Docker Dockerfile

Eko Kurniawan Khannedy

Eko Kurniawan Khannedy

- Technical architect at one of the biggest ecommerce company in Indonesia
- 11+ years experiences
- www.programmerzamannow.com
- youtube.com/c/ProgrammerZamanNow

Eko Kurniawan Khannedy

- Telegram : @khannedy
- Facebook : fb.com/ProgrammerZamanNow
- Instagram : instagram.com/programmerzamannow
- Youtube : youtube.com/c/ProgrammerZamanNow
- Telegram Channel : t.me/ProgrammerZamanNow
- Email : echo.khannedy@gmail.com

Sebelum Belajar

- Docker Dasar

Agenda

- Pengenalan Dockerfile
- Dockerfile Format
- From Instruction
- Label Instruction
- Expose Instruction
- Environment Variable Instruction
- Entrypoint Instruction
- Volume Instruction
- Dan lain-lain

Pengenalan Dockerfile

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 khannedy/app:latest folder-dockerfile

Dockerfile Format

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

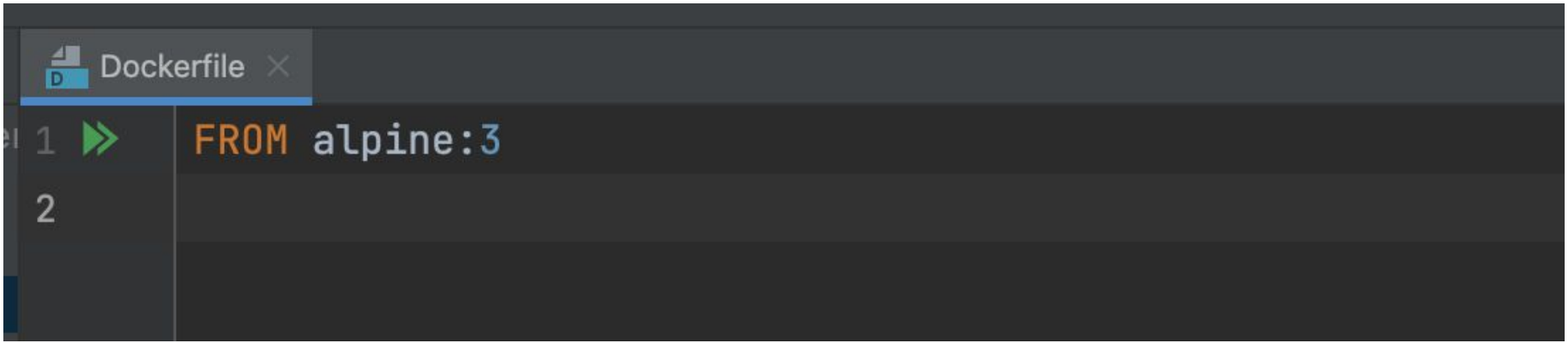
- Secara sederhana berikut adalah format untuk file Dockerfile :
  # Komentar
  INSTRUCTION arguments
- # digunakan untuk menambah komentar, kode dalam baris tersebut secara otomatis dianggap komentar
- INSTRUCTION adalah perintah yang digunakan di Dockerfile, ada banyak perintah yang tersedia, dan penulisan 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

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

Kode : FROM Instruction



Kode : Docker Build

```
→  docker-dockerfile docker build -t khannedy/simple simple
[+] Building 1.6s (5/5) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 56B
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load metadata for docker.io/library/alpine:3
 => CACHED [1/1] FROM docker.io/library/alpine:3@sha256:4edbd2beb5f78b1014028f4fbb99f3237d9561100b6881aabbf5acce2c4f9454
 => exporting to image
 => => exporting layers
 => => writing image sha256:04ede911b5cbcec2a9d42c1c0eaa867ce88fd9f079c31f8360ee2e8f3a968aca
 => => naming to docker.io/khannedy/simple
```

Run Instruction

Run Instruction

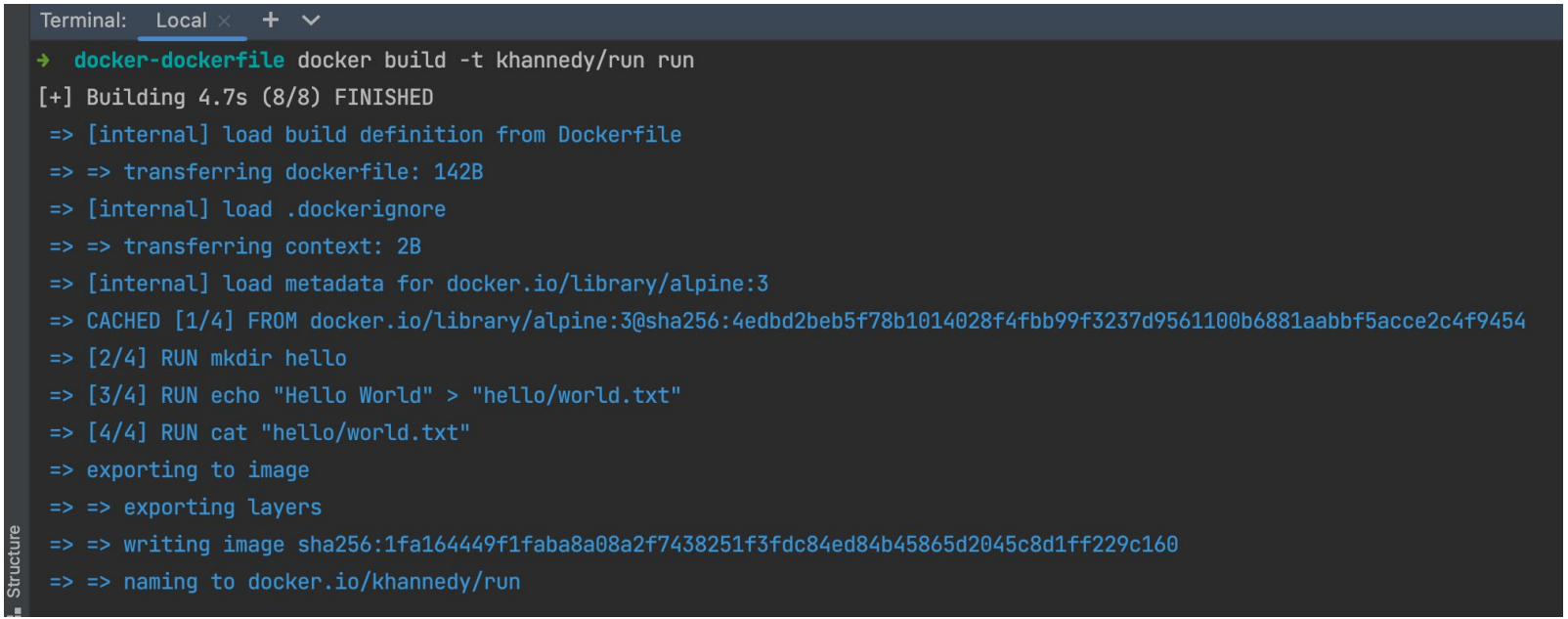
- RUN adalah sebuah instruksi untuk mengeksekusi perintah di dalam image pada saat build stage.
- Hasil perintah RUN akan di commit dalam perubahan image tersebut, jadi perintah RUN akan dieksekusi pada saat proses docker build saja, setelah menjadi Docker Image, perintah tersebut tidak akan dijalankan 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”, “...”]

Kode : Run Instruction

Kode : Docker Build



Display Output

- Secara default, di docker terbaru tidak akan menampilkan tulisan detail dari build-nya
- Jika kita ingin menampilkan detailnya, kita bisa gunakan perintah --progress=plain
- Selain itu juga docker build juga melakukan cache, jika kita ingin mengulangi lagi tanpa menggunakan cache, kita bisa gunakan perintah --no-cache

Command Instruction

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

Kode : Command Instruction

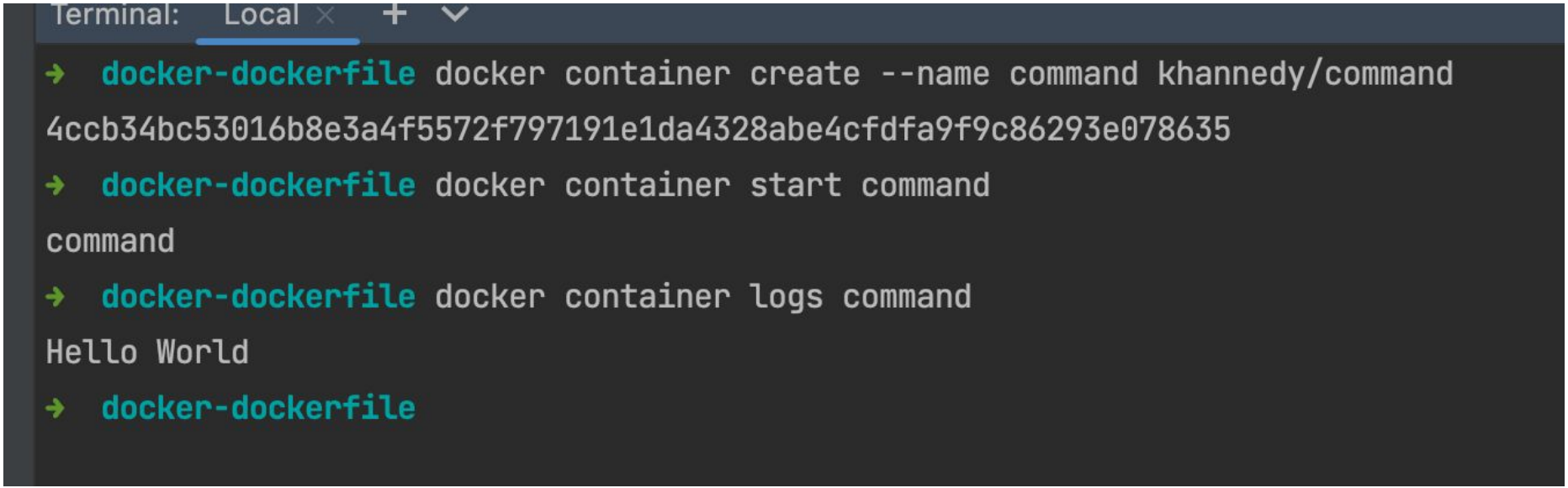
Dockerfile

```
FROM alpine:3

RUN mkdir hello
RUN echo "Hello World" > "hello/world.txt"

CMD cat "hello/world.txt"
```

