Docker Dasar

1.Docker Image

Docker Image mirip seperti installer aplikasi, dimana di dalam Docker Image terdapat aplikasi dan dependency

Sebelum kita bisa menjalankan aplikasi di Docker, kita perlu memastikan memiliki Docker Image aplikasi tersebut

Melihat Docker Image

Untuk melihat Docker Image yang terdapat di dalam Docker Daemon, kita bisa menggunakan perintah :

# docker image ls

cara mendonwload image nya

# docker image pull redis:latest

cara menghapus image nya

# docker image rm alpine:latest

2.Docker Container

Jika Docker Image seperti installer aplikasi, maka Docker Container mirip seperti aplikasi hasil installernya

Satu Docker Image bisa digunakan untuk membuat beberapa Docker Container, asalkan nama Docker Container nya berbeda

Jika kita sudah membuat Docker Container, maka Docker Image yang digunakan tidak bisa dihapus, hal ini dikarenakan sebenarnya Docker Container tidak meng-copy isi Docker Image, tapi hanya menggunakannya isinya saja

Status Container

Saat kita membuat container, secara default container tersebut tidak akan berjalan

Mirip seperti ketika kita menginstall aplikasi, jika tidak kita jalankan, maka aplikasi tersebut tidak akan berjalan, begitu juga container

Oleh karena itu, setelah membuat container, kita perlu menjalankannya jika memang ingin menjalankan container ny

Melihat Container

Untuk melihat semua container, baik yang sedang berjalan atau tidak di Docker Daemon, kita bisa gunakan perintah :

docker container ls -a

Sedangkan jika kita ingin melihat container yang sedang berjalan saja, kita bisa gunakan perintah :

docker container ls

Membuat Container

Untuk membuat container, kita bisa gunakan perintah :

docker container create --name namacontainer namaimage:tag

Menjalankan Container

Untuk menjalankan container yang sudah kita buat, kita bisa gunakan perintah :

docker container start containerId/namacontainer

menghentikan Container

Untuk menghentikan container, kita bisa gunakan perintah :

docker container stop containerId/namacontainer

Menghapus Container

Untuk menghapus container yang sudah berhenti, kita bisa gunakan perintah :

docker container rm containerId/namacontainer

3. Container Log

Kadang saat terjadi masalah dengan aplikasi yang terdapat di container, sering kali kita ingin melihat detail dari log aplikasinya

Hal ini dilakukan untuk melihat detail kejadian apa yang terjadi di aplikasi, sehingga akan memudahkan kita ketika mendapat masalah

Melihat Container Log

Untuk melihat log aplikasi di container kita, kita bisa menggunakan perintah :

docker container logs containerId/namacontainer

Atau jika ingin melihat log secara realtime, kita bisa gunakan perintah :

docker container logs -f containerId/namacontainer

# docker container create --name contohredis redis:latest

# docker container start contohredis

# docker container logs contohredis

# docker container logs -f contohredis

4.Container Exec

Saat kita membuat container, aplikasi yang terdapat di dalam container hanya bisa diakses dari dalam container

Oleh karena itu, kadang kita perlu masuk ke dalam container nya itu sendiri

Untuk masuk ke dalam container, kita bisa menggunakan fitur Container Exec, dimana digunakan untuk mengeksekusi kode program yang terdapat di dalam container

Masuk ke Container

Untuk masuk ke dalam container, kita bisa mencoba mengeksekusi program bash script yang terdapat di dalam container dengan bantuan Container Exec :

docker container exec -i -t containerId/namacontainer /bin/bash

-i adalah argument interaktif, menjaga input tetap aktif

-t adalah argument untuk alokasi pseudo-TTY (terminal akses)

Dan /bin/bash contoh kode program yang terdapat di dalam container

contoh nya # docker container exec -i -t contohredis /bin/bash

5.Container Port

Saat menjalankan container, container tersebut terisolasi di dalam Docker

Artinya sistem Host (misal Laptop kita), tidak bisa mengakses aplikasi yang ada di dalam container secara langsung, salah satu caranya adalah harus menggunakan Container Exec untuk masuk ke dalam container nya.

