



# Docker Compose

Eko Kurniawan Khannedy

# Eko Kurniawan Khannedy

- Technical architect at one of the biggest ecommerce company in Indonesia
- 11+ 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

- Docker Dasar
- Docker Dockerfile



# Agenda

- Pengenalan Docker Compose
- Configuration File
- Services
- Project
- Environment Variable
- Volume
- Network
- Dan lain-lain

---

# Pengenalan Docker Compose



# Pengenalan Docker Compose

- Docker Compose adalah tool yang digunakan untuk mendefinisikan dan menjalankan multiple Docker Container secara sekaligus
- Dengan menggunakan Docker Compose, kita bisa menggunakan file YAML untuk melakukan konfigurasi Docker Container nya
- Lalu dengan sebuah perintah, kita bisa membuat semua Docker Container dan menjalankannya sekaligus dari file konfigurasi tersebut
- Dengan begitu, kita tidak perlu lagi mengetikkan perintah docker create secara manual ketika ingin membuat Docker Container



# Fitur Docker Compose

- Memiliki multiple isolated environment dalam satu docker host / server, atau dibilang project. Hal ini memungkinkan kita bisa membuat banyak sekali jenis environment untuk Docker Compose. Secara default nama project akan menggunakan nama folder konfigurasi
- Hanya membuat container yang berubah. Docker Compose bisa mendeteksi container mana yang harus dibuat dan tidak perlu dibuat ulang dari perubahan file konfigurasi





# Kapan Menggunakan Docker Compose

- Membuat Development Environment. Ketika kita develop aplikasi, kita sering butuh tool-tool berbeda untuk tiap project. Kita bisa gunakan Docker Compose untuk melakukan setup nya
- Automated Testing. Kadang ketika kita membuat automation testing, banyak sekali hal yang harus kita jalankan secara manual. Docker Compose bisa membantu kita untuk otomatisasi proses setup nya
- Deployment. Docker Compose juga bisa digunakan untuk kasus deployment aplikasi kita. Jadi kita tidak perlu lakukan start manual aplikasi kita di server, cukup jalankan menggunakan Docker Compose

---

# Menginstall Docker Compose



# Menginstall Docker Compose

- Dulu, aplikasi Docker Compose terpisah dengan aplikasi Docker.
- Dulu, kita perlu menggunakan perintah docker-compose untuk menggunakan Docker Compose
- Nemu di Docker versi terbaru, Docker Compose sudah tersedia secara otomatis di dalam Docker nya
- Dan untuk menggunakan Docker Compose, kita bisa gunakan perintah :  
docker compose



## Kode : Docker Compose

```
→ ~ docker compose version  
Docker Compose version v2.10.2  
→ ~
```

---

# Configuration File



# Configuration File

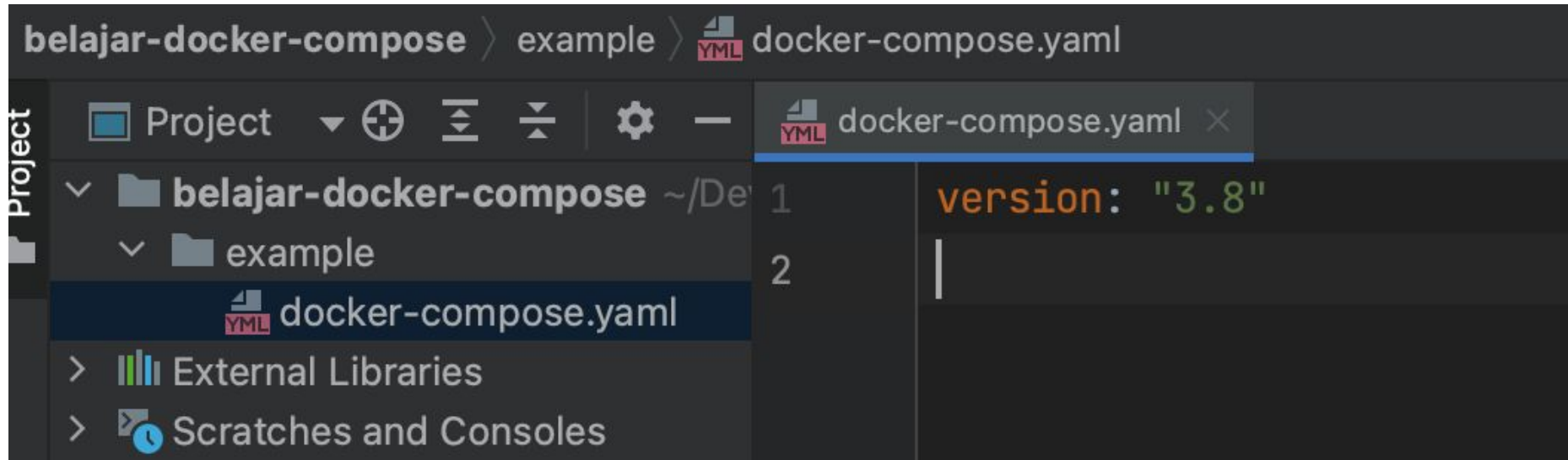
- Docker Compose menyimpan konfigurasi nya dalam bentuk file YAML : <https://yaml.org/>
- File YAML mirip JSON, namun lebih sederhana
- Biasanya file konfigurasinya disimpan dalam file bernama docker-compose.yaml
- Seperti yang dibahas di awal, nama project secara default akan menggunakan nama folder lokasi docker-compose.yaml tersebut berada



# Versi Konfigurasi

- Saat video ini dibuat, sekarang konfigurasi Docker Compose masih menggunakan versi 3.x
- Kita bisa lihat versi terbaru di halaman ini :  
<https://docs.docker.com/compose/compose-file/compose-file-v3/>
- Ingat ini adalah versi konfigurasi file, bukan versi aplikasi Docker Compose

## Kode : Docker Compose File



The screenshot shows an IDE interface with a dark theme. The breadcrumb navigation at the top reads: `belajar-docker-compose > example > docker-compose.yml`. The left sidebar, labeled "Project", shows a tree view with the following structure:

- belajar-docker-compose ~/Dev
  - example
    - docker-compose.yml (selected)
  - External Libraries
  - Scratches and Consoles

The main editor area displays the content of `docker-compose.yml` with line numbers 1 and 2 on the left. The code shown is:

```
1 version: "3.8"
2
```



---

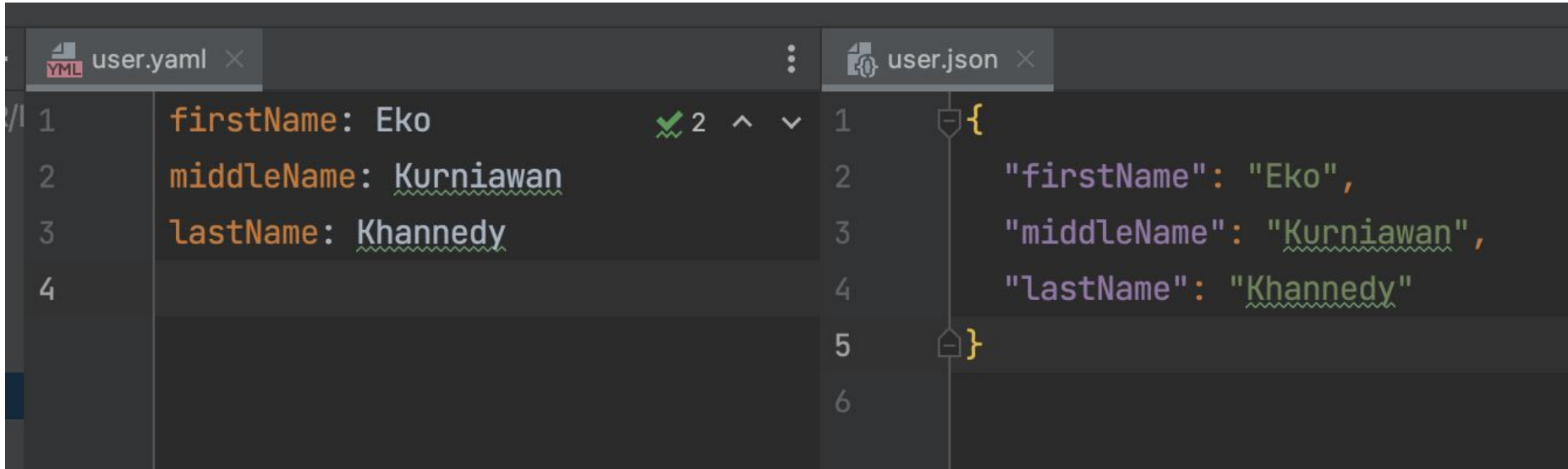
# Yaml



# Yaml

- Yaml adalah sebuah jenis file yang biasa digunakan untuk menyimpan konfigurasi
- Yaml mirip seperti JSON, hanya saja tidak menggunakan kurung kurawal
- Yaml akan memiliki attribute dan value
- <https://yaml.org/>

## Kode : Yaml Attribute



The image shows a code editor with two files open: `user.yaml` and `user.json`. The `user.yaml` file contains the following content:

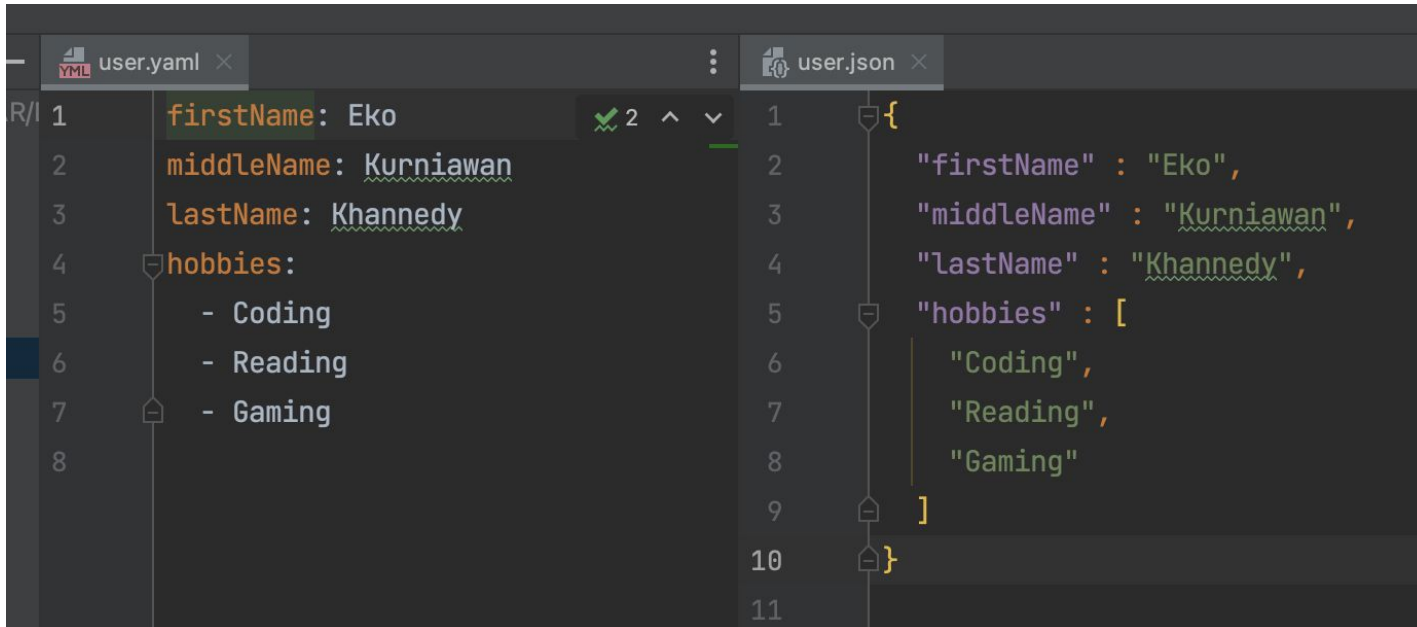
```
1 firstName: Eko
2 middleName: Kurniawan
3 lastName: Khannedy
4
```

The `user.json` file contains the following content:

```
1 {
2   "firstName": "Eko",
3   "middleName": "Kurniawan",
4   "lastName": "Khannedy"
5 }
```

The editor interface includes a dark theme, a sidebar on the left, and a top bar with file names and icons. The `user.yaml` file has a green checkmark icon next to line 2, indicating a successful validation or linting operation.

