

Docker 基本教學 - 從無到有 Docker-Beginners-Guide 教你用 Docker 建立 Django + PostgreSQL

#docker #django #django-rest-framework #tutorial #postgresql

25 commits

1 branch

0 releases

1 contributor

MIT

Branch: master

New pull request

Create new file

Upload files

Find file

Clone or download

twtrubiks

Fixed #5

Latest commit 6c9bffd on Nov 17 2018

api	django <2.0	a year ago
LICENSE	Initial commit	a year ago
README.md	Fixed #5	3 months ago
docker-compose.yml	better documentation and code	a year ago

## README.md

# docker-tutorial

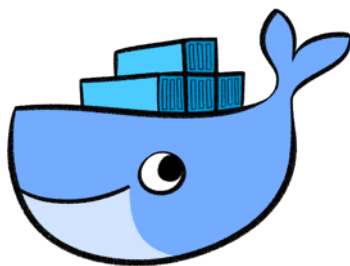
Docker 基本教學 - 從無到有 Docker-Beginners-Guide

教你用 Docker 建立 Django + PostgreSQL

- [Youtube Tutorial PART 1 - Docker 基本教學 - 從無到有 Docker-Beginners-Guide](#)
- [Youtube Tutorial PART 2 - 用 Docker 實戰 Django 以及 Postgre](#)
- [Youtube Tutorial PART 3 - Docker 基本教學 - 透過 portainer 管理 Docker](#)
- [Youtube Tutorial PART 4 - Docker push image to Docker Hub 教學](#)

## 簡介

Docker



**Containers as a Service ( CaaS ) - 容器如同服務**

算是近幾年才開始紅的技術，蠻多公司都有使用 Docker，而且真的很方便，值得大家去了解一下

如果你有環境上不統一的問題？請用 Docker

如果你有每次建立環境都快抓狂的問題？請用 Docker

如果你想要高效率、輕量、秒開的環境，請用 Docker

如果你不想搞死自己，請用 Docker

如果你想潮到出水，請一定要用 Docker

## 什麼是 Docker

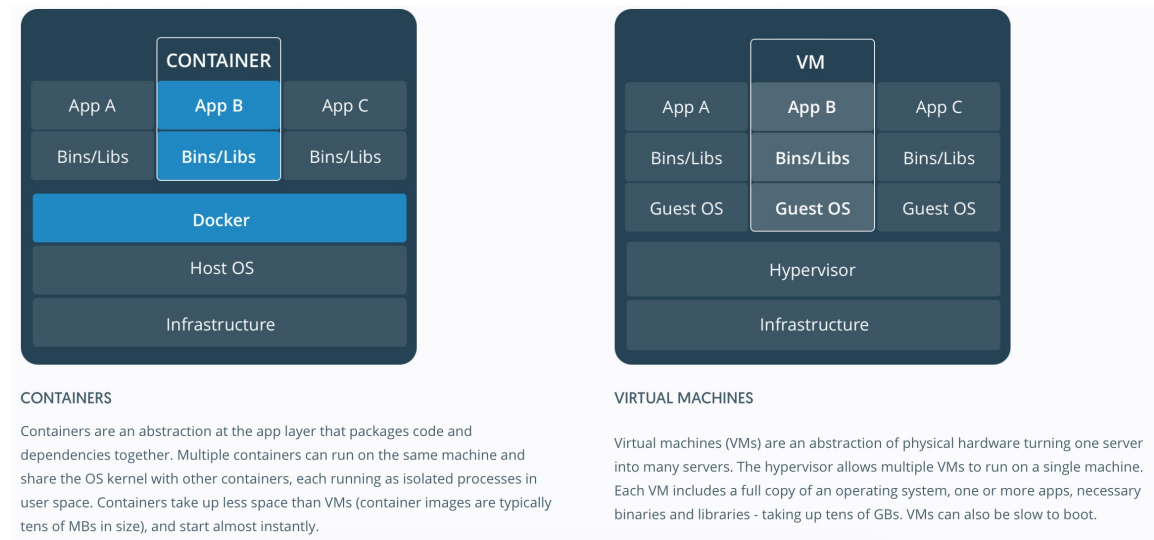
Docker 是一個開源專案，出現於 2013 年初，最初是 Dotcloud 公司內部的 Side-Project。

它基於 Google 公司推出的 Go 語言實作。（ Dotcloud 公司後來改名為 Docker ）

技術原理我們這邊就不提了，簡單提一下他的好處。

我們先來看看官網的說明

### Comparing Containers and Virtual Machines (傳統的虛擬化)



從這張圖可以看出 Containers 並沒有 OS，容量自然就小，而且啟動速度神快

詳細可參考 <https://www.docker.com/what-container>

Virtual Machines 是什麼？

類似 <https://www.virtualbox.org/>，我們可能用它裝裝看其他作業系統，例如說

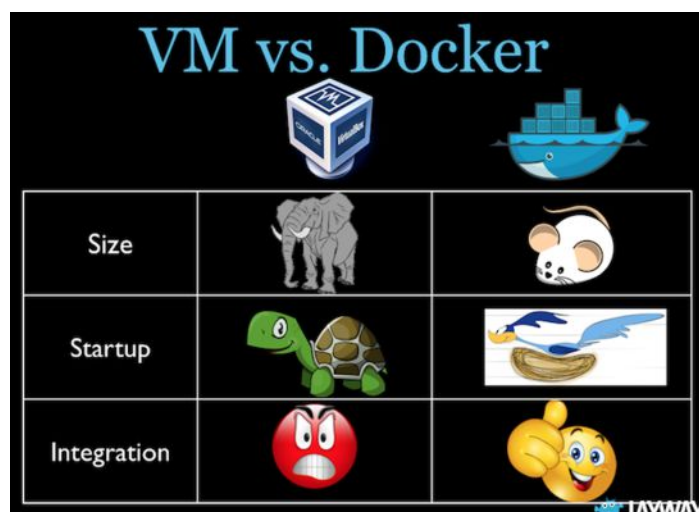
我是 MAC，但我想玩 Windows，我就會在 MAC 中裝 VM 並且灌 Windows 系統。

一個表格了解 Docker 有多棒 🍊

Feature	Containers	Virtual Machines (傳統的虛擬化)
啟動	秒開	最快也要分鐘
容量	MB	GB
效能	快	慢
支援數量	非常多 Containers	10多個就很了不起了
複製相同環境	快	超慢

不理解 ???