Biasanya, sebuah aplikasi berjalan pada port tertentu, misal saat kita menjalankan aplikasi Redis, dia berjalan pada port 6379, kita bisa melihat port apa yang digunakan ketika melihat semua daftar container

kita buat image baru jika kita belum punya image

# docker image pull nginx:latest

cara masukan port nya

#docker container create --name contohnginx --publish 8080:80 nginx:latest

6.Port Forwarding

Docker memiliki kemampuan untuk melakukan port forwarding, yaitu meneruskan sebuah port yang terdapat di sistem Host nya ke dalam Docker Container

Cara ini cocok jika kita ingin mengekspos port yang terdapat di container ke luar melalui sistem Host nya

7.Container Environment Variable

Saat membuat aplikasi, menggunakan Environment Variable adalah salah satu teknik agar konfigurasi aplikasi bisa diubah secara dinamis

Dengan menggunakan environment variable, kita bisa mengubah-ubah konfigurasi aplikasi, tanpa harus mengubah kode aplikasinya lagi

Docker Container memiliki parameter yang bisa kita gunakan untuk mengirim environment variable ke aplikasi yang terdapat di dalam container

Menambah Environment Variable

Untuk menambah environment variable, kita bisa menggunakan perintah --env atau -e, misal :

docker container create --name namacontainer --env KEY=”value” --env KEY2=”value” image:tag

contoh nya

docker image pull mongo:latest

docker container create --name contohmongo --publish 27017:27017 --env MONGO\_INITDB\_ROOT\_USERNAME=rafli --env MONGO\_INITDB\_ROOT\_PASSWORD=rafli mongo:latest

8.Container Stats

Saat menjalankan beberapa container, di sistem Host, penggunaan resource seperti CPU dan Memory hanya terlihat digunakan oleh Docker saja

Kadang kita ingin melihat detail dari penggunaan resource untuk tiap container nya

Untungnya docker memiliki kemampuan untuk melihat penggunaan resource dari tiap container yang sedang berjalan

Kita bisa gunakan perintah :

# docker container stats

9.Container Resource Limit

Saat membuat container, secara default dia akan menggunakan semua CPU dan Memory yang diberikan ke Docker (Mac dan Windows), dan akan menggunakan semua CPU dan Memory yang tersedia di sistem Host (Linux)

Jika terjadi kesalahan, misal container terlalu banyak memakan CPU dan Memory, maka bisa berdampak terhadap performa container lain, atau bahkan ke sistem host

Oleh karena itu, ada baiknya ketika kita membuat container, kita memberikan resource limit terhadap container nya

# docker container create --name smallnginx --memory 100m --cpus 0.5 --publish 8081:80 nginx:latest

Memory

Saat membuat container, kita bisa menentukan jumlah memory yang bisa digunakan oleh container ini, dengan menggunakan perintah --memory diikuti dengan angka memory yang diperbolehkan untuk digunakan

Kita bisa menambahkan ukuran dalam bentu b (bytes), k (kilo bytes), m (mega bytes), atau g (giga bytes), misal 100m artinya 100 mega bytes

CPU

Selain mengatur Memory, kita juga bisa menentukan berapa jumlah CPU yang bisa digunakan oleh container dengan parameter --cpus

Jika misal kita set dengan nilai 1.5, artinya container bisa menggunakan satu dan setengah CPU core

Bind Mounts

Bind Mounts merupakan kemampuan melakukan mounting (sharing) file atau folder yang terdapat di sistem host ke container yang terdapat di docker

Fitur ini sangat berguna ketika misal kita ingin mengirim konfigurasi dari luar container, atau misal menyimpan data yang dibuat di aplikasi di dalam container ke dalam folder di sistem host

Jika file atau folder tidak ada di sistem host, secara otomatis akan dibuatkan oleh Docker

Untuk melakukan mounting, kita bisa menggunakan parameter --mount ketika membuat container

Isi dari parameter --mount memiliki aturan tersendiri

Melakukan Mounting

Untuk melakukan mounting, kita bisa menggunakan perintah berikut :

docker container create --name namacontainer --mount “type=bind,source=folder,destination=folder,readonly” image:tag

docker container create --name mongodata \

--publish 17018:27017 \

--mount type=bind,source=/home/rafli/mongo-data,destination=/data/db \

--env MONGO\_INITDB\_ROOT\_USERNAME=rafli \

--env MONGO\_INITDB\_ROOT\_PASSWORD=rafli \

mongo:latest

docker volume

Fitur Bind Mounts sudah ada sejak Docker versi awal, di versi terbaru direkomendasikan menggunakan Docker Volume

