

LAPORAN PRAKTIKUM
TEKNOLOGI CLOUD
PERTEMUAN KE – 7



Disusun Oleh :

NAMA : TARISA DWI SEPTIA
NIM : 205410126
JURUSAN : TEKNIK INFORMATIKA
JENJANG : S1

Sekolah Tinggi Management Informatika Komputer

AKAKOM

YOGYAKARTA

2020

Modul 7

Pengenalan Container Docker

A. Tujuan

- Dapat mengimplementasikan dan memahami container docker

B. Dasar Teori

Dengan container, sebuah program 'diikat' beserta library-nya, file konfigurasi, dan seluruh hal yang dibutuhkannya. Perbedaan yang sangat terlihat dibandingkan dengan virtualisasi adalah container memiliki ukuran file yang jauh lebih kecil karena tidak perlu menyiapkan sistem operasi secara penuh. Dalam hal ini, pengembang biasa menyebutnya sebagai 'lightweight' platform. Aplikasi yang berjalan menggunakan container pun jauh lebih cepat dan lebih efisien. Docker adalah salah satu platform yang dibangun berdasarkan teknologi container. Docker merupakan sebuah project open-source yang menyediakan platform terbuka untuk developer maupun sysadmin untuk dapat membangun, mengemas, dan menjalankan aplikasi dimanapun sebagai sebuah wadah (container) yang ringan. Dengan sangat populernya docker, sebagian orang sering menganggap docker adalah sebutan lain untuk container.

C. Praktik

1. Menjalankan container

Semua container berawal dari docker image. Docker image merupakan sistem operasi yang didalamnya terdapat library / runtime / aplikasi tertentu. Image ini memiliki semua yang dibutuhkan untuk menjalankan proses sehingga host tidak perlu melakukan konfigurasi atau *dependensi* (ketergantungan terhadap packet atau library). Untuk menjalankan container, dapat menggunakan Docker image yang buat sendiri (Build) atau menggunakan Docker image yang telah disediakan oleh Docker dan komunitas. Beberapa perintah docker memerlukan penggunaan *sudo*, namun pada docker for windows dan katacoda, anda tidak perlu menggunakan perintah *sudo*. Dapat menggunakan perintah :

`docker search <nama-image>`

Output yang dihasilkan :

```
$ docker search alpine
```

NAME	DESCRIPTION	STARS
OFFICIAL		
alpine	A minimal Docker image based on Alpine Linux.	8088
[OK]		
whart/alpine-node	Minimal Node.js built on Alpine Linux	484
anapsix/alpine-java	Oracle Java 8 (and 7) with GLIBC 2.28 over A...	475
[OK]		
frolvlad/alpine-glibc	Alpine Docker image with glibc (+12MB)	271
[OK]		
alpine/git	A simple git container running in alpine li...	193
[OK]		
yobasystems/alpine-mariadb	MariaDB running on Alpine Linux [docker] [am...	98
[OK]		
alpine/socat	Run socat command in alpine container	77
[OK]		
kiasaki/alpine-postgres	PostgreSQL docker image based on Alpine Linux	44
[OK]		
jfloffe/alpine-python	A small, more complete, Python Docker image -	41
[OK]		
zenika/alpine-chrome	Chrome running in headless mode in a tiny AL...	35
[OK]		
hermsi/alpine-ssh	Dockerize your OpenSSH-server with rsync and...	35
[OK]		
byrnedo/alpine-curl	Alpine linux with curl installed and set as w...	35
[OK]		
hermsi/alpine-fpm-php	FPM-PHP 7.0 to 8.0, shipped along with tons -	25

Secara detail anda dapat melihat nama image, deskripsi singkat, stars, official dan automated. Stars menunjukkan popularitas dari image tersebut, sementara official adalah versi stable yang dikeluarkan langsung oleh pembuat / pengembang aplikasi / sistem operasi. Direkomendasikan untuk memilih Docker image berdasarkan kedua kriteria ini.

Selanjutnya, jalankan container secara background menggunakan image yang tersedia atau yang telah anda pilih setelah proses searching image. Gunakan perintah **docker run <options> <image-name>** untuk menjalankan container. Secara default, Docker akan menjalankan perintah menggunakan foreground mode, sehingga untuk menjalankan di mode background, anda perlu menambahkan opsi **-d**. Pada langkah ini, anda tidak memiliki Docker image di komputer anda, sehingga akan mendownload langsung dari Docker registry. Jika anda menjalankan container untuk kedua kalinya, maka local image yang akan digunakan.

