINX menampilkan landing page tersebut tentunya ada beberapa konfigurasi yang perlu dilakukan. Kali ini kita akan mulai belajar menyusun Dockerfile dari hal yang sederhana, yaitu meng-*inject* file html dan php ke image NGINX.

Langkah pertama, kita clone terlebih dahulu file landing page yang akan dimasukkan ke dalam *image* Docker. Apabila Anda telah memiliki file ini, Anda bisa langsung ke langkah selanjutnya.

```
$ git clone https://github.com/sdcilsy/landing-page.git
```

Untuk membuat Dockerfile, kita perlu membuat direktori khusus dimana file *source code* disimpan di direktori yang sama dengan Dockerfile. Selanjutnya, kita akan melakukan *pull* terhadap image nginx dengan versi alpine dengan perintah berikut.

```
$ docker pull nginx:alpine
```

```
fakhri37@fakhri37-ThinkPad-L450:~/Documents/Project/docker$ docker pull nginx:alpine
alpine: Pulling from library/nginx
4167d3e14976: Pull complete
db94a93dfca0: Pull complete
Digest: sha256:9e81b8f9cef5a095f892183688798a5b2c368663276aa0f2be4b1cd283ace53d
Status: Downloaded newer image for nginx:alpine
docker.io/library/nginx:alpine
fakhri37@fakhri37-ThinkPad-L450:~/Documents/Project/docker$
```

Kemudian kita edit Dockerfile dengan menambahkan konfigurasi berikut.

```
FROM nginx:alpine
COPY landing-page/ /usr/share/nginx/html/
```



```
fakhri37@fakhri37-ThinkPad-L450: ~/Documents/Project/docker
File Edit View Search Terminal Help
GNU nano 2.9.3 Dockerfile

FROM nginx:alpine
COPY landing-page/ /usr/share/nginx/html/

[ Read 2 lines ]
^G Get Help  ^O Write Out  ^W Where Is   ^K Cut Text   ^J Justify    ^C Cur Pos    M-U Undo
^X Exit      ^R Read File  ^\ Replace    ^U Uncut Text ^T To Spell   ^_ Go To Line  M-E Redo
```

Penjelasan:

- FROM nginx:alpine, artinya kita menggunakan image nginx versi alpine yang telah di-pull sebelumnya.
- COPY landing-page /usr/share/nginx/html/, artinya kita melakukan penyalinan file-file yang ada di direktori landing-page ke direktori /usr/share/nginx/html/ yang ada pada image NGINX nantinya.

Setelah itu, lakukan perintah docker build untuk membuat docker image yang telah ditambahkan file landing page diatas. Lakukan perintah berikut:

```
$ docker image build -t ngingx-lp:1.0 .
```

```
fakhri37@fakhri37-ThinkPad-L450:~/Documents/Project/docker$ docker build -t nginx-lp:1.0 .
Sending build context to Docker daemon 21.58MB
Step 1/2 : FROM nginx:alpine
--> 48c8a7c47625
Step 2/2 : COPY landing-page/ /usr/share/nginx/html/
--> Using cache
--> 03296e8a0e3f
Successfully built 03296e8a0e3f
Successfully tagged nginx-lp:1.0
fakhri37@fakhri37-ThinkPad-L450:~/Documents/Project/docker$
```



Setelah berhasil melakukan build image, coba jalankan image tersebut dengan perintah berikut.

```
$ docker run --rm -it -p 8080:80 nginx-lp:1.0
```

Selanjutnya, kita akses menggunakan web browser dengan alamat

<http://localhost:8080>



8.9.2. Membuat Dockerfile untuk Aplikasi Python

Aplikasi yang menggunakan Python saat ini memang sedang cukup *hype* lagi, karena adanya trend *Artificial Intelligence* dan *Machine Learning*, ditambah adanya data science, yang menggunakan Python untuk menjalankan aplikasi mereka. Pada studi kasus kali ini, kita akan membuat Dockerfile untuk mendeploy aplikasi Python.