Docker Volume mirip dengan Bind Mounts, bedanya adalah terdapat management Volume, dimana kita bisa membuat Volume, melihat daftar Volume, dan menghapus Volume

Volume sendiri bisa dianggap storage yang digunakan untuk menyimpan data, bedanya dengan Bind Mounts, pada bind mounts, data disimpan pada sistem host, sedangkan pada volume, data di manage oleh DockerMelihat Docker Volume

Melihat Docker Volume

Saat kita membuat container, dimanakah data di dalam container itu disimpan, secara default semua data container disimpan di dalam volume

Jika kita coba melihat docker volume, kita akan lihat bahwa ada banyak volume yang sudah terbuat, walaupun kita belum pernah membuatnya sama sekali

Kita bisa gunakan perintah berikut untuk melihat daftar volume :

docker volume ls

Untuk membuat volume, kita bisa gunakan perintah :

docker volume create namavolume

membuat volume

# docker volume create mongovolume

menghapus volume

# docker volume rm mongovolume

container volume volume yang sudah kita buat.bisa kita gunakan di container

ke untungan menggunakan volume adalah,jika container kita hapus,data akan tetap aman di volume

cara menggunakan volume di container sama dengan menggunakan bind monut,kita bisa menggunakan parameter --mount,namun dengan menggunakan type volume dan source nama volume

# docker volume create mongodata

# docker container create --name mongovolume --publish 27019:27017 --mount "type=volume,source=mongodata,destination=/data/db" --env MONGO\_INITDB\_ROOT\_USERNAME=rafli --env MONGO\_INITDB\_ROOT\_PASWORD=rafli mongo:latest

Backup volume

kita bisa bisa memanfaatkan container untuk melakukan backup data yang ada di dalam volume kedalam archive seperti zip atau tar.gz

tahapan melakukan backup

matikan container yang menggunakan volumer yang ingin kita backup

buat container baru dengan dua mont,volumer yang ingin kita backup, dan bind mount folder dari sistem host

lakukan backup menggunakan container dengan cara meng-archive isi volumer,dan simpan di band mount folder

isi file backup sekarang adalah folder sistem host

delete container yang kita ginakan untuk melakuakan backup

docker container stop mongovolume

# docker container create --name nginxbackup --mount "type=bind,source=/home/rafli/backup,destination=/backup" --mount "type=volume,source=mongodata,destination=/data" nginx:latest

# docker container start nginxbackup

# docker container exec -i -t nginxbackup /bin/bash

# tar cvf /backup/backup.tar.gz /data atau klo nggak bisa kita bisa pake # tar xvf /backup/backup.tar.gz

# docker container stop nginxbackup

# docker container rm nginxbackup

# docker container start mongovolume

bagaimana cara sederhana

Menjalankan Container Secara Langsung

Melakukan backup secara manual agak sedikit ribet karena kita harus start container terlebih dahulu, setelah backup, hapus container nya lagi

Kita bisa menggunakan perintah run untuk menjalankan perintah di container dan gunakan parameter --rm untuk melakukan otomatis remove container setelah perintahnya selesai berjalan

# docker image pull ubuntu:latest

# docker container stop mongovolume

# docker container run --rm --name ubuntubackup \

--mount "type=bind,source=/home/rafli/backup,destination=/backup" \

--mount "type=volume,source=mongodata,destination=/data" \

ubuntu:latest tar xvf /backup/backup-lagi.tar.gz -C /data

data/

# docker container start mongovolume

Restore Volume

Setelah melakukan backup volume ke dalam file archive, kita bisa menyimpan file archive backup tersebut ke tempat yang lebih aman, misal ke cloud storage

