Daftar Isi

Daftar Gambar

Abcdefg

Kata Pengantar

Docker adalah

BAB I
Warming UP

Dalam bab ini kita akan membahas bagaimana memulai menggunakan docker. Pada tahapan awal docker akan di download dan instal pada PC / Laptop dengan beberapa sistem operasi yang berbeda, begitu juga dasar-dasar perintah yang sering digunakan untuk berinteraksi dengan docker.

Container dan Hypervisor

Dalam sebuah teknologi virtualisasi, mungkin anda kenal dengan beberapa perangkat lunak seperit Proxmox, VMware, Xen, Virtualbox, Hyper-V dan lain-lain. Beberapa perangkat lunak tersebut adalah jenis Hypervisor atau Virtual Machines. Hypervisor sendiri ada dua kategori, yaitu Native Hypervisor dan Hosted Hypervisor.

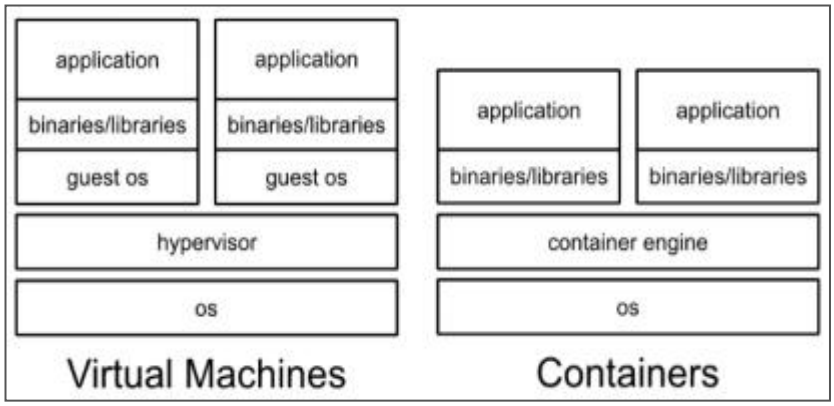
Native Hypervisor merupakan virtual mesin yang berjalan / diinstalasi langsung pada hardware (PC). Contohnya : Proxmox, VMware ESX, Hyper-V, dll.

Sedangkan Hosted Hypervisor adalah virtual mesin yang diinstalasikan diatas operating system, yang kemudian guest operating sistem berjalan pada proses aplikasi virtual mesin tersebut. Contohnya : VirtualBox, VMware Player / Workstation, Parallels Desktop, maupun Qemu.

Pada intinya sebuah hypervisor akan menyediakan layanan virtualisasi lengkap sebuah virtual operating system, atau yang disebut dengan guest os yang didalam guest os tersebut diinstalasikan library dan dependency untuk menjalankan suatu layanan (aplikasi).

Maka dari itu kebutuhan resource (cpu, ram, hdd, dll) akan semakin besar seiring dengan banyaknya menjalankan guest OS, begitu juga diketahui, untuk menjalankan suatu guest OS akan memakan waktu yang relatif lebih lama.

Sedangkan Container, juga merupakan virtul operating system yang dapat mengisolasi user space, proses, maupun file system. Namun container menggunakan shared kernel, pada sistem operasi host yang dijalankannya. Share kernel memugkinkan juga berbagi library dan dependency yang dibutuhkan, sehingga sebuah layanan (aplikasi) dapat dijalankan tanpa harus menghidupkan guest os dahulu, proses tersebut dihilangkan, sehingga kebutuhan resource (cpu, ram) berkurang dan waktu tempuh untuk menjalankan layanan semakin cepat.



Gambar 1.1

Docker, apaan sih ?

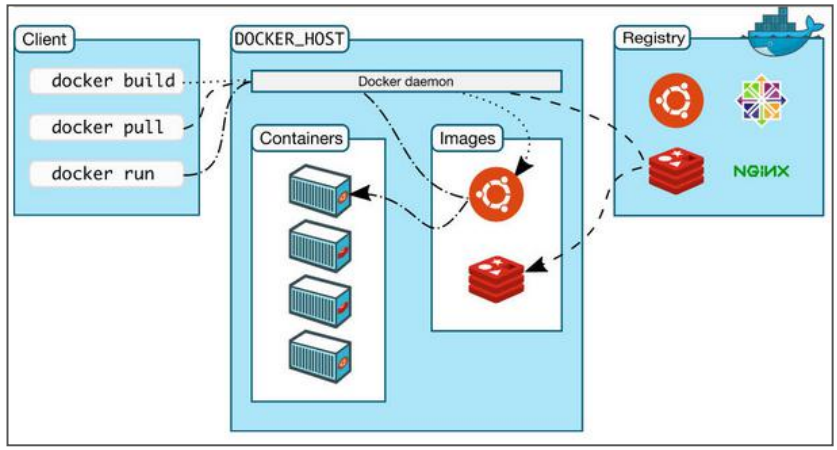
Definisi dari docker dapat kita lihat pada official website dari docker sendiri, yang beralamat di https://www.docker.com/

"Docker is an open platform for developers and sysadmins to build, ship, and run distributed applications."

Dalam arti yang lebih praktis, docker adalah sebuah tools yang memungkinkan penyertaan sebuah service dalam environment terisolasi, yang disebut dengan **Container.** Sebuah container dapat mengemas library maupun dependency yang dibutuhkan oleh suatu layanan (aplikasi), sehingga yakin bahwa layanan tersebut dapat developer jalankan dimanapun docker berjalan.

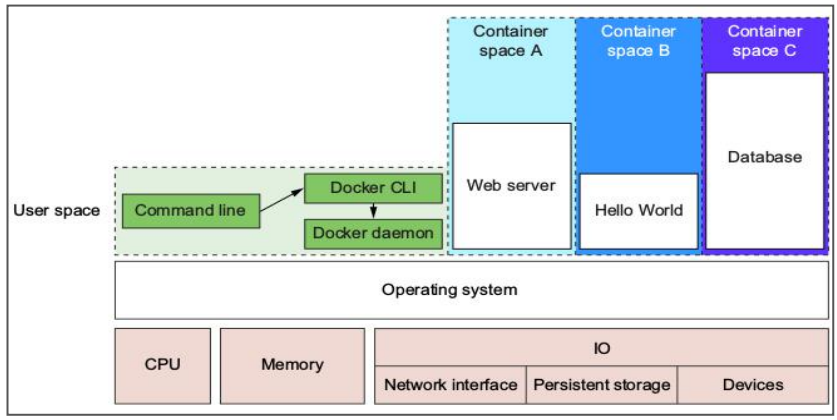
Arsitektur Docker

Arsitektur docker menggunakan client dan server. Docker client mengirimkan request ke docker daemon untuk untuk membangun, mendistribusikan, dan menjalankan container docker. Keduanya (docker client dan daemon) dapat berjalan pada sistem yang sama. Antara docker client dan docker daemon berkomunikasi via socket menggunakan *RESTfull API*.



Gambar 1.2

Untuk lebih jelasnya mengenai arsitektur docker, dibawah ini adalah ilustrasi docker yang menjalankan tiga buah container.



Gambar 1.3

Komponen Docker

Docker Images

Docker images merupakan dasar template untuk docker container, sebuah image biasanya berisi OS maupun aplikasi yang telah diinstall dan telah jadi. Image ini digunakan untuk menjalankan container, di docker hub terdapat banyak image yang bisa dipilih dan digunakan sebagai base image. Untuk melihat images yang tersedia pada PC anda, lakukan perintah `docker images`, seperti yang terlihat pada gambar dibawah ini.

Gambar 1.4

Docker Images diidentifikasi oleh image ID yang panjangnya terdiri dari 64 karakter Hexadesimal, tetapi biasanya ketika menjalankan images hampir tidak menggunakan ID, melainkan menggunakan nama dari images itu sendiri, dikarenakan lebih mudah menghafal nama dari pada image ID.

Apabila diperhatikan pada gambar diatas yang menunjukkan list dari image docker yang tersedia, anda melihat sebuah kolom dengan label TAG. TAG ini merupakan mekanisme pendistribusian dari sebuah image, dimana satu image dapat memiliki versi yang berbeda dengan identifikasi dari Tag. Anda dapat melakukan pull terhadap image docker yang berbeda dengan memberikan tag yang spesifik, misalnya seperti contoh berikut :

```
docker pull ubuntu:14.04
docker pull ubuntu:16.04
```

Docker Container

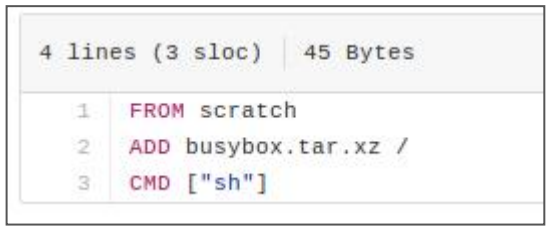
Docker container sendiri merupakan sebuah image yang dapat dikemas dan dibaca tulis, container berjalan diatas image. Pada setiap perubahan yang disimpan pada container akan menyebabkan terbentuknya layer baru di atas base image. Kita dapat melakukan instalasi aplikasi didalamnya dan melakukan penyimpanan terhadap docker container tersebut.

Docker Registry

Docker registry merupakan repositori distribusi kumpulan docker image yang terpusat, baik bersifat public mauoun private. Registry public Docker disebut dengan *Docker Hub*. Disini kita dapat push image yang kita buat sendiri maupun pull image dari pengguna lain yang tersedia. Halaman docker hub dapat anda akses pada alamat url https://hub.docker.com/explore/

Dockerfile

Dockerfile merupakan script yang berisi atau terdiri dari serangkaian perintah (intruksi/code) yang akan dieksekusi secara otomatisasi dan berurutan untuk membangun sebuah image. Contoh docker file dapat anda lihat pada gambar dibawah.



Gambar 1.5

Coba tebak, ini dockerfile apa ?, yups, anda benar jika menebak docker file untuk busybox, karena untuk contoh saja, jadi saya ambil dari dockerfile yang sederhana.

Instalasi Docker

Docker dapat di berjalan diberbagai sistem operasi seperti Windows, Mac, Linux, Amazon Web Service, dan juga Microsoft Azure. Namun kali ini penulis hanya contohkan instalasi di Windows, Mac, dan Linux.

Untuk dapat menggunakan docker, download paket docker pada official website docker dengan alamat url : *"https://www.docker.com/products/docker"*

dan pilih sesuai sistem operasi yang anda gunakan.

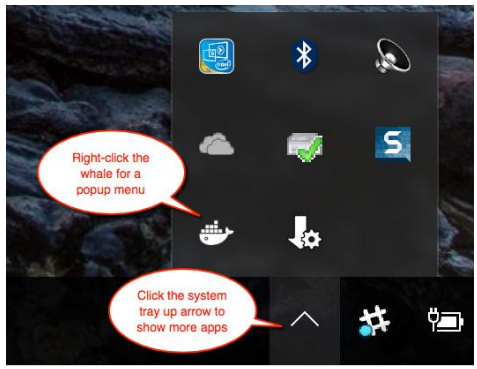
A. **Instalasi Docker di Windows**

Untuk instalasi docker pada windows, anda harus memenuhi minimum requirement diantaranya adalah 64 bit Windows 10 Pro, Enterprise, Education (1511 November update, Build 10586 atau lebih baru) atau Microsoft Hyper-V.

Setelah memenuhi requirement tersebut, lanjutkan dengan mengeksekusi file *InstallDocker.msi* yang telah di download sebelumnya dan lanjutkan proses instalasi hingga selesai.

Gambar 1.6

Setelah selesai diinstalasi, docker akan berjalan secara otomatis. Indikator bergambar ikan paus pada tray anda menunjukkan bahwa docker berjalan dan dapat diakses melalui terminal.



Gambar 1.7

Untuk mengecek versi docker yang terinstalasi pada mesin anda, lakukan perintah berikut ini melalui shell favoritmu, misalnya cmd.exe atau PowerShell. Lakukan perintah berikut :

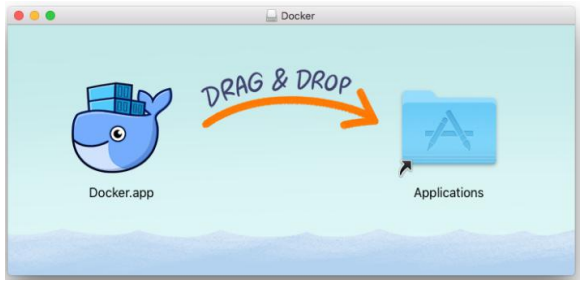
```
C:\Users\docker> docker --version
Docker version 1.12.0, build 8eab29e, experimental

C:\Users\docker> docker-compose --version
docker-compose version 1.8.0, build d988a55

C:\Users\docker> docker-machine --version
docker-machine version 0.8.0, build b85aac1
```

Jika perintah tersebut berhasil dijalankan, berarti docker siap digunakan pada mesin windows anda.

B. **Instalasi Docker di Mac**

Sama halnya dengan instalasi pada Windows, sebelum anda menginstalasi docker pada Mac, system requirement yang harus dipenuhi adalah OS X 10.10.3 Yosemite atau model terbaru, dengan Intel's hardware support memory management unit (MMU) dan virtualization, seperti Extended Page Tables (EPT). Minimal memiliki 4GB of RAM, dan Virtualbox versi 5 keatas.

Setelah memenuhi requirement tersebut, lanjutkan dengan mengeksekusi file *Docker.dmg* yang telah di download sebelumnya, kemudian jendela instalasi akan terbuka, lanjutkan dengan men-*drag* logo docker ke folder Application.



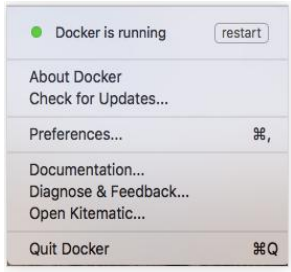
Gambar 1.8

Anda akan diminta izin akses untuk menginstalasi docker network dan linking ke docker apps, masukkan password login anda, sehingga docker selesai menjalankan proses instalasi.

Double klik pada docker.app untuk menjalankan docker, sehingga indikator docker pada notification bar terlihat.

Gambar 1.9

Pada notification bar, anda dapat melakukan beberapa pengaturan yang disediakan, cek versi docker, cek update aplikasi terbaru, melihat dokumentasi docker, dll.



Gambar 1.10

Bagaimana jika kita lakukan beberapa perintah untuk mengetes versi docker seperti pada instalasi di mesin windows sebelumnya. Coba buka command-line terminal anda, kemudian lakukan beberapa perintah berikut ini :

```
$ docker --version
Docker version 1.12.0, build 8eab29e

$ docker-compose --version
docker-compose version 1.8.0, build f3628c7

$ docker-machine --version
docker-machine version 0.8.0, build b85aac1
```

Jika perintah tersebut berhasil dijalankan, maka instalasi docker pada mesin Mac anda sukses.

C. **Instalasi Docker di linux**

Distro yang paling mudah dalam meng-instalasi docker adalah menggunakan Ubuntu, mulai dari Ubuntu Trusty 14.04 LTS paket docker sudah ada pada repositori resminya. Namun tenang saja, menggunakan distro linux lain tidak lebih sulit dari pada menginstalasi paket docker di ubuntu. Docker menyediakan script instalasi untuk berbagai distro linux yang bisa anda gunakan.

Untuk pengguna Ubuntu anda dapat menginstalasi paket docker dengan menggunakan apt-get melalui terminal, dan nama paket

Docker itu sendiri adalah *docker.io*

Lakukan perintah berikut ini pada terminal :

```
$ sudo apt-get update
$ sudo apt-get install docker.io
$ source /etc/bash_completion.d/docker.io
```

Disini pertama kali kita akan meng-update repository ubuntu untuk mendapatkan informasi versi paket-paket aplikasi terbaru yang tersedia pada daftar *sources.list* yang berlokasi di */etc/apt/*. Kemudian perintah selanjutnya adalah menginstalasi paket docker, dan menambahkan perintah autocomplete docker pada baris terakhir.

Apabila proses instalasi sudah selesai dan tidak ada error, anda dapat mencoba perintah `sudo docker version` untuk memastikan bahwa paket docker sudah berjalan. Perintah tersebut akan menampilkan versi dari paket docker yang terpasang pada mesin linux anda.

```
m1lk@docker:~$ sudo docker version
Client version: 1.6.2
Client API version: 1.18
Go version (client): go1.2.1
Git commit (client): 7c8fca2
OS/Arch (client): linux/amd64
Server version: 1.6.2
Server API version: 1.18
Go version (server): go1.2.1
Git commit (server): 7c8fca2
OS/Arch (server): linux/amd64
```

Seperti disinggung pada awal pembahasan, apabila anda menggunakan distro linux selain ubuntu, anda dapat menggunakan script instalasi yang terdapat pada alamat "*https://get.docker.com/*". Script tersebut akan menambahkan official docker repositori pada mesin linux anda, sehingga dipastikan paket docker yang terinstalasi merupakan paket yang docker terbaru. Untuk menggunakan script tersebut, lakukan perintah berikut pada terminal :

```
curl -sSL https://get.docker.com/ | sudo sh
```

Apabila anda menggunakan instalasi dengan script diatas, paket yang anda instalasikan bukan lagi *docker.io* melainkan *docker-engine*. Tidak ada yang berbeda dalam menggunakan perintah-perintah docker, baik ketika anda melakukan instalasi melalui apt-get atau menggunakan script.

Anda bebas memilih mana paket docker yang ingin dipasang, namun disini penulis lebih merekomendasikan untuk menggunakan script instalasi, mengapa ?, karena selain mudah, menggunakan script instalasi juga memberikan paket docker yang *up to date*. Dibawah ini adalah versi paket docker yang penulis lakukan dengan menggunakan script instalasi tersebut.

```
m1lk@docker:~$ sudo docker version
Client:
 Version:      1.12.1
 API version:  1.24
 Go version:   go1.6.3
 Git commit:   23cf638
 Built:        Thu Aug 18 05:33:38 2016
 OS/Arch:      linux/amd64

Server:
 Version:      1.12.1
 API version:  1.24
 Go version:   go1.6.3
 Git commit:   23cf638
 Built:        Thu Aug 18 05:33:38 2016
 OS/Arch:      linux/amd64
```

Konfigurasi Tambahan.
Tidak nyaman rasanya apabila menjalankan docker harus mengawali dengan perintah sudo. Agar hal ini tidak terjadi, pastikan user yang anda gunakan menjadi anggota dari group docker. Hal ini dapat dilakukan dengan melakukan perintah berikut :

```
sudo adduser user-anda docker
```

Atau,

```
sudo usermod -aG docker user-anda
```

BAB II
Docker Basic

Pada bab ini kita akan membahas dasar-dasar perintah docker, docker image, docker container, dan dockerfile, juga perintah lainnya yang berguna untuk memantau aplikasi yang dibuat dengan menggunakan docker.

Perintah “Docker Help”

Kita akan mengoperasikan docker dengan docker command-line, oleh karena itu saya akan membahas perintah yang pertama, yaitu `docker help`. Dimana kebiasaan kebanyakan orang Indonesia adalah coba dulu, kalo terjadi error baru liat manualnya :p, oleh karena itu disini kita coba untuk lihat dulu manualnya baru selanjutnya mulai operasikan perintah docker lainnya.

Coba ketikkan : `docker help`

Gambar 2.1

Atau jika anda ingin mengetahui lebih spesifik fungsi dari salah satu perintah pada docker, ketikkan saja `docker [perintah] --help`, misalnya untuk mengetaui perintah `cp`, maka ketikkan `docker cp --help` dan hasilnya dapat anda lihat pada gambar dibawah.

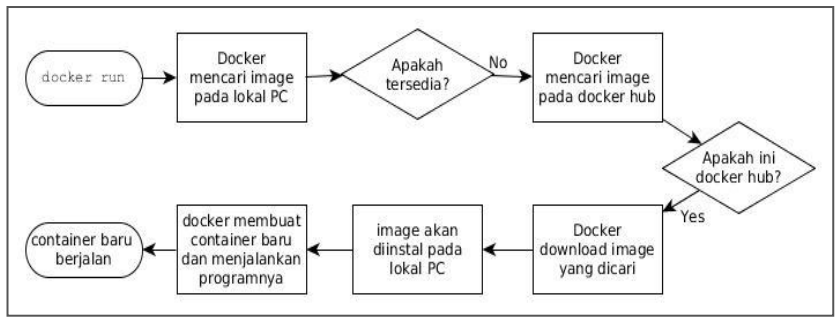
Gambar 2.2

Seperti yang anda lihat, kita bisa melihat lebih spesifik tentang perintah yang digunakan. Jadi jika anda menemui kesulitan dalam menggunakan perintah pada docker, jangan segan-segan untuk menggunakan perintah `docker help` ini.

Say "*Hello World*" to Docker

Untuk anda yang baru memulai belajar bahasa pemrograman mungkin tidak asing dengan kata "Hello World", begitu juga disini kita akan memulai dengan menggunakan image *hello-word* yang disediakan oleh Official Docker, kita akan mulai dengan mengetikan perintah `docker run hello-world`

Apa yang terjadi ketika anda melakukan perintah tersebut ?. yang terjadi adalah, docker akan mengecek apakah image yang dibutuhkan (dalam hal ini adalah image hello-word) sudah tersedia di lokal PC, jika belum makan docker akan melakukan pull (download) terhadap image hello-word yang berada pada docker registry, kemudian membuat container dari image tersebut dan menjalankan image hello-world.



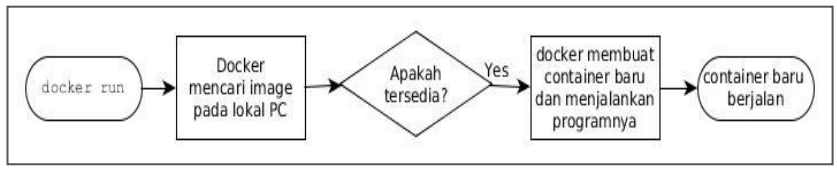
Gambar 2.3

Gambar 2.4

Setelah menjalankan perintah `docker run hello-world`, seharusnya anda mendapatkan hasil seperti pada gambar 2.4 diatas, setelah itu container akan berhenti. Untuk memahami *running state* dari sebuah container adalah dengan membuat satu program terkait didalam container tersebut, jika program berjalan, maka container berjalan, dan apabila program berhenti maka container juga berhenti.

