



# Docker Dasar

Eko Kurniawan Khannedy

# Eko Kurniawan Khannedy

- Technical architect at one of the biggest ecommerce company in Indonesia
- 10+ years experiences
- [www.programmerzamannow.com](http://www.programmerzamannow.com)
- [youtube.com/c/ProgrammerZamanNow](https://youtube.com/c/ProgrammerZamanNow)





# Eko Kurniawan Khannedy

- Telegram : [@khannedy](https://t.me/khannedy)
- Facebook : [fb.com/ProgrammerZamanNow](https://fb.com/ProgrammerZamanNow)
- Instagram : [instagram.com/programmerzamannow](https://instagram.com/programmerzamannow)
- Youtube : [youtube.com/c/ProgrammerZamanNow](https://youtube.com/c/ProgrammerZamanNow)
- Telegram Channel : [t.me/ProgrammerZamanNow](https://t.me/ProgrammerZamanNow)
- Email : [echo.khannedy@gmail.com](mailto:echo.khannedy@gmail.com)



# Sebelum Belajar

- Mengerti tentang sistem operasi
- Mengerti cara menginstall aplikasi
- Mengerti cara menggunakan perintah di terminal / command line
- Mengerti tentang Virtual Machine



# Agenda

- Pengenalan Container
- Pengenalan Docker
- Menginstall Docker
- Arsitektur Docker
- Docker Image
- Docker Registry
- Docker Container
- Docker Volume
- Docker Network
- Dan lain-lain

---

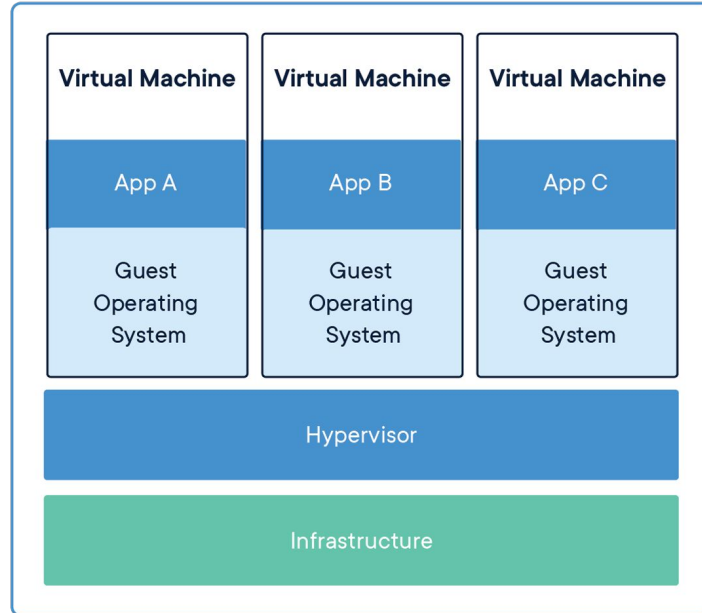
# Pengenalan Container



# Virtual Machine

- Dalam dunia Infrastructure, kita sudah terbiasa dengan yang namanya VM (Virtual Machine)
- Saat membuat sebuah VM, biasanya kita akan menginstall sistem operasi juga di VM nya
- Masalah ketika kita menggunakan VM adalah proses yang lambat ketika pembuatan VM nya, dan butuh waktu untuk boot sistem operasi di dalam VM tersebut ketika kita menjalankan VM atau me-restart VM tersebut

# Diagram Virtual Machine



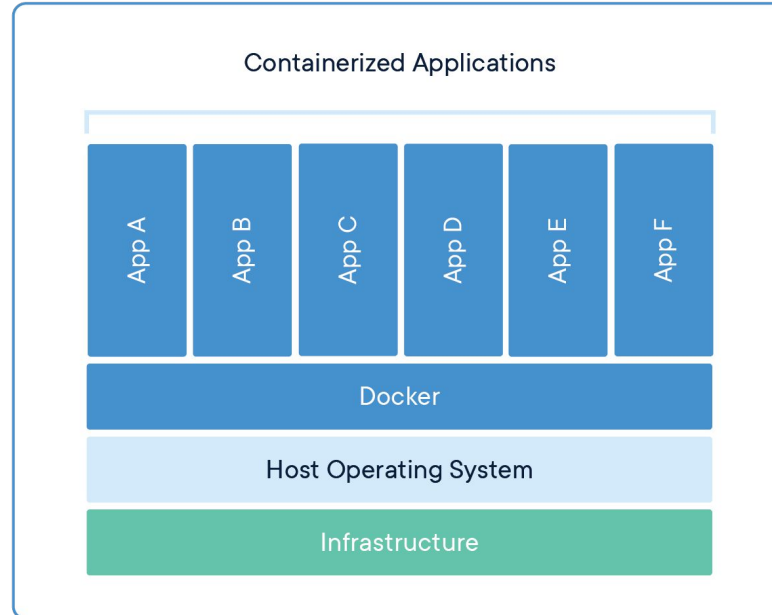




# Container

- Berbeda dengan VM, Container sendiri berfokus pada sisi Aplikasi
- Container sendiri sebenarnya berjalan diatas aplikasi Container Manager yang berjalan di sistem operasi.
- Yang membedakan dengan VM adalah, pada Container, kita bisa mem-package aplikasi dan dependency-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 jalan nya, karena tidak butuh sistem operasi sendiri
- Ukuran Container biasanya hanya hitungan MB, berbeda dengan VM yang bisa sampai GB karena di dalamnya ada sistem operasinya

# Diagram Container



---

# Pengenalan Docker



# Pengenalan Docker

- Docker adalah salah satu implementasi 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
- <https://www.docker.com/>

