

# Docker 講義

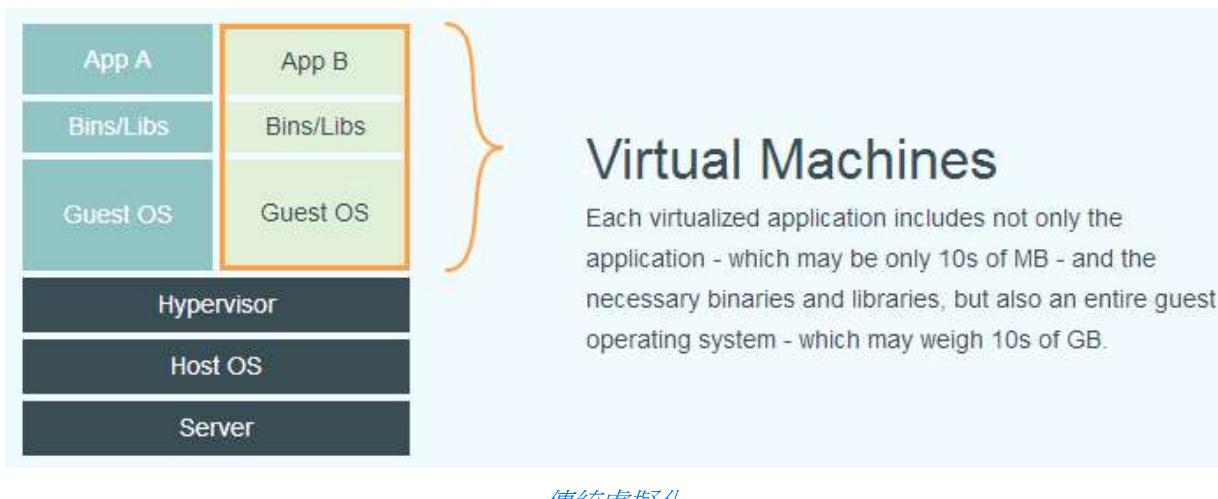
## VM與Docker的基本概念

### VM

傳統VM存在的問題

1. VM開啟速度較慢
2. VM閒置時亦會佔用資源
3. VM容易被更改 · 且不會被記錄（即使紀錄也不容易查詢）

傳統虛擬化(VM)架構



### Docker

什麼是容器化技術

引用AWS對於容器化技術的解釋：

- 容器是一種作業系統虛擬化方法
- 可讓您在資源隔離的程序中執行應用程式及其相依性。
- 容器可讓您輕鬆地將應用程式的程式碼、組態和相依性封裝成易於使用的建置區塊
- 提供環境一致性、操作效率、開發人員生產力和版本控制。
- 無論部署環境為何，容器都可協助確保以快速、可靠和一致的方法部署應用程式。
- 容器還能讓您對資源進行更精細地控制，使基礎設施更有效率。

容器化技術的優勢：

- 保持環境一致性
- 較好的操作效率
- 提高開發人員生產力

- 容易進行版本控制

## 容器化技術(Docker)架構

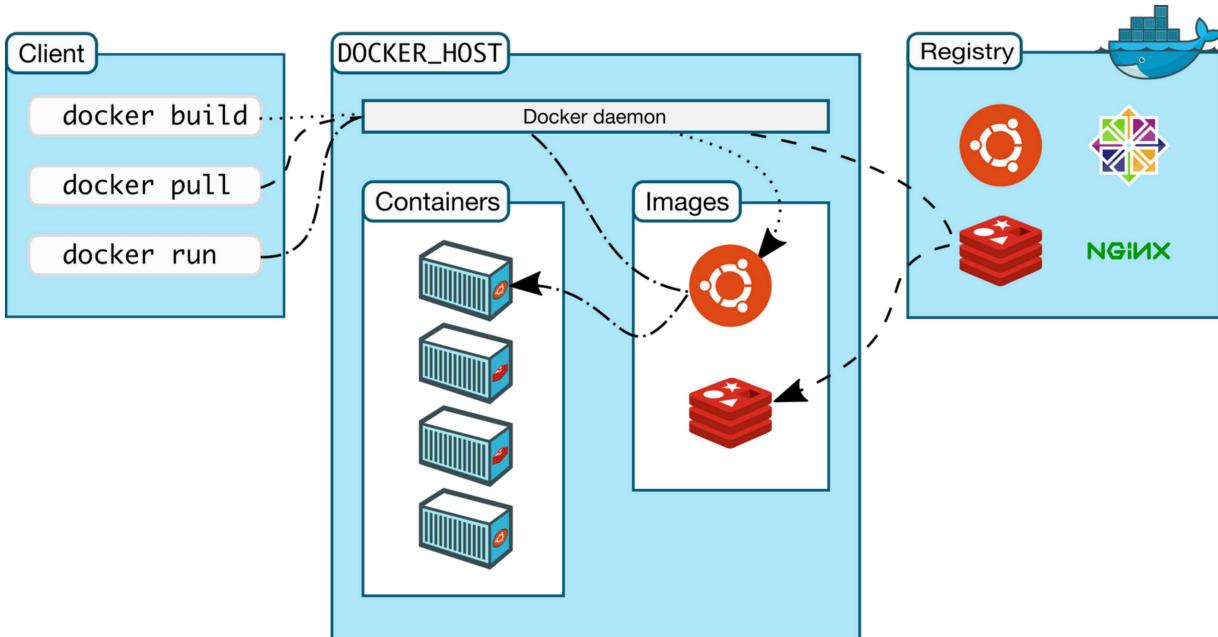


*Docker*

## Docker的三個基本觀念

- 映像檔 (Image)
- 容器 (Container)
- 倉庫 (Repository)

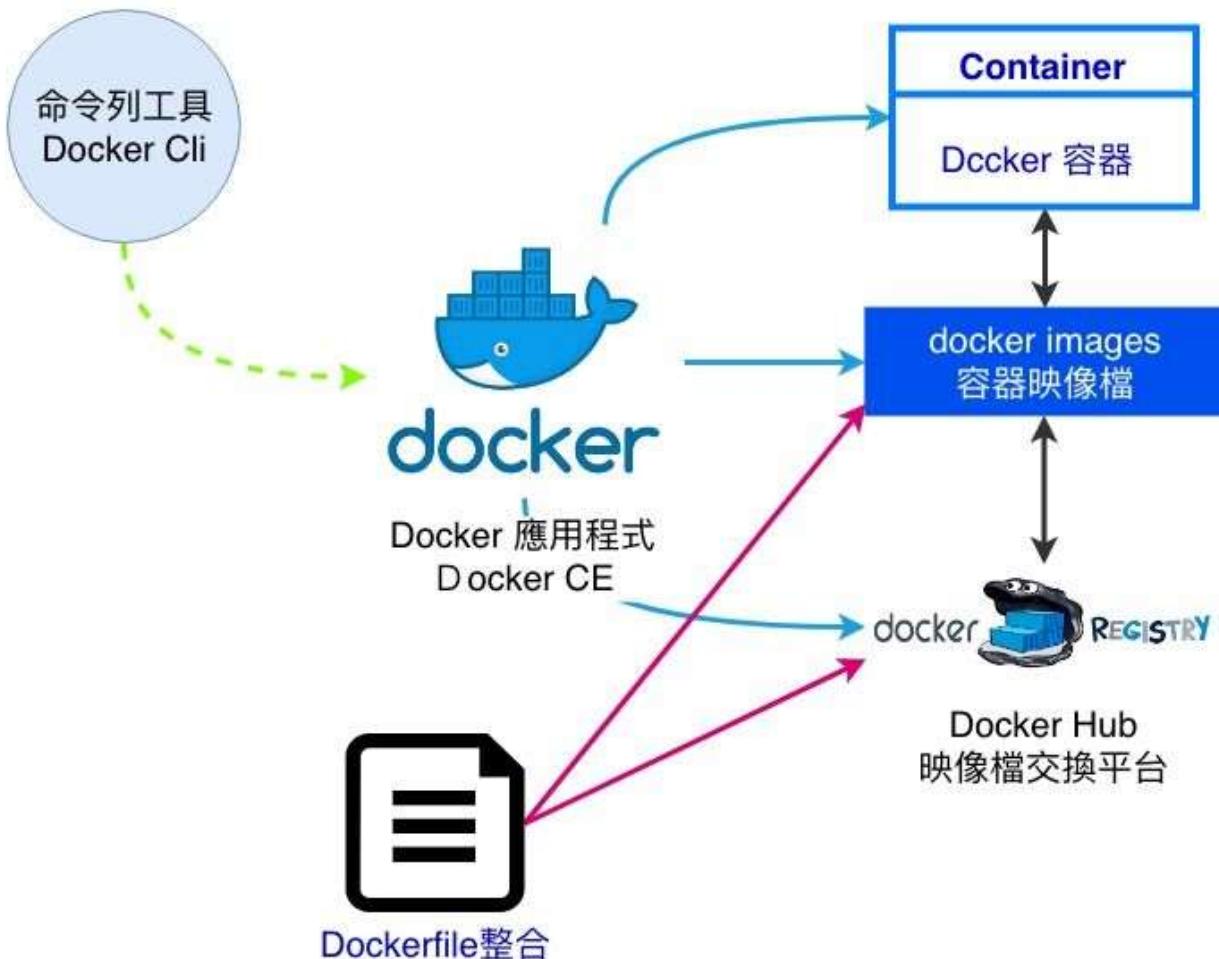
## docker的基本架構



<https://docs.docker.com/engine/docker-overview/#docker-architecture>

## Docker應用流程與架構

## Docker 應用流程架構



**Docker 和虛擬機(VM)的區別與特點**

### 對比傳統虛擬機總結

特性	容器	虛擬機
啟動	秒級	分鐘級
硬碟容量	一般為 MB	一般為 GB
效能	接近原生	比較慢
系統支援量	單機支援上千個容器	一般幾十個

[https://philipzheng.gitbooks.io/docker\\_practice/content/introduction/why.html](https://philipzheng.gitbooks.io/docker_practice/content/introduction/why.html)

