

Pengenalan Container

Berbeda dengan VM(Virtual Machine), Container sendiri berfokus pada sisi Aplikasi. Container sendiri sebenarnya berjalan diatas aplikasi Container yang berjalan di sistem operasi. Untuk membedakan VM dengan Container adalah pada sisi **Container** kita bisa mempackage aplikasi dan depedency-nya tanpa harus menggabungkan sistem operasi. Container akan menggunakan sistem operasi host dimana Container Manager-nya berjalan,oleh karena itu, Container akan lebih hemat resource dan lebih cepat jalannya, karena tidak butuh sistem operasi sendiri. Ukuran Container terbilang sekitar MB-an , berbeda dengan VM yang bisa sampai GB karena didalamnya ada Sistem Operasi

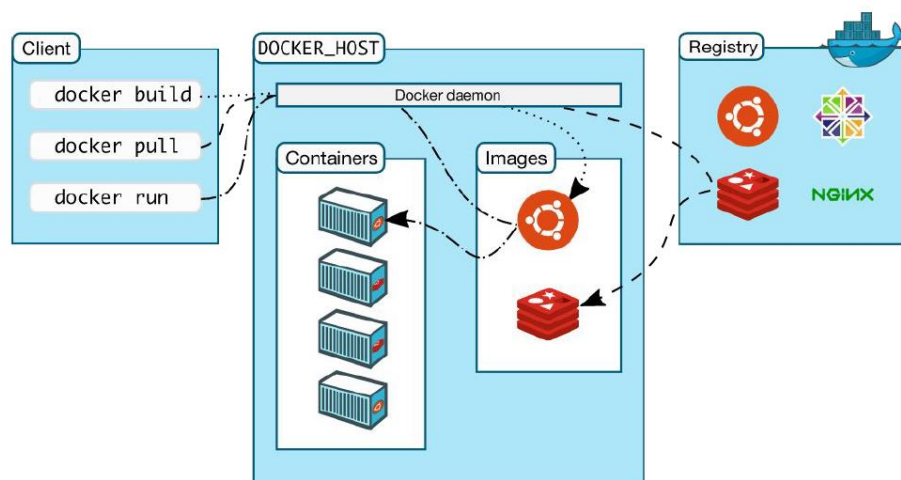
Pengenalan Docker

Docker adalah salah satu implementaasi Container Manager yang saat ini paling populer, Docker merupakan teknologi yang masih baru, karena baru diperkenalkan sekitar tahun 2013. Docker adalah aplikasi yang free dan Open Source, sehingga bisa kita gunakan secara bebas.

Diagram Docker Architecture

Docker menggunakan arstektur Client-Server, Docker Client berkomunikasi Docker Daemon (server). Saat kita menginstall Docker, Biasanya dididalamnya sudah terdapat Docker Client dan Docker Daemon:

1. Docker Client dan Docker Daemon bisa berjalan di satu sistem yang sama
2. Docker Client dan Docker Daemon berkominikasi mengguakan REST API(tata cara berkomunikasi)



1. Mengecek Docker, Untuk mengecek apakah Docker Daemon sudah Berjalan(Docker Server) versi Docker Client dan Servernya “sudo docker version”

```
root@anommudita:/home/anommudita# sudo docker version
Client: Docker Engine - Community
Version: 20.10.18
API version: 1.41
Go version: go1.18.6
Git commit: b40c2f6
Built: Thu Sep 8 23:11:45 2022
OS/Arch: linux/amd64
Context: default
Experimental: true

Server: Docker Engine - Community
Engine:
Version: 20.10.18
API version: 1.41 (minimum version 1.12)
Go version: go1.18.6
Git commit: e42327a
Built: Thu Sep 8 23:09:37 2022
OS/Arch: linux/amd64
Experimental: false
containerd:
Version: 1.6.8
GitCommit: 9cd3357b7fd7218e4aec3eae239db.f68a5a6ec6
runc:
Version: 1.1.4
GitCommit: v1.1.4-0-g5fd4c4d
docker-init:
Version: 0.19.0
GitCommit: de40ad0
root@anommudita:/home/anommudita#
```

2. **Docker Registry**, Docker Registry adalah tempat kita menyimpanan Docker Image dengan menggunakan Docker Registry, kita bisa menyimpanan Image yang kita buat, dan digunakan di Docker Daemon dimamanpun selama bisa terkoneksi ke Docker Registry. Contoh Docker Registry
 1. Docker Hub(free) : <https://hub.docker.com/>
 2. Digital Ocean Container Registry : <https://www.digitalocean.com/products/container-registry/>
 3. Google Cloud Container Registry : <https://cloud.google.com/container-registry>
 4. Azure Container Registry : <https://azure.microsoft.com/en-us/services/container-registry/>
3. Docker Image mirip seperti installer aplikasi, dimana di dalam Docker Image terdapat aplikasi dan dependency. Sebelum kita bisa menjalankan aplikasi di Docker, kita harus memastikan memiliki Docker Image aplikasi tersebut. Mengecek image docker “docker image ls” disini saya belum mendownload dependency atau image-nya.

```
root@anommudita:/home/anommudita# docker image ls
REPOSITORY TAG IMAGE ID CREATED SIZE
root@anommudita:/home/anommudita#
```

4. Mengunduh docker image “docker image pull namaimage:tag(version)”, saya disini mengunduh image redis

```
root@anommudita:/home/anommudita# docker image pull redis:latest
latest: Pulling from library/redis
bd159e379b3b: Downloading 21.92MB/31.42MB
729b630784ac: Download complete
065c77bf222a: Download complete
961784053f68: Downloading 7.904MB/9.561MB
b42f16846808: Download complete
0f1fa5bb0033: Download complete
-
```

5. Jika sudah terinstall coba cek docker imagesnya “docker image ls”

```
root@anommudita:/home/anommudita# docker image ls
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
redis         latest    f8528f17261c   4 days ago    117MB
root@anommudita:/home/anommudita# _
```

6. Perintah untuk menghapus docker images “docker image rm namaimage:tag”

```
root@anommudita:/home/anommudita# docker image rm redis:latest_
```

7. Docker Container

Jika Docker Image seperti installer aplikasi, maka Docker Container mirip seperti hasil installernya. Satu Docker Image bisa digunakan untuk membuat beberapa Docker Container, asalkan nama Docker Containernya berbeda. Jika kita sudah membuat Docker Container, maka Docker Image yang digunakan tidak bisa dihapus, hal ini dikarenakan sebenarnya Docker Container tidak meng-copy isi Docker Image, tapi hanya menggunakan isinya saja.