我們來看一張圖，包準你懂



圖的來源 <https://blog.jayway.com/2015/03/21/a-not-very-short-introduction-to-docker/>

## 為什麼要使用 Docker

潮 ~ 不解釋 😏

### 更有效率的利用資源

比起像是 <https://www.virtualbox.org/>，Docker 的利用率更高，我們可以設定更多的 Containers，而且啟動速度飛快！！😄

### 統一環境

相信大家都有每次搞電腦的環境都搞的很煩的經驗 😞

假設今天公司來了個新同事，就又要幫他建立一次環境 😐

不然就是，我的電腦 run 起來正常阿～ 你的怎麼不行，是不是 xxx 版本的關係阿 😂

相信大家多多少少都遇過上面這些事情，我們可以透過 Docker 來解決這些問題，

保持大家環境一致，而且要建立的時候也很快 😊

### 對於 DevOps 的好處

對於 DevOps 來說，最希望的就是可以設定一次，將來在其他地方都可以快速建立環境且正常執行。

## Docker 概念

建議大家先了解一下 Docker 中的幾個名詞，分別為

### Image

映像檔，可以把它想成是以前我們在玩 VM 的 Guest OS ( 安裝在虛擬機上的作業系統 )。

Image 是唯讀 ( R\O )

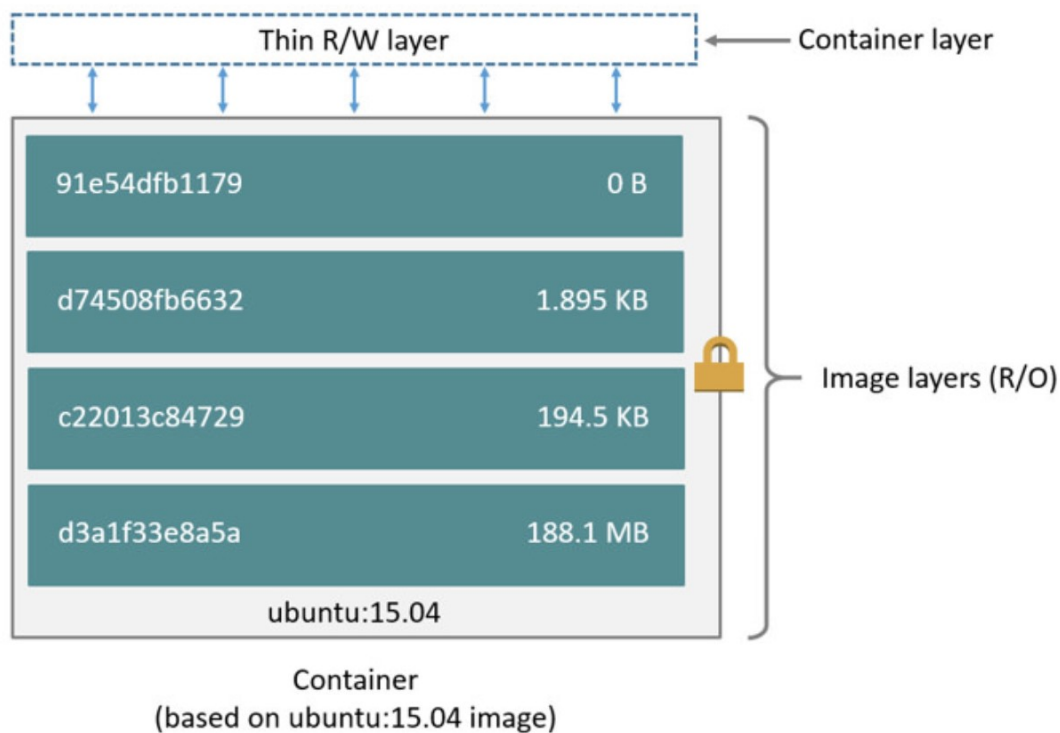
### Container

容器，利用映像檔 ( Image ) 所創造出來的，一個 Image 可以創造出多個不同的 Container，

Container 也可以被啟動、開始、停止、刪除，並且互相分離。

Container 在啟動的時候會建立一層在最外 ( 上 ) 層並且是讀寫模式 ( R\W )。

這張圖解釋了 Image 是唯讀 ( R\O ) 以及 Container 是讀寫模式 ( R\W ) 的關係



更多關係可參考 <https://docs.docker.com/engine/userguide/storagedriver/imagesandcontainers/#images-and-layers>

### Registry

可以把它想成類似 [GitHub](#)，裡面存放了非常多的 Image，可在 [Docker Hub](#) 中查看。

更詳細的我這邊就不再解釋惹，留給大家做功課 😊

## 安裝

### Windows

請先到 [Docker 官網](#)

<https://www.docker.com/docker-windows>

下載 stable 版本

### Docker CE for Windows

Docker CE for Windows is Docker designed to run on Windows 10. It is a native Windows application that provides an easy-to-use development environment for building, shipping, and running dockerized apps. Docker CE for Windows uses Windows-native Hyper-V virtualization and networking and is the fastest and most reliable way to develop Docker apps on Windows. Docker CE for Windows supports running both Linux and Windows Docker containers.

#### Get Docker CE for Windows

**Stable channel**

Stable is the best channel to use if you want a reliable platform to work with. Stable releases track the Docker platform stable releases.

You can select whether to send usage statistics and other data. Stable releases happen once per quarter.

[Get Docker CE for Windows \(stable\)](#)

**Edge channel**

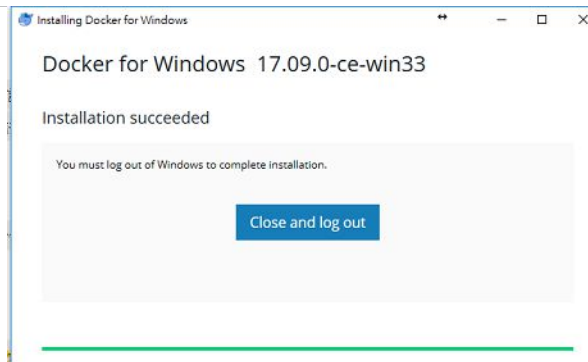
Use the Edge channel if you want to get experimental features faster, and can weather some instability and bugs.

We collect usage data on Edge releases. Edge builds are released once per month.

[Get Docker CE for Windows \(Edge\)](#)

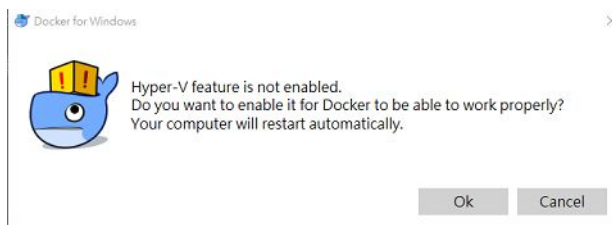
**Install**

接下來就是無腦安裝，安裝完後他會叫你登出電腦，點下去就會幫你登出電腦

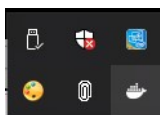


接著如果你的電腦沒有啟用 [Hyper-V](#)，他會叫你重啟電腦（一樣，點下去就對惹）

（更多可 [Hyper-V](https://docs.microsoft.com/zh-tw/virtualization/hyper-v-on-windows/about/) 介紹請參考 <https://docs.microsoft.com/zh-tw/virtualization/hyper-v-on-windows/about/>）