## Docker Cli 基本應用

## 環境設置

### VM虛擬環境安裝及操作

1. 下載virtualbox(windows)並安裝
2. 匯入ova檔案
3. 確認有一張網卡2為Host-only(僅限主機)



4. 確認全域工具host-only位址為192.168.56.1



5. VM開機

### MobaXterm下載及使用

#### MobaXterm下載

[https://download.mobatek.net/1102018093083521/MobaXterm\\_Portable\\_v11.0.zip](https://download.mobatek.net/1102018093083521/MobaXterm_Portable_v11.0.zip)

ssh連入192.168.56.111

```
1 ssh iii@192.168.56.111
2 密碼為iii
```

```
ping - iii@localhost:~ — ssh iii@192.168.56.111 — 80x24
Pingde-MacBook-Pro:~ ping$ ssh iii@192.168.56.111
iii@192.168.56.111's password:
Last login: Thu Dec  6 19:52:54 2018 from 192.168.56.1
[iii@localhost ~]$
```

## Docker安裝

安裝方式可以參考官方文件：<https://docs.docker.com/install/linux/docker-ce/centos/>

本範例以Centos 進行操作

切換到docker-tutorial/step0資料夾

```
cd docker-tutorial/step0
```

執行dockerinstall.sh

```
./dockerinstall.sh
```

dockerinstall.sh

```
1 #從docker官方下載安裝腳本檔並執行
2 curl -fsSL get.docker.com -o get-docker.sh
3 sudo sh get-docker.sh
4 #安裝完成，開啟(start)及預設開啟(enable)docker服務
5 sudo systemctl start docker.service
6 sudo systemctl enable docker.service
7 #將現在使用者加入docker群組，具備操作docker的權限(不用sudo)
8 sudo usermod -aG docker `whoami`
```

完成後exit退出連線並重新登入

## Docker Cli 操作示範

### Step1: Docker Hello World

```
1 # 所有的學習都是從hello-world開始的(笑)
2 # 開始Docker的hello-world
3 # 試試T一下下面的指令：
4 docker run hello-world
5 #如果你非常懶惰的話...
```

6 ./step1.sh

```
[iii@localhost iii-network-engineer-tutorial]$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
d1725b59e92d: Pull complete
Digest: sha256:0add3ace90ecb4adbf7777e9aacf18357296e799f81cabc9fde470971e499788
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/
```

## docker run

### Step2: 第一次docker run

試想：我們今天需要一個python3.7的測試環境，但是本機只有python2.7，你可以怎麼做？

(A)升級本機的python環境→可能導致很多與python相依的服務(例如yum...等)錯誤，修改設定麻煩。

(B)用pipenv創立虛擬環境→設定繁瑣，還是很麻煩。

(C)直接用Docker開一個Container，測試完畢後關閉並刪除→開啟方便快速，用完即刪，選我選我。

現在，讓我們來嘗試一下

用docker快速開啟一個python3.7的環境

```
1 docker run -it --rm python:3.7 /bin/bash
2 #謎之音：docker 我需要一個python3.7的環境，我需要交互模式進行互動(-it)，用完即丟(--rm)，環境開啟之後進到/bin/bash。
3 #再次提供偷懶腳本
4 ./step2.sh
```

```
[iii@localhost step1]$ docker run -it python:3.7 /bin/bash
Unable to find image 'python:3.7' locally
3.7: Pulling from library/python
54f7e8ac135a: Pull complete
d6341e30912f: Pull complete
087a57faf949: Pull complete
5d71636fb824: Pull complete
0c1db9598990: Pull complete
bfb904e99f24: Pull complete
78a3d3a96a32: Pull complete
885a0ed92c89: Pull complete
dd7cc9ace242: Pull complete
Digest: sha256:3870d35b962a943df72d948580fc66ceaaee1c4fbd205930f32e0f0760eb1077
Status: Downloaded newer image for python:3.7
[root@dab6769a0d24:/#
```

環境開啟後，我們來測試一下python版本

```
[root@dab6769a0d24:/# python -V
Python 3.7.1
```

### step3: 參數介紹：-v(--volume list)參數

環境開好了，該怎麼把程式碼放進去呢？

-v 參數可以把在本機的指定資料夾掛載到容器內，讓本機與容器資料同步，常常用來處理設定檔，程式碼...運行服務時需要的檔案及資料夾。

- 目標：
  - 將code資料夾掛載到container內/tmp位置，並改名為py\_code
  - python開啟py\_code資料夾內的hello.py並查看運行結果

開啟容器：

```
1 docker run -it --rm -v $(pwd)/code:/tmp/py_code python:3.7 /bin/bash
2 #偷懶專用
3 ./step3.sh
```

查看/tmp下有沒有py\_code資料夾及hello.py

```
[root@0ee9a1d981a4:/# ls -R /tmp
/tmp:
py_code

/tmp/py_code:
hello.py
```

執行hello.py，並查看運行結果：

```
python /tmp/py_code/hello.py
```

```
[root@0ee9a1d981a4:/# python /tmp/py_code/hello.py
Hello World!
```

沒錯～～又是Hello-world

#### step4 : -p (--publish list)端口映射

現在，我們已經可以把程式碼放進去container內了，接下來我們想在container內運行一個Flask服務，讓我們試試看吧。

```
1 docker run -it --rm -v $(pwd)/code:/tmp/py_code python:3.7 /bin/bash
2 #偷懶
3 ./step4.sh
```

進入container 後安裝flask

```
pip install flask==0.12
```

```
root@74c567b13759:/# pip install flask==0.12
Collecting flask==0.12
  Downloading https://files.pythonhosted.org/packages/0e/e9/37ee66dde483dceef45bb5e92b387f990d4f097df40c400cf816dcebba4/Flask-0.12-py2.py3-none-any.whl (82kB)
    100% |██████████| 92kB 761kB/s
Collecting Werkzeug>=0.7 (from flask==0.12)
  Downloading https://files.pythonhosted.org/packages/20/c4/12e3e56473e52375aa29c4764e70d1b8f3efa6682bef8d0aae04fe335243/Werkzeug-0.14.1-py2.py3-none-any.whl (322kB)
    100% |██████████| 327kB 1.6MB/s
Collecting Jinja2>=2.4 (from flask==0.12)
  Downloading https://files.pythonhosted.org/packages/7f/ff/ae64bacdfc95f27a016a7bed8e8686763ba4d277a78ca76f32659220a731/Jinja2-2.10-py2.py3-none-any.whl (126kB)
    100% |██████████| 133kB 667kB/s
Collecting click>=2.0 (from flask==0.12)
  Downloading https://files.pythonhosted.org/packages/fa/37/45185cb5abbc30d7257104c434fe0b07e5a195a6847506c074527aa599ec/Click-7.0-py2.py3-none-any.whl (81kB)
    100% |██████████| 81kB 1.0MB/s
Collecting itsdangerous>=0.21 (from flask==0.12)
  Downloading https://files.pythonhosted.org/packages/76/ae/44b03b253d6fade317f32c24d100b3b35c2239807046a4c953c7b89fa49e/itsdangerous-1.1.0-py2.py3-none-any.whl
Collecting MarkupSafe>=0.23 (from Jinja2>=2.4->flask==0.12)
  Downloading https://files.pythonhosted.org/packages/e4/c4/adcc2d6f2ac2146cc04e076f14f1006c1de8e1e747fa067668b6573000b8/MarkupSafe-1.0.0-cp37-cp37m-manylinux1_x86_64.whl
Installing collected packages: Werkzeug, MarkupSafe, Jinja2, click, itsdangerous, flask
Successfully installed Jinja2-2.10 MarkupSafe-1.1.0 Werkzeug-0.14.1 click-7.0 flask-0.12 itsdangerous-1.1.0
```

container ip查詢

```
ip a
```

```
root@74c567b13759:/# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
  link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
  inet 127.0.0.1/8 scope host lo
    valid_lft forever preferred_lft forever
73: eth0@if74: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
  link/ether 02:42:ac:11:00:02 brd ff:ff:ff:ff:ff:ff link-netnsid 0
  inet 172.17.0.2/16 brd 172.17.255.255 scope global eth0
    valid_lft forever preferred_lft forever
```

運行程式碼

```
python /tmp/py_code/app.py
```

```
root@74c567b13759:/# python /tmp/py_code/app.py
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
```

在本機（虛擬機）內使用瀏覽器或curl工具連線container-ip:5000  
瀏覽器：

## Curl

```
[iii@localhost code]$ curl 172.17.0.2:5000
Hello World![iii@localhost code]$
```

如果想要開放外網連線呢？我們必須要用到iptables進行轉址

`sudo iptables -t nat ..... 指令實在太長了 很麻煩`

所以 docker 提供我們一個簡單的參數 `<-p>`，我們可以用-p參數來做端口映射，省去前面麻煩的內容，專注於開發及部署。

再一次重新開始吧！！

```
docker run -it --rm -v $(pwd)/code:/tmp/py_code -p 5000:5000 python:3.7 /bin/bash
```

進入container後

- 1 `pip install flask`
- 2 `python /tmp/py_code/app.py`

直接從本機瀏覽器瀏覽192.168.56.111:5000

Hello World!