## Kode : Yaml Array

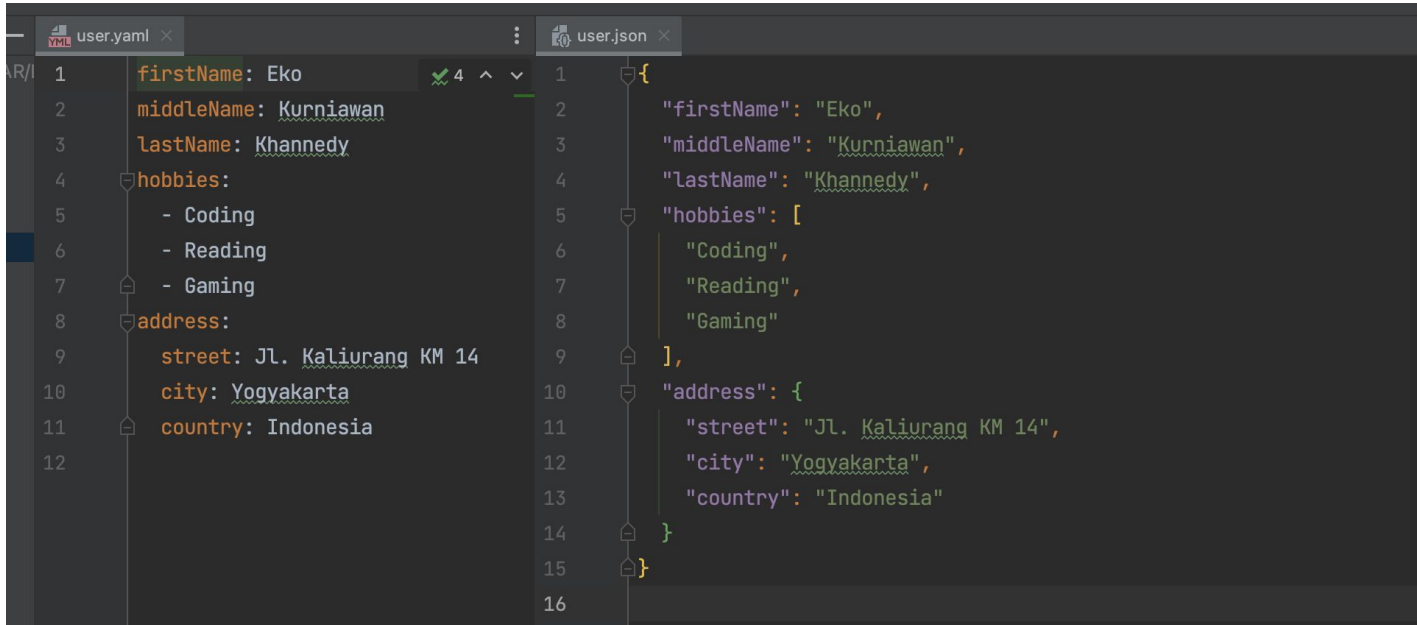


The image shows a code editor with two files open: `user.yaml` and `user.json`. The `user.yaml` file on the left contains a user profile with a list of hobbies. The `user.json` file on the right shows the equivalent JSON representation, where the hobbies are stored in an array.

```
user.yaml
1 firstName: Eko
2 middleName: Kurniawan
3 lastName: Khannedy
4 hobbies:
5   - Coding
6   - Reading
7   - Gaming

user.json
1 {
2   "firstName" : "Eko",
3   "middleName" : "Kurniawan",
4   "lastName" : "Khannedy",
5   "hobbies" : [
6     "Coding",
7     "Reading",
8     "Gaming"
9   ]
10 }
11
```