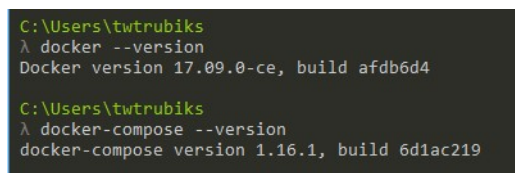


重新開機後，你就會發現可愛的 Docker 在右下角蹦出來惹

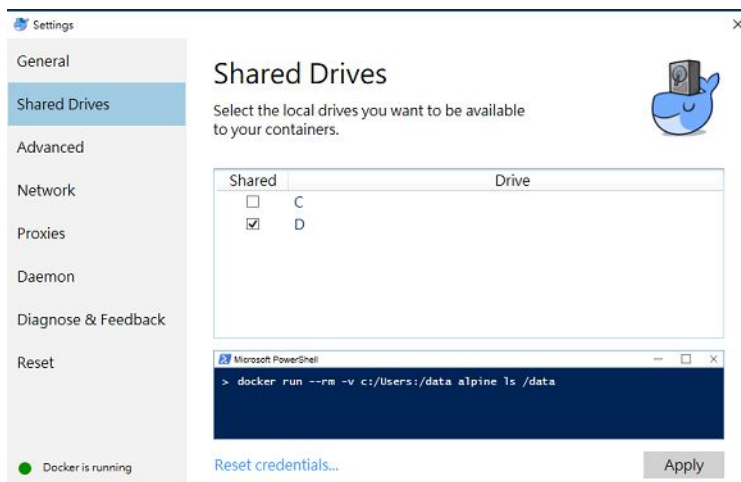


我們可以再用 cmd 確認一下是否有成功安裝

```
docker --version
docker-compose --version
```



記得再設定一個東西 Shared Drives

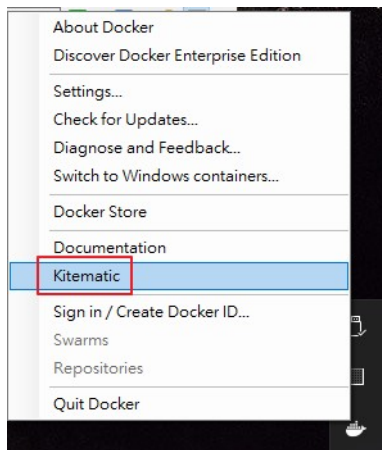


裝完了之後，建議大家再多裝一個 [Kitematic](#)，他是 GUI 介面的，方便你使用 Docker，

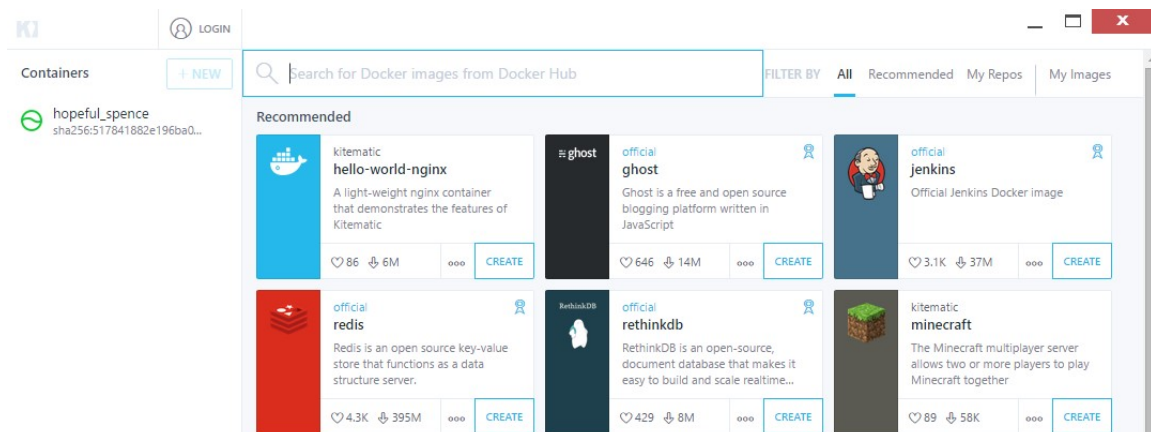
（後面會再介紹一個更贊的 GUI 介面 [portainer](#) 🐳）

我知道打指令很潮，但還是建議裝一下。

直接對著你的 Docker 圖示右鍵，就可以看到 [Kitematic](#)



下載回來直接解壓縮雙點擊即可使用



MAC

MAC 我本身也有，但因為更早之前就裝了，步驟就沒記錄了，基本上大同小異

<https://www.docker.com/docker-mac>

## 指令介紹

接著介紹一些 Docker 的指令，

Docker 的指令真的很多，這裡就介紹我比較常用的或是實用的指令

查看目前 images

```
docker images
```

建立 image

```
docker create [OPTIONS] IMAGE [COMMAND] [ARG...]
```

詳細的參數可參考 <https://docs.docker.com/engine/reference/commandline/create/>

範例 ( 建立一個名稱為 busybox 的 image )

```
docker create -it --name busybox busybox
```

刪除 Image

```
docker rmi [OPTIONS] IMAGE [IMAGE...]
```

查看目前運行的 container

```
docker ps
```

查看目前全部的 container ( 包含停止狀態的 container )

```
docker ps -a
```

新建並啟動 Container

```
docker run [OPTIONS] IMAGE[:TAG|@DIGEST] [COMMAND] [ARG...]
```

舉個例子

```
docker run -d -p 80:80 --name my_image nginx
```

-d 代表在 Detached ( 背景 ) 執行，如不加 -d，預設會 foreground ( 前景 ) 執行

-p 代表將本機的 80 port 的所有流量轉發到 container 中的 80 port

--name 設定 container 的名稱

在舉一個例子

```
docker run -it --rm busybox
```

--rm 代表當 exit container 時，會自動移除 container。( incompatible with -d )

更詳細的可參考 <https://docs.docker.com/engine/reference/run/>

啟動 Container

```
docker start [OPTIONS] CONTAINER [CONTAINER...]
```

( container ID 寫幾個就可以了，和 Git 的概念是一樣的，

不了解 Git 可以參考 [Git-Tutorials GIT基本使用教學](#) )