Kode : Docker Build

```
➜  docker-dockerfile docker build -t khannedy/command command
[+] Building 3.6s (7/7) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 143B
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load metadata for docker.io/library/alpine:3
 => [1/3] FROM docker.io/library/alpine:3@sha256:4edbd2beb5f78b1014028f4fbb99f3237d9561100b6881aabbf5acce2c4f9454
 => CACHED [2/3] RUN mkdir hello
 => CACHED [3/3] RUN echo "Hello World" > "hello/world.txt"
 => exporting to image
 => => exporting layers
 => => writing image sha256:479d4b715d9d77177f785842c8361c16fbee6c42ae549a024bf0e02bcfa94ffb
 => => naming to docker.io/khannedy/command
```

Kode : Docker Container



Label Instruction

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, lisensi dan 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> …

Kode : LABEL Instruction

script.sh | Dockerfile

```
1  FROM alpine:3
2
3  LABEL author="Eko Kurniawa Khanedy"
4  LABEL company="Programmer Zaman Now" website="https://www.programmerzamannow.com"
5
6  RUN mkdir hello
7  RUN echo "Hello World" > "hello/world.txt"
8
9  CMD cat "hello/world.txt"
10
```

Kode : Docker Build

```
➜  docker-dockerfile docker build -t khannedy/label label
[+] Building 4.7s (7/7) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 264B
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load metadata for docker.io/library/alpine:3
 => [1/3] FROM docker.io/library/alpine:3@sha256:4edbd2beb5f78b1014028f4fbb99f3237d9561100b6881aabbf5acce2c4f9454
 => CACHED [2/3] RUN mkdir hello
 => CACHED [3/3] RUN echo "Hello World" > "hello/world.txt"
 => exporting to image
 => => exporting layers
 => => writing image sha256:6123f422aa4c25eae07b8593777092825f7cc4d4f3da7a349d969bc90d1e098b
 => => naming to docker.io/khannedy/label
```

Kode : Inspect Docker Image

```
→  docker-dockerfile docker image inspect khannedy/label
[
        "OnBuild": null,
        "Labels": {
            "author": "Eko Kurniawa Khanedy",
            "company": "Programmer Zaman Now",
            "website": "https://www.programmerzamannow.com"
        }
    },
    "Architecture": "amd64",
    "Os": "linux",
```

Add Instruction

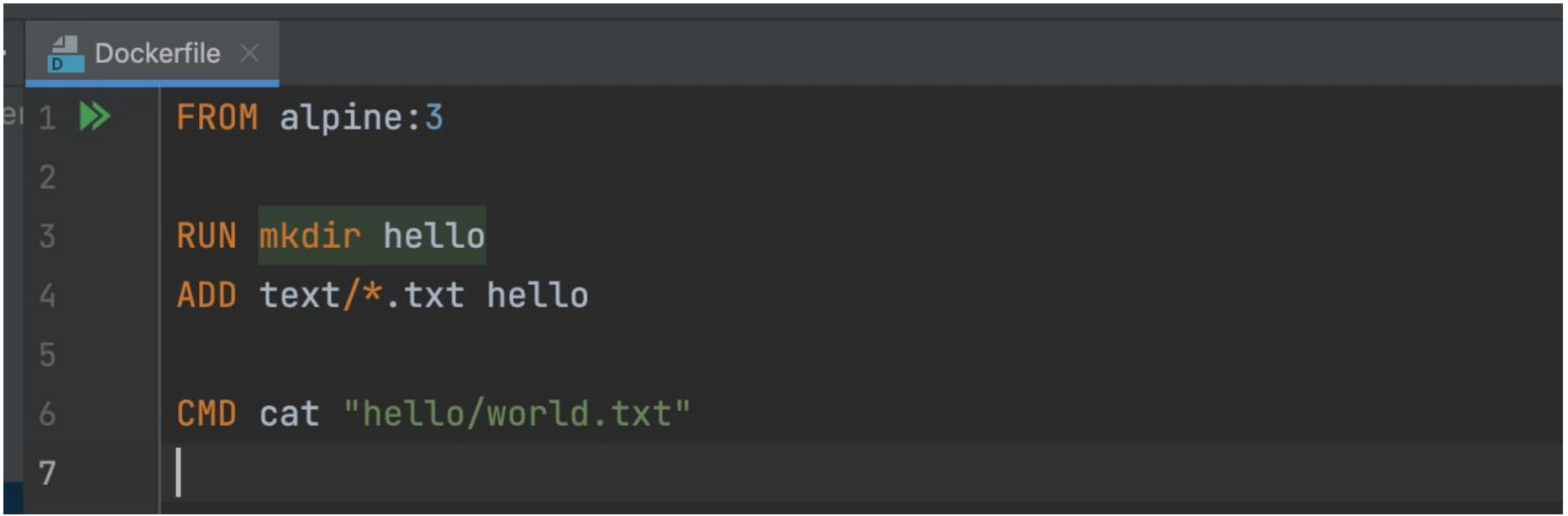
Add Instruction

- 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

Kode : ADD Instruction



Kode : Docker Build

```
➜  docker-dockerfile docker build -t khannedy/add add
[+] Building 3.0s (8/8) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 121B
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load metadata for docker.io/library/alpine:3
 => [internal] load build context
 => => transferring context: 81B
 => [1/3] FROM docker.io/library/alpine:3@sha256:4edbd2beb5f78b1014028f4fbb99f3237d9561100b6881aabbf5acce2c4f9454
 => CACHED [2/3] RUN mkdir hello
 => [3/3] ADD text/*.txt hello
 => exporting to image
 => => exporting layers
 => => writing image sha256:579ad4f330e1200927ed87cd17962342f51dcc9f53feb29dcc4cc70c777188f3
 => => naming to docker.io/khannedy/add
```

Copy Instruction

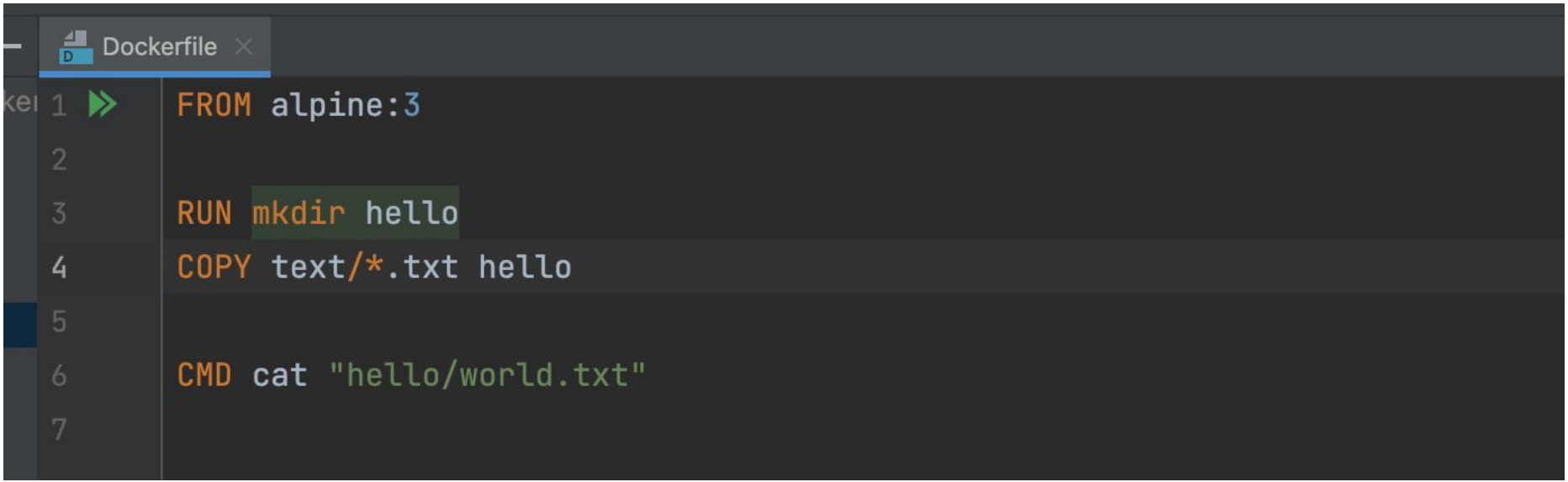
Copy Instruction

- COPY adalah instruksi yang dapat digunakan untuk menambahkan file dari source ke dalam folder destination di Docker Image
- Lantas apa bedanya dengan instruksi ADD kalo begitu?
- COPY hanya melakukan copy file saja, sedangkan ADD selain melakukan copy, dia bisa mendownload source dari URL dan secara otomatis melakukan extract file kompres
- Namun best practice nya, sebisa mungkin menggunakan COPY, jika memang butuh melakukan extract file kompres, gunakan perintah RUN dan jalankan aplikasi untuk extract file kompres tersebut

