多奇·教育訓練

容器化技術概念介紹與實作

徹底掌握最新的容器技術

多奇數位創意有限公司

技術總監 黃保翕（Will 保哥）

https://blog.miniasp.com

Introduction to Containers

認識 Container 容器技術

簡介容器技術 (Container Technology)

- 將「軟體/應用程式」封裝成一個標準化的單位「容器映像」(container image)
- 所謂「容器映像」會包含軟體所需的執行環境、系統工具、系統設定與主程式
- 執行這份「軟體」的必要檔案、設定、工具，都會存在所謂的「容器映像」中
- 所謂「容器」則是基於這份「容器映像」的「執行個體」(Running instance)
- 透過「容器」來部署應用程式，可以徹底抹平軟體在不同主機運行的差異！
- 「容器技術」是一種輕量的作業系統虛擬化技術 (Windows, Linux)

什麼是容器 (Container)？

- **容器**可以有效**隔離**應用程式在**同一個作業系統**中運行時**對彼此的影響**。
- **容器技術**用來**標準化**應用程式**發行**與**部署**的方式，並允許應用程式可以執行在 Linux 或 Windows 容器在相對應的 Host 作業系統上。
- **容器**會共享 Host 作業系統上的核心 (Kernel)，
  因此在執行時會比透過**傳統虛擬機器**來的更為**輕量**！

容器化的絕佳優勢

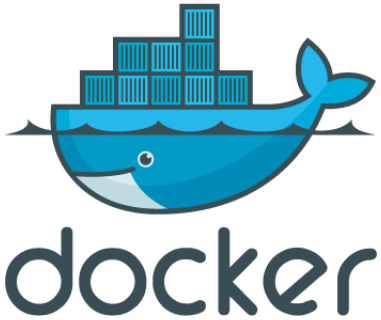
- **應用程式隔離 (Isolation)**
  - 應用程式在同一台 Host OS 上不會互相影響
- **可攜性非常好 (Portability)**
  - 大幅消除環境設定帶來的各種困擾 (問題、錯誤)
- **管理非常敏捷 (Agility)**
  - 啟動容器的速度極快，就跟啟動應用程式一樣
- **可擴充性極高 (Scalability)**
  - 階層化的容器映像可隨時擴充與更新
- **可控制整個應用程式開發生命週期/工作流程**
  - 有效隔離 Dev 與 Ops 之間的工作流程 (責任釐清)

使用容器技術的好處 - 軟體部署方面

- 容器技術可以讓你動態**改變不同的設定**、**新增功能**、**橫向延展服務**，更能迅速的反應需求的變化！

- 透過 **微服務** (Micro-service) 架構，應用系統中不同類型的服務都可透過容器技術進行**分類**與**管理**，搭配適合的**容器管理工具**，就能做到**動態資源分配**與**分散式軟體部署**的目的。

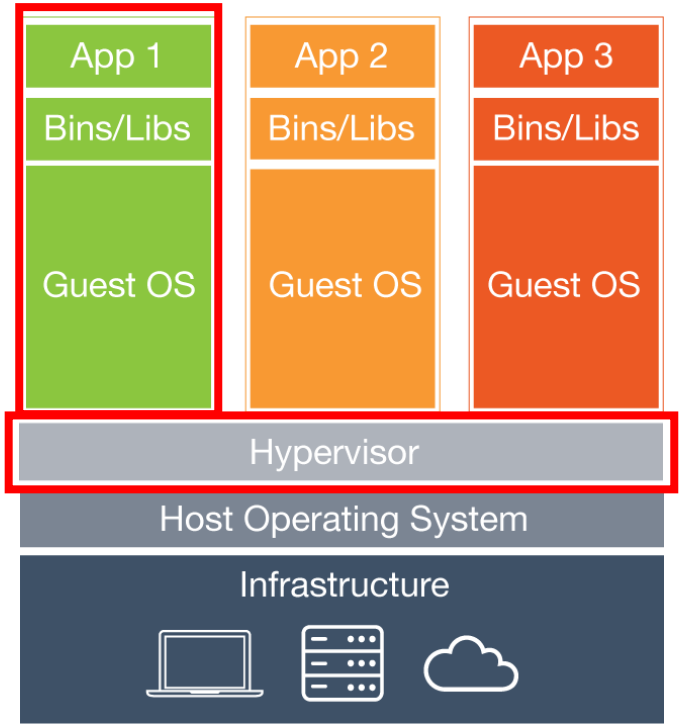
什麼是 Docker？

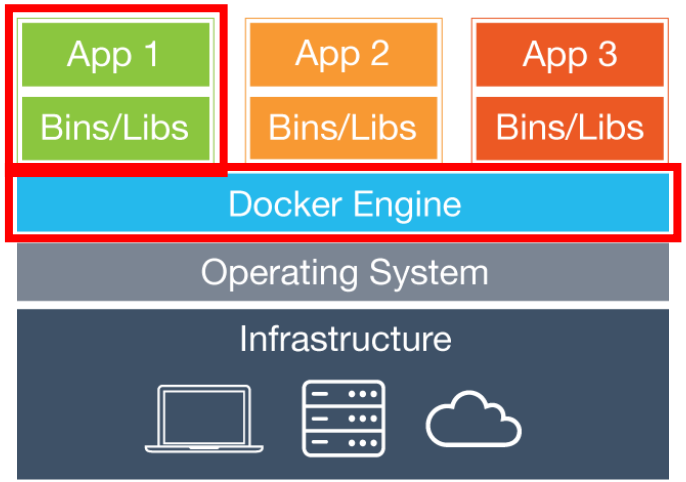


- 是一間負責提供容器技術的公司
  - 提供多家雲端與作業系統技術支援
- 是一個完整實作容器技術的開放原始碼專案
  - 這套技術幾乎成為業界公認的標準容器技術平台，採用此技術的公司有 Microsoft, Amazon, Google, ...
  - 完整實作**作業系統虛擬化**容器技術
    - 原生支援 Linux 與 Windows 作業系統
      - ➢Linux 容器必須執行在 Linux 作業系統上
      - ➢Windows 容器必須執行在 Windows 作業系統上