停止 Container

```
docker stop [OPTIONS] CONTAINER [CONTAINER...]
```

重新啟動 Container

```
docker restart [OPTIONS] CONTAINER [CONTAINER...]
```

刪除 Container

```
docker rm [OPTIONS] CONTAINER [CONTAINER...]
```

--volumes , -v 加上這個參數，會移除掉連接到這個 container 的 volume。

可參考 <https://docs.docker.com/engine/reference/commandline/rm/>

## 進入 Container

```
docker exec [OPTIONS] CONTAINER COMMAND [ARG...]  
docker exec -it <Container ID> bash
```

打指令比較潮，或是說你也可以透過 [Kitematic](#) 進入。

當我們進入了 Container 之後，有時候想看一下裡面 Linux 的版本，

這時候可以使用以下指令查看

```
cat /etc/os-release
```

## 查看 Container 詳細資料

```
docker inspect [OPTIONS] NAME|ID [NAME|ID...]
```

## 查看 log

```
docker logs [OPTIONS] CONTAINER
```

--follow , -f , Follow log output

更詳細的可參考 <https://docs.docker.com/engine/reference/commandline/logs/>

顯示容器資源 ( CPU , I/O ..... )

```
docker stats [OPTIONS] [CONTAINER...]
```

## 停止指定的 CONTAINER 中全部的 processes

```
docker pause CONTAINER [CONTAINER...]
```

執行 docker pause 之後，可以試這用 docker ps 查看，會發現

還是有在執行，這裡拿 docker stop 比較一下，差異如下。

docker stop : process 級別。

docker pause : container 級別。

## 恢復指定暫停的 CONTAINER 中全部的 processes

```
docker unpause CONTAINER [CONTAINER...]
```

## docker tag

```
docker tag SOURCE_IMAGE[:TAG] TARGET_IMAGE[:TAG]
```

## 範例

```
docker tag 0e5574283393 twtrubiks/nginx:version1.0
```

更多可參考 <https://docs.docker.com/engine/reference/commandline/tag/>

## 儲存 image 成 tar 檔案

```
docker save [OPTIONS] IMAGE [IMAGE...]
```



## 範例

```
docker save busybox > busybox.tar
```

更多可參考 <https://docs.docker.com/engine/reference/commandline/save/>

## 載入 image

```
docker load [OPTIONS]
```

## 範例

```
docker load < busybox.tar
```

更多可參考 <https://docs.docker.com/engine/reference/commandline/load/>

## 顯示 image 的 history · 查詢 image 的每一層

```
docker history [OPTIONS] IMAGE
```

在 docker 中，一層一層的概念很重要。

```
λ docker history 883778d18544
IMAGE          CREATED          CREATED BY          SIZE
883778d18544   13 days ago     /bin/sh -c pip install -r requirements.txt  51.5MB
2ad88c90984e   13 days ago     /bin/sh -c #(nop) COPY file:d5c9c783328554...  40B
34584c3daf94   13 days ago     /bin/sh -c #(nop) WORKDIR /docker_api      0B
728e5f1a303b   13 days ago     /bin/sh -c mkdir /docker_api               0B
9329fd7ee9ea   13 days ago     /bin/sh -c #(nop) ENV PYTHONUNBUFFERED=1    0B
1fbc02cc22fb   13 days ago     /bin/sh -c #(nop) LABEL maintainer=twtrubiks 0B
26acbad26a2c   3 months ago    /bin/sh -c #(nop) CMD ["python3"]          0B
<missing>      3 months ago    /bin/sh -c set -ex; wget -O get-pip.py '...  5.23MB
<missing>      3 months ago    /bin/sh -c #(nop) ENV PYTHON_PIP_VERSION=... 0B
<missing>      3 months ago    /bin/sh -c cd /usr/local/bin && ln -s idl...  32B
<missing>      3 months ago    /bin/sh -c set -ex && buildDeps=' dpkg-...  62.3MB
<missing>      3 months ago    /bin/sh -c #(nop) ENV PYTHON_VERSION=3.6.2  0B
<missing>      3 months ago    /bin/sh -c #(nop) ENV GPG_KEY=0D96DF4D411... 0B
<missing>      3 months ago    /bin/sh -c apt-get update && apt-get insta...  8.67MB
<missing>      3 months ago    /bin/sh -c #(nop) ENV LANG=C.UTF-8         0B
<missing>      3 months ago    /bin/sh -c #(nop) ENV PATH=/usr/local/bin... 0B
<missing>      3 months ago    /bin/sh -c set -ex; apt-get update; apt-...  323MB
<missing>      3 months ago    /bin/sh -c apt-get update && apt-get insta...  123MB
<missing>      3 months ago    /bin/sh -c set -ex; if ! command -v gpg >...  0B
<missing>      3 months ago    /bin/sh -c apt-get update && apt-get insta...  44.6MB
<missing>      3 months ago    /bin/sh -c #(nop) CMD ["bash"]             0B
<missing>      3 months ago    /bin/sh -c #(nop) ADD file:d7333b3e0bc6479...  123MB
```

更多可參考 <https://docs.docker.com/engine/reference/commandline/history/>

## 其他指令

### 刪除所有 dangling images

```
docker rmi $(docker images -q -f dangling=true)
docker rmi $(docker images --quiet --filter dangling=true)
```

### 停止所有正在運行的 Container

```
docker stop $(docker ps -q)
```

## Volume

接下來要介紹 Volume · Volume 是 Docker 最推薦存放 persisting data ( 數據 ) 的機制 ·

### 使用 Volume 有下列優點

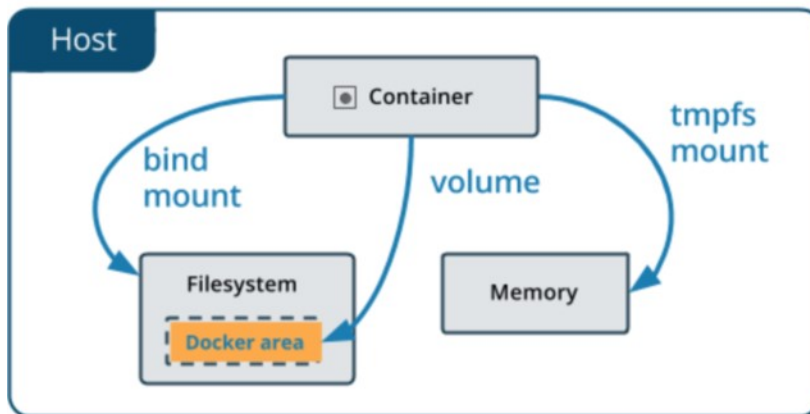
- Volumes are easier to back up or migrate than bind mounts.
- You can manage volumes using Docker CLI commands or the Docker API.
- Volumes work on both Linux and Windows containers.
- Volumes can be more safely shared among multiple containers.

- Volume drivers allow you to store volumes on remote hosts or cloud providers, to encrypt the contents of volumes, or to add other functionality.
- A new volume's contents can be pre-populated by a container.

在 container 的可寫層中，使用 volume 是一個比較好的選擇，因為他不會增加 container 的容量。

volume 的內容存在於 container 之外。

也可參考下圖



更詳細的可參考 <https://docs.docker.com/engine/admin/volumes/volumes/>

查看目前的 volume

```
docker volume ls [OPTIONS]
```

創造一個 volume

```
docker volume create [OPTIONS] [VOLUME]
```

刪除一個 volume

```
docker volume rm [OPTIONS] VOLUME [VOLUME...]
```

查看 volume 詳細資料

```
docker volume inspect [OPTIONS] VOLUME [VOLUME...]
```

移除全部未使用的 volume

```
docker volume prune [OPTIONS]
```

## network

建議大家花點時間研究 docker 中的 network，會蠻有幫助的 😊

查看目前 docker 的網路清單

```
docker network ls [OPTIONS]
```

詳細可參考 <https://docs.docker.com/engine/userguide/networking/>

docker 中的網路主要有三種 Bridge、Host、None，預設皆為 Bridge 模式。

指定 network 範例（指定使用 host 網路）

```
docker run -it --name busybox --rm --network=host busybox
```

## 建立 network

```
docker network create [OPTIONS] NETWORK
```

## 移除 network

```
docker network rm NETWORK [NETWORK...]
```

## 移除全部未使用的 network

```
docker network prune [OPTIONS]
```

## 查看 network 詳細資料

```
docker network inspect [OPTIONS] NETWORK [NETWORK...]
```

## 將 container 連接 network

```
docker network connect [OPTIONS] NETWORK CONTAINER
```

更多詳細資料可參考 [https://docs.docker.com/engine/reference/commandline/network\\_connect/](https://docs.docker.com/engine/reference/commandline/network_connect/)

## Disconnect container network

```
docker network disconnect [OPTIONS] NETWORK CONTAINER
```

更多詳細資料可參考 [https://docs.docker.com/engine/reference/commandline/network\\_disconnect/](https://docs.docker.com/engine/reference/commandline/network_disconnect/)

