GETTING STARTED WITH DOCKER

Introduction to Docker

Apa itu docker?

Docker adalah layanan yang menyediakan kemampuan untuk mengemas dan menjalankan sebuah aplikasi dalam sebuah lingkungan terisolasi yang disebut dengan container. Dengan adanya isolasi dan keamanan yang memadai memungkinkan kamu untuk menjalankan banyak container di waktu yang bersamaan pada host tertentu.

Docker ini diperkenalkan pada tahun 2013 oleh Solomon Hykes pada acara PyCon. Beberapa bulan setelahnya docker secara resmi diluncurkan, tepatnya pada tahun 2014. Semenjak itu docker menjadi sangat populer di kalangan developer luar negeri, tetapi belum terlalu populer di Indonesia.

Fitur-fitur docker

Setelah mengetahui pengertiannya, sekarang kita masuk ke fitur-fitur dari docker yang dapat kamu gunakan sesuai dengan kebutuhanmu.

- Docker engine
  Yang pertama ada docker engine. Ia digunakan untuk membuat image dan container.
- Docker Hub
  Selanjutnya adalah docker hub. Ia adalah registry yang berisikan kumpulan dari image-image. Dengan menggunakan docker hub ini kamu dapat mengumpulkan image. Hub ini berbeda dengan docker engine yang hanya membuat image.
- Docker Compose
  Docker compose ini adalah salah satu fitur unggulan yang berfungsi untuk menjalankan beberapa container atau biasa disebut multi-container sehingga dapat menghemat banyak waktu.
- Docker for Mac
  Untuk fitur yang satu ini, kamu pasti sudah tau dari namanya. Fitur ini memungkinkan pengguna docker untuk menjalankan container pada sistem operasi Mac.
- Docker for Linux

Sama seperti fitur sebelumnya, fitur ini juga memungkinkan penggunanya untuk menjalankan container pada sistem operasi Linux.

- Docker for Windows
  Fitur terakhir dan sudah pasti fitur yang paling banyak digunakan dibandingkan dengan fitur-fitur lainnya yaitu docker for windows. Fitur ini memungkinkan penggunanya untuk menjalankan kontainer pada sistem operasi windows.

Cara kerja Docker

Docker bertindak sebagai alat yang digunakan untuk menjalankan container. Container ini bertindak seperti mesin virtual, yang seperti simulasi komputer yang berjalan di dalam komputer asli pengembang. Pada mesin virtual ini nantinya semua kode sistem tersimpan untuk menjalankan simulasi seolah-olah adalah operasi sistem utama. Docker bertindak melakukan virtualisasi sistem operasi di dalam sistem operasi host.

Docker membangun container berdasarkan gambar yang berisi kode program. Gambar atau images ini ditumpuk satu sama lain untuk lantas membangun pengaturan yang lengkap. Gambar bertumpuk dapat berbagi gambar inti yang sama, seperti cabang-cabang dari batang pohon yang sama.

Proses ini dapat dimisalkan ketika seorang pengembang ingin menguji tampilan situs web baru di sebuah browser web berbeda, tetapi pengembang tidak ingin langsung melakukan instalasi setiap browser ke komputernya. Melakukan hal ini dapat menyebabkan masalah dengan peramban pribadi pengembang. Dalam pengujian inilah nantinya Docker berguna sebagai tempat uji coba.

Docker bekerja menggunakan sistem arsitektur client-server. Dalam hal ini nantinya klien akan berkomunikasi dengan apa yang disebut Daemon Docker atau proses pengelolaan Docker images, container, network, dan volume penyimpanan. Docker Daemon nantinya akan menerima permintaan pemrosesan dari Docker Engine Rest API yang berguna untuk interaksi dan bisa diakses oleh klien melalui hypertext transfer protocol (HTTP).

Klien Docker yang lain adalah Docker Compose yang dapat memungkinkan pengembang untuk bekerja dengan aplikasi yang terdiri dari sekumpulan container. Dalam ilustrasi di atas container yang dipakai adalah Ubuntu, sehingga nantinya akan menempel secara interaktif ke sesi baris perintah yang dimiliki pengembang.

