全面易懂的Docker指令大全

joshhu

Published
with GitBook

目錄

1. 介紹
2. 基礎系統及Docker Hub指令
    i. 和Linux系統相關的指令
    ii. 和Docker Registry相關的指令
3. 和映像檔相關的指令
    i. 映像檔名稱基礎
    ii. 映像檔指令說明
4. Container入門篇
    i. 先了解Container
        i. Container的生命週期
        ii. 指令基礎
    ii. 最重要的Docker Run
        i. 執行中的Container
        ii. 進入執行中的Container
        iii. 其它執行的操作
    iii. 針對Container本身的操作
    iv. 用VM的方式運行Container
5. Docker執行時環境參數

本章重點

docker的指令主要分為四大類，分別是：

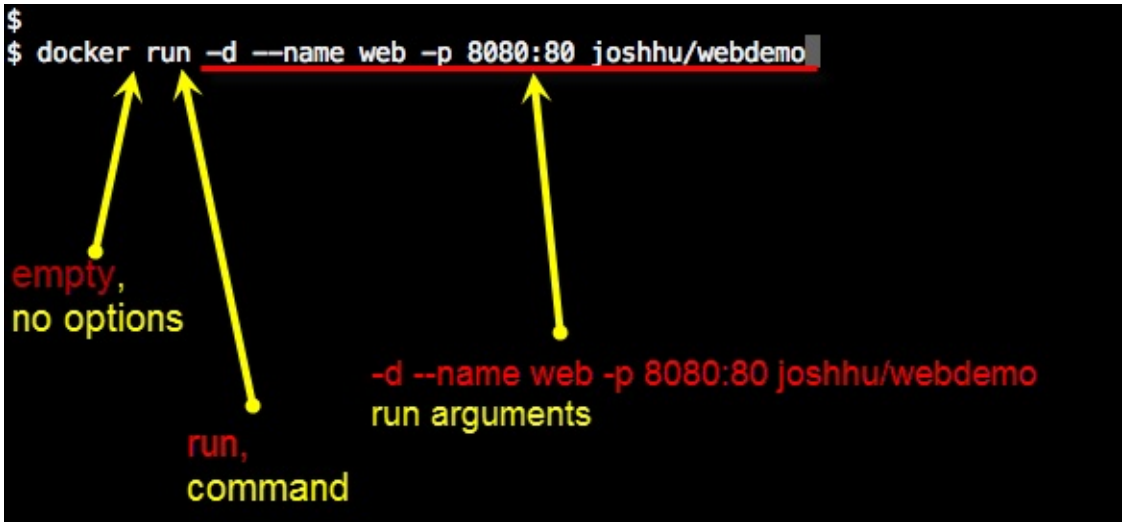
- Linux系統相關指令
- Docker Registry相關指令
- 映像檔相關指令
- Container相關指令