## User-defined networks

這個方法是官方推薦的 👍

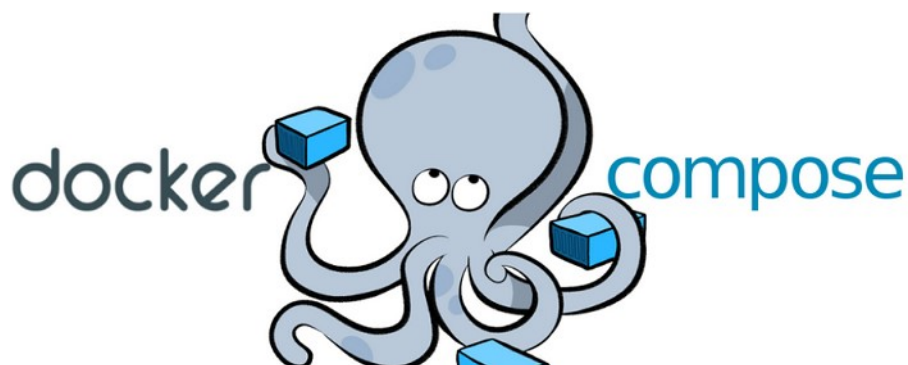
透過內建的 DNS 伺服器，可以直接使用容器名稱，就可解析出 IP，不需要再使用 IP 讓容器互相

溝通，我們只需要知道容器的名稱就可以連接到容器。

更多詳細資料可參考 <https://docs.docker.com/engine/userguide/networking/#user-defined-networks>

## docker-compose

再來要介紹 docker-compose，可參考官網 <https://docs.docker.com/compose/>



Compose 是定義和執行多 Container 管理的工具，不懂我在說什麼 ???

試著想想看，通常一個 Web 都還會有 DB，甚至可能還有 Redis 或 Celery，

所以說我們需要 Compose 來管理這些，透過 docker-compose.yml（YML 格式）文件。

docker-compose.yml 的寫法可參考 <https://docs.docker.com/compose/compose-file/>

也可以直接參考官網範例 <https://docs.docker.com/compose/compose-file/#compose-file-structure-and-examples>

Compose 的 Command-line 很多和 Docker 都是類似的。

可參考 <https://docs.docker.com/glossary/?term=compose>

查看目前 Container

```
docker-compose ps
```

加上 `-q` 的話，只顯示 id

```
docker-compose ps -q
```

啟動 Service 的 Container

```
docker-compose start [SERVICE...]
```

停止 Service 的 Container ( 不會刪除 Container )

```
docker-compose stop [options] [SERVICE...]
```

重啟 Service 的 Container

```
docker-compose restart [options] [SERVICE...]
```

Builds, (re)creates, starts, and attaches to containers for a service

```
docker-compose up [options] [--scale SERVICE=NUM...] [SERVICE...]
```

加個 `-d`，會在背景啟動，一般建議正式環境下使用。

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

up 這個功能很強大，建議可以參考 <https://docs.docker.com/compose/reference/up/>

docker-compose down

```
docker-compose down [options]
```

down 這個功能也建議可以參考 <https://docs.docker.com/compose/reference/down/>

舉個例子

```
docker-compose down -v
```

加個 `-v` 就會順便幫你把 volume 移除 ( 移除你在 `docker-compose.yml` 裡面設定的 volume )

在指定的 Service 中執行一個指令

```
docker-compose run [options] [-v VOLUME...] [-p PORT...] [-e KEY=VAL...] SERVICE [COMMAND] [ARGS...] [ARGS...]
```

舉個例子

```
docker-compose run web bash
```

在 web Service 中執行 `bash` 指令

可參考 <https://docs.docker.com/compose/reference/run/>

觀看 Service logs

```
docker-compose logs [options] [SERVICE...]
```

檢查 `docker-compose.yml` 格式是否正確

```
docker-compose config
```

如下指令，和 `docker exec` 一樣

```
docker-compose exec
```

範例 ( 進入 web 這個 service 的 bash )

```
docker-compose exec web bash
```

顯示被使用到的 container 中的 images 清單

```
docker-compose images
```

移除 service containers

```
docker-compose rm
```

Pushes images 到 docker hub

```
docker-compose push
```

目前這個指令其實我也搞不太懂，可參考 <https://github.com/docker/compose/issues/4283>

官網也解釋的沒有很清楚 <https://docs.docker.com/compose/reference/push/>

## Docker Registry

---



可以把它想成是一個類似 github 的地方，只不過裡面變成是存 docker 的東西，當然，

也可以自己架，但會有一些額外的成本，像是網路，維護等等，這部分就要自己衡量了 🤖

接下來教大家如何將 image push 到 Docker Registry 😊

- [Youtube Tutorial PART 4 - Docker push image to Docker Hub 教學](#)

首先，先登入 [Docker Registry](#) ( 註冊流程很簡單，我就跳過了 )

```
docker login
```

```
λ docker login
Login with your Docker ID to push and pull images
Username: twtrubiks
Password:
Login Succeeded
```

舉個例子，先 run 一個 busybox 的容器

```
docker run -it busybox
```

接著在裡面新增一筆資料

```
echo 'text' > data.txt
```

```
λ docker run -it busybox
/ # ls
bin    dev    etc    home   proc   root   sys    tmp     usr     var
/ # echo 'text' > data.txt
/ # ls
bin      data.txt  dev      etc      home     proc
```

然後打開另一個 terminal，使用 `docker ps` 查看目前容器的 id

```
λ docker ps
CONTAINER ID   IMAGE          COMMAND         CREATED        STATUS
4fb4ef51e917   busybox        "sh"           4 minutes ago  Up 4 minutes
c9/c91e51f64   portainer/portainer "/portainer"    / weeks ago   Up 8 hours
```

再來使用像 git 一樣的方式 commit

`docker commit`

```
docker commit [OPTIONS] CONTAINER [REPOSITORY[:TAG]]
```

可參考 <https://docs.docker.com/engine/reference/commandline/commit/>

```
docker commit -m "test" 4fb4ef51e917 twtrubiks/my_busybox
```

`-m` commit message，和 git 一樣。

twtrubiks/my\_busybox 則為我們定義的 REPOSITORY。

這時候可以用 `docker images` 查看

```
λ docker images
REPOSITORY          TAG          IMAGE ID          CREATED          SIZE
twtrubiks/my_busybox latest       0b3d286cc743     3 minutes ago   1.13MB
```

最後 push

```
docker push twtrubiks/my_busybox
```

```
λ docker push twtrubiks/my_busybox
The push refers to a repository [docker.io/twtrubiks/my_busybox]
0b95d1d77799: Pushing [=====>] 3.584kB
0271b8eebde3: Preparing
```


docker 是一層一層的概念，他只會 push 自己新增的幾層上去而已，

所以不用擔心整個 image 很大，要上傳很久 😊

最後可以到 <https://hub.docker.com/> 確認是否有成功 😊

## Repositories

Type to filter repositories by name

 <b>twtrubiks/my_busybox</b> public	0 STARS	1 PULLS	> DETAILS
---	------------	------------	--------------

## 用 Docker 實戰 Django 以及 Postgre

- [Youtube Tutorial PART 2 - 用 Docker 實戰 Django 以及 Postgre](#)

上面介紹了那麼多，來實戰一下是必須的 😊

我們使用 [Django-REST-framework 基本教學 - 從無到有 DRF-Beginners-Guide](#) 來當範例

有幾個地方必須修改一下，

將 `settings.py` 裡面的 db 連線改成 [PostgreSQL](#)

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'postgres',
        'USER': 'postgres',
        'PASSWORD': 'password123',
        'HOST': 'db',
        'PORT': 5432,
    }
}
```

建議也將 `ALLOWED_HOSTS = []` 改為 `ALLOWED_HOSTS = ['*']`（這只是方便，實務上不會這樣使用）

再來是兩個很重要的檔案，分別為 `Dockerfile` 和 `docker-compose.yml`

`Dockerfile`