Untuk melihat Docker images di host, anda dapat gunakan perintah :
docker images

Hasil output :

```
$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
redis	latest	b8477f2e393b	5 weeks ago	113MB
mongo	latest	cla14d3979c5	6 weeks ago	691MB
mariadb	10	b7220a722ce2	6 weeks ago	409MB
mariadb	latest	b7220a722ce2	6 weeks ago	409MB
ubuntu	latest	597ce1600cf4	6 weeks ago	72.8MB
postgres	12	fe603fe275ba	6 weeks ago	371MB
postgres	latest	6ce504119cc8	6 weeks ago	374MB
mysql	8	2fe463762680	6 weeks ago	514MB
mysql	latest	2fe463762680	6 weeks ago	514MB
alpine	latest	14119a10abf4	2 months ago	5.59MB
weaveworks/scope	1.11.4	a082d48f0b39	2 years ago	78.5MB

Sekarang kita bisa mencoba untuk menjalankan container. Agar mudah diingat, anda dapat menggunakan opsi **--name** sehingga container tersebut mudah untuk dikenali. gunakan perintah :

docker run -d nginx:alpine

Hasil output :

```
$ docker run -d nginx:alpine
Unable to find image 'nginx:alpine' locally
alpine: Pulling from library/nginx
97518928ae5f: Pull complete
a4e156412037: Pull complete
e0bae2ade5ec: Pull complete
3f3577460f48: Pull complete
e362c27513c3: Pull complete
a2402c2da473: Pull complete
Digest: sha256:12aa12ec4a8ca049537dd486044b966b0ba6cd8890c4c900ccb5e7e630e03df0
Status: Downloaded newer image for nginx:alpine
c2f1f945071e1df070419bafb11758c410399b7726e0ee02b181b9e3b595524
```

Dalam proses ini akan menjalankan image alpine sebagai container, karena Docker image alpine belum terdapat di local.

2. Melihat proses docker container

Docker ps dapat digunakan untuk melihat listing proses container yang berjalan, gunakan perintah :

`docker ps`

Hasil output :

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAME
dc2f1f945071	nginx:alpine	"/docker-entrypoint...."	3 minutes ago	Up 3 minutes	80/tcp	xenodochial_kapitsa

Untuk melihat lebih detail terkait volume, network, dan detail lain dari Docker container yang telah berjalan, anda dapat gunakan perintah :

`docker inspect <container id / friendly name>`

Sekarang kita coba lihat kembali hasil dari perintah sebelumnya tadi dengan menggunakan docker ps. pada kolom tersebut terdapat container ID. Ambil 3 karakter didepannya untuk menguji:

`docker inspect dc2`

Hasil output :

```
[
  {
    "Id":
"dc2f1f945071e1df070419bafb11758c410399b7726e0ee02b181b9e3b595524",
    "Created": "2021-11-14T12:33:01.969839778Z",
    "Path": "/docker-entrypoint.sh",
    "Args": [
      "nginx",
      "-g",
      "daemon off;"
    ],
    "State": {
      "Status": "running",
      "Running": true,
      "Paused": false,
      "Restarting": false,
      "OOMKilled": false,
      "Dead": false,
      "Pid": 1483,
      "ExitCode": 0,
      "Error": "",
      "StartedAt": "2021-11-14T12:33:04.339528931Z",
      "FinishedAt": "0001-01-01T00:00:00Z"
    },
    "Image":
"sha256:b46db85084b80a87b94cc930a74105b74763d0175e14f5913ea5b07c312870f8",
    "ResolvConfPath":
"/var/lib/docker/containers/dc2f1f945071e1df070419bafb11758c410399b7726e0ee02b181b9e3b595524/resolv.conf",
```