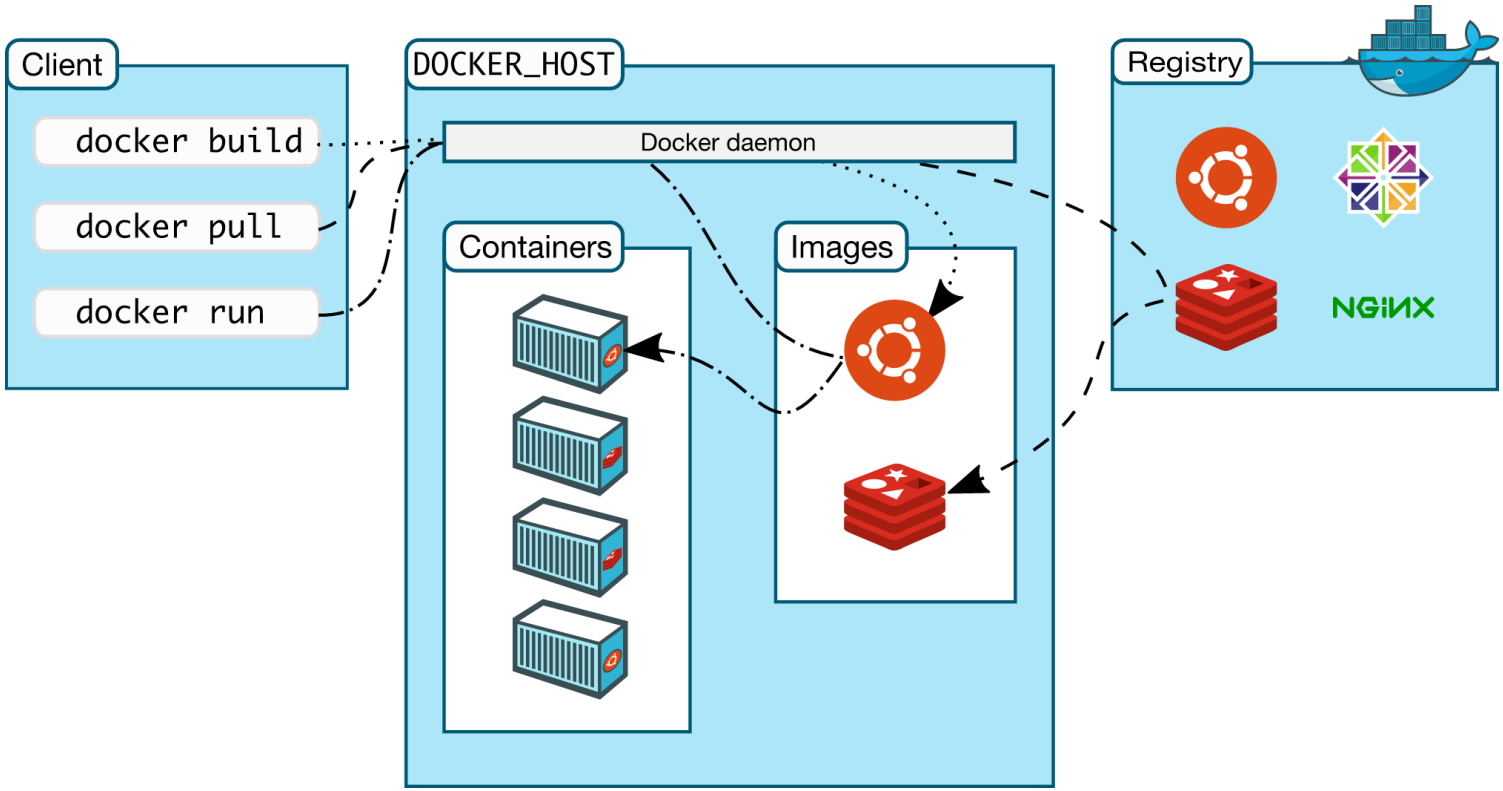
作業系統虛擬化

- Docker 實作**輕量級**的**作業系統虛擬化**解決方案！





認識 Docker 主從式架構

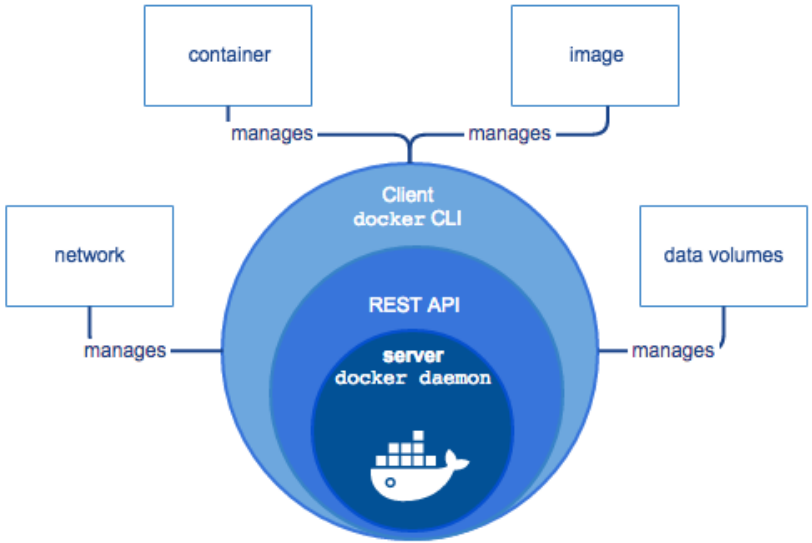


關於 Docker Engine

- 採用**主從式架構** (client-server architecture)
- 長時間執行的伺服器軟體，負責管理以下物件
  - images
  - containers
  - networks
  - data volumes
  - …
- 有時候也稱為 **Docker Daemon** 或 **Docker Server**
- 提供一組標準的 REST API 介面 (可遠端呼叫)
- 提供一組 Command-Line Interface (CLI) 用戶端介面

關於 Docker Client

- 通常透過 CLI 用戶端介面進行管理，也可透過 GUI 工具進行管理
- CLI 用戶端介面骨子裡其實是用 REST API 進行操作
- 可以控制多台遠端的 Docker Engine（主從式架構）

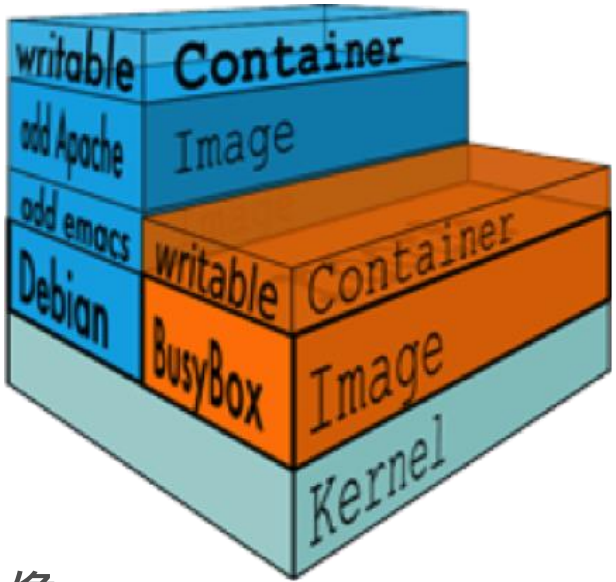


什麼是 docker image？

- 一個 docker container 的基礎映像
- 一個 docker image 包含
  1. 包含一系列從 root filesystem 開始的所有變更紀錄
  2. 一組預備給 container runtime 執行的命令與參數
- docker image 都是**無狀態的**且**永遠不會更新**！
  (唯讀) (無法寫入)