# Kode : Yaml Nested Object

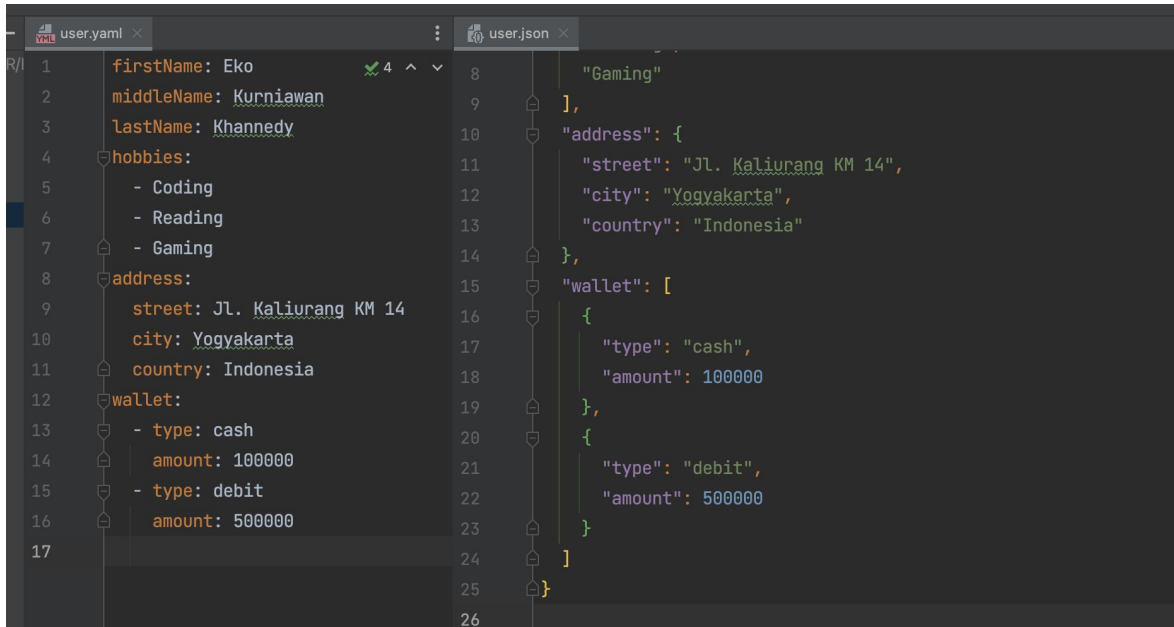


The image shows a side-by-side comparison of a user object in YAML and JSON formats. The left pane, titled 'user.yaml', displays a YAML document with a root object containing 'firstName', 'middleName', 'lastName', 'hobbies' (an array), and 'address' (a nested object). The right pane, titled 'user.json', shows the equivalent JSON representation, where the root is an object with the same structure, using double quotes for strings and a colon for object delimiters. The editor interface includes line numbers, a search bar, and file tabs.

```
user.yaml
1 firstName: Eko
2 middleName: Kurniawan
3 lastName: Khannedy
4 hobbies:
5   - Coding
6   - Reading
7   - Gaming
8 address:
9   street: Jl. Kaliurang KM 14
10  city: Yogyakarta
11  country: Indonesia
12

user.json
1 {
2   "firstName": "Eko",
3   "middleName": "Kurniawan",
4   "lastName": "Khannedy",
5   "hobbies": [
6     "Coding",
7     "Reading",
8     "Gaming"
9   ],
10  "address": {
11    "street": "Jl. Kaliurang KM 14",
12    "city": "Yogyakarta",
13    "country": "Indonesia"
14  }
15 }
16
```

# Kode : Yaml Array Nested Object



The image shows a code editor with two tabs: 'user.yaml' and 'user.json'. The 'user.yaml' tab is active, displaying a YAML file with the following content:

```
1 firstName: Eko
2 middleName: Kurniawan
3 lastName: Khannedy
4 hobbies:
5   - Coding
6   - Reading
7   - Gaming
8 address:
9   street: Jl. Kaliurang KM 14
10  city: Yogyakarta
11  country: Indonesia
12 wallet:
13   - type: cash
14     amount: 100000
15   - type: debit
16     amount: 500000
17
```

The 'user.json' tab is also visible, displaying the corresponding JSON file:

```
8   "Gaming"
9 ],
10 "address": {
11   "street": "Jl. Kaliurang KM 14",
12   "city": "Yogyakarta",
13   "country": "Indonesia"
14 },
15 "wallet": [
16   {
17     "type": "cash",
18     "amount": 100000
19   },
20   {
21     "type": "debit",
22     "amount": 500000
23   }
24 ]
25 }
26
```



# JSON to Yaml

- <https://www.json2yaml.com/>

---

# Membuat Container





# Membuat Konfigurasi Container

- Sebelumnya untuk membuat container, kita selalu menggunakan perintah docker create
- Namun sekarang kita bisa buat container hanya menggunakan configuration file di Docker Compose
- Pada file yaml, kita bisa tambahkan bagian services untuk menentukan container-nya
- Dalam service tersebut, kita bisa tentukan container name dan image untuk docker container yang akan kita buat

## Kode : Membuat Container

```
docker-compose.yml x
1  version: "3.8"
2
3  >> services:
4  >   nginx-example:
5       container_name: nginx-example
6       image: nginx:latest
7
```



# Membuat Container

- Setelah membuat konfigurasi file, Container tidak langsung jadi, kita harus membuatnya dengan menggunakan Docker Compose, yaitu dengan perintah :  
`docker compose create`



## Kode : Membuat Container

```
→ example docker compose create  
[+] Running 2/2  
:: Network example_default Created  
:: Container nginx-example Created  
→ example
```

---

# Menjalankan Container



# Menjalankan Container

- Setelah membuat Container, Container tidak akan berjalan otomatis
- Kita harus menjalankannya secara manual, bisa menggunakan perintah `docker container start`, atau bisa menggunakan Docker Compose
- Untuk menggunakan Docker Compose, kita bisa gunakan perintah `docker compose start`



## Kode : Menjalankan Container

```
→ example docker compose start
[+] Running 1/1
:: Container nginx-example Started
→ example
```

---

# Melihat Container





# Melihat Container

- Biasanya, saat kita ingin melihat Container, kita biasanya gunakan perintah `docker container ls`
- Namun menggunakan perintah itu, akan menampilkan semua container, baik itu yang dibuat oleh Docker Compose, atau dibuat manual
- Jika kita ingin melihat status Container yang hanya terdapat di konfigurasi file Docker Compose, kita bisa gunakan perintah :  
`docker compose ps`



## Kode : Melihat Container

```
→ example docker compose ps
```

NAME	COMMAND	SERVICE	STATUS	PORTS
nginx-example	"/docker-entrypoint...."	nginx-example	running	80/tcp

```
→ example
```

---

# Menghentikan Container



# Menghentikan Container

- Untuk menghentikan Container, kita bisa menggunakan perintah :  
docker compose stop
- Menghentikan Container hanya men-stop Container, tidak akan menghapus Container nya



## Kode : Menghentikan Container

```
→ example docker compose stop
[+] Running 1/1
  :: Container nginx-example Stopped
→ example
```

---

# Menghapus Container



# Menghapus Container

- Jika kita sudah tidak butuh lagi container yang terdapat di file konfigurasi, kita bisa menghapusnya
- Kita bisa hapus secara manual menggunakan perintah docker container rm, atau menggunakan Docker Compose
- Untuk menghapus container menggunakan Docker Compose, kita bisa gunakan perintah :  
docker compose down
- Secara otomatis semua Container dan Network dan Volume yang digunakan oleh Container tersebut akan dihapus



## Kode : Menghapus Container

```
→ example docker compose down
[+] Running 2/1
  :: Container nginx-example   Removed
  :: Network example_default   Removed
→ example
```



---

**Project Name**



## Project Name

- Seperti yang sudah dijelaskan di awal, saat kita menggunakan Docker Compose, informasi konfigurasi Docker Compose akan disimpan dalam project
- Secara default nama project-nya adalah nama folder lokasi file docker-compose.yaml
- Untuk melihat daftar project yang sedang berjalan, kita bisa gunakan perintah :  
docker compose ls