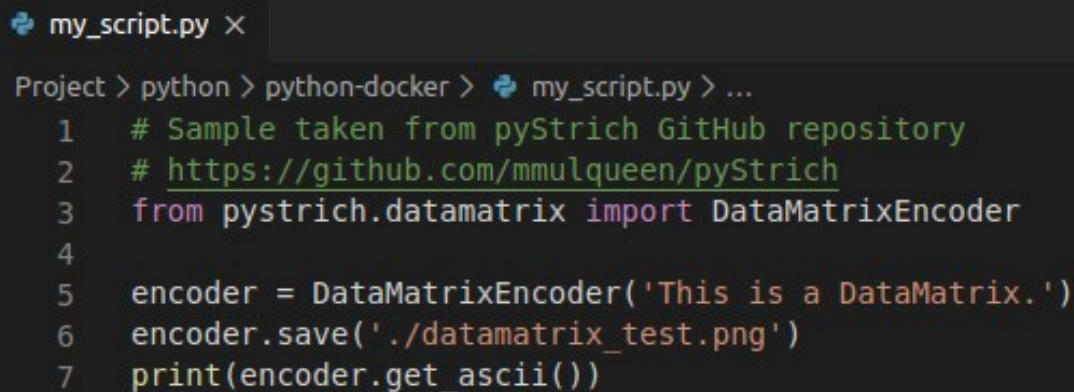
Langkah awal yang perlu dilakukan sama seperti sebelumnya, yaitu membuat direktori yang nantinya akan diisi file aplikasi Python dan Dockerfile. Kemudian, kita siapkan terlebih dahulu aplikasi Python yang akan dijalankan. Nama aplikasi yang akan kita jalankan bernama `my_script.py`.

```
$ nano my_script.py
```

```
# Sample taken from pyStrich GitHub repository  
# https://github.com/mmulqueen/pyStrich
```

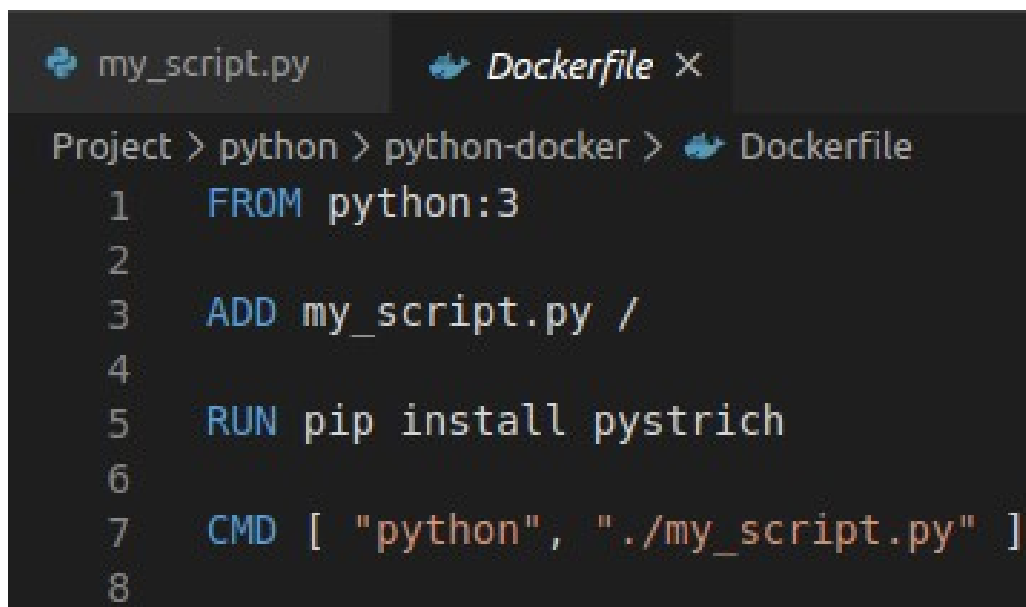
```
from pystrich.datamatrix import DataMatrixEncoder

encoder = DataMatrixEncoder('This is a DataMatrix.')
encoder.save('./datamatrix_test.png')
print(encoder.get_ascii())
```



```
my_script.py ×
Project > python > python-docker > my_script.py > ...
1 # Sample taken from pyStrich GitHub repository
2 # https://github.com/mmulqueen/pyStrich
3 from pystrich.datamatrix import DataMatrixEncoder
4
5 encoder = DataMatrixEncoder('This is a DataMatrix.')
6 encoder.save('./datamatrix_test.png')
7 print(encoder.get_ascii())
```