Saat pengembang mulai menjalankan perintah terhadap container Ubuntu maka akan terjadi konfigurasi oleh Docker dan pembuatan container baru. Selanjutnya, Docker akan mengalokasikan dokumen sistem ke dalam container sebagai lapisan

terakhirnya. Hal ini memungkinkan container yang sedang berjalan untuk membuat atau memodifikasi dokumen dan direktori ke sistem dokumen lokal.

Di samping itu, Docker juga membuat antarmuka jaringan untuk menghubungkan container ke sebuah jaringan default. Hal ini terjadi jika pengembang tidak menentukan opsi jaringan apapun. Termasuk seperti penetapan alamat internet protocol (IP) ke dalam container. Secara default, container akan terhubung ke jaringan eksternal menggunakan koneksi dari jaringan host.

Docker akan mulai mengeksekusi container yang berjalan secara interaktif dan terpasang di terminal pemrograman milik pengembang. Dalam tahap ini, pengembang dapat memberikan input menggunakan keyboard dan juga saat output tercatat ke terminal milik pengembang. Pengembang dapat mengetik “exit” untuk menghentikan perintah, meski begitu container yang dihentikan tidak akan terhapus dan pengembang tetap bisa menggunakannya atau menghapusnya lain waktu.

Kelebihan

Dalam penggunaannya, docker memiliki beragam manfaat atau kelebihan yang menjadikannya populer di kalangan developer. Berikut ini adalah beberapa kelebihannya.

- Memiliki konfigurasi yang sederhana
  Docker memiliki konfigurasi yang cukup sederhana dan dapat kamu sesuaikan dengan kebutuhan aplikasi yang sedang kamu kembangkan. Hanya dengan menentukan beberapa kode, ia akan membuat environment sendiri yang berbeda dengan environment dari server utama.
- Tingkat keamanan yang baik
  Docker memiliki tingkat keamanan yang baik. Ia akan memastikan aplikasi yang sedang berjalan tidak dapat memengaruhi container. Selain itu, ia juga memiliki fitur keamanan lain seperti pengaturan OS host mount dengan akses read-only sehingga tidak akan mengubah konfigurasi apa pun, kecuali ada yang memiliki akses secara penuh.
- Dapat dijalankan pada beberapa platform cloud
  Salah satu penyebab docker banyak diminati oleh banyak perusahaan adalah karena ia dapat dijalankan pada beberapa platform cloud. Dengan begitu, penggunanya akan lebih fleksibel dalam melakukan porting aplikasi.
- Dapat melakukan debugging
  Kelebihan berikutnya adalah ia dapat melakukan debugging. Waktu yang dibutuhkannya juga tergolong cepat, yakni hanya sekitar satu menit saja untuk melakukan proses debug pada Sandbox.
- Dapat digunakan pada berbagai sistem operasi

Sebelumnya kamu sudah mengetahui fitur dari docker yang dapat berjalan di sistem operasi seperti Windows, Mac, dan Linux. Hal tersebut akan memudahkan pengguna dari fleksibilitas.

Kesimpulan?

Jadi, docker adalah layanan yang menyediakan kemampuan untuk mengemas dan menjalankan aplikasi dalam suatu lingkungan terisolasi yang disebut dengan container. Dari kelebihan yang sudah dijelaskan di atas dapat disimpulkan bahwa docker dapat membantu untuk meningkatkan produktivitas dari developer dalam membuat perangkat lunak yang berkualitas. Jadi, itulah pembahasan kali ini. Semoga kamu menjadi lebih mengerti mengenai docker.

Installing and Run Docker

Lab 1 : Installing Docker

sudo apt install docker.io

2. Menampilkan versi docker.

docker version

3. Menampilkan detail instalasi docker.

docker info

4. Uji instalasi docker.

docker run hello-world

5. Menampilkan image yang sudah di-download.

docker image ls

6. Menampilkan semua container (active ataupun exit).

docker container ls -a

7. Menambahkan user ke dalam group docker.

groupadd docker
su - ratix
sudo usermod -aG docker $USER

8. Exit terminal kemudian masuk kembali.

9. Test docker tanpa sudo.

docker container ls

lab 2 : Docker run part 1

1. Check image redis

docker search redis

2. Menjalankan image redis

docker run redis # CTRL + c untuk keluar
docker run -d redis # Berjalan di belakang layar (Background)
docker run -d --name redis1 redis # Memberi nama pada container