```
"HostnamePath":
"/var/lib/docker/containers/dc2f1f945071e1df070419bafb11758c410399b77
26e0ee02b181b9e3b595524/hostname",
"HostsPath":
"/var/lib/docker/containers/dc2f1f945071e1df070419bafb11758c410399b77
26e0ee02b181b9e3b595524/hosts",
"LogPath":
"/var/lib/docker/containers/dc2f1f945071e1df070419bafb11758c410399b77
26e0ee02b181b9e3b595524/dc2f1f945071e1df070419bafb11758c410399b7
726e0ee02b181b9e3b595524-json.log",
"Name": "/xenodochial_kapitsa",
"RestartCount": 0,
"Driver": "overlay",
"Platform": "linux",
"MountLabel": "",
"ProcessLabel": "",
"AppArmorProfile": "docker-default",
"ExecIDs": null,
"HostConfig": {
  "Binds": null,
  "ContainerIDFile": "",
  "LogConfig": {
    "Type": "json-file",
    "Config": {}
  },
  "NetworkMode": "default",
  "PortBindings": {},
  "RestartPolicy": {
    "Name": "no",
    "MaximumRetryCount": 0
  },
  "AutoRemove": false,
  "VolumeDriver": "",
  "VolumesFrom": null,
  "CapAdd": null,
  "CapDrop": null,
  "CgroupnsMode": "host",
  "Dns": [],
  "DnsOptions": [],
  "DnsSearch": [],
  "ExtraHosts": null,
  "GroupAdd": null,
  "IpcMode": "private",
  "Cgroup": "",
  "Links": null,
  "OomScoreAdj": 0,
  "PidMode": "",
  "Privileged": false,
  "PublishAllPorts": false,
  "ReadonlyRootfs": false,
  "SecurityOpt": null,
```

```
"UTSMode": "",
"UsersnsMode": "",
"ShmSize": 67108864,
"Runtime": "runc",
"ConsoleSize": [
  0,
  0
],
"Isolation": "",
"CpuShares": 0,
"Memory": 0,
"NanoCpus": 0,
"CgroupParent": "",
"BlkioWeight": 0,
"BlkioWeightDevice": [],
"BlkioDeviceReadBps": null,
"BlkioDeviceWriteBps": null,
"BlkioDeviceReadIOps": null,
"BlkioDeviceWriteIOps": null,
"CpuPeriod": 0,
"CpuQuota": 0,
"CpuRealtimePeriod": 0,
"CpuRealtimeRuntime": 0,
"CpusetCpus": "",
"CpusetMems": "",
"Devices": [],
"DeviceCgroupRules": null,
"DeviceRequests": null,
"KernelMemory": 0,
"KernelMemoryTCP": 0,
"MemoryReservation": 0,
"MemorySwap": 0,
"MemorySwappiness": null,
"OomKillDisable": false,
"PidsLimit": null,
"Ulimits": null,
"CpuCount": 0,
"CpuPercent": 0,
"IOMaximumIOps": 0,
"IOMaximumBandwidth": 0,
"MaskedPaths": [
  "/proc/asound",
  "/proc/acpi",
  "/proc/kcore",
  "/proc/keys",
  "/proc/latency_stats",
  "/proc/timer_list",
  "/proc/timer_stats",
  "/proc/sched_debug",
  "/proc/scsi",
  "/sys/firmware"
```

```

    ],
    "ReadonlyPaths": [
        "/proc/bus",
        "/proc/fs",
        "/proc/irq",
        "/proc/sys",
        "/proc/sysrq-trigger"
    ]
},
"GraphDriver": {
    "Data": {
        "LowerDir":
"/var/lib/docker/overlay/db3900e444ecc1acf68e4d376238d1d589deeb490b2
015cb7518e0eb4668a5e4/root",
        "MergedDir":
"/var/lib/docker/overlay/196b7da3c870d20befe35ff775252ba6bb36290f308
148c4d2ab4c9535bf1792/merged",
        "UpperDir":
"/var/lib/docker/overlay/196b7da3c870d20befe35ff775252ba6bb36290f308
148c4d2ab4c9535bf1792/upper",
        "WorkDir":
"/var/lib/docker/overlay/196b7da3c870d20befe35ff775252ba6bb36290f308
148c4d2ab4c9535bf1792/work"
    },
    "Name": "overlay"
},
"Mounts": [],
"Config": {
    "Hostname": "dc2f1f945071",
    "Domainname": "",
    "User": "",
    "AttachStdin": false,
    "AttachStdout": false,
    "AttachStderr": false,
    "ExposedPorts": {
        "80/tcp": {}
    },
    "Tty": false,
    "OpenStdin": false,
    "StdinOnce": false,
    "Env": [
        "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
        "NGINX_VERSION=1.21.4",
        "NJS_VERSION=0.7.0",
        "PKG_RELEASE=1"
    ],
    "Cmd": [
        "nginx",
        "-g",
        "daemon off;"
    ],

```