8. Status Container

Saat kita membuat Container, secara default Container tersebut kita tidak tidak berjalan. Mirip seperti ketika kita menginstall aplikasi, jika tidak dijalankan, maka aplikasi tersebut tidak akan berjalan, begitu juga Container, oleh karena itu, setelah membuat Container, kita perlu menjalankannya jika memang ingin menjalankan Containernya.

9. Melihat isi container , Container kosong karena belum membuat container. Untuk melihat semua Container, baik yang sedang berjalan atau tidak Di Docker Daemon, kita bisa gunakan perintah “**docker container ls -a**” -a artinya all, sedangkan jika ingin melihat container yang sedang berjalan saja bisa gunakan perintah “**docker container ls**”

```
root@anommudita:/home/anommudita# docker container ls
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS   NAMES
root@anommudita:/home/anommudita# _
```

10. Membuat container untuk redis

```
root@anommudita:/home/anommudita# docker container create --name redis1 redis:latest
fb19b16ca41e8443265db7834d0e58a49a74970b4d135f5a0fb81a3b360ee529
root@anommudita:/home/anommudita# _
```

11. Melihat container yang sudah dibuat tadi

```
root@anommudita:/home/anommudita# docker container ls -a
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS        NAMES
fb19b16ca41e   redis:latest "docker-entrypoint.s..." 2 minutes ago Created        redis1
root@anommudita:/home/anommudita#
```

12. Menjalankan Container dan mengecek container apakah sudah berjalan “docker start containerid /namacontainer”

```
root@anommudita:/home/anommudita# docker container start redis1
redis1
root@anommudita:/home/anommudita# docker container ls -a
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS        NAMES
fb19b16ca41e   redis:latest "docker-entrypoint.s..." 6 minutes ago Up 22 seconds 6379/tcp     redis1
root@anommudita:/home/anommudita#
```

13. Menghentikan Container dengan perintah “docker stop containerid/ namacontainerid”

```
root@anommudita:/home/anommudita# docker container stop redis2
redis2
root@anommudita:/home/anommudita# docker container ls
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS        NAMES
fb19b16ca41e   redis:latest "docker-entrypoint.s..." 18 minutes ago Up 12 minutes 6379/tcp     redis1
root@anommudita:/home/anommudita# docker container ls -a
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS        NAMES
cb7bb141a274   redis:latest "docker-entrypoint.s..." 7 minutes ago Exited (0) 2 minutes ago redis2
fb19b16ca41e   redis:latest "docker-entrypoint.s..." 20 minutes ago Up 14 minutes 6379/tcp     redis1
root@anommudita:/home/anommudita#
```

14. Untuk menghapus container kalian harus menghentikan terlebih dahulu containernya supaya container bisa terhapus dengan perintah

```
root@anommudita:/home/anommudita# docker container rm redis2
redis2
root@anommudita:/home/anommudita# docker container ls -a
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS        NAMES
fb19b16ca41e   redis:latest "docker-entrypoint.s..." 23 minutes ago Up 17 minutes 6379/tcp     redis1
root@anommudita:/home/anommudita# _
```

15. Container Log

Container Log adalah Aktivitas Containernya, terkadang saat terjadi masalah dengan aplikasi yang terdapat di Container, sering sekali kita ingin melihat detail dari log aplikasinya. Hal ini dilakukan untuk melihat detail kejadian apa yang terjadi di aplikasi, sehingga akan memudahkan kita ketika mendapat masalah.

16. Melihat log container , jika ingin melihat log cointainer secara realtime gunakan sintak “docker container logs -f namecontainer” sedangkan jika tidak secara realtime “docker container logs namecontainer”

```
root@anommudita:/home/anommudita# docker container logs redis1
1:C 09 Oct 2022 11:56:19.153 # o000o000o000o Redis is starting o000o000o000o
1:C 09 Oct 2022 11:56:19.154 # Redis version=7.0.5, bits=64, commit=00000000, modified=0, pid=1, just started
1:C 09 Oct 2022 11:56:19.154 # Warning: no config file specified, using the default config. In order to specify a config file use redis-server /path/to/redis.conf
1:M 09 Oct 2022 11:56:19.154 * monotonic clock: POSIX clock_gettime
1:M 09 Oct 2022 11:56:19.155 * Running mode=standalone, port=6379.
1:M 09 Oct 2022 11:56:19.156 # Server initialized
1:M 09 Oct 2022 11:56:19.156 # WARNING overcommit_memory is set to 0! Background save may fail under low memory condition. To fix this issue add 'vm.overcommit_memory = 1' to /etc/sysctl.conf and then reboot or run the command 'sysctl vm.overcommit_memory=1' for this to take effect.
1:M 09 Oct 2022 11:56:19.165 * Ready to accept connections
root@anommudita:/home/anommudita# _
```

17. Container Exec

Saat kita membuat container, aplikasi yang terdapat di dalam container hanya bisa diakses dari dalam container. Oleh karena itu, kadang kita perlu masuk ke dalam containernya itu sendiri , untuk masuk kedalam container, kita bisa menggunakan fitur Container Exec, dimana digunakan untuk mengeksekusi kode program yang terdapat didalam Continer