Copy Instruction Format

- Instruksi COPY memiliki format sebagai berikut :
- COPY source destination
- Contoh :
- COPY world.txt hello # menambah file world.txt ke folder hello
- COPY *.txt hello # menambah semua file .txt ke folder hello

Kode : COPY Instruction



Kode : Docker Build

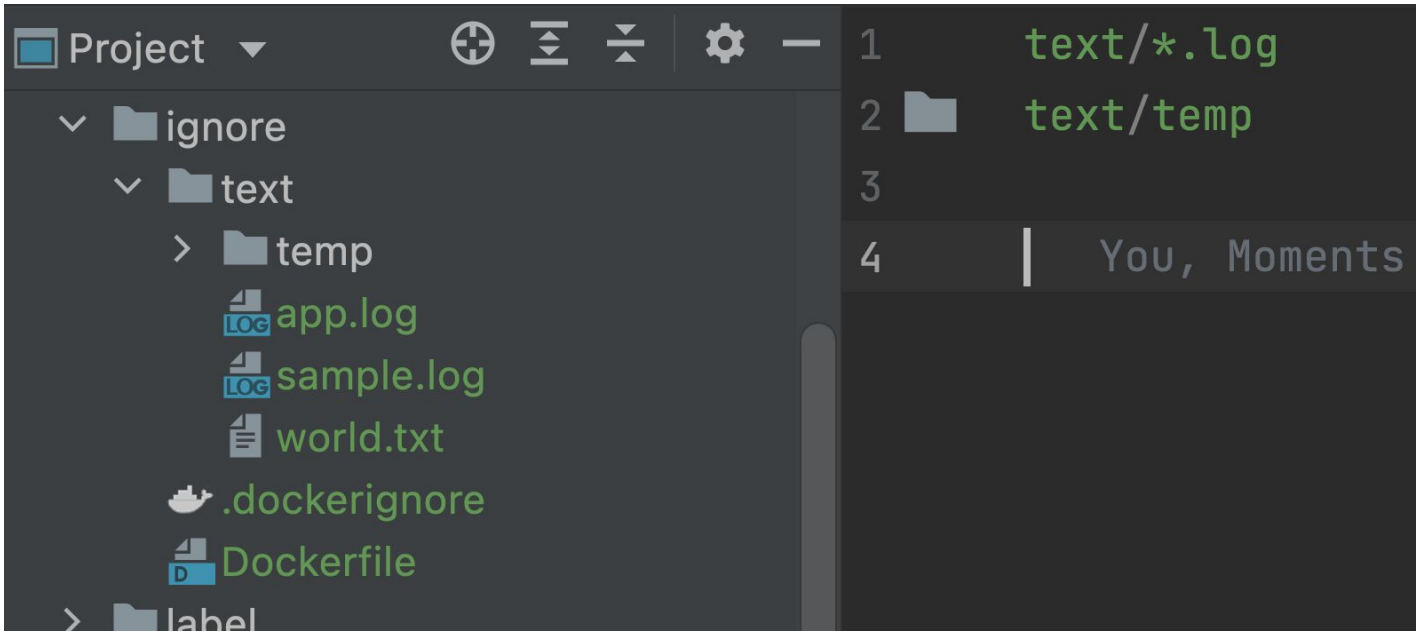
```
→  docker-dockerfile docker build -t khannedy/copy copy
[+] Building 4.5s (8/8) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 122B
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load metadata for docker.io/library/alpine:3
 => [1/3] FROM docker.io/library/alpine:3@sha256:4edbd2beb5f78b1014028f4fbb99f3237d9561100b6881aabbf5acce2c4f9454
 => [internal] load build context
 => => transferring context: 81B
 => CACHED [2/3] RUN mkdir hello
 => [3/3] COPY text/*.txt hello
 => exporting to image
 => => exporting layers
 => => writing image sha256:59a7d83811d49750f93f3313029b65dc0291fc2ca89858dd6cff65cf423efb53
 => => naming to docker.io/khannedy/copy
```

.dockerignore File

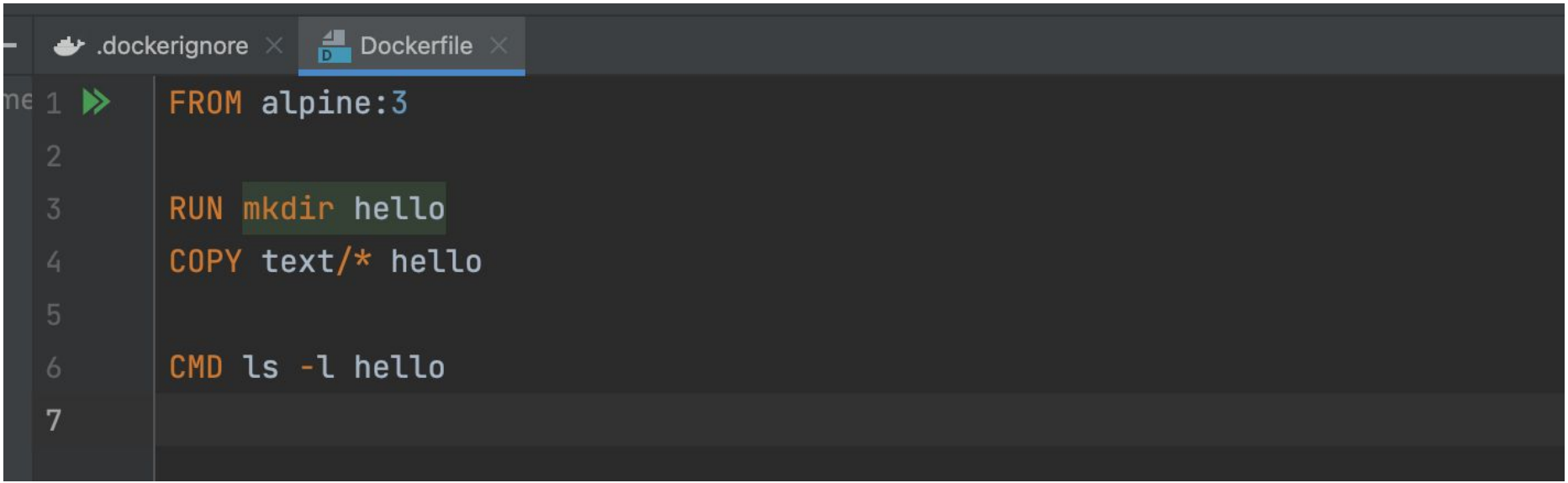
.dockerignore File

- Saat kita melakukan ADD atau COPY dari file source, pertama Docker akan membaca file yang bernama .dockerignore
- File .dockerignore ini seperti file .gitignore, dimana kita bisa menyebutkan file-file apa saja yang ingin kita ignore (hiraukan)
- Artinya jika ada file yang kita sebut di dalam file .dockerignore, secara otomatis file tersebut tidak aka di ADD atau di COPY
- File .dockerignore juga mendukung ignore folder atau menggunakan regular expression

Kode : .dockerignore File



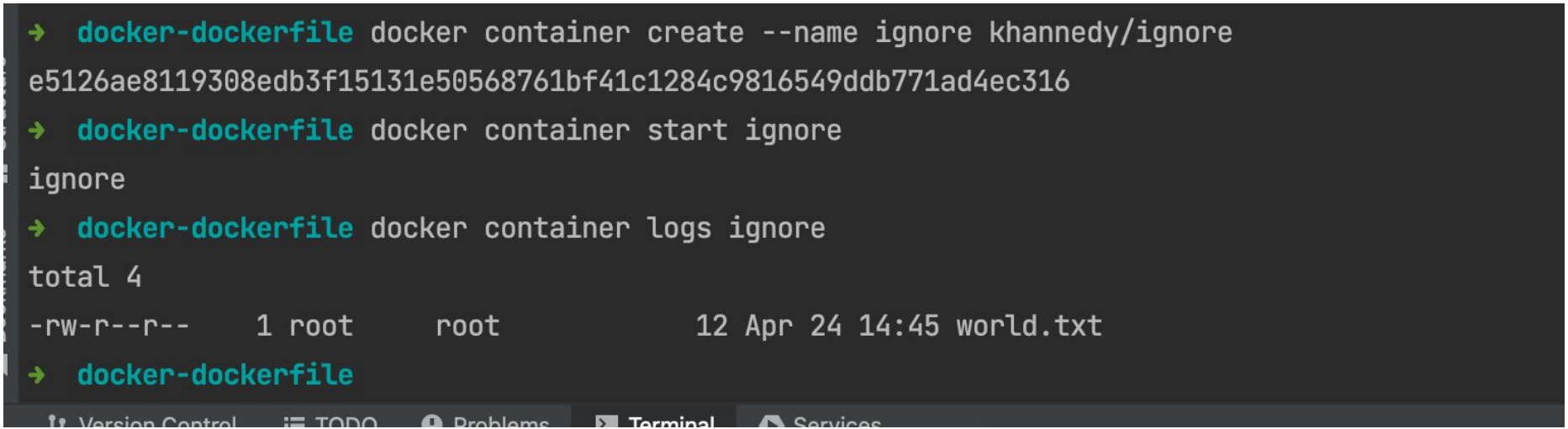
Kode : Dockerfile



Kode : Docker Build

```
➜  docker-dockerfile docker build -t khannedy/ignore ignore
[+] Building 4.6s (8/8) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 108B
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load metadata for docker.io/library/alpine:3
 => [1/3] FROM docker.io/library/alpine:3@sha256:4edbd2beb5f78b1014028f4fbb99f3237d9561100b6881aabbf5acce2c4f9454
 => CACHED [2/3] RUN mkdir hello
 => [internal] load build context
 => => transferring context: 277B
 => [3/3] COPY text/* hello
 => exporting to image
 => => exporting layers
 => => writing image sha256:efb63d518f06e2f2e91fc413854946f1691763ac20e184e1d4cb000a01767dda
 => => naming to docker.io/khannedy/ignore
```