## Kode : Project Name

```
→ example docker compose ls
```

NAME	STATUS	CONFIG FILES
example	running(1)	/Users/khannedy/Developments/BELAJA

```
→ example
```

---

Service



# Service

- Dalam konfigurasi Docker Compose, container disimpan dalam konfigurasi bernama services
- Kita bisa menambahkan satu atau lebih services dalam konfigurasi file nya
- <https://docs.docker.com/compose/compose-file/compose-file-v3/#service-configuration-reference>

# Kode : Service

```
docker-compose.yml x
1  version: "3.9"
2
3  >> services:
4  >  nginx-example:
5      image: nginx:latest
6      container_name: nginx-example
7  >  mongodb-example:
8      image: mongo:latest
9      container_name: mongodb-example
10
```



## Kode : Menjalankan Container

```
.. Network services_default    Removed
→ services docker compose create
[+] Running 3/3
  :: Network services_default    Created
  :: Container mongodb-example   Created
  :: Container nginx-example     Created
→ services docker compose start
[+] Running 2/2
  :: Container nginx-example     Started
  :: Container mongodb-example   Started
→ services
```

—

# Komentar





# Komentar

- Salah satu keunggulan menggunakan Yaml dari pada JSON adalah, di Yaml kita bisa menambahkan komentar dengan diawali karakter #
- Di JSON kita tidak bisa menambahkan komentar
- Komentar secara otomatis akan dihiraukan oleh Docker Compose



## Kode : Komentar

```
2
3 >> services:
4
5     # This is example nginx
6 > nginx-example:
7     image: nginx:latest
8     container_name: nginx-example
9
10    # This is example mongodb
11 > mongodb-example:
12     image: mongo:latest
13     container_name: mongodb-example
14
15
```

---

Port



# Port

- Saat membuat Container, kita bisa mengekspose port di Container keluar menggunakan Port Forwarding
- Kita juga bisa melakukan hal tersebut di konfigurasi file Docker Compose dengan menggunakan attribute ports
- Attribute ports berisi array object port
- <https://docs.docker.com/compose/compose-file/compose-file-v3/#ports>



## Short Syntax

- Saat menentukan port, kita bisa gunakan dua cara, pertama adalah short syntax, yang berisi string port HOST:CONTAINER
- Misal “8080:80”, artinya kita akan menggunakan port 8080 di Host untuk di forward ke port 80 di Container



# Long Syntax

- Sedangkan untuk long syntax, kita bisa buat dalam bentuk object yang berisi :
- target: Port di dalam container
- published: Port yang digunakan di host
- protocol: Protocol port (tcp atau udp)
- mode: host untuk port di tiap Node, atau ingress untuk swarm mode. Karena kita tidak menggunakan docker swarm, jadi kita cukup gunakan nilai host

# Kode : Port

```
2
3 >> services:
4 > nginx-port1:
5     image: nginx:latest
6     container_name: nginx-port1
7     ports:
8     - protocol: tcp
9       published: 8080
10      target: 80
11 > nginx-port2:
12     image: nginx:latest
13     container_name: nginx-port2
14     ports:
15     - "8081:80"
16
17
```



## Kode : Menjalankan Container

```
→ ports docker compose create
[+] Running 3/3
  :: Network ports_default Created
  :: Container nginx-port2 Created
  :: Container nginx-port1 Created
→ ports docker compose start
[+] Running 2/2
  :: Container nginx-port2 Started
  :: Container nginx-port1 Started
→ ports
```



---

# Environment Variable



# Environment Variable

- Saat membuat container, kita juga menambahkan environment variable untuk digunakan di dalam container
- Saat menggunakan konfigurasi file Docker Compose, kita bisa tambahkan environment variable dengan menggunakan attribute environment



## Kode : Environment Variable

```
3  >> services:
4  > mongodb-example:
5      image: mongo:latest
6      container_name: mongodb-example
7      ports:
8      - "27017:27017"
9      environment:
10         MONGO_INITDB_ROOT_USERNAME: khannedy
11         MONGO_INITDB_ROOT_PASSWORD: khannedy
12         MONGO_INITDB_DATABASE: admin
```

13



## Kode : Menjalankan Container

```
Terminal: Local x + v
→ environments docker compose create
[+] Running 2/2
  :: Network environments_default Created
  :: Container mongodb-example Created
→ environments docker compose start
[+] Running 1/1
  :: Container mongodb-example Started
→ environments
```

---

# Bind Mount



# Bind Mount

- Untuk melakukan bind mount, kita juga bisa lakukan di konfigurasi file Docker Compose
- Kita bisa gunakan attribute volumes di services
- Kita bisa tambahkan satu atau lebih bind mount jika kita mau
- <https://docs.docker.com/compose/compose-file/compose-file-v3/#volumes>



## Short Syntax

- Untuk Bind Mount, kita bisa gunakan short syntax dan long syntax
- Untuk short syntax, kita bisa gunakan nilai SOURCE:TARGET:MODE, dimana SOURCE adalah lokasi di host, dan TARGET adalah lokasi di container
- MODE adalah mode bind mount, ro untuk readonly, rw untuk read write (default)
- SOURCE bisa menggunakan relative path dengan diawali . (titik), atau absolute path

## Kode : Bind Mount Short Syntax

```
3  >> services:
4  >  mongodb1:
5      image: mongo:latest
6      container_name: mongodb1
7      ports:
8      - "27017:27017"
9      environment:
10         MONGO_INITDB_ROOT_USERNAME: khannedy
11         MONGO_INITDB_ROOT_PASSWORD: khannedy
12         MONGO_INITDB_DATABASE: admin
13     volumes:
14     - "./data-mongo1:/data/db"
15
```





# Long Syntax

- Untuk menggunakan long syntax, kita bisa buat dalam bentuk nested object di volumes dengan attribute
- type: tipe mounth, volume atau bind. Volume akan dibahas di materi selanjutnya
- source: sumber path di host atau nama volume
- target: target path di container
- read\_only: flag readonly atau tidak, default nya false