這樣是不是簡單多了呢！！！

```
[iii@localhost code]$ curl 172.17.0.2:5000
Hello World![iii@localhost code]$ docker ps
CONTAINER ID        IMAGE       COMMAND       CREATED          STATUS          PORTS          NAMES
c9f829f9ba20        python:3.7   "/bin/bash"   3 minutes ago   Up 3 minutes   0.0.0.0:5000->5000/tcp   gracious_vohard
```

其實`<-p>`這個參數，就是讓docker自動幫我們做iptables的動作

```
[iii@localhost code]$ sudo iptables -t nat -L DOCKER
Chain DOCKER (2 references)
target     prot opt source               destination
RETURN    all  --  anywhere             anywhere
DNAT      tcp  --  anywhere             anywhere           tcp dpt:complex-main to:172.17.0.2:5000
```

補充 docker run 可以基於bash或sh進行多行指令：

每次都要進入container再安裝flask太麻煩了，有沒有辦法可以在docker run 執行的時候順便安裝並開啟app.py呢？

答案是：YES!

試試下面的指令：

```

1 docker run -it --rm -v $(pwd)/code:/tmp/py_code -p 5000:5000 python:3.7 /bin/bash -c 'pip install flask==0.12 && python /tmp/py_code/app.py'
2 #bash -c: 讓bash 後面的指令已字串讀入，並以&&做區隔
3 #偷懶：
4 ./try.sh

```

```
[iiii@localhost step4]$ docker run -it --rm -v $(pwd)/code:/tmp/py_code -p 5000:5000 python:3.7 /bin/bash -c 'pip install flask==0.12 && python /tmp/py_code/app.py'
Collecting flask==0.12
  Downloading https://files.pythonhosted.org/packages/0e/e9/37ee66dde483dceef45bb5e92b387f990d4f097df40c400cf816dcebba4/Flask-0.12-py2.py3-none-any.whl (82kB)
    100% |██████████| 92kB 922kB/s
Collecting Jinja2>=2.4 (from flask==0.12)
  Downloading https://files.pythonhosted.org/packages/7f/ff/ae64bacdfc95f27a016a7bed8e8686763ba4d277a78ca76f32659220a731/Jinja2-2.10-py2.py3-none-any.whl (126kB)
    100% |██████████| 133kB 2.2MB/s
Collecting click>=2.0 (from flask==0.12)
  Downloading https://files.pythonhosted.org/packages/fa/37/45185cb5abbc30d7257104c434fe0b07e5a195a6847506c074527aa599ec/Click-7.0-py2.py3-none-any.whl (81kB)
    100% |██████████| 81kB 2.4MB/s
Collecting itsdangerous>=0.21 (from flask==0.12)
  Downloading https://files.pythonhosted.org/packages/76/ae/44b03b253d6fade317f32c24d100b3b35c2239807046a4c953c7b89fa49e/itsdangerous-1.1.0-py2.py3-none-any.whl
Collecting Werkzeug>=0.7 (from flask==0.12)
  Downloading https://files.pythonhosted.org/packages/20/c4/12e3e56473e52375aa29c4764e70d1b8f3efa6682bef8d0aae04fe335243/Werkzeug-0.14.1-py2.py3-none-any.whl (322kB)
    100% |██████████| 327kB 1.6MB/s
Collecting MarkupSafe>=0.23 (from Jinja2>=2.4->flask==0.12)
  Downloading https://files.pythonhosted.org/packages/e4/c4/adcc2d6f2ac2146cc04e076f14f1006c1de8e1e747fa067668b6573000b8/MarkupSafe-1.1.0-cp37-cp37m-manylinux1_x86_64.whl
Installing collected packages: MarkupSafe, Jinja2, click, itsdangerous, Werkzeug, flask
Successfully installed Jinja2-2.10 MarkupSafe-1.1.0 Werkzeug-0.14.1 click-7.0 flask-0.12 itsdangerous-1.1.0
* Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
```

## step5 docker run 其他參數：

-d, --detach：前面我們已經可以用一個指令安裝Flask並執行app.py了，如果我們不想上她佔用我們的畫面的話，我們可以用-d讓他在背景執行。

```

1 docker run -dit --rm -v $(pwd)/code:/tmp/py_code -p 5000:5000 python:3.7 /bin/bash -c 'pip install flask==0.12 && python /tmp/py_code/app.py'
2 #偷懶：
3 ./detach.sh

```

--name：當我們有很多個container 時可以針對個別container進行命名，這樣就不用每次都要查container\_id了

```

1 docker run -dit --rm --name flask_test -v $(pwd)/code:/tmp/py_code -p 5000:5000 python:3.7 /bin/bash -c 'pip install flask==0.12 && python /tmp/py_code/app.py'
2 #偷懶：
3 ./name.sh

```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
eea47b6ae1f1	python:3.7	"/bin/bash -c 'pip i..."	4 seconds ago	Up 3 seconds	0.0.0.0:5000->5000/tcp	flask test

## docker run 小結整理

指令 : docker run [參數] <image> <命令>

- 1 docker run -dit --name flask --rm -v \$(pwd)/code:/code -p 5000:5000 python:3.6 /bin/bash -c 'pip install flask==0.12 && python /tmp/py\_code/app.py'
- 2 常用參數 :
- 3 -v, --volume list 在容器中掛載檔案（或資料夾）
- 4 -p, --publish list : 端口 (Port) 映射，格式<本地端口>:<容器端口>
- 5 -it : 互動式tty，交互互動模式
- 6 --rm : 容器結束後自動刪除。
- 7 -d, --detach : 背景執行，不顯示log
- 8 --name : 容器命名，命名為唯一不可重複
- 9 --restart : 如果容器運行失敗，重新運行。

## Step6 : docker exec : 連入已開啟的容器。

現在我們有一個容器正在背景執行，如果我們想要連入進行操作，該怎麼做呢？

這時候我們就需要用到 docker exec 這個指令了

連入容器：

```
docker exec -it flask_test /bin/bash
```

```
[iii@localhost step6]$ docker exec -it flask_test /bin/bash
root@32aa7e6f880:/#
```

或是，直接讓容器執行指定命令：

```
docker exec flask_test echo "HI"
```

```
[iii@localhost step6]$ docker exec flask_test echo "HI"
HI
```

其他：對 docker 容器進行操作

- kill/stop : 停止
  - 指令 : docker kill/stop <container\_id>/<container\_name>
  - 區別 : stop 會按照程序停止；kill 會強制停止

- 1 docker kill flask\_test
- 2 docker stop flask\_test

- start/restart : 開啟
  - 指令 : docker start/restart <container\_id>/<container\_name>

- 兩個都是開啟的意思，通用

```
1 docker start flask_test
2 docker restart flask_test
```

- rm：刪除
  - 完全刪除容器（容器內的資料不會被保留！！）
  - 有需要留存的資料須先放到-v所mapping的資料夾內。
  - `docker rm <container_id>/<container_name>`

```
docker rm flask_test
```

## docker 查詢

針對容器進行查詢：`docker ps`：

前面，我們把我們的docker背景執行了，那我們要怎麼查看container現在有沒有在運行呢？試試`docker ps`這個指令吧。

```
1 docker ps <參數>
2 -a 查看所有（包括停止）的容器
3 -q 只顯示容器的 id，常搭配docker kill或docker rm 使用。
```

```
[iiii@localhost step6]$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS              NAMES
32aa7e6f880          python:3.7        "/bin/bash -c 'pip i..."   44 minutes ago    Up 44 minutes     0.0.0.0:5000->5000/tcp   flask_test
[iiii@localhost step6]$ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS              NAMES
32aa7e6f880          python:3.7        "/bin/bash -c 'pip i..."   44 minutes ago    Up 44 minutes     0.0.0.0:5000->5000/tcp   flask_test
```

## Step7 : Dockerfile : 訂製Image

### Dockerfile基本介紹

前面，我們有提到過VM常會遇到的問題：我們無法很明確的知道誰對這台虛擬機做了什麼，是不是有人更改過檔案或設定。在前面練習`docker run`或`docker exec`的過程中，似乎也沒有解決相對應的問題，因此我們需要用`Dockerfile`來統一我們對image所做的設定，進而達到*Infrastructure-as-code (IaC)*的目的。