Kode : Docker Container Logs



Expose Instruction

Expose Instruction

- EXPOSE adalah instruksi untuk memberitahu bahwa container akan listen port pada nomor dan protocol tertentu
- Instruksi EXPOSE tidak akan mempublish port apapun sebenarnya, Instruksi EXPOSE hanya digunakan sebagai dokumentasi untuk memberitahu yang membuat Docker Container, bahwa Docker Image ini akan menggunakan port tertentu ketika dijalankan menjadi Docker Container

Expose Instruction Format

- Berikut adalah format untuk instruksi EXPOSE :
- EXPOSE port # default nya menggunakan TCP
- EXPOSE port/tcp
- EXPOSE port/udp

Kode : Hello World Go-Lang Web

- https://gist.github.com/khannedy/166be48cabb637b5beefc4e7998f2c7e
- Simpan dalam file main.go

Kode : Expose Instruction

Dockerfile

```
1  FROM golang:1.18-alpine
2
3  RUN mkdir app
4  COPY main.go app
5
6  EXPOSE 8080
7
8  CMD go run app/main.go
9
```

Kode : Docker Build

```
→  docker-dockerfile docker build -t khannedy/expose expose
[+] Building 96.7s (8/8) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 135B
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load metadata for docker.io/library/golang:1.18-alpine
 => [1/3] FROM docker.io/library/golang:1.18-alpine@sha256:42d35674864fbb577594b60b84ddfba1be52b4d4298c961b46ba95e9fb4712e8
 => => resolve docker.io/library/golang:1.18-alpine@sha256:42d35674864fbb577594b60b84ddfba1be52b4d4298c961b46ba95e9fb4712e8
 => => sha256:dd6fd110e957a512038107a4e1230207963dfa7b5f3d4e8c9669689afd18ada0 5.17kB / 5.17kB
 => => sha256:52dc419b0ee22d1bb5f3b95a57b2edc733b7b585277f8101db7b2c9b53605c1b 271.96kB / 271.96kB
 => => sha256:25bc292e5bacdd8b97a1632a82d0fd0ee6c2692409bd4b35cb9e7e652efbc7c8 153B / 153B
 => => sha256:6858aa20941bdfe0d406b2e4db477feeccf42ca6e40478043b721c6a66f9d1b6 115.57MB / 115.57MB
 => => sha256:42d35674864fbb577594b60b84ddfba1be52b4d4298c961b46ba95e9fb4712e8 1.65kB / 1.65kB
 => => sha256:f94174c5262af3d8446833277aa27af400fd1a880277d43ec436df06ef3bb8ab 1.36kB / 1.36kB
 => => extracting sha256:52dc419b0ee22d1bb5f3b95a57b2edc733b7b585277f8101db7b2c9b53605c1b
```

Kode : Docker Image Inspect

```
→  docker-dockerfile docker image inspect khannedy/expose
[
            "AttachStderr": false,
            "ExposedPorts": {
                "8080/tcp": {}
            },
            "Tty": false,
            "OpenStdin": false,
            "StdinOnce": false,
```

Kode : Docker Container

```
→  docker-dockerfile docker container create --name expose -p 8080:8080 khannedy/expose
6263c6a67630e82299f2501ab08d07ea08d7bc31b4eff38c328202455e2541a7
→  docker-dockerfile docker container start expose
expose
→  docker-dockerfile docker container ls
CONTAINER ID   IMAGE             COMMAND                  CREATED         STATUS         PORTS                    NAMES
6263c6a67630   khannedy/expose   "/bin/sh -c 'go run …"   8 seconds ago   Up 4 seconds   0.0.0.0:8080->8080/tcp   expose
→  docker-dockerfile
```

Environment Variable Instruction

Environment Variable Instruction

- ENV adalah instruksi yang digunakan untuk mengubah environment variable, baik itu ketika tahapan build atau ketika jalan dalam Docker Container
- ENV yang sudah di definisikan di dalam Dockerfile bisa digunakan kembali dengan menggunakan sintaks ${NAMA_ENV}
- Environment Variable yang dibuat menggunakan instruksi ENV disimpan di dalam Docker Image dan bisa dilihat menggunakan perintah docker image inspect
- Selain itu, environment variable juga bisa diganti nilainya ketika pembuatan Docker Container dengan perintah docker container create --env key=value

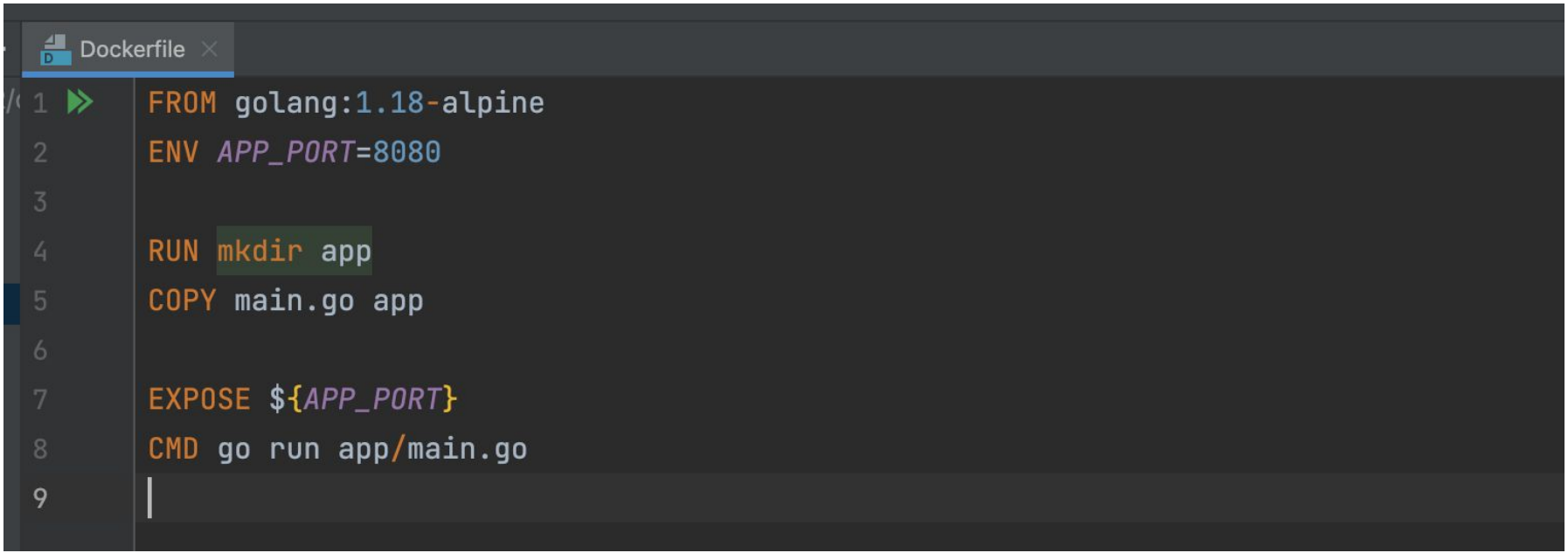
Environment Variable Instruction Format

- Berikut adalah format untuk instruksi ENV :
- ENV key=value
- ENV ke1=value1 key2=value2 …

Kode : Hello World Go-Lang Web dengan Port

- https://gist.github.com/khannedy/e8574fdd9bebfb433a256e7e89f1d5ca
- Simpan dalam file main.go

Kode : ENV Instruction



Kode : Docker Build

```
➜  docker-dockerfile docker build -t khannedy/env env
[+] Building 4.4s (8/8) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 159B
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load metadata for docker.io/library/golang:1.18-alpine
 => CACHED [1/3] FROM docker.io/library/golang:1.18-alpine@sha256:42d35674864fbb577594b60b84ddfba1be52b4d4298c961b46ba95e9fb4712e8
 => [internal] load build context
 => => transferring context: 368B
 => [2/3] RUN mkdir app
 => [3/3] COPY main.go app
 => exporting to image
 => => exporting layers
 => => writing image sha256:7ae6d173afe9f147432c129cd0023a14ba8c39c25e579f5953702f2b6d7225db
 => => naming to docker.io/khannedy/env
```

Kode : Inspect Docker Image

```
→  docker-dockerfile docker image inspect khannedy/env
[
    "StdinOnce": false,
    "Env": [
        "PATH=/go/bin:/usr/local/go/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
        "GOLANG_VERSION=1.18.1",
        "GOPATH=/go",
        "APP_PORT=8080"
    ],
    "Cmd": [
        "/bin/sh",
        "-c",
        "go run app/main.go"
    ]
```

Kode : Docker Container

```
→  docker-dockerfile docker container create --name env --env APP_PORT=8080 -p 8080:8080 khannedy/env
1e021d713f39107aac756320f59572d05250d3b2aa87c08f84ee490c7bbcfc1e
→  docker-dockerfile docker container start env
env
→  docker-dockerfile docker container logs env
Run app in port : 8080
→  docker-dockerfile
```

Volume Instruction

Volume Instruction

- VOLUME merupakan instruksi yang digunakan untuk membuat volume secara otomatis ketika kita membuat Docker Container
- Semua file yang terdapat di volume secara otomatis akan otomatis di copy ke Docker Volume, walaupun kita tidak membuat Docker Volume ketika membuat Docker Container nya
- Ini sangat cocok pada kasus ketika aplikasi kita misal menyimpan data di dalam file, sehingga data bisa secara otomatis aman berada di Docker Volume

Volume Instruction Format

- Berikut adalah format untuk instruksi VOLUME :
- VOLUME /lokasi/folder
- VOLUME /lokasi/folder1 /lokasi/folder2 …
- VOLUME [“/lokasi/folder1”, “/lokasi/folder2”, “...”]