Docker最方便的地方就是所有的動作都由`docker`這個指令完成，你只要熟悉了指令的使用，docker的使用就完全沒問題了。

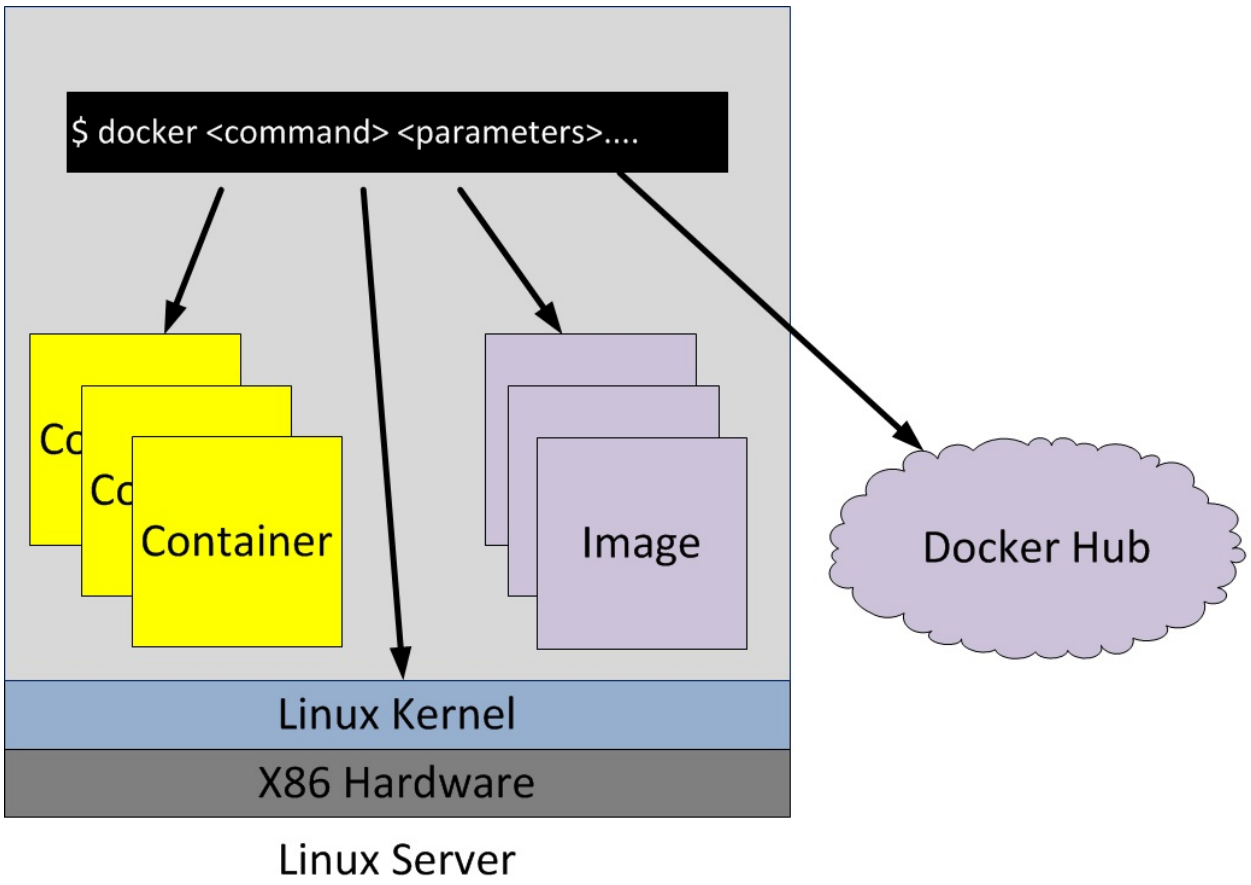
Docker的指令標準格式如下：

```
docker [選項] 指令 [參數1...] [參數2...] [參數n...]
```

- [選項]：設定Docker本身執行環境的選項，屬於進階部分，我們稍後討論。
- 指令：Docker的指令集，是本章的重點。
- [參數...]：伴隨指令集的參數，也是Docker的精華。



所有的指令都是圍繞這幾個元件上


$ docker <command> <parameters>....
Container
Image
Docker Hub
Linux Kernel
X86 Hardware
Linux Server

基礎指令

- 和**Linux**系統有關的指令：介紹Docker的資訊。事實上Docker的設定十分複雜，我們會在之後介紹更多有關Docker在Linux的環境設定。
- 和**Registry**有關的指令：這邊主要針對公開的官方Docker Hub的幾個指令查詢，後續有關自行建立私有的Docker Registry會有完整架設。

和系統有關的指令及參數

和系統相關的指令很少就2個，分別如下。

info

列出和系統相關的資訊，如映像檔數、Container數、檔案系統目錄、Linux核心版本，使用Linux版本、CPU及記憶體等。

用途：用來檢查系統是否能正確執行*Docker*所有指令。

```
root@ubuntu:/# docker info
Containers: 0
Images: 14
Storage Driver: aufs
 Root Dir: /var/lib/docker/aufs
 Backing Filesystem: extfs
 Dirs: 14
Execution Driver: native-0.2
Kernel Version: 3.16.0-31-generic
Operating System: Ubuntu 14.10
CPUs: 2
Total Memory: 3.847 GiB
Name: ubuntu
ID: SBV6:E3KX:GSE5:EOL6:YR3Q:6LRU:BD44:UEZK:5RVW:PHPS:IR6V:C7WI
root@ubuntu:/#
```