18. Untuk masuk ke Containter Exec, kita bisa mencoba mengeksekusi program bash script yang terdapat didalam container dengan bantuan Container Exec :

“docker container exec -i -t namacontainer/bin/bash” ➔ kita akan ngoding di folder bashnya karena disana pintu aplikasinya

-i adalah argument interaktif, menjaga input tetap aktif

-t adalah argument untuk lokasi pseudo-TTY(terminal akses) ➔ agar diberi akses lewat Terminal

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

```
root@anommudita:/home/anommudita# docker container exec -i -t redis1 /bin/bash
root@fb19b16ca41e:/data# redis-cli
127.0.0.1:6379> set hello "World"
OK
127.0.0.1:6379> get hello
"World"
127.0.0.1:6379> _
```

19. Container Port

Saat menjalankan Container, Container tersebut terisolasi di dalam Docker, artinya sistem Host(misal Laptop kita), tidak bisa mengakses aplikasi yang ada di dalam Container secara langsung, salah satu caranya adalah harus menggunakan Container Exec untuk ke dalam container nya(jka langsung di docker server). Biasanya, sebuah aplikasi berjalan pada port tertentu, misal saat kita menjalankan aplikasi redis, dia

berjalan pada 6379, kita melihat port apa saja digunakan ketika melihat semua daftar Container

20. Port Forwarding

Docker memiliki kemampuan untuk melakukan port forwarding, yaitu meneruskan sebuah port yang terdapat di sistem Hostnya ke dalam Docker Container. Cara ini cocok jika kita ingin mengekspos port yang terdapat di container ke luar melalui sistem Hostnya.

Melakukan Port Forwarding

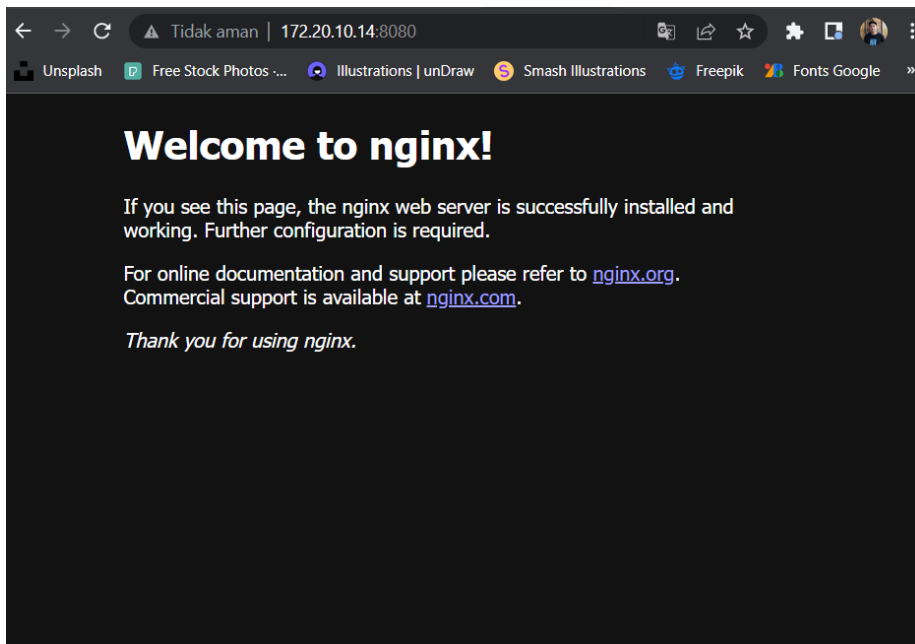
Untuk melakukan Port Forwarding, kita bisa menggunakan perintah berikut ketika membuat container nya: “docker container create – name namacontainer –publish porthost:portcontainer image:tag”.

Jika ingin melakukan Port Forwarding lebih dari satu, kita bisa tambahkan dua kali parameter langsung beri –publish publish juga bisa disingkat menggunakan -p

21. **Container Port dan Port Forwarding**, cara mengakses container dari luar(laptop kita) dengan port docker containernya yang disebut dengan Port Forwarding, yaitu meneruskan sebuah port yang terdapat di sistem Hostnya ke dalam Docker Container. Jika ingin membuat port forwarding pada container harus buat containernya dan diset portnya dan jangan lupa di jalankan containernya.

```
root@anommudita:/home/anommudita# docker container create --name nginx1 --publish 8080:80 nginx:latest
0ccb468f480df35f991b9def849dec23ed3f97837087b0e6f25bb50e5721b208
root@anommudita:/home/anommudita# docker container ls -a
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS
0ccb468f480d   nginx:latest  "/docker-entrypoint..."  2 minutes ago  Created
fb19b16ca41e   redis:latest  "docker-entrypoint.s..."  About an hour ago  Up About an hour  6379/tcp
root@anommudita:/home/anommudita# docker container start nginx1
nginx1
root@anommudita:/home/anommudita# _
```

Nginx berjalan di port 8080 dan bisa diakses dari luar



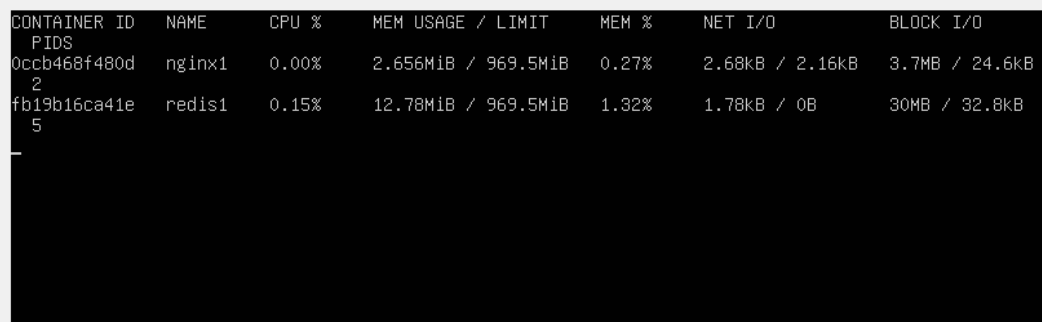
22. Container Environment Variable

Saat membuat aplikasi, menggunakan Environment Variable adalah salah satu Teknik agar konfigurasi aplikasi bisa diubah secara dinamis. Dengan menggunakan Environment Variable, kita bisa mengubah-ubah konfigurasi aplikasi tanpa harus mengubah kode aplikasinya lagi. Docker Container memiliki parameter yang bisa kita gunakan untuk mengirim Environment Variable ke aplikasi yang terdapat di dalam Container.