Golang Web dengan Write File

- https://gist.github.com/khannedy/d788b386297caf04b39640bec43f3131
- Simpan dalam file main.go

Kode : Volume Instruction

Dockerfile

```
FROM golang:1.18-alpine
ENV APP_PORT=8080
ENV APP_DATA=/logs

RUN mkdir ${APP_DATA}
RUN mkdir app
COPY main.go app

EXPOSE ${APP_PORT}
VOLUME ${APP_DATA}
CMD go run app/main.go
```

Kode : Docker Build

```
➜  docker-dockerfile docker build -t khannedy/volume volume
[+] Building 5.2s (9/9) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 221B
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load metadata for docker.io/library/golang:1.18-alpine
 => [internal] load build context
 => => transferring context: 657B
 => CACHED [1/4] FROM docker.io/library/golang:1.18-alpine@sha256:42d35674864fbb577594b60b84ddfba1be52b4d4298c961b46ba95e9fb4712e8
 => [2/4] RUN mkdir /logs
 => [3/4] RUN mkdir app
 => [4/4] COPY main.go app
 => exporting to image
 => => exporting layers
 => => writing image sha256:29b8f45cf5b6945093720570758b0074eebd5dbe5d4441c1f8b68cfe83c7f2b8
```

Kode : Docker Image Inspect

```
→  docker-dockerfile docker image inspect khannedy/volume
[
            "Volumes": {
                "/logs": {}
            },
            "WorkingDir": "/go",
            "Entrypoint": null,
            "OnBuild": null,
```

Kode : Docker Container

```
→  docker-dockerfile docker container create --name volume --env APP_PORT=8080 -p 8080:8080 khannedy/volume
0e93ff9a94db5246975d9ff81264a12ea3224f66dd3ef270cce4a35cf9413a27
→  docker-dockerfile docker container start volume
volume
→  docker-dockerfile docker container logs volume
Run app in port : 8080
DONE Write File : /logs/.txt
DONE Write File : /logs/favicon.ico.txt
DONE Write File : /logs/eko.txt
DONE Write File : /logs/favicon.ico.txt
DONE Write File : /logs/kurniawan.txt
DONE Write File : /logs/favicon.ico.txt
DONE Write File : /logs/khannedy.txt
DONE Write File : /logs/favicon.ico.txt
```

Kode : Docker Container Inspect

```
→  docker-dockerfile docker container inspect volume
[
```


```
"Mounts": [
    {
        "Type": "volume",
        "Name": "3a61af87680b1b0b75b3ce526ea971d06b0a5931268cb0ce027639f510c9d1e9",
        "Source": "/var/lib/docker/volumes/3a61af87680b1b0b75b3ce526ea971d06b0a5931268cb0ce027639f510c9d1e9/_data",
        "Destination": "/logs",
        "Driver": "local",
        "Mode": "",
        "RW": true,
        "Propagation": ""
    }
],
```

Kode : Docker Volume

```
→  docker-dockerfile docker volume ls
DRIVER    VOLUME NAME
local     0b7613d13bb4ed2b9a42fc950c3a593594f734852b958750720f74b9fd085cfd
local     0ba76867b670f0b2eda830c9e42562aac1f9da2eb260afee58ec0ce9fd972808
local     001d972446803a3d28e474e2a9794731f36d6d8fbdda5f0e125dd08b51be609a
local     2a7e8a917c5742d9ce9e27c4ca904bd4be85c6b52a089a45a2caf274e067762d
local     2fbb713d1896a4081375888ddae92f0d70362c7f00fa4a39469103505042d886
local     3a61af87680b1b0b75b3ce526ea971d06b0a5931268cb0ce027639f510c9d1e9
local     025eba913e25e7c4aca5508cf4dbf8fe2014c6a0de824db71ad1d672ed4f8645
```

Working Directory Instruction

Working Directory Instruction

- WORKDIR adalah instruksi untuk menentukan direktori / folder untuk menjalankan instruksi RUN, CMD, ENTRYPOINT, COPY dan ADD
- Jika WORKDIR tidak ada, secara otomatis direktorinya akan dibuat, dan selanjutnya setelah kita tentukan lokasi WORKDIR nya, direktori tersebut dijadikan tempat menjalankan instruksi selanjutnya
- Jika lokasi WORKDIR adalah relative path, maka secara otomatis dia akan masuk ke direktori dari WORKDIR sebelumnya
- WORKDIR juga bisa digunakan sebagai path untuk lokasi pertama kali ketika kita masuk ke dalam Docker Container

Working Directory Instruction Format

- Berikut adalah format untuk instruksi WORKDIR :
- WORKDIR /app # artinya working directory nya adalah /app
- WORKDIR docker # sekarang working directory nya adalah /app/docker
- WORKDIR /home/app # sekarang working directory nya adalah /home/app

Golang Web Hello World

- https://gist.github.com/khannedy/9262c7784a9ef65ced9dac712822a853
- Simpan dalam file main.go

Kode : Working Directory Instruction

Dockerfile

```
1  FROM golang:1.18-alpine
2
3  WORKDIR /app
4  COPY main.go /app
5
6  EXPOSE 8080
7  CMD go run main.go
8
```

Kode : Docker Build

```
➜  docker-dockerfile docker build -t khannedy/workdir workdir
[+] Building 1.5s (8/8) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 130B
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load metadata for docker.io/library/golang:1.18-alpine
 => CACHED [1/3] FROM docker.io/library/golang:1.18-alpine@sha256:42d35674864fbb577594b60b84ddfba1be52b4d4298c961b46ba95e9fb4712e8
 => [internal] load build context
 => => transferring context: 29B
 => [2/3] WORKDIR /app
 => [3/3] COPY main.go /app
 => exporting to image
 => => exporting layers
 => => writing image sha256:299dcac8dcd2324a007a1c130fe223e00ab208f4f7f1aebea9df1c987a4e0f44
 => => naming to docker.io/khannedy/workdir
```

Kode : Docker Container

```
➜  docker-dockerfile docker container create --name workdir -p 8080:8080 khannedy/workdir
f0088a4d9a62c7cb55515acffb838dc288cabb946ffd1da967108762661725 30
➜  docker-dockerfile docker container start workdir
workdir
➜  docker-dockerfile docker container exec -i -t workdir /bin/sh
/app # pwd
/app
/app #
```

User Instruction

User Instruction

- USER adalah instruksi yang digunakan untuk mengubah user atau user group ketika Docker Image dijalankan
- Secara default, Docker akan menggunakan user root, namun pada beberapa kasus, mungkin ada aplikasi yang tidak ingin jalan dalam user root, maka kita bisa mengubah user nya menggunakan instruksi USER

User Instruction Format

- Berikut adalah format untuk instruksi USER:
- USER <user> # mengubah user
- USER <user>:<group> # mengubah user dan user group

Kode : User Instruction

Dockerfile script.sh

```
FROM golang:1.18-alpine

RUN mkdir /app

RUN addgroup -S pzngroup
RUN adduser -S -D -h /app pznuser pzngroup
RUN chown -R pznuser:pzngroup /app
USER pznuser

COPY main.go /app

EXPOSE 8080
CMD go run /app/main.go

```

Kode : Docker Build

```
➜  docker-dockerfile docker build -t khannedy/user user
[+] Building 5.7s (11/11) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 257B
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load metadata for docker.io/library/golang:1.18-alpine
 => [1/6] FROM docker.io/library/golang:1.18-alpine@sha256:42d35674864fbb577594b60b84ddfba1be52b4d4298c961b46ba95e9fb4712e8
 => [internal] load build context
 => => transferring context: 29B
 => CACHED [2/6] RUN mkdir /app
 => CACHED [3/6] RUN addgroup -S pzngroup
 => CACHED [4/6] RUN adduser -S -D -h /app pznuser pzngroup
 => CACHED [5/6] RUN chown -R pznuser:pzngroup /app
 => [6/6] COPY main.go /app
 => exporting to image
 => => exporting layers
 => => writing image sha256:1975f378100162fd30350d2f33b94e1a0ca2127d5331d9c6355cb4848db7893f
 => => naming to docker.io/khannedy/user
```

Kode : Docker Container

```
→  docker-dockerfile docker container create --name user -p 8080:8080 khannedy/user
2a970b7fcc71b19e5820bea93510f3daeb2828c8d8dee57b4cee36e2fae1b028
→  docker-dockerfile docker container start user
user
→  docker-dockerfile docker container exec -i -t user /bin/sh
/go $ whoami
pznuser
/go $ exit
→  docker-dockerfile
```