認識 Docker 映像階層架構 (Image Layers)

- Docker 映像檔採用一種**分層堆疊**的運作方式
  - 早期採用了 aufs 檔案系統
  - 後來發展出 OverlayFS 檔案系統（`overlay`, `overylay2`）
  - 分層、輕量級並且高效能的檔案系統
  - Union 檔案系統是 Docker 映像檔的基礎



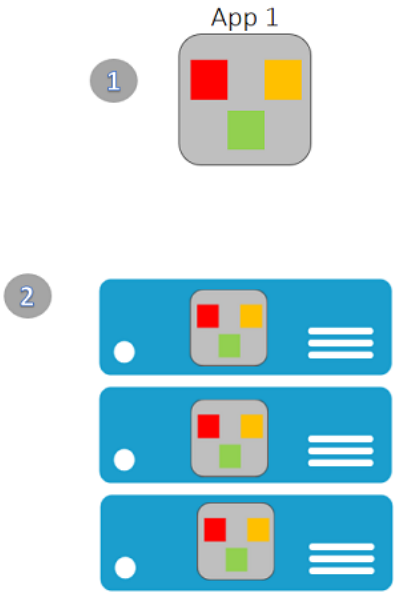
  - **映像**其實是由**多個映像**堆疊而成的
  - 建立映像的同時，**每個步驟都會建立一層映像**

什麼是 docker container？

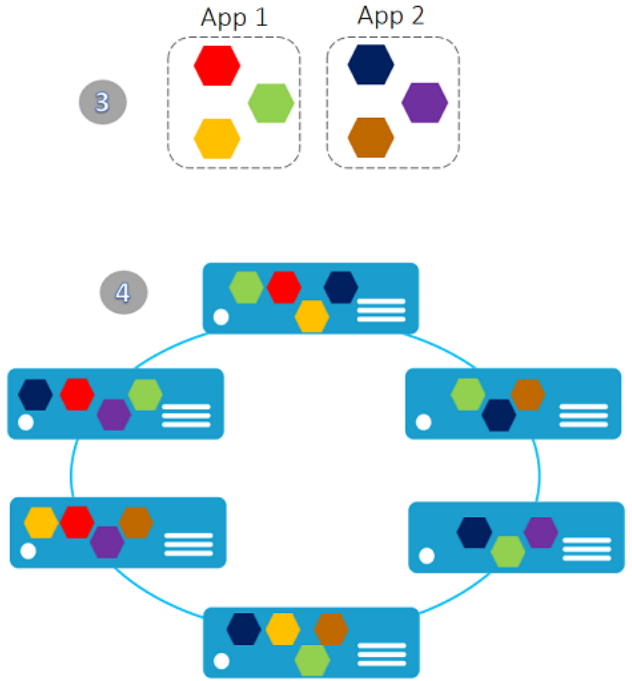
- 一個 docker image 的執行個體 (runtime instance)
- 一個 docker container 包含
  1. 一份基礎容器映像 (docker image)
  2. 應用程式的執行環境 (Execution environment) (**可讀寫**)
  3. 一組預備執行的命令集 (A standard set of instructions)

應用程式開發方法的比較

Monolithic application approach



Microservices application approach



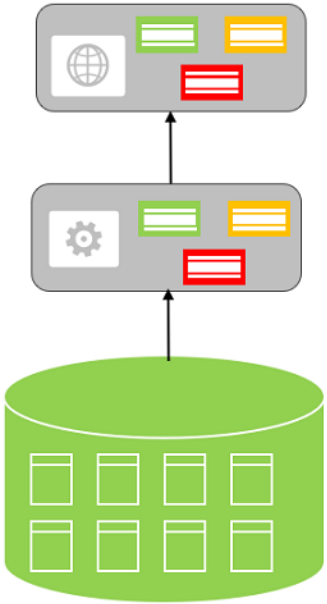
了解微服務架構 (Microservice Architecture)

- 何謂**微服務**
  - **獨立的服務**共同組成整個應用系統
  - **個別的服務**都可以**獨立部署**與**運作**
  - **每一個服務**都能夠**獨立開發**與**維護**
  - 分散式的管理（可延展性高）
- 微服務的目的
  - 將應用程式**拆分**成**多個服務**
  - 實現**敏捷開發**和**部署自動化**