---

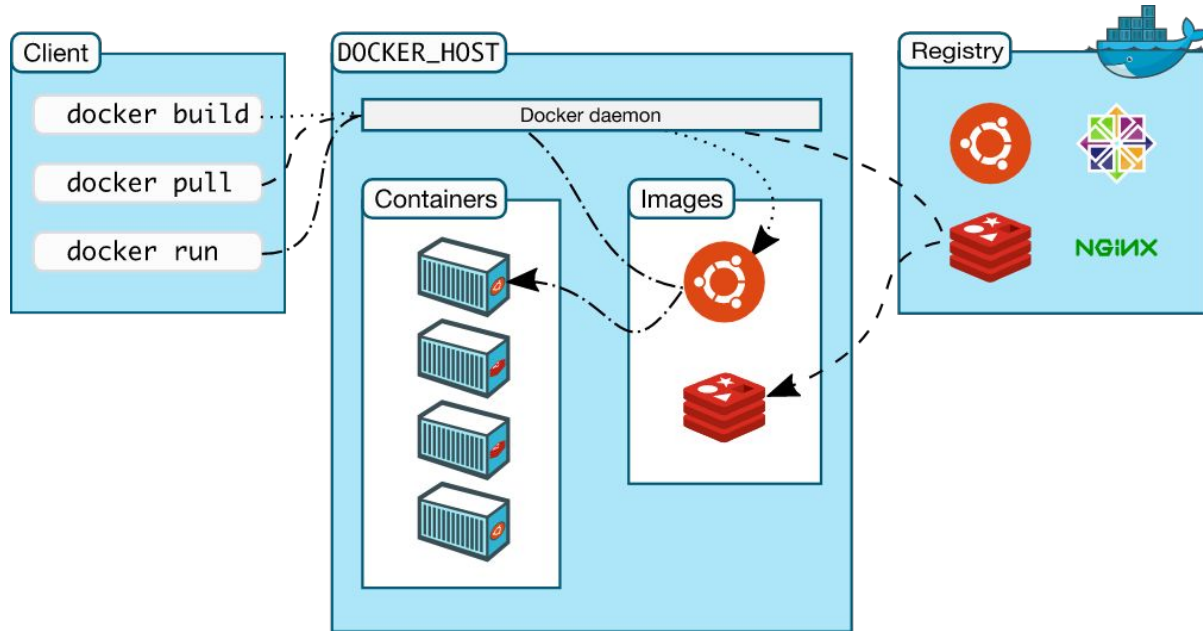
# Arsitektur Docker



# Docker Architecture

- Docker menggunakan arsitektur Client-Server
- Docker client berkomunikasi dengan Docker daemon (server)
- Saat kita menginstall Docker, biasanya didalamnya sudah terdapat Docker Client dan Docker Daemon
- Docker Client dan Docker Daemon bisa berjalan di satu sistem yang sama
- Docker Client dan Docker Daemon berkomunikasi menggunakan REST API

# Diagram Docker Architecture



---

# Menginstall Docker





# Menginstall Docker

- Docker bisa di install hampir disemua sistem operasi
- Untuk menginstall di Windows dan Mac, kita bisa menggunakan Docker Desktop
- <https://docs.docker.com/get-docker/>
- Untuk Linux, kita bisa install dari repository sesuai distro linux masing-masing
- <https://docs.docker.com/engine/install/>



# Mengecek Docker

- Untuk mengecek apakah Docker Daemon sudah berjalan, kita bisa gunakan perintah :  
docker version

---

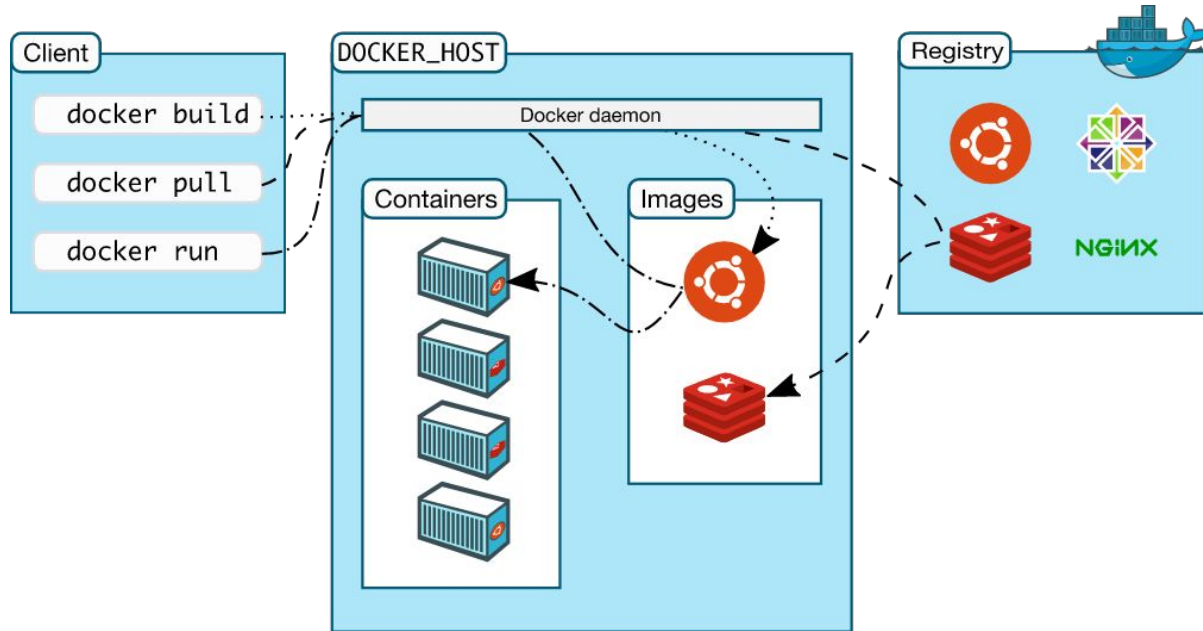
# Docker Registry



# Docker Registry

- Docker Registry adalah tempat kita menyimpan Docker Image
- Dengan menggunakan Docker Registry, kita bisa menyimpan Image yang kita buat, dan bisa digunakan di Docker Daemon dimanapun selama bisa terkoneksi ke Docker Registry