## Kode : Bind Mount Long Syntax

```
16  ▶ mongodb2:
17      image: mongo:latest
18      container_name: mongodb2
19      ports:
20      - "27018:27017"
21      environment:
22      MONGO_INITDB_ROOT_USERNAME: khannedy
23      MONGO_INITDB_ROOT_PASSWORD: khannedy
24      MONGO_INITDB_DATABASE: admin
25      volumes:
26      - type: bind
27        source: "./data-mongo2"
28        target: "/data/db"
29        read_only: false
30
```

---

**Volume**



# Volume

- Docker Compose juga tidak hanya bisa digunakan untuk membuat container, tapi bisa juga digunakan untuk membuat volume
- Kita bisa menggunakan attribute volumes pada konfigurasi file
- <https://docs.docker.com/compose/compose-file/compose-file-v3/#volume-configuration-reference>



## Kode : Volume

```
docker-compose.yml ×
1  version: "3.9"
2
3  volumes:
4    mongo-data1:
5      name: mongo-data1
6    mongo-data2:
7      name: mongo-data2
8
```



# Menggunakan Volume

- Untuk menggunakan Volume, kita gunakan seperti menggunakan bind mount, dengan ketentuan :
- Pada short syntax, kita bisa ganti SOURCE dengan nama volume
- Pada long syntax, kita bisa ganti type menjadi volume, dan source menjadi nama volume

## Kode : Menggunakan Volume Short Syntax

```
mongodb1:
  image: mongo:latest
  container_name: mongodb1
  ports:
    - "27017:27017"
  environment:
    MONGO_INITDB_ROOT_USERNAME: khannedy
    MONGO_INITDB_ROOT_PASSWORD: khannedy
    MONGO_INITDB_DATABASE: admin
  volumes:
    - "mongo-data1:/data/db"
```

# Kode : Menggunakan Volume Long Syntax

```
mongodb2:
  image: mongo:latest
  container_name: mongodb2
  ports:
    - "27018:27017"
  environment:
    MONGO_INITDB_ROOT_USERNAME: khannedy
    MONGO_INITDB_ROOT_PASSWORD: khannedy
    MONGO_INITDB_DATABASE: admin
  volumes:
    - type: volume
      source: mongo-data2
      target: "/data/db"
      read_only: false
```





# Menghapus Volume

- Saat kita menggunakan perintah docker compose down, yang dihapus hanyalah Container dan Network saja
- Volume tidak akan dihapus, hal ini agar jangan sampai kita tidak sengaja menghapus volume
- Jika ingin menghapus volume, kita bisa lakukan manual dengan perintah docker volume rm nama-volume



**Network**



# Network

- Selain membuat Container dan Volume, kita juga bisa menggunakan Docker Compose untuk membuat Network secara otomatis



# Default Network

- Saat kita menjalankan file menggunakan Docker Compose, secara default semua container akan dihubungkan dalam sebuah Network bernama nama-project\_default
- Jadi sebenarnya kita tidak perlu membuat Network secara manual
- Silahkan inspect container yang sudah berjalan menggunakan Docker Compose, lalu lihat pada bagian Network



## Kode : Default Network

```
"MacAddress": "",
"Networks": {
  "example_default": {
    "IPAMConfig": null,
    "Links": null,
    "Aliases": [
      "nginx-example",
      "nginx-example",
      "53c648a82d62"
    ],
    "NetworkID": "23138bb65b5ba5b1ca8e27f25f63343f789b77dc93462c29f394b9a0675f8531",
    "EndpointID": "4849667eb9423d586a8deb6e7104cdde377f361949f8f7ea269e12cf710d84bc",
    "Gateway": "192.168.32.1",
    "IPAddress": "192.168.32.2",
```



# Membuat Network

- Tapi jika kita ingin membuat Network secara manual, kita juga bisa menggunakan Docker Compose
- Kita bisa buat satu atau lebih Network menggunakan attribute networks, dimana kita perlu tentukan :
  - name: Nama network
  - driver: Driver network seperti bridge, host atau none



## Kode : Membuat Network

```
1 version: "3.9"
```

```
3 networks:
```

```
4   network_example:
```

```
5     name: network_example
```

```
6     driver: bridge
```



# Menggunakan Network

- Setelah membuat Network, jika kita ingin menggunakan Network tersebut di Container
- Kita bisa menggunakan attribute networks, dan sebutkan satu atau lebih Network yang ingin kita gunakan



## Kode : Menggunakan Network

```
services:
4  mongodb-example:
5      image: mongo:latest
6      container_name: mongodb-example
7      ports:
8          - "27017:27017"
9      environment:
10         MONGO_INITDB_ROOT_USERNAME: khannedy
11         MONGO_INITDB_ROOT_PASSWORD: khannedy
12         MONGO_INITDB_DATABASE: admin
13     networks:
14         - network_example
```

---

Depends On



# Depends On

- Saat membuat file Docker Compose yang berisi banyak Container
- Kadang kita membuat Container yang butuh Container lain sebelum berjalan
- Atau sederhananya, kita ingin ada urutan Container berjalan
- Secara default, Docker Compose akan menjalankan semua Container secara bersamaan, tanpa ada urutan pasti
- Kita bisa membuat urutan menjalankan Container dengan menggunakan attribute `depends_on`
- Kita bisa sebutkan pada Container, bahwa Container ini hanya bisa berjalan, kalo Container yang lain sudah berjalan
- Kita bisa sebutkan satu atau lebih Container lainnya pada attribute `depends_on`



## Kode : Depends On

```
15 ► mongodb-express-example:
16   image: mongo-express:latest
17   container_name: mongodb-express-example
18   depends_on:
19     - mongodb-example
20   ports:
21     - "8081:8081"
22   environment:
23     ME_CONFIG_MONGODB_ADMINUSERNAME: khannedy
24     ME_CONFIG_MONGODB_ADMINPASSWORD: khannedy
25     ME_CONFIG_MONGODB_SERVER: mongodb-example
26   networks:
27     - network_example
28
```