```
FROM python:3.6.2
LABEL maintainer twtrubiks
ENV PYTHONUNBUFFERED 1
RUN mkdir /docker_api
WORKDIR /docker_api
COPY . /docker_api/
RUN pip install -r requirements.txt
```

詳細可參考 <https://docs.docker.com/engine/reference/builder/>

`docker-compose.yml`

```
version: '3'
services:

  db:
    container_name: 'postgres'
    image: postgres
    environment:
      POSTGRES_PASSWORD: password123
    ports:
      - "5432:5432"
      # (HOST:CONTAINER)
    volumes:
      - pgdata:/var/lib/postgresql/data/

  web:
    build: ./api
    command: python manage.py runserver 0.0.0.0:8000
```

```

restart: always
volumes:
  - api_data:/docker_api
    # (HOST:CONTAINER)
ports:
  - "8000:8000"
    # (HOST:CONTAINER)
depends_on:
  - db

volumes:
  api_data:
  pgdata:

```

詳細可參考 <https://docs.docker.com/compose/compose-file/#compose-file-structure-and-examples>

溫馨小提醒 1 ❤️

可能有人會問為什麼我是使用 0.0.0.0，而不是使用 127.0.0.1 ??

```
python manage.py runserver 0.0.0.0:8000
```

127.0.0.1，並不代表真正的本機，我們經常認為他是本機是因為我們電腦的 host 預設都幫你設定好了 😊

詳細的 host 設定教學可參考 [hosts-設定檔](#) 以及 [查詢內網 ip](#)。

0.0.0.0 才是真正的代表，當下（本）網路中的本機 ✍️

如果大家想更深入的了解，可 google 再進一步的了解 127.0.0.1 以及 0.0.0.0 的差異 😊

溫馨小提醒 2 ❤️

這邊要特別提一下 depends\_on 這個參數，

詳細可參考 [https://docs.docker.com/compose/compose-file/#depends\\_on](https://docs.docker.com/compose/compose-file/#depends_on)。

上面連結中有一段說明很值得看

**depends\_on does not wait for db and redis to be 「ready」 before starting web - only until they have been started. If you need to wait for a service to be ready, see Controlling startup order for more on this problem and strategies for solving it.**

以我的 [docker-compose.yml](#) 為例，啟動順序雖然為 db -> web，但他不會等待 db 啟動完成後才啟動 web。

也就是說，還是有可能 web 比 db 先啟動完成，這樣就需要重啟 web service，否則會無法連上 db 😭

如果真的要控制啟動順序，請參考 [Controlling startup order](#)。

溫馨小提醒 3 ❤️

[docker-compose.yml](#) 其實使用 `docker run` 也是可以完成的，例如這個範例中，如果使用

`docker run` 來寫，會變成這樣。

首先，為了讓容器彼此可以溝通，我們先建立一個網路 (User-defined networks)。

```
docker network create my_network
```

db 容器

```
docker run --name db -v pgdata:/var/lib/postgresql/data/ -p 5432:5432 --network=my_network -e POSTGRES_PASSWORD=password
```

接下來先去 api 資料夾中 build 出 image

```
docker build --tag web_image .
```

--tag, -t, tag 這個 image 名稱為 web\_image

web 容器



```
docker run --name web -v api_data:/docker_api -p 8000:8000 --network=my_network --restart always web_image python mana
```

以上這樣，和 docker-compose.yml 其實是一樣的 😊

設定完了之後，接下來我們就可以啟動他了

```
docker-compose up
```

接下來你會看到類似的畫面

```
D:\temp\docker (master)
λ docker-compose up
Pulling db (postgres:latest)...
latest: Pulling from library/postgres
3e17c6eae66c: Downloading [=====>] 29.35MB/45.13MB
3d89ae9a47e4: Download complete
f7726fda7efe: Download complete
d1838499bd8f: Download complete
a5ec5aa60735: Downloading [=====>] 5.815MB/6.578MB
1571d7170291: Download complete
0d6e41e13732: Download complete
787e3c45a9bb: Waiting
7b234cf83b22: Waiting
3a8ad2440289: Waiting
c58cd00fd1b1: Waiting
ff781a2b3014: Waiting
5dc973df69: Waiting
```

```
Status: Downloaded newer image for python:3.6.2
--> 26acbad26a2c
Step 2/8 : LABEL maintainer twtubiks
--> Running in 02a0f8815ec1
--> bee318e1a920
Removing intermediate container 02a0f8815ec1
Step 3/8 : ENV PYTHONUNBUFFERED 1
--> Running in a4fdc8afb2f
--> 55406561e031
Removing intermediate container a4fdc8afb2f
Step 4/8 : RUN mkdir /code
--> Running in a012ba9a1a52
--> 79b09e123715
Removing intermediate container a012ba9a1a52
Step 5/8 : WORKDIR /code
--> d6d518b50b79
Removing intermediate container 1322feb69e9e
Step 6/8 : COPY requirements.txt /code/
--> 517841882e19
Step 7/8 : RUN pip install -r requirements.txt
--> Running in 5c35ccc38dcb
Collecting django (from -r requirements.txt (line 1))
  Downloading Django-1.11.6-py2.py3-none-any.whl (6.9MB)
```

假如你出現了類似的畫面

```
web_1 | File "/usr/local/lib/python3.6/site-packages/django/db/backends/base/base.py", line 213, in ens
ure_connection
web_1 |     self.connect()
web_1 | File "/usr/local/lib/python3.6/site-packages/django/db/backends/base/base.py", line 189, in con
nect
web_1 |     self.connection = self.get_new_connection(conn_params)
web_1 | File "/usr/local/lib/python3.6/site-packages/django/db/backends/postgresql/base.py", line 176,
in get_new_connection
web_1 |     connection = Database.connect(**conn_params)
web_1 | File "/usr/local/lib/python3.6/site-packages/psycopg2/__init__.py", line 130, in connect
web_1 |     conn = _connect(dsn, connection_factory=connection_factory, **kwargs)
web_1 | django.db.utils.OperationalError: could not connect to server: Connection refused
web_1 |         Is the server running on host "db" (172.18.0.2) and accepting
web_1 |         TCP/IP connections on port 5432?
web_1 |
db_1 | syncing data to disk ... ok
```

代表 database 還在建立的時候，你的 web ( Django ) 就去連接他，

所以導致連接不上，這時候我們可以先終止他 ( 按 Ctrl+C )

接著在重新 docker-compose up

我們成功啟動了 ( db 連線也正常 )

```

D:\temp\docker\docker_tutorial (master)
λ docker-compose up
Starting dockertutorial_db_1 ...
Starting dockertutorial_db_1 ... done
Starting dockertutorial_web_1 ...
Starting dockertutorial_web_1 ... done
Attaching to dockertutorial_db_1, dockertutorial_web_1
db_1 | 2017-10-14 09:20:21.377 UTC [1] LOG: listening on IPv4 address "0.0.0.0", port 5432
db_1 | 2017-10-14 09:20:21.378 UTC [1] LOG: listening on IPv6 address ":::", port 5432
db_1 | 2017-10-14 09:20:21.387 UTC [1] LOG: listening on Unix socket "/var/run/postgresql/..
db_1 | 2017-10-14 09:20:21.410 UTC [20] LOG: database system was shut down at 2017-10-14 09
db_1 | 2017-10-14 09:20:21.418 UTC [1] LOG: database system is ready to accept connections
web_1 | Performing system checks...
web_1 |
web_1 | System check identified no issues (0 silenced).
web_1 |
web_1 | You have 13 unapplied migration(s). Your project may not work properly until you apply
s for app(s): admin, auth, contenttypes, sessions.
web_1 | Run 'python manage.py migrate' to apply them.
web_1 | October 14, 2017 - 09:20:24
web_1 | Django version 1.11.6, using settings 'django_rest_framework_tutorial.settings'
web_1 | Starting development server at http://0.0.0.0:8000/
web_1 | Quit the server with CONTROL-C.

```

但你仔細看上圖，你會發現他說你還沒 migrate

接下來我們開啟另一個 cmd 進入 web 的 service。

透過剛剛介紹的指令進入 service

