Docker - build environment

#使用Docker的指令建置環境

container管理

使用HTTP server服務-Apache

```
#Apache 的 image 名稱叫 httpd
docker run --name web httpd

#停止 Apache 後，移除已停止的container
docker ps -a
docker rm web

# 加帶 --run 參數 跑完自動刪除
docker run -rm --name web httpd
docker ps -a #檢查


#Ctrl+C & Ctrl+Q強制停止
```

背景執行 container

```
#執行完馬上回到host上，適合用於長時間運行的server上面
docker run -d httpd
#用於執行不停的程式
```

強制移除container

```
#執行完馬上回到host上
docker run -d --name web httpd

#確認正在執行中
docker ps

#執行中無法刪除container，使用rm -f發出docker kill的sigkill訊號，用於error終止上
docker rm -f web
```

 # -d用於執行不停的程式
 # --rm用於執行完就離開的時候

使用port forwarding開放服務

Docker ⇒ 啟動HTTP server ⇒ 背景執行container ⇒瀏覽器看見預測頁面

```
#背景啟動 Apache
docker run -d --name web httpd

#確認ports
docker ps

# curl 試試 HTTP 服務，也可以用瀏覽器測試
curl http://localhost/
#被拒絕了??why?? #存取被拒，host連線方式有誤

#背景啟動 多個pache
docker run -d --name web httpd
docker run -d --name web1 httpd
docker run -d --name web2 httpd
docker run -d --name web3 httpd
#查看多個container，所有container沒有衝突代表彼此有獨立性!!各自有80 port可以用
docker ps
#如何從外界存取呢?
```

每個container都是獨立的(包括port也是)，該如何外界存取呢???
利用—publish 跟 -p 設定port forwarding

port fordwarding功能，可以應用於任何有開port的server，如MySQL、Redis、Memcached、自己寫的服務等

```
# 加上 -p 參數
docker run -d --name web1 -p 8080:80 httpd
# 轉接到 container 開啟的 80/tcp port 確認 port 有被開出來
curl http://localhost:8080/

# 如果有兩個 container 重複 8080:80 的話就會 error "port is already allocated" !!
```

使用Volume同步檔案 ⇒ 進去container中修改檔案，再重新裝container檔案就回來了!

```
#啟動 Apache
docker run -d --name web1 -p 8080:80 httpd
curl http://localhost:8080/

#使用exec 進入container，與啟動shell指令的 bash
docker exec -it web1 bash

#找出apache 回 curl 的 html
cat htdocs/index.html

#使用新的內容取代indel.html 並離開 container
echo Hello volumn > htdocs/index.html && exit

#離開並測試
curl http://localhost:8080/

#移除container
docker rm -f web1

#再啟動一次
docker run -d --name web1 -p 8080:80 httpd

#確認內容
curl http://localhost:8080/
```

使用Bind Mount

如何讓寫好的程式出現在container裡面? 方法很多種，而Bind Mount是其中一種方法

Bind Mount 可以把host的檔案或目錄，綁定再container的某個目錄或檔案

使用 "-v" 將當前目錄綁定再Apache container 的 目錄下!

```
# 執行 container
docekr run -d -it -v 'pwd:/user/local/apache2/htdocs' --name web -p 8080:80 httpd

# 測試對應路徑
curl http://localhost:8080/test.html
```

使用Network 連結 container

網路是由多層式組成的，如三層式架構需要application server跟RDBMS兩種server連結起來，來提供服務

觀察 container log，使用兩個Terminal來觀察

```
# Terminal 1
docker run --rm -it --name web -p 8080:80 httpd

# Terminal 2
curl http://localhost:8080/
```

測試imgae功能時，測試關閉就會自動移除，需要使用-d時就會改用docker logs指令來觀察log

```
# Terminal 1
docker run -d -it --name web -p 8080:80 httpd

# 加 --follow 或 -f 選項後，container log 只要有更新，畫面就會更新
docker logs -f web
#就可以看到container內部運作狀況

# Terminal 2
curl http://localhost:8080/

# ctrl+C 離開
```

連結 Container

使用 —link 參數可以讓 container 透過 NAMES 來認識另一個 container :

```
# wget 為 curl 的替代指令
docker run --rm busybox --name no-link wget -q -o - http://web/

# 使用link的差異
docker run --rm --link web busybox wget --name have-link -q -o - http://web/

# 使用別名，使用後web跟alias都會生效
docker run --rm --link web:alias busybox wget -q -O - http://alias/
```

docker run image 後面可以加上想要執行的指令為何

```
# Busybox 裡面執行的其他程式
docker run --rm busybox whoami
docker run --rm busybox ls
docker run --rm busybox wget -q -O - http://web/
docker run --rm --link web busybox wget -q -O - http://web/
```

```
#BusyBox 預設為執行 sh，因此下面兩個指令結果相同
docker run --rm busybox
docker run --rm busybox sh
```