23. Menambahkan Container Environment Variable, kita bisa menggunakan perintah `--env` atau `-e` misalnya : `"docker container create --name namacontainer -env KEY="value" --env KEY2="value" image:tag"` . Disini sebagai contoh menggunakan MongoDB karena untuk menjalankan MongoDB harus menjalankan environment variable ➔ mengubah username dan passwordnya Environment Variablenya. Untuk testing kita buka dengan RoboMongo

```
root@anommudita:/home/anommudita# docker image ls
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
mongo         latest    1cca5cf68239   6 days ago    695MB
nginx         latest    51086ed63d8c   6 days ago    142MB
redis         latest    f8528f17261c   6 days ago    117MB
root@anommudita:/home/anommudita# docker container create --name contohmongo --publish 27017:27017 --env MONGO_INITDB_ROOT_USERNAME=anommudita --env MONGO_INITDB_ROOT_PASSWORD=12345 mongo:latest
93f9f22abc8c0a48ddb9ebc8951b6cb16a6cb3ea11b41bce3b12e9423a2b32cf
root@anommudita:/home/anommudita# _
```

24. Container Stats, untuk melihat penggunaan resource seperti CPU dan memory dari tiap containernya dengan command “**docker container stats**”



CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O
0ccb468f480d	nginx1	0.00%	2.656MiB / 969.5MiB	0.27%	2.68kB / 2.16kB	3.7MB / 24.6kB
fb19b16ca41e	redis1	0.15%	12.78MiB / 969.5MiB	1.32%	1.78kB / 0B	30MB / 32.8kB

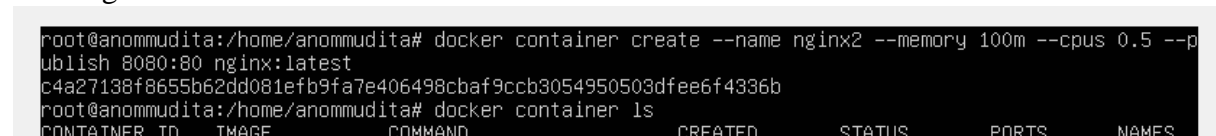
25. Container Resource Limit, membatasi penggunaan CPU dan Memory pada Containernya di Linux karena linux sudah jadi satu resource-nya dengan docker maka dari itu harus batasi. Dengan Perintah “**docker container create --name namecontainer --memory 100m --cpu 0.5 -publish 8080:80 nginx:latest**”

B = byte

K = kilo byte

M = Mega

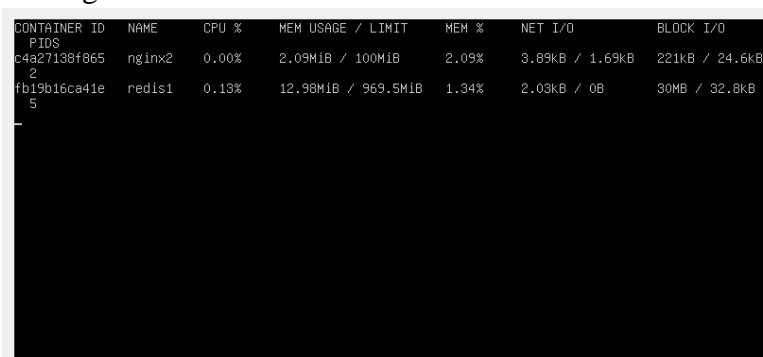
G = Giga



```
root@anommudita:/home/anommudita# docker container create --name nginx2 --memory 100m --cpus 0.5 --publish 8080:80 nginx:latest
c4a27138f865b62dd081efb9fa7e406498cbaf9ccb3054950503dfee6f4336b
root@anommudita:/home/anommudita# docker container ls
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
--------------	-------	---------	---------	--------	-------	-------

Check langsung resource apakah container yang kita buat sudah sesuai dengan kita setting atau tidak



CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O
c4a27138f865	nginx2	0.00%	2.09MiB / 100MiB	2.09%	3.89kB / 1.69kB	221kB / 24.6kB
fb19b16ca41e	redis1	0.13%	12.98MiB / 969.5MiB	1.34%	2.03kB / 0B	30MB / 32.8kB

26. Bind Mounts

Bind Mounts merupakan kemampuan melakukan mounting(sharing) file atau folder yang terdapat di sistem host ke container yang terdapat di docker(data dibackup lalu simpan dihost). Fitur ini sangat berguna ketika misal kita ingin mengirim konfigurasi dari luar Container, atau misal menyimpan data yang dibuat di aplikasi di dalam Container ke dalam folder di sistem host. Jika file atau folder tidak ada sistem host, secara otomatis akan dibuatkan oleh docker. Untuk melakukan mounting, kita bisa menggunakan parameter **--mount** ketika membuat container. Isi dari parameter **--mount** memiliki aturan tersendiri.

Parameter Mount

Parameter	Keterangan
type	Tipe mount, bind atau volume
source	Lokasi file atau folder di sistem host
destination	Lokasi file atau folder di container
readonly	Jika ada, maka file atau folder hanya bisa dibaca di container, tidak bisa ditulis

27. Melakukan Mounting(Pemasangan)

Untuk melakukan Mounting, kita bisa menggunakan perintah berikut:

docker container create -name namacontainer --mount “type=bind, source=folder, destination=folder, readonly” image:tag. Jangan lupa buat folder sumbernya(sebagai backupnya) dan destination ialah letak penyimpanan mongodb