3. Menampilkan container yang berjalan.

docker ps
docker container ls

4. Menampilkan semua container docker.

docker ps -a
docker container ls -a

5. Menampilkan deskripsi container .

docker inspect CONTAINER_NAME/CONTAINER_ID
docker inspect redis1

6. Menampilkan isi logs dari container.

docker logs CONTAINER_NAME/CONTAINER_ID
docker logs redis1

7. Menampilkan live stream resource yang terpakai pada container.

docker stats CONTAINER_NAME/CONTAINER_ID
docker stats redis1

8. Menampilkan running process pada container.

docker top CONTAINER_NAME/CONTAINER_ID
docker top redis1

9. Mematikan container.

docker stop CONTAINER_NAME/CONTAINER_ID
docker stop redis1

docker run (latihan02)

1. Check image nginx.

docker search nginx

2. Menjalankan image nginx dan mengexpose port host.

docker run -d --name nginx1 -p 80:80 nginx:latest

3. Menampilkan deskripsi container nginx.

docker inspect nginx1

4. Menjalankan image nginx dan mendeklarasikan port container.

docker run -d --name nginx2 -p 80 nginx:latest

5. Uji browsing.

curl localhost:$(docker port nginx2 80| cut -d : -f 2)

6. Menampilkan container (activate/exit).

docker ps -a
docker container ls -a

7. Menampilkan docker image.

docker images

lab 3 : Docker run part 2

1. Check image nginx.

docker search nginx

2. Menjalankan image nginx dan mengexpose port.

docker run -d --name nginx1 -p 8080:80 nginx:latest

3. Menampilkan deskripsi container nginx.

docker inspect nginx1

4. Menjalankan image nginx dan mengexpose port.

docker run -d --name nginx2 -p 8081:80 nginx:latest

5. Menampilkan container (activate/exit).

docker ps -a
docker container ls -a

6. Check access pada container.

curl localhost:8080
curl localhost:8081

7. Masuk ke dalam container.

docker exec -it nginx2 /bin/bash

8. Update dan install editor pada container.

apt-get update -y && apt-get install nano -y

9. Edit pada index.html dan ubah pada welcome to nginx.

nano index.html
...
hello, testing.
...

mv index.html /usr/share/nginx/html

10. Restart service nginx.

service nginx restart

11. Menjalankan ulang container.

docker start nginx2

12. Menampilkan container.

docker ps
docker container ls

13. Check access pada container.

curl localhost:8080
curl localhost:8081

14. Menampilkan deskripsi container .

docker inspect nginx1
docker inspect nginx2

15. Menampilkan isi logs dari container.

docker logs nginx1
docker logs nginx2

16. Menampilkan live stream resource yang terpakai pada container.

docker stats nginx1
docker stats nginx2

17. Menampilkan running process pada container.

docker top nginx1
docker top nginx2

docker run (latihan04)

1. Check image ubuntu.

docker search ubuntu

2. Mengambil image ubuntu.

docker pull ubuntu

3. Menjalankan container dan langsung menuju consolenya.

docker run -it --name ubuntu1 ubuntu

keluar dari container dengan Ctrl+d atau exit

docker ps -a

4. Menjalankan container dan langsung dihapus.

docker run -it --rm --name ubuntu2 ubuntu

keluar dari container dengan Ctrl+d atau exit

docker ps -a

Lab 4 : Docker Run part 3

1. Menjalankan container mysql.

docker run -d --name my-own-mysql -e MYSQL_ROOT_PASSWORD=RAHASIA -e MYSQL_DATABASE=latihan05 -p 3306:3306 mysql

2. Pull image phpmyadmin.

docker pull phpmyadmin/phpmyadmin:latest

3. Menjalankan container phpmyadmin dan menghubungkannya dengan container mysql.

docker run --name my-own-phpmyadmin -d --link my-own-mysql:db -p 8090:80 phpmyadmin/phpmyadmin

4. Uji browsing.

ssh root@labX.adinusa.id -p10XX1 -DYYYY

buka dibrowser http://10.X.X.10:8090 kemudian login dengan user: `root` dan password: `RAHASIA`

docker run (latihan06)