使用Network (官方建議方法)

使用Network建立虛擬網路，container在網路裡可以使用container name或hostname互相連結

docker network為Network元件的主要指令，其中有個指令為create為建立虛擬網路，可以設定連線模式、IP range、gateway參數等，建立時需要給一個唯一的名稱，本例使用my-net

當兩個container處於相同虛擬網路時，就可以相互使用container name 跟 network名稱 找到對方了!

```
# 建立 network
docker network create my-net

# Terminal 1 啟動Apache
docker run --rm --name web -p 8080:80 --network my-net httpd

# Terminal 2 透過busybox 連結Apache，加上--network 指定使用哪個虛擬網路
docker run --rm --network my-net busybox wget -q -O - web
```

—link的問題

—link 目前有個問題是無解的，建議以使用 Network 為主

link 的用法有先後順序之分，也就是被 link 的會不知道誰 link 它，所以 link 無法做到相向或循環連結

先建立main container 連到後建立sub container

```
# Terminal 1 啟動 main container
docker run --rm -it --name main busybox

# Terminal 2 啟動 sub container
dockr run --rm -it --name sub --link main busybox

# Terminal 2 ping main
```

```
ping main

# Terminal 1 ping sub
ping sub
 #被 link 的main會不知道誰 link 它
```

使用 environment 控制環境變數

經常同個 image 開啟多個 container，而且每個container都要能夠使用不同 DB 連線設定

如果使用volume共享設定檔可解決，但管理上百份container跟設定就是不這麼輕鬆了!

解決辦法 ⇒ 使用 —environment or -e 參數來指定 key=value 即可，就會設定到 container 的環境變數裡了

```
# 查看原本的 env
docker run --rm busybox env

# 給 env 設定後再看 env 的內容
docker run --rm -e DB_HOST=mysql busybox env
```

不同程式使用取得環境變數的方法不相同

```
#PHP
docker run --rm php -r "echo getenv('DB_HOST');"
docker run -rm -e DB_HOST=my_sql php -r "echo getenv('DB_HOST');"

#node.js
docker run --rm node node -e "console.log(process.env.DB_HOST)"
docker run --rm -e DB_HOST=mysql node node -e "console.log(process.env.DB_HOST)"
```

如何應用 environment？

evironment會搭配實際運行的需求做設定

示範MYSQL_ROOT_PASSWORD，他會在Percona啟動的時候設定root帳號的密碼

```
# Termianl 1
docker run --rm -it --name db -e MYSQL_ROOT_PASSWORD=pass percona

# Terminal 2
docker run --rm -it --link db percona mysql -hdb -uroot -ppass

#MySQL 指令, #client端
show databases;
```

MYSQL_DATABASE 則是啟動後建立對應名稱的資料庫

client 最後執行的show database; 會出現多一個some

```
# Terminal 1
docker run --rm -it --name db -e MYSQL_ROOT_PASSWORD=pass -e MYSQL_DATABASE=some percona

#Terminal 2 , -u=user -p=password
docker run --rm -it --link db percona mysql -hdb -uroot -ppass

#MySQL 指令, #client端
show databases;
```

MYSQL_USER 設定root以外的新帳號

MYSQL_PASSWORD 設定root以外的新密碼

同時給予MYSQL_DATABASE所有存取權限

```
# Terminal 1
docker run --rm -it --name db -e MYSQL_RANDOM_ROOT_PASSWORD=yes -e MYSQL_USER=wunyu -e
MYSQL_PASSWORD=1234 -e MYSQL_DATABASE=some percona

# Termical 2
docker run --rm -it --link db percona mysql -hdb -uwunyu -p123

#MySQL 指令, #client端
```

使用phpMYAdmin 連結 Percona 資料庫

```
# Terminal 1, 使用 MYSQL_ROOT_PASSWORD 與 MYSQL_DATABASE
docker run --rm -it --name db -e MYSQL_ROOT_PASSWORD=pass -e MYSQL_DATABASE=some perco
na

# Terminal 2
docker run --rm -it --link db -p 8080:80 phpmyadmin

#這個範例應用連結container與環境變數功能，讓host能透過phpMyadmin管理資料庫
```