```

      "Image": "nginx:alpine",
      "Volumes": null,
      "WorkingDir": "",
      "Entrypoint": [
        "/docker-entrypoint.sh"
      ],
      "OnBuild": null,
      "Labels": {
        "maintainer": "NGINX Docker Maintainers <docker-
maint@nginx.com>"
      },
      "StopSignal": "SIGQUIT"
    },
    "NetworkSettings": {
      "Bridge": "",
      "SandboxID":
"d0d97315e62ebe4856192156fa4956f14c4e6e54c27e4a4a320677c93dca511
4",
      "HairpinMode": false,
      "LinkLocalIPv6Address": "",
      "LinkLocalIPv6PrefixLen": 0,
      "Ports": {
        "80/tcp": null
      },
      "SandboxKey": "/var/run/docker/netns/d0d97315e62e",
      "SecondaryIPAddresses": null,
      "SecondaryIPv6Addresses": null,
      "EndpointID":
"474a9a6b0c10293980cff08d9b16cb20127a93f63e66c91d8fced0b385ac2fa5"
    ,
      "Gateway": "172.18.0.1",
      "GlobalIPv6Address": "",
      "GlobalIPv6PrefixLen": 0,
      "IPAddress": "172.18.0.2",
      "IPPrefixLen": 24,
      "IPv6Gateway": "",
      "MacAddress": "02:42:ac:12:00:02",
      "Networks": {
        "bridge": {
          "IPAMConfig": null,
          "Links": null,
          "Aliases": null,
          "NetworkID":
"37f7e022d9808efe025920a58fe382dd4521fd06076c4755c1d7318e6640ba0
a",
          "EndpointID":
"474a9a6b0c10293980cff08d9b16cb20127a93f63e66c91d8fced0b385ac2fa5"
        ,
          "Gateway": "172.18.0.1",
          "IPAddress": "172.18.0.2",
          "IPPrefixLen": 24,

```



```

        "IPv6Gateway": "",
        "GlobalIPv6Address": "",
        "GlobalIPv6PrefixLen": 0,
        "MacAddress": "02:42:ac:12:00:02",
        "DriverOpts": null
    }
}
}
]

```

Sementara untuk melihat catatan sistem log untuk container anda dapat menggunakan perintah :

`docker logs <container id / friendly name>`

Perintah ini digunakan untuk proses troubleshooting ketika layanan didalam container tidak berjalan atau ada problem pada container. Perintah yang digunakan secara lengkap adalah sebagai berikut (asumsi masih menggunakan container ID yang sama dg perintah sebelumnya):

`docker logs dc2`

Hasil output :

```

$ docker logs dc2
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2021/11/14 12:33:04 [notice] 1#1: using the "epoll" event method
2021/11/14 12:33:04 [notice] 1#1: nginx/1.21.4
2021/11/14 12:33:04 [notice] 1#1: built by gcc 10.3.1 20210424 (Alpine 10.3.1_git20210424)
2021/11/14 12:33:04 [notice] 1#1: OS: Linux 5.4.0-88-generic
2021/11/14 12:33:04 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2021/11/14 12:33:04 [notice] 1#1: start worker processes
2021/11/14 12:33:04 [notice] 1#1: start worker process 31
2021/11/14 12:33:04 [notice] 1#1: start worker process 32

```

Jika ingin menggunakan perintah inspect atau logs pada docker, anda perlu tahu id container atau nama dari container. Sebelum menjalankan perintah logs atau inspect, jalankan dulu perintah **docker ps**.

3. Binding Port

Container yang berjalan terkadang perlu untuk dapat di akses secara eksternal, untuk memudahkan Docker dapat melakukan binding port untuk host dan container. Dalam contoh dibawah ini, melakukan binding pada port `<host port>:<container port>` dengan menggunakan opsi `-p`. `--name` adalah opsi untuk memberikan *friendly name (tag)* pada container, sementara itu `nginx:alpine` adalah image yang digunakan untuk membuat container. Dibawah ini adalah contoh implementasi perintahnya