Setelah itu, kita buat Dockerfile pada direktori yang sama.



```
my_script.py Dockerfile ×
Project > python > python-docker > Dockerfile
1 FROM python:3
2
3 ADD my_script.py /
4
5 RUN pip install pystrich
6
7 CMD [ "python", "./my_script.py" ]
8
```

Penjelasan:

FROM python:3 → Command ini berarti kita akan menggunakan image python:3

ADD my_script.py / → Fungsi ini bertujuan untuk memasukkan my_script.py ke dalam image Python dengan path /.



RUN pip install pystrich → Berfungsi untuk melakukan instalasi pystrich pada Python, yang merupakan dependencies module untuk menjalankan my_script.py

CMD ["python", "./my_script.py"] → bertujuan untuk menjalankan command ketika image di-load.

Setelah itu, save Dockerfile.

Kemudian lakukan compile menggunakan perintah berikut.

```
docker build -t python-barcode .
```

Setelah selesai, kita akan coba menjalankan image python-barcode dengan menggunakan perintah berikut.

```
docker run python-barcode
```

```
fakhri37@fakhri37-ThinkPad-L450:~/Documents/Project/python/python-docker$ docker run python-barcode

XX XX XX XX XX XX XX XX XX
XX XX XX XXXX XX XX XX XXXX
XXXX XX XX XXXX XXXXXX XX
XX XXXX XXXX XXXX XX XX
XX XX XXXX XX XX
XXXXXXXX XX XX XX XXXX XXXXXX XXXX
XXXX XX XXXX XXXX XX XX
XXXX XX XXXXXX XXXX XXXXXX
XX XX XX XXXXXX XX XX XX
XX XX XXXX XX XXXX XXXXXX XX
XXXX XX XX XXXX XXXXXX XX
XX XX XXXXXX XX XX XXXX XX XX
XX XX XXXX XX XXXXXXXX
XXXXXXXX XXXXXX XXXX XXXXXXXXXXXXXX XX
XX XXXXXX XX XX XXXXXXXX
XX XX XX XX XX XXXXXXXXXXXX XX
XX XXXXXXXX XX XXXXXX XX XX XX
XX XX XXXX XX XXXX XX XX
XXXX XX XXXX XX XX XXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Kita akan melihat hasil generate ASCII QR code .



8.9.3. Membuat Dockerfile untuk Aplikasi GoLang

Golang (atau biasa disebut dengan Go) adalah bahasa pemrograman baru yang dikembangkan di Google oleh Robert Griesemer, Rob Pike, dan Ken Thompson pada tahun 2007 dan mulai diperkenalkan di publik tahun 2009. Penciptaan bahasa Go didasari bahasa C dan C++, oleh karena itu gaya sintaks-nya mirip.

Saat ini, cukup banyak startup dan industri yang menggunakan Golang dalam proses development aplikasi mereka. Maka dari itu, tidak ada salahnya kita belajar mengenai membuat Dockerfile untuk aplikasi dengan bahasa Go.

Pertama-tama, buatlah sebuah file bernama `hello_server.go` yang berisi script sebagai berikut.

```
package main

import (
    "context"
    "fmt"
    "log"
    "net/http"
    "os"
    "os/signal"
    "syscall"
    "time"

    "github.com/gorilla/mux"
    "gopkg.in/natefinch/lumberjack.v2"
)

func handler(w http.ResponseWriter, r *http.Request) {
    query := r.URL.Query()
    name := query.Get("name")
```



```

    if name == "" {
        name = "Guest"
    }
    log.Printf("Received request for %s\n", name)
    w.Write([]byte(fmt.Sprintf("Hello, %s\n", name)))
}

func main() {
    // Create Server and Route Handlers
    r := mux.NewRouter()

    r.HandleFunc("/", handler)

    srv := &http.Server{
        Handler:      r,
        Addr:         ":8080",
        ReadTimeout:  10 * time.Second,
        WriteTimeout: 10 * time.Second,
    }