```
root@anommudita:/home/anommudita# docker container create --name mongodata --publish 27018:27017 --mount "type=bind,source=/home/anommudita/mongo-data,destination=/data/db" --env MONGO_INITDB_ROOT_USERNAME=anommudita --env MONGO_INITDB_ROOT_PASSWORD=12345 mongo:latest
d4b3f903bc677a2d90b6bbcf92a5a73bffa61829e089354836e0f00672b7ba29
root@anommudita:/home/anommudita#
```

28. Docker Volume

Fitur Bind Mounts sudah ada sejak Docker versi awal, diversi terbaru direkomendasikan menggunakan Docker Volume. Docker Volume mirip dengan Bind Mounts, bedanya adalah terdapat management Volume, dimana kita bisa membuat Volume, melihat daftar Volume, menghapus Volume. Volume sendiri bisa dianggap Storage yang digunakan untuk menyimpan data, bedanya dengan Bind Mounts, pada bind mounts, data disimpan pada sistem host, sedangkan Volume data di manage oleh Docker.

29. Melihat Volume Docker

Saat kita membuat Container, dimanakah data di dalam Container itu disimpan, secara default semua data Container disimpan di dalam Volume. Jika kita coba melihat docker Volume, kita akan lihat bahwa ada banyak Volume yang sudah tersebut, walaupun kita belum pernah membuat sama sekali, kita bisa gunakan perintah berikut untuk melihat daftar volume : “**docker volume ls**” merupakan sharing file tetapi disimpan di sistem dockernya dan direkomendasikan. Kita bisa membuat, melihat, menghapus dockernya.

```
root@anommudita:/home/anommudita# docker volume ls
DRIVER      VOLUME NAME
local       a291f53fae335395b5a5128cbd0abad6b0d011a12a45f7412cb3681f6756b75f
local       b96e55bd176687b5f78d7bb08e45d6d044af01a310bacddd13349ae1d55b6ffc
root@anommudita:/home/anommudita# _
```

Membuat Volume Docker “docker volume create namavolume”

```
root@anommudita:/home/anommudita# docker volume create mongovolume
mongovolume
root@anommudita:/home/anommudita# _
```

Menghapus Volume “docker volume rm namavolume”, volume yang tidak digunakan oleh container bisa kita hapus, tapi jika volume digunakan oleh container, maka tidak bisa kita hapus sampai containernya dihapus

```
root@anommudita:/home/anommudita# docker volume rm mongovolume
mongovolume
root@anommudita:/home/anommudita# _
```

30. Container Volume

Volume yang sudah kita buat, bisa kita gunakan di container. Keuntungan menggunakan volume adalah jika container kita dihapus data akan tetap aman di volume. Cara menggunakan volume di container sama dengan menggunakan bind mount, kita bisa menggunakan parameter `--mount`, namun dengan menggunakan type volume dan source `namavolume` **“docker container create --name namacontainer --mount “type=mount, source=namavolume, destination=folder” --env MONGO_INITDB_ROOT_USERNAME=anommudita --env MONGO_INITDB_ROOT_PASSWORD=12345 image:tag”**

```
root@anommudita:/home/anommudita# docker container create --name mongovolume --publish 27019:27017 --mount "type=volume,source=mongovolume,destination=/data/db" --env MONGO_INITDB_ROOT_USERNAME=anommudita --env MONGO_INITDB_ROOT_PASSWORD=12345 mongo:latest
5d9241f169d11417156bdbd46f49e6ac31df3a751600999606fb2f3c687209a5
root@anommudita:/home/anommudita# docker volume ls
DRIVER      VOLUME NAME
local       9b6846624817c9960d4766ef48d023dea00d7fafd871aefbcef0e7fbf865867d
local       446253f8e9303c356b4155ecd612a7240bb207160a21333b58782a9515ad2ec1
local       a9b2ce90b0fca02ec3de504d61e2957d47c73e777171e7150410c4d46e5d2cbe
local       a291f53fae335395b5a5128cbd0abad6b0d011a12a45f7412cb3681f6756b75f
local       b96e55bd176687b5f78d7bb08e45d6d044af01a310bacddd13349ae1d55b6ffc
local       b49813e745ae08a74a7b98b65676050e7536582fd3f546871397ea0c36463826
local       mongovolume
local       mongovolumedata
root@anommudita:/home/anommudita#
```

31. Backup Volume, Sayangnya, sampai saat ini, tidak ada cara otomatis melakukan backup volume yang sudah kita buat karena volume di manage oleh docker, namun kita bisa memanfaatkan container untuk melakukan backup data yang ada di dalam volume ke dalam archive seperti zip atau tar.gz

Tahapan Melakukan Backup pada Volume :

- Matikan container yang menggunakan volume dan yang ingin kita backup

```
root@anommudita:/home/anommudita# docker container stop mongodata
mongodata
root@anommudita:/home/anommudita# _
```

- Buat Container baru dengan dua mount, volume yang ingin kita backup, dan bind mount folder dari sistem host

```
root@anommudita:/home/anommudita# docker container create --name nginxbackup --mount "type=bind,source=/home/anommudita/backup,destination=/backup" --mount "type=volume,source=mongodata,destination=/data" nginx:latest
2f2a6de4eb1bd890a9d6e315da84a194352a5f65c1d15680db2d702410ece7bc
root@anommudita:/home/anommudita#
```

- Lakukan backup menggunakan container dengan cara mengarchive isi volume, dan simpan di bind mount folder