`Dockerfile`是一個文本文件，其內包含了一條條的指令(**Instruction**)，每一條指令構建一層，因此每一條指令的內容，就是描述該層應當如何構建。

在`Dockerfile`註解使用“#”字號

### FROM : 指定基礎鏡像

`FROM`：所謂定製鏡像，那一定是以一個鏡像為基礎，在其上進行定制。就像我們之前運行了一個nginx鏡像的容器，再進行修改一樣，基礎鏡像是必須指定的。而`FROM`就是指定基礎鏡像，因此一個`Dockerfile`中`FROM`是必備的指令，並且必須是第一條指令。

將前面的`docker run Image`的部分寫成`Dockerfile`會變成這樣：

```
1 #基於IMAGE python:3.7
2 FROM python:3.7
```

### RUN : 執行命令

**RUN**指令是用來執行命令行命令的。由於命令行的強大能力，**RUN**指令在定製鏡像時是最常用的指令之一。

前面，我們在練習**docker run**的時候，每一次都需要先安裝**flask**，那我們可不可以基於**Dockerfile**的概念，建構一個有以**python:3.7**為基礎，並安裝好**flask**的**Image**呢？

```

1 #基於IMAGE python:3.7
2 FROM python:3.7
3 #安裝Flask 0.12版
4 RUN pip install flask==0.12

```

- 在**dockerfile**中使用**RUN**的重要概念：
  - 前面有提到**dockerfile**會“每一條指令構建一層”，因此使用**RUN**時應該把相應的指令寫在同一層，減少容器建構時的程序及佔用的空間。
- 舉例：在安裝**flask**之前我們想要先更新**apt-get**套件庫，並使用**pip**安裝**flask 0.12**版本及使用**apt-get**安裝**vim**方便做程式碼的編輯。
  - 錯誤示範：

```

1 #基於IMAGE python:3.7
2 FROM python:3.7
3 ##更新apt套件庫
4 RUN apt-get update -qq
5 ##安裝Flask 0.12版
6 RUN pip install -q flask==0.12
7 #安裝vim
8 RUN apt-get install vim -qqy

```

```

[pii@localhost step7]$ docker build -t ping/flask .
Sending build context to Docker daemon 2.048kB
Step 1/4 : FROM python:3.7
--> 1e80caffd59e
Step 2/4 : RUN apt-get update -qq
--> Using cache
--> 3e95714d817c
Step 3/4 : RUN pip install -q flask==0.12
--> Using cache
--> 464e12400c3f
Step 4/4 : RUN apt-get install vim -qqy
--> Using cache
--> 2b3ab8b66b21
Successfully built 2b3ab8b66b21
Successfully tagged ping/flask:latest

```

- 正確寫法

```

1 #基於IMAGE python:3.7
2 FROM python:3.7
3 ##更新apt套件庫、安裝Flask 0.12版、安裝VIM
4 RUN apt-get update -qq \
    && pip install -q flask==0.12 \
    && apt-get install vim apt-utils -qqy

```

```

[iii@localhost step7]$ docker build -t ping/flask .
Sending build context to Docker daemon 2.048kB
Step 1/2 : FROM python:3.7
--> 1e80caffd59e
Step 2/2 : RUN apt-get update -qq     && pip install -q flask==0.12     && apt-get install vim apt-utils -qqy
--> Using cache
--> 2659daecb76f
Successfully built 2659daecb76f
Successfully tagged ping/flask:latest

```

## ADD : 複製文件

更進一步，每次都要用-v 挂載實在是太麻煩了，如果我們想要把我們的程式碼包進去Image中，可不可以呢？

這時候需要用到ADD語法

```

1 #基於IMAGE python:3.7
2 FROM python:3.7
3 #更新apt套件庫、安裝Flask 0.12版、安裝VIM
4 RUN apt-get update -qq \
    && pip install -q flask==0.12 \
    && apt-get install vim apt-utils -qqy
7 #將./code複製到image的/tmp內，並重新命名為py_code
8 COPY ./code /tmp/py_code

```

```

[iii@localhost step7]$ docker build -t ping/flask .
Sending build context to Docker daemon 3.584kB
Step 1/3 : FROM python:3.7
--> 1e80caffd59e
Step 2/3 : RUN apt-get update -qq     && pip install -q flask==0.12     && apt-get install vim apt-utils -qqy
--> Using cache
--> 2659daecb76f
Step 3/3 : COPY ./code /tmp/py_code
--> Using cache
--> b1259d0d0003
Successfully built b1259d0d0003
Successfully tagged ping/flask:latest

```

```

[iii@localhost step7]$ docker run -it --rm ping/flask /bin/bash
root@d4f8b789e845:/# ls /tmp/py_code/
app.py

```

- 使用**ADD**功能須謹慎，尤其是可能會將**IMAGE**公開（上傳到github、docker hub....等平台），上傳前需檢查是否包含一些個人資訊，如：帳號、密碼、key等不應該暴露在公共環境下的訊息。
- 如果加入的檔案為壓縮檔，例如**.tar**、**.tar.xz** docker 會在**build**的時候自行解壓縮。

## WORKDIR：預設工作目錄

開啟容器後我們有可能需要編輯**app.py**檔案，每次都要切換到**/tmp/py\_code**實在是太麻煩了，而且，當把**image**給別人時，還要特別交代程式碼放在哪邊，有沒有更便捷的方法呢？

我們可以使用**WORKDIR**來預設工作目錄，當容器啟動時，會自動切換到我們指定的目錄，如此方便許多。

```

1 #基於IMAGE python:3.7
2 FROM python:3.7
3 #更新apt套件庫、安裝Flask 0.12版、安裝VIM
4 RUN apt-get update -qq \
5     && pip install -q flask==0.12 \
6     && apt-get install vim apt-utils -qqy
7 #將./code複製到image的/tmp內，並重新命名為py_code
8 COPY ./code /tmp/py_code
9 #切換到預設工作目錄/tmp/py_code
10 WORKDIR /tmp/py_code

```

```

[iii@localhost step7]$ docker build -t ping/flask .
Sending build context to Docker daemon 3.584kB
Step 1/4 : FROM python:3.7
--> 1e80caffd59e
Step 2/4 : RUN apt-get update -qq && pip install -q flask==0.12 && apt-get install vim apt-utils -qqy
--> Using cache
--> 2659daecb76f
Step 3/4 : COPY ./code /tmp/py_code
--> Using cache
--> b1259d0d0003
Step 4/4 : WORKDIR /tmp/py_code
--> Using cache
--> cafa541123d9
Successfully built cafa541123d9
Successfully tagged ping/flask:latest

```

```

[iii@localhost step7]$ docker run -it --rm ping/flask /bin/bash
root@950966ac9a91:/tmp/py_code# ls
app.py

```

## ENV：設置環境變數

這個指令很簡單，就是設置環境變數而已，無論是後面的其它指令，如**RUN**，還是運行時的應用，都可以直接使用這裡定義的環境變數。

**EX**：在撰寫**Dockerfile**時將**Flask**版本以變數的方式呈現，方便後續做修改。

```

1 #基於IMAGE python:3.7
2 FROM python:3.7
3 #添加FLASK_VER為環境變數，並設置版本為0.12
4 ENV FLASK_VER 0.12
5 #更新apt套件庫、安裝Flask，版本由FLASK_VER讀入、安裝VIM
6 RUN apt-get update -qq \
    && pip install -q flask==$FLASK_VER \
    && apt-get install vim -qqy
9 #將./code複製到image的/tmp內，並重新命名為py_code
10 COPY ./code /tmp/py_code
11 #切換到預設工作目錄/tmp/py_code
12 WORKDIR /tmp/py_code

```

```

[iii@localhost step7]$ docker build -t ping/flask .
Sending build context to Docker daemon 3.584kB
Step 1/5 : FROM python:3.7
--> 1e80caffd59e
Step 2/5 : ENV FLASK_VER 0.12
--> Using cache
--> bff15a73c9f6
Step 3/5 : RUN apt-get update -qq     && pip install -q flask==$FLASK_VER     && apt-get install vim apt-utils -qqy
--> Using cache
--> dbeaf54fa538
Step 4/5 : COPY ./code /tmp/py_code
--> Using cache
--> 10b72055db3e
Step 5/5 : WORKDIR /tmp/py_code
--> Using cache
--> 64b5f818e14f
Successfully built 64b5f818e14f
Successfully tagged ping/flask:latest

```

```

[iii@localhost step7]$ docker run -it --rm ping/flask /bin/bash
root@dd937e328721:/tmp/py_code# echo "$FLASK_VER"
0.12
root@dd937e328721:/tmp/py_code# flask --version
Flask 0.12
Python 3.7.1 (default, Nov 16 2018, 22:26:09)
[GCC 6.3.0 20170516]

```

## CMD：預設開啟container執行的動作

我們可以透過CMD的指令，來指定開啟container後預設的動作，如下：

```

1 #基於IMAGE python:3.7
2 FROM python:3.7
3 #添加FLASK_VER為環境變數，並設置版本為0.12
4 ENV FLASK_VER 0.12
5 #更新apt套件庫、安裝Flask，版本由FLASK_VER讀入、安裝VIM