version

列出目前Docker的版本，以及Go語言的版本等。

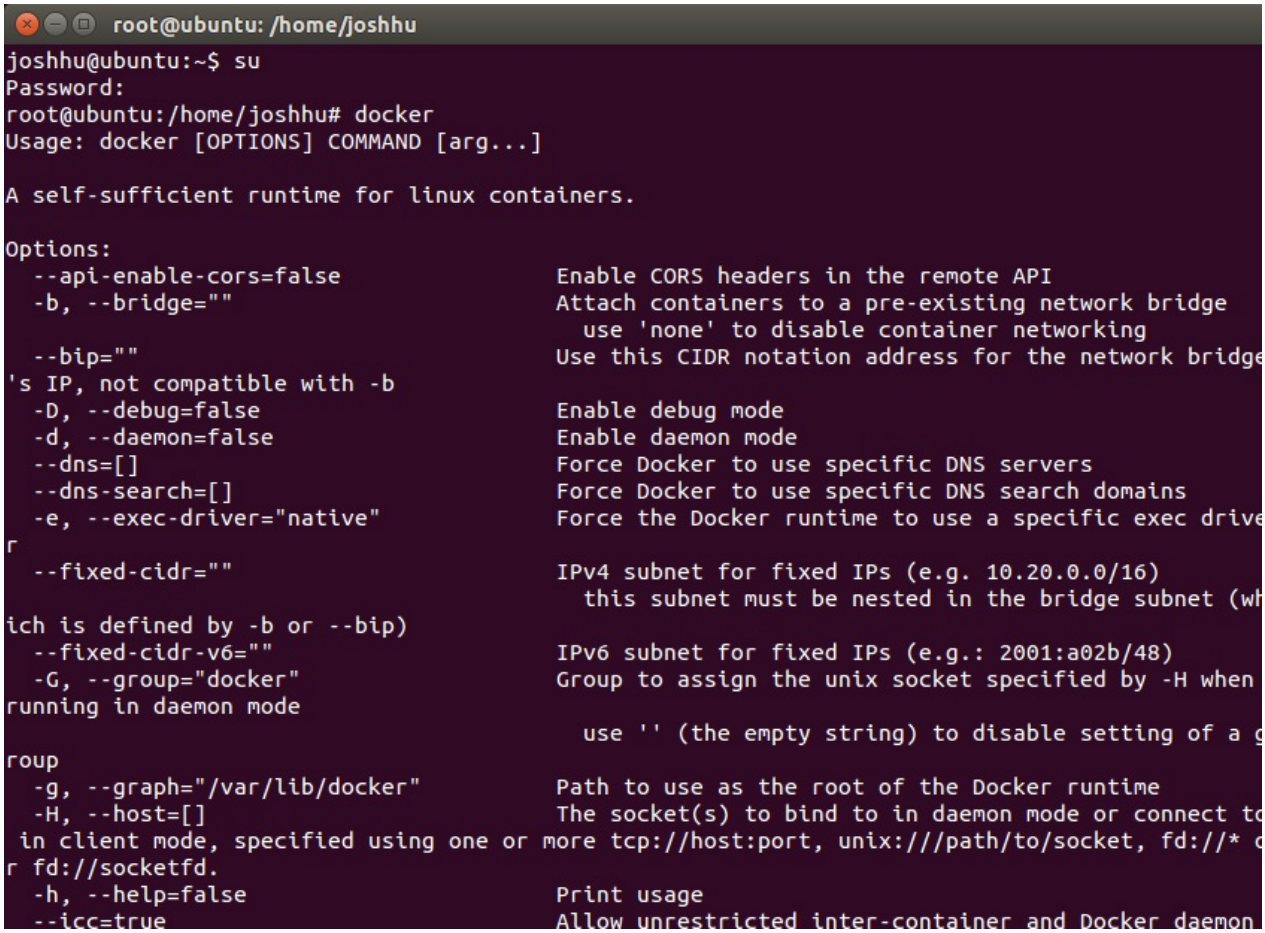
用途：常用來檢查是否要昇級到較高的*Docker*版本。

```
$ docker version
Client version: 1.5.0
Client API version: 1.17
Go version (client): go1.4.1
Git commit (client): a8a31ef
OS/Arch (client): linux/amd64
Server version: 1.5.0
Server API version: 1.17
Go version (server): go1.4.1
Git commit (server): a8a31ef
```