Sekarang kita akan coba melakukan restore data backup ke volume baru, untuk memastikan data backup yang kita lakukan tidak corrupt

docker volume create mongorestore

docker container run --rm --name ubunturestore --mount "type=bind,source=/home/rafli/backup,destination=/backup" --mount "type=volume,source=mongorestore,destination=/data" ubuntu:latest bash -c "cd /data && tar xvf /backup/backup.tar.gz --strip 1"

docker container create --name mongorestore --publish 27020:27017 --mount "type=volume,source=mongorestore,destination=/data/db" --env MONGO\_INITDB\_ROOT\_USERNAME=eko --env MONGO\_INITDB\_ROOT\_PASSWORD=eko mongo:latest

docker container start mongorestore

Docker Network

Saat kita membuat container di docker, secara default container akan saling terisolasi satu sama lain, jadi jika kita mencoba memanggil antar container, bisa dipastikan bahwa kita tidak akan bisa melakukannya

Docker memiliki fitur Network yang bisa digunakan untuk membuat jaringan di dalam Docker

Dengan menggunakan Network, kita bisa mengkoneksikan container dengan container lain dalam satu Network yang sama

Jika beberapa container terdapat pada satu Network yang sama, maka secara otomatis container tersebut bisa saling berkomunikasi

Network Driver

Saat kita membuat Network di Docker, kita perlu menentukan driver yang ingin kita gunakan, ada banyak driver yang bisa kita gunakan, tapi kadang ada syarat sebuah driver network bisa kita gunakan.

bridge, yaitu driver yang digunakan untuk membuat network secara virtual yang memungkinkan container yang terkoneksi di bridge network yang sama saling berkomunikasi

host, yaitu driver yang digunakan untuk membuat network yang sama dengan sistem host. host hanya jalan di Docker Linux, tidak bisa digunakan di Mac atau Windows

none, yaitu driver untuk membuat network yang tidak bisa berkomunikasi

Melihat NetworkMelihat Network

Untuk melihat network di Docker, kita bisa gunakan perintah :

# docker network ls

Membuat Network

Untuk membuat network baru, kita bisa menggunakan perintah :

# docker network create --driver namadriver namanetwork

Menghapus Network

Untuk menghapus Network, kita bisa gunakan perintah :

# docker network rm namanetwork

Container Network

Setelah kita membuat Network, kita bisa menambahkan container ke network

Container yang terdapat di dalam network yang sama bisa saling berkomunikasi (tergantung jenis driver network nya)

Container bisa mengakses container lain dengan menyebutkan hostname dari container nya, yaitu nama container nya

Bagaiman cara membuat Container dengan Network

Untuk menambahkan container ke network, kita bisa menambahkan perintah --network ketika membuat container, misal :

docker container create --name namacontainer --network namanetwork image:tag

Membuat Container Dengan Network

#docker network create --driver bridge mongonetwork

# docker container create --name mongodb --network mongonetwork --env MONGO\_INITDB\_ROOT\_USERNAME=eko --env MONGO\_INITDB\_ROOT\_PASSWORD=eko mongo:latest

# docker image pull mongo-express:latest

# docker container create --name mongodbexpress --network mongonetwork --publish 8081:8081 --env ME\_CONFIG\_MONGODB\_URL="mongodb://eko:eko@mongodb:27017/" mongo-express:latest

# docker container start mongodb

# docker container start mongodbexpress

# docker network disconnect mongonetwork mongodb

# docker network connect mongonetwork mongodb

Menghapus Container dari Network

Jika diperlukan, kita juga bisa menghapus container dari network dengan perintah :

# docker network disconnect namanetwork namacontainer

# docker network disconnect mongonetwork mongodb

Menambahkan Container ke Network

Jika containernya sudah terlanjur dibuat, kita juga bisa menambahkan container yang sudah dibuat ke network dengan perintah :

# docker network connect namanetwork namacontainer

# docker network connect mongonetwork mongodb

Inspect

Setelah kita men-download image, atau membuat network, volume dan container. Kadang kita ingin melihat detail dari tiap hal tersebut

Misal kita ingin melihat detail dari image, perintah apa yang digunakan oleh image tersebut? Environment variable apa yang digunakan? Atau port apa yang digunakan?