```

```

6 RUN apt-get update \
7     && pip install -q flask==$FLASK_VER \
8     && apt-get install vim apt-utils -qqy
9 #將./code複製到image的/tmp內，並重新命名為py_code
10 COPY ./code /tmp/py_code
11 #切換到預設工作目錄/tmp/py_code
12 WORKDIR /tmp/py_code
13 #預設開啟container時執行app.py
14 CMD python ./app.py

```

```
[iii@localhost step7]$ docker run -it -p 5000:5000 --rm ping/flask 不加命令
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
```

- 如果執行`docker run`時有指定動作，會自動忽略CMD該執行的動作。如下：

```
[iii@localhost step7]$ docker run -it -p 5000:5000 --rm ping/flask /bin/bash
root@28b8aeae780d:/tmp/py_code#
```

### **dockerfile小結：**

透過**dockerfile**自訂**IMAGE**我們將原本冗長的指令

```
docker run -idt --rm --name flask -v $(pwd)/code:/tmp/py_code
-p 5000:5000 python:3.7 /bin/bash -c 'pip install flask==0.12
&& python /tmp/py_code/app.py'
```

縮減成

```
docker run -d --rm --name flask -p 5000:5000 <image>
```

是不是簡潔及快速多了呢？

## **Step8：撰寫Dockerfile**

由於用VIM寫python實在太麻煩了，因此我們改用jupyter/base-notebook作為python的開發環境，試著寫一個以Jupyter/base-notebook為基礎的**dockerfile**。

- Dockerfile
  - 用pip更新pip版本
  - 用pip安裝flask0.12版、requests
  - 預設工作目錄為/home/jovyan/work，
  - 預設執行start-notebook.sh --NotebookApp.token="禁用所有身份驗證機制"
- docker run
  - 映射5000及8888 port
  - 映射code資料夾到/home/jovyan/work

### **dockerfile**

```

1 FROM jupyter/base-notebook
2 #使用pip 套件庫更新pip 並安裝flask 0.12 requests
3 RUN pip install --upgrade pip \
4   && pip install flask==0.12 requests
5 #指定預設工作目錄/home/jovyan/work
6 WORKDIR /home/jovyan/work
7 #開啟notebook並設定token為空
8 CMD start-notebook.sh --NotebookApp.token=''

```

## Docker image相關操作

### build

撰寫完dockerfile後我們需要建構image，這時候需要用到docker build指令

指令：docker build [參數] <dockerfile位置>

回到step7試試第一次寫的docker file吧

```

1 docker build -t iii_docker/flask:0.1 .
2 . 在同一層資料夾尋找檔名為dockerfile的檔案構建
3 常用參數：
4 -t：給images 加上標籤，一般建議格式為 <docker hub帳號>/<IMAGE名稱>:<
  版本>
5 偷懶一點：
6 ./build.sh
7 • dockerfile 每一條指令都會建構一層

```

```
[iii@localhost step7]$ ./build.sh
請輸入 repo名稱（帳號）：iii-docker
請輸入 應用名稱：flask
請輸入 版本號：0.2
產出命令 "docker build -t iii-docker/flask:0.2 ."
命令執行中.....
Sending build context to Docker daemon 5.632kB
Step 1/6 : FROM python:3.7
--> 1e80caffd59e
Step 2/6 : ENV FLASK_VER 0.12
--> Using cache
--> f5105237f20d
Step 3/6 : RUN apt-get update -qq && pip install -q flask==$FLASK_VER && apt-get install vim -qq
--> Using cache
--> 7b0ec50e6e73
Step 4/6 : COPY ./code /tmp/py_code
--> Using cache
--> 188465f8442a
Step 5/6 : WORKDIR /tmp/py_code
--> Using cache
--> 79763e921f48
Step 6/6 : CMD python ./app.py
--> Using cache
--> 035655291608
Successfully built 035655291608
Successfully tagged iii_docker/flask:0.2
```

探討幾個問題：

Q1. 不加上tag會怎麼樣？

build還是會正常執行，但是<REPOSITORY>及<TAG>會變成<none>造成管理問題。

Q2. 不加上版本號呢？

不加上版本號預設會是latest，但是當下次以相同的TAG構建時，這一版的TAG會被取消。

Q3. 一定要叫做dockerfile嗎？

如果有多個dockerfile檔案放在同一個資料夾內，可以使用-f參數，指定dockerfile  
EX : docker build -t test/test:0.1 test ./code

## docker images管理與操作

針對映像檔(image)進行查詢：**docker images**：

前面，我們提到docker容器都是基於映像檔(image)所建構的，那我們該如何管理本機的映像檔呢？

Docker提供了**docker images**這個指令讓我們進行映像檔查詢

- 1 docker images [參數] [REPOSITORY] [:TAG]
- 2 -q 只顯示映像檔id，通常配合**docker rmi** 批次刪除映像檔。

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
python	3.7	1e80caffd59e	3 weeks ago	923MB
hello-world	latest	4ab4c602aa5e	3 months ago	1.84kB

針對映像檔(image)進行刪除：**docker rmi**：

使用**docker**越久，本機內一定會有很多用不到的映像檔，是時候來個大掃除了，這時候我們使用**docker rmi**這個指令，來刪除用不到的映像檔。

指令 : docker rmi <參數> <IMAGE ID>/<REPOSITORY>:<TAG>

- 常用參數 : -f 強制刪除

```
[iii@localhost step6]$ docker rmi hello-world
Untagged: hello-world:latest
Untagged: hello-world@sha256:0add3ace90ecb4adb7777e9aacf18357296e799f81cabc9fde470971e499788
Deleted: sha256:4ab4c602aa5eed5528a6620ff18a1dc4faef0e1ab3a5eddeddb410714478c67f
Deleted: sha256:428c97da766c4c13b19088a471de6b622b038f3ae8efa10ec5a37d6d31a2df0b
```

```
[iii@localhost step6]$ docker rmi python:3.6
Untagged: python:3.6
Untagged: python@sha256:fc34d5b6cf5d00a6139a74370dc27ddc9ce18303e2210d0f199a6050cc29aa45
Deleted: sha256:1ec4d11819ad20606d652f663f3c90ad89f05cf1d624189fd4184eb41555e21c
Deleted: sha256:9f2fe45222be2dd641cf485b0e16cdd623cd5d49c370ed93037544f44b016dae
Deleted: sha256:d45ef8355fb9bb3ae1ff38b08bc480306444934c11e845f864108ce637d8137b
Deleted: sha256:4339fec09220bad06376370dd15398c6833cee9dbf97fa188dd8fa4609397749
Deleted: sha256:fc0333d2d70d227eae8b1ed40965190950a7c858677f2b16ac01ea15cecd0358
```

註1：移除image前，要先移除相要先移除相對應的container

註2：如果image的TAG為latest，可以不加上<TAG>

註3：當一個image有多個tag時，使用image id無法直接移除，若要強制移除，使用-f參數

### docker tag 貼個標籤吧

我們在build的時候會用-t來加入Tag，方便我們整理。但是，實務上我們可能需要不只一個Tag，那我們可不可以對一個IMAGE下多個Tag呢？

這時候可以使用docker tag這個指令來對image新增Tag

指令 : docker tag <原本IMAGE資訊(REPO:TAG 或是ID)> <新的TAG資訊>

- 1 例如：
- 2 docker tag jupyter/base-notebook:**latest** pynb
- 3 也可以用IMAGE ID
- 4 docker tag 8253370cbe2d jupyter

```
[iii@localhost step8]$ docker tag jupyter/base-notebook:latest pynb
[iii@localhost step8]$ docker images
REPOSITORY          TAG        IMAGE ID      CREATED       SIZE
jupyter/base-notebook  latest    8253370cbe2d  5 days ago   738MB
pynb                latest    8253370cbe2d  5 days ago   738MB
python              3.7       1e80caffd59e   5 weeks ago  923MB
[iii@localhost step8]$
[iii@localhost step8]$ docker tag 8253370cbe2d jupyter
[iii@localhost step8]$ docker images
REPOSITORY          TAG        IMAGE ID      CREATED       SIZE
jupyter/base-notebook  latest    8253370cbe2d  5 days ago   738MB
jupyter              latest    8253370cbe2d  5 days ago   738MB
pynb                latest    8253370cbe2d  5 days ago   738MB
python              3.7       1e80caffd59e   5 weeks ago  923MB
[iii@localhost step8]$
```

TAG常用的應用場景：版本複雜時，可以對大版本及最後版本下標籤，方便管理  
TAG的移除

移除Tag使用docker rmi 指令，例如：

```
docker rmi jupyter:latest
```

```
[iii@localhost step8]$ docker rmi jupyter:latest
Untagged: jupyter:latest
```

- 當一個IMAGE存在多個標籤的時候docker rmi 當作移除標籤使用
- 當對象為唯一標籤時，docker rmi 會直接刪除IMAGE
- 用docker rmi 移除標籤時"不可以使用IMAGE ID"