Argument Instruction

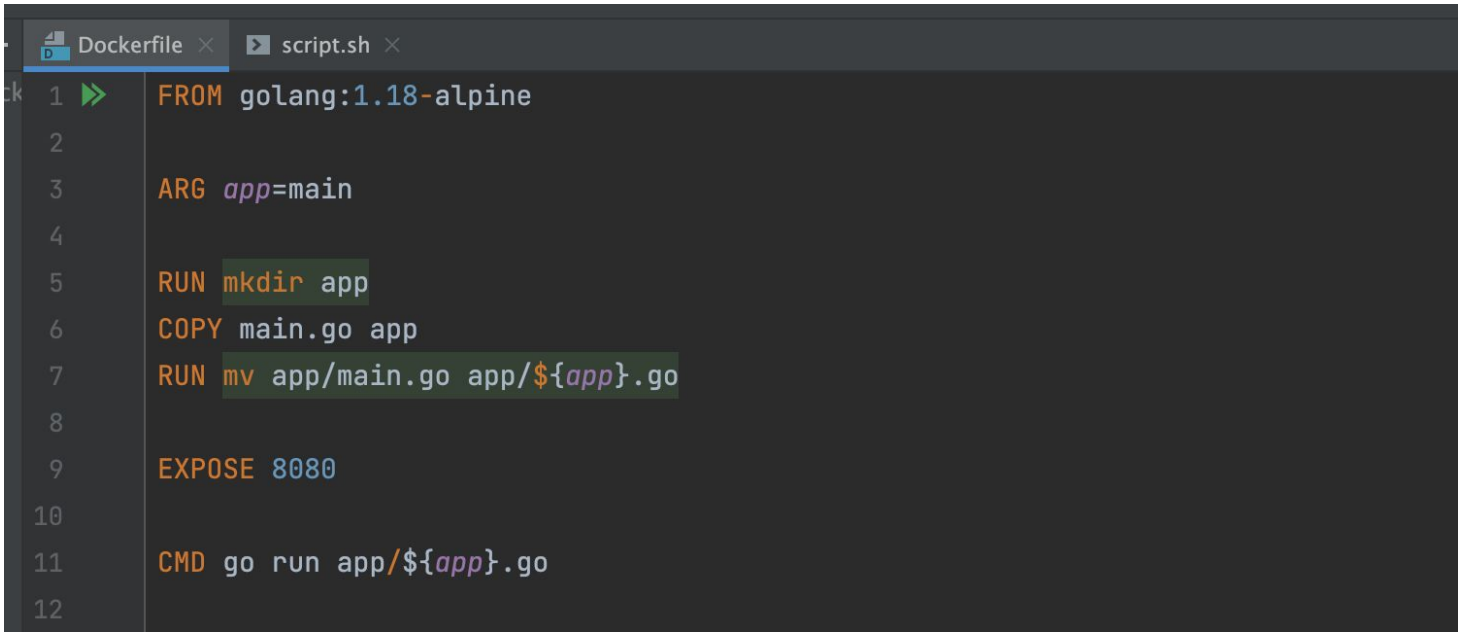
Argument Instruction

- ARG merupakan instruksi yang digunakan untuk mendefinisikan variable yang bisa digunakan oleh pengguna untuk dikirim ketika melakukan proses docker build menggunakan perintah --build-arg key=value
- ARG hanya digunakan pada saat proses build time, artinya ketika berjalan dalam Docker Container, ARG tidak akan digunakan, berbeda dengan ENV yang digunakan ketika berjalan dalam Docker Container
- Cara mengakses variable dari ARG sama seperti mengakses variable dari ENV, menggunakan ${variable_name}

Argument Instruction Format

- Berikut adalah format untuk instruksi ARG:
- ARG key # membuat argument variable
- ARG key=defaultvalue # membuat argument variable dengan default value jika tidak diisi

Kode : Argument Instruction



Kode : Docker Build

```
➜  docker-dockerfile docker build -t khannedy/arg arg --build-arg app=pzn
[+] Building 2.4s (9/9) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 37B
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load metadata for docker.io/library/golang:1.18-alpine
 => [internal] load build context
 => => transferring context: 29B
 => CACHED [1/4] FROM docker.io/library/golang:1.18-alpine@sha256:42d35674864fbb577594b60b84ddfba1be52b4d4298c961b46ba95e9fb4712e8
 => [2/4] RUN mkdir app
 => [3/4] COPY main.go app
 => [4/4] RUN mv app/main.go app/pzn.go
 => exporting to image
 => => exporting layers
 => => writing image sha256:f698c2710a827b7c631b0e5514aa3563ab73f098fdd959120cf6882fcd6fefdf
 => => naming to docker.io/khannedy/arg
```

Kode : Docker Container

```
➜  docker-dockerfile docker container create --name arg -p 8080:8080 khannedy/arg
a96f99875ac11a7433481c4edb1b8fd9ff00e6c2891adfc86e8771e13bc44e82
➜  docker-dockerfile docker container start arg
arg
➜  docker-dockerfile docker container exec -i -t arg /bin/sh
Error response from daemon: Container a96f99875ac11a7433481c4edb1b8fd9ff00e6c2891adfc86e8771e13bc44e82 is not running
➜  docker-dockerfile docker container logs arg
stat app/.go: no such file or directory
```

Kenapa Error?

- Hal ini dikarenakan ARG hanya bisa diakses pada waktu build time, sedangkan CMD itu dijalankan pada saat runtime
- Jadi jika kita ingin menggunakan ARG pada CMD, maka kita perlu memasukkan data ARG tersebut ke ENV

Kode : Argument dan Environment Instruction

Dockerfile | script.sh

```
2
3   ARG app=main
4
5   RUN mkdir app
6   COPY main.go app
7   RUN mv app/main.go app/${app}.go
8
9   EXPOSE 8080
10
11  ENV app=${app}
12  CMD go run app/${app}.go
13
14
```

Health Check Instruction

Health Check Instruction

- HEALTHCHECK adalah instruksi yang digunakan untuk memberi tahu Docker bagaimana untuk mengecek apakah Container masih berjalan dengan baik atau tidak
- Jika terdapat HEALTHCHECK, secara otomatis Container akan memili status health, dari awalnya bernilai starting, jika sukses maka bernilai healthy, jika gagal akan bernilai unhealty

Health Check Instruction Format

- Berikut adalah format untuk instruksi HEALTHCHECK :
- HEALTHCHECK NONE # artinya disabled health check
- HEALTHCHECK [OPTIONS] CMD command
- OPTIONS :
    - --interval=DURATION (default: 30s)
    - --timeout=DURATION (default: 30s)
    - --start-period=DURATION (default: 0s)
    - --retries=N (default: 3)

Simple Golang Web dengan Health Check

- https://gist.github.com/khannedy/08d52d1d9b7b41b34535df85509417b4
- Simpan dalam file main.go

Kode : Health Check Instruction

Dockerfile | script.sh

```
FROM golang:1.18-alpine

RUN apk --no-cache add curl
RUN mkdir app
COPY main.go app

EXPOSE 8080

HEALTHCHECK --interval=5s --start-period=5s CMD curl -f http://localhost:8080/health
CMD go run app/main.go
```

Kode : Docker Build

```
➜  docker-dockerfile docker build -t khannedy/health health
[+] Building 7.8s (9/9) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 250B
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load metadata for docker.io/library/golang:1.18-alpine
 => CACHED [1/4] FROM docker.io/library/golang:1.18-alpine@sha256:42d35674864fbb577594b60b84ddfba1be52b4d4298c961b46ba95e9fb4712e8
 => [internal] load build context
 => => transferring context: 504B
 => [2/4] RUN apk --no-cache add curl
 => [3/4] RUN mkdir app
 => [4/4] COPY main.go app
 => exporting to image
 => => exporting layers
 => => writing image sha256:039ffb148687153f1cf0fef22e4a8d7cd10cd6eec92743f3194ef241400da59d
 => => naming to docker.io/khannedy/health
```

Kode : Docker Container

```
→  docker-dockerfile docker container create --name health -p 8080:8080 khannedy/health
81f50db51f70295a62fcfa3e10d3c69bd124ffceff76e606c72b69b051fe19c4
→  docker-dockerfile docker container start health
health
→  docker-dockerfile docker container ls
CONTAINER ID   IMAGE             COMMAND                  CREATED         STATUS                            PORTS                    NAMES
81f50db51f70   khannedy/health   "/bin/sh -c 'go run …"   7 seconds ago   Up 3 seconds (health: starting)   0.0.0.0:8080->8080/tcp   health
→  docker-dockerfile docker container ls
CONTAINER ID   IMAGE             COMMAND                  CREATED          STATUS                    PORTS                    NAMES
81f50db51f70   khannedy/health   "/bin/sh -c 'go run …"   17 seconds ago   Up 13 seconds (healthy)   0.0.0.0:8080->8080/tcp   health
```

```
→  docker-dockerfile docker container ls
CONTAINER ID   IMAGE             COMMAND                  CREATED          STATUS                      PORTS                    NAMES
81f50db51f70   khannedy/health   "/bin/sh -c 'go run …"   50 seconds ago   Up 46 seconds (unhealthy)   0.0.0.0:8080->8080/tcp   health
→  docker-dockerfile
```

Kode : Docker Container Inspect

```
→  docker-dockerfile docker container inspect health
[
    "Health": {
        "Status": "unhealthy",
        "FailingStreak": 15,
        "Log": [
            {
                "Start": "2022-04-26T03:21:07.411374726Z",
                "End": "2022-04-26T03:21:07.483623973Z",
                "ExitCode": 22,
                "Output": "  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current\n                                 Dload  Upload
0 --:--:-- --:--:-- --:--:--     0\r  0     2    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0\ncurl: (22) The requested URL returned e
            },
            {
                "Start": "2022-04-26T03:21:12.488254743Z",
                "End": "2022-04-26T03:21:12.559295072Z",
```

Entrypoint Instruction

Entrypoint Instruction

- ENTRYPOINT adalah instruksi untuk menentukan executable file yang akan dijalankan oleh container
- Biasanya ENTRYPOINT itu erat kaitannya dengan instruksi CMD
- Saat kita membuat instruksi CMD tanpa executable file, secara otomatis CMD akan menggunakan ENTRYPOINT