Docker的環境變數

一個執行Docker的Linux系統其中有許多有關Docker的環境變數設定，我們會在本系列的「進階**Docker**指令」部分說明。

*Docker*的環境變數十分強大



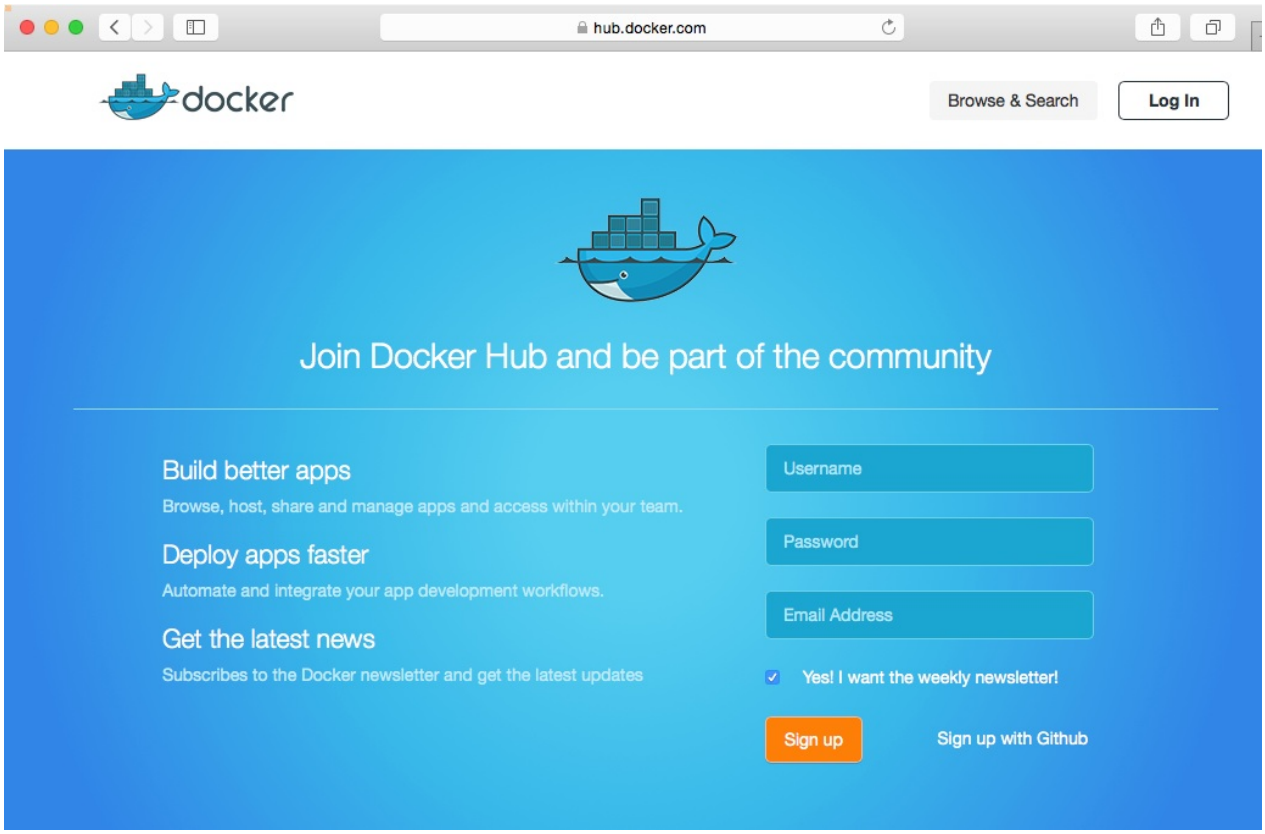
Docker Hub相關指令

和Docker Hub或映像檔資料庫相關的指令就3個，分別如下：

login/logout

Docker的元件中，最重要的就是官方的Docker Hub，充滿了全世界的大公司的、個人、玩家自製的Docker映像檔。你當然也可以貢獻自製的映像檔，只要到Docker的官方雲端映像檔資料庫註冊帳號即可。

前往註冊Docker Hub的帳號



註冊完之後，在本機可以使用帳號密碼登入，以便之後的映像檔上傳等。

```
$ docker login
Username: joshhu
Password:
Email: josh.hu@yahoo.com
Login Succeeded
```

如果你有自行架設私有的Docker映像檔資料庫，則輸入該資料庫的位址，如：

```
docker login 192.168.1.100:8000
```

一般如果在 `login/logout` 後方留空，則自動指向公有的docker hub，https://index.docker.io/v1/。

注意-正常使用不需要帳號密碼

一般使用Docker的映像檔時，是完全不需要帳號密碼的，只有要上傳映像檔才需要。

search

這個指令是在Docker Hub找中尋找映像檔用的。最好用的就是 `-s` 參數，用來找評等較高(星級)的映像檔。由於Docker Hub是人人可上傳的，因此映像檔的品質參差不齊。查看大家評等較高的映像檔，品質較有口碑。舉例來說， `nginx` 的映像檔一定有一大堆，我們可以找出星級在100以上的，輸入：

```
$ docker search nginx
NAME                              DESCRIPTION                                     STARS
```

```
nginx                             Official build of Nginx.                        706
jwilder/nginx-proxy               Automated Nginx reverse proxy for docker c...   166
maxexcloo/nginx-php               Docker framework container with Nginx and ...   30
richarvey/nginx-php-fpm           Container running Nginx + PHP-FPM capable ...   12
marvambass/nginx-registry-proxy   Docker Registry Reverse Proxy with Basic A...   12
million12/nginx-php               Nginx + PHP-FPM, CentOS-7 based.                6
maxexcloo/nginx                   Docker framework container with Nginx inst...   5
zenithar/nano-nginx               Nano NGiNX Container, compiled from scratch.    5
dylanlindgren/docker-nginx        Docker image for Nginx, to be used in comb...   5