# Diagram Docker Registry





# Contoh Docker Registry

- Docker Hub : <https://hub.docker.com/>
- Digital Ocean Container Registry : <https://www.digitalocean.com/products/container-registry/>
- Google Cloud Container Registry : <https://cloud.google.com/container-registry>
- Amazon Elastic Container Registry : <https://aws.amazon.com/id/ecr/>
- Azure Container Registry : <https://azure.microsoft.com/en-us/services/container-registry/>

---

# Docker Image



# Docker Image

- Docker Image mirip seperti installer aplikasi, dimana di dalam Docker Image terdapat aplikasi dan dependency
- Sebelum kita bisa menjalankan aplikasi di Docker, kita perlu memastikan memiliki Docker Image aplikasi tersebut





# Melihat Docker Image

- Untuk melihat Docker Image yang terdapat di dalam Docker Daemon, kita bisa menggunakan perintah :  
`docker image ls`



## Kode : Melihat Docker Image

```
→ ~ docker image ls
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
redis	latest	aea9b698d7d1	7 days ago	113MB

```
→ ~ █
```



# Download Docker Image

- Untuk download Docker Image dari Docker Registry, kita bisa gunakan perintah :  
docker image pull namaimage:tag
- Kita bisa mencari Docker Image yang ingin kita download di <https://hub.docker.com/>



## Kode : Download Docker Image

```
→ ~ docker image pull redis:latest
latest: Pulling from library/redis
Digest: sha256:2f502d27c3e9b54295f1c591b3970340d02f8a5824402c817
Status: Image is up to date for redis:latest
docker.io/library/redis:latest
→ ~ █
```



# Menghapus Docker Image

- Jika kita tidak ingin menggunakan Docker Image yang sudah kita download, kita bisa gunakan perintah :  
`docker image rm namaimage:tag`



## Kode : Menghapus Docker Image

```
→ ~ docker image rm redis:latest
Untagged: redis:latest
Untagged: redis@sha256:2f502d27c3e9b54295f1c591b3970340d02f8a5824402c8179dcd20d407
Deleted: sha256:aea9b698d7d1d2fb22fe74868e27e767334b2cc629a8c6f9db8cc1747ba299fd
Deleted: sha256:beb6c508926e807f60b6a3816068ee3e2cece7654abaff731e4a26bcfebe04d8
Deleted: sha256:a5b5ed3d7c997ffd7c58cd52569d8095a7a3729412746569cdbda0dfdd228d1f
Deleted: sha256:ee76d3703ec1ab8abc11858117233a3ac8c7c5e37682f21a0c298ad0dc09a9fe
Deleted: sha256:60abc26bc7704070b2977b748ac0fd4ca94b818ed4ba1ef59ca8803e95920161
Deleted: sha256:6a2f1dcfa7455f60a810bb7c4786d62029348f64c4fcff81c48f8625cf0d995a
Deleted: sha256:9321ff862abbe8e1532076e5fdc932371eff562334ac86984a836d77dfb717f5
```

---

# Docker Container



# Docker Container

- Jika Docker Image seperti installer aplikasi, maka Docker Container mirip seperti aplikasi hasil installernya
- Satu Docker Image bisa digunakan untuk membuat beberapa Docker Container, asalkan nama Docker Container nya 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 menggunakannya isinya saja





# Status Container

- Saat kita membuat container, secara default container tersebut tidak akan berjalan
- Mirip seperti ketika kita menginstall aplikasi, jika tidak kita jalankan, maka aplikasi tersebut tidak akan berjalan, begitu juga container
- Oleh karena itu, setelah membuat container, kita perlu menjalankannya jika memang ingin menjalankan container nya



# Melihat Container

- Untuk melihat semua container, baik yang sedang berjalan atau tidak di Docker Daemon, kita bisa gunakan perintah :  
docker container ls -a
- Sedangkan jika kita ingin melihat container yang sedang berjalan saja, kita bisa gunakan perintah :  
docker container ls



## Kode : Melihat Docker Container

```
→ ~ docker container ls -a
```

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

```
→ ~ █
```



# Membuat Container

- Untuk membuat container, kita bisa gunakan perintah :  
`docker container create --name namacontainer namaimage:tag`



## Kode : Membuat Container

```
→ ~ docker container create --name contohredis redis:latest  
079f1d6245adfb159a7848e480b608d060c36899c4b59dcf0217dfcdeac87ff3
```

```
→ ~ docker container ls -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
079f1d6245ad	redis:latest	"docker-entrypoint.s..."	13 seconds ago	Created

```
→ ~ █
```



# Menjalankan Container

- Untuk menjalankan container yang sudah kita buat, kita bisa gunakan perintah :  
`docker container start containerId/namacontainer`



## Kode : Menjalankan Container

```
079f1d6245ad redis:latest docker-entrypoint.s... 44 seconds ago created
→ ~ docker container start contohredis
contohredis
→ ~ docker container ls -a
CONTAINER ID   IMAGE             COMMAND                  CREATED         STATUS
079f1d6245ad   redis:latest      "docker-entrypoint.s..." 44 seconds ago  Up 11 seconds
→ ~
```



# Menghentikan Container

- Untuk menghentikan container, kita bisa gunakan perintah :  
`docker container stop containerId/namacontainer`





## Kode : Menghentikan Container

```
079f1d6245ad redis:latest docker-entrypoint.s... 44 seconds ago Up 11 seconds 079f1d6245ad
→ ~ docker container stop contohredis
contohredis
→ ~ docker container ls -a
CONTAINER ID   IMAGE          COMMAND                  CREATED         STATUS
079f1d6245ad   redis:latest   "docker-entrypoint.s..." About a minute ago Exited (0) 1 se
→ ~ █
```