---

# Restart



# Restart

- Secara default, saat Container mati, maka Docker tidak akan menjalankan lagi Container nya
- Kita harus menjalankan lagi Container nya secara manual
- Kita bisa memaksa sebuah container untuk selalu melakukan restart jika misal terjadi masalah pada Container nya
- Kita bisa tambahkan attribute restart, dengan beberapa value :
- no: default nya tidak pernah restart
- always: selalu restart jika container berhenti, tapi jika di hentikan manual, dia akan restart ketika pertama kali docker restart
- on-failure: restart jika container error dengan indikasi error ketika exit
- unless-stopped: selalu restart container, kecuali ketika dihentikan manual



## Kode : Restart

```
mongodb-express-example:  
  image: mongo-express:latest  
  container_name: mongodb-express-example  
  restart: always  
  depends_on:  
    - mongodb-example  
  ports:  
    - "8081:8081"  
  environment:  
    ME_CONFIG_MONGODB_ADMINUSERNAME: khannedy  
    ME_CONFIG_MONGODB_ADMINPASSWORD: khannedy  
    ME_CONFIG_MONGODB_SERVER: mongodb-example  
  networks:  
    - network_example
```



# Monitor Docker Events

- Untuk melihat kejadian apa saja yang terjadi di Docker secara realtime, kita bisa menggunakan perintah :  
docker events
- <https://docs.docker.com/engine/reference/commandline/events/>
- Contohnya kita bisa memonitor kejadian yang terjadi pada sebuah container dengan perintah :  
docker events --filter 'container=nama'



---

# Resource Limit



# Resource Limit

- Kita juga bisa menggunakan file konfigurasi Docker Compose untuk mengatur Resource Limit untuk CPU dan Memory dari tiap Container yang akan kita buat
- Kita bisa menggunakan attribute deploy, lalu didalamnya menggunakan attribute resources
- Di dalam attribute resources kita bisa tentukan limit dan reservations
- reservation adalah resource yang dijamin bisa digunakan oleh container
- limit adalah limit maksimal untuk resource yang diberikan ke container, namun ingat bisa saja limit ini rebutan dengan container lain




## Kode : Resource Limit

```
nginx-example:  
  image: nginx:latest  
  container_name: nginx-example  
  ports:  
    - "8080:80"  
  deploy:  
    resources:  
      reservations:  
        cpus: "0.25"  
        memory: 50M  
      limits:  
        cpus: "0.5"  
        memory: 100M
```



## Kode : docker container stats



```
CONTAINER ID   NAME          CPU %          MEM USAGE / LIMIT   MEM %           NET I/O        BLOCK I/O      PIDS
b5cca8191932   nginx-example 0.00%          2.57MiB / 100MiB    2.57%           1.32kB / 0B    0B / 12.3kB    3
```

---

# Dockerfile



# Dockerfile

- Sebelumnya kita selalu membuat Container dari Image yang sudah ada
- Docker Compose juga bisa digunakan untuk membuat Container dari Dockerfile yang kita buat
- Hal ini mempermudah kita sehingga tidak perlu membuat Image nya terlebih dahulu secara manual, semua bisa dilakukan otomatis oleh Docker Compose



# Membuat Dockerfile

- Sekarang kita akan buat contoh Dockerfile
- File main.go :  
<https://github.com/ProgrammerZamanNow/belajar-docker-dockerfile/blob/main/env/main.go>
- File Dockerfile :  
<https://github.com/ProgrammerZamanNow/belajar-docker-dockerfile/blob/main/env/Dockerfile>



# Build

- Ketika kita ingin membuat Container dari Dockerfile, kita tidak menggunakan attribute image lagi di service nya
- Kita harus menggunakan attribute build, dimana terdapat attribute :
  - context: berisi path ke file Dockerfile
  - dockerfile: nama file Dockerfile, bisa diganti jika mau
  - args: argument yang dibutuhkan ketika melakukan docker build
- <https://docs.docker.com/compose/compose-file/compose-file-v3/#build>



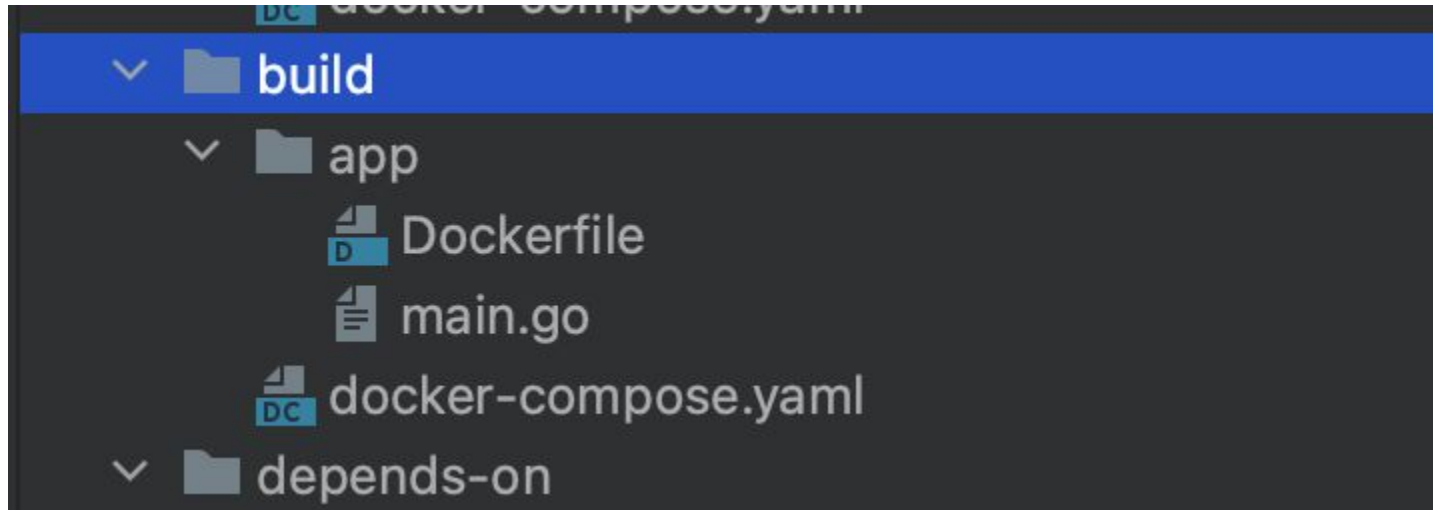


# Image Name

- Secara default, Docker Compose akan membuat Image dengan nama random ketika melakukan build Dockerfile
- Jika kita ingin menentukan namanya, kita bisa tambahkan attribute image pada service, secara otomatis Docker Compose akan membuat image dengan nama sesuai dengan itu