Pertanyaan selanjutnya adalah, apakah setiap kali menjalankan perintah `docker run`, akan selalu mengambil (pull / download) image dari dcoker hub ?. Oh tentu saja tidak, docker akan melakukan pull pada image yang belum tersedia saja, jika image yang dibutuhkan sudah tersedia pada lokal PC anda, docker akan langsung membuat container baru dan menjalankannya saja. Wait..wait.. Bener tuh bikin container baru lagi, bukan pakai container yang sudah ada ?, yups bener, docker ini mengadopsi versioning, jadi setiap anda menjalankan perintah `docker run`, setiap itu pula docker akan membuat container baru walaupun itu dari image yang sama.

Nah, sedangkan untuk ilustrasi proses `docker run` pada image yang sudah tersedia, dapat dilihat pada gambar dibawah ini.



Gambar 2.5

Introduction : Docker Container

Pada sub materi ini akan sedikit mengenalkan bagaimana membuat docker container yang dijalankan dapat lebih memberikan informasi tentang penggunaan docker kepada anda.

Perintah pertama adalah mencari docker images di docker registry. Seperti yang dibahas sebelumnya, docker menyimpan banyak image pada docker registry yang beralamat di *https://hub.docker.com*. Untuk melakukannya, sebenarnya anda tidak perlu membuka web browser, namun bisa dilakukan dengan perintah `docker search [docker_images]`, saya akan mencontohkan untuk mencari docker images yang bernama *redis*, maka saya akan mengetikkan `docker search redis`. Pencarian tersebut juga dapat dipersempit dengan menggunakan opsi filter, seperti misalnya `docker search redis -f stars=10`, dimana akan mencari image redis dengan minimal 10

stars. Opsi lainnya anda bisa lihat dengan menggunakan `docker search --help`.

Gambar 2.6

Gambar 2.7

Dua gambar diatas merupakan hasil dari pencarian image docker yang bernama redis. Saya rekomendasikan untuk memilih OFFICIAL image, yang mana sudah terverifikasi oleh Docker. Ataupun jika memilih image dari komunitas, maka pilihlah dengan rating terbaik yang biasanya ditandai dengan banyaknya STARS.

Setelah mencari image yang diinginkan, lakukan perintah `docker run` untuk menjalankannya, docker run merupakan perintah untuk menjalankan sebuah docker container, coba ketikkan perintah `docker run redis` dan apabila anda sukses, maka hasilnya akan tampak seperti gambar dibawah.

Gambar 2.8

Sebagaimana yang terlihat, setiap container akan dijalankan pada foreground process yang menutupi seluruh terminal PC host anda, lalu bagaimana misalnya jika ingin menjalankan dua container bersamaan..?, untuk itu kita gunakan opsi `-d` atau `--detach` yang membuat sebuah container dapat berjalan pada background process. Sebagai contoh :

```
$ docker run -d redis
```

Gambar 2.9

Dapat dilihat 2.9, bahwa sekarang container redis berjalan pada background proses dengan Container ID `e9df83f05078xxxxxx`.

Sedangkan untuk melihat container yang sedang berjalan, anda dapat menggunakan perintah `docker ps`, dan begitu juga hasilnya dapat terlihat pada gambar 2.9.

Selanjutnya, semua docker container yang dijalankan akan diberikan sebuah ID dan Nama oleh docker daemon secara otomatis, sebagai identifikasi yang digunakan untuk melakukan perintah-perintah docker lainnya, jadi anda harus menghafal ID atau Nama container yang bersangkutan apabila ingin bekerja dengannya.

Untuk memudahkan hal tersebut, docker sudah menyediakan opsi `--name` yang dapat anda gunakan untuk memberikan nama secara manual terhadap container yang dibuat agar lebih mudah untuk diingat. Sebagai contoh saya akan menjalankan container baru dari images redis dengan memberikan nama *my-db* menggunakan opsi `--name` tersebut.

```
$ docker run -d --name my-db redis
```

Lalu dicek apakah nama yang diberikan sudah sesuai, coba ketikkan perintah `docker ps`, dan perhatikan hasilnya.

Gambar 2.10

Catatan : Memberikan nama container harus unik, artinya nama dari setiap container tidak boleh ada yang sama, apabila terdapat nama yang sama akan terjadi pesan error seperti berikut :

```
docker: Error response from daemon: Conflict. The name
"/my-db" is already in use by container
60f2f5a7a6a7714da1055861b3913e6beac78c8f885f944b545
32693c480fb57. You have to remove (or rename) that
container to be able to reuse that name..
See 'docker run --help'.
```

Solusinya, anda harus menghapus atau rename dahulu container tersebut atau dengan menggunakan nama yang lain. Untuk menghapus docker container anda dapat menggunakan perintah berikut :

```
$ docker rm [nama_container]
```
contoh,
```
$ docker rm my-db
```

Kemudian, seperti yang dijelaskan sebelumnya, setiap docker container terisolasi antara satu dengan lainnya. Jika terdapat suatu service yang ingin diakses dari suatu container, anda harus meng-expose port service dari container tersebut kemudian di mapping ke port host pc kita, barulah service tersebut dapat diakses melalui port host bukan pada port docker container.

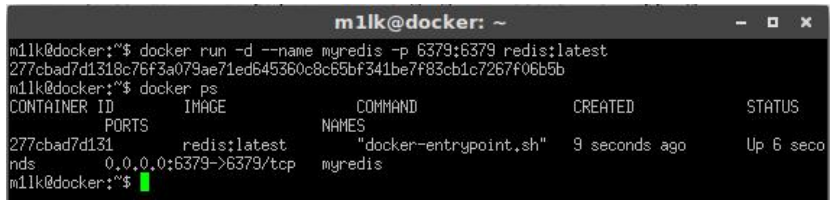
Apabila anda perhatikan pada gambar 2.10, pada kolom PORTS yang isinya adalah 6379/tcp, yang artinya bahwa aplikasi redis yang dijalankan pada container tersebut berjalan pada protocol tcp dan port 6379.

Oke, kemudian kita akan mencoba menjalankan container tersebut sekali lagi, tapi kali ini untuk dapat mengakses service pada image redis, maka gunakan opsi `-p <host port>:<container port>` (publish container port to the host port). Sebagaimana yang diketahui, container redis meng-expose service di port 6379. Jadi, jika kita ingin me-mapping port 6379 langsung kepada host pc, maka gunakan opsi `-p 6379:6379`.

Secara default, docker me-mapping port pada alamat ip 0.0.0.0, yang artinya semua alamat ip. jika anda menginginkan docker me-mapping pada alamat ip tertentu, anda bisa mendefinisikan alamat ip nya, sebagai contoh `-p 192.168.1.1:6379:6379`.

Baiklah, sekarang mari kita coba jalankan redis container baru yang berjalan pada backgroud proses dengan nama *myredis* dan bind port 6379 pada container ke port 6379 pada host pc kita. Maka perintah yang dilakukan adalah :

```
$ docker run -d --name myredis -p 6379:6379 redis:latest
$ docker ps
```



Gambar 2.11

Jika pada perintah sebelumnya, kita telah mendefinisikan port yang dimapping pada docker container ke port pada host pc kita, opsi -p juga dapat membuat mapping *myredis* container port secara random, yaitu dengan opsi `-p 6379` saja. Hal ini dimaksudkan apabila anda menjalankan beberapa sevice yang sama pada container yang berbeda tanpa harus mengganti konfigurasi aplikasinya.

Note : untuk mencari mapping port dinamis yang diberikan oleh docker pada sebuah container, anda dapat menggunakan perintah `docker port [nama_container]`, sebagai contoh anda dapat melihat ilustrasi seperti berikut ini :

```
m1lk@docker:~$ docker run -d --name myredis \
-p 6379 redis:latest
m1lk@docker:~$ docker port myredis
6379/tcp -> 0.0.0.0:32768
```

Terlihat bahwa saat ini port 6379 pada container di mapping ke port 32768 pada pc host.

Anothers Docker Command Line

Pada materi kali ini, saya akan menerangkan perintah-perintah docker command line lainnya, walaupun ada beberapa perintah yang pernah dilakukan pada pembahasan sebelumnya.

Perintah docker version

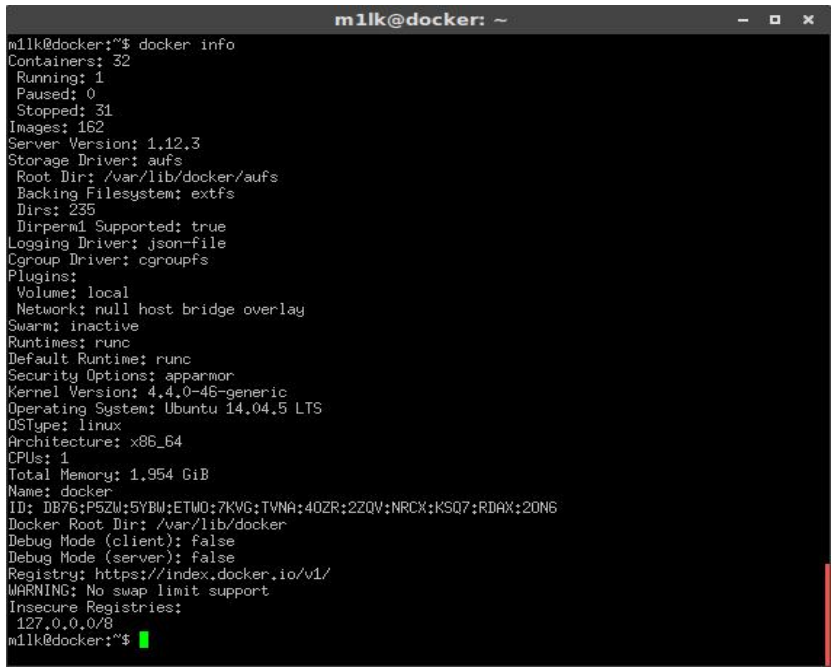
Perintah docker version akan memberikan informasi tentang versi docker yang sedang digunakan.

```
$ docker -v
Docker version 1.12.3, build 6b644ec
```

Perintah docker info

Perintah ini akan memberikan detail konfigurasi dari docker daemon, seperti banyaknya container, image, volume, dll.

```
$ docker info
```



Gambar 2.12

Perintah docker pull

Perintah `docker pull` digunakan untuk men-download / pull image dari docker registry. Secara default, pull image akan mengambil dari docker registry, tapi tidak menutup kemungkinan apabila anda memiliki registry yang dibuat sendiri. Perintah docker pull dapat dilakukan dengan cara sebagai berikut :

```
$ docker pull [registry]/[image_atau_repository]
$ docker pull redis  # pull image redis dari docker hub
$ docker pull redis:3.2  #pull image redis dgn tag 3.2
```

Perintah docker start

Sebagaimana yang telah dibahas tentang *runnig state* dari docker, ketika anda melakukan perintah `docker run`, container docker akan dibekukan setelah berhenti, kecuali secara explisit container tersebut dihapus. Nah, perintah `docker start` ini adalah perintah untuk menjalankan kembali container docker tersebut.

```
$ docker start [opsi] [container]
```

Berikut ini contoh penggunaan perintah docker start, namun sebelumnya kita lihat container yang pernah dibuat dengan perintah docker ps :

```
$ docker ps -a
```

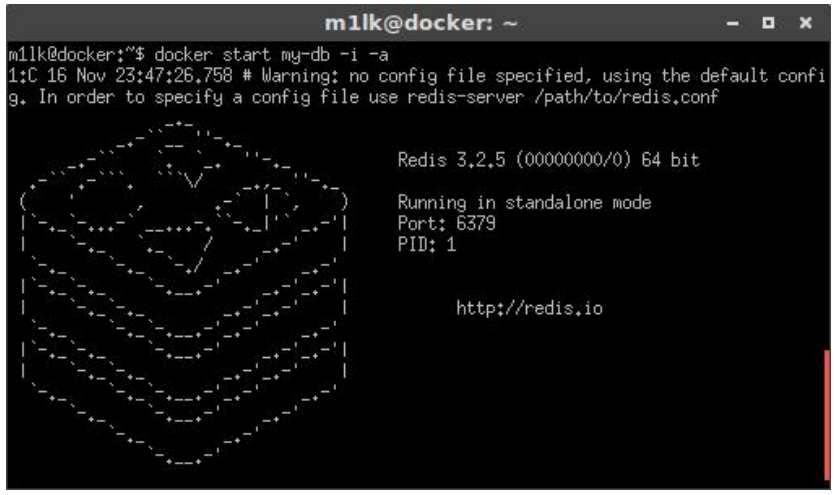
Gambar 2.13

Selajutnya, coba jalankan kembali salah satu container tersebut, misalnya container dengan nama *my-db* :

```
$ docker start my-db
my-db
```

Contoh lainnya, kita gunakan opsi -i dan -a :

```
$docker start my-db -i -a
```



Gambar 2.14

Perintah docker stop

Perintah `docker stop` berfungsi untuk memberhentikan sebuah docker container dengan cara mengirimkan perintah `SIGTERM` dan `SIGKILL`.

```
$ docker stop [container]
```

Note : `SIGTERM` dan `SIGKILL` merupakan suatu signal perintah interprocess communication dari sitem operasi keluarga Unix / POSIX-Complaint. `SIGTERM` digunakan untuk memberhentikan process (*terminate*), sedangkan `SIGKILL` digunakan untuk memaksa mematikan suatu process (*force kill*).

Perintah docker restart

Perintah `docker restart` digunakan untuk me-*restart* container yang sedang berjalan.

```
$ docker restart [container]
```

Perintah docker rm

Perintah ini digunakan untuk menghapus docker container.

```
$ docker rm [container]
```

Sebagai contoh, berikut ini hasil dari perintah `docker ps -a` :

Gambar 2.15

Kemudian kita coba hapus salah satu container dengan nama *my-db* :

```
$ docker rm my-db
my-db
```

Selain contoh diatas, kita juga bisa menggabungkan dua perintah secara bersamaan, misalnya :

```
$ docker rm $(docker ps -aq)
226b05bb2d0c
303dff1934a0
a47884b1e265
```

Note: Perintah tersebut akan menampilkan semua ID docker container yang dijalankan oleh perintah `docker ps -aq`, kemudian menjalankan perintah `docker rm`, dan hasilnya akan menghapus semua container yang ada dengan sekali perintah.

Perintah docker ps

Sebagaimana kita ketahui, sejauh ini perintah `docker ps` merupakan perintah yang sering digunakan. Perintah `docker ps` itu sendiri adalah perintah untuk menampilkan list container yang ada, perintah `docker ps` dapat digunakan dengan berbagai opsi berikut :

```
$ docker ps [opsi]
```

Opsi	Keterangan
`-a, --all`	Menampilkan seluruh container, termasuk yang berhenti.
`-q, --quiet`	Hanya menampilkan prameter ID container.
`-s, --size`	Menampilkan ukuran container.
`-l, --latest`	Hanya menampilkan container terakhir, termasuk yang berhenti.
`-n=""`	Menampilkan sejumlah "n" container terakhir, termasuk yang berhenti.
`--before=""`	Menampilkan container yang dibuat sebelum ID atau Nama container yang ditentukan, termasuk yang berhenti.
`--after=""`	Menampilkan container yang dibuat sesudah ID atau Nama container yang ditentukan, termasuk yang berhenti.

Apabila anda menjalankan perintah `docker ps` saja tanpa opsi apapun, maka docker hanya akan menampilkan docker container yang sedang berjalan, tapi apabila anda menambahkan opsi "`-a`" misalnya, maka anda akan mendapatkan semua container yang pernah dibuat.

Perintah docker logs

Perintah ini akan menampilkan *log* dari sebuah container yang berjalan.

```
$ docker logs [container]
```

Contoh :

```
$ docker logs my-redis
```

Perintah docker inspect

Perintah ini memungkinkan anda untuk melihat detail dari sebuah docker image maupun container dengan format JSON array.

```
$ docker inspect [image] / [container]
```

Contoh dengan image *alpine* :

```
$ docker inspect alpine
[
    {
        "Id":
"sha256:8ea015e36e695788f194179c7710e9e8ba9f7e4fa53
f2df54edcbf5661482549",
        "RepoTags": [
            "alpine:latest"
        ],
        "RepoDigests": [],
        "Parent": "",
....
..
```

Contoh dengan container ID *39f...* :

```
$ docker inspect 39f
[
    {
        "Id":
"39f2947d7971681ab5c59d84f2187174150d93e5824de84b26
dade0e34b64c1a",
```

```
        "Created": "2016-11-19T20:47:47.503114862Z",
        "Path": "docker-entrypoint.sh",
        "Args": [
            "redis-server"
        ],
        "State": {
            "Status": "running",
            "Running": true,
            "Paused": false,
....
..
```

Perintah docker top

Untuk para pengguna sistem operasi GNU/Linux, mungkin tidak asing lagi dengan perintah ini, yups, perintah docker top ini adalah perintah untuk melihat proses yang berjalan pada sebuah docker container.

```
$ docker top [container]
```

Perintah docker attach

Perintah `docker attach` ini berfungsi mengambil alih kendali dari docker container yang berjalan secara background, ketika anda menjalankan sebuah container dengan perintah `docker run` dan opsi `-d` (`--detach`), dimana diketahui bahwa perintah tersebut akan menjalankan container dengan mode background, nah untuk mengambil alih kontrol pada setiap container yang berjalan dengan mode tersebut, anda dapat menggunakan perintah `docker attach` ini. Perintah docker attach dapat digunakan dengan ID atau Nama container.

```
$ docker attach [container]
```

Perintah docker kill

Perintah `docker kill` ini akan memberhentikan sebuah container yang berjalan, juga memberi signal `SIGTERM` kepada proses yang berjalan di dalam container tersebut.

```
$ docker kill [container]
```

Perintah docker cp

Perintah docker cp berfungsi meng-copy file atau folder dari container ke pc host anda maupun sebaliknya.

```
$ docker cp [opsi] container:src_path host_dest_path|-
$ docker cp [opsi] host_src_path|- container:dst_path

Opsi :
-L, --follow-link : ikuti symbolic link dari sumber
```

Note : Tanda "-" sebagai source adalah untuk membaca tar achive dari `stdin` dan mengekstrak kedalam direktori tujuan pada container. Sedangkan penggunaan tanda "-" ke direktori tujuan, berfungsi untuk mengirimkan output dari hasil ekstrak tar archive ke `stdout`.

Contoh meng-copy file dari container ke host :

- Jalankan container baru dengan opsi interaktif, misal docker image m1lk/hello-docker
  ```
  $ docker run -it m1lk/hello-docker /bin/bash
  083d0062b7b9
  ```
- Buat file hai.txt di folder /home
  ```
  $ touch /home/hai.txt
  ```
- Isi file tersebut dengan kata "Halo Docker..!"
  ```
  $ echo "Halo Docker..!" > /home/hai.txt
  ```
- Keluar dari container tersebut, kemudian jalankan kembali
  ```
  root@083d0062b7b9:/home# exit
  m1lk@docker:~/hello-docker$ docker start 083d
  ```

- Mulai proses meng-copy file hai.txt ke host folder /
```
$ docker cp \
083d:/home/hai.txt /home/m1lk/hello-docker
```

  atau bisa juga dengan cara berikut
```
$ docker cp 083d:/home/hai.txt $PWD/hello-docker
```

- Lihat hasilnya pada host directori anda :
```
m1lk@docker:~/hello-docker$ ls
hai.txt
m1lk@docker:~/hello-docker$ cat hai.txt
Halo Docker..!
```

Contoh meng-copy file dari host ke container :

- Proses meng-copy file index.html ke /var/www
```
m1lk@dom1lk@docker:~/hello-docker$  docker  cp  \
index.html 083d:/var/www
```

Dockerfile Instructions

Setelah mencoba percobaan-percobaan kecil bagaimana berinteraksi dengan docker container, mungkin terfikir oleh anda untuk mecoba membuat suatu image baru yang disesuaikan kebutuhan anda sendiri, untuk itu anda dapat menggunakan sebuah dockerfile.

Sebagaimana sebelumnya telah dibahas tentang apa itu dockerfile. Dockerfile biasa digunakan untuk membentuk image baru dengan serangkaian baris kode, untuk melakukan perubahan yang disesuaikan terhadap suatu image, kemudian di *rebuild* menjadi image yang baru.

Oleh karena itu, sebelum bekerja dengan dockerfile, ada baiknya anda mengetahui perintah-perintah yang digunakannya. Sebuah dockerfile mempunyai sintaks seperti berikut :

```
# Comment
INSTRUCTION argument
```

Setiap baris yang diawali dengan tanda "#" merupakan sebuah komentar, jika tanda "#" ini ada dibagian lain, baris tersebut dianggap sebagai bagian dari komentar. Kode Instruksi dari dockerfile tidak case sensitif, meskipun biasa menggunakan huruf besar, itu hanya digunakan untuk membedakan dari kode argument.

Instruksi FROM

Instruksi `FROM` adalah untuk mendefinisikan docker image yang akan digunakan pada instruksi-instruksi berikutnya, biasanya dockerfile yang tanpa memberikan komentar, instruksi `FROM` ini akan berada pada perintah awal. Berikut contoh penggunaanya :

```
FROM <image>:<tag>
```

Contoh :

```
FROM ubuntu
```

Atau,

```
FROM ubuntu:14.04
```

Image yang digunakan bisa didapat dari lokal atau public image, jika tidak terdapat pada lokal pc anda, maka docker akan melakukan pull terhadap image di docker registry. Sedangkan *tag* merupakan opsional, tanpa tag yang spesifik, image dengan tag *latest* akan menjadi pilihan default, kemudian jika terdapat tag yang benar, maka image dengan tag tersebutlah yang akan di pull dari docker registry, sedagkan apabila tag tidak sesuai, docker akan memberikan pesan kesalahan ketika melakukan perintah docker build.

Instruksi MAINTAINER

Instruksi ini mengijinkan anda membuat keterangan identitas sipembuat image tersebut.

```
MAINTAINER <nama>
```

Contoh :

```
MAINTAINER m1lk <m1lk@email.com>
```

Instruksi RUN

Instruksi `RUN` ini akan mengeksekusi suatu argument diatas layer image yang digunakan dan meng-commit kedalam image baru, image baru ini yang akan digunakan pada instruksi selanjutnya.

Instruksi `RUN` mempunyai dua bentuk :

```
RUN <argument>
RUN ["executable", "arg1", "arg2" ...]
```

Pada bentuk pertama, sebuah argument akan dijalankan pada sebuah *shell* (/bin/sh -c). Sedangkan bentuk kedua berguna apabila sebuah instance tidak memiliki *shell*.