```

root@anommudita:/home/anommudita# docker container start nginxbackup
nginxbackup
root@anommudita:/home/anommudita# docker container exec -i -t nginxbackup /bin/bash
root@2f2a6de4eb1b:/# ls -l
total 88
drwxr-xr-x  2 root root 4096 Oct 13 03:36 backup
drwxr-xr-x  2 root root 4096 Oct  4 00:00 bin
drwxr-xr-x  2 root root 4096 Sep  3 12:10 boot
drwxr-xr-x  2 root root 4096 Oct 13 03:43 data
drwxr-xr-x  5 root root 340 Oct 13 03:46 dev
drwxr-xr-x  1 root root 4096 Oct  5 12:44 docker-entrypoint.d
-rwxrwxr-x  1 root root 1202 Oct  5 12:44 docker-entrypoint.sh
drwxr-xr-x  1 root root 4096 Oct 13 03:43 etc
drwxr-xr-x  2 root root 4096 Sep  3 12:10 home
drwxr-xr-x  1 root root 4096 Oct  4 00:00 lib
drwxr-xr-x  2 root root 4096 Oct  4 00:00 lib64
drwxr-xr-x  2 root root 4096 Oct  4 00:00 media
drwxr-xr-x  2 root root 4096 Oct  4 00:00 mnt
drwxr-xr-x  2 root root 4096 Oct  4 00:00 opt
dr-xr-xr-x 185 root root  0 Oct 13 03:46 proc
drwx----- 2 root root 4096 Oct  4 00:00 root
drwxr-xr-x  1 root root 4096 Oct 13 03:46 run
drwxr-xr-x  2 root root 4096 Oct  4 00:00/sbin
drwxr-xr-x  2 root root 4096 Oct  4 00:00/srv
dr-xr-xr-x 13 root root  0 Oct 13 03:46 sys
drwxrwxrwt  1 root root 4096 Oct  5 12:44 tmp
drwxr-xr-x  1 root root 4096 Oct  4 00:00 usr
drwxr-xr-x  1 root root 4096 Oct  4 00:00 var
root@2f2a6de4eb1b:/#

```

- Isi file backup sekarang ada di folder sistem host ➔ lakukan seperti ini jika sudah masuk folder nginx

```

root@2f2a6de4eb1b:/# ls -l
total 88
drwxr-xr-x  2 root root 4096 Oct 13 03:52 backup
drwxr-xr-x  2 root root 4096 Oct  4 00:00 bin
drwxr-xr-x  2 root root 4096 Sep  3 12:10 boot
drwxr-xr-x  2 root root 4096 Oct 13 03:43 data
drwxr-xr-x  5 root root 340 Oct 13 03:46 dev
drwxr-xr-x  1 root root 4096 Oct  5 12:44 docker-entrypoint.d
-rwxrwxr-x  1 root root 1202 Oct  5 12:44 docker-entrypoint.sh
drwxr-xr-x  1 root root 4096 Oct 13 03:43 etc
drwxr-xr-x  2 root root 4096 Sep  3 12:10 home
drwxr-xr-x  1 root root 4096 Oct  4 00:00 lib
drwxr-xr-x  2 root root 4096 Oct  4 00:00 lib64
drwxr-xr-x  2 root root 4096 Oct  4 00:00 media
drwxr-xr-x  2 root root 4096 Oct  4 00:00 mnt
drwxr-xr-x  2 root root 4096 Oct  4 00:00 opt
dr-xr-xr-x 182 root root  0 Oct 13 03:46 proc
drwx----- 2 root root 4096 Oct  4 00:00 root
drwxr-xr-x  1 root root 4096 Oct 13 03:46 run
drwxr-xr-x  2 root root 4096 Oct  4 00:00/sbin
drwxr-xr-x  2 root root 4096 Oct  4 00:00/srv
dr-xr-xr-x 13 root root  0 Oct 13 03:46 sys
drwxrwxrwt  1 root root 4096 Oct  5 12:44 tmp
drwxr-xr-x  1 root root 4096 Oct  4 00:00 usr
drwxr-xr-x  1 root root 4096 Oct  4 00:00 var
root@2f2a6de4eb1b:/# cd /backup
root@2f2a6de4eb1b:/backup# tar cvf /backup/backup.tar.gz /data
tar: Removing leading '/' from member names
/data/
root@2f2a6de4eb1b:/backup# ls
backup.tar.gz
root@2f2a6de4eb1b:/backup#

```

- Delete Container yang kita gunakan untuk melakukan backup

```

root@anommudita:/home/anommudita# docker container stop nginxbackup
nginxbackup
root@anommudita:/home/anommudita# docker container rm nginxbackup
nginxbackup
root@anommudita:/home/anommudita# docker container start mongovolume
Command 'docker' not found, did you mean:
  command 'docker' from snap docker (20.10.14)
  command 'docker' from deb docker.io (20.10.12-0ubuntu4)
  command 'docker' from deb podman-docker (3:4.4-ds1-1ubuntu1)
See 'snap info <snapname>' for additional versions.
root@anommudita:/home/anommudita# docker container start mongovolume
mongovolume
root@anommudita:/home/anommudita# ls
backup_mongo-data mongovolume tftp
root@anommudita:/home/anommudita# cd /backup
bash: cd: /backup: No such file or directory
root@anommudita:/home/anommudita# cd backup
root@anommudita:/home/anommudita/backup# ls
backup.tar.gz
root@anommudita:/home/anommudita/backup#

```

32. Menjalankan Container secara langsung

Melakukan backup secara manual agak sedikit ribet karena kita harus start container terlebih dahulu(seperti diatas), setelah backup, hapus, containernya lagi. Kita bisa menggunakan perintah run untuk menjalankan perintah di Container dan gunakan

parameter `-rm` untuk melakukan otomatis remove Container setelah perintah selesai berjalan. `“docker container run -rm -name ubuntu -mount “type=bind,source=/home/anommudita/backup,destination=/backup” -mount “type=volume,source=mongodata,destination=/data” ubuntu:latest tar cvf /backup/backup.tar.gz /data”` → bisa menggunakan image ubuntu, kenapa tidak menggunakan nginx? Karena program Nginx itu programnya nyala terus. Kita butuh image yang memang sekali eksekusi langsung mati image-nya . Stop dulu containernya

Eksekusi Code diatas dan terakhir Start Containernya (lebih simple)

33. Restore Volume, setelah melakukan backup volume ke dalam file archive, kita bisa menyimpan file archive backup tersebut ke tempat yang lebih aman, misal ke cloud storage. Sekarang kita akan coba melakukan restore data backup ke volume baru, untuk memastikan data backup yang kita lakukan tidak corrupt

Tahapan Melakukan Restore

- Buat Volume baru untuk lokasi restore data backup