## Kode : Struktur Folder





## Kode : Build

```
▶ app:
  container_name: app
  build:
    context: "./app"
    dockerfile: Dockerfile
  image: "app-golang:1.0.0"
  environment:
    - "APP_PORT=8080"
  ports:
    - "8080:8080"
```



# Build Dockerfile

- Ketika kita menggunakan perintah docker compose start, secara otomatis Docker Compose akan melakukan build terlebih dahulu jika Image nya belum terbuat
- Tapi jika kita hanya ingin melakukan build Image saja, tanpa membuat Container, kita juga bisa menggunakan perintah : docker compose build



# Menghapus Image

- Hasil Image dari Docker Compose tidak akan dihapus ketika melakukan perintah docker image down
- Jadi untuk menghapusnya kita harus hapus manual menggunakan perintah docker image rm nama-image:tag



# Build Ulang

- Perlu diingat, ketika kita mengubah kode program, lalu kita coba stop dan start ulang container menggunakan Docker Compose, bukan berarti kode program terbaru akan berjalan
- Hal ini karena Image versi baru otomatis terbuat, sehingga jika kita ingin menggunakan Image versi baru, kita harus hapus dulu Container nya, lalu buat ulang dengan Image baru

---

# Health Check



# Health Check

- Kita pernah bahas tentang Container Health Check di materi Docker Dockerfile
- Secara default Container yang dibuat, baik itu secara manual ataupun menggunakan Docker Compose, pasti akan selalu menggunakan Health Check yang dibuat di Dockerfile
- Namun, jika kita ingin mengubah Health Check tersebut, itu juga bisa kita lakukan
- Kita bisa ubah di file konfigurasi Docker Compose pada attribute healthcheck di services





# Health Check Attribute

- Health Check memiliki banyak attribute, seperti
- test: berisikan cara melakukan test health check
- interval: interval melakukan health check
- timeout: timeout melakukan health check
- retries: total retry ketika gagal
- start\_period: waktu mulai melakukan health check
- Hampir mirip dengan ketika kita membuat Health Check di Dockerfile



# Dockerfile

- Gunakan file main.go :  
<https://github.com/ProgrammerZamanNow/belajar-docker-dockerfile/blob/main/health/main.go>
- Gunakan Dockerfile :  
<https://github.com/ProgrammerZamanNow/belajar-docker-dockerfile/blob/main/health/Dockerfile> dan hapus bagian HEALTHCHECK nya



# Kode : Health Check

```
container_name: app
build:
  context: "./app"
  dockerfile: Dockerfile
image: "app-golang:1.0.0"
environment:
  - "APP_PORT=8080"
ports:
  - "8080:8080"
healthcheck:
  test: [ "CMD", "curl", "-f", "http://localhost:8080/health" ]
  interval: 5s
  timeout: 5s
  retries: 3
  start_period: 5s
```



## Disable Health Check

- Jika kita tidak mau ada health check, kita juga bisa menonaktifkan nya
- Secara otomatis health check bawaan dari Docker Image nya pun tidak akan diaktifkan
- Cukup di attribute healthcheck, tambahkan attribute disabled: true

---

# Extend Service



# Masalah Banyak File Konfigurasi

- Saat membuat aplikasi menggunakan Docker, kadang kita ingin menjalankan aplikasi tersebut ke beberapa server
- Baik itu di local laptop, di server development, atau server production
- Kadang ada kalanya beberapa hal berbeda, misal konfigurasi misalnya
- Pada kasus ini, mau tidak mau kita harus membuat banyak file konfigurasi Docker Compose, misal untuk di local, di development dan di production



## Extend Service

- Docker Compose memiliki fitur bernama extend service, dimana kita bisa melakukan merge beberapa file konfigurasi sekaligus
- Dengan begitu, kita bisa membuat file konfigurasi umum, dan spesial untuk setiap jenis environment misalnya
- Saat menjalankan Docker Compose, kita bisa gunakan perintah `-f namafile.yaml` jika ingin menggunakan nama file yang bukan `docker-compose.yaml`



# Contoh Program

- Kode main.go : <https://gist.github.com/khannedy/b20b0ef60d2febaf2acda690fff4a57>
- Kode Dockerfile : <https://gist.github.com/khannedy/60d4a23b7f744d12d2d84c63952ae744>



# Kode : Konfigurasi Utama

```
docker-compose.yml x
1  version: "3.9"
2
3  >> services:
4  > app:
5      container_name: app
6      build:
7          context: "./app"
8          dockerfile: Dockerfile
9      image: "app-golang:1.0.0"
10     environment:
11         - "APP_PORT=8080"
12         - "MODE=local"
13     ports:
14         - "8080:8080"
15
```

## Kode : Konfigurasi Dev dan Prod

```
dev.yaml
1  version: "3.9"
2
3  >> services:
4  > app:
5      environment:
6      - "MODE=dev"
7
8
```

```
prod.yaml
1  version: "3.9"
2
3  >> services:
4  > app:
5      environment:
6      - "MODE=prod"
7
8
```



## Kode : Extend Service

```
→ extend-services docker compose -f docker-compose.yaml -f dev.yaml create
[+] Running 2/2
  :: Network extend-services_default Created
  :: Container app Created
→ extend-services docker compose -f docker-compose.yaml -f dev.yaml start
[+] Running 1/1
  :: Container app Started
→ extend-services curl localhost:8080
Hello dev%
→ extend-services
```

---

# Materi Selanjutnya



# Bisa Mulai Belajar

- Integrasikan teknologi yang digunakan dengan Docker
- Belajar Kubernetes