Contoh :

```
RUN apt-get update && apt-get install -y apache2
RUN mkdir -p /var/www/html
```

Instruksi CMD

Instruksi `CMD` ini memungkinkan anda menset perintah default, yang akan dijalankan ketika anda menjalankan sebuah container tanpa perintah yang spesifik, jika anda menjalankan suatu perintah yang spesifik pada container tersebut, argument `CMD` ini akan diabaikan.

Ada tiga bentuk Instruksi `CMD` :

```
CMD ["executable", "arg1", "arg2"...]
CMD ["arg1", "arg2"...]
CMD command arg1 arg2
```

Pada bentuk instruksi pertama dan ketiga, dijalankan pada bagian *shell* dan *exec*. Sedangkan bentuk yang kedua digunakan bersama dengan Instruksi `ENTRYPOINT` dalam bentuk *exec*, bentuk ini akan menset default parameter yang akan ditambahkan kedalam parameter `ENTRYPOINT` jika container dijalankan tanpa perintah yang spesifik.

Contoh Instruksi `CMD` dari potongan dockerfile :

```
CMD echo "Hello Docker"
```

Jika container tersebut dijalankan dengan `docker run -it [image]`, hasilnya :

```
Hello Docker
```

Namun jika container tersebut dijalankan dengan tambahan instruksi seperti `docker run -it [image] /bin/bash` , Instruksi `CMD` akan diabaikan, sehingga hasilnya hanya akan membuka shell bash, contohnya seperti berikut :

```
root@b6f258caef04:/#
```

Instruksi ENTRYPOINT

Instruksi `ENTRYPOINT` ini memungkinkan anda membuat container bertindak sebagai *executable*. Mirip dengan Instruksi `CMD`, hanya saja perbedaan pada Instruksi `ENTRYPOINT` ini perintah dan parameternya tidak diabaikan ketika docker container berjalan dengan tambahan perintah spesifik.

Instruksi `ENTRYPOINT` memiliki dua bentuk :

```
ENTRYPOINT ["executable", "arg1", "arg2"]
ENTRYPOINT command arg1 arg2
```

Pada bentuk pertama merupakan bentuk *exec form* dari `ENTRYPOINT`, yang memungkinkan anda menset sebuah perintah dan parameter yang kemudian dapat digunakan oleh Instruksi `CMD` untuk menambahkan sebuah parameter tambahan, sehingga hasil eksekusi dari Instruksi `ENTRYPOINT` dapat dirubah. Instruksi dari `ENTRYPOINT` akan selalu dijalankan, sedangkan Instruksi `CMD` dapat di gantikan (*overwrite*) dengan tambahan argument ketika anda menjalankan sebuah container, untuk lebih jelasnya mari perhatikan contoh berikut ini :

```
ENTRYPOINT ["/bin/echo", "Hello"]
CMD ["World"]
```

Ketika perintah tersebut dijalankan dengan perintah docker `run -it [image]`, akan menghasilkan output :

```
"Hello World"
```

Sedangkan ketika anda menjalankan perintah docker `run -it [image] Docker` , akan menghasilkan output :

```
"Hello Docker"
```

Terlihat pada contoh yang kedua, bahwa Instruksi CMD telah digantikan dengan parameter tambahan ketika anda menjalankan perintah `docker run`.

Kemudian, untuk bentuk Instruksi `ENTRYPOINT` yang kedua merupakan bentuk *shell form*, yang mana akan mengabaikan (*ignore*) semua perintah dari `CMD` maupun tambahan perintah ketika anda menjalankan `docker run`.

Instruksi WORKDIR

Instruksi `WORKDIR` akan memberikan *working directory* untuk Instruksi seperti `RUN`, `CMD`, dan `ENTRYPOINT` pada dockerfile, Instruksi `WORKDIR` mempunyai format sebagai berikut :

```
WORKDIR /lokasi/working/directory
```

Instruiksi ini juga dapat diberikan lebih dari satu kali pada dockerfile yang sama. Jika diberikan *relative path*, maka Instruksi `WORKDIR` akan bersifat relatif pada Instruksi `WORKDIR` sebelumnya.

Instruksi EXPOSE

`EXPOSE` merupakan sebuah instruksi untuk memberitahukan docker daemon untuk mempublish port tertentu ketika sebuah container dijalankan.

```
EXPOSE port1 port2 ...
```

Meskipun setelah menggunakan Instruksi `EXPOSE` ini pada dockerfile, anda tetap harus menggunakan opsi "`-p`" ketika menjalankan perintah `docker run` untuk me-mapping port yang akan digunakan, hal ini juga berguna ketika anda melakukan linking antar container.

Instruksi ENV

Instruksi `ENV` digunakan untuk menset environtment variable kedalam sebuah nilai. Instruksi `ENV` memiliki dua bentuk dan ditulis dengan format sebagai berikut :

```
ENV <key> <value>
ENV <key>=<value> ...
```

Pada bentuk pertama, akan menset *single variable* kedalam sebuah *value*. Seluruh *string* setelah spasi pertama akan diperlakukan sebagai *value*, termasuk karakter seperti spasi dan tanda kutip.

Sedangkan pada bentuk kedua, memungkinkan anda menset beberapa

variable sekaligus dalan satu perintah, perhatikan pada bentuk ini terdapat tanda sama dengan "=" yang tidak digunakan pada bentuk pertama. Seperti halnya sebuah command line, tanda *quote* (") dan *backslash* (\) dapat digunakan untuk memasukan *spasi* kedalam *value*.

Instruksi LABEL

Instruksi ini akan menambahkan metadata kedalam sebuah image. Untuk menambahkan spasi kedalam label, anda dapat menggunakan *quote* (") dan *backslash* (\) sebagaimana digunakan dalam command line. Berikut format penggunaan Instruksi `LABEL` :

```
LABEL <key>=<value> <key>=<value> <key>=<value> ...
```

```
LABEL "com.example.vendor"="ACME Incorporated"
LABEL com.example.label-with-value="foo"
LABEL version="1.0"
LABEL description="This text illustrates \
that label-values can span multiple lines."
```

Sebuah `LABEL` bisa terdapat lebih dari satu instruksi, untuk memberikan multiple `LABEL`, docker merekomendasikan untuk menggabungkan beberapa LABEL menjadi satu, jika memungkinkan. Masing-masing Instruksi `LABEL` akan membuat layer baru yang mana menjadikan image yang dibuat tidak efisien. Berikut contoh membuat multi `LABEL` menjadi satu instruksi :

```
LABEL label1="value1" label2="value2" other="value3"
```

Atau dapat juga ditulis seperti berikut :

```
LABEL label1="value1" \
      label2="value2" \
      other="value3"
```

Note : Jika docker menemui `key` dari `LABEL` yang sama, maka nilai pada `LABEL` yang baru akan menggantikan nilai `LABEL` sebelumya. Instruksi `LABEL` dapat dilihat dengan perintah `docker inspect`.

Instruksi USER

Instruksi `USER` digunakan untuk menset sebuah *username* atau *UID* yang digunakan ketika menjalankan sebuah image dan setiap direktif dari perintah RUN.

```
USER namauser
```

Instruksi VOLUME

Instruksi `VOLUME` ini digunakan untuk membuat *mount point* yang di-*mounting* sebagai external volume dari pc host atau container.

```
VOLUME [lokasi]
```

Contoh penggunaan dapat berupa :

```
VOLUME ["/data"]
VOLUME /var/www/html
```

Instruksi ADD

Instruksi `ADD` ini digunakan untuk menambahkan file atau folder kedalam sebuah image. Format penggunaan instruksi `ADD` adalah sebagai berikut :

```
ADD <sumber> <tujuan>
ADD ["<sumber>",... "<tujuan>"]
```

Lokasi sumber bersifat relatif dari file atau folder yang ingin dibuat kedalam image atau dapat juga berupa *remote URL*. Sedangkan, lokasi tujuan merupakan lokasi absolut dimana merupakan tempat sumber file atau folder diletakkan kedalam container.

Instruksi `ADD` memiliki beberapa aturan sebagai berikut :

- Sebuah `<sumber>` harus merupakan lokasi yang pasti, anda tidak dapat menggunakan `../../file`.
- Jika `<sumber>` berasal dari URL dan pada `<tujuan>` tidak terdapat tanda trailing slash "/" yang menandakan sebuah file, instruksi `ADD` akan mengkopi terhadap file dari URL tersebut.

- Jika `<sumber>` berasal dari URL dan pada `<tujuan>` terdapat tanda trailing slash yang menandakan sebuah folder, kemudian isi dari URL tersebut akan diambil dan nama filenya akan digunakan kemudian disimpan kedalam lokasi `<tujuan>`.
- Jika `<sumber>` merupakan sebuah folder, seluruh forder beserta isinya akan dikopikan bersama dengan *filesystem metadata*.
- Jika `<sumber>` berupa tar archive, file tersebut akan di ekstrak kedalam lokasi `<tujuan>`.
  Hasil ekstraksi akan digabungkan dengan :
  - Apapun yang terdapat pada lokasi tujuan.
  - Isi dari file ekstraksi tar tersebut, jika terdapat file yang konflik, harus diselesaikan dulu pada sumbernya, file per file.
- Jika lokasi `<tujuan>` tidak tersedia, maka akan otomatis dibuatkan, juga terhadap setiap folder yang hilang.

Contoh penggunaan Instruksi `ADD` :

`<sumber>` pada Instruksi `ADD` dapat menggunakan wildcards dan matching syntaks pada bahasa pemrograman GOLANG (golang.org), contohnya sebagai berikut :

```
# menambahkan semua file yang berawalan hom
ADD hom* /mydir/
# ? Akan menggantikan single karakter, misal "home.txt"
ADD hom?.txt /mydir/
# menambahkan "lat" ke 'WORKDIR'/relativeDir
ADD lat relativeDir/
# menambahkan "lat" ke /absoluteDir/
ADD lat /absoluteDir/
```

Instruksi COPY

Instruksi ini akan mengkopi terhadap file atau folder dari sumber dan menambahkannya kedalam *filesystem* sebuah image.

```
COPY <sumber> <tujuan>
COPY ["<sumber>",... "<tujuan>"]
```

Instruksi `COPY` ini mirip dengan Instruksi `ADD`, perbedaannya adalah Instruksi `COPY` tidak mengijinkan file diluar konteks, misalnya ketika anda *build* menggunakan `STDIN` (`docker build - < somefile`), perintah `COPY` tidak dapat digunakan. Semua file atau folder yang dikopikan, akan terbuat dengan `UID` dan `GID 0` (*root*).

Instruksi `COPY` juga memiliki beberapa aturan, sebagai berikut :

- Sebuah `<sumber>` harus merupakan lokasi yang pasti, anda tidak dapat menggunakan `../../file`.
- Jika `<sumber>` adalah folder, seluruh isi dari folder tersebut akan dikopikan, termasuk *filesystem metadata,* namun folder itu sendiri tidak ikut terkopi, hanya isinya saja.
- Jika `<sumber>` merupakan sebuah file, itu akan dikopikan secara individual bersama metadatanya. Apabila `<tujuan>` diakhiri sebuah tanda trailing slash "/" yang menandakan sebuah folder, maka isi dari `<sumber>` akan ditulis sebagi `<tujuan>/base(<sumber>)`.
- Jika terdapat multiple `<sumber>`, baik secara langsung ataupun menggunakan wildcards, maka `<tujuan>` harus merupakan folder dan diakhiri dengan trailing slash "/".
- Jika `<tujuan>` tidak terdapat trailing slash, ini berarti dianggap sebagai file biasa, dan konten dari `<sumber>` akan dituliskan ke `<tujuan>`.
- Jika lokasi `<tujuan>` tidak tersedia, maka akan otomatis dibuatkan, juga terhadap setiap folder yang hilang.

Instruksi ARG

Instruksi `ARG` berguna untuk mendefinisikan sebuah variable yang dapat dilewati user ketika melakukan perintah `docker build` dengan opsi `--build-arg <variable>=<value>`. Jika user mendefinisikan sebuah build argument yang tidak terdapat pada dockerfile, maka akan menyebabkan error pada docker build output. Bentuk Instuksi `ARG` adalah sebagai berikut :

```
ARG <variable>[=<default value>]
```

Anda dapat menuliskan Instruksi `ARG` satu kali atau lebih pada dockerfile, sebagai contoh penulisan Instruksi `ARG` pada dockerfile yang benar :

```
FROM busybox
ARG user1
ARG buildno
```

Atau dengan cara menambahkan nilai default pada Instruksi `ARG` :

```
FROM busybox
ARG user1=someuser
ARG buildno=1
```

Apabila Instruksi `ARG` memiliki nilai default, dan tidak diberikan nilai ketika melakukan perintah docker build, maka nilai default akan digunakan.

Instruksi ONBUILD

Instruksi `ONBUILD` akan menambahkan sebuah pemicu (*trigger*) kedalam sebuah image yang akan dijalankan ketika image digunakan sebagai *base image* untuk proses build selanjutnya. Format Instruksi `ONBUILD` adalah sebagai berikut :

```
ONBUILD [Instruksi]
```

Instruksi ini berguna ketika sebuah kode sumber suatu aplikasi melibatkan generator untuk meng-*compile* sebelum aplikasinya bisa digunakan. Setiap instruksi dockerfile kecuali `FROM`, `MAINTAINER`, dan `ONBUILD` dapat digunakan.

Beginilah cara Instruksi `ONBUILD` bekerja :

- Ketika melakukan build, jika menemui Instruksi `ONBUILD`, intruksinya akan didaftarkan sebagai pemicu yang ditambahkan kedalam metadata sebuah image. Proses build akan berpengaruh pada build selanjutnya yang menggunakan image tersebut.
- Ketika akhir melakukan build, daftar dari semua pemicu (*trigger*) disimpan pada image manifest sebagai key yang diberi nama OnBuild. Key tersebut dapat diinspect dengan perintah `docker inspect`.
- Kemudian, image yang mungkin saja anda gunakan sebagai base untuk proses build baru, menggunakan Instruksi `FROM`. Sebagai bagian dari Instruksi `FROM`, builder akan melihat pemicu dari `ONBUILD` dan menjalankannya bersama bagian dimana ia telah didaftarkan. Jika beberapa pemicu terdapat kesalahan, Instruksi `FROM` akan dibatalkan karena proses build juga akan gagal, namun jika semua pemicu sukses, maka proses build akan dilanjutkan sebagaimana umumnya.
- Pemicu akan dibersihkan dari final image setelah dijalankan, dengan kata lain pemicu tidak diturunkan (*inherit*) pada proses build selanjutnya (cucu / *grand-childern*).

Contoh menggunakan Instruksi ONBUILD :

```
[...]
ONBUILD ADD . /app/src
ONBUILD RUN /usr/local/bin/python-build --dir /app/src
[...]
```

BAB III
Docker Playground

Working with Dockerfile

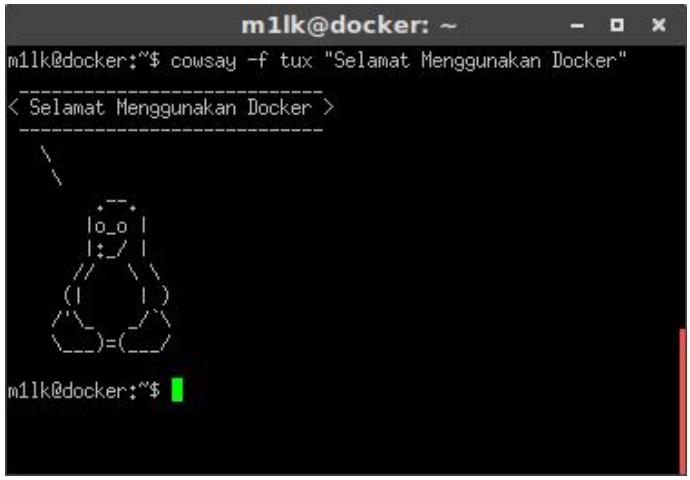
Setelah mengenal macam-macam perintah command line docker, dan instruksi-instruksi yang digunakan untuk membuat dockerfile, kali ini untuk membuat kita lebih jauh memahami bagaimana docker bekerja, kita akan membuat sebuah eksperimen sederhana, yaitu membuat docker image kita sendiri.

Pada contoh project kali ini, saya akan membuat tiga macam dockerfile dimana akan menggunakan Ubuntu sebagai base imagenya, dan menjalankan aplikasi sederhana seperti cowsay, fortune, dan webserver.

Cowsay

Sebelum melanjutkan project ini, sedikit saya review tentang apa itu program cowsay.

Cowsay adalah sebuah program yang menghasilkan sebuah gambar dari ASCII code berupa gambar sapi (default) yang sedang mengucapkan sebuah pesan pada terminal anda. Cowsay juga dapat menampilkan gambar-gambar selain sapi, misalnya Tux the Penguin, koala, kucing, naga, dan lainnya. Contoh cowsay dapat anda lihat pada gambar berikut :



Gambar 3.1

Untuk melakukan project cowsay pertama ini, buat file kosong dengan nama Dockerfile, kemudian tuliskan instruksi-instruksi sebagai berikut :

```
FROM ubuntu:latest
MAINTAINER m1lk <hmdilham@gmail.com>
RUN apt-get update && apt-get install -y cowsay
RUN rm -Rf /var/lib/apt/lists/*
ENTRYPOINT ["/usr/games/cowsay", "-f", "tux"]
CMD ["Halo, Ini Dockerfile pertamaku"]
```

Penjelasan pada Dockerfile diatas :

- `FROM ubuntu:latest`
  Menggunakan base image ubuntu dengan tag latest.
- `MAINTAINER m1lk <hmdilham@gmail.com>`
  Informasi pembuat Dockerfile tersebut.
- `RUN apt-get update && apt-get install -y cowsay`
  Perintah untuk meng-update repository ubuntu dan menginstal aplikasi cowsay.
- `RUN rm -Rf /var/lib/apt/lists/*`
  Menghapus semua file dari apt list, direkomendasikan untuk

mengurangi ukuran images.

- `ENTRYPOINT ["/usr/games/cowsay", "-f", "tux"]`
  Mengeset executable image pada perintah "`/usr/share/cowsay -f tux`"
- `CMD ["Halo, Ini Dockerfile pertamaku"]`
  Menambahkan opsional perintah untuk `ENTRYPOINT` apabila tidak diberikan perintah spesifik ketika menjalankan docker run, maka Instruksi `CMD` ini yang dijalankan (lihat kembali Instruksi `CMD` dan `ENTRYPOINT`).

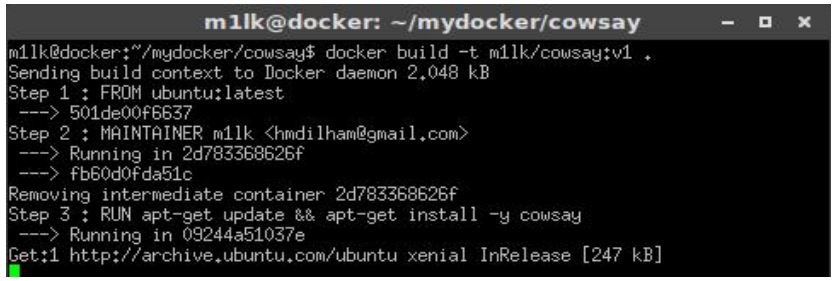
Oke, setelah membuat Dockerfile dengan beberapa Instruksi diatas, kemudian pastikan kursor anda berada dalam satu lokasi dengan Dockerfile tersebut, selanjutnya kita akan membuild image baru dengan perintah `docker build` seperti dibawah ini :

```
$ docker build -t m1lk/cowsay:v1 .
```

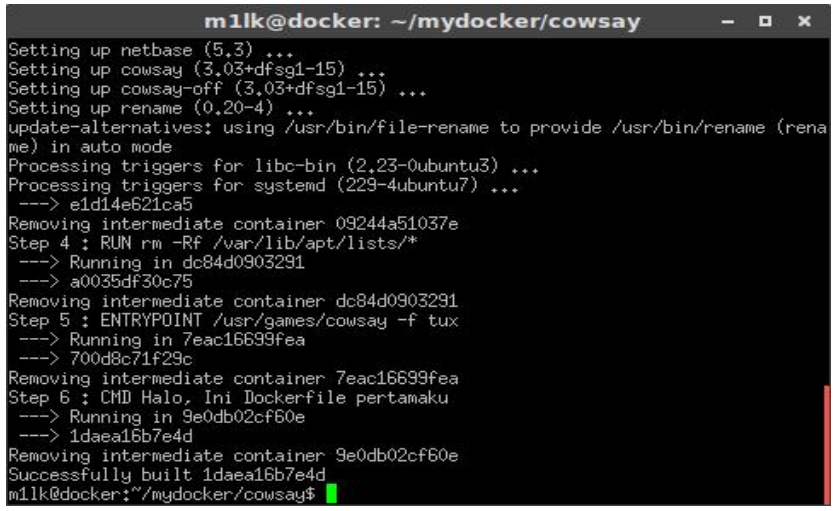
Perhatikan ada tanda "." pada akhir perintah, yang artinya anda sedang menjalankan `docker build` pada lokasi kursor anda berada.

Biarkan proses berlangsung, dan pastikan tidak ada yang error pada proses tersebut. Untuk lebih jelasnya, perhatikan gambar-gambar berikut :

Gambar 3.2



Gambar 3.3



Gambar 3.4

Ketika image tersebut selesai dibuild, anda dapat melihat image baru anda ada di list docker images. Coba ketikkan perintah `docker images`, dan lihat hasilnya.

Gambar 3.5

Langkah selanjutnya, mari kita jalankan image yang baru tersebut, apakah yang terjadi...jeng..jeng..jeng.. :D

```
$ docker run -it --rm --name mycowsay m1lk/cowsay:v1
```

Gambar 3.6

Atau apabila anda menginginkan output dengan kata-kata anda sendiri, anda dapat meng-override Instruksi `CMD` dengan perintah spesifik ketika menjalankan perintah docker run, misalnya :

```
$ docker run -it --name mycowsay1 m1lk/cowsay:v1 \
Berhasil..berhasil..berhasil Hore :D
```

Gambar 3.7

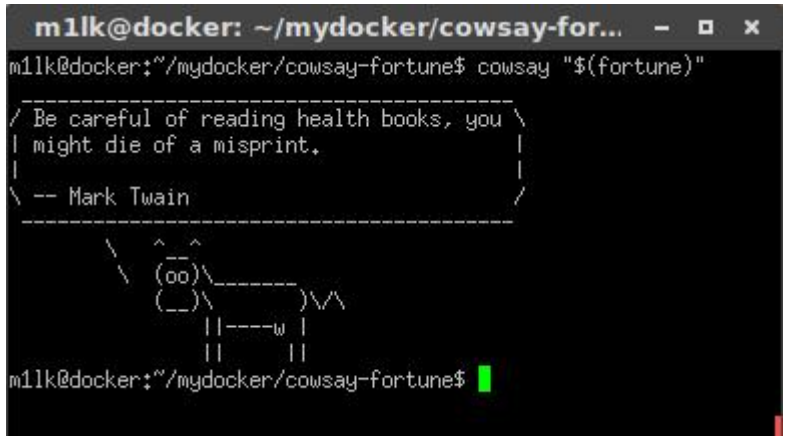
Selamat anda telah berhasil membuat satu image baru, mudah bukan menggunakan docker.