```
root@anommudita:/home/anommudita# docker volume create mongorestore
mongorestore
root@anommudita:/home/anommudita# _
```

- Buat Container baru dengan dua mount, Volume baru untuk restore backup, dan bin mount folder dari sistem host yang berisi file backup
 - Lakukan restore menggunakan Container dengan cara meng-extract isi backup file ke dalam volume
 - Isi file backup sekarang sudah direstore ke volume
 - Delete Container yang kita gunakan untuk melakukan restore
34. Docker Network, saat kita membuat container di docker, secara default container akan saling terisolasi satu sama lain, jadi kita mencoba memanggil antar container, bisa dipastikan bahwa kita tidak akan bisa melakukannya. Docker memiliki fitur Network yang bisa digunakan untuk membuat jaringan di dalam Docker, dengan menggunakan Network, kita bisa mengkoneksikan container dengan container lain dalam satu Network yang sama. Jika beberapa container terdapat pada satu Network yang sama, maka secara otomatis container tersebut bisa saling berkomunikasi

Network Driver

- Saat kita membuat Network di Docker, kita perlu menentukan driver yang ingin kita gunakan, ada banyak driver yang bisa kita gunakan, tapi kadang ada syarat sebuah driver network bisa gunakan :
 1. Bridge, yaitu driver yang digunakan untuk membuat network secara virtual yang memungkinkan container yang terkoneksi di bridge network yang sama saling berkomunikasi
 2. Host, yaitu driver yang digunakan untuk membuat network yang sama dengan sistem host. Host hanya dijalankan di Docker Linux, tidak bisa digunakan di Mac atau Windows
 3. None, yaitu driver untuk membuat network yang tidak bisa saling berkomunikasi

Melihat Docker Network “docker network ls”

```
root@anommudita:/home/anommudita# docker network ls
NETWORK ID      NAME      DRIVER      SCOPE
9826251b9ff     bridge   bridge      local
6e09f3c1d46b     host     host        local
22fcd1a8c631     none     null        local
root@anommudita:/home/anommudita#
```

Membuat Docker Network “docker network create --driver namedriver namenetwork”

```
root@anommudita:/home/anommudita# docker network create --driver bridge network1
63e60c4d27122d00910e0d896793e103de3efb297f7824e253a2109453d4d63a
root@anommudita:/home/anommudita# docker network ls
NETWORK ID      NAME      DRIVER      SCOPE
9826251b9ff     bridge   bridge      local
6e09f3c1d46b     host     host        local
63e60c4d2712     network1 bridge      local
22fcd1a8c631     none     null        local
root@anommudita:/home/anommudita#
```

Menghapus Network “docker network rm namenetwork”, Network tidak bisa dihapus jika sudah digunakan oleh container. Kita harus menghapus containernya terlebih dahulu dari Networknya

```
root@anommudita:/home/anommudita# docker network rm network1
network1
root@anommudita:/home/anommudita# docker network ls
NETWORK ID      NAME      DRIVER      SCOPE
9826251b9ff     bridge   bridge      local
6e09f3c1d46b     host     host        local
22fcd1a8c631     none     null        local
root@anommudita:/home/anommudita#
```

35. Container Network

Setelah kita membuat Network, kita bisa menambahkan container ke network. Container yang terdapat di dalam network yang sama bisa saling berkomunikasi (tergantung jenis driver networknya). Container bisa mengakses Container lain dengan menyebutkan hostname dari Containernya, yaitu nama containernya. Dipraticum kali ini akan menggunakan 2 container yaitu mongoDB dan mongoExpress (sebuah web untuk melihat data dari mongoDB tersebut)

Membuat Container dengan Network

Untuk menambahkan Container ke Network, kita bisa menambahkan perintah **–network** ketika membuat container “**docker container create –name namacontainer –network namanetwork image:tag**”

Tahapan Container Network

1. Membuat Networknya terlebih dahulu

```
root@anommudita:/home/anommudita# docker network create --driver bridge mongonetwork
a566aa29c58d1e9e82512edaa041d69d9b8c46f9390826945d1aebfb7da7d2db
root@anommudita:/home/anommudita# _
```

2. Membuat Container MongoDB dan Mongo Express

MongoDB

```
root@anommudita:/home/anommudita# docker container create --name mongodb --network mongonetwork --env MONGO_INITDB_ROOT_USERNAME=anommudita --env MONGO_INITDB_ROOT_PASSWORD=12345 mongo:latest
0bd9c865464dc06ab9d0b90de6c21e3af08b1d7e0b504d7489b31c5628a80c0f
root@anommudita:/home/anommudita#
```

MongoExpress

```
root@anommudita:/home/anommudita# docker container create --name mongodbexpress1 --network mongonetwork --publish 8081:8081 --env ME_CONFIG_MONGODB_URL="mongodb://anommudita:12345@mongodb:27017/" mongo-express:latest
cd187e618e5807b045f9f59f77315a65e4723214e892ab7b5fb2a04eb1cf2052
root@anommudita:/home/anommudita#
```

3. Jalankan Container MongoDB dan MongoExpress

```
root@anommudita:/home/anommudita# docker container start mongodb
mongodb
root@anommudita:/home/anommudita# docker container start mongodbexpress
mongodbexpress
root@anommudita:/home/anommudita#
```

Menghapus Container dari Network

Jika diperlukan, kita juga bisa menghapus container dari network dengan perintah “**docker network disconnect namanetwork namacontainer**”