Entrypoint Instruction Format

- Berikut adalah format untuk instruksi ENTRYPOINT:
- ENTRYPOINT [“executable”, “param1”, “param2”]
- ENTRYPOINT executable param1 param2
- Saat menggunakan CMD [“param1”, “param2”], maka param tersebut akan dikirim ke ENTRYPOINT

Kode : Entrypoint Instruction

Dockerfile | script.sh

```
1  FROM golang:1.18-alpine
2
3  RUN mkdir /app/
4  COPY main.go /app/
5
6  EXPOSE 8080
7  ENTRYPOINT ["go", "run"]
8  CMD ["/app/main.go"]
9
```

Kode : Docker Build

```
➜  docker-dockerfile docker build -t khannedy/entrypoint entrypoint
[+] Building 4.6s (8/8) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 161B
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load metadata for docker.io/library/golang:1.18-alpine
 => [1/3] FROM docker.io/library/golang:1.18-alpine@sha256:42d35674864fbb577594b60b84ddfba1be52b4d4298c961b46ba95e9fb4712e8
 => [internal] load build context
 => => transferring context: 29B
 => CACHED [2/3] RUN mkdir /app/
 => CACHED [3/3] COPY main.go /app/
 => exporting to image
 => => exporting layers
 => => writing image sha256:92b0085de1487947c6e80def976c084c23fb1273147be4dc6124a96b83b33998
 => => naming to docker.io/khannedy/entrypoint
```

Kode : Docker Container

```
➜  docker-dockerfile docker container create --name entrypoint -p 8080:8080 khannedy/entrypoint
eeb06e8bba5f67a4daf2bb55abe1137683e557abe4e7f3997e1eee5c0ed2aa23
➜  docker-dockerfile docker container start entrypoint
entrypoint
➜  docker-dockerfile docker container logs entrypoint
➜  docker-dockerfile docker container exec -i -t entrypoint /bin/sh
/go # exit
➜  docker-dockerfile
```

Multi Stage Build

Masalah Dengan Image Size

- Saat kita membuat Dockerfile dari base image yang besar, secara otomatis ukuran Image nya pun akan menjadi besar juga
- Oleh karena itu, usahakan selalu gunakan base image yang memang kita butuhkan saja, jangan terlalu banyak menginstall fitur di Image padahal tidak kita gunakan

Kode : Image Size

```
→  docker-dockerfile docker image ls
REPOSITORY                                  TAG         IMAGE ID       CREATED          SIZE
khannedy/env                                latest      7ae6d173afe9   2 minutes ago    329MB
khannedy/expose                             latest      b8aa53ba95e6   6 hours ago      329MB
khannedy/ignore                             latest      dc94bb5e9cd7   7 hours ago      5.57MB
khannedy/copy                               latest      59a7d83811d4   7 hours ago      5.57MB
khannedy/add                                latest      579ad4f330e1   7 hours ago      5.57MB
khannedy/run                                latest      28a57041faf4   12 hours ago     5.57MB
khannedy/label                              latest      6123f422aa4c   12 hours ago     5.57MB
khannedy/command                            latest      479d4b715d9d   12 hours ago     5.57MB
```

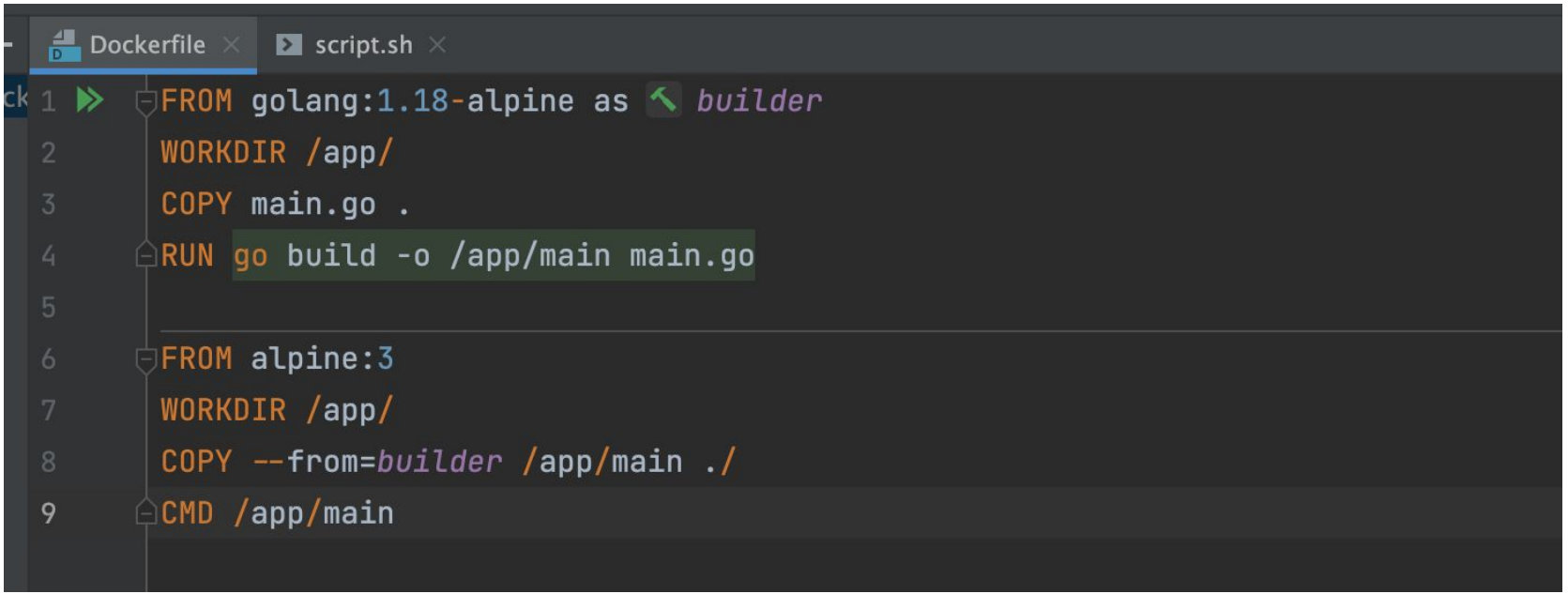
Contoh Solusi Dengan Image Size Besar

- Sebelumnya kita menggunakan bahasa pemrograman Go-Lang untuk membuat web sederhana.
- Sebenarnya, Go-Lang memiliki fitur untuk melakukan kompilasi kode program Go-Lang menjadi binary file, sehingga tidak membutuhkan Image Go-Lang lagi
- Kita bisa melakukan proses kompilasi di laptop kita, lalu file binary nya yang kita simpan di Image, dan cukup gunakan base image Linux Alpine misal nya
- Namun pada kasus Go-Lang, kita di rekomendasikan melakukan kompilasi file binary di sistem operasi yang sama, pada kasus ini saya menggunakan Mac, sedangkan ingin menggunakan Image Alpine, jadi tidak bisa saya lakukan

Multi Stage Build

- Docker memiliki fitur Multi Stage Build, dimana dalam Dockerfile, kita bisa membuat beberapa Build Stage atau tahapan build
- Seperti kita tahu, bahwa di awal build, biasanya kita menggunakan instruksi FROM, dan di dalam Dockerfile, kita bisa menggunakan beberapa instruksi FROM
- Setiap Instruksi FROM, artinya itu adalah build stage
- Hal build stage terakhir adalah build stage yang akan dijadikan sebagai Image
- Artinya, kita bisa memanfaatkan Docker build stage ini untuk melakukan proses kompilasi kode program Go-Lang kita

Kode : Multi Stage Build



Kode : Docker Build

```
➜  docker-dockerfile docker build -t khannedy/multi multi
[+] Building 4.3s (13/13) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 217B
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load metadata for docker.io/library/alpine:3
 => [internal] load metadata for docker.io/library/golang:1.18-alpine
 => [builder 1/4] FROM docker.io/library/golang:1.18-alpine@sha256:42d35674864fbb577594b60b84ddfba1be52b4d4298c961b46ba95e9fb4712e8
 => [stage-1 1/3] FROM docker.io/library/alpine:3@sha256:4edbd2beb5f78b1014028f4fbb99f3237d9561100b6881aabbf5acce2c4f9454
 => CACHED [stage-1 2/3] WORKDIR /app/
 => [internal] load build context
 => => transferring context: 29B
 => CACHED [builder 2/4] WORKDIR /app/
 => CACHED [builder 3/4] COPY main.go .
 => [builder 4/4] RUN go build -o /app/main main.go
 => [stage-1 3/3] COPY --from=builder /app/main ./
 => exporting to image
```

Kode : Docker Image & Container

```
→  docker-dockerfile docker image ls | grep multi
khannedy/multi                                  latest       81245736f901   29 seconds ago   11.9MB
→  docker-dockerfile docker container create --name multi -p 8080:8080 khannedy/multi
35acea5ee3db709cf55f38f6799a760f1c54f5530fdb997a7719c057a3de5ad0
→  docker-dockerfile docker container start multi
multi
→  docker-dockerfile docker container exec -i -t multi /bin/sh
/app # ps aux
PID   USER     TIME  COMMAND
    1 root      0:00 /app/main
   13 root      0:00 /bin/sh
   20 root      0:00 ps aux
/app # exit
→  docker-dockerfile
```