Cowsay dan Fortune

Pada project sebelumnya kita telah membuat satu image baru dengan menjalankan satu aplikasi yang bernama cowsay, setelah ini kita akan mencoba melakukan tambahan aplikasi yang bernama *fortune*.

Fortune memang biasanya digunakan bersamaan dengan aplikasi cowsay, jika aplikasi cowsay ini memberikan sebuah gambar yang disusun menggunakan ASCII, sedangkan fortune ini merupakan aplikasi yang isinya memberikan pesan random.

Nah nantinya gabungan kedua aplikasi ini akan memberikan pesan random dari fortune dan akan ditampilkan oleh gambar yang di generate oleh cowsay, seperti apa contohnya, anda dapat melihat gambar dibawah ini :



Gambar 3.8

Gambar diatas mungkin terlihat sama dengan contoh dockerfile sebelumnya, tapi coba anda perhatikan kembali, kata-kata yang ditampilkan oleh cowsay saat ini tidak dilakukan secara manual, melainkan menggunakan aplikasi fortune.

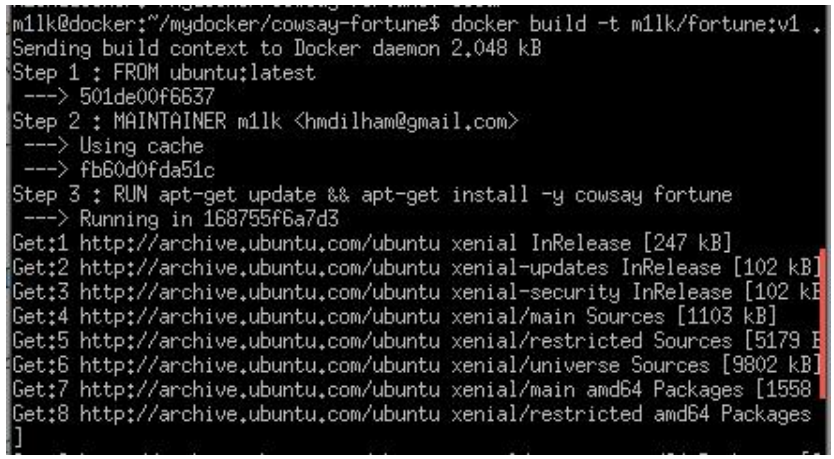
Baiklah, langsung saja kita buat Dockerfile untuk image ini, dan jangan lupa untuk menempatkannya pada working directory yang tepat, jangan dicampur dengan Dockerfile lainnya.

```
FROM ubuntu:latest
MAINTAINER m1lk <hmdilham@gmail.com>
RUN apt-get update && apt-get install -y cowsay fortune
RUN rm -Rf /var/lib/apt/lists/*
CMD /usr/games/cowsay -f dragon $(/usr/games/fortune)
```

Setelah Dockerfile dibuat, kemudian lakukan perintah untuk membuild Dockerfile menjadi image, masih ingat kan perintahnya ?

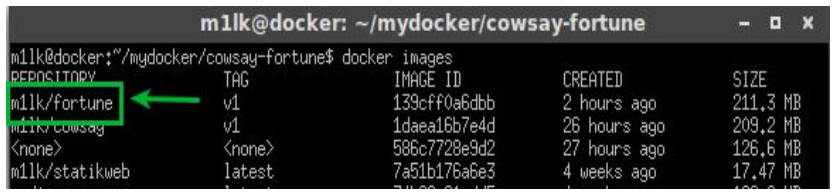
```
$ docker build -t m1lk/fortune:v1 .
```

Kali ini image yang kita build diberi nama m1lk/fortune:v1, silahkan anda ganti dengan nama anda sendiri. Biarkan prosesnya berlangsung, dan pastikan tidak ada pesan kesalahan hingga prosesnya selesai.



Gambar 3.9

Kemudian cek image baru yang telah dibuild dengan perintah `docker images`, dan perhatikan image baru anda.



Gambar 3.10

Selanjutnya mari kita coba image yang telah dibuat tadi, jalankan dengan menggunakan perintah yang sama dengan sebelumnya, hanya saja image yang digunakan adalah m1lk/fortune:v1 :

```
$ docker run -it --rm --name fortune1 m1lk/fortune:v1
```

Apabila berhasil, akan tampak seperti gambar dibawah ini :

Gambar 3.11

Untuk selanjutnya, kita akan sedikit melakukan modifikasi terhadap image yang telah dibuat ini, yaitu dengan mengganti database random message built in fortune, dengan kata-kata yang kita buat sendiri.

Hal yang harus anda ketahui tentang cara membuat docker image adalah seperti anda menjalankan sistem operasi seperti biasanya, begitu juga dengan aplikasi yang dijalankannya.

Fortune menyimpan database random message-nya pada lokasi */usr/share/games/fortunes* dimana didalamnya ada file yang berisi database random message yang dapat anda ganti dengan file database anda sendiri, nama filenya dapat berupa apapun.

Dalam skenario ini kita akan membuat file database yang bernama "pesanku" kemudian isinya kita tambahkan beberapa kata-kata mutiara :p dengan mengikuti format yang telah ditentukan oleh fortune, sebagai berikut :

Pesan 1
%
Pesan 2
%
....

Setiap pesan baru diberikan tanda persen "%" sebagai pemisahnya, berikut ini beberapa contoh kata yang diambil dari website katabijakbagus.com, kemudian disusun seperti format diatas :

Hanya berkata aku selalu berusaha menjadi lebih baik, tak akan buatmu lebih baik, karena yang kamu butuhkan adalah tindakan nyata.
%
Hargai perasaan orang lain. Meskipun hal itu tidak berarti apapun untukmu, bisa jadi hal itu sangat berarti baginya.
%
Hidup hanya sebentar, maka lakukan yang terbaik, nikmati tiap waktunya dan bersyukur atas segala yang ada.
%

Sebagai catatan, ketika anda memasukkan file baru untuk dibaca oleh aplikasi fotrune, file *plaintext* tersebut harus diolah dahulu dengan perintah sebagai berikut :

```
$ strfile /lokasi/namafile
$ strfile /usr/share/games/pesanku
```

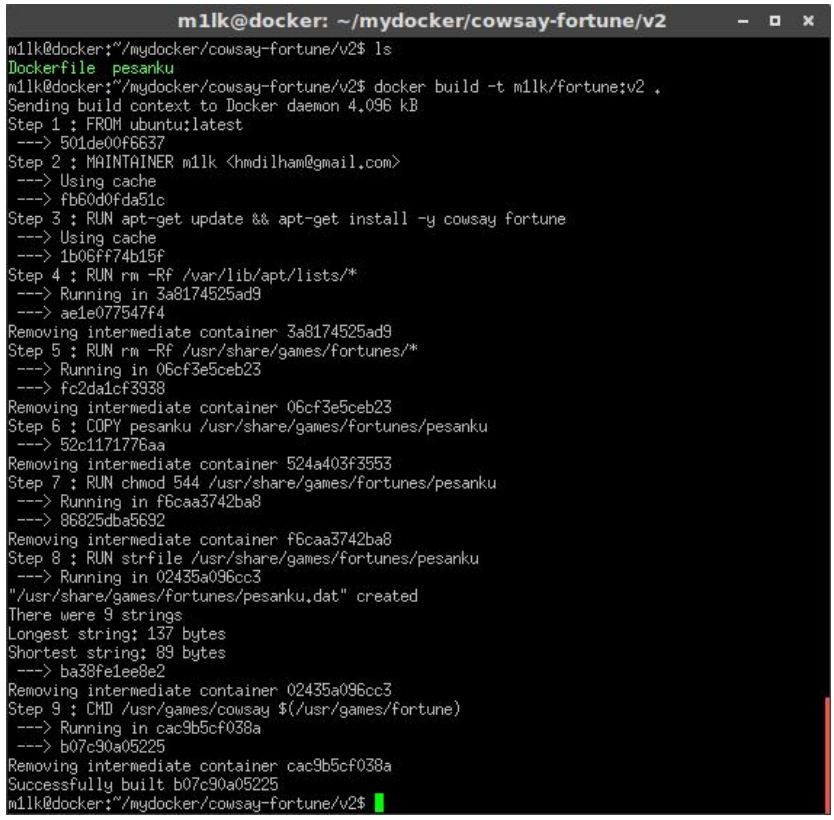
Setelah semua yang dibutuhkan siap, mari kita buat Dockerfile untuk image baru ini, sekedar catatan file pesanku harus berada pada folder yang sama dengan file Dockerfile, dan Dockerfile untuk image kali ini adalah sebagai berikut :

```
FROM ubuntu:latest
MAINTAINER m1lk <hmdilham@gmail.com>
RUN apt-get update && apt-get install -y cowsay fortune
RUN rm -Rf /var/lib/apt/lists/*
RUN rm -Rf /usr/share/games/fortunes/*
COPY pesanku /usr/share/games/fortunes/pesanku
```

```
RUN chmod 544 /usr/share/games/fortunes/pesanku
RUN strfile /usr/share/games/fortunes/pesanku
CMD /usr/games/cowsay $(/usr/games/fortune)
```

Kali ini kita build image tersebut dengan menggunakan tag m1lk/fortune:v2 :

```
$ docker build -t m1lk/fortune:v2 .
```



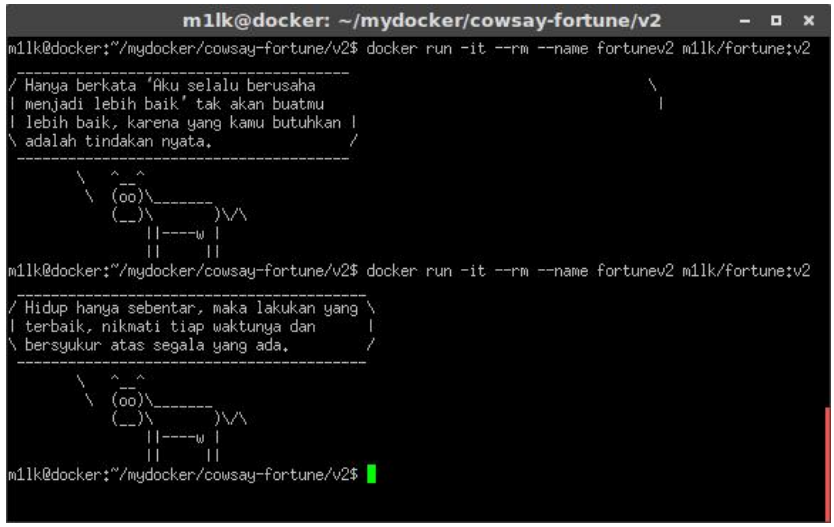
Gambar 3.12

Pastikan proses build berjalan sukses, dan cek kembali image yang sudah terbuat dengan perintah `docker images` :

Gambar 3.13

Kemudian coba anda jalankan image yang dibuat tersebut lebih dari satu kali, dan perhatikan apakah kata-kata mutiara anda sudah ditampilkan :

```
$ docker run -it --rm --name fortunev2 m1lk/fortune:v2
```



Gambar 3.14

Bagaimana ? Sukses kah dengan apa yang sudah anda coba.. :D. Sebelum melangkah untuk mencoba membuat image lainnya, kita akan sedikit membahas perbedaan Dockerfile antara image m1lk/fortune:v1 dengan m1lk/fortune:v2 ini.

Perbedaannya ada pada baris 5-8, dan sedikit pada Instruksi `CMD`, berikut penjelasannya :

- `RUN rm -Rf /usr/share/games/fortunes/*`
  Menghapus eksisting database program fortune.
- `COPY pesanku /usr/share/games/fortunes/pesanku`
  Mengkopi file pesanku ke lokasi dimana database fortune diletakkan, dan menamakan sesuai nama file sumbernya.
- `RUN chmod 544 /usr/share/games/fortunes/pesanku`
  Merubah akses permission file pesanku.
- `RUN strfile /usr/share/games/fortunes/pesanku`
  mem-format file pesanku, agar dapat dibaca oleh program fortune.
- Pada Instruksi `CMD` hanya membuang opsi `-f` dimana opsi ini akan menggunakan bentuk default dari aplikasi cowsay, yaitu akan mengeluarkan gambar sapi.

Setelah melakukan perubahan-perubahan instruksi diatas, image yang dibuat sudah memberikan hasil yang kita inginkan.

Dengan kedua contoh ini, diharapkan anda sudah lebih dalam bagaimana cara membuat sebuah image baru dengan docker, walaupun hanya dengan contoh yang sederhana.

Static HTML Website

Untuk project kali ini kita akan membuat sebuah image baru dengan website sederhana didalamnya.

Base image yang digunakan adalah nginx:alpine dimana image tersebut merupakan image dengan sistem operasi GNU/Linux Alpine yang sudah dikonfigurasi untuk menjalankan webserver Nginx.

Untuk detail mengenai image ini anda bisa kunjungi halaman website https://hub.docker.com/_/nginx/, dan untuk detail Dockerfile image ini dapat anda lihat pada alamat url github berikut https://github.com/nginxinc/docker-nginx/tree/de8822d8d91ff0802989bc0a12f8ab55596a513c/mainline/alpine.

Untuk membuat project kali ini kita harus menyiapkan satu file index.html dan tentu saja Dockerfile itu sendiri, dan jangan lupa menempatkan kedua file tersebut dalam folder yang sama.

Untuk Dockerfilenya cukup sederhana, hanya dengan beberapa baris berikut ini :

```
FROM nginx:alpine
MAINTAINER m1lk <hmdilham@gmail.com>
COPY . /usr/share/nginx/html
```

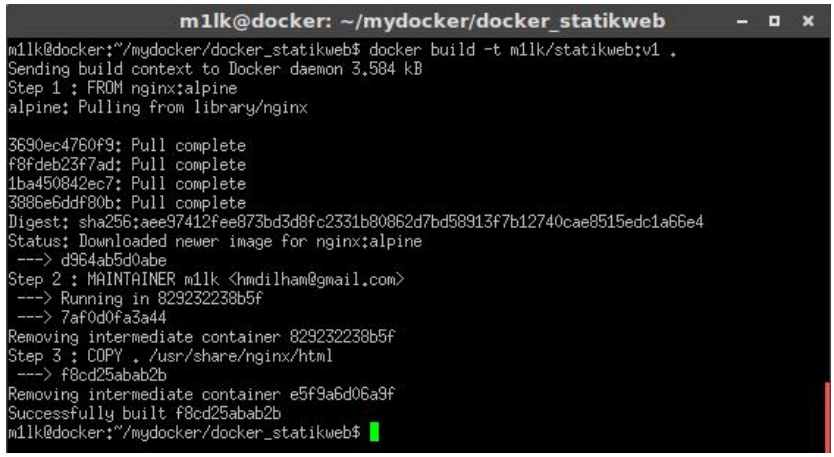
Sedangkan untuk file index.html anda bisa berkreasi sendiri, sebagai contoh saya membuat index.html sederhana dengan isi filenya sebagai berikut :

```
<!DOCTYPE  HTML>
<html>
    <title>Docker Statik Website</title>
    <style>

        html, body {
            height: 100%;
            margin: 0;
            padding: 0;
            width: 100%;
        }

        body {
            display: table;
```

```
        }

        .my-block {
            text-align: center;
            display: table-cell;
            vertical-align: middle;
        }
    </style>
    <body bgcolor="#3d4861">
        <div class="my-block">
            <h2 align="center" style="color: white">
                :: Hello World :: <br />
                ~: Ini adalah website pertamaku dengan
                    menggunakan Docker :~
            </h2>
        </div>
    </body>
</html>
```

Setelah semua file tersedia, lanjutkan untuk membuild Dockerfile menjadi image baru, kali ini saya beri tag image tersebut sebagai m1lk/statikweb:v1, silahkan sesuaikan dengan yang anda inginkan :

```
$ docker build -t m1lk/statikweb:v1 .
```



Gambar 3.15

Cek image yang baru anda build, masih ingatkan perintahnya ?

```
$ docker images
```

Gambar 3.16

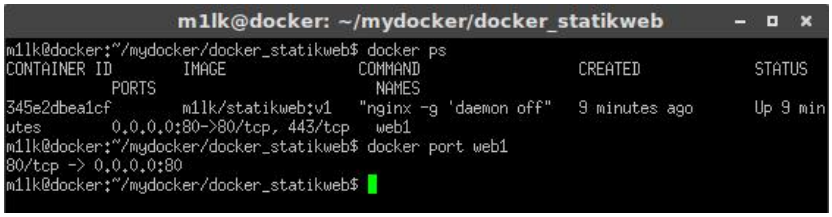
Setelah image berhasil dibuild, selanjutnya kita jalankan image tersebut dengan perintah `docker run` dan ditambahkan opsi `-d` dan `-p` agar service webserver nginx kita berjalan dalam mode backgroud dan expose port service webserver nginx.

Untuk lebih jelasnya, perhatikan perintah berikut ini :

```
$ docker run -d -p 80:80 --name web1 m1lk/statikweb
345e2dbea1cff735331b5ac3b3b1de1dfae8a7d5a06f73392fe
83add20f59a53
```

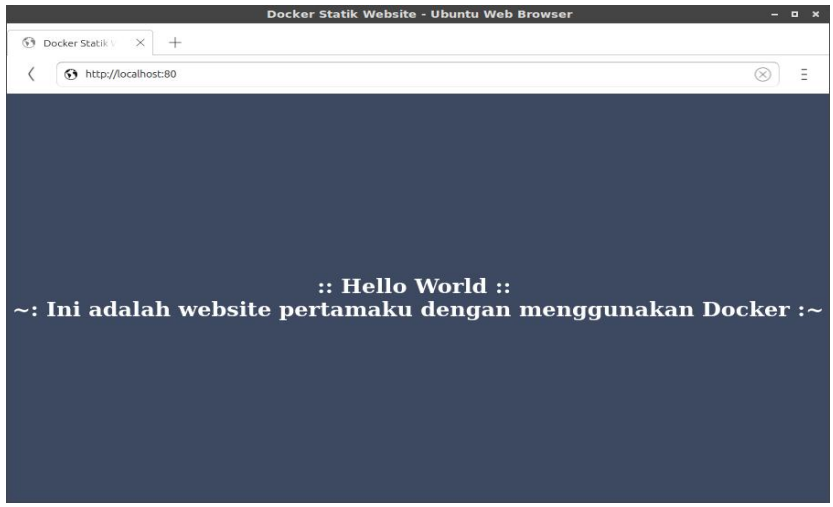
Cek kembali, pastikan container anda berjalan dengan baik :

```
$ docker ps
$ docker port web1
```



Gambar 3.17

Terlihat bahwa container baru anda dengan nama web1 berjalan, dan service webserver nginx di binding port 80 pada host anda. Kemudian, untuk mengakses hasil dari apa yang sudah kita buat, silahkan buka browser anda ketikkan pada url anda http://localhost atau http://127.0.0.1, dan hasilnya dapat dilihat pada gambar dibawah ini.



Gambar 3.18

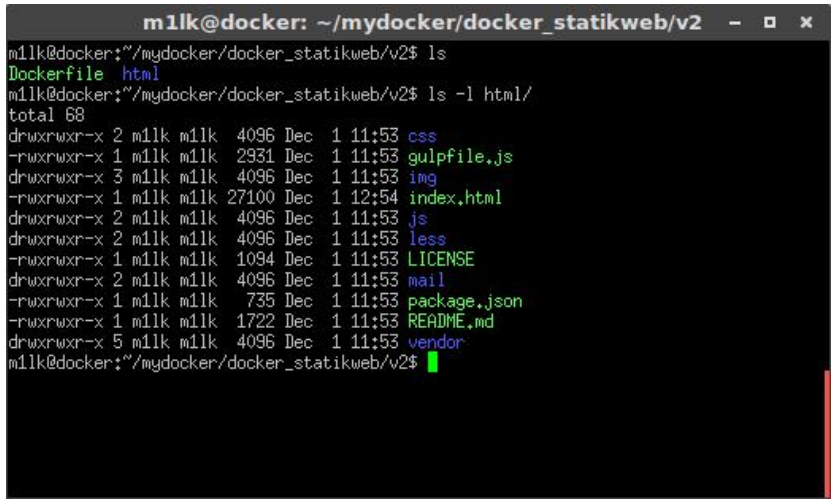
Untuk selanjutnya, mari kita lakukan improvisasi kembali terhadap image m1lk/statikweb ini, sekarang coba anda buat page html yang sedikit lebih menarik dari sebelumnya, kalau tidak mau repot, cukup download saja free template yang banyak tersedia di internet.

Untuk template yang digunakan, saya dapat dari website https://startbootstrap.com/ dengan nama template *freelancer*. Kemudian sedikit dilakukan modifikasi terhadap isinya, agar sesuai dengan apa yang diinginkan.

Oke, setelah selesai melakukan modifikasi terhadap template tersebut, langkah selanjutnya adalah mempersiapkan kembali Dockerfile untuk image ini, dan pastikan juga tata letak dari folder template dan Dockerfile sesuai, agar tidak terjadi kesalahan dalam prosesnya nanti.

Untuk mempermudah, template website yang tadi didownload di ganti namanya (rename) menjadi html yang didalamnya berisi lengkap file-file yang dibutuhkan untuk tampilan website tersebut,

kemudian setara dengan folder html, terdapat Dockerfile, dan pada folder ini pula posisi kursor berada (*working directory*). Untuk lebih jelasnya, lihat gambar berikut ini.