```
root@anommudita:/home/anommudita# docker network disconnect mongonetwork mongodb
root@anommudita:/home/anommudita# _
```

Menambahkan Container dari Network

Jika Containernya sudah terlanjur dibuat, kita juga menambahkan Container yang sudah dibuat ke network dengan perintah “**docker network connect namanetwork namacontainer**”

```
root@anommudita:/home/anommudita# docker network connect mongonetwork mongodb
root@anommudita:/home/anommudita# _
```

36. Inspect,

setelah kita mendownload image, atau membuat network, volume dan container. Kadang kita ingin melihat detail dari tiap hal tersebut. Misal kita ingin melihat detail dari image. Docker memiliki fitur bernama inspect, yang bisa digunakan di image, container, volume dan network, dengan fitur ini, kita bisa melihat detail dari tiap hal yang ada di Docker.

Menggunakan Inspect

- Untuk melihat detail dari **image**, gunakan perintah “**docker image inspect namaimage**”
- Untuk melihat detail dari **container**, gunakan perintah “**docker container inspect namacontainer**”
- Untuk melihat detail dari **volume**, gunakan perintah “**docker volume inspect namavolume**”
- Untuk melihat detail dari **network**, gunakan perintah “**docker network inspect namanetwork**”

Inspect Docker Image

```
root@anommudita:/home/anommudita# docker image ls
REPOSITORY TAG IMAGE ID CREATED SIZE
mongo latest 1cca5cf68239 3 days ago 695MB
nginx latest 51086ed63d8c 4 days ago 142MB
redis latest f8528f17261c 4 days ago 117MB
root@anommudita:/home/anommudita# docker image inspect nginx
```


Hasilnya akan terlihat code pemrograman

```
"Os": "linux",
"Size": 141677874,
"VirtualSize": 141677874,
"GraphDriver": {
  "Data": {
    "LowerDir": "/var/lib/docker/overlay2/37a807b95240b5c4d9d8e752b496e9bb78445fe80873bae3894d56f8f8c11777/diff:/var/lib/docker/overlay2/91e2c5bdc8832017d484711336bc3d4088bf8544edd2594fe483d65ebb4c3828/diff:/var/lib/docker/overlay2/2cb035fed1fb30838026509f91ef8a6f903a9bb8b122a72b47055da0e3730c81/diff:/var/lib/docker/overlay2/3074b96eedc4a9c9c8242cc8c45c853dd8d150d6ac8aedf92f3e49ff4f571293/diff:/var/lib/docker/overlay2/f4af987270261109a24b369343fe3cf3a1c444dec6a073dcc73e3a50563b8a07/diff",
    "MergedDir": "/var/lib/docker/overlay2/0a1b658187cdf90fcf53b80865e886e896ddf05a5a4678fea8437e4d161a5e7/merged",
    "UpperDir": "/var/lib/docker/overlay2/0a1b658187cdf90fcf53b80865e886e896ddf05a5a4678fea8437e4d161a5e7/diff",
    "WorkDir": "/var/lib/docker/overlay2/0a1b658187cdf90fcf53b80865e886e896ddf05a5a4678fea8437e4d161a5e7/work"
  },
  "Name": "overlay2"
},
"RootFS": {
  "Type": "layers",
  "Layers": [
    "sha256:fe7b1e9bf7922fbc22281bcc6b4f5ac8f1a7b4278929880940978c42fc9d0229",
    "sha256:58a06a0d345c56af21bae2c06c4072837a88f9b5c89bb903515597657b9bc2c9",
    "sha256:6f4f3ce1dca0771aac350d475b68834c56223f4e710fb9062870f1fd49c79e00",
    "sha256:5eda6fa69be4649906647545d309718863ae2c7d71f9c4a7979f6301f2ca2200",
    "sha256:819eb3a4563250968960f3e06b007b8695c674c58d985fb6a709018b8e36361e",
    "sha256:d6a3537fc36ad52422aa5e4451395644ad0b25c5669b1ec8f72a9e554c16f470"
  ]
},
"Metadata": {
  "LastTagTime": "0001-01-01T00:00:00Z"
}
}
```

37. Prune

Saat kita menggunakan docker, kadang ada kalanya kita ingin membersihkan hal-hal sudah tidak digunakan lagi di Docker, misal Container yang sudah distop, image yang tidak digunakan oleh container, atau volume yang tidak digunakan oleh container. Fitur untuk membersihkan secara otomatis di Docker bernama Prune, Hampir di semua perintah di Docker mendukung Prune.

Perintah Prune

- Untuk menghapus semua container yang sudah stop, gunakan : docker container prune
- Untuk menghapus semua image yang tidak digunakan container, gunakan : docker image prune
- Untuk menghapus semua network yang tidak digunakan container, gunakan : docker network prune
- Untuk menghapus semua volume yang tidak digunakan container, gunakan : docker volume prune
- Atau kita bisa menggunakan satu perintah untuk menghapus container, network dan image yang sudah tidak digunakan menggunakan perintah : docker system prune

Seperti gambar dibawah ini, saya menghapus image yang tidak digunakan oleh container hasil 0 karena masing-masing image sudah digunakan oleh container.

```
root@anommudita:/home/anommudita# docker image prune
WARNING! This will remove all dangling images.
Are you sure you want to continue? [y/N] Y
Total reclaimed space: 0B
root@anommudita:/home/anommudita# _
```

Materi Selanjutnya

Docker Dockerfile → membuat image baru, cocok banget untuk developer dan untuk DevOps hanya menggunakan saja

Docker Compose → Fitur Compose, mengatur container, volume, network menggunakan konfigurasi dan tidak perlu lagi satu-satu mengkonfigurasi menggunakan CLI dan bisa sekali setting saja

Referensi : https://youtu.be/3_yxVjV88Zk