Misal kita juga ingin melihat detail dari container, Volume apa yang digunakan? Environment variable apa yang digunakan? Port forwarding apa yang digunakan? dan lain-lain

Docker memiliki fitur bernama inspect, yang bisa digunakan di image, container, volume dan network

Dengan fitur ini, kita bisa melihat detail dari tiap hal yang ada di Docker

Menggunakan Inspect

Untuk melihat detail dari image, gunakan : docker image inspect namaimage

Untuk melihat detail dari container, gunakan : docker container inspect namacontainer

Untuk melihat detail dari volume, gunakan : docker volume inspect namavolume

Untuk melihat detail dari network, gunakan : docker network inspect namanetwork

# docker image inspect mongo-express:latest

# docker container inspect contohnginx

# docker volume inspect mongodata

Prune

Saat kita menggunakan Docker, kadang ada kalanya kita ingin membersihkan hal-hal yang sudah tidak digunakan lagi di Docker, misal container yang sudah di stop, image yang tidak digunakan oleh container, atau volume yang tidak digunakan oleh container

Fitur untuk membersihkan secara otomatis di Docker bernama prune

Hampir di semua perintah di Docker mendukung prune

Perintah Prune

Untuk menghapus semua container yang sudah stop, gunakan : docker container prune

Untuk menghapus semua image yang tidak digunakan container, gunakan : docker image prune

Untuk menghapus semua network yang tidak digunakan container, gunakan : docker network prune

Untuk menghapus semua volume yang tidak digunakan container, gunakan : docker volume prune

Atau kita bisa menggunakan satu perintah untuk menghapus container, network dan image yang sudah tidak digunakan menggunakan perintah : docker system prune

docker dockerfile

Pengenalan Dockerfile

Dockerfile Format

From Instruction

Label Instruction

Expose Instruction

Environment Variable Instruction

Entrypoint Instruction

Volume Instruction

Dan lain-lain

Pengenalan Dockerfile

Pada kelas Docker Dasar, kita sudah banyak belajar bagaimana cara kerja Docker, dari menggunakan Docker Image, sampai membuat Docker Container

Sekarang bagaimana jika kita ingin membuat Docker Image sendiri?

Pembuatan Docker Image bisa dilakukan dengan menggunakan instruksi yang kita simpan di dalam file Dockerfile

Dockerfile

Dockerfile adalah file text yang berisi semua perintah yang bisa kita gunakan untuk membuat sebuah Docker Image

Anggap saja semua instruksi untuk, menjalankan aplikasi kita, kita simpan di dalam Dockerfile, nanti Dockerfile tersebut akan dieksekusi sebagai perintah untuk membuat Docker Image

Docker Build

Untuk membuat Docker Image dari Dockerfile, kita bisa menggunakan perintah docker build.

Saat membuat Docker Image dengan docker build, nama image secara otomatis akan dibuat random, dan biasanya kita ingin menambahkan nama/tag pada image nya, kita bisa mengubahnya dengan menambahkan perintah -t

Misal berikut adalah contoh cara menggunakan docker build :

docker build -t khannedy/app:1.0.0 folder-dockerfile

docker build -t khannedy/app:1.0.0 -t rafli/app:latest folder-dockerfile

Dockerfile File Format

Seperti namanya, Dockerfile biasanya dibuat dalam sebuah file dengan nama Dockerfile, tidak memiliki extension apapun

Walaupun sebenarnya bisa saja kita membuat dengan nama lain, namu direkomendasikan menggunakan nama Dockerfile

Instruction Format

Secara sederhana berikut adalah format untuk file Dockerfile :

# Komentar

INSTRUCTION arguments

# digunakan untuk menambah komentar, kode dalam baris tersebut secara otomatis dianggap komentar

INSTRUCTION adalah perintah yang digunakan di Dockerfile, ada banyak perintah yang tersedia, dan penulisan perintahnya case insensitive, sehingga kita bisa gunakan huruf besar atau kecil. Namun rekomendasinya adalah menggunakan UPPPER CASE

Arguments adalah data argument untuk INSTRUCTION, yang menyesuaikan dengan jenis INSTRUCTION yang digunakan

From Instruction

Saat kita membuat Docker Image, biasanya perintah pertama adalah melakukan build stage dengan instruksi FROM

FROM digunakan untuk membuat build stage dari image yang kita tentukan

Biasanya, jarang sekali kita akan membuat Docker Image dari scratch (kosongan), biasanya kita akan membuat Docker Image dari Docker Image lain yang sudah ada

Untuk menggunakan FROM, kita bisa gunakan perintah :

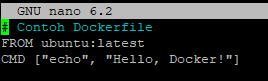
FROM image:version

# docker build -t rafli385/from .

# ls /home/rafli/Belajar-docker-dockerfile/

# cd /home/rafli/Belajar-docker-dockerfile/

# nano Dockerfile



# docker build -t rafli385/from .

# docker image ls

**Run Instruction**

* RUN adalah sebuah instruksi untuk mengeksekusi perintah di dalam image pada saat build stage.