Gambar 3.19

Sedangkan untuk instruksi Dockerfile yang digunakan adalah sebagai berikut :

```
FROM nginx:alpine
MAINTAINER m1lk <hmdilham@gmail.com>
COPY html/. /usr/share/nginx/html/
```

Sedikit berbeda dengan Dockerfile sebelumnya, dimana pada Instruksi `COPY` untuk image m1lk/statikweb:v1 berbentuk :

```
COPY . /usr/share/nginx/html
```

Sedangkan pada image yang akan diberi tag m1lk/statikweb:v2 Instruksi `COPY` nya berbentuk :

```
COPY html/. /usr/share/nginx/html/
```

Setelah semuanya siap, lakukan perintah docker build untuk membuat image baru, dan sesuai tag yang sebelumnya sudah disebutkan.

```
$ docker build -t m1lk/statikweb:v2 .
```

```
m1lk@docker:~/mydocker/docker_statikweb/v2$ docker build -t m1lk/statikweb:v2 .
Sending build context to Docker daemon 2.432 MB
Step 1 : FROM nginx:alpine
 ---> d964ab5d0abe
Step 2 : MAINTAINER m1lk <hmdilham@gmail.com>
 ---> Using cache
 ---> 7af0d0fa3a44
Step 3 : COPY html/. /usr/share/nginx/html/
 ---> d35f3dbb6e62
Removing intermediate container b4df474a7823
Successfully built d35f3dbb6e62
```

Gambar 3.20

Seperti biasanya, cek kembali image yang sudah terbuat dengan perintah docker images.

```
$ docker images
```

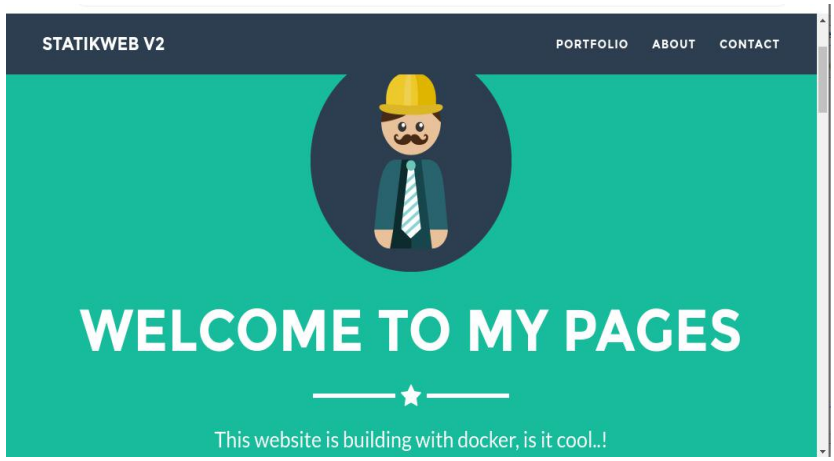
Gambar 3.21

Kemudian coba jalankan kembali image yang sudah dimodifikasi tersebut.

```
$ docker run -d -p 80:80 --name web2 m1lk/statikweb:v2
$ docker ps
$ docker port web2
```

Gambar 3.22

Silahkan cek halaman website anda menggunakan internet browser dengan mengetikkan alamat http://localhost atau http://127.0.0.1, apakah hasilnya sudah sesuai dengan apa yang dibuat sebelumnya.



Gambar 3.23

Mengabaikan file tertentu dengan .dockerignore

Pada pembahasan ini, kita akan membahas bagaimana mengabaikan file tertentu ketika melakukan proses build terhadap docker image yang bertujuan untuk keamanan serta mempercepat proses waktu build terhadap suatu image.

Untuk mencegah file atau folder yang sifatnya sensitif seperti password, dll, terbawa pada proses docker build, anda dapat membuat file yang bernama ***.dockerignore***.

Sebagai contoh, terdapat sebuah Dockerfile yang memiliki perintah untuk mengkopi working directory kedalam docker image. Hasilnya, mungkin saja ini akan memasukan informasi-informasi sensitif seperti halnya file password yang mana seharusnya diatur diluar image.

Isi dari working directory tersebut misalnya sebagai adalah :

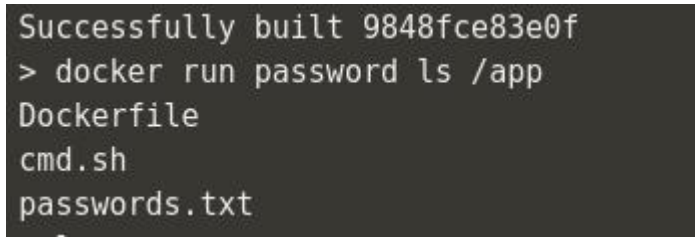
```
$ ls
Dockerfile  cmd.sh  passwords.txt
```

Dan isi dari Dockerfile tersebut adalah :

```
FROM ubuntu
ADD . /app
COPY cmd.sh /cmd.sh
CMD ["sh", "-c", "/cmd.sh"]
```

Setelah image ini dibuild, maka anda dapat melihat bahwa file password ikut masuk kedalam image.

```
$ docker build -t password .
$ docker run password ls /app
```



Gambar 3.24

Untuk mencegah hal tersebut, anda dapat membuat file `.dockerignore` untuk memastikan file-file yang tidak diinginkan ikut terbawa, file `.dockerignore` juga bisa diletakan pada source control untuk dapat di share pada team.

File `.dockerignore` mendukung directory dan regular expression untuk mendefinisikan pengabaian file, jika anda familiar dengan `.gitignore`, file `.dockerignore` pun mirip dengannya.

Untuk menggunakan file `.dockerignore` ini anda dapat melihat contoh dibawah ini :

```
$ echo passwords.txt >> .dockerignore
```

Perintah ini akan memasukan file passwords.txt kedalam file `.dockerignore`. Selanjutnya, coba build image tersebut dan cek apakah file password.txt terdapat masih ada.

```
$ docker build -t nopassword .
$ docker run nopassword ls /app
```

```
> echo passwords.txt >> .dockerignore
> docker build -t nopassword .
Sending build context to Docker daemon 4.096 kB
Step 1 : FROM ubuntu
 ---> 4b1e42b414f6
Step 2 : ADD . /app
 ---> a4ee4fede990
Removing intermediate container 3e6c9e26c509
Step 3 : COPY cmd.sh /cmd.sh
 ---> cedfaf9f7d32
Removing intermediate container 9dff60fd2cad
Step 4 : CMD sh -c /cmd.sh
 ---> Running in 6a31189fbd0a
 ---> d48c5a1ce255
Removing intermediate container 6a31189fbd0a
Successfully built d48c5a1ce255
> docker run nopassword ls /app
Dockerfile
cmd.sh
```

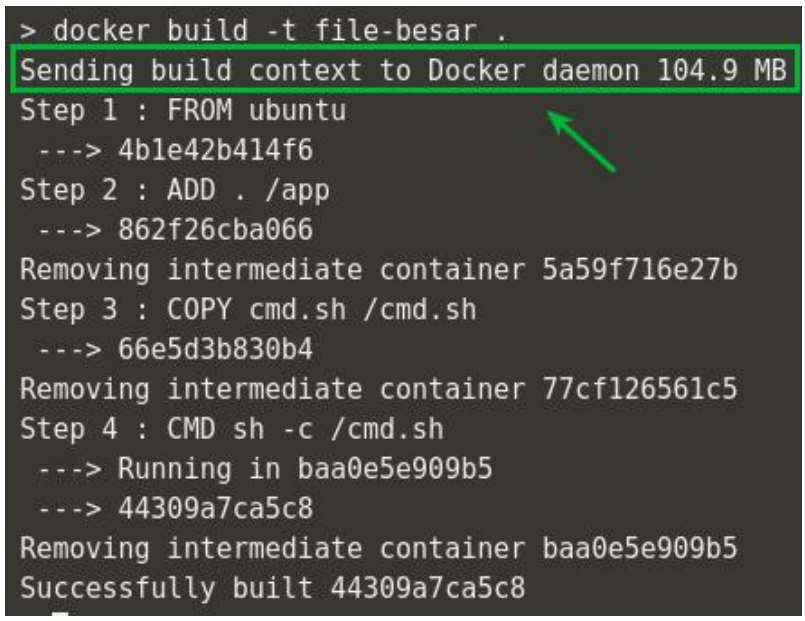
Gambar 3.25

Seperti yang sebelumnya dikatakan, file `.dockerignore` juga dapat mempercepat waktu build ketika misalnya terdapat file-file temporary yang cukup besar.

Dalam percobaan ini, misalnya terdapat temporary file yang mencapai hingga 100M. Ketika anda melakukan perintah docker build, docker akan mengirimkan seluruh file kepada docker daemon untuk dikalkulasi file-file mana yang diikutkan. Sebagai hasilnya, mengirimkan file dengan ukuran 100M yang tidak dibutuhkan akan menambahkan waktu build dan menjadikannya lebih lambat.

```
> ls -lh
total 12K
-rw-r--r-- 1 scrapbook scrapbook   71 Dec  5 00:20 Dockerfile
-rw-r--r-- 1 scrapbook scrapbook 100M Dec  5 00:20 big-temp-file.img
-rwxr-xr-x 1 scrapbook scrapbook   19 Dec  5 00:20 cmd.sh
-rw-r--r-- 1 scrapbook scrapbook   12 Dec  5 00:20 passwords.txt
```

Gambar 3.26

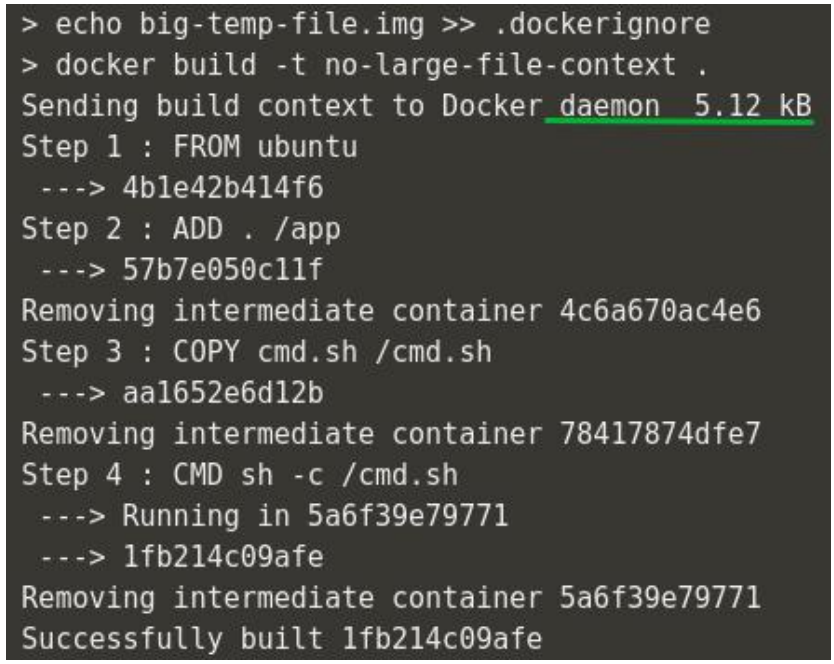


Gambar 3.27

Dengan cara yang sama, kita gunakan file `.dockerignore` untuk memasukan file-file yang tidak seharusnya diikutkan kedalam image yang akan dibuat.

Untuk membuat image anda lebih optimal, cukup dengan cara yang sederhana, yaitu dengan memasukan file big-temp-file.img kedalam `.dockerignore` anda akan merasakan perbedaan waktu yang lebih cepat dalam proses build.

Coba sekarang anda lakukan build terhadap image tersebut, kemudian perhatikan perbedaannya. Sebagai contoh perbandingannya anda dapat perhatikan pada gambar berikut ini.



Gambar 3.28

Note : .dockerfile ini biasanya akan sangat berpengaruh terhadap penggunaan file-file besar seperti directory .git.

Working with Docker Container

Setelah melakukan beberapa eksperimen dengan docker images, diharapkan anda sudah dapat lebih memahami tentang bagaimana membuat docker image.

Untuk kali ini, kita akan membuat beberapa project sederhana dengan beberapa menggunakan docker container yang dapat menambah wawasan anda dalam mengenal fitur-fitur docker.

Container Penyimpan Data

Dalam studi kasusu tertentu, adakalanya sebuah container ingin menyimpan data yang seterusnya akan digunakan, sebagai contoh database misalnya, atau konfigurasi-konfigurasi tertentu, atau juga digunakan sebagai backup data maupun ketika men-debug container.

Untuk melakukan hal tersebut, kita dapat menggunakan fitur docker daemon dengan opsi `-v` yang berguna untuk me-mounting data kedalam container.

Secara umum perintah tersebut menggunakan aturan sebagai berikut :

```
-v <host-dir>:<container-dir>
```

Pada project kali ini kita akan membuat sebuah docker container yang bertanggung jawab untuk menjadi tempat menyimpan atau mengelola data.

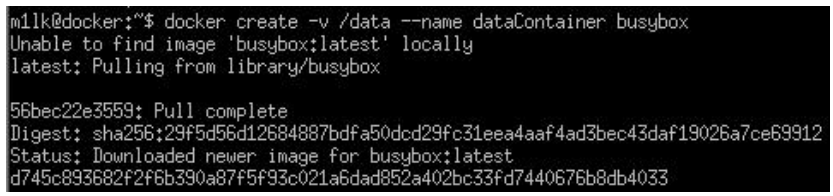
Untuk membuat sebuah Data Container, pertama kali kita buat container dengan nama yang mudah diingat sebagai referensi yang akan digunakan selanjutnya.

Kemudian, kita gunakan image busybox sebagai base image, yang mana kita ketahui, busybox adalah linux dengan ukuran kecil dan sangat ringan, sehingga mudah ketika kita ingin meng-eksplore dan memindahkan container tersebut ke host lainnya.

Langkah selanjutnya ketika membuat sebuah container, jangan lupa menambahkan opsi `-v` untuk mendefinisikan dimana container lain akan membaca atau menyimpan datanya kedalam container yang dibuat itu.

Perintah berikut akan membuat sebuah Data Container sebagai contoh untuk menyimpan file-file data yang berlokasi di /data :

```
$ docker create -v /data --name dataContainer busybox
```



Gambar 3.29

Sekarang kita sudah mempunyai container yang siap diisi file-file kedalamnya. Untuk dapat mengkopi file kedalam container, gunakan perintah docker cp. Sebagai contoh saya akan mengkopi file someData kedalam container dataContainer yang tadi dibuat, dengan perintah sebagai berikut :

```
$ docker cp someData dataContainer:/data/
```

Dengan langkah ini, kita telah memasukkan sebuah data kedalam container dataContainer yang kemudian data tersebut dapat digunakan pada container lain yang membutuhkan.

Menggunakan perintah `--volumes-from <container>` kita dapat me-mounting volume dari sebuah container kedalam container lain yang baru dibuat.

Sebagai contoh, kita akan membuat ubuntu container baru yang kemudaian akan menggunakan data dari container dataContainer yang sebelumnya sudah dibuat, dan selanjutnya coba dilihat pada directory /data ubuntu container anda, seharusnya data yang anda simpan sudah berada disana.

```
$ docker run --volumes-from dataContainer \
ubuntu ls /data
someData
```

Gambar 3.30

Terlihat bahwa data yang berada pada container dataContainer berhasil di mounting kedalam container baru yang dibuat melalui ubuntu image.

Sebagai catatan, apabila directory yang di-mounting dari container lain terdapat kesamaan pada container yang me-mountingnya, lokasi

tersebut akan digantikan (*replace*) dengan directory dari opsi `--volumes-from` ini, dan anda juga dapat me-mounting banyak volume kedalam sebuah container.

Kemudian, apabila anda ingin memindahkan data kedalam mesin/PC yang berbeda, anda dapat menggunakan fasilitas import / export yang disediakan oleh docker daemon, perintah ini akan melakukan compresi file teradap container yang di export dengan tipe file .tar.

Perintah export dapat dijalankan dengan perintah sebagai berikut :

```
$ docker export dataContainer > dataContainer.tar
```

Sedangkan untuk melakukan import, gunakan perintah berikut :

```
$ docker import dataContainer.tar
```

Komunikasi Antar Container Dengan Link

Pada project kali ini kita akan mengeksplore bagaimana beberapa container saling berkomunikasi antar satu dan yang lainnya dengan fasilitas link.

Untuk environment yang digunakan kali, kembali kita gunakan container redis sebagai data store yang akan dihubungkan oleh sebuah container aplikasi yang terpisah.

Sangat umum digunakan ketika anda menghubungkan sebuah aplikasi dengan data store. Dengan docker, kunci ketika menggunakan link untuk menghubungkan antar container adalah penamaan dari container itu sendiri.

Setiap container memang memiliki nama, namun untuk membuat lebih mudah ketika menghubungkan container dengan link, sangat penting untuk memberikan nama yang mudah diingat bagi container sumber yang akan dihubungkan.

Untuk itu, langsung saja kita jalankan sebuah container redis dengan nama *redis-server* yang akan dijadikan container sumber data store anda.

```
$ docker run -d --name redis-server redis
```

Kemudian gunakan opsi `--link <nama|id container>:<alias>` ketika menjalankan container baru yang ingin dihubungkan kepada

container redis-server tersebut. Nama container mendefinisikan sumber container yang telah dibuat pada langkah sebelumnya, sedangkan alias mendefinisikan nama dari host-nya.

Dengan adanya alias, kita memisahkan bagaimana aplikasi kita diatur dan bagaimana infrastruktur digunakan. Dengan kata lain konfigurasi aplikasi tidak perlu diubah sebagaimana dia terhubung ke berbagai environment lainnya.

Bagaimana sebuah link bekerja ?

Dalam contoh ini kita coba gunakan kembali image ubuntu yang akan di-link ke container *redis-server*, dan kita akan mendefinisikan aliasnya sebagai *redis*.

Ketika link tersebut terbuat, docker akan melakukan dua hal, yaitu :

Pertama, docker akan men-set beberapa environment variable berdasarkan link ke sebuah container. Environment variable ini akan memberikan informasi seperti port dan alamat IP. Anda dapat melihat semua environment variable dengan perintah env, sebagai contoh, lihat perintah berikut ini :

```
$ docker run --link redis-server:redis ubuntu env
```

Gambar 3.31

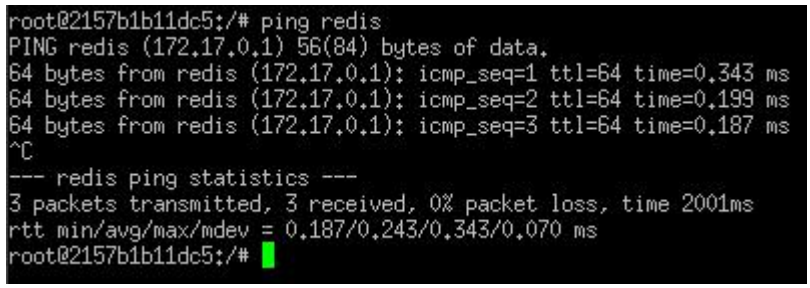
Kedua, docker akan meng-update file HOSTS ubuntu container dengan menambahkan sebuah input untuk sumber container berupa

nama asli container, alias, dan hash id. Untuk dapat lebih jelasnya, perhatikan output perintahnya pada gambar dibawah ini :

Gambar 3.32

Dari hasil tersebut, sudah dipastikan bahwa antar kedua container (ubuntu dan redis-server) sudah terhubung dengan menggunakan fasilitas link, anda dapat mencoba melakukan ping antar container sebagaimana suatu host terhubung dalam satu jaringan.

```
$ docker run -it --link redis-server:redis ubuntu
# apt-get update && apt-get install -y iputils-ping
# ping redis
```



Gambar 3.33

Wordpress Container

Setelah sebelumnya kita mempelajari bagaimana cara kerja link dari docker, saat ini kita akan coba membuat sebuah project container mysql yang akan berkomunikasi dengan wordpress menggunakan link.

Pada docker registry hub terdapat official image dari wordpress yang dapat anda lihat pada halaman https://hub.docker.com/_/wordpress/, abaikan intro dari image tersebut, dan fokus pada manual penggunaannya.

How to use this image

```
$ docker run --name some-wordpress --link some-mysql:mysql -d wordpress
```

The following environment variables are also honored for configuring your WordPress instance:

- `-e WORDPRESS_DB_HOST=...` (defaults to the IP and port of the linked `mysql` container)
- `-e WORDPRESS_DB_USER=...` (defaults to "root")
- `-e WORDPRESS_DB_PASSWORD=...` (defaults to the value of the `MYSQL_ROOT_PASSWORD` environment variable from the linked `mysql` container)
- `-e WORDPRESS_DB_NAME=...` (defaults to "wordpress")
- `-e WORDPRESS_TABLE_PREFIX=...` (defaults to "", only set this when you need to override the default table prefix in wp-config.php)
- `-e WORDPRESS_AUTH_KEY=...`, `-e WORDPRESS_SECURE_AUTH_KEY=...`, `-e WORDPRESS_LOGGED_IN_KEY=...`, `-e WORDPRESS_NONCE_KEY=...`, `-e WORDPRESS_AUTH_SALT=...`, `-e WORDPRESS_SECURE_AUTH_SALT=...`, `-e WORDPRESS_LOGGED_IN_SALT=...`, `-e WORDPRESS_NONCE_SALT=...` (default to unique random SHA1s)

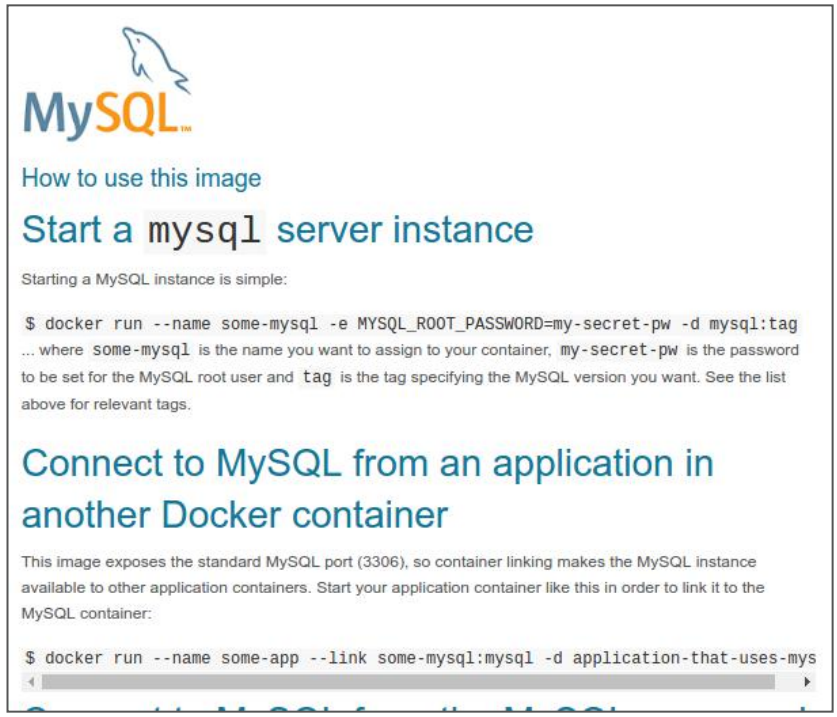
If the `WORDPRESS_DB_NAME` specified does not already exist on the given MySQL server, it will be created automatically upon startup of the `wordpress` container, provided that the `WORDPRESS_DB_USER` specified has the necessary permissions to create it.