# Menghapus Container

- Untuk menghapus container yang sudah berhenti, kita bisa gunakan perintah :  
`docker container rm containerId/namacontainer`



## Kode : Menghapus Container

```
→ ~ docker container rm contohredis  
contohredis
```

```
→ ~ docker container ls -a
```

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

```
→ ~ █
```

---

# Container Log



# Container Log

- Kadang saat terjadi masalah dengan aplikasi yang terdapat di container, sering kali 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



# Melihat Container Log

- Untuk melihat log aplikasi di container kita, kita bisa menggunakan perintah :  
`docker container logs containerId/namacontainer`
- Atau jika ingin melihat log secara realtime, kita bisa gunakan perintah :
- `docker container logs -f containerId/namacontainer`



## Kode : Melihat Container Log

```
→ ~ docker container logs contohredis
1:C 11 Dec 2021 02:59:45.798 # o000o000o000o Redis is starting o000o000o000o
1:C 11 Dec 2021 02:59:45.798 # Redis version=6.2.6, bits=64, commit=00000000, m
1:C 11 Dec 2021 02:59:45.798 # Warning: no config file specified, using the def
1:M 11 Dec 2021 02:59:45.799 * monotonic clock: POSIX clock_gettime
1:M 11 Dec 2021 02:59:45.799 * Running mode=standalone, port=6379.
1:M 11 Dec 2021 02:59:45.799 # Server initialized
1:M 11 Dec 2021 02:59:45.800 * Ready to accept connections
→ ~
```

---

# Container Exec





# 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 container nya itu sendiri
- Untuk masuk ke dalam container, kita bisa menggunakan fitur Container Exec, dimana digunakan untuk mengeksekusi kode program yang terdapat di dalam container



# Masuk ke Container

- Untuk masuk ke dalam container, kita bisa mencoba mengeksekusi program bash script yang terdapat di dalam container dengan bantuan Container Exec :  
`docker container exec -i -t containerId/namacontainer /bin/bash`
- -i adalah argument interaktif, menjaga input tetap aktif
- -t adalah argument untuk alokasi pseudo-TTY (terminal akses)
- Dan /bin/bash contoh kode program yang terdapat di dalam container



## Kode : Container Exec

```
1.4 11 Dec 2021 02:55:15.000 - ready to accept connections  
→ ~ docker container exec -i -t contohredis /bin/bash  
root@e6699fd1007c:/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> █
```

---

# Container Port



# 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 masuk ke dalam container nya.
- Biasanya, sebuah aplikasi berjalan pada port tertentu, misal saat kita menjalankan aplikasi Redis, dia berjalan pada port 6379, kita bisa melihat port apa yang digunakan ketika melihat semua daftar container



# Port Forwarding

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



# Melakukan Port Forwarding

- Untuk melakukan port forwarding, kita bisa menggunakan perintah berikut ketika membuat container nya :  
`docker container create --name namacontainer --publish posthost:portcontainer image:tag`
- Jika kita ingin melakukan port forwarding lebih dari satu, kita bisa tambahkan dua kali parameter `--publish`
- `--publish` juga bisa disingkat menggunakan `-p`



## Kode : Melakukan Port Forwarding

```
→ ~ docker container create --name contohnginx --publish 8080:80 nginx:latest
ca6b61b5c67039d4abd25d2103153b226d29e5e18c93f3ba038fee4821b78a1d
→ ~ docker container start contohnginx
contohnginx
→ ~ docker container ls
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
ca6b61b5c670	nginx:latest	"/docker-entrypoint..."	15 seconds ago	Up 5 seconds	0.0.0.0:8080->80/tcp
e6699fd1007c	redis:latest	"docker-entrypoint.s..."	23 minutes ago	Up 23 minutes	6379/tcp

```
→ ~ █
```



---

# Container Environment Variable



# 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



# Menambah Environment Variable

- Untuk menambah environment variable, kita bisa menggunakan perintah `--env` atau `-e`, misal :  
`docker container create --name namacontainer --env KEY="value" --env KEY2="value" image:tag`

## Kode : Menambah Environment Variable

```
→ ~ docker container create --name contohmongo --publish 27017:27017 --env MONGO_INITDB_ROOT_USERNAME=eko --env MONGO_INITDB_ROOT_PASSWORD=eko mongo:latest
f5cb1faf6df17920423d254673c7cb0ed65e7be35c8a54679c8356c9d07a6715
→ ~ docker container ls -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
f5cb1faf6df1	mongo:latest	"docker-entrypoint.s..."	6 seconds ago	Created	
ca6b61b5c670	nginx:latest	"/docker-entrypoint..."	About an hour ago	Up About an hour	0.0.0.0:8080->80/tcp
e6699fd1007c	redis:latest	"docker-entrypoint.s..."	2 hours ago	Up 2 hours	6379/tcp

```
→ ~ █
```

---

# Container Stats



# Container Stats

- Saat menjalankan beberapa container, di sistem Host, penggunaan resource seperti CPU dan Memory hanya terlihat digunakan oleh Docker saja
- Kadang kita ingin melihat detail dari penggunaan resource untuk tiap container nya
- Untungnya docker memiliki kemampuan untuk melihat penggunaan resource dari tiap container yang sedang berjalan
- Kita bisa gunakan perintah :  
docker container stats