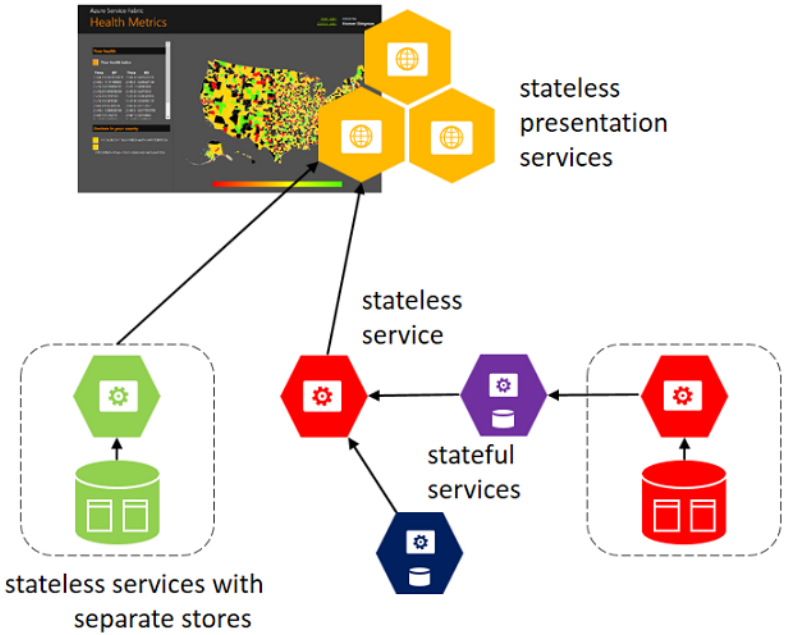
為何要用微服務方式建置應用程式？

應用程式類型之間的狀態儲存比較

State in Monolithic approach



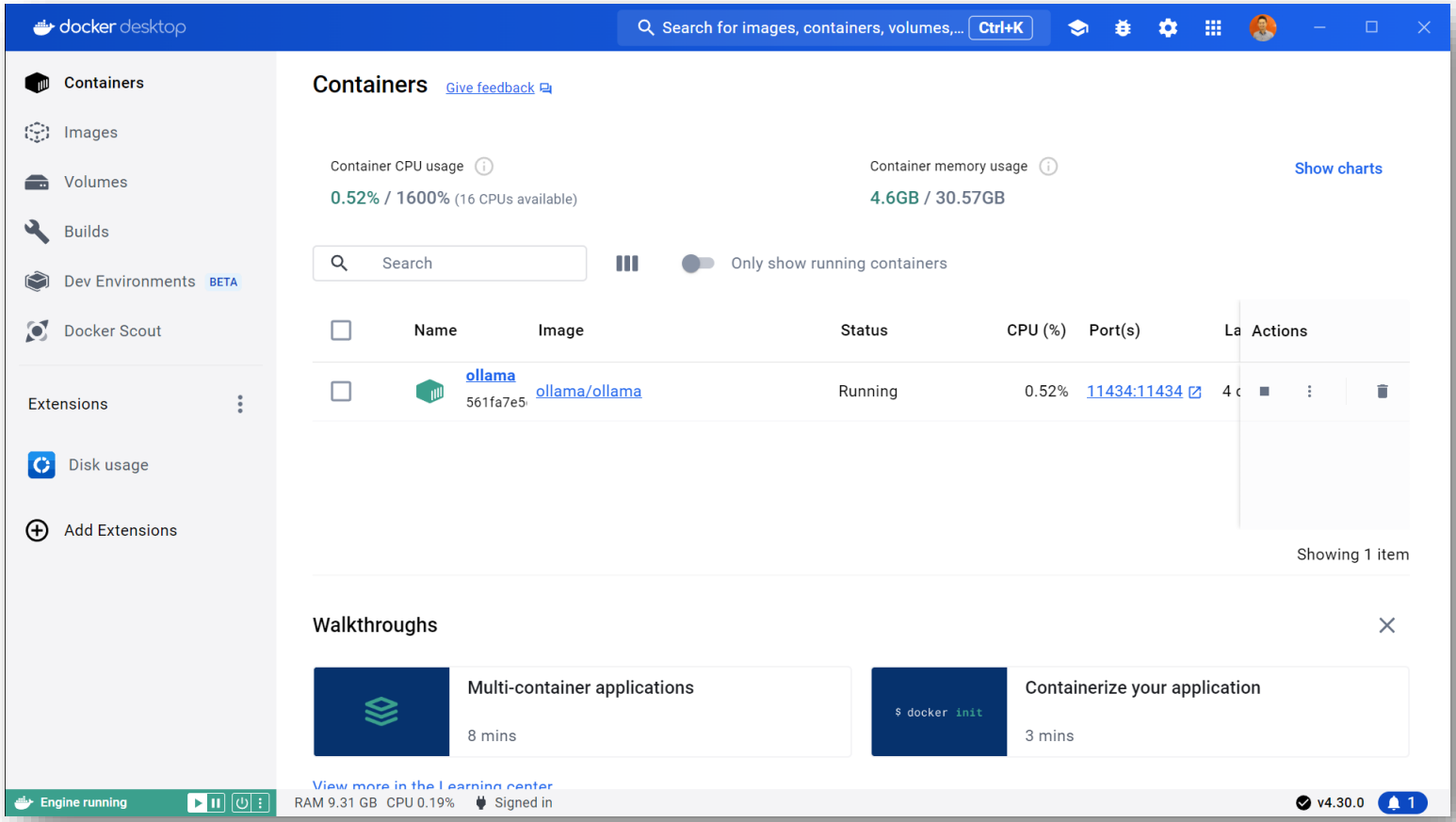
State in Microservices approach



Containers Quick Start

快速上手 Container 容器執行

認識 Docker Desktop



執行你的第一個 Windows 容器

- Hello World

```
docker run -it --rm hello-world
```

- Busybox

```
docker run busybox echo "hello world"
```

- Ubuntu

```
docker run -it ubuntu bash
```

管理容器生命週期與基本指令

- 執行容器
  - docker run -it IMAGE COMMAND
- 列出容器
  - docker ps
  - docker ps -a
- 啟動指定容器
  - docker start CONTAINER
- 停止指定容器
  - docker stop CONTAINER
- 附加目前終端機的串流到容器中
  - docker attach CONTAINER
- 在指定容器中執行命令
  - docker exec -it CONTAINER cmd
- 刪除指定容器
  - docker rm CONTAINER
- 查詢完整命令
  - docker container
  - docker container COMMAND --help

管理容器映像生命週期與基本指令

- 列出容器映像
  - docker image ls
  - docker image ls -a
- 下載/上傳容器映像
  - docker image pull IMAGE
  - docker image push IMAGE
- 查詢容器映像階層歷史紀錄
  - docker image history
- 刪除指定容器映像
  - docker image rm IMAGE
- 建立容器映像 (從現有容器建立)
  - docker commit CONTAINER IMAGE
- 建立容器映像(從 Dockerfile 建立)
  - docker build -t REPO:TAG PATH
- 標記容器映像 (有點像替映像取個別名)
  - docker tag IMAGE[:TAG] NEWIMAGE[:TAG]
- 查詢完整命令
  - docker image
  - docker image COMMAND --help

執行容器的參數說明

- 執行以下命令

```
docker run --name test1 -it --rm busybox:latest echo "hello world"
```

- 參數說明

run	執行新的容器 (建立容器)
--name test1	指定新建立的容器名稱 test1
-it	
-i	保持容器中的應用程式接受 STDIN 管道輸入
-t	配置一個虛擬終端機 (pseudo-TTY) 讓你輸入命令
--rm	當容器中的應用程式結束就自動刪除容器
busybox	基礎映像 (Base image)
:latest	基礎映像的標籤名稱 (Tag name)
echo "hello world"	在容器中執行的應用程式

每個容器只會有一個主程式

- 每次執行容器時，都必須指定一支「**主程式**」
- **主程式**只要一**退出**，**容器也會跟著被關閉**！
- 退出容器不讓容器停止的方法
  - 直接按下快速鍵：**Ctrl-p Ctrl-q**
- 讓容器**永遠不停止**的方法 (只要程式停止就會立刻砍掉重練)

```
docker run -it --restart=always IMAGE COMMAND
```