1. Jalankan container ubuntu.

docker run -dit --name ubuntu1 ubuntu
docker run -dit --name ubuntu2 ubuntu

2. Menampilkan daftar container.

docker ps

3. Pause container ubuntu.

docker pause ubuntu1
docker pause ubuntu2

4. Check pada daftar container jika status container sudah menjadi pause.

docker ps

5. Check pemakaian resource pada saat container ubuntu dalam keadaan pause.

docker stats ubuntu1
docker stats ubuntu2

6. Unpause container ubuntu1.

docker unpause ubuntu1

docker run (latihan07)

1. Membuat container database.

docker container run -d --name ch6_mariadb --memory 256m --cpu-shares 1024
--cap-drop net_raw -e MYSQL_ROOT_PASSWORD=test mariadb:5.5

2. Membuat container wordpress.

docker container run -d -p 80:80 -P --name ch6_wordpress --memory 512m
--cpu-shares 512 \
--cap-drop net_raw --link ch6_mariadb:mysql -e
WORDPRESS_DB_PASSWORD=test wordpress:5.0.0-php7.2-apache

3. Check logs, running process, and resource.

docker logs ch6_mariadb
docker logs ch6_wordpress

docker top ch6_mariadb
docker top ch6_wordpress

docker stats ch6_mariadb
docker stats ch6_wordpress

4. Uji browsing.

buka di browser http://10.X.X.10 kemudian selesaikan tahap instalasinya.

Managing Docker Container

Merancang dan membuat kode Dockerfile untuk membuat image container khusus.

Tujuan:
Memahami pendekatan untuk membuat image container khusus.
Membuat image container menggunakan perintah Dockerfile.

Lab 2.1 : Mount Volume

1. membuat working directory.

```
cd $HOME
mkdir -p latihan/latihan01-volume
cd latihan/latihan01-volume
```

2. membuat container untuk data volume dengan nama test-volume.

```
for i in {1..10};do touch file-$i;done
sudo docker create -v /test-volume --name test-volume busybox
sudo docker cp . test-volume:/test-volume
```

3. mount volumes.

```
sudo docker run --volumes-from test-volume ubuntu ls /test-volume
```

Lab 2.2 : Mount Volume with Read-only Mode

1. Membuat volume

docker volume create my-vol

2. Menampilkan daftar volume.

docker volume ls

3. Menampilkan detil volume

docker volume inspect my-vol

4. Jalankan container dengan volume

docker run -d --name=nginx-test -v my-vol:/usr/share/nginx/html -p 4005:80 nginx:latest

5. Tampilkan alamat IP container

docker inspect nginx-test | grep -i ipaddress

6. Uji browsing ke alamat IP container.

curl http://172.17.XXX.XXX

7. Buat file index.html dan pindahkan ke direktori source volume

sudo echo "This is from my-vol source directory" > index.html
sudo mv index.html /var/lib/docker/volumes/my-vol/_data

8. Uji browsing ke alamat IP container.

curl http://172.17.XXX.XXX

9. Jalankan container dengan read-only volume

docker run -d --name=nginxtest-rovol -v my-vol:/usr/share/nginx/html:ro nginx:latest

10. Tampilkan detil container nginxtest-rovol dan perhatikan Mode di section Mounts

docker inspect nginxtest-rovol

Lab 2.3 : Default Bridge Network

1. Tampilkan daftar network saat ini.

sudo docker network ls

2. Jalankan 2 container alpine yang menjalankan shell ash.

docker run -dit --name alpine1 alpine ash
docker run -dit --name alpine2 alpine ash
docker container ls

3. Buat network baru dan tambahkan ke container.

docker network create --driver bridge bridge1
docker network connect bridge1 alpine1

4. cek ip container alpine2.

docker inspect alpine2 | grep -i ipaddress

5. Masuk ke container alpine1.

docker exec -it alpine1 sh

6. Tampilkan alamat IP container alpine1.

ip add

7. Uji ping ke internet (sukses).

ping -c 3 8.8.8.8

8. Uji ping ke alamat IP container alpine2 (sukses).

ping -c 3 172.17.YYY.YYY

Uji ping ke nama container alpine2 (gagal).