    // Configure Logging
    LOG_FILE_LOCATION := os.Getenv("LOG_FILE_LOCATION")
    if LOG_FILE_LOCATION != "" {
        log.SetOutput(&lumberjack.Logger{
            Filename:   LOG_FILE_LOCATION,
            MaxSize:   500, // megabytes
            MaxBackups: 3,
            MaxAge:    28, //days
            Compress:  true, // disabled by default
        })
    }

    // Start Server
    go func() {
        log.Println("Starting Server")
    }()
}

```




```

        if err := srv.ListenAndServe(); err != nil {
            log.Fatal(err)
        }
    }()

    // Graceful Shutdown
    waitForShutdown(srv)
}

func waitForShutdown(srv *http.Server) {
    interruptChan := make(chan os.Signal, 1)
    signal.Notify(interruptChan, os.Interrupt, syscall.SIGINT,
syscall.SIGTERM)

    // Block until we receive our signal.
    <-interruptChan

    // Create a deadline to wait for.
    ctx, cancel := context.WithTimeout(context.Background(), time.Second*10)
    defer cancel()
    srv.Shutdown(ctx)

    log.Println("Shutting down")
    os.Exit(0)
}

```

Siapkan juga file dependencies berupa file go.mod dengan isi script sebagai berikut.

```

module github.com/callicoder/go-docker

require (
    github.com/gorilla/context v1.1.1
    github.com/gorilla/mux v1.6.2
    gopkg.in/natefinch/lumberjack.v2 v2.0.0-20170531160350-a96e63847dc3
)

```



Kemudian, buatlah file go.sum dengan isi script sebagai berikut

```
github.com/gorilla/context v1.1.1/go.mod
h1:kBGZzfjB9CEq2AlWe17Uuf7NDRt0dE0s8S51q0aT7Yg=

github.com/gorilla/mux v1.6.2 h1:Pgr17XTNXAk3q/r4CpKzC5xBM/qW1uVLV+IhRZpIIk=
github.com/gorilla/mux v1.6.2/go.mod
h1:1lud6UwP+6orDFRuTfBEV8e9/a0M/c4fVVCaMa2zaAs=

gopkg.in/natefinch/lumberjack.v2 v2.0.0-20170531160350-a96e63847dc3
h1:AFxeG48hTWHhDTQDk/m2gorfVHUEa9vo3tp3D7TzwjI=

gopkg.in/natefinch/lumberjack.v2 v2.0.0-20170531160350-a96e63847dc3/go.mod
h1:l0ndWWf7gzL7RNwBG7wST/UCcT4T24xpD6X8LsfU/+k=
```

Setelah itu, buatlah Dockerfile di direktori yang sama dengan script diatas. Isi dari Dockerfilenya adalah sebagai berikut.

```
# Dockerfile References: https://docs.docker.com/engine/reference/builder/

# Start from the latest golang base image
FROM golang

# Add Maintainer Info
MAINTANER Your Name "youremail@domain.tld"

# Set the Current Working Directory inside the container
WORKDIR /app

# Copy go mod and sum files
COPY go.mod go.sum ./

# Download all dependencies. Dependencies will be cached if the go.mod and
go.sum files are not changed
RUN go mod download

# Copy the source from the current directory to the Working Directory inside the
container
COPY . .

# Build the Go app
RUN go build -o main .
```



```
# Expose port 8080 to the outside world
EXPOSE 8080

# Command to run the executable
CMD ["/main"]
```

```
Project > docker > go lang > Dockerfile > ...
1 # Dockerfile References: https://docs.docker.com/engine/reference/builder/
2
3 # Start from the latest go lang base image
4 FROM go lang
5
6 # Add Maintainer Info
7 LABEL MAINTAINER="Muhammad Fakhri Abdillah <fakhri@cilsyfiolution.com>"
8
9 # Set the Current Working Directory inside the container
10 WORKDIR /app
11
12 # Copy go mod and sum files
13 COPY go.mod go.sum ./
14
15 # Download all dependencies. Dependencies will be cached if the go.mod and go.sum files are not changed
16 RUN go mod download
17
18 # Copy the source from the current directory to the Working Directory inside the container
19 COPY . .
20
21 # Build the Go app
22 RUN go build -o main .
23
24 # Expose port 8080 to the outside world
25 EXPOSE 8080
26
27 # Command to run the executable
28 CMD ["/main"]
29
```

Selanjutnya, kita *build* Dockerfile diatas menggunakan perintah berikut.