常用 PowerShell 命令

- 批次刪除所有容器 (info)
  - `docker rm $(docker ps -a -q)`
- 批次刪除所有 Exited 的容器 (info)
  - `docker rm $(docker ps -a -f "status=exited" -q)`
- 取得 Docker 資訊
  - `$DockerInfo = (docker info --format '{{json .}}') | ConvertFrom-Json`
- 取得容器 IP 地址
  - `docker inspect --format "{{ .NetworkSettings.Networks.nat.IPAddress }}" 3f`
  - `docker inspect -f "{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}" 3f`
  - `(docker container inspect 3f | ConvertFrom-Json).NetworkSettings.Networks.nat.IPAddress`

Nginx Containers

下載並執行 Nginx 容器

下載基礎容器映像

- 下載 image
  - docker pull nginx
- 執行 container
  - docker run -d -p 8080:80 --name my-nginx nginx
- 查看網頁
  - http://localhost:8080/

執行 Nginx 容器

- 基本命令

```
docker run --name my-nginx -d -p 8080:80 nginx
```

- 參數說明

`run`	建立容器執行
`--name my-nginx`	指定新建立的容器名稱
`-d`	進入 detach 模式 (背景執行模式)
`-p 8080:80`	設定 Host 主機埠號 8080 對應到容器埠號 80
`nginx`	基礎映像名稱 (base image name)

建立容器映像標籤

- 使用方法

```
docker tag IMAGE[:TAG] IMAGE[:TAG]
```

- 使用範例
  - 建立預設標籤為 latest 的容器映像
    ```
    docker tag image_name newname
    ```
  - 建立**自訂標籤**的容器映像
    ```
    docker tag image_name:latest newname:v1.0
    ```
  - 刪除容器映像的**自訂標籤** (刪到最後一個就會刪除容器映像)
    ```
    docker rmi image_name:latest
    docker image rm image_name:latest
    ```

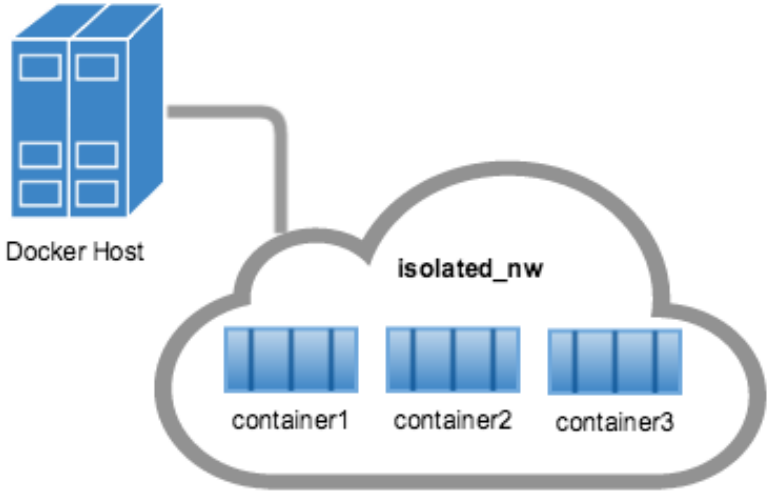
了解容器映像的標籤命名規則 - 1

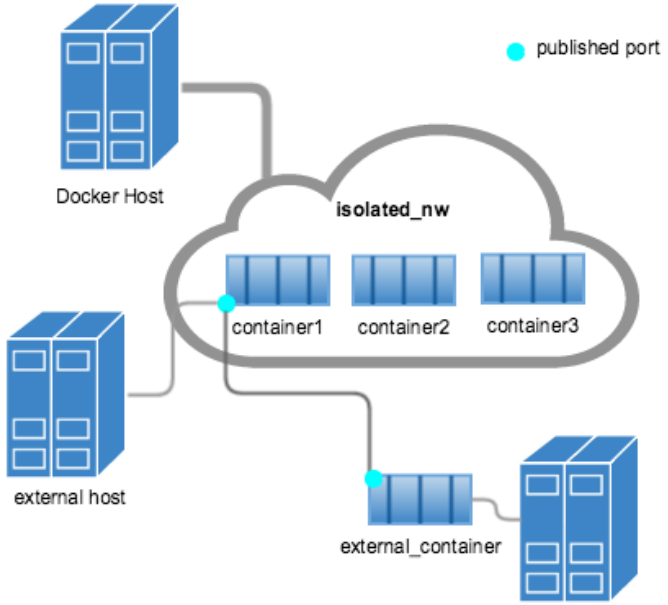
- **docker pull ubuntu**
  - 完整名稱等同於 **docker pull docker.io/library/ubuntu:latest**
  - 代表預設 Registry 為 **docker.io**（Docker Hub）
  - 代表預設 Username 為 **library**
  - 指定 Image Name 為 **ubuntu**
  - 代表預設 Tag 名稱為 **latest**
- **docker pull microsoft/iis**
  - 完整名稱等同於 **docker pull docker.io/microsoft/iis:latest**
  - 代表預設 Registry 為 **docker.io**（Docker Hub）
  - 指定 Username 為 **microsoft**
  - 指定 Image Name 為 **iis**
  - 代表預設 Tag 名稱為 **latest**

了解容器映像的標籤命名規則 - 2

- **docker pull microsoft/iis:4.7**
  - 完整名稱等同於 **docker pull docker.io/microsoft/iis:4.7**
  - 代表預設 Registry 為 **docker.io**（Docker Hub）
  - 代表 Username 為 **microsoft**
  - 指定 Image Name 為 **iis**
  - 代表 Tag 名稱為 **4.7**
- **docker pull moderndotnet.azurecr.io/duotify/myapp:1.0**
  - 代表預設 Registry 為 **moderndotnet.azurecr.io** (Docker Hub)
  - 指定 Image Name 為 **duotify/myapp**
  - 代表 Tag 名稱為 **1.0**

容器的網路環境





https://docs.docker.com/network/

查詢 Container 網路位址 (IP)

- 使用 Command Prompt + docker inspect

```
docker inspect --format="{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}" CONTAINER
docker inspect --format "{{ .NetworkSettings.Networks.nat.IPAddress }}" CONTAINER
```

命令參考：`docker inspect --help`

- 使用 PowerShell + docker inspect

```
((docker inspect CONTAINER) | ConvertFrom-Json).NetworkSettings.Networks.nat.IPAddress
```