# System Monitoring Mac

Activity Monitor									
My Processes									
CPU Memory Energy Disk Network									
>> 🔍 docker									
Process Name	% CPU	CPU Time	Threads	Idle Wake Ups	% GPU	GPU Time	PID	Us	
Docker	0,0	8,72	10	0	0,0	0,00	64778	khanne	
com.docker.dev-envs	0,0	0,75	18	0	0,0	0,00	64783	khanne	
docker	0,0	2,36	17	2	0,0	0,00	64806	khanne	
com.docker.driver.amd64-lin...	0,0	2,98	19	4	0,0	0,00	64808	khanne	
com.docker.vpnkit	0,0	28,49	5	2	0,0	0,00	64805	khanne	
com.docker.backend	0,0	6,41	25	5	0,0	0,00	64782	khanne	
com.docker.hyperkit	12,5	21:06,00	15	385	0,0	0,00	64811	khanne	



# Container Stats

⏏ 🔍 2

## docker container stats

CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O	PIDS
f5cb1faf6df1	contohmongo	1.01%	186.3MiB / 1.939GiB	9.39%	66.3kB / 289kB	25.8MB / 2.87MB	36
ca6b61b5c670	contohnginx	0.00%	5.301MiB / 1.939GiB	0.27%	3.7kB / 2.43kB	0B / 12.3kB	7
e6699fd1007c	contohredis	0.43%	2.145MiB / 1.939GiB	0.11%	1.69kB / 0B	0B / 16.4kB	5



---

# Container Resource Limit



# Container Resource Limit

- Saat membuat container, secara default dia akan menggunakan semua CPU dan Memory yang diberikan ke Docker (Mac dan Windows), dan akan menggunakan semua CPU dan Memory yang tersedia di sistem Host (Linux)
- Jika terjadi kesalahan, misal container terlalu banyak memakan CPU dan Memory, maka bisa berdampak terhadap performa container lain, atau bahkan ke sistem host
- Oleh karena itu, ada baiknya ketika kita membuat container, kita memberikan resource limit terhadap container nya



# Memory

- Saat membuat container, kita bisa menentukan jumlah memory yang bisa digunakan oleh container ini, dengan menggunakan perintah `--memory` diikuti dengan angka memory yang diperbolehkan untuk digunakan
- Kita bisa menambahkan ukuran dalam bentuk b (bytes), k (kilo bytes), m (mega bytes), atau g (giga bytes), misal 100m artinya 100 mega bytes



# CPU

- Selain mengatur Memory, kita juga bisa menentukan berapa jumlah CPU yang bisa digunakan oleh container dengan parameter `--cpus`
- Jika misal kita set dengan nilai 1.5, artinya container bisa menggunakan satu dan setengah CPU core

## Kode : Menambah Resource Limit

```
→ ~ docker container create --name smallnginx --publish 8081:80 --memory 100m --cpus 0.5 nginx:latest  
daaac25d579d50c7ec2e7d4fc80601ca727cb589b2afb54e19e2c88d041ec02c
```

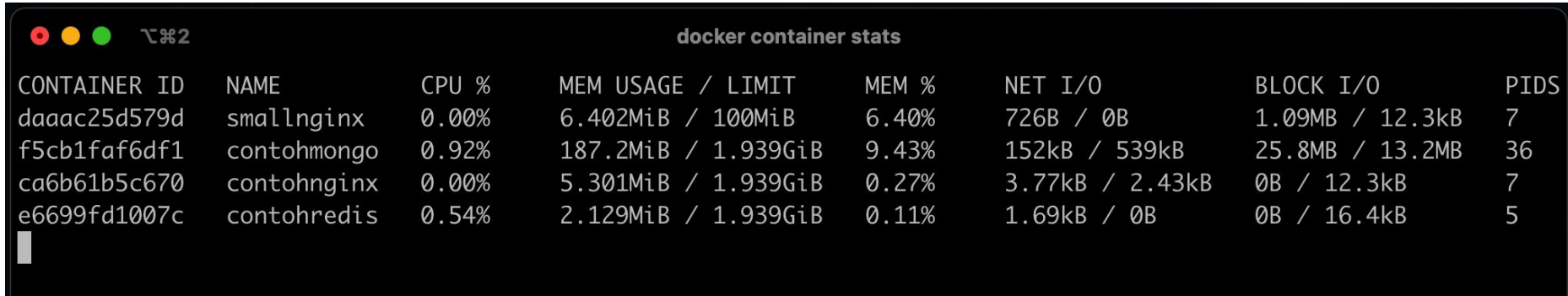
```
→ ~ docker container start smallnginx  
smallnginx
```

```
→ ~ docker container ls
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
daaac25d579d	nginx:latest	"/docker-entrypoint..."	12 seconds ago	Up 4 seconds	0.0.0.0:8081->80/
tcp	smallnginx				
f5cb1faf6df1	mongo:latest	"docker-entrypoint.s..."	About an hour ago	Up About an hour	0.0.0.0:27017->27
017/tcp	contohmongo				
ca6b61b5c670	nginx:latest	"/docker-entrypoint..."	2 hours ago	Up 2 hours	0.0.0.0:8080->80/
tcp	contohnnginx				
e6699fd1007c	redis:latest	"docker-entrypoint.s..."	3 hours ago	Up 3 hours	6379/tcp
	contohredis				

```
→ ~
```

# Kode : Container Stats



CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O	PIDS
daaac25d579d	smallnginx	0.00%	6.402MiB / 100MiB	6.40%	726B / 0B	1.09MB / 12.3kB	7
f5cb1faf6df1	contohmongo	0.92%	187.2MiB / 1.939GiB	9.43%	152kB / 539kB	25.8MB / 13.2MB	36
ca6b61b5c670	contohnginx	0.00%	5.301MiB / 1.939GiB	0.27%	3.77kB / 2.43kB	0B / 12.3kB	7
e6699fd1007c	contohredis	0.54%	2.129MiB / 1.939GiB	0.11%	1.69kB / 0B	0B / 16.4kB	5

---

# Bind Mounts



# Bind Mounts

- Bind Mounts merupakan kemampuan melakukan mounting (sharing) file atau folder yang terdapat di sistem host ke container yang terdapat di docker
- 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 di 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



# Melakukan Mounting

- Untuk melakukan mounting, kita bisa menggunakan perintah berikut :  
docker container create --name namacontainer --mount  
"type=bind,source=folder,destination=folder,readonly" image:tag

## Kode : Melakukan Mounting