Gambar 3.34

Dari informasi tersebut, terlihat bagaimana cara menggunakan image ini, yaitu dengan cara melakukan linking terhadap container data store, dalam contoh diatas yang digunakan adalah MySql, dan juga diberikan beberapa injeksi environment variable kedalam container wordpress ini (`docker run -e VARIABLE_NAME=VALUE`).

Jadi dari informasi yang didapatkan, kita harus menjalankan terlebih dahulu container MySql kemudian baru bisa menjalankan container Wordpress ini.

Baiklah, untuk itu sekarang anda coba buka kembali docker hub dan cari image MySql (https://hub.docker.com/_/mysql/), kemudian perhatikan cara penggunaannya.



Gambar 3.35

Setelah mengetahui informasi cara penggunaan image tersebut, langsung saja kita jalankan container MySql dengan mode *detach,* nama container *mysqldb,* dan mysql password root adalah *my-pw* :

```
$ docker run -d --name mysqdb \
-e MYSQL_ROOT_PASSWORD=my-pw mysql
```

Gambar 3.36

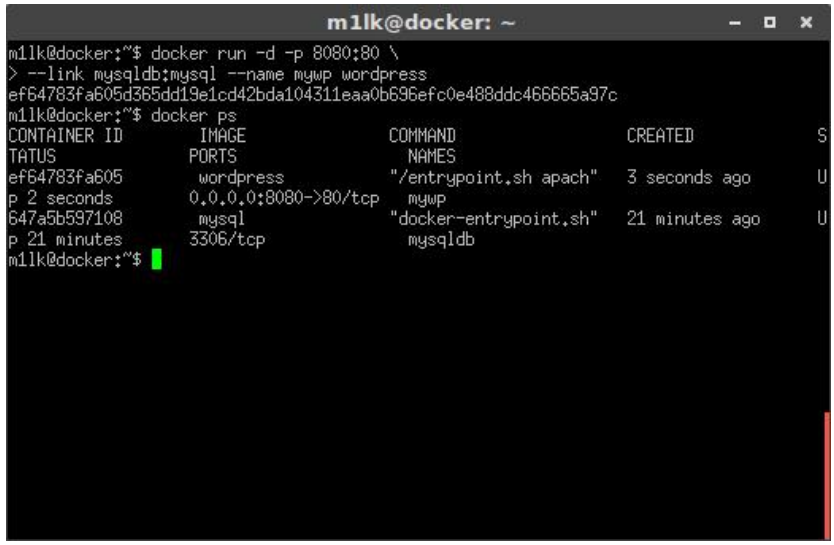
Setelah memiliki satu container MySql yang berjalan, anda bisa lihat statusnya dengan perintah `docker ps`.

Langkah selanjutnya adalah menjalankan container wordpress yang akan di link bersama contaier mysql ini, dan jangan lupa meng-ekspose port yang digunakan.

Untuk melakukannya, lakukan perintah berikut ini :

```
$ docker run -d -p 8080:80 \
--link mysqldb:mysql --name mywp wordpress
```

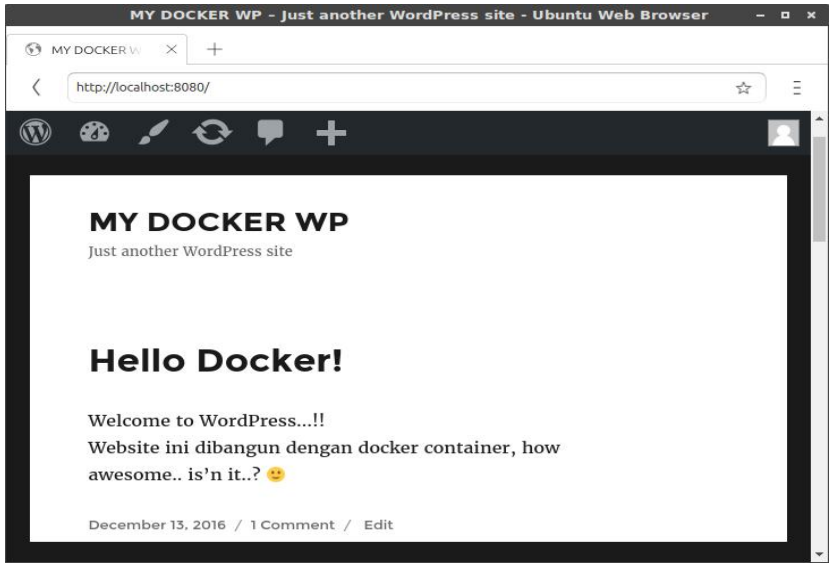
Kemudian cek apakah semua container berjalan dengan baik, dengan perintah `docker ps`.



Gambar 3.37

Terlihat semua container sudah berjalan dengan baik, selanjutnya coba kita pastikan dengan membuka web browser anda, dan ketikkan alamat http://localhost:8080 pada url, apabila anda mendapatkan tampilan instalasi wordpress, tandanya anda telah berhasil me-linking container wordpress dengan mysql.

Gambar 3.38



Gambar 3.39

Komunikasi Antar Container Dengan Network

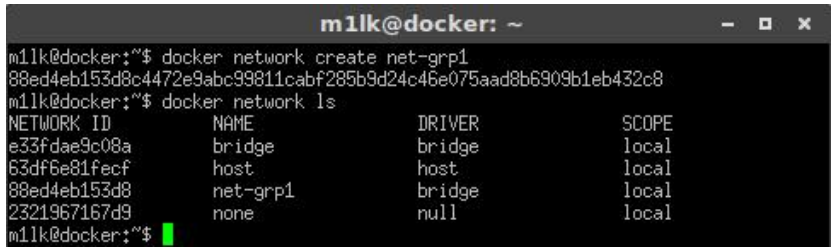
Sebelumnya kita sudah mempelajari bagaimana menggunakan link untuk menghubungkan antar container, ada cara lain ketika ingin menghubungkan beberapa container, yaitu dengan menggunakan fasilitas network ini.

Pada penggunaan link, anda sudah mengetahui bahwa docker melakukan modifikasi terhadap dua hal, yaitu dengan meng-update file /etc/hosts dan environment variable pada container tersebut, tetapi dengan menggunakan network ini, docker akan menggunakan fasilitas embedded DNS Server yang tertanam pada docker daemon.

Untuk menghubungkan beberapa container menggunakan network, anda harus membuat sebuah "network" terlebih dahulu dengan docker CLI, yang kemudian digunakan untuk meletakkan container-container pada network tersebut sehingga dapat saling berkomunikasi antar satu dengan lainnya.

Dalam contoh berikut ini, saya akan membuat sebuah network dengan nama *net-grp1* dengan perintah sebagai berikut :

```
$ docker network create net-grp1
$ docker network ls
```



Gambar 3.40

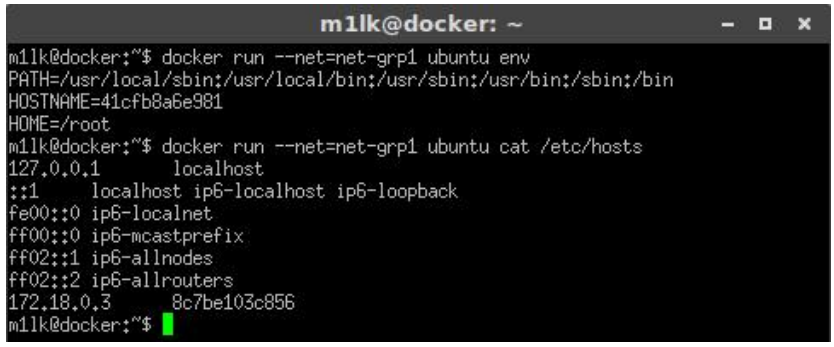
Untuk menghubungkan sebuah container kedalam network yang dibuat, anda harus memasukan opsi `--net=network_name`, sebagai contoh, saya akan menjalankan container redis yang akan dimasukan kedalam network net-grp1, berikut perintahnya :

```
$ docker run -d --name=redis1 --net=net-grp1 redis
```

Cara menghubungkan container link dan dengan network sangat berbeda, docker network berperilaku sama seperti jaringan tradisional pada umumnya, dimana setiap node dapat digabung ataupun dipisah.

Hal pertama yang perlu anda ingat dengan docker network ini adalah, docker tidak lagi menambahkan environment variable dan meng-update file /etc/hosts pada container. Kita akan buktikan hal tersebut, dengan cara menjalankan kembali container ubuntu seperti pada percobaan sebelumnya, lakukan perintah berikut :

```
$ docker run --net=net-grp1 ubuntu env
$ docker run --net=net-grp1 ubuntu cat /etc/hosts
```

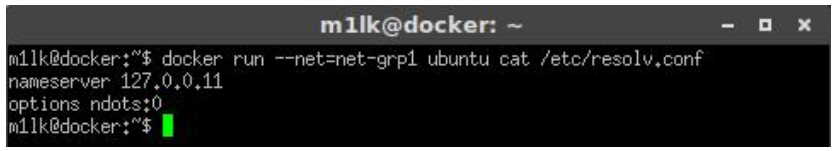


Gambar 3.41

Bandingkan hasilnya antara gambar 3.31 dan 3.32 diatas, dengan gambar 3.41. Terbukti bahwa menggunakan docker network tidak melakukan update terhadap environment variable maupun file hosts pada container. Sebagai gantinya, docker menggunakan *embedded DNS Server* sebagai alat komunikasi yang digunakan oleh para container yang tergabung dalam network tersebut, alamat IP DNS Server ini ditambahkan kepada semua container melalui file /etc/resolv.conf.

Untuk membuktikannya, coba lakukan perintah sebagai berikut :

```
$ docker run --net=net-grp1 \
ubuntu cat /etc/resolv.conf
```



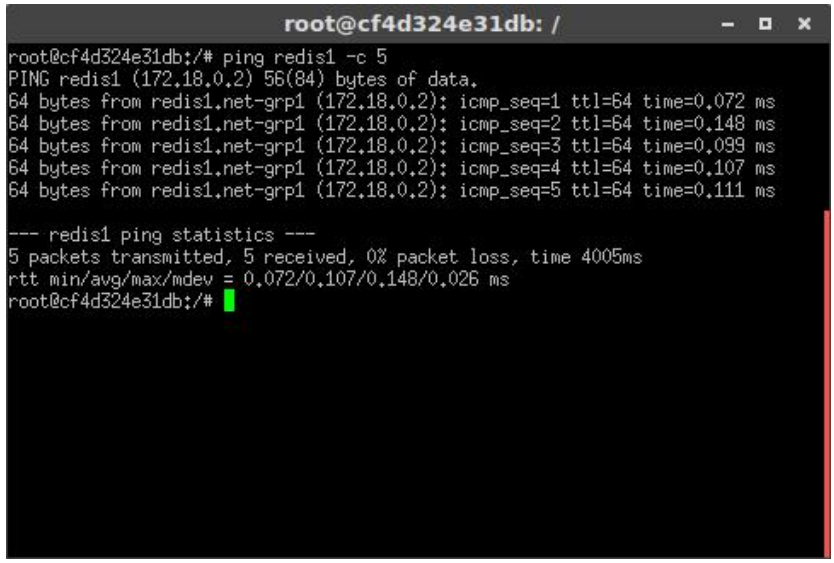
Gambar 3.42

Terlihat bahwa container tersebut menggunakan alamat IP DNS Server dengan alamat 127.0.0.11.

Sebagaimana DNS Server bekerja, apabila ada suatu container yang mengakses container lainnya melalui nama yang dikenal, misalnya redis1 container, DNS Server akan me-resolve nama domain container tersebut kedalam alamat IP, sehingga container tersebut dapat dihubungi, pada gambar dibawah ini terlihat bahwa redis1 memiliki FQDN redis1.ne-grp1.

```
$ docker run -it --net=net-grp1
# apt-get update && apt-get install -y iputils-ping
# ping redis1
```

See the result below.. :D



Gambar 3.43

Memastikan Container Tetap Berjalan

Sebagaimana sebuah proses, docker container juga bisa saja mengalami crash yang mengakibatkan berhentinya container tersebut. Pada pembahasan kali ini, kita akan mempelajari bagaimana memastikan container tetap berjalan dan merestart secara otomatis apabila container tersebut mengalami crash.

Bagaimana docker menilai sebuah container itu berhenti karena

kesalahan sistem ?. Docker daemon akan menganggap setiap container yang berhenti dengan *non-zero exit code* merupakan kegagalan sistem, dan secara default container yang mengalami crash juga akan berhenti.

Pada contoh berikut ini kita menjalankan sebuah container yang sengaja dibut untuk mensimulasikan exit code 1 sebagai indikasi container yang crash.

```
$ docker run -d --name restart-default \
scrapbook/docker-restart-example
```

Kemudian, jika lakukan perintah `docker ps -a` untuk melihat status container tersebut, maka anda lihat bahwa container yang dijalankan telah berhenti dengan exit code 1.

Gambar 3.44

Begitu juga apabila melihat logs dari container tersebut, yang dalam penggunaannya dapat memberikan informasi penting dari sebuah container untuk mempermudah menganalisa masalah yang terjadi, apabila kita lihat logs container restart-default maka akan terlihat tampilan seperti dibawah ini :

```
$ docker logs restart-default
Wed Dec 14 08:02:30 UTC 2016 Booting up...
```

Berdasarkan skenario kita, me-restart sebuah proses yang fail mungkin saja dapat menyelesaikan masalah. Untuk itu, docker daemon memiliki fasilitas untuk melakukan restart secara otomatis dengan beberapa kali percobaan sebelum akhirnya berhenti.

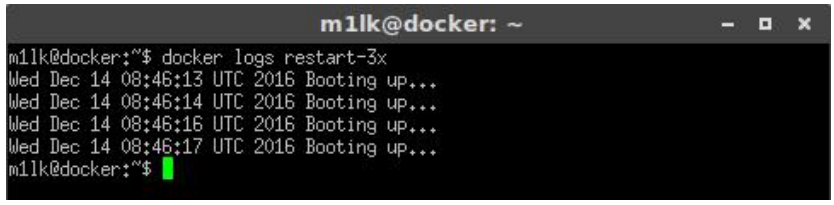
Untuk melakukan hal tersebut, opsi --restart=on-failure:# mengijinkan anda untuk melakukan berapa kali docker akan restart otomatis terhadap container anda.

Contoh berikut ini akan menunjukan bagaimana container dijalankan sebanyak tiga kali sebelum akhirnya berhenti.

```
$ docker run -d --name restart-3x \
--restart=on-failure:3 \
scrapbook/docker-restart-example
```

Begitu juga apabila anda lihat pada logs container tersebut akan menampilkan bahwa ia sudah di restart sebanyak tiga kali.

```
$ docker logs restart-3x
```

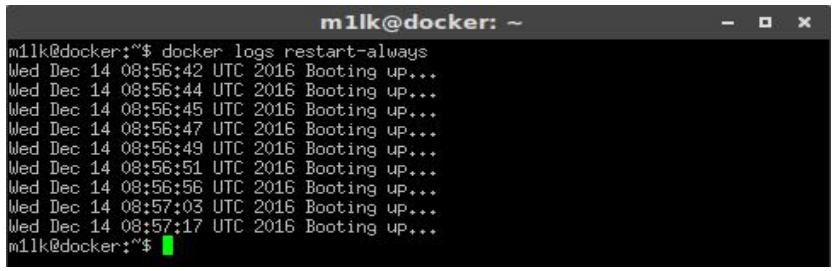


Gambar 3.45

Terakhir, docker juga dapat selalu melakukan restart otomatis tanpa ada batasan berapa kali terhadap failed container, docker akan terus mencoba menjalankannya sampai secara eksplisit container tersebut diperintahkan untuk berhenti. Opsi untuk melakukan hal tersebut ialah menggunakan perintah --restart=always, sebagaimana dicontohkan pada perintah dibawah ini:

```
$ docker run -d --name restart-always \
--restart=always \
scrapbook/docker-restart-example
```

Kemudian apabila anda perhatikan pada logs nya, akan didapati bahwa container tersebut berulang kali di restart.



Gambar 3.46

Menampilkan Statistik Container

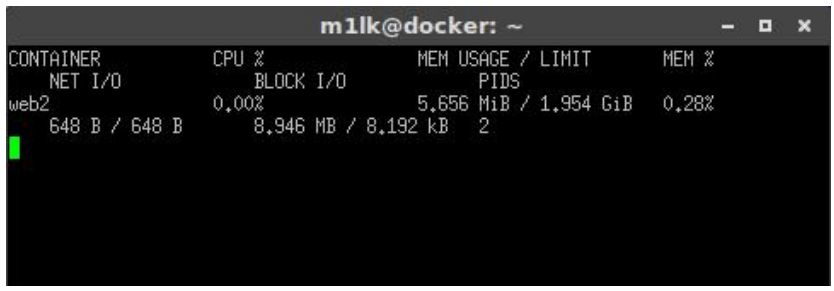
Ketika menjalankan docker container untuk running production, sangat penting dilakukan monitoring resource seperti menjalankan service pada umumnya, untuk mengetahui statistik penggunaan CPU, dan memory agar memastikan container berjalan sebagaimana mestinya.

Pada pembahasan kali ini, kita akan mempelajari tentang `docker stats`, yang berfungsi untuk menampilkan metrics resource terhadap container yang berjalan.

Untuk melakukannya, coba kita jalankan salah satu container yang pernah dibuat sebelumnya, untuk kali ini saya menggunakan container static html website yang bernama *web2*. Sebagai catatan, nama yang digunakan dapat saja berbeda-beda, perhatikan kembali nama container yang pernah anda buat sebelumnya.

```
$ docker restart web2
$ docker stats web2
```

Untuk hasilnya akan tampak seperti gambar dibawah ini :



Gambar 3.47

Untuk memberhentikan proses monitoring ini, tekan key `CTRL+C`.

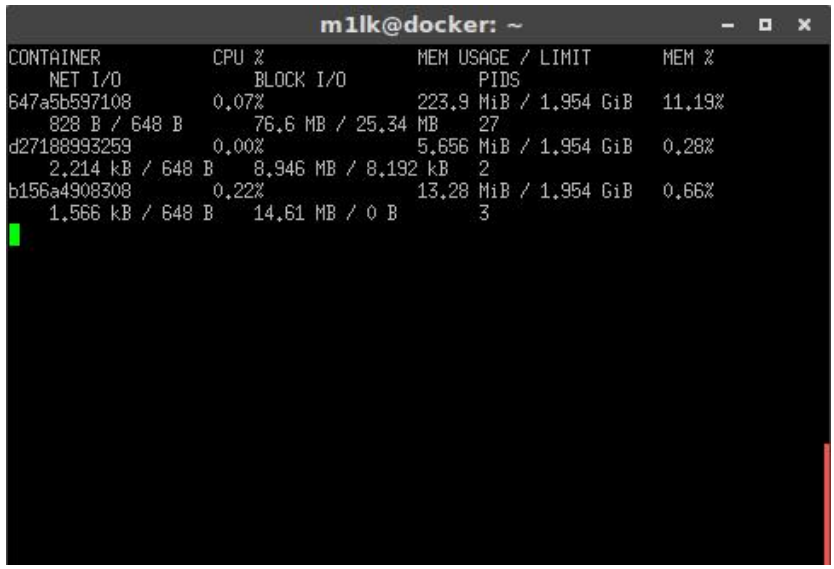
Lalu bagaimana apabila ingin melakukan monitoring terhadap lebih dari satu container, sebagaimana docker mampu menjalankan container lebih dari satu.

Pada percobaan kali ini, coba jalankan berbagai container yang anda miliki, kemudian untuk melihat statistik dari container-container yang sudah dijalankan, kita dapat mengkombinasikan perintah `docker ps -q` dan `xargs docker stats` dan dengan bantuan `pipes "|"`.

Sebagaimana kita ketahui, perintah `docker ps` akan memberikan informasi tentang container yang berjalan, opsi `-q` akan menampilkan berdasarkan id container, kemudian `pipes` akan melewatkan output terhadap suatu perintah yang nantinya akan ditangkap oleh `xargs` menjadi suatu inputan baru, dan kemudian menjalankan perintah `docker stats` untuk menghasilkan output berupa statistik dari container yang berjalan.

Langsung saja lakukan perintah berikut :

```
$ docker ps -q | xargs docker stats
```



Gambar 3.48

Publishing Images to Docker Registry

Setelah sebelumnya kita telah membuat beberapa docker images untuk kebutuhan produksi, saatnya kita publish image yang telah dibuat ke docker registry.

Seperti yang diketahui, docker registry adalah sebuah layanan yang menyediakan repository docker images yang biasa digunakan ketika kita melakukan perintah docker pull.

Ada banyak docker registy yang dapat anda gunakan, beberapa diantaranya adalah Docker Hub, Quay.io, Tutum.co, dan Google Container Registry, namun secara default docker akan mempublish imagenya ke Docker Hub.

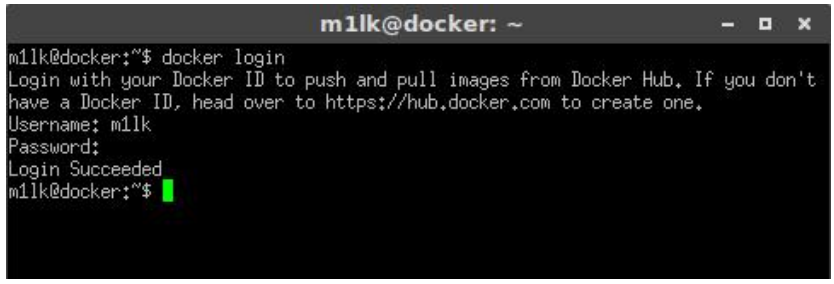
Untuk mempublish image anda ke Docker Hub, pastinya anda harus mempunyai akun di Docker Hub ini, jika belum, buatlah terlebih dahulu. Docker Hub beralamat di https://hub.docker.com/.

Setelah anda memiliki akun pada Docker Hub, selanjutnya kita bersiap untuk mempublish image yang pernah dibuat sebelumnya. Mempublish image di Docker Hub dengan docker command line, membutuhkan autentikasi antara anda dan Docker Hub, anda dapat melakukannya dengan perintah `docker login`.