- 使用 docker exec + ipconfig

```
docker exec CONTAINER ipconfig
```

在 Host 與 Container 之間複製檔案

- 使用方法
  - `docker cp [OPTIONS] CONTAINER:SRC_PATH DEST_PATH|-`
  - `docker cp [OPTIONS] SRC_PATH|- CONTAINER:DEST_PATH`
- 使用範例
  - 複製單一檔案

    ```
    docker cp a.txt container_name:/a.txt
    docker cp CONTAINER:/a.txt a.txt
    ```

  - 複製整個資料夾

    ```
    docker cp data CONTAINER:/data
    docker cp CONTAINER:/logs logs
    ```

使用 docker cp 的注意事項 - 1

- 使用 **Hyper-V 隔離模式** 無法使用 docker cp 命令！
- **啟動中**或**停止中**的容器都可以透過執行 docker cp 命令複製檔案
- 執行 docker cp 如同 cp -a 命令（預設複製所有子目錄/檔案/權限）
- 執行 docker cp -a 則會包含 uid/gid 都複製進去 (Linux container)
- 執行 docker cp -L 則會進入 Symbolic Link 指向的目錄複製檔案
- 容器的路徑預設是 / 根目錄為主使用相對路徑即可複製檔案：

```
- docker cp a.txt CONTAINER:aa
- docker cp a.txt CONTAINER:aa/
- docker cp a.txt CONTAINER:/aa
- docker cp a.txt CONTAINER:/aa/
```

使用 docker cp 的注意事項 - 2

- 如果 SRC_PATH 是個**檔案**
  - DEST_PATH 不存在
    - 會直接複製成檔案
  - DEST_PATH 不存在 (包含斜線結尾)
    - ***發生錯誤***
    - 必須事先建立資料夾
  - DEST_PATH 存在 (且是一個**檔案**)
    - 該檔案會被覆寫
  - DEST_PATH 存在 (且是一個**資料夾**)
    - 檔案會被複製進該目錄
- 如果 SRC_PATH 是個**目錄**
  - DEST_PATH 不存在
    - 會直接複製完整資料夾過去
  - DEST_PATH 不存在 (包含斜線結尾)
    - 會直接複製完整資料夾過去
  - DEST_PATH 存在 (且是一個**檔案**)
    - ***發生錯誤***
    - 不能將目錄複製為檔案
  - DEST_PATH 存在 (且是一個**資料夾**)
    - SRC_PATH 結尾是 **/.**
      - **來源資料夾**下的**檔案**會複製到**目的資料夾**下
    - SRC_PATH 結尾**不是 /.**
      - **來源資料夾**會被複製到**目的資料夾**下

建立容器映像

- 使用方法

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

- 使用範例
  - 建立**預設標籤**為 **latest** 的容器映像
    ```
    docker commit container_name willh/mylogs
    ```
  - 建立**自訂標籤**的容器映像
    ```
    docker commit container_name willh/mylogs:v1.0
    ```

- 相關連結
  - docker commit - Docker

設定容器與主機的共用資料夾 (Volume)

- 執行以下命令
  - `docker run --name app1 -d -p 80:80 -v C:\Projects\app:/app -w /app myimage`
- 參數說明

`-v`	VOLUME (shared filesystems)
• `C:\Projects\app`	容器主機 (host) 的資料夾 (**必須已經存在**)
• `/app`	容器實體 (instance) 的資料夾 (**可以不存在**) **如果資料夾已經存在，所有檔案會被忽略！**
`-w`	設定預設工作目錄
`--read-only`	將容器內的檔案系統設定為**唯讀模式**

實戰演練

1. 建立一個 nginx 容器執行，將容器命名為 **my-nginx**
2. 進入 **my-nginx** 容器
3. 在 **my-nginx** 容器中編輯一個 **/etc/nginx/conf.d/default.conf** 檔案

```
location / {
    proxy_pass https://www.skl.com.tw;
    proxy_set_header Host www.skl.com.tw;
    proxy_set_header X-Real-IP $remote_addr;
}
```

4. 重新載入設定檔 **nginx -s reload**
5. 檢查 http://localhost:8080/ 是否正常顯示網頁
6. 停止容器、刪除容器

IIS Containers

下載並執行 SQL Server 容器

執行容器

- 下載基礎容器映像

```
docker pull mcr.microsoft.com/mssql/server:2019-latest
```

- 執行 SQL Server 容器

```
docker run -e "ACCEPT_EULA=Y" -e
"SA_PASSWORD=Ver7CompleXPW"  -p 1433:1433 --name sql1
-d mcr.microsoft.com/mssql/server:2019-latest
```

- 進入 SQL Server 容器執行

```
docker exec -it sql1 "bash"
```

使用 Docker 執行 SQL Server on Linux 容器之常用工具與命令

啟動參數說明

- 啟動時指定環境變數（-e）

```
-e "SA_PASSWORD=Ver7CompleXPW"
-e "ACCEPT_EULA=Y"
```

- 設定 Port 通訊埠對應（-p）

```
-p 1433:1433
```

- 環境變數說明
  - SA_PASSWORD 必須設定 sa 密碼須符合 SQL Server 強式密碼複雜度要求
  - ACCEPT_EULA 必須設定為 Y 接受授權條款才能啟動 SQL 伺服器

Dockerfile

使用 Dockerfile 封裝應用程式

使用 Dockerfile 建置容器映像

- 建立工作區
  - C:\Build
- 建立 Dockerfile 定義檔
  - C:\Build\Dockerfile
- 編輯 Dockerfile 定義檔內容
  - FROM nginx
  - COPY default.conf /etc/nginx/conf.d/default.conf
- 建置容器映像
  - docker build -t IMAGE c:\Build
  - 參數說明
    - `-t IMAGE` 指定容器映像的名稱 (t = tag)
    - `c:\Build` 指定建置內容(build context)所在的目錄 (可以用 `.` 代表當前目錄)
      ※ 執行 **docker build** 時，會先將指定目錄複製進 Docker 才開始建置

關於 docker build 的參數用法

- 在當前目錄建置（自動在目錄中找尋 Dockerfile 檔）
  - `docker build .`
- 在當前目錄建置（指定 Dockerfile 檔案路徑）
  - `docker build -f /path/to/a/Dockerfile .`
- 在當前目錄建置（指定標籤名稱）
  - `docker build -t willh/myapp .`
- 在當前目錄建置（指定多重標籤名稱）
  - `docker build -t willh/myapp:1.0.2 -t willh/myapp:latest .`