Hasil perintah RUN akan di commit dalam perubahan image tersebut, jadi perintah RUN akan dieksekusi pada saat proses docker build saja, setelah menjadi Docker Image, perintah tersebut tidak akan dijalankan lagi.

Jadi ketika kita menjalankan Docker Container dari Image tersebut, maka perintah RUN tidak akan dijalankan lagi

**Run Instruction Format**

Perintah RUN memiliki 2 format :

RUN command

RUN [“executable”, “argument”, “...”]

**Command Instruction**

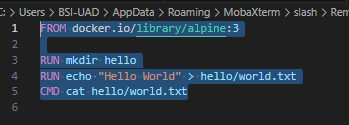
* CMD atau Command, merupakan instruksi yang digunakan ketika Docker Container berjalan
* CMD tidak akan dijalankan ketika proses build, namun dijalankan ketika Docker Container berjalan
* Dalam Dockerfile, kita tidak bisa menambah lebih dari satu instruksi CMD, jika kita tambahkan lebih dari satu instruksi CMD, maka yang akan digunakan untuk menjalankan Docker Container adalah instruksi CMD yang terakhir

**Command Instruction Format**

* Perintah CMD memiliki beberapa format :
* CMD command param param
* CMD [“executable”, “param”, “param”]
* CMD [“param”, “param”], akan menggunakan executable ENTRY POINT, yang akan dibahas di chapter terpisah

Buat folder baru dengan nama Command dan di dalam folder kita buat file Dockerfile

Edit file dengan perintah nano Dockerfile



# Ls

# docker build -t rafli385/command /home/rafli/Belajar-docker-dockerfile/Command

# cd /home/rafli/Belajar-docker-dockerfile

# docker build -t rafli385/command ./Command

# docker image inspect rafli385/Command

# docker container create --name command rafli/command

# docker container start command

# docker container logs command

**Label Instruction**

Instruksi LABEL merupakan instruksi yang digunakan untuk menambahkan metadata ke dalam Docker Image yang kita buat

Metadata adalah informasi tambahan, misal seperti nama aplikasi, pembuat, website, perusahaan, lisensidan lain-lain

Metadata hanya berguna sebagai informasi saja, tidak akan digunakan ketika kita menjalankan Docker Container

Label Instruction Format

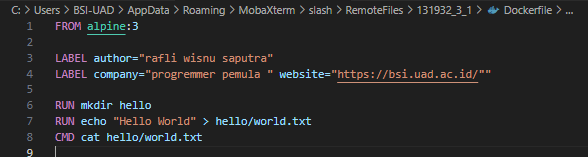
Berikut adalah format instruksi LABEL

LABEL <key>=<value>

LABEL <key1>=<value1> <key2>=<value2>

Buat folder baru di /Belajar-docker-dockerfile dengan nama label

Dan buat file baru di dalam label yang isi nya seperti in



# cd /home/rafli/Belajar-docker-dockerfile/Label docker build -t rafli385/label .

# docker build -t rafli385/label /home/rafli/Belajar-docker-dockerfile/Label

# docker image inspect rafli385/label

**Add Instruction**

ADD adalah instruksi yang dapat digunakan untuk menambahkan file dari source ke dalam folder destination di Docker Image

Perintah ADD bisa mendeteksi apakah sebuah file source merupakan file kompres seperti tar.gz, gzip, dan lain-lain. Jika mendeteksi file source adalah berupa file kompress, maka secara otomatis file tersebut akan di extract dalam folder destination

Perintah ADD juga bisa mendukung banyak penambahan file sekaligus

Penambahan banyak file sekaligus di instruksi ADD menggunakan Pattern di Go-Lang : <https://pkg.go.dev/path/filepath#Match>