Setelah mengetikan perintah `docker login`, selanjutnya anda akan ditanya username, alamat email, dan password. Untuk memasukan beberapa parameter tersebut, anda dapat gunakan flag `--username`, `--email`, `--password`.

Coba lakukan perintah docker login untuk otentikasi ke Docker Hub :

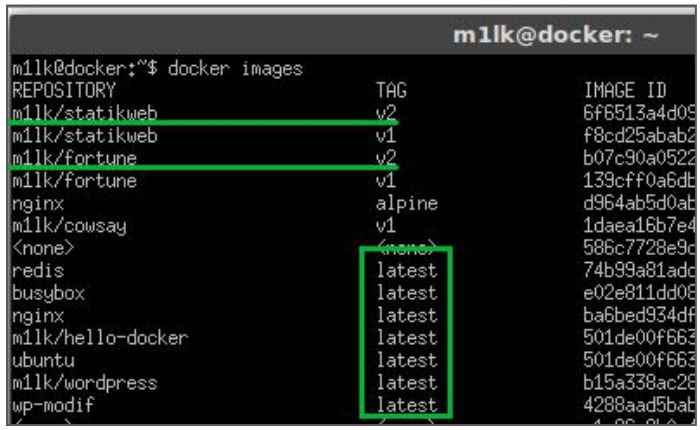
```
$ docker login
```



Gambar 3.49

Kemudian, ketika sudah login, persiapkan image yang ingin anda publish. Namun sebelum itu, perlu di ingat bahwa ketika melakukan pull terhadap image yang tanpa menyertakan tag dari image tersebut, docker daemon akan secara default mengambil image yang berkaitan dengan tag *latest*.

Dari beberapa images yang dibuat sebelumnya, ada image yang dibuat dengan dua versi berbeda, yaitu image statikweb dan fortune. Image ini memiliki versi tag v1 dan v2. Untuk lebih jelasnya perhatikan gambar dari perintah docker images berikut :



Gambar 3.50

Image-image ini, secara default tidak dapat di pull terkecuali menyebutkan tag yang spesifik. Untuk itu, perlu kita ubah tag v2 menjadi latest. Caranya dengan menggunakan perintah sebagai berikut :

```
$ docker tag <id_container> <new_tag>
$ docker tag 6f651 m1lk/statikweb:latest
$ docker tag b07c9 m1lk/fortune:latest
```

Periksa kembali tag image yang sudah diubah dengan perintah `docker images`.

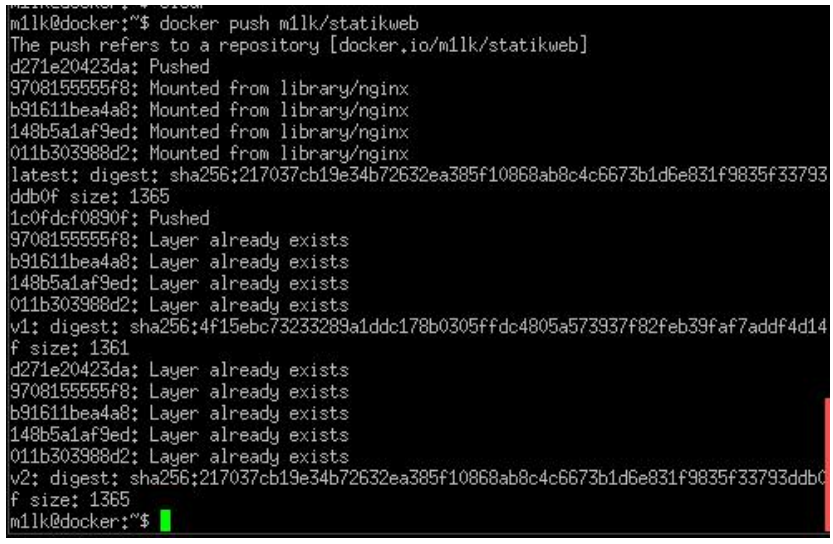
Gambar 3.51

Setelah memiliki tag latest, saatnya kita publis image tersebut menggunakan perintah docker push. Perintah docker push memiliki sintak sebagai berikut :

```
$ docker push <docker_username>/<image>
```

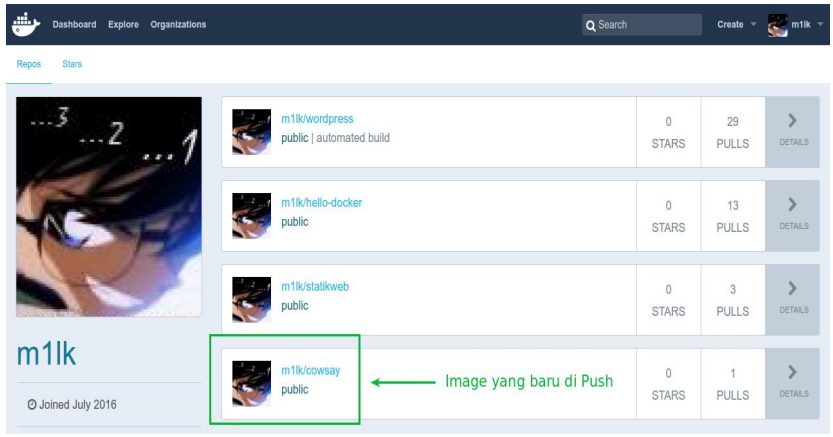
Sebagai contoh saya akan melakukan push terhadap image statikweb, maka saya akan melakukan perintah :

```
$ docker push m1lk/statikweb
```



Gambar 3.52

Seperti yang terlihat, proses pull telah selesai dan tidak ada pesan error yang terjadi, selanjutnya saya akan pastikan bahwa image tersebut sudah berada pada akun Docker Hub.



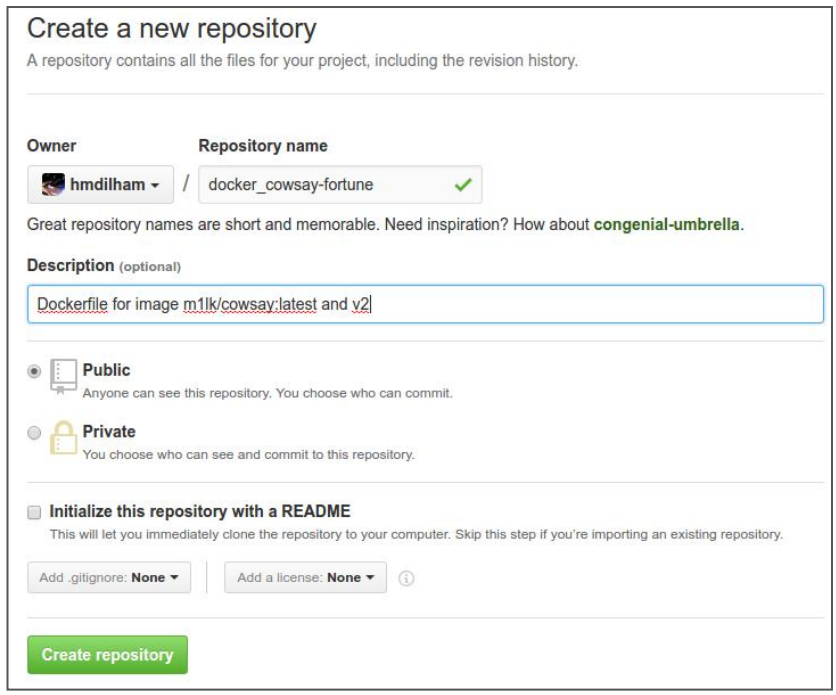
Gambar 3.53

Automatic Build

Selain melakukan push image secara manual dengan perintah `docker push`. Docker juga men-support automatic build dari website versioning seperti GitHub dan Bitbucket. Anda dapat menggunakan yang manapun, namun untuk kali ini, saya akan menggunakan GitHub.

Pada akun GitHub anda, silahkan buat satu repository baru untuk Dockerfile yang akan dipush, dalam hal ini saya akan membuat repository untuk image cowsay, karenanya repository yang dibuat diberi nama docker_cowsay-fortune, anda bebas menamakan repository sesuai keinginan.

Gambar 3.54



Gambar 3.55

Setelah repository terbuat, langkah selanjutnya adalah mem-push file-file yang dibutuhkan, ingat kembali dimana anda meletakan file-file pendukung ketika anda membuat image tersebut, dan pastikan working directory anda ada disana. Dalam hal ini lokasi file yang saya gunakan berada pada directory ~/mydocker/cowsay-fortune.

```
$ cd ~/mydocker/cowsay-fortune
$ ls -l
```

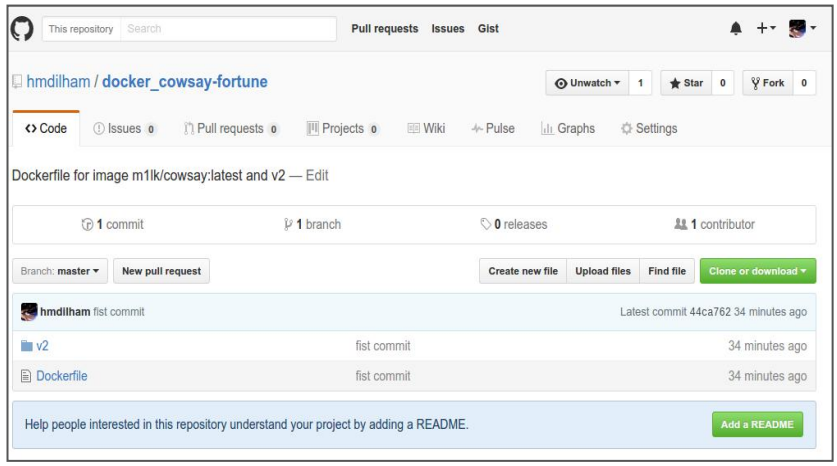
Gambar 3.56

Setelah posisi kursor berada pada diretory yang tepat, selanjutnya lakukan perintah git untuk mem-push file-file tersebut, disini saya tidak menjelaskan secara detail perintah-perintah git, anda dapat mencari referensinya sendiri.

```
$ git add .
$ git remote add origin \
https://github.com/hmdilham/docker_cowsay-fortune.git
$ git commit -m "first commit"
$ git push -u origin master
```

Gambar 3.57

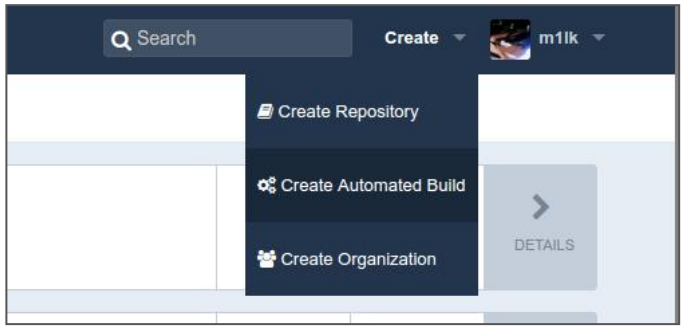
Apabila diminta username dan password, silahkan anda masukan pada baris masing-masing. Setelah login sukses, proses upload akan dilakukan, dan setelah upload sukses, anda bisa lihat file-file tersebut sudah terupload pada repository yang dibuat.



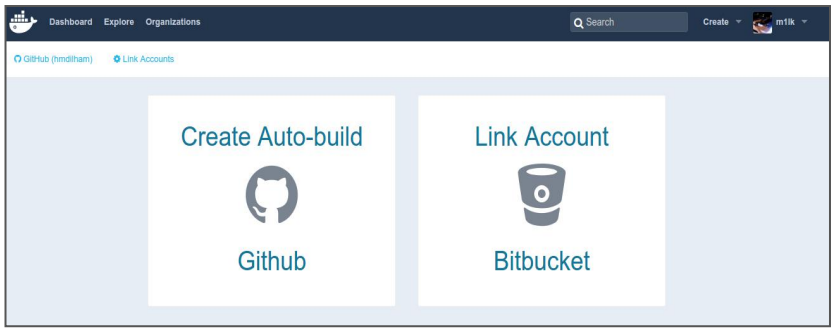
Gambar 3.58

Setelah membuat repository pada GitHub terbuat, selanjutnya kita jalankan proses automatic build di Docker Hub.

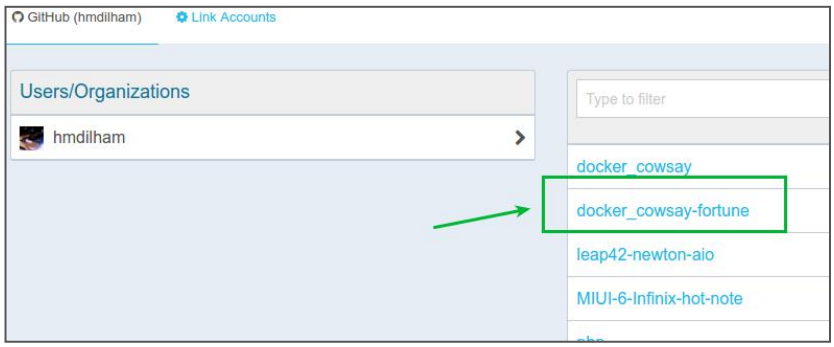
Pada halaman akun Docker Hub anda, di sisi pojok kanan atas klik menu **"Create -> Create Automated Build",** kemudian pilih akun GitHub, dan pilih repository yang akan digunakan.



Gambar 3.59

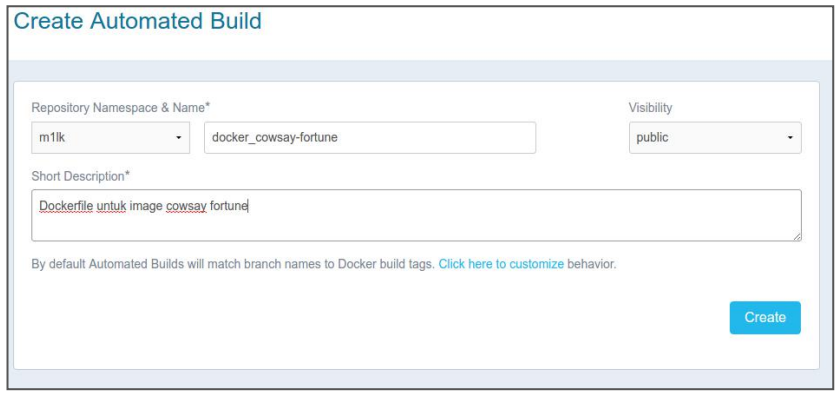


Gambar 3.60

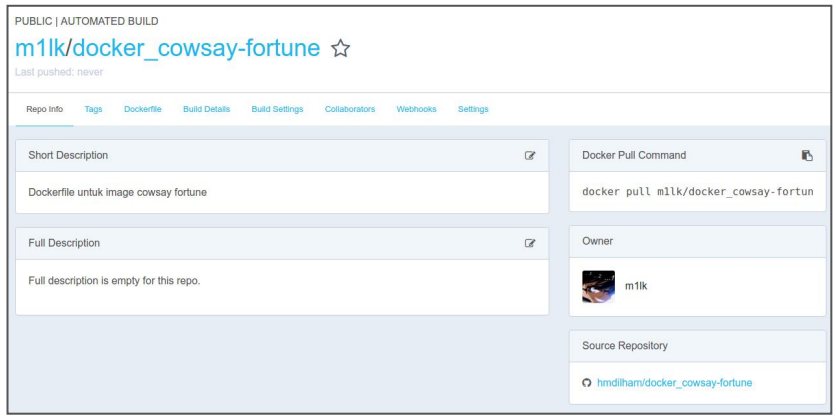


Gambar 3.61

Kemudian, isikan keterangan dari image yang akan di build, dan klik tombol **Create,** dan satu repository baru berhasil dibuat.



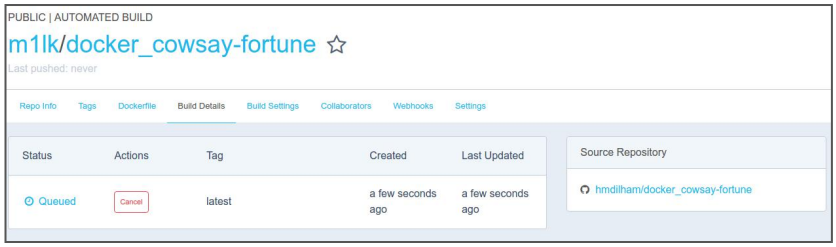
Gambar 3.62



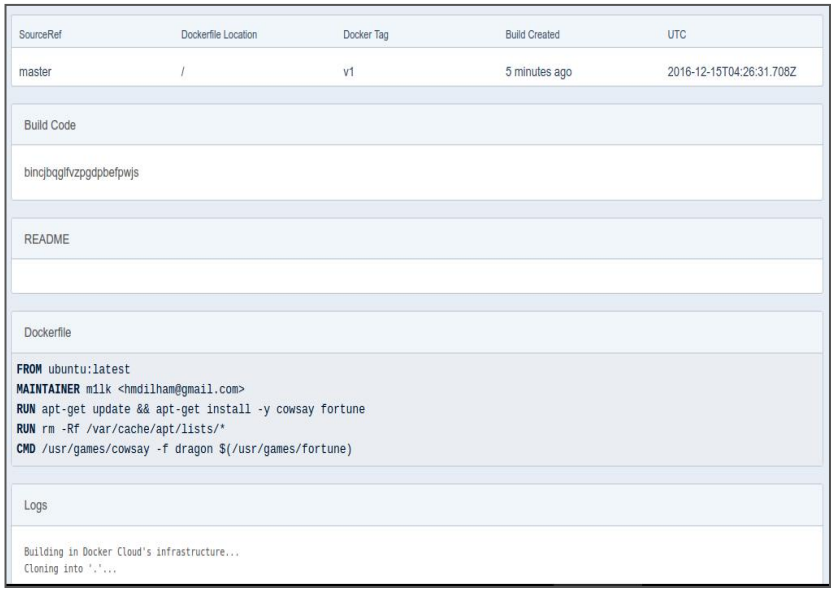
Gambar 3.63

Untuk melihat proses build yang dilakukan Docker Hub, coba klik menu **Build Details**, perhatikan statusnya apakah Queued, Building atau sudah Success.

Anda juga dapat melihat prosesnya dengan cara meng-klik pada proses yang terdapat disana, detailnya akan menampilkan berbagai macam informasi mengenai image yang dibuat, meliputi : SourceRef, Build Code, README, Dockerfile, hingga Logs proses pembuatan image tersebut.



Gambar 3.64

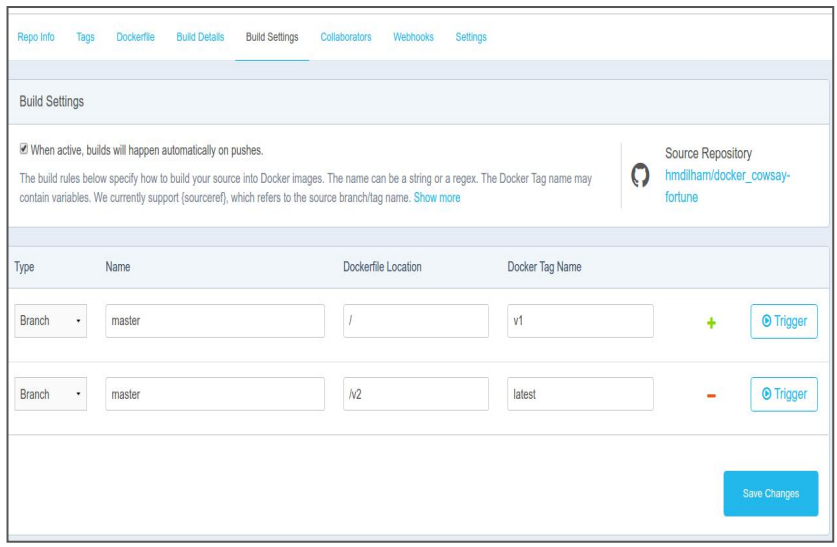


Gambar 3.65

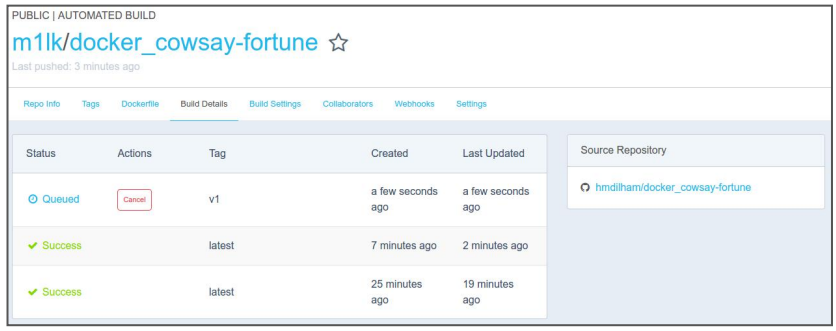
Anda dapat merubah beberapa pengaturan pada menu **Build Settings** seperti apabila anda memiliki lebih dari satu Dockerfile yang biasanya untuk membedakan versi dari image tersebut.

Sebagaimana pada image ini, saya memiliki dua versi Dockerfile yang terdapat pada directory root / dan directory /v2, maka pada menu **Build Settings** saya sesuaikan parameter **Dockerfile Location**, dan sesuaikan juga **Docker Tag Name**, kemudian klik tombol **Save Changes** dan **Trigger** untuk memulai proses build.

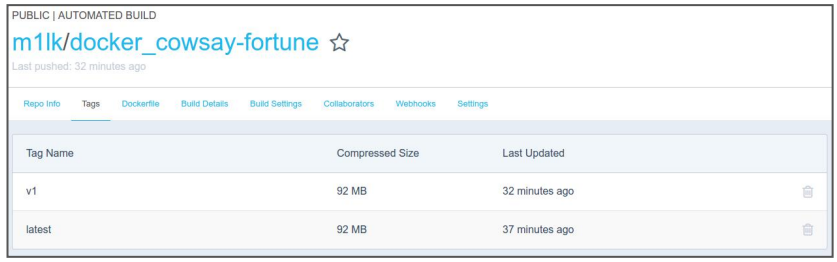
Lalu perhatikan pada menu **Tags**, sekarang image kita memiliki versi yang berbeda.



Gambar 3.66



Gambar 3.67



Gambar 3.68

BAB IV
Docker GUI

Menggunakan Docker tidak harus selalu dengan command line interface, bagi yang sudah sering berhadapan dengan command line mungkin hal ini tidak menjadi kendala, malahan ada istilah kalau pake CLI kesannya sangar. Hihihi... :D.