```
docker build -t go-docker .
```

```
fakhri37@fakhri37-ThinkPad-L450:~/Documents/Project/docker/golang$ docker build -t go-docker .
Sending build context to Docker daemon 7.168kB
Step 1/9 : FROM go lang
----> 297e5bf50f50
Step 2/9 : LABEL MAINTAINER="Muhammad Fakhri Abdillah <fakhri@cilsyfiolution.com>"
----> Using cache
----> 1c0d2139fda5
Step 3/9 : WORKDIR /app
----> Using cache
----> c88f7b670bd7
Step 4/9 : COPY go.mod go.sum ./
----> Using cache
----> 56214636d368
Step 5/9 : RUN go mod download
----> Using cache
----> 9168b85fc516
Step 6/9 : COPY . .
----> Using cache
----> 076b22136d47
Step 7/9 : RUN go build -o main .
----> Using cache
----> b64cae1ac367
Step 8/9 : EXPOSE 8080
----> Using cache
----> c6d24dfb7c08
Step 9/9 : CMD ["/main"]
----> Using cache
----> b8e519a80ede
Successfully built b8e519a80ede
Successfully tagged go-docker:latest
```



Untuk mengetes docker images yang telah dibuat, Anda dapat menggunakan perintah berikut.

```
$ docker run -d -p 8080:8080 go-docker
```

Untuk mencoba aplikasi yang telah dijalankan, kita coba menggunakan perintah berikut.

```
$ curl http://localhost:8080?name=SDCilisy
```

```
Hello, SDCilisy
```

```
fakhri37@fakhri37-ThinkPad-L450:~/Documents/Project/docker/golang$ curl http://localhost:8080?name=SDCilisy
Hello, SDCilisy
fakhri37@fakhri37-ThinkPad-L450:~/Documents/Project/docker/golang$
```

kita juga dapat melakukan pengujian lewat web browser dengan menggunakan alamat <http://localhost:8080>, sehingga akan muncul tampilan sebagai berikut.



8.10. Push ke Docker Hub

Untuk melakukan push ke Docker Hub, seperti biasa kita harus mengubah tagging image ini menjadi format :

```
<nama user di Docker hub>/<nama image>:<tag>
```

Berikut adalah contohnya (disini kita menggunakan tag latest saja) :



```
$ docker image tag nginx-custom:latest cilsy/nginx-custom:latest
```

Setelah sudah diubah tagnya, langsung lakukan push dengan perintah berikut :

```
$ docker image push cilsy/nginx-custom
```



Sampai tahap ini seharusnya image anda sudah masuk ke Docker Hub. Sebagai testing, kita akan coba menghapus image nginx-custom dari local untuk nantinya kita akan coba mengambil ulang image ini dari Docker Hub.

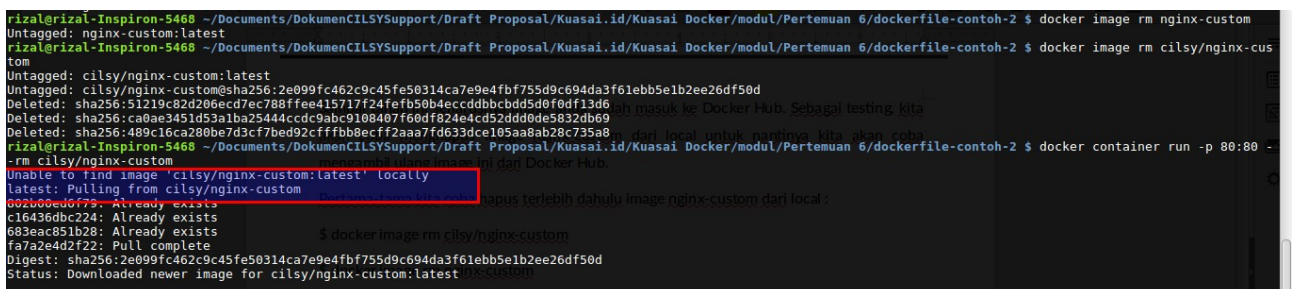
Pertama-tama kita coba hapus terlebih dahulu image nginx-custom dari local :