## Image匯入匯出

### Step9 : docker image 本地環境匯入匯出

dockerfile十分方便也便於攜帶，但是如果我們要到一個沒有網路的環境呢？

我們可以先將我們製作好的IMAGE包成.tar檔匯出，到我們的目標主機再匯入。

這樣做的好處是包裝好的tar檔解壓後的環境，跟我們原本製作的環境一樣，而且不需要像Dockerfile建構的時候有些更新或安裝套件的命令還需要聯網。

- 匯出：
  - 指令：docker save -o <匯出的檔名> <IMAGE資訊或ID>
  - 不建議使用IMAGE ID做匯出，TAG資訊會消失

1 EX: 使用IMAGE資訊

2 docker save -o python3.7.tar python:3.7

3 或用IMAGE ID

4 docker save -o python3.7.tar 1e80caffd59e

- 匯入
  - 指令：docker load < xxx.tar 檔
  - docker load --input xxx.tar 檔

1 EX:

2 docker load < python3.7.tar

3 docker load --input python3.7.tar

- docker load 如果出現重複名稱時系統的處理方式
  - 現存的IMAGE 建構時間較舊：會被直接覆蓋
  - 現存的IMAGE 建構的時間較新：現存的IMAGE會被untag，load的檔案會生成指定的IMAGE

```
[iii@localhost step8]$ docker load < hi.tar
The image hi:latest already exists, renaming the old one with ID sha256:858ba460c3f7d0f9d7630a86e0cff75ec
18190ac9b605a9491c0757baf54a097 to empty string
Loaded image: hi:latest
[iii@localhost step8]$ docker images
REPOSITORY          TAG           IMAGE ID            CREATED             SIZE
<none>              <none>        858ba460c3f7    44 seconds ago   923MB
hi                  latest        fe3f96b50bc1    2 minutes ago    923MB
jupyter/base-notebook  latest        8253370cbc2d    5 days ago       738MB
python               3.7          1e80caffd59e    5 weeks ago       923MB
```

練習：

先將剛剛做的jupyter用docker save打包，打包完成後，用docker rmi刪除原本的IMAGE，再用docker load匯入.tar檔。

```
[iii@localhost step9]$ docker images
REPOSITORY          TAG           IMAGE ID      CREATED        SIZE
jupyter/base-notebook  latest        8253370cbe2d  5 days ago    738MB
python               3.7          1e80caffd59e   5 weeks ago   923MB
[iii@localhost step9]$ docker save -o python3.7.tar python:3.7
[iii@localhost step9]$ ls
load.sh  python3.7.tar  rm.sh  save.sh
[iii@localhost step9]$ docker rmi python:3.7
Untagged: python:3.7
Deleted: sha256:1e80caffd59ed9edbd05c6ad47427caadf297f399ec069ec014a5aadb2f261a0
Deleted: sha256:75de781673aa27608b6e3b754cf66485b34fd924083cc63c3f1ba0d78ad07d43
Deleted: sha256:6e9427716f7ba4774440faf9275580feeaa200d8a6e09629334cb577f9ea44bda
Deleted: sha256:c6972c722173a8236ff40812b8e35f75660dfb09ab53478f3ac21aabf90d01bb
Deleted: sha256:0e649538ffe0f434d12f700d54c227d1915d1d6cb27b758a3e1f224e031e8491
Deleted: sha256:4ae80746f1129db339b77b93687aaa8a55359f86a9a616848bb8fe5454b4f95f
Deleted: sha256:eb91e8adb4541f0df58f6d896fee1355e04c0984b3104da1038178b8d6649b6b
Deleted: sha256:b7f634c63f4704fdffd2f4f2ec90a3e188448c5a99e7f4418555e61d63497d9f
Deleted: sha256:27a0b0f4d1e3c92972729fdb938c433c5b01107f895f2498e94929f2b096f4a3
Deleted: sha256:90d1009ce6fe3102fee728742a3bd73eea2b39c88cdda99977a3fb130dbc17ac
[iii@localhost step9]$ docker images
REPOSITORY          TAG           IMAGE ID      CREATED        SIZE
jupyter/base-notebook  latest        8253370cbe2d  5 days ago    738MB
[iii@localhost step9]$ docker load < python3.7.tar
90d1009ce6fe: Loading layer  105.5MB/105.5MB
c23711a84ad4: Loading layer  23.99MB/23.99MB
8f7ee6d76fd9: Loading layer  8.005MB/8.005MB
f75e64f96dbc: Loading layer  146.4MB/146.4MB
e02b32b1fff99: Loading layer  570.6MB/570.6MB
9e89ea4aeda3: Loading layer  17.5MB/17.5MB
3294fae58626: Loading layer  70.77MB/70.77MB
9e701d0e0edd: Loading layer  4.608kB/4.608kB
c660d5dc256a: Loading layer  6.353MB/6.353MB
Loaded image: python:3.7
[iii@localhost step9]$ docker images
REPOSITORY          TAG           IMAGE ID      CREATED        SIZE
jupyter/base-notebook  latest        8253370cbe2d  5 days ago    738MB
python               3.7          1e80caffd59e   5 weeks ago   923MB
```

```
1 #偷懶用
2 #step9資料夾
3 #匯出
4 ./save.sh
5 #刪除
6 ./rm.sh
7 #匯入
8 ./load.sh
```

## 將IMAGE上傳到docker hub

如果，我們今天想把我們製作的IMAGE公開出去我們有兩種方式：

- 用上面提到的docker save指令，把我們的IMAGE打包成tar檔，放到雲端空間上在給出連結，讓人下載。
- 上傳到docker hub

上傳到docker hub有什麼好處呢：

- 可以直接使用docker run 指定IMAGE docker 會自動下載匯入並使用。

開始使用之前須先至docker hub進行註冊

註冊完成後，使用docker login登入到docker hub

```
docker login
```

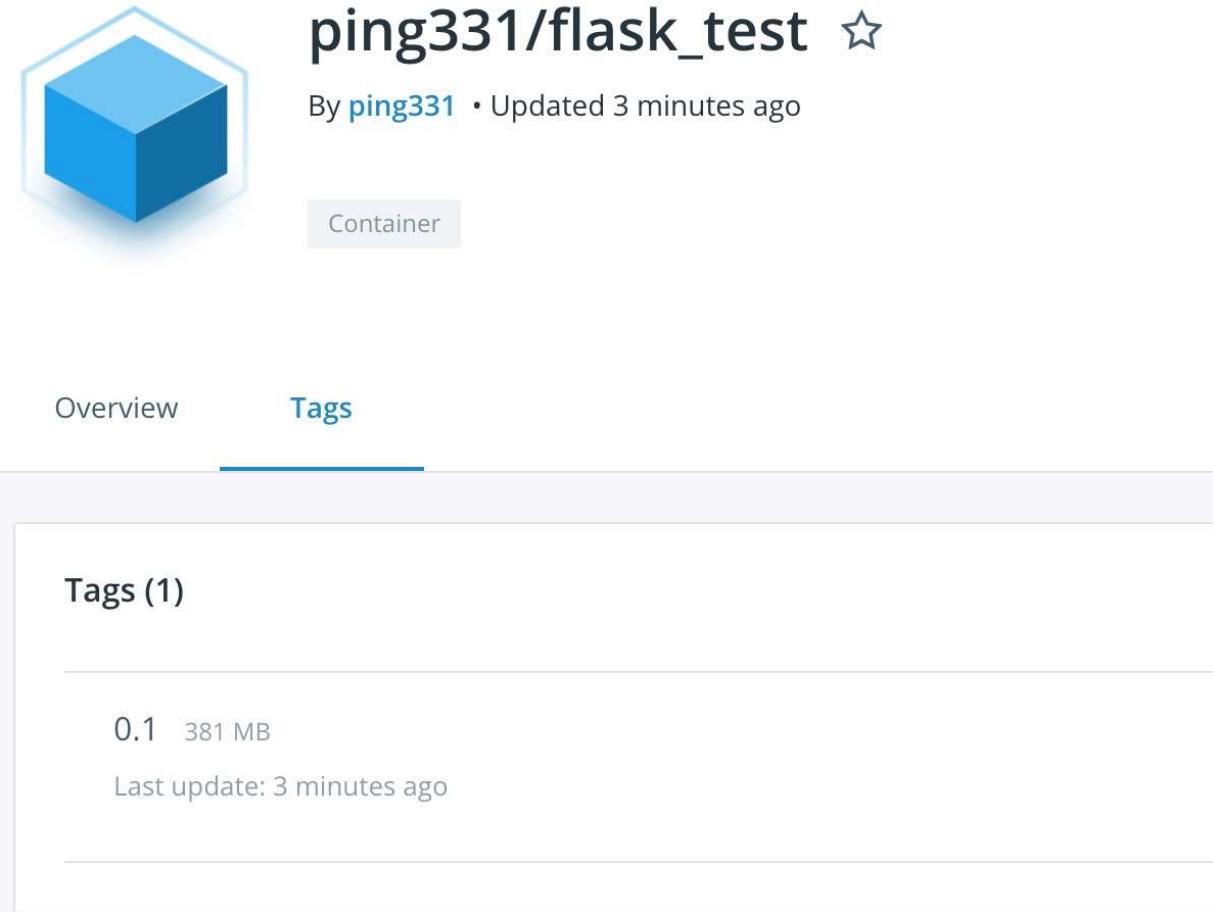
```
[iii@localhost step8]$ docker login
Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head over to https://hub.docker.com to create one.
Username: ping331
>Password:
WARNING! Your password will be stored unencrypted in /home/iii/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
```