```

docker ps
docker exec -it <Container ID> bash

```

或是說也可以從 [Kitematic](#) 進入。

進入後我們可以開始 migrate

```

python manage.py makemigrations musics
python manage.py migrate

```

```

D:\temp\docker\docker_tutorial (master)
λ docker exec -it 8dfb bash
root@8dfb66654a83:/code# python manage.py makemigrations musics
Migrations for 'musics':
  musics/migrations/0001_initial.py
    - Create model Music
root@8dfb66654a83:/code# python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, musics, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK

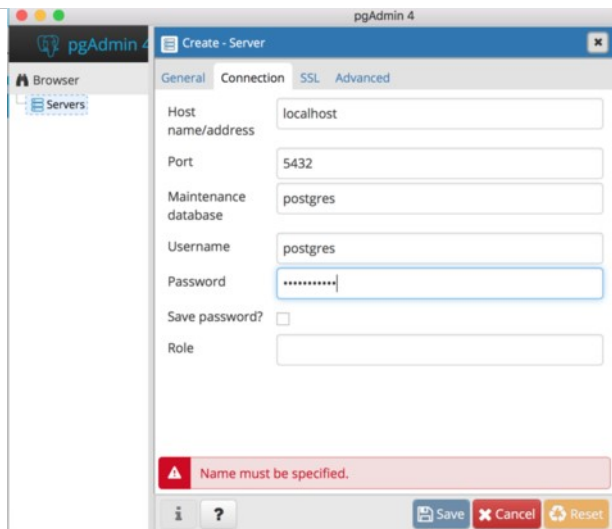
```

順便在建立一個 superuser

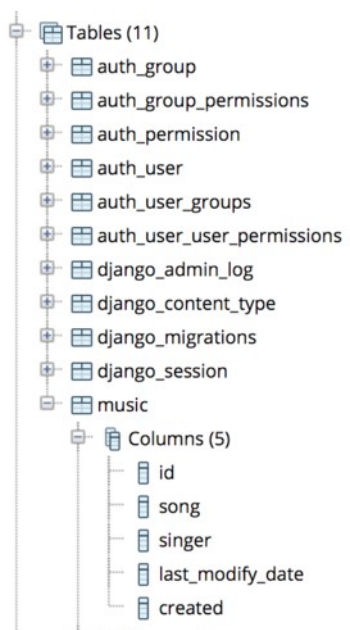
```
python manage.py createsuperuser
```

接著我們可以試著使用 GUI 介紹連接 db。

因為我們是用 PostgreSQL，可以透過 [pgadmin](#) 連線



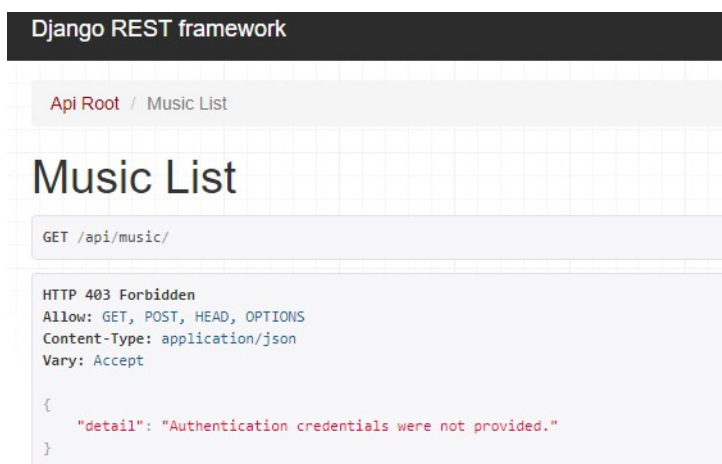
我們剛剛 migrate 的東西確實有存在



我們不需要再重新啟動

直接可以開開心心的去瀏覽 <http://127.0.0.1:8000/api/music/>

大家一定會看到很熟悉的畫面



接著依照自己剛剛設定的帳密登入進去即可

# Music List

[OPTIONS](#)[GET](#)

POST /api/music/

HTTP 201 Created  
Allow: GET, POST, HEAD, OPTIONS  
Content-Type: application/json  
Vary: Accept

```
{
  "id": 1,
  "song": "222",
  "singer": "222",
  "last_modify_date": "2017-10-14T09:56:17.444397Z",
  "created": "2017-10-14T09:56:17.444414Z",
  "days_since_created": 0
}
```

	id [PK] serial	song text	singer text	last_modify_date timestamp with time zone	created timestamp with time zone
1	1	222	222	2017-10-14 09:56:17.444397+00	2017-10-14 09:56:17.444414+00

以上整個環境，都是在 Docker 中 🙄

如果我們再 Ctrl+C 退出，重新啟動一次 `docker-compose up`

這次就不會再和你說你沒有 migrate 了

```
D:\temp\docker\docker_tutorial (master)
λ docker-compose up
Starting dockertutorial_db_1 ...
Starting dockertutorial_db_1 ... done
Starting dockertutorial_web_1 ...
Starting dockertutorial_web_1 ... done
Attaching to dockertutorial_db_1, dockertutorial_web_1
db_1 | 2017-10-14 09:39:32.964 UTC [1] LOG: listening on IPv4 address "0.0.0.0", port 5432
db_1 | 2017-10-14 09:39:32.964 UTC [1] LOG: listening on IPv6 address ":::", port 5432
db_1 | 2017-10-14 09:39:32.969 UTC [1] LOG: listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
db_1 | 2017-10-14 09:39:32.990 UTC [20] LOG: database system was shut down at 2017-10-14 09:38:48 UTC
db_1 | 2017-10-14 09:39:33.000 UTC [1] LOG: database system is ready to accept connections
web_1 | Performing system checks...
web_1 |
web_1 | System check identified no issues (0 silenced).
web_1 | October 14, 2017 - 09:39:34
web_1 | Django version 1.11.6, using settings 'django_rest_framework_tutorial.settings'
web_1 | Starting development server at http://0.0.0.0:8000/
web_1 | Quit the server with CONTROL-C.
```

## 其他管理 Docker GUI 的工具

- [Youtube Tutorial PART 3 - Docker 基本教學 - 透過 portainer 管理 Docker](#)

除了 [Kitematic](#) 之外，還有其他不錯的推薦給大家，

這次要介紹的就是 [portainer](#) 功能強大又好用 🔥

其實如果去看看 [Kitematic](#) 以及 [portainer](#) 的 github，

你會發現 [portainer](#) 感覺比較有在 maintenance 😊

而且我使用了 [portainer](#) 之後，真心大推 😄

安裝方法可參考 <https://portainer.io/install.html>