```
$ docker image rm cilsy/nginx-custom
```

```
$ docker image rm nginx-custom
```

Setelah itu kita coba jalankan container menggunakan image ini, tapi dengan langsung pull ulang image ini dari Docker Hub :

```
$ docker container run -p 80:80 --rm cilsy/nginx-custom
```



Anda bisa coba lihat bagian yang ditandai merah, bahwa image cilsy/nginx-custom sudah tidak dapat ditemukan di local dan Docker akan mengambilnya dari Docker Hub. Setelah running, pastikan ketika Anda test di browser, container ini tetap membuka halaman custom nginx seperti sebelumnya.



8.11. Exercise

Teori

1. Menurut Anda seperti apa proses yang terjadi jika kita menjalankan Container menggunakan images yang sudah ada di local?

Praktek

1. Soal 1

1. Diasumsikan Anda adalah Devops di suatu perusahaan. Salah satu programmer meminta bantuan Anda untuk menjalankan aplikasi berbasis Node JS milik perusahaan agar bisa berjalan di Docker.
2. Anda hanya diberikan file aplikasi Node JS beserta petunjuk requirement agar aplikasi ini bisa berjalan. Hal ini lumrah terjadi. Dimana kita sebagai orang infrastruktur sama sekali tidak perlu tahu Node JS itu apa, bagaimana codingnya, namun kita cukup tahu untuk bagaimana menjalankan aplikasi ini di server.
3. Seluruh file aplikasi sudah tersedia di dalam folder dockerfile-exercise-1. Anda tidak perlu mengedit file manapun kecuali file Dockerfile.
4. Bukalah Dockerfile dan ikuti petunjuk untuk membangun aplikasi ini.
5. Hasil akhirnya kita dapat mengakses hasil web apps node JS ini pada browser : <http://localhost>
6. Setelah berhasil berjalan, push image ini ke Dockerhub dengan nama bebas.
7. Hapus image dari local. Kemudian jalankan ulang container menggunakan image yang sudah Anda upload di Dockerhub.
8. Hal-hal yang perlu anda perhatikan saat melakukan proses ini :
 - Proses ini iteratif. Akan sangat jarang Anda membangun 1 Dockerfile dan langsung berjalan.



- Proses iteratifnya adalah : Sesuaikan Dockerfile, Build, Test proses build berhasil, ulangi, Jalankan dalam Container. Hapus container, ulangi lagi.
- Proses membuat Dockerfile adalah proses yang paling menyenangkan dalam meng-containerisasi suatu apps. Karena salah satu kunci utama proses meng-containerisasi adalah membangun image yang tepat dan berjalan sesuai kebutuhan.
- Kuncinya adalah Docker Hub, Docker Docs, dan Googling.

2. Soal 2

Cobalah beberapa aplikasi lainnya, seperti:

- Ruby on Rails
- PHP Laravel
- Node.JS dengan aplikasi berbeda

8.12. Summary

Berikut adalah rangkuman poin-poin penting materi yang sudah dipelajari dari pertemuan kali ini :

1. Image hanya berisi dependensi, library serta metadata. Tidak ada kernel dan modul driver OS didalamnya. Image juga terdiri dari tumpukan layer, dimana layer ini diwakili oleh instruksi/command pada Dockerfile.
2. Usahakan selalu menggunakan image yang bagus. Yaitu official image atau image yang memiliki jumlah pull dan stars yang bagus.
3. Untuk bisa mengupload image ke Docker Hub harus memiliki format tag `<username di dockerhub>/<nama image>:<tag>`



4. Dockerfile adalah resep untuk membangun sebuah image, setiap command/instruksi pada Dockerfile mewakili sebuah layer image dan pastikan command-command yang teratas pada Dockerfile adalah yang paling jarang berubah, dan yang terbawah adalah yang paling sering berubah.
5. Biasanya kita menggunakan official image sebagai basis kita untuk memperoleh image yang lebih sesuai dengan kebutuhan kita.