ping -c 3 alpine2

10. Keluar dari container alpine1 tanpa menutup shell tekan Ctrl+P, Ctrl+Q.

Lab 2.4 : Host Network

1. Jalankan container dari image nginx.

sudo docker run -itd --network host --name my_nginx nginx

2. Uji browsing ke localhost.

curl http://localhost

3. Verifikasi alamat IP dan bound port 80.

ip add
sudo netstat -tulpn | grep :80

Creating Custom Docker Container Image

Lab 3.1: Exploring Dockerfile

Dockerfile (latihan01)

1. Clone repository latihan.

```
git clone https://github.com/spkane/docker-node-hello.git \
        --config core.autocrlf=input latihan01
```

2. Masuk ke dalam direktori.

```
cd latihan01
```

3. Buat image.

```
docker build -t node-latihan01 .
```

4. Buat container.

```
docker run -d --rm --name node-latihan01 -p 8080:8080 node-latihan01
```

5. Coba akses port.

```
curl localhost:8080
```

Dockerfile (latihan02)

1. Buat direktori latihan02.

```
cd $HOME
mkdir latihan02
cd latihan02
```

2. Buat file Dockerfile.

```
vim Dockerfile
...
# Use whalesay image as a base image
FROM docker/whalesay:latest
```

```
# Install fortunes
RUN apt -y update && apt install -y fortunes

# Execute command
CMD /usr/games/fortune -a | cowsay
...
```

3. Bangun image dari Dockerfile.

```
docker build -t docker-whale .
```

4. Tampilkan image yang sudah dibangun.

```
docker image ls
```

5. Uji jalankan image.

```
docker run docker-whale
```

6. Menampilkan container.

```
docker ps
docker container ls -a
```

Lab 3.2: Exploring Dockerfile (Flask Apps)

1. Jika belum punya Docker ID, register di https://id.docker.com.

2. Login dengan Docker ID.

```
sudo docker login
```

3. Buat direktori latihan03.

```
cd $HOME
mkdir latihan03
cd latihan03
```

4. Membuat file flask.

```
vim app.py
...
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
        return 'Hey, we have Flask in a Docker container!'


if __name__ == '__main__':
        app.run(debug=True, host='0.0.0.0')
...
```

5. Membuat requirements.txt.

```
vi requirements.txt
...
Flask==0.10.1
Werkzeug==1.0.0
Jinja2==2.8.1
MarkupSafe==1.1.1
itsdangerous==1.1.0
...
```

6. Membuat file Dockerfilenya.

vim Dockerfile
...
FROM ubuntu:16.04
RUN mkdir /app
RUN apt-get update -y && \
        apt-get install python-pip python-dev -y

COPY ./requirements.txt /app
COPY . /app

WORKDIR /app
RUN pip install -r requirements.txt

ENTRYPOINT ["python"]
CMD ["app.py"]
...

7. Buat image dari Dockerfile.

docker build -t flask-latihan03 .

8. Tag image.

sudo docker tag flask-latihan03 [username]/flask-latihan03:latest

9. Push image.

sudo docker push [username]/flask-latihan03:latest

10. Menjalankan image .

sudo docker run -d -p 5000:5000 --name flask03 [username]/flask-latihan03

11. Uji browsing.

curl localhost:5000

Docker Compose

Lab 4.1: Using Docker Compose

1. Unduh Compose.

sudo apt install -y docker-compose

2. Uji instalasi.

sudo docker-compose --version

Eksekusi di pod-[username]-node01

3. Buat direktori my_wordpress dan masuk ke direktori tersebut.

```
cd $HOME
mkdir -p latihan/my_wordpress
cd latihan/my_wordpress
```

4. Buat file docker-compose.yml.

vim docker-compose.yml

```
version: '3.2'

services:
  db:
        image: mysql:5.7
        volumes:
        - dbdata:/var/lib/mysql
        restart: always
        environment:
        MYSQL_ROOT_PASSWORD: somewordpress
        MYSQL_DATABASE: wordpress
        MYSQL_USER: [username]
        MYSQL_PASSWORD: [password]

  wordpress:
        depends_on:
        - db
        image: wordpress:latest
        ports:
        - "8000:80"
        restart: always
        environment:
```

```
        WORDPRESS_DB_HOST: db:3306
        WORDPRESS_DB_USER: [username]
        WORDPRESS_DB_PASSWORD: [password]
volumes:
        dbdata:
```