```
→ ~ docker container create --name mongodata --mount "type=bind,source=/Users/khannedy/mongo-data,destination=/data/db" --publish 27018:27017 --env MONGO_INITDB_ROOT_USERNAME=eko --env MONGO_INITDB_ROOT_PASSWORD=eko mongo:latest
9c56e34698f5e9b6b3cbf62acbbcd8956e22180627b258ba134ebb330f5de76f
→ ~ docker container start mongodata
mongodata
→ ~ docker container ls
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
9c56e34698f5 mongodata	mongo:latest	"docker-entrypoint.s..."	11 seconds ago	Up 4 seconds	0.0.0.0:27018->27017/tcp
daaac25d579d smallnginx	nginx:latest	"/docker-entrypoint...."	8 hours ago	Up 8 hours	0.0.0.0:8081->80/tcp
f5cb1faf6df1 contohmongo	mongo:latest	"docker-entrypoint.s..."	9 hours ago	Up 9 hours	0.0.0.0:27017->27017/tcp

---

# Docker Volume



# Docker Volume

- Fitur Bind Mounts sudah ada sejak Docker versi awal, di versi terbaru direkomendasikan menggunakan Docker Volume
- Docker Volume mirip dengan Bind Mounts, bedanya adalah terdapat management Volume, dimana kita bisa membuat Volume, melihat daftar Volume, dan 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 pada volume, data di manage oleh Docker



# Melihat Docker Volume

- 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 terbuat, walaupun kita belum pernah membuatnya sama sekali
- Kita bisa gunakan perintah berikut untuk melihat daftar volume :  
docker volume ls



## Kode : Melihat Volume

```
→ ~ docker volume ls
```

DRIVER	VOLUME NAME
local	3e6f4cf1171c55e27a94e61a6ecd33b56faeba566cecc41a85706653bcd21b52
local	3f21a9aa15fd4bef0f66a7eac893cd7be3d801431437617def054fb95b9dfe7f
local	6ac94dd52b647f20f5a86ee54fef6a69a5a7f68b4285ad4fa81724238d47c0690
local	6b180fdd4546574c4e77c7d2ac2eb2e9042506c5bb4b7cc8f658462bebff4bcd
local	6351f48941257e0f2a8862e5f601c663ad0cdbcf83ce33dcd5eaba5bebc499ca
local	70885a1c2c79470baf9cc649aeb8c9aaaada2155d266dda88e3f1cb50c6ba472
local	a2c01f25c2591bc767d307530d32229914ef1dc978fe321f186d205e803fe95a
local	afaeed76be0f51b2c666fec9b941c39493bf84d57fc78ed7ba8dd031b114ee3b
local	d06728583d98d8f1f8e07884bda8c7cceb7a10db5abcbf81cdb8c07798e0fbbd

```
→ ~ █
```



# Membuat Volume

- Untuk membuat volume, kita bisa gunakan perintah :  
`docker volume create namavolume`





## Kode : Membuat Volume

```
→ ~ docker volume create mongovolume
```

```
mongovolume
```

```
→ ~ docker volume ls
```

DRIVER	VOLUME NAME
local	3e6f4cf1171c55e27a94e61a6ecd33b56faeba566cecc41a85706653bcd21b52
local	3f21a9aa15fd4bef0f66a7eac893cd7be3d801431437617def054fb95b9dfe7f
local	6ac94dd52b647f20f5a86ee54fef6a69a5a7f68b4285ad4fa81724238d47c0690
local	6b180fdd4546574c4e77c7d2ac2eb2e9042506c5bb4b7cc8f658462bebff4bcd
local	6351f48941257e0f2a8862e5f601c663ad0cdbcf83ce33dcd5eaba5bebc499ca
local	70885a1c2c79470baf9cc649aeb8c9aaaaada2155d266dda88e3f1cb50c6ba472
local	a2c01f25c2591bc767d307530d32229914ef1dc978fe321f186d205e803fe95a
local	afaeed76be0f51b2c666fec9b941c39493bf84d57fc78ed7ba8dd031b114ee3b
local	d06728583d98d8f1f8e07884bda8c7cceb7a10db5abcbf81cdb8c07798e0fbdb
local	mongovolume

```
→ ~
```



# Menghapus Volume

- Volume yang tidak digunakan oleh container bisa kita hapus, tapi jika volume digunakan oleh container, maka tidak bisa kita hapus sampai container nya di hapus
- Untuk menghapus volume, kita bisa gunakan perintah :  
`docker volume rm namavolume`



## Kode : Menghapus Volume

```
→ ~ docker volume rm mongovolume
```

```
mongovolume
```

```
→ ~ docker volume ls
```

```
DRIVER      VOLUME NAME
```

```
local      3e6f4cf1171c55e27a94e61a6ecd33b56faeba566cecc41a85706653bcd21b52
```

```
local      3f21a9aa15fd4bef0f66a7eac893cd7be3d801431437617def054fb95b9dfe7f
```

```
local      6ac94dd52b647f20f5a86ee54fef6a69a5a7f68b4285ad4fa81724238d47c0690
```

```
local      6b180fdd4546574c4e77c7d2ac2eb2e9042506c5bb4b7cc8f658462bebff4bcd
```

```
local      6351f48941257e0f2a8862e5f601c663ad0cdbcf83ce33dcd5eaba5bebc499ca
```

```
local      70885a1c2c79470baf9cc649aeb8c9aaaada2155d266dda88e3f1cb50c6ba472
```

```
local      a2c01f25c2591bc767d307530d32229914ef1dc978fe321f186d205e803fe95a
```

```
local      afaeed76be0f51b2c666fec9b941c39493bf84d57fc78ed7ba8dd031b114ee3b
```

```
local      d06728583d98d8f1f8e07884bda8c7cceb7a10db5abcbf81cdb8c07798e0fbbd
```

```
→ ~ █
```