- 上傳到docker hub

- 現在讓我們試著把step7製作的flask IMAGE上傳到自己的docker hub吧！
- 指令：docker push <docker hub 帳號>/<應用名稱>:<TAG>

```
[iii@localhost step7]$ docker images
REPOSITORY          TAG      IMAGE ID      CREATED       SIZE
ping331/flask_test  0.1      05328ce0f2e9   11 seconds ago  981MB
jupyter/base-notebook  latest   8253370cbe2d   5 days ago    738MB
python              3.7      1e80caffd59e   5 weeks ago   923MB
[iii@localhost step7]$ docker push ping331/flask_test:0.1
The push refers to repository [docker.io/ping331/flask_test]
829163af1240: Pushed
2da99eba1264: Pushed
c660d5dc256a: Mounted from library/python
9e701d0e0edd: Mounted from library/python
3294fae58626: Mounted from library/python
9e89ea4aeda3: Mounted from library/python
e02b32b1ff99: Mounted from library/python
f75e64f96dbc: Mounted from library/python
8f7ee6d76fd9: Mounted from library/python
c23711a84ad4: Mounted from library/python
90d1009ce6fe: Mounted from library/python
0.1: digest: sha256:0900ef5051446046bd4477716f1a7eb983ebe440598f184b5cb9ff98c01b97e size: 2637
```



- 從docker hub 上抓取image

接著，我們試著把本機上的image刪除從docker hub上拉下我們剛上傳的image進行應用。

```
1 #刪除image  
2 docker rmi <image>  
3 #從docker hub 上拉取image  
4 docker pull <image資訊>
```

```
[iis@localhost step7]$ docker rmi ping331/flask_test:0.1
Untagged: ping331/flask_test:0.1
Untagged: ping331/flask_test@sha256:0900ef50514460466bd4477716f1a7eb983ebe440598f184b5cb9ff98c01b97e
Deleted: sha256:05328ce0f2e9a965831bf057d778b6213b9f920ef67bbfed0d246e39ea820d1e
Deleted: sha256:eda43389b596c6c115d60d980e9197cbdd7a0b00e42e72de72a50841a0180069
Deleted: sha256:850705e6abf0b8fbe429c564485aa61223918dcbb6cff08dce0c088ca62ab531
Deleted: sha256:561e005fb835c41a1b85800ccccaf86c9c7758d08c967d7382e160232d2d8ede
Deleted: sha256:bc7d4e0b6ca15343ad294ed996a7b6ab39d43d90c843bd11effd116ef95bf77
Deleted: sha256:8054061c096cf3f97332bcb71c227a4b293bcd9b3faaf527fb89444f1effcee8
Deleted: sha256:6867c553a57013d9f0c1a3eedb530150cc2c7c3a0d7a9cd893164a712fc0f5db
[ii@localhost step7]$ docker images
REPOSITORY          TAG           IMAGE ID      CREATED        SIZE
jupyter/base-notebook    latest       8253370cbe2d   5 days ago    738MB
python               3.7          1e80caffd59e   5 weeks ago   923MB
[ii@localhost step7]$ docker pull ping331/flask_test:0.1
0.1: Pulling from ping331/flask_test
54f7e8ac135a: Already exists
d6341e30912f: Already exists
087a57faf949: Already exists
5d71636fb824: Already exists
0c1db9598990: Already exists
bfb904e99f24: Already exists
78a3d3a96a32: Already exists
885a0ed92c89: Already exists
dd7cc9ace242: Already exists
55ed3d130883: Pull complete
42641a224b22: Pull complete
Digest: sha256:0900ef50514460466bd4477716f1a7eb983ebe440598f184b5cb9ff98c01b97e
Status: Downloaded newer image for ping331/flask_test:0.1
[ii@localhost step7]$ docker images
REPOSITORY          TAG           IMAGE ID      CREATED        SIZE
ping331/flask_test    0.1          05328ce0f2e9   9 minutes ago  981MB
jupyter/base-notebook    latest       8253370cbe2d   5 days ago    738MB
python               3.7          1e80caffd59e   5 weeks ago   923MB
```

- 1 #或是可以用docker run指定剛剛上傳的image
- 2 docker run -d --rm --name flask -p 5000:5000 <image資訊>

```
[iis@localhost step7]$ docker rmi ping331/flask_test:0.1
Untagged: ping331/flask_test:0.1
Untagged: ping331/flask_test@sha256:0900ef50514460466bd4477716f1a7eb983ebe440598f184b5cb9ff98c01b97e
Deleted: sha256:05328ce0f2e9a965831bf057d778b6213b9f920ef67bbfed0d246e39ea820d1e
Deleted: sha256:561e005fb835c41a1b85800ccccaf86c9c7758d08c967d7382e160232d2d8ede
Deleted: sha256:8054061c096cf3f97332bcb71c227a4b293bcd9b3faaf527fb89444f1effcee8
[ii@localhost step7]$ docker run -d --rm --name flask -p 5000:5000 ping331/flask_test:0.1
Unable to find image 'ping331/flask_test:0.1' locally
0.1: Pulling from ping331/flask_test
54f7e8ac135a: Already exists
d6341e30912f: Already exists
087a57faf949: Already exists
5d71636fb824: Already exists
0c1db9598990: Already exists
bfb904e99f24: Already exists
78a3d3a96a32: Already exists
885a0ed92c89: Already exists
dd7cc9ace242: Already exists
55ed3d130883: Pull complete
42641a224b22: Pull complete
Digest: sha256:0900ef50514460466bd4477716f1a7eb983ebe440598f184b5cb9ff98c01b97e
Status: Downloaded newer image for ping331/flask_test:0.1
ec65cf4f9dc17a9039c5d105f0001561f938c4ae2aa4a780772a1dbc69e0417
```

## Step10 : container互聯

前面，我們應用的都是單container的應用，那如果我們的服務是需要用到一台以上的container呢？

EX：網頁服務需包含伺服器與資料庫，ELK須包含Elasticsearch、Logstash、Kibana、Beats等服務。

容器的連接（linking）系統是除了連接埠映射外，另一種跟容器中應用互動的方式。該系統會在來源端容器和接收端容器之間創建一個隧道，接收端容器可以看到來源端容器指定的信息。

### 來源

假設：我們flask應用需要一個帶有https連線的url，這時候我們想要使用ngrok來幫我們進行轉址的動作，這時候我們該怎麼做？

- 在docker run指令下使用--link來達到兩台容器互聯的目的

1 先開啟一個命名為flask的container：

```
2 #偷懶 ./flask.sh
3 docker run -d --name flask --rm -p 5000:5000 ping331/flask_test:0.1
```

4 開啟一個ngrok的container：

```
5 #偷懶 ./ngrok.sh
6 docker run -d --rm --name ngrok -p 4040:4040 --link flask wernight/ngrok ngrok http flask:5000
7 抓取ngrok的url：
8 #偷懶 ./geturl.sh
9 curl -s "localhost:4040/api/tunnels" | awk -F',' '{print $3}'
| awk -F"'{" '{print $4}' | awk -F'{}' '{print $2}'
```

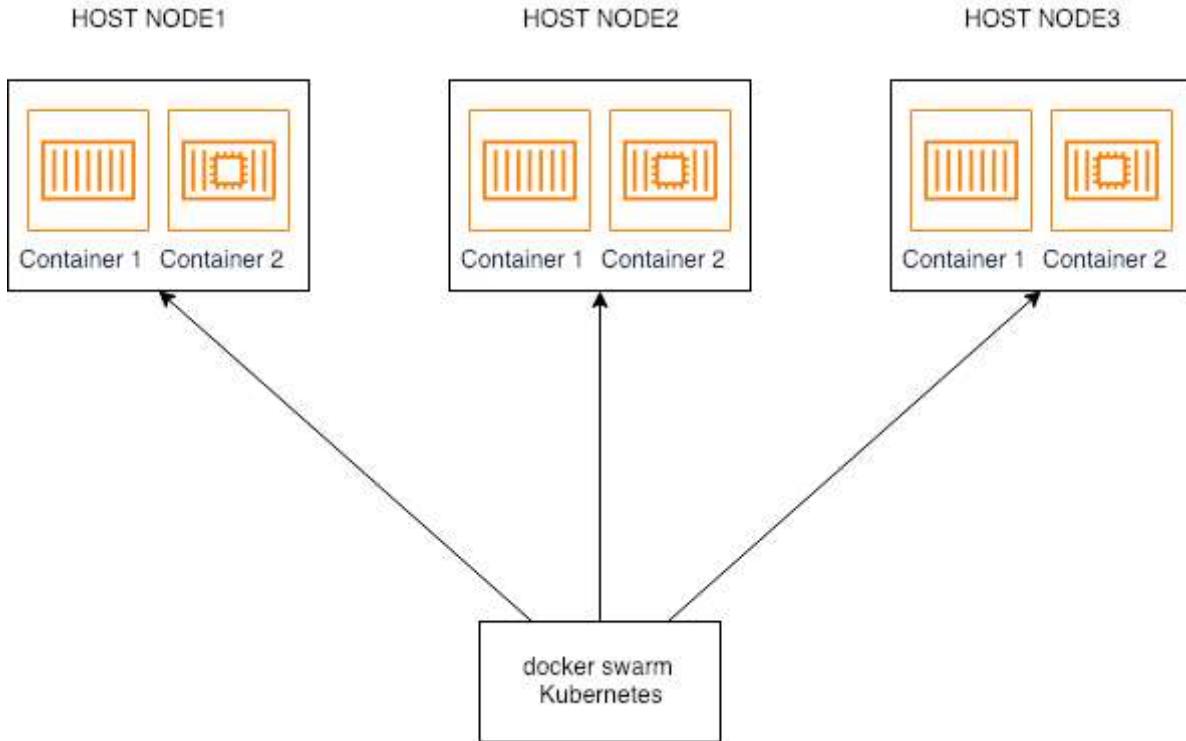
ngrok抓網址：[+在docker宿主機上自動生成secret\\_key](#)

```
[iiii@localhost step0]$ docker run -d --name flask --rm -p 5000:5000 ping331/flask_test:0.1
018a211a73c33b2b4d60ff8815acbe6118c11be8cd27792087b84ac601f1171a
[iiii@localhost step0]$
[iiii@localhost step0]$ docker run -d --rm --name ngrok -p 4040:4040 --link flask wernight/ngrok ngrok http flask:5000
3d4e8c34a8f1f4289d419f30c0954c284530b6aa33dfe85e6cde0699af64c42
[iiii@localhost step0]$ curl -s "localhost:4040/api/tunnels" | awk -F',' '{print $3}' | awk -F'{"' '{print $4}' | awk -F'{}' '{print $2}''
89e798e3.ngrok.io
[iiii@localhost step0]$ curl https://89e798e3.ngrok.io
Hello World![iiii@localhost step0]$
```