```

$ docker run -d --name nginxPort -p 8080:8080 nginx:alpine
f5ce26f1f81e0c9c923eef53a7fb062759e4db9f8eadc40db649fd2fcb9adab1

```

Setelah berjalan, anda dapat melakukan pengecekan dengan melihat proses dari Docker. Pada hasil dibawah, terlihat ada 2 proses container yang sedang berjalan. Jika menggunakan perintah `docker ps`, akan menampilkan seperti berikut:

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAME
f5ce26f1f81e	nginx:alpine	"/docker-entrypoint. ..."	About a minute ago	Up About a minute	80/tcp, 0.0.0.0:8080->8080/tcp, :::8080->8080/tcp	Nginx Port
dc2f1f945071	nginx:alpine	"/docker-entrypoint. ..."	22 minutes ago	Up 22 minutes	80/tcp	xenodochial_kapitsa

Docker juga memungkinkan container dapat berjalan sekaligus pada 1 lingkungan host, sehingga container port akan di konfigurasi secara default. Berikut contoh implementasinya:

```
$ docker run -d --name nginxdynamic -p 8080 nginx:alpine
87570e7d7f34c1f2aebd644dcebb83a78f9abbc0d9273784efa4f347079ae193
```

4. Binding Data

Containers secara desain bersifat stateless. Seluruh data yang kita inginkan untuk tetap ada setelah container berhenti dijalankan harus tersimpan pada mesin host. Untuk mengatasi persoalan ini, dapat dilakukan dengan cara binding host direktori ke dalam container. Dibawah ini adalah contoh implementasi:

```
$ mkdir bindingdata
$ docker run -d --name nginxMapDir -v "$PWD/bindingdata":/usr/share/nginx/html nginx:alpine
27a488fee65a69cdafb49d78321812d8ef8d3883a28f53a6da954489e3b62f52
```

5. Membuat Docker Images

Docker memungkinkan untuk membuat image sendiri dengan base Docker image yang tersedia. Untuk membuat diperlukan file Dockerfile. Berikut ini langkah untuk membuat image tersebut, dalam contoh ini, seluruh file kebutuhan diletakkan pada direktori imageku. Buat direktori imageku dengan perintah:

```
mkdir imageku
```

Masuk ke dalam direktori tersebut, dengan perintah:

```
cd imageku
```

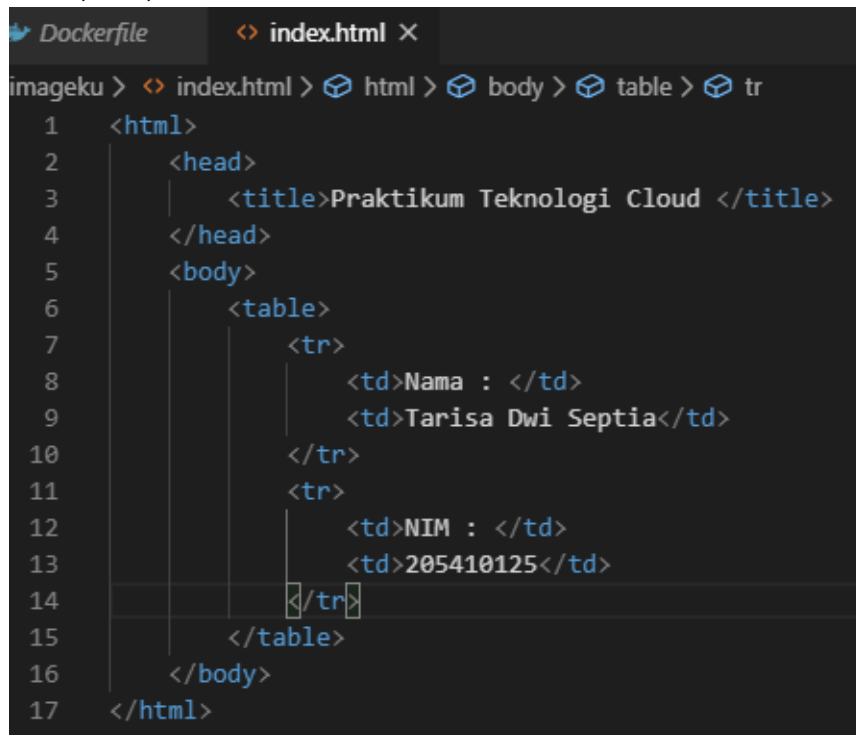
Lalu gunakan text editor pico untuk membuat file bernama Dockerfile :