---

# Container Volume



# Container Volume

- Volume yang sudah kita buat, bisa kita gunakan di container
- Keuntungan menggunakan volume adalah, jika container kita hapus, 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 nama volume



## Kode : Container Volume

```
→ ~ docker volume create mongodata
mongodata
→ ~ docker container create --name mongovolume --mount "type=volume,source=mongodata,destination=/data/db" --p
ublish 27019:27017 --env MONGO_INITDB_ROOT_USERNAME=eko --env MONGO_INITDB_ROOT_PASSWORD=eko mongo:latest
5832061cd6851642bda0fbf59c36bf39988f212128043b38a1212a8c0d8efc04
→ ~ docker container start mongovolume
mongovolume
→ ~ █
```

---

# Backup Volume



# Backup Volume

- Sayangnya, sampai saat ini, tidak ada cara otomatis melakukan backup volume yang sudah kita buat
- 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

- Matikan container yang menggunakan volume yang ingin kita backup
- Buat container baru dengan dua mount, volume yang ingin kita backup, dan bind mount folder dari sistem host
- Lakukan backup menggunakan container dengan cara meng-archive isi volume, dan simpan di bind mount folder
- Isi file backup sekarang ada di folder sistem host
- Delete container yang kita gunakan untuk melakukan backup



## Kode : Membuat Backup Container

```
→ ~ docker container create --name nginxbackup --mount "type=bind,source=/Users/khannedy/backup,destination=/b
ackup" --mount "type=volume,source=mongodata,destination=/data" nginx:latest
0853e94deae274f88488bdb277d12e1df191a25c4c23fc88df75aad9ca55f58f
→ ~ docker container start nginxbackup
nginxbackup
→ ~ docker container exec -i -t nginxbackup /bin/bash
root@0853e94deae2:/# tar cvf /backup/backup.tar.gz /data
tar: Removing leading `/' from member names
/data/
/data/WiredTiger.wt
/data/mongod.lock
/data/index-1--1735267302161298671.wt
/data/collection-4--7430306916140548523.wt
/data/diagnostic.data/
/data/diagnostic.data/metrics.2021-12-11T15-44-20Z-00000
```



# Menjalankan Container Secara Langsung

- Melakukan backup secara manual agak sedikit ribet karena kita harus start container terlebih dahulu, setelah backup, hapus container nya lagi
- Kita bisa menggunakan perintah run untuk menjalankan perintah di container dan gunakan parameter `--rm` untuk melakukan otomatis remove container setelah perintahnya selesai berjalan



## Kode : Backup Dengan Container Run

```
→ ~ docker container run --rm --name ubuntu --mount "type=bind,source=/Users/khannedy/backup,destination=/back  
up" --mount "type=volume,source=mongodata,destination=/data" ubuntu:latest tar cvf /backup/backup.tar.gz /data  
tar: Removing leading `/' from member names  
/data/  
/data/WiredTiger.wt  
/data/mongod.lock  
/data/index-1--1735267302161298671.wt  
/data/collection-4--7430306916140548523.wt  
/data/diagnostic.data/  
/data/diagnostic.data/metrics.2021-12-11T15-44-20Z-00000  
/data/WiredTiger
```

---

# Restore Volume



# 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
- Buat container baru dengan dua mount, volume baru untuk restore backup, dan bind 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 di restore ke volume
- Delete container yang kita gunakan untuk melakukan restore
- Volume baru yang berisi data backup siap digunakan oleh container baru



## Kode : Restore Backup

```
→ ~ docker volume create mongodatabackup
mongodatabackup
→ ~ docker container run --rm --name ubuntu --mount "type=bind,source=/Users/khannedy/backup,destination=/back
up" --mount "type=volume,source=mongodatabackup,destination=/data" ubuntu:latest bash -c "cd /data && tar xvf /
backup/backup.tar.gz --strip 1"
data/WiredTiger.wt
data/mongod.lock
data/index-1--1735267302161298671.wt
data/collection-4--7430306916140548523.wt
data/diagnostic.data/
data/diagnostic.data/metrics.2021-12-11T15-44-20Z-00000
data/WiredTiger
data/collection-2--7430306916140548523.wt
data/journal/
```



---

# Docker Network



# Docker Network

- Saat kita membuat container di docker, secara default container akan saling terisolasi satu sama lain, jadi jika 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 kita gunakan.
- bridge, yaitu driver yang digunakan untuk membuat network secara virtual yang memungkinkan container yang terkoneksi di bridge network yang sama saling berkomunikasi
- host, yaitu driver yang digunakan untuk membuat network yang sama dengan sistem host. host hanya jalan di Docker Linux, tidak bisa digunakan di Mac atau Windows
- none, yaitu driver untuk membuat network yang tidak bisa berkomunikasi



# Melihat Network

- Untuk melihat network di Docker, kita bisa gunakan perintah :  
docker network ls



## Kode : Melihat Network

```
→ ~ docker network ls
```

NETWORK ID	NAME	DRIVER	SCOPE
d44619a34165	bridge	bridge	local
02363b8211c2	host	host	local
e8b61b96a792	none	null	local

```
→ ~ █
```



# Membuat Network

- Untuk membuat network baru, kita bisa menggunakan perintah :  
`docker network create --driver namadriver namanetwork`