```
docker volume create portainer_data
docker run --name=portainer -d -p 9000:9000 -v /var/run/docker.sock:/var/run/docker.sock -v portainer_data:/data porta
```

< >

-d -p 在前面的 `docker run` 有介紹過代表的含意，--name 只是命名而已。

Note 1 : The -v /var/run/docker.sock:/var/run/docker.sock option is available on Linux environments only.

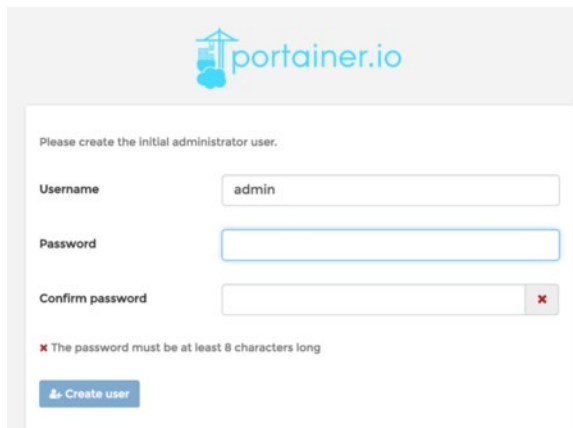
Note 2 : The -v portainer\_data:/data portainer/portainer option will persist Portainer data in portainer\_data on the host where Portainer is running. You can specify another location on your filesystem.

( 建立起來之後，就依照 container 的操作即可 )



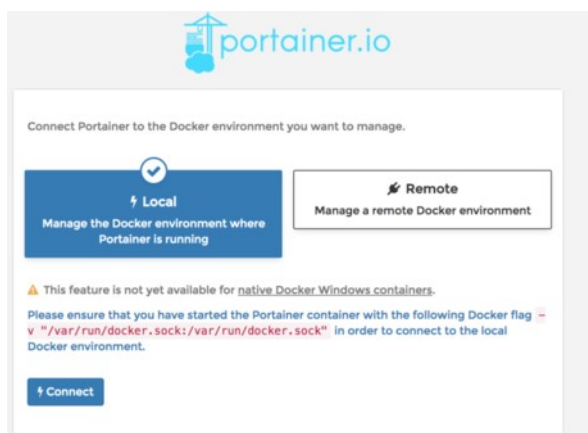
之後查看 <http://localhost:9000/> 就會看到下圖

然後設定帳號、密碼



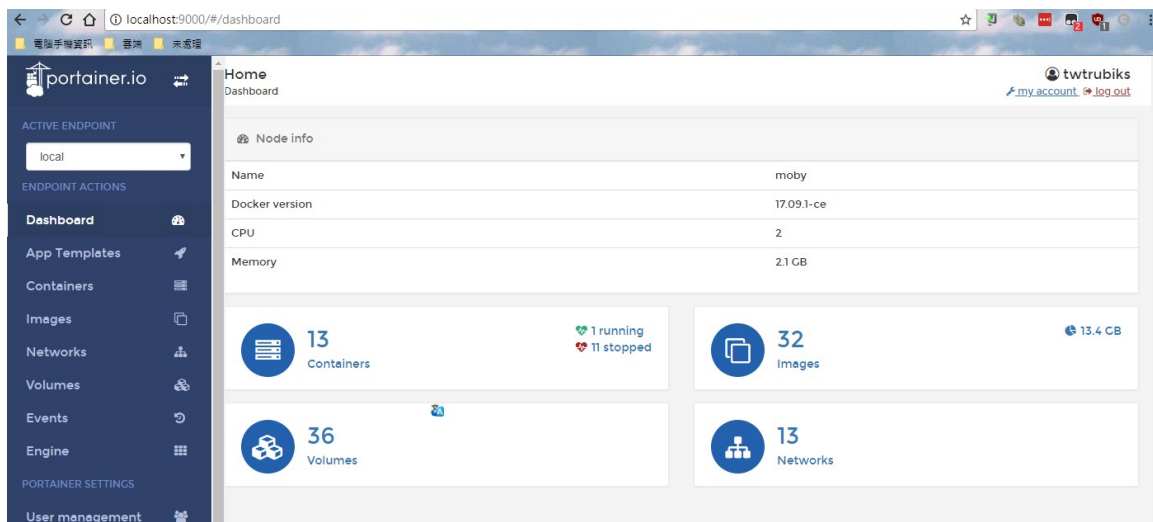
The image shows the Portainer.io user creation interface. At the top is the Portainer.io logo. Below it, a message says "Please create the initial administrator user." There are three input fields: "Username" with the value "admin", "Password" (empty), and "Confirm password" (empty). A red error message below the password fields states: "The password must be at least 8 characters long". At the bottom is a blue button labeled "Create user".

選 Local or Remote



The image shows the Portainer.io connection selection screen. At the top is the Portainer.io logo. Below it, a message says "Connect Portainer to the Docker environment you want to manage." There are two main options: "Local" (selected) and "Remote". The "Local" option is highlighted in blue and says "Manage the Docker environment where Portainer is running". The "Remote" option says "Manage a remote Docker environment". Below these options, a warning message states: "This feature is not yet available for native Docker Windows containers." A note below that says: "Please ensure that you have started the Portainer container with the following Docker flag - v '/var/run/docker.sock:/var/run/docker.sock' in order to connect to the local Docker environment." At the bottom is a blue button labeled "Connect".

畫面真的不錯看，而且資訊也很豐富 🤖



相信我，你使用完他之後，你會默默的邊緣化 Kitematic 🤖

## 查看 port 佔用狀況

這個推薦給大家，有時候會遇到 port 被佔用，用指令查比較方便

Windows

查看所有 port 的佔用狀況

```
netstat -ano
```

查看指定 port 的佔用狀況，例如現在想要查看 port 5432 佔用的狀況

```
netstat -aon|findstr "5432"
```

查看 PID 對應的 process

```
tasklist|findstr "2016"
```

停止 PID 為 6093 的 process

```
taskkill /f /PID 6093
```

停止 vscode.exe process

```
taskkill /f /t /im vscode.exe
```

MAC

將 port 為 8000 的 process 全部停止

```
sudo lsof -t -i tcp:8000 | xargs kill -9
```

查看指定 port 的佔用狀況，例如現在想要查看 port 5432 佔用的狀況

```
lsof -i tcp:5432
```

## 後記：

Docker 算是我最近才開始接觸的，所以也算是新手，如果我有任何講錯的，歡迎和我說，我會再修改 🙏

Docker 可以玩的真的很多，延伸閱讀

- [實戰 Docker + Jenkins + Django + Postgres](#) 📄 - 結合 Jenkins
- [Docker + Django + Nginx + uWSGI + Postgres 基本教學 - 從無到有](#)
- [實戰 Docker + Django + Nginx + uWSGI + Postgres - Load Balance](#) 📄

也可以再玩玩 **Docker Swarm** (分散式系統) 😞

- [Docker Swarm 基本教學 - 從無到有 Docker-Swarm-Beginners-Guide](#) 📄

最後，希望大家在學習 Docker 的過程中，遇到不懂的，可以去找資料並且了解他，順便補足一些之前不足的知識。

## 執行環境

- Mac
- Python 3.6.2
- windows 10

## Reference

- <https://docs.docker.com/>
- [portainer](#)

## Donation

文章都是我自己研究內化後原創，如果有幫助到您，也想鼓勵我的話，歡迎請我喝一杯咖啡 ☕



贊助者付款

## License

---

MIT license