5. Jalankan compose.

```
sudo docker-compose up -d
```

6. Tampilkan daftar container dan uji browsing ke ke halaman wordpress yang sudah dibuat.

```
sudo docker container ls
```

CI Using Docker

Just summary

Logging and Error Handling

Lab 7.1: Log Check

1. Check history image.

docker history nginx:latest

2. Jalankan nginx.

docker run -dit -p 80:80 --name nginx1 nginx

3. Check Filesystem pada nginx.

docker diff nginx1

4. Cek Log.

docker logs --details nginx1
docker logs --timestamps nginx1
docker logs nginx1

Logging Driver

Lab 8.1: Configuring Logging Driver

1. Buat file daemon.json.

```
vim /etc/docker/daemon.json
...
{
  "log-driver": "json-file",
  "log-opts": {
            "max-size": "10m",
            "max-file": "100"
  }
}
...
```

2. Restart Daemon dan Docker.

```
systemctl daemon-reload
systemctl restart docker
```

3. Jalankan container.

```
docker run --log-driver json-file --log-opt max-size=10m alpine echo hello world
```

4. Check log pada direktori docker.

```
cat /var/lib/docker/containers/CONTAINER/xxx-json.log
```

Note: Untuk ID-nya bisa dilihat dari perintah docker ps -a.

Health Check

Lab 9.1: Health Check

Health Check latihan01

1. Buat direktori.

```
cd $HOME
mkdir hc-latihan01
cd hc-latihan01
```

2. Buat file Dockerfile.

```
vim Dockerfile
...
FROM katacoda/docker-http-server:health
HEALTHCHECK --interval=1s --retries=3 \
        CMD curl --fail http://localhost:80/ || exit 1
...
```

3. Buat image.

```
docker build -t http-healthcheck .
```

4. Jalankan image.

```
docker run -d -p 80:80 --name http-healthcheck http-healthcheck
```

5. Check image.

```
docker ps
# check pada bagian status
```

6. check dengan curl.

```
curl http://localhost/
# maka dapat diakses
```

7. Check untuk di unhealthy.

```
curl http://localhost/unhealthy
```

8. Check pada docker container status.

docker container ls

9. Check dengan curl.

curl http://localhost/

Health Check latihan02

1. Buat direktori baru.

```
cd $HOME
mkdir hc-latihan02
cd hc-latihan02
```

2. Buat file server.js.

```
vim server.js
...
"use strict";
const http = require('http');

function createServer () {
        return http.createServer(function (req, res) {
        res.writeHead(200, {'Content-Type': 'text/plain'});
        res.end('OK\n');
        }).listen(8080);
}

let server = createServer();

http.createServer(function (req, res) {
        res.writeHead(200, {'Content-Type': 'text/plain'});
        if (server) {
        server.close();
        server = null;
        res.end('Shutting down...\n');
        } else {
        server = createServer();
        res.end('Starting up...\n');
        }
}).listen(8081);
```

...

3. Buat file Dockerfile.

vim Dockerfile
...
FROM node
COPY server.js /
EXPOSE 8080 8081
HEALTHCHECK --interval=5s --timeout=10s --retries=3 CMD curl -sS
127.0.0.1:8080 || exit 1
CMD ["node","server.js"]
...

4. Buat image.

docker build -t node-server .

5. Jalankan image.

docker run -d --name nodeserver -p 8080:8080 -p 8081:8081 node-server

6. Check container.

curl 127.0.0.1:8080
docker ps
docker inspect --format "{{ json .State.Health }}" nodeserver

7. Check container.

curl 127.0.0.1:8081
docker ps
docker inspect --format "{{ json .State.Health }}" nodeserver

8. Check Container.

curl 127.0.0.1:8081
docker ps
docker inspect --format "{{ json .State.Health }}" nodeserver

Docker Security

Just summary

Storage Driver

Lab 10.1: Configuring Storage Driver

1. Edit file daemon.json.

```
vim /etc/docker/daemon.json
...
{
  "storage-driver": "vfs"
}
...
```

2. Restart service docker.

```
sudo service docker restart
```

3. Check pada docker info.

```
docker info
```

4. Check dengan docker pull.

```
docker pull ubuntu
```

5. Check pada direktori docker.

```
ls -al /var/lib/docker/vfs/dir/
du -sh /var/lib/docker/vfs/dir/
```