Hello World!

補充：大型叢集部署用：K8s及dockr swarm



在大型叢集的部署上，一般會使用kubernetes(K8s)或是docker swarm  
比較：[來源](#)

- **K8S :**
  - 可以支援多種 Container 工具，像是 Docker、Rocket
  - Kubernetes 有提供自己的 Restful API
  - 如果需要使用 Docker-Compose 需要先轉換
- **Swarm :**
  - 只支援 Docker 的 Container
  - 直接使用 Docker Restful API
  - 直接就可以使用 Docker-Compose

## docker-compose

### docker compose 介紹

之前有介紹過使用 `docker run` 指令就可以把 Docker Container 啟動起來，但是如果我們要啟動很多個 Docker Container 時，就需要輸入很多次 `docker run` 指令，另外 `container` 和 `container` 之間要做關聯的話也要記得它們之間要如何的連結(link) Container，這樣在要啟動多個 Container 的情況下，就會顯得比較麻煩。因此就出現了 Docker-Compose，只要寫一個 `docker-compose.yml`，把所有要使用 Docker Image 寫上去，另外也可以把 Container 之間的關係連結(link)起來，最後只要下 `docker-compose up` 指令，就可以把所有的 Docker Container 執行起來，這樣就可以很快速和方便的啟動多個 container。

[來源](#)

- docker-compose是基於YAML格式撰寫，非常注重階層的概念，而最大的忌諱就是不支持“tab”作為定位字元，這點需特別注意。

### Step11 : docker compose 安裝

至[官網](#)查看LINUX安裝方法

```

1 #安裝
2 sudo curl -L "https://github.com/docker/compose/releases/download/1.23.1/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
3 #權限
4 sudo chmod +x /usr/local/bin/docker-compose
5 #偷懶
6 ./install-compose.sh

```

安裝完成後使用docker-compose -v查看版本及是否安裝成功

```
[[iii@localhost step11]$ docker-compose -v
docker-compose version 1.23.1, build b02f1306
```

### Step12 : docker-compose.yml 常用模板參數介紹

下面我們開始逐步講解docker-compose.yml檔案該怎麼撰寫：

- 首先指定docker-compose的版本：

```
version: '3'
```

- 接下來開始定義服務名稱：

```
1 version: '3'
```

```
2 services:
```

```
3   flask:
```

- 指定IMAGE有兩種方式

- image：直接指定IMAGE

```
1 version: '3'
```

```
2 services:
```

```
3   flask:
```

```
4     image: python:3.7
```

- build：透過dockerfile 建構

- dockerfile在同一個資料夾且檔名為dockerfile

```
1 version: '3'
```

```
2 services:
```

```

3   flask:
4     build: .

```

- 指定dockerfile路徑(例如:在dockerfile資料夾內的dockerfile-jupyter)

```

1 version: '3'
2 services:
3   flask:
4     build:
5       context: ./dockerfile
6       dockerfile: dockerfile-flask

```

- **container\_name** : 定義容器名稱

```

1 version: '3'
2 services:
3   flask:
4     build:
5       context: ./dockerfile
6       dockerfile: dockerfile-flask
7       container_name: jupyter

```

- **depends\_on** : 解決容器相依問題，在哪些服務開啟之後才啟動

例如：ngrok服務運行前，要先確定flask服務已經開啟了

```

1 version: '3'
2 services:
3   flask:
4     ...
5     ...
6   ngrok:
7     images: wernight/ngrok
8     depends_on:
9       - flask

```

- **restart** : 服務意外停止後，是否重啟

```

1 version: '3'
2 services:
3   flask:
4     build:

```

```

5   context: ./dockerfile
6     dockerfile: dockerfile-flask
7     container_name: flask
8     restart: always

```

- **environment** : 指定環境變量

```

1 例如，在MySQL常用到的指定root_password
2   environment:
3     - MYSQL_ROOT_PASSWORD=iii

```

- **ports** : 指定映射端口，相當於**docker run**的-p

```

1 version: '3'
2 services:
3   flask:
4     build:
5       context: ./dockerfile
6         dockerfile: dockerfile-flask
7         container_name: flask
8         restart: always
9     ports:
10      - "5000:5000"

```

- **volumes** : 資料夾映射，相當於**docker run**的-v

```

1 version: '3'
2 services:
3   flask:
4     build:
5       context: ./dockerfile
6         dockerfile: dockerfile-flask
7         container_name: flask
8         restart: always
9     ports:
10      - "5000:5000"
11     volumes:
12       - ./code:/tmp/py_code

```

- **working\_dir** : 指定容器中工作資料夾，類似**dockerfile**的**WORKDIR**

- **command**：容器開啟後默認執行的命令，類似**dockerfile**的**CMD**

```

1 version: '3'
2 services:
3   flask:
4     build:
5       context: ./dockerfile
6       dockerfile: dockerfile-flask
7     container_name: flask
8     restart: always
9     ports:
10      - "5000:5000"
11     volumes:
12      - ./code:/tmp/py_code
13     command: python /tmp/py_code/app.py

```

- **dockerfile-flask**內容

```

1 #基於IMAGE python:3.7
2 FROM python:3.7
3 #添加FLASK_VER為環境變數，並設置版本為0.12
4 ENV FLASK_VER 0.12
5 #更新apt套件庫、安裝Flask，版本由FLASK_VER讀入、安裝VIM
6 RUN pip install -q flask==$FLASK_VER
7 #切換到預設工作目錄/tmp/py_code
8 WORKDIR /tmp/py_code

```

## **docker-compose**命令介紹

### 開啟一個**docker-compose**應用

```

1 docker-compose [-f 位置/檔名] up [-d]
2 -f：指定docker-compose的名稱，當docker-compose的名稱不是預設值(docker-compose.yml)時，需使用
3 -d：背景執行

```

```
[iii@localhost docker-compose_tutorial]$ docker-compose -f step1/jupyter-compose up -d
Creating network "step1_default" with the default driver
Creating jupyter-notebook ... done
```

### 關閉現行的**docker-compose**應用

- 1 docker-compose [-f 位置/檔名] down
- 2 注意：**docker-compose up**與**down** 建議在**docker-compose.yml**的同一層進行操作，不同層需使用 -f 參數指定檔名及位置

```
[ii@localhost docker-compose_tutorial]$ docker-compose -f step1/jupyter-compose down
Stopping jupyter-notebook ... done
Removing jupyter-notebook ... done
Removing network step1_default
```

### 連入container：

連入container的方式有兩種

- docker exec
- docker-compose exec

- 1 docker-compose exec範例：
- 2 docker-compose exec <service名稱> <命令>

### 練習：開始撰寫第一個**docker-compose**

將Step8、Step10的docker應用寫成docker-compose

### chatbot開發歷程與應用環境展示與說明

#### 其他補充

**docker-compose** 還可以做什麼？

比如說**ELK**：



**deviantony/docker-elk**

The ELK stack powered by Docker and Compose. Contribute to deviantony/docker-elk development by creating an account on...

[github.com](https://github.com/deviantony/docker-elk)

 [deviantony/docker-elk](https://github.com/deviantony/docker-elk) • [github.com](https://github.com/deviantony/docker-elk)

**你離**ELK**只有一句**docker-compose**的距离**

[WordPress : Quickstart: Compose and WordPress](#)

還有許許多的**docker-compose**應用散步在網路上，下次，再決定安裝應用之前，不妨想想：**docker** 或**docker-compose**能不能幫我更簡單地做到？