## Kode : Membuat Network

```
→ ~ docker network create --driver bridge contohnetwork
8dc60b1ae6e303c3fc5f13ea9c330143250a719433e9aad49ea111234e0fa96d
→ ~ docker network ls
NETWORK ID      NAME                DRIVER              SCOPE
d44619a34165    bridge              bridge               local
8dc60b1ae6e3    contohnetwork       bridge               local
02363b8211c2    host                host                 local
e8b61b96a792    none                null                 local
→ ~ █
```



# Menghapus Network

- Untuk menghapus Network, kita bisa gunakan perintah :  
`docker network rm namanetwork`
- Network tidak bisa dihapus jika sudah digunakan oleh container. Kita harus menghapus container nya terlebih dahulu dari Network





## Kode : Menghapus Network

```
→ ~ docker network rm contohnetwork
contohtnetwork
→ ~ docker network ls
NETWORK ID      NAME      DRIVER      SCOPE
d44619a34165    bridge    bridge      local
02363b8211c2    host      host        local
e8b61b96a792    none      null        local
→ ~
```

---

# Container Network



# 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 network nya)
- Container bisa mengakses container lain dengan menyebutkan hostname dari container nya, yaitu nama container nya



# Membuat Container dengan Network

- Untuk menambahkan container ke network, kita bisa menambahkan perintah `--network` ketika membuat container, misal :  
`docker container create --name namacontainer --network namanetwork image:tag`



## Kode : Membuat Container Dengan Network

```
→ ~ docker network create mongonetwork
f21355aac4d48e6f0e6c4cfcf568e4b536d0c3543e33565f97ffb1c6325e5009
→ ~ docker container create --name mongodb --network mongonetwork --env MONGO_INITDB_ROOT_USERNAME=eko --env MONGO_INITDB_ROOT_PASSWORD=eko mongo:latest
7a64d7bbb831c3d4bcfed9db11fe75bfb3845a9cb2d0f74cd3c8963c0270035e
→ ~ docker container create --name mongodbexpress --network mongonetwork --publish 8081:8081 --env ME_CONFIG_MONGODB_URL="mongodb://eko:eko@mongodb:27017/" mongo-express:latest
04a15fde2cf9a8040fb17c01aa5bba26709138d89891ac3e534a5728e89a92bf
→ ~ docker container start mongodb
mongodb
→ ~ docker container start mongodbexpress
mongodbexpress
→ ~
```



# Menghapus Container dari Network

- Jika diperlukan, kita juga bisa menghapus container dari network dengan perintah :  
`docker network disconnect namanetwork namacontainer`



## Kode : Menghapus Container dari Network

```
→ ~ docker network disconnect mongonetwork mongodb  
→ ~ █
```



# Menambahkan Container ke Network

- Jika containernya sudah terlanjur dibuat, kita juga bisa menambahkan container yang sudah dibuat ke network dengan perintah :  
`docker network connect namanetwork namacontainer`





## Kode : Menambahkan Container ke Network

```
→ ~ docker network connect mongonetwork mongodb  
→ ~ █
```

—

**Inspect**



# Inspect

- Setelah kita men-download image, atau membuat network, volume dan container. Kadang kita ingin melihat detail dari tiap hal tersebut
- Misal kita ingin melihat detail dari image, perintah apa yang digunakan oleh image tersebut? Environment variable apa yang digunakan? Atau port apa yang digunakan?
- Misal kita juga ingin melihat detail dari container, Volume apa yang digunakan? Environment variable apa yang digunakan? Port forwarding apa yang digunakan? dan lain-lain
- 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 : `docker image inspect namaimage`
- Untuk melihat detail dari container, gunakan : `docker container inspect namacontainer`
- Untuk melihat detail dari volume, gunakan : `docker volume inspect namavolume`
- Untuk melihat detail dari network, gunakan : `docker network inspect namanetwork`



# Kode : Menggunakan Inspect

```
→ ~ docker image inspect redis:latest
[
  {
    "Id": "sha256:aea9b698d7d1d2fb22fe74868e27e767334b2cc629a8c6f9db8cc1747ba299fd",
    "RepoTags": [
      "redis:latest"
    ],
    "RepoDigests": [
      "redis@sha256:2f502d27c3e9b54295f1c591b3970340d02f8a5824402c8179dcd20d4076b796"
    ],
    "Parent": "",
    "Comment": "",
    "Created": "2021-12-03T03:39:27.036869939Z",
    "Container": "5a7860a86d8bf094066ab13418f0af0e950079918a8694f71fb476117dd833cd",
    "ContainerConfig": {
      "Hostname": "5a7860a86d8b",
      "Domainname": "",
      "User": "",
      "AttachStdin": false,
      "AttachStdout": false,
      "AttachStderr": false,
      "ExposedPorts": {
```

---

# Prune



# Prune

- Saat kita menggunakan Docker, kadang ada kalanya kita ingin membersihkan hal-hal yang sudah tidak digunakan lagi di Docker, misal container yang sudah di stop, 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`





## Kode : Menggunakan Prune

```
→ ~ docker image prune
WARNING! This will remove all dangling images.
Are you sure you want to continue? [y/N] y
Total reclaimed space: 0B
→ ~ docker network prune
WARNING! This will remove all custom networks not used by at least one container.
Are you sure you want to continue? [y/N] y
Deleted Networks:
contohnetwork
mongonetwork

→ ~ docker volume prune
WARNING! This will remove all local volumes not used by at least one container.
Are you sure you want to continue? [y/N] y
Deleted Volumes:
e00c5a4b790555751ef522ecaa00de7bc9635a6df51c15bbabc3853fa41a4059
3980cbe30b7aef43d5836894679f8d844b07c5a7b653a1b4c9e6daf302e0082a
d06728583d08d8f1f8e07884bda8c7c6eb7a10db5abcbf81c4b8c07708e0fbbd
```

---

# Materi Selanjutnya



# Materi Selanjutnya

- Docker Dockerfile
- Docker Compose