執行 docker build 的注意事項

- 注意建置快取機制 (Build cache)
  - 建置的過程會依照 Dockerfile 定義的順序來逐條執行
  - 每執行一個命令 (instruction) 都會建立一個新的 image
  - 如果使用了相同的 base image 外加相同的指令，預設的情況下不會建立重複的 image (這就是建置快取機制)
  - 如要暫時停用建置快取，請在執行時使用以下參數
  - `docker build --no-cache=true`
- 使用 Dockerfile 建置映像最佳實務
  - 請參考 Best practices for writing Dockerfiles 文件

常用的 Dockerfile 命令 - 1

- FROM　　設定 base image
  - FROM mcr.microsoft.com/dotnet/framework/aspnet:4.7.2
- LABEL (Docker object labels) 定義自訂標籤 (Label)
  - LABEL maintainer="myname@example.com"
- WORKDIR　　定義預設工作目錄
  - 會套用到 RUN, CMD, ENTRYPOINT, COPY, ADD 等命令。
- ADD 或 **COPY**　　複製資料夾或檔案進容器
  - COPY <src>... <dest>
  - COPY ["<src>",... "<dest>"]

常用的 Dockerfile 命令 - 2

- ENV　　設定環境變數
  - ENV <key>=<value> <key2>=<value2> …　(建議格式)
  - 在 Dockerfile 的其他地方可以用 ${key} 來取得環境變數的內容
  - 啟動容器時 docker run 可以加入 -e 覆寫這裡定義的環境變數
- EXPOSE　　單純用來宣告容器對外公告的 port 為何
  - EXPOSE 80 2322
- SHELL　　定義預設 Shell 的啟動命令
  - SHELL ["powershell", "-Command", "$ErrorActionPreference = 'Stop'; $ProgressPreference = 'SilentlyContinue';"]
- VOLUME　　設定一個磁碟掛載點 (bind)
  - VOLUME ["/etc/nginx/conf.d"]

常用的 Dockerfile 命令 - 3

- ENTRYPOINT 指定容器啟動時要執行的命令 (只能設定一組)
  - 主要用途
    - 要把容器當成「工具程式」來跑的時候會使用這個命令
    - `docker run -it willh/sneakers`
    - `docker run -it willh/sneakers -a`
    - `docker run -it --entrypoint="" willh/sneakers bash`
  - 支援兩種不同的執行模式
    1. 執行檔模式
       `ENTRYPOINT ["executable","param1","param2"]`
    2. Shell 模式
       `ENTRYPOINT command param1 param2`

常用的 Dockerfile 命令 - 4

- CMD　　指定容器啟動時要執行的命令 (只能設定一組)
  - 主要用途
    - 要把容器當成「服務」來跑的時候會使用這個命令
    - 當使用 docker run 啟動容器的時候可以覆寫這個設定
  - 支援三種不同的執行模式
    1. 執行檔模式
       ```
       CMD ["executable","param1","param2"]
       ```
    2. 當成 ENTRYPOINT 的參數
       ```
       CMD ["param1","param2"]
       ```
    3. Shell 模式
       ```
       CMD command param1 param2
       ```

常用的 Dockerfile 命令 - 5

- RUN　　在**建置映像**的過程中會執行的命令
  - 主要用途
    - 當你想在建置 image 的過程中想執行程式的時候會用 (建置工作)
  - 注意事項
    - 每一次執行都會產生新的映像階層 (image layer)
  - 支援兩種不同的執行模式
    1. Shell 模式
       ```
       RUN command param1
       ```
    2. 執行檔模式
       ```
       RUN ["executable","param1","param2"]
       ```

關於 Shell 模式

- 注意事項
  - 使用 ENTRYPOINT 、 CMD 與 RUN 都有 Shell 模式可執行
  - Shell 模式執行的命令都會自動接在 SHELL 之後，組成完整命令來執行！

- Linux 容器的預設值
  - **SHELL ["/bin/sh", "-c"]**

- Windows 容器的預設值
  - **SHELL ["cmd", "/S", "/C"]**

```
# Executed as powershell -command Write-Host hello
SHELL ["powershell", "-command"]
RUN Write-Host hello

# Executed as cmd /S /C echo hello
SHELL ["cmd", "/S", "/C"]
RUN echo hello
```

- 設定以 PowerShell 為預設 Shell 的宣告方法
  - **SHELL ["powershell", "-Command", "$ErrorActionPreference = 'Stop'; $ProgressPreference = 'SilentlyContinue';"]**

使用跳脫字元 (Escape Characters)

- 跳脫字元的主要用途
  - 可以用跳脫某些特殊的符號（**& ！ $ " 空白字元**）
  - 可以用來**接續下一行**命令（使用範例）
- 兩種跳脫字元可供選擇
  - Dockerfile 預設跳脫字元為 \（反斜線）
  - 但是在 **Windows 容器** 建議改用 `（反引號）
    - 因為 Windows 平台下的**路徑分隔符號**是用反斜線（\）
    - 因為 PowerShell 預設跳脫字元也是 `（反引號）
- 宣告跳脫字元的方式 (在 Dockerfile 第一行加入以下註解)

```
# escape=`
```

多階段容器建置 (multi-stage build)

- 利用不同的容器來完成不同的工作
  - 在容器 A 中將原始碼**編譯**成**可部署的檔案**
    - 容器 A 中會包含建置專案所需的 SDK
  - 將**可部署的檔案**複製到容器 B 裡面
    - 容器 B 中僅包含 Runtime
  - 最後將容器 B 建立成 image
- 透過**多階段容器**建置，大幅降低部署映像檔大小！
  - 以 ASP.NET Core 8 為例