```
pico Dockerfile
```

Kemudian copy paste baris code berikut

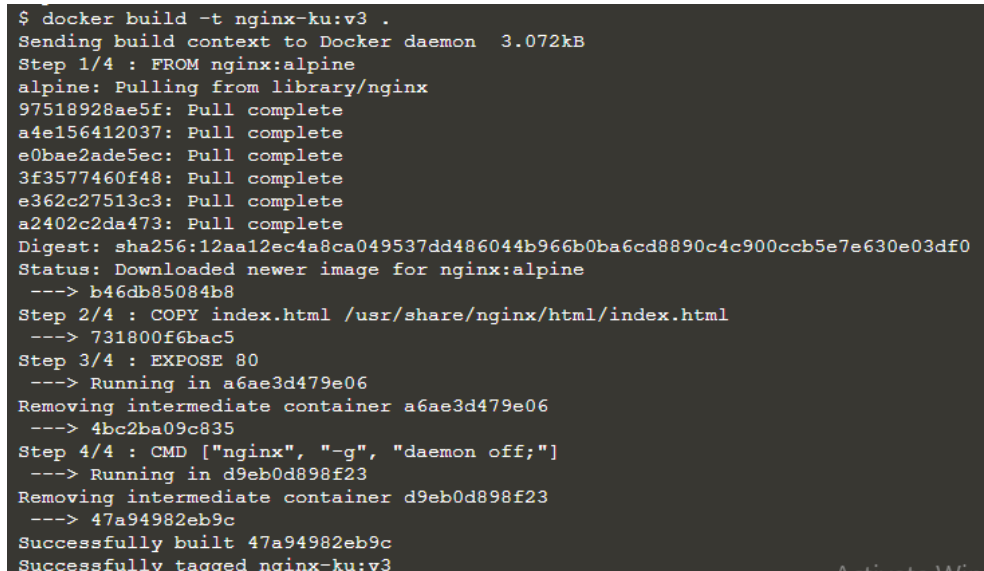
```
GNU nano 4.8 Dockerfile
FROM nginx:alpine
COPY index.html /usr/share/nginx/html/index.html
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

Selanjutnya, buatlah file html sederhana bernama *index.html* untuk menampilkan NAMA, NIM, dan DESKRIPSI DIRI.



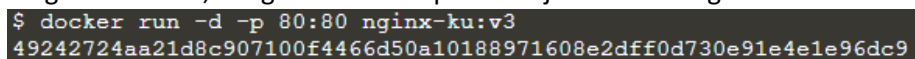
```
Dockerfile  index.html X
imageku > index.html > html > body > table > tr
1  <html>
2    <head>
3      <title>Praktikum Teknologi Cloud </title>
4    </head>
5    <body>
6      <table>
7        <tr>
8          <td>Nama : </td>
9          <td>Tarisa Dwi Septia</td>
10       </tr>
11       <tr>
12         <td>NIM : </td>
13         <td>205410125</td>
14       </tr>
15     </table>
16   </body>
17 </html>
```

Setelah seluruh file tersebut tersedia, anda dapat membuat image dengan format penamaan nama-aplikasi:versi, contoh nginx-ku:v3 :



```
$ docker build -t nginx-ku:v3 .
Sending build context to Docker daemon 3.072kB
Step 1/4 : FROM nginx:alpine
alpine: Pulling from library/nginx
97518928ae5f: Pull complete
a4e156412037: Pull complete
e0bae2ade5ec: Pull complete
3f3577460f48: Pull complete
e362c27513c3: Pull complete
a2402c2da473: Pull complete
Digest: sha256:12aa12ec4a8ca049537dd486044b966b0ba6cd8890c4c900ccb5e7e630e03df0
Status: Downloaded newer image for nginx:alpine
----> b46db85084b8
Step 2/4 : COPY index.html /usr/share/nginx/html/index.html
----> 731800f6bac5
Step 3/4 : EXPOSE 80
----> Running in a6ae3d479e06
Removing intermediate container a6ae3d479e06
----> 4bc2ba09c835
Step 4/4 : CMD ["nginx", "-g", "daemon off;"]
----> Running in d9eb0d898f23
Removing intermediate container d9eb0d898f23
----> 47a94982eb9c
Successfully built 47a94982eb9c
Successfully tagged nginx-ku:v3
```

Langkah terakhir, image tersebut dapat anda jalankan sebagai container.



```
$ docker run -d -p 80:80 nginx-ku:v3
49242724aa21d8c907100f4466d50a10188971608e2dff0d730e91e4e1e96dc9
```

Hasil :

Nama : Tarisa Dwi Septia

NIM : 205410125

D. Kesimpulan

Setelah melakukan praktik diatas sesuai dengan tujuan pada modul 7 ini, dapat disimpulkan bahwa mahasiswa dapat mempelajari dan mengimplementasikan container docker