**Add Instruction Format**

Instruksi ADD memiliki format sebagai berikut :

ADD source destination

Contoh :

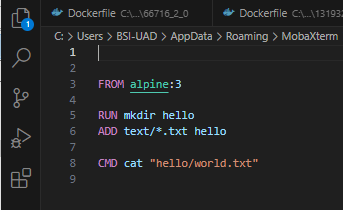
ADD world.txt hello # menambah file world.txt ke folder hello

ADD \*.txt hello # menambah semua file .txt ke folder hello

# ADD Instruction

1.buat folder baru dengan nama add

2.di dalam folder add kita buat filer Dockerfile



3.di dalam add kita buat folder baru dengan nama text

4.didalam folder text kita buat file txt contoh nya word.txt dan banyak lagi file di dalam nya



Contoh isi nya



Jika sudah lalu kita build

# cd /home/rafli/Belajar-docker-dockerfile docker build -t rafli385/add add



Kita buat container baru dengan nama rafli385

# docker container create --name add rafli385/add

Lalu kita start add nya

# docker container start add

**Agenda**

Pengenalan Dockerfile

Dockerfile Format

From Instruction

Label Instruction

Expose Instruction

Environment Variable Instruction

Entrypoint Instruction

Volume Instruction

Dan lain-lain

**Pengenalan Dockerfile**

Pada kelas Docker Dasar, kita sudah banyak belajar bagaimana cara kerja Docker, dari menggunakan Docker Image, sampai membuat Docker Container

Sekarang bagaimana jika kita ingin membuat Docker Image sendiri?

Pembuatan Docker Image bisa dilakukan dengan menggunakan instruksi yang kita simpan di dalam file Dockerfile

**Dockerfile**

Dockerfile adalah file text yang berisi semua perintah yang bisa kita gunakan untuk membuat sebuah Docker Image

Anggap saja semua instruksi untuk, menjalankan aplikasi kita, kita simpan di dalam Dockerfile, nanti Dockerfile tersebut akan dieksekusi sebagai perintah untuk membuat Docker Image

Untuk membuat Docker Image dari Dockerfile, kita bisa menggunakan perintah docker build.

Saat membuat Docker Image dengan docker build, nama image secara otomatis akan dibuat random, dan biasanya kita ingin menambahkan nama/tag pada image nya, kita bisa mengubahnya dengan menambahkan perintah -t

Misal berikut adalah contoh cara menggunakan docker build :  
  
docker build -t khannedy/app:1.0.0  folder-dockerfile  
docker build -t khannedy/app:1.0.0 -t khannedy/app:latest folder-dockerfile

**Dockerfile Format**

Seperti namanya, Dockerfile biasanya dibuat dalam sebuah file dengan nama Dockerfile, tidak memiliki extension apapun

Walaupun sebenarnya bisa saja kita membuat dengan nama lain, namu direkomendasikan menggunakan nama Dockerfile

**Instruction Format**

Secara sederhana berikut adalah format untuk file Dockerfile :  
# Komentar  
INSTRUCTION arguments

# digunakan untuk menambah komentar, kode dalam baris tersebut secara otomatis dianggap komentar

INSTRUCTION adalah perintah yang digunakan di Dockerfile, ada banyak perintah yang tersedia, dan penulisan perintahnya case insensitive, sehingga kita bisa gunakan huruf besar atau kecil. Namun rekomendasinya adalah menggunakan UPPPER CASE

Arguments adalah data argument untuk INSTRUCTION, yang menyesuaikan dengan jenis INSTRUCTION yang digunakan

**From Instruction**

Saat kita membuat Docker Image, biasanya perintah pertama adalah melakukan build stage dengan instruksi FROM

FROM digunakan untuk membuat build stage dari image yang kita tentukan

Biasanya, jarang sekali kita akan membuat Docker Image dari scratch (kosongan), biasanya kita akan membuat Docker Image dari Docker Image lain yang sudah ada

Untuk menggunakan FROM, kita bisa gunakan perintah :  
FROM image:version

https://github.com/ProgrammerZamanNow/belajar-docker-dasar/blob/main/docker-container-network.sh