Docker Hub Registry

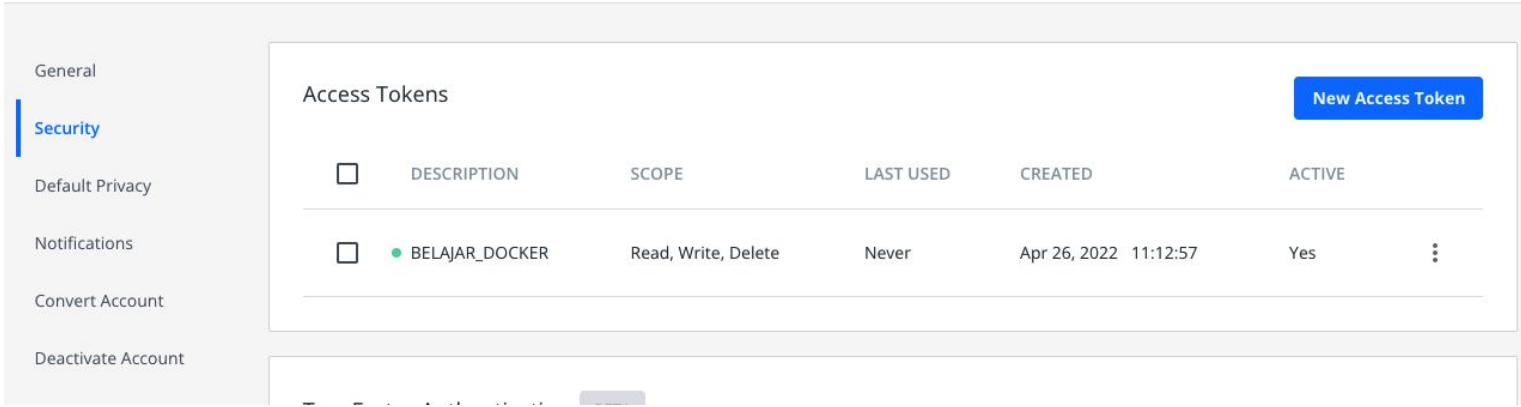
Docker Hub Registry

- Setelah kita selesai membuat Image, selanjutnya hal yang biasa dilakukan adalah mengupload Image tersebut ke Docker Registry
- Salah satu Docker Registry yang gratis contohnya adalah Docker Hub
- https://hub.docker.com/

Docker Hub Access Token

khannedy

User Joined June 28, 2015



Kode : Docker Push

```
→  docker-dockerfile docker login -u khannedy
Password:
Login Succeeded
→  docker-dockerfile docker push khannedy/multi
Using default tag: latest
The push refers to repository [docker.io/khannedy/multi]
50b3b1472d7b: Pushed
9bea334b9008: Pushed
4fc242d58285: Mounted from library/golang
latest: digest: sha256:ee32464f295f77e284a6387285ef9a589453379df3a92932d8d1496927183e48 size: 946
→  docker-dockerfile
```

Digital Ocean Container Registry

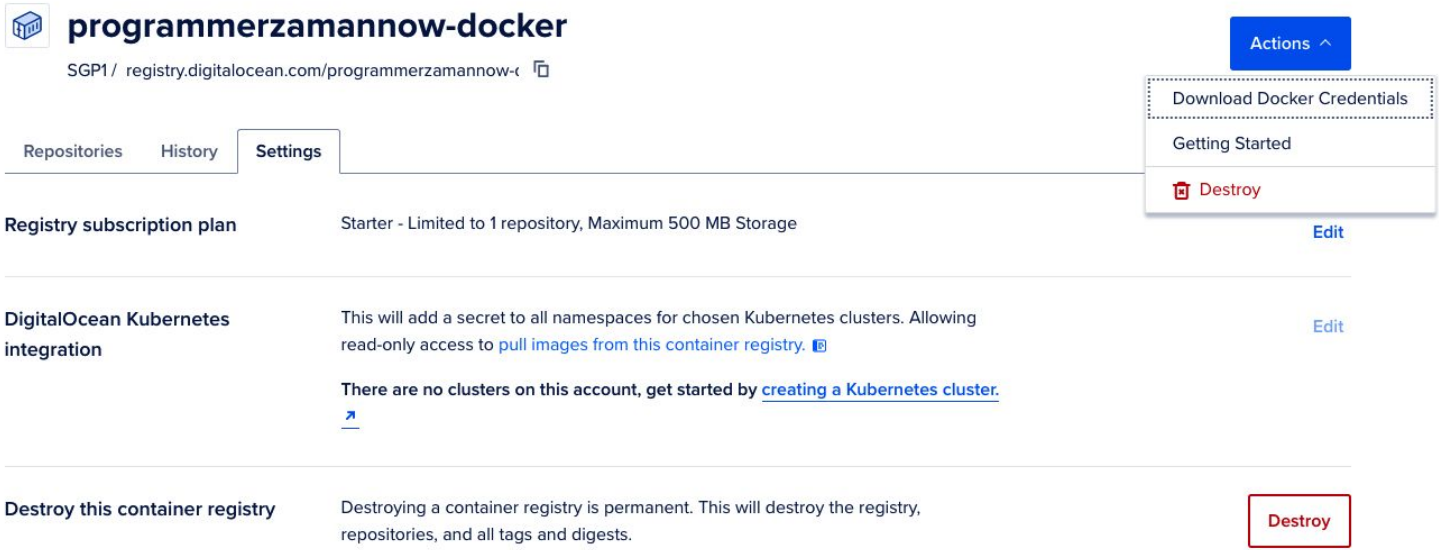
Digital Ocean Container Registry

- Digital Ocean adalah salah satu cloud provider yang populer, dan memiliki fitur Docker Registry bernama Container Registry
- Terdapat Free Version untuk ukuran sampai 500MB yang bisa kita gunakan
- https://www.digitalocean.com/products/container-registry
- Silahkan buat Container Registry terlebih dahulu

Docker Config

- Berbeda dengan Docker Hub yang kita diperlukan melakukan login ketika ingin melakukan push ke Registry
- Di Digital Ocean, kita akan menggunakan Docker Config untuk mengirim Image ke Digital Ocean Container Registry
- Ini lebih mudah karena kita bisa dengan gampang push Image dari manapun selama menggunakan config file yang sama

Download Docker Credential



Konfigurasi Docker Config

- Secara default, Docker akan membaca config yang terdapat di $HOME/.docker
- Di dalamnya terdapat file config.json yang berisi konfigurasi credential yang sudah kita gunakan ketika login ke Docker Hub
- Agar tidak mengganggu, khusus untuk Digital Ocean, kita akan buat folder terpisah, misal .docker-digital-ocean
- Selanjunya file creadential yang sudah di download, silahkan ganti namanya menjadi config.json dan simpan di folder .docker-digital-ocean tersebut

Docker Push

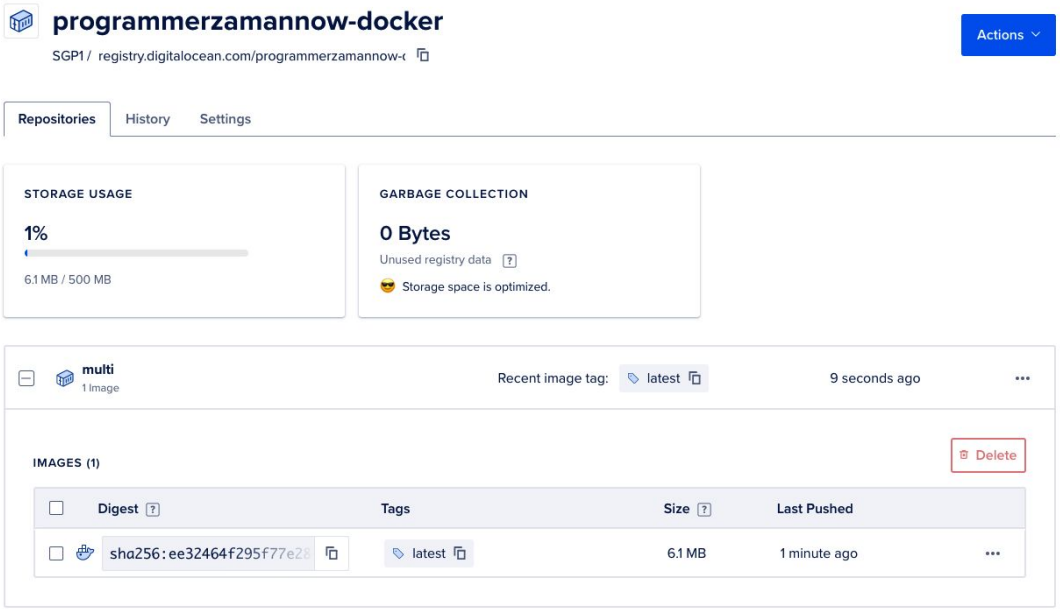
- Jika kita menggunakan perintah docker push, secara default itu akan melakukan push ke Container Registry yang teregistrasi di $HOME/.docker
- Karena kita menggunakan lokasi yang berbeda untuk Digital Ocean, jadi ketika melakukan push, kita perlu mengubah default config nya menggunaka perintah :

  docker --config /lokasi/folder/config/ push image

Kode : Docker Push ke Digital Ocean

```
→  docker-dockerfile docker tag khannedy/multi registry.digitalocean.com/programmerzamannow-docker/multi
→  docker-dockerfile docker --config /Users/khannedy/.docker-digital-ocean push registry.digitalocean.com/programmerzamannow-docker/multi
Using default tag: latest
The push refers to repository [registry.digitalocean.com/programmerzamannow-docker/multi]
50b3b1472d7b: Pushed
9bea334b9008: Pushed
4fc242d58285: Pushed
latest: digest: sha256:ee32464f295f77e284a6387285ef9a589453379df3a92932d8d1496927183e48 size: 946
→  docker-dockerfile
```

Digital Ocean Container Registry



Materi Selanjutnya

Materi Selanjutnya

- Docker Compose