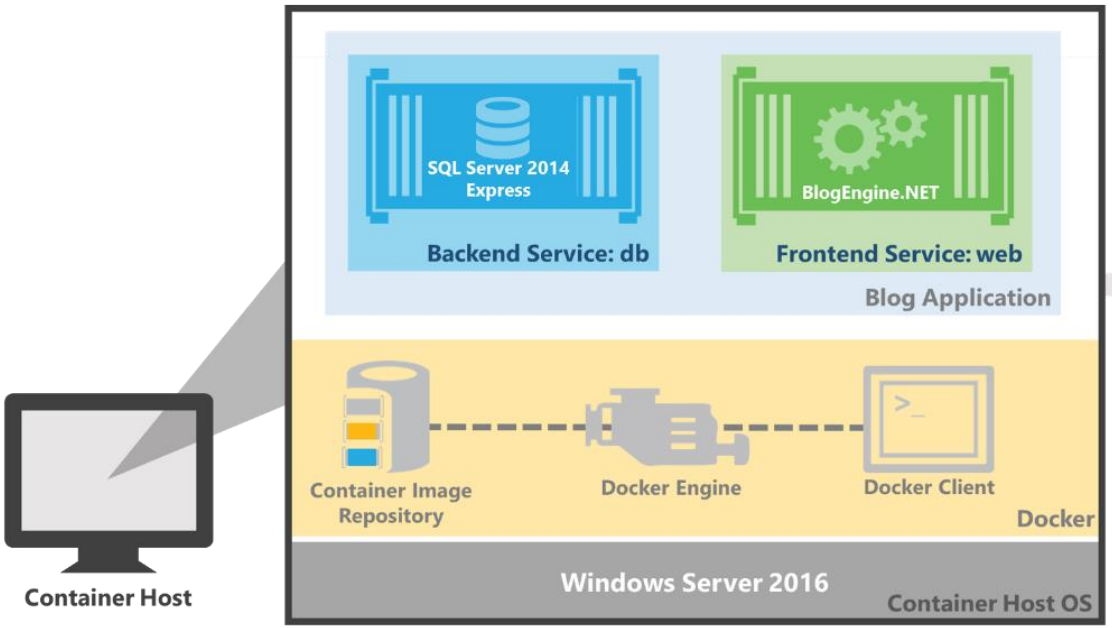
Docker Compose

使用 Docker Compose 管理多容器執行

認識 Docker Compose

- 可以同時執行/管理多個容器的工具
  - 方便用來建置複雜的開發測試環境
  - 快速建立隔離的複雜的應用程式執行環境
- 使用 Docker Compose 的三個步驟
  1. 定義 `Dockerfile` 建置步驟並用來建置映像檔
  2. 需要定義一個 docker-compose.yml 宣告檔
  3. 執行 `docker-compose up` 同時啟動多容器
- 參考連結
  - Docker Compose
  - Using Compose in production

使用 Docker Compose 的情境圖示




```yaml
# docker-compose.yml

version: '2'

services:
 web:
   image: blogengine
   ports:
    - "80:80"
   depends_on:
    - db
   tty:
     true
 db:
   image: blogdb
   expose:
    - "1433"
   tty:
     true

networks:
 default:
  external:
   name: "nat"
```

安裝 Docker Compose

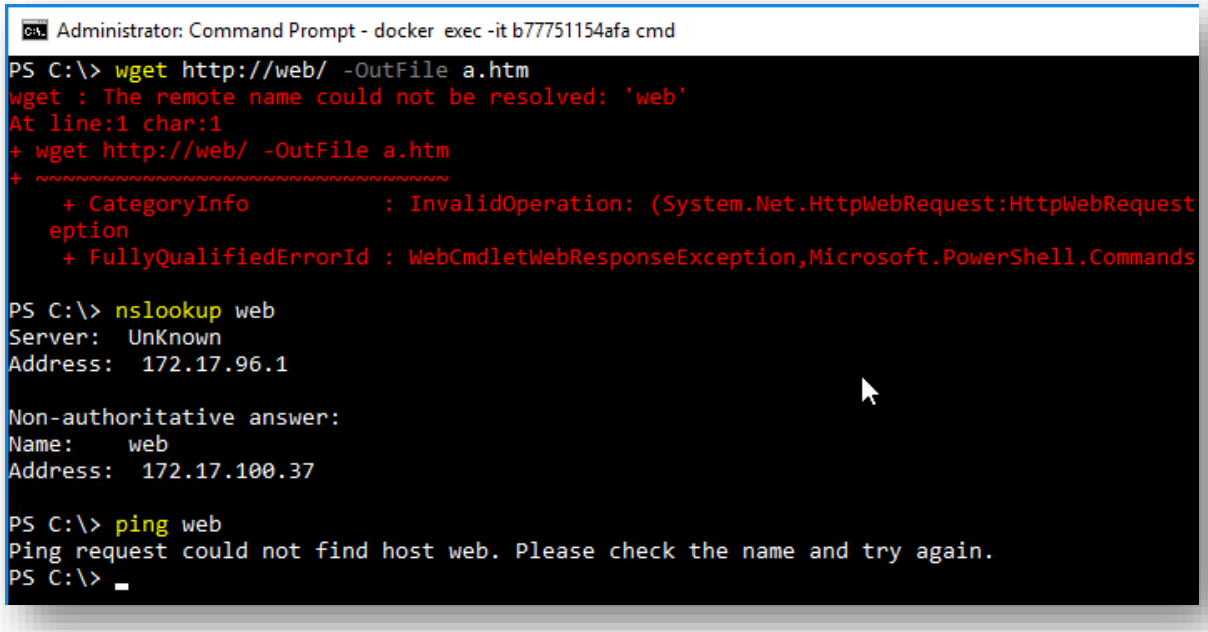
- 安裝 Docker Desktop for Windows 的時候就會自動安裝好
- 查看版本資訊
  - `docker-compose version`

```
C:\>docker-compose version
Docker Compose version v2.27.0-desktop.2

C:\>_
```

撰寫 docker-compose.yml 的注意事項

- 撰寫格式務必正確
  - 縮排請用 2 個空白字元，千萬不要用 Tab 符號縮排
  - 不要混合 4 個空白與 2 個空白的縮排間距！（否則如下）



發行容器映象到 Docker Hub

容器發行管理

- 自行架設私用的 Registry 服務
  - Harbor
  - Sonatype Nexus
  - Docker Registry (如何部署)
  - Docker Trusted Registry (DTR)
- 雲端架設私用的 Registry 服務
  - GitHub Container Registry
  - Azure Container Registry (ACR) (文件)
  - Amazon Elastic Container Registry (ECR)
    - Amazon ECR Public Gallery

發行容器映象到 Docker Hub

- 使用範例
  - 登入到 Docker registry
    `docker login`
  - 建立符合 Docker Hub 需要的自訂標籤格式
    `docker tag image_name:tag username/imagename:tagname`
  - 上傳容器映像到 Docker Hub 裡
    `docker push username/imagename:tagname`
  - 登出帳號
    `docker logout`

- 相關連結
  - Store images on Docker Hub - Docker

相關連結

- Docker Engine overview
- Overview of Docker Desktop
- Containerize a .NET application
- Containerize a Java application
- Building best practices | Docker Docs
- Configure Nginx Docker
- Docker — 從入門到實踐 (正體中文版)
- Docker — 從入門到實踐 (正體中文版) (新版)

聯絡資訊

The Will Will Web

網路世界的學習心得與技術分享

http://blog.miniasp.com/

Facebook

Will 保哥的技術交流中心

http://www.facebook.com/will.fans

Twitter

https://twitter.com/Will_Huang

多奇·教育訓練

THANK YOU!

Q&A