Namun bagi pemula mungkin sedikit kesulitan dengan hal ini, tapi jangan khawatir, karena saat ini banyak 3rd party software yang dikembangkan oleh berbagai organisasi dan komunitas untuk dapat mengoperasika docker melalui GUI (Graphical User Interface).

GUI pada docker selain untuk mempermudah penggunaanya, juga sangat membantu dalam hal melakukan monitoring container-container yang berjalan.

Banyak sekali referensi yang dapat anda gunakan untuk menjalankan Docker via GUI, beberapa diantaranya adalah Kitematic, Panamax, Docker.ui, Rancher, dan Portainer.io.

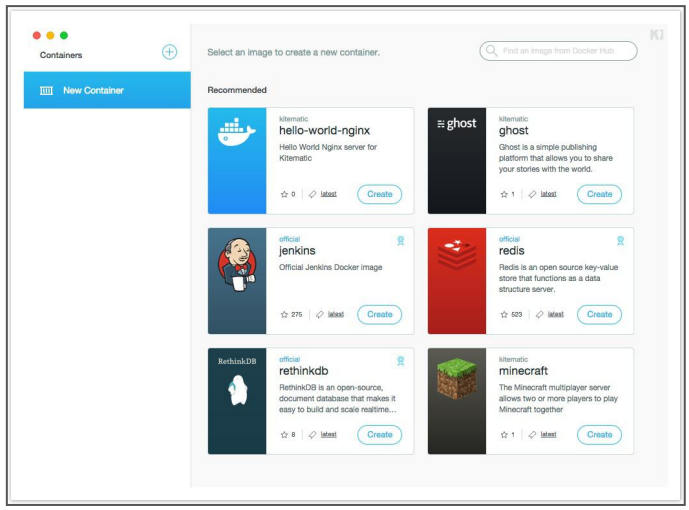
Kitematic

Kitematic merupakan sebuah project Open Source yang dibuat untuk memudahkan menggunakan Docker pada mesin MAC dan Windows. Kitematic memudahkan anda dalam proses automasi instalasi docker dan setup proses dengan menggunakan GUI yang mudah dalam menjalankan berbagai macam docker container.

Kitematic sudah terintegrasi dengan Docker Machine dan VirtualBox ketika anda mulai menginstalasi Docker Toolbox pada mesin anda. Kitematic mempunyai layout interface yang mirip seperti VirtualBox.

Setelah Docker Toolbox terinstalasi, Kitematic GUI akan terbuka, dan dari home screen Kitematic anda disuguhkan berbagai macam images yang dapat anda gunakan. Dari Kitematic interface anda juga dapat langsung mencari docker images di Docker Hub dengan cara mengetikan langsung pada menu pencarian. Di sini juga anda dapat langsung membuat, menjalankan, dan mengatur container dengan hanya meng-klik sebuah tombol.

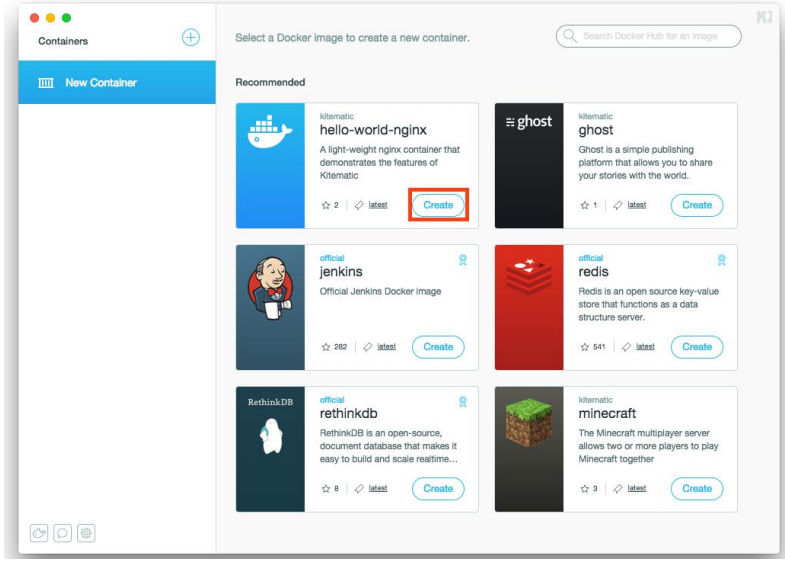
Kitematic juga memungkinkan anda untuk beralih antara Docker CLI dan GUI, dengannya juga anda dapat mengotomasi fitur-fitur advance dari docker, seperti mengelola ports, volume, merubah environment variable, melihat logs, dan masuk ke terminal docker container melalui interface GUI.



Gambar 4.1

Bekerja dengan Container

Pada menu “New Container” di sisi sebelah kiri memungkinkan anda untuk mencari dan memilih docker images dari Docker Hub, apabila anda sudah menemukan image yang dicari, klik tombol “Create” dan Kitematic akan mem-pull image tersebut, membuat, kemudian menjalankan container yang anda pilih.

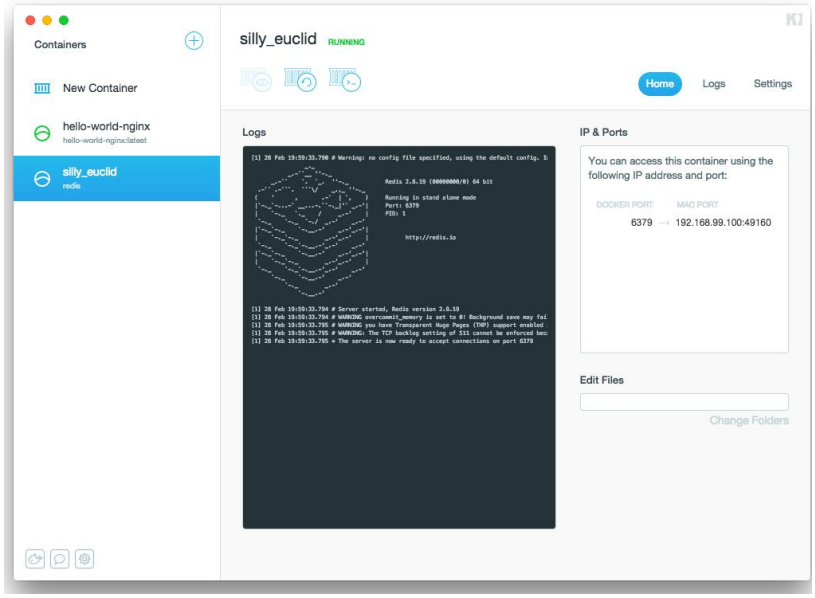


Gambar 4.2

Setelah container baru terbuat, daftar container-container anda ada pada menu sebelah kiri. Pada container dengan status berhenti, atau pause, anda bisa me-restart atau memberhentikan dengan icon-icon yang berada di atas status container anda, begitu juga dengan proses dari container anda, output logs, serta menu setting terdapat disana.

Sedangkan dengan memilih container yang berjalan pada menu sebelah kiri, anda dapat melihat status dari container yang berjalan pada main interface Kitematic, disana juga dapat terlihat hal-hal yang telah anda konfigurasi seperti port yang dibuka, serta volume yang di mounting kedalam contaier.

Untuk lebih jelasnya, anda dapat melihat pada gambar dibawah ini.



Gambar 4.3

Melihat Container Logs

Anda dapat melihat logs dari docker container yang dijalankan, dengan dua cara yaitu meng-klik pada preview image dan logs tab yang berada pada menu atas sebelah kanan.

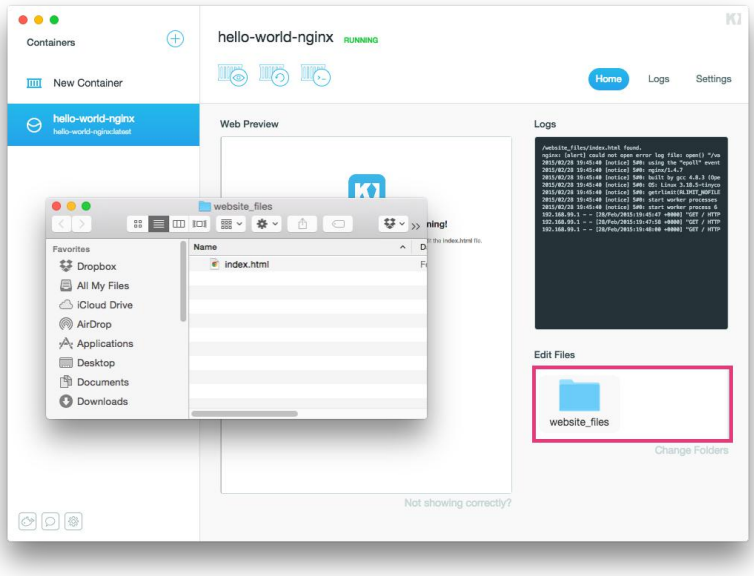
Logs yang ditampilkan untuk container ini dapat anda scroll kebawah, tapi perlu diingat, ketika anda melakukan perubahan pada container dan melakukan restart, maka logs ini akan akan terhapus juga.

Membuka Terminal Untuk Sebuah Container

Dimana ketika menjalankan docker container, ada kalanya anda membutuhkan untuk mengakses shell dari container tersebut yang dilakukan melalui terminal. Untuk melakukan hal ini, anda dapat meng-klik icon terminal pada menu diatas, perintah ini akan menjalankan `docker exec sh <container>` yang memungkinkan anda untuk melakukan perubahan cepat atau men-debug ketika terjadi masalah.

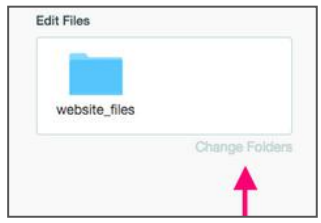
Pengaturan Volume

Anda dapat mengatur container yang anda jalankan dapat menggunakan volume yang termapping ke dalam lokal folder pc anda dengan cara meng-klik “Edit Files” yang terletak pada sebelah kanan bawah main interface Kitematic.



Gambar 4.4

Begitupula ketika anda ingin mengganti lokasi folder yang di mounting kedalam container, anda dapat melakukannya dengan meng-klik “Change Folder” yang terdapat tepat pada menu “Edit Files”. Ketika anda mengganti mounting volume, docker akan berhenti, menghapus, dan membuat ulang container dengan mounting volume yang baru.



Gambar 4.5

Mengganti Nama Container

Secara default, Kitematic akan memberikan nama container sesuai dengan nama image-nya, dan tambahan nomer apabila ada kesamaan nama image. Untuk mengganti nama container anda, masuk pada menu “Settings” kemudian anda dapat merubahnya disana.

Sebagai catatan, setelah anda mengganti nama container, docker akan menghentikan container, menghapus dan membuat ulang container dengan nama yang telah anda sesuaikan.

Menambahkan Environment Variable

Kebanyakan dari image docker membutuhkan environment variable untuk berjalan. Masih pada menu “Settings” disana anda dapat mengatur berbagai macam environment variable yang dibutuhkan sebelum memulai menggunakan container.

Sama halnya ketika anda merubah pengaturan untuk sebuah container, penambahan environment variable juga akan mempengaruhi docker daemon untuk menghentikan container anda, menghapus, dan membuat ulang container dengan environment variable yang telah disesuaikan.

Menampilkan Expose Port dan Cara Mengaksesnya

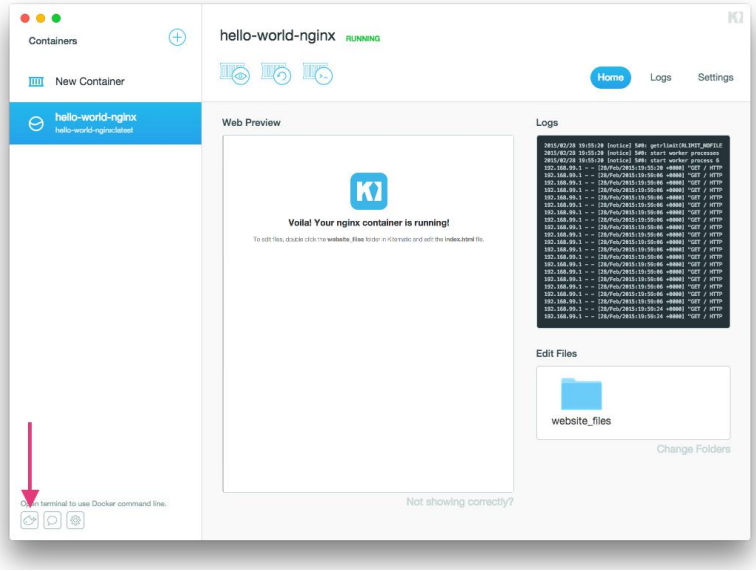
Pada menu “Settings” ini pula, anda dapat melihat port yang di expose pada sebuah docker container.

Navigasikan cursor anda pada menu “Settings -> Ports”, pada halaman ini anda akan melihat semua port yang di ekspose pada container tersebut, juga alamat IP serta jaringan yang terhubung yang dapat anda gunakan untuk mengakses container tersebut dari mesin anda.

Mengakses Command Line

Menggunakan container tidak akan pernah bisa lepas dari yang namanya console (CLI), walaupun dengan tampilan grafis, Kitematic juga memungkinkan anda untuk berinteraksi dengan docker console / docker CLI.

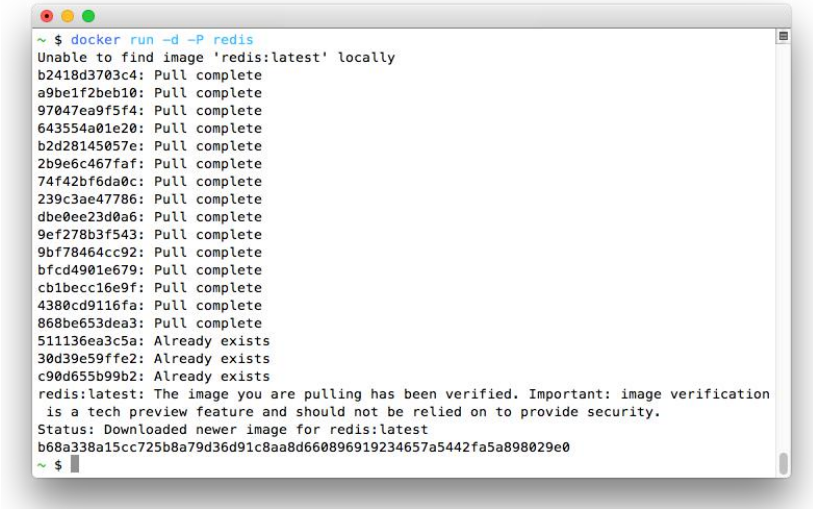
Untuk mengakses docker console / docker CLI, pada interface Kitematic, tekan icon docker (icon ikan hiu) yang berada pada pojok kiri bawah. Untuk lebih jelasnya, perhatikan gambar berikut ini.



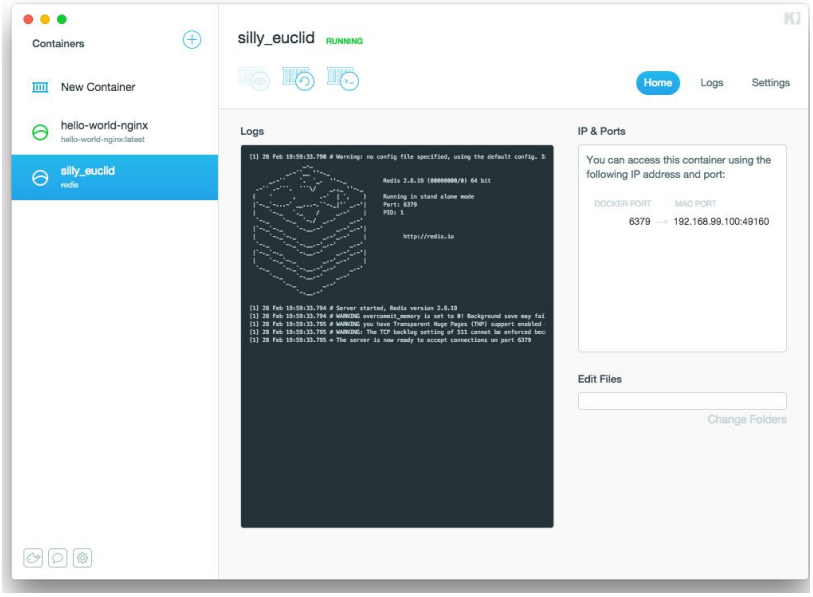
Gambar 4.6

Setelah docker console terbuka, maka anda dapat menggunakan semua perintah-perintah docker sebagaimana biasanya. Sebagai contoh ketika saya mengetikan perintah docker run -d -P redis, maka docker akan mem-pull image redis dan membuat container redis baru, dan ketika anda kembali kepada interface grafis Kitematic, anda mendapatkan satu container baru sudah terbuat dan berada pada list menu sebelah kiri anda.

Catatan : Jika anda membuat container baru, gunakan opsi `docker run -d`, container yang dijalankan tanpa opsi tersebut akan menyebabkan kegagalan.



Gambar 4.7



Gambar 4.8

Panamax

Panamax merupakan Open Source Project yang dibuat oleh Century Link yang dibuat untuk memudahkan pengembangan dan sharing docker container semudah drag-n-drop dengan tampilan GUI yang mudah digunakan.

Menjalankan banyak container yang kompleks, begitu juga multi server dengan container, memang tidak mudah dilakukan oleh pemula, tapi dengan Panamax, hal itu dapat dilakukan dengan sederana dan mudah.

Instalasi Panamax

Saat tulisan ini dibuat, Panamax masih berstatus Beta dan hanya mensupport sistem operasi Mac OS X dan Ubuntu.

Sebelum anda melakukan instalasi, Panamax membutuhkan beberapa hal yang harus terinstal, diantaranya :

- ✓ VirtualBox versi 4.2 atau lebih tinggi
- ✓ Vagrant versi 1.6 atau lebih tinggi

Pada prosesnya, Panamax instaler akan membuat sebuah VM pada VirtualBox yang bernama *panamax-vm* yang terbuat dari CoreOS.

A. Instalasi Panamax di Mac OS

Untuk melakukan instalasi pada Mac OS, anda dapat menggunakan Homebrew, kemudian lakukan langkah-langkah berikut ini :
Dalam terminal, lakukan perintah :

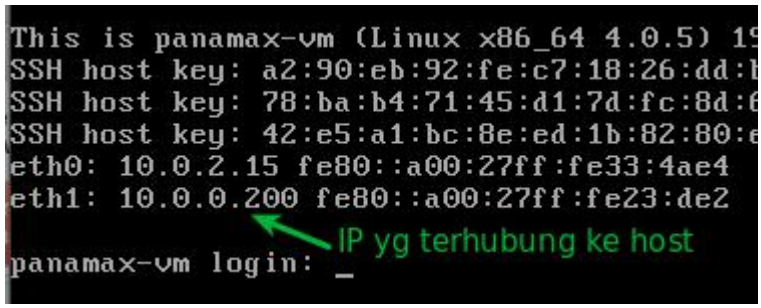
```
$ brew install \
http://download.panamax.io/installer/brew/panamax.rb
```

Setelah proses tersebut selesai, jalankan instalasi Panamax dengan perintah berikut :

```
$ panamax init
```

Setelah instalasi selesai, Panamax akan membuka web browser secara otomatis. Namun apabila web browser GUI dari Panamax tidak terbuka, anda dapat mengetikan alamat pada brower anda dengan http://ip_panamax:3000, dimana ip_panamax dapat anda lihat pada panamax-vm di VirtualBox.

Untuk lebih jelasnya, lihat gambar berikut ini.



Gambar 4.9

Gambar 4.10

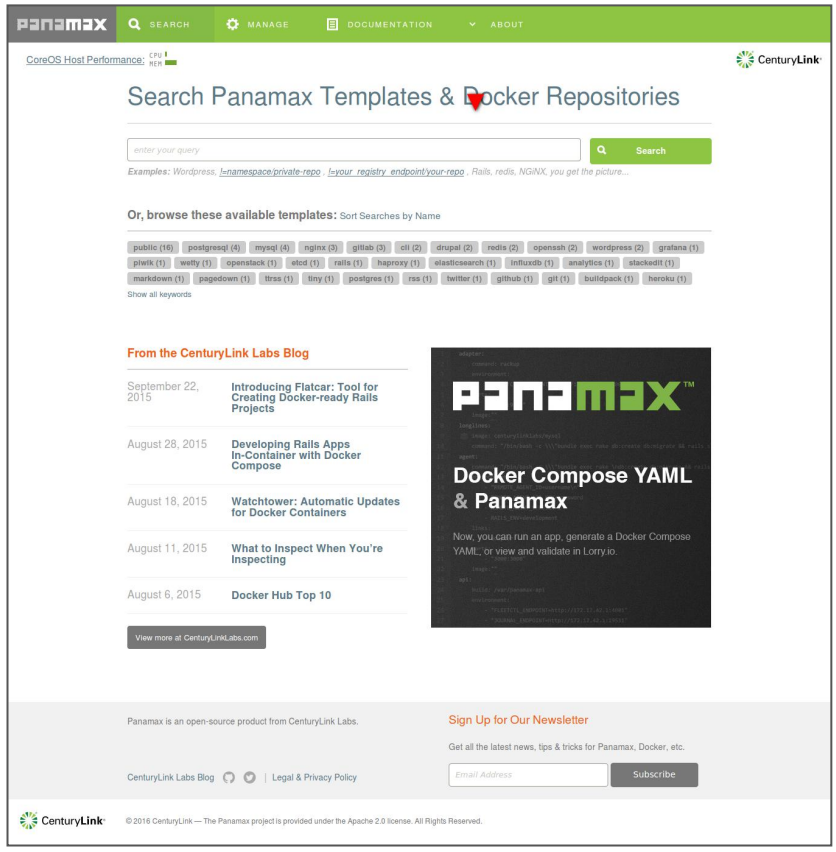
B. Instalasi Panamax di Ubuntu

Instalasi Panamax pada mesin Ubuntu tidak lebih sulit dari pada menginstalasi di Mac OS, developer Panamax telah menyediakan script yang dapat mengautomasi instalasi Panamax pada mesin Ubuntu anda yang dapat anda gunakan.

Untuk melakukan instalasi, buka terminal Ubuntu anda, kemudian ketikan perintah berikut ini :

```
$ curl \
http://download.panamax.io/installer/ubuntu.sh | bash
```

Biarkan proses instalasi berjalan hingga selesai, kemudian buka web browser anda, dan ketikan alamat ip_panamax untuk membuka interface GUI dari Panamax.



Gambar 4.11