h3nrik/nginx-ldap                 NGINX web server with LDAP/AD, SSL and pro...   4
million12/nginx                   Nginx: extensible, nicely tuned for better...   2
devries/nginx                     A standard ubuntu nginx installation with ...   2
abevoelker/nginx                  nginx                                           1
klaemo/nginx                      nginx 1.7.0 (mainline)                          1
radial/nginx                      Spoke container for Nginx, a high performa...   1
rnbwd/nginx                       fork of jwilder/nginx-proxy with spdy           1
dperson/nginx                                                                     1
jakexks/tiny-nginx                Tiny (under 7MB!) nginx 1.6.2 container, b...   1
dock0/nginx                       Arch container running nginx                    0
densuke/nginx-php5                NginxとPHP5を使えるように調整し...
aegypius/nginx                    confd managed nginx reverse proxy               0
abcum/nginx                       Docker container for nginx                      0
```

會有一大堆，但如果輸入：

```
$
$ docker search -s 10 nginx
NAME                              DESCRIPTION                                     STARS
nginx                             Official build of Nginx.                        706
jwilder/nginx-proxy               Automated Nginx reverse proxy for docker c...   166
maxexcloo/nginx-php               Docker framework container with Nginx and ...   30
richarvey/nginx-php-fpm           Container running Nginx + PHP-FPM capable ...   12
marvambass/nginx-registry-proxy   Docker Registry Reverse Proxy with Basic A...   12
$
```

這樣的搜索結果精準多了，我們也會發現，通常由軟體發行公司提供的官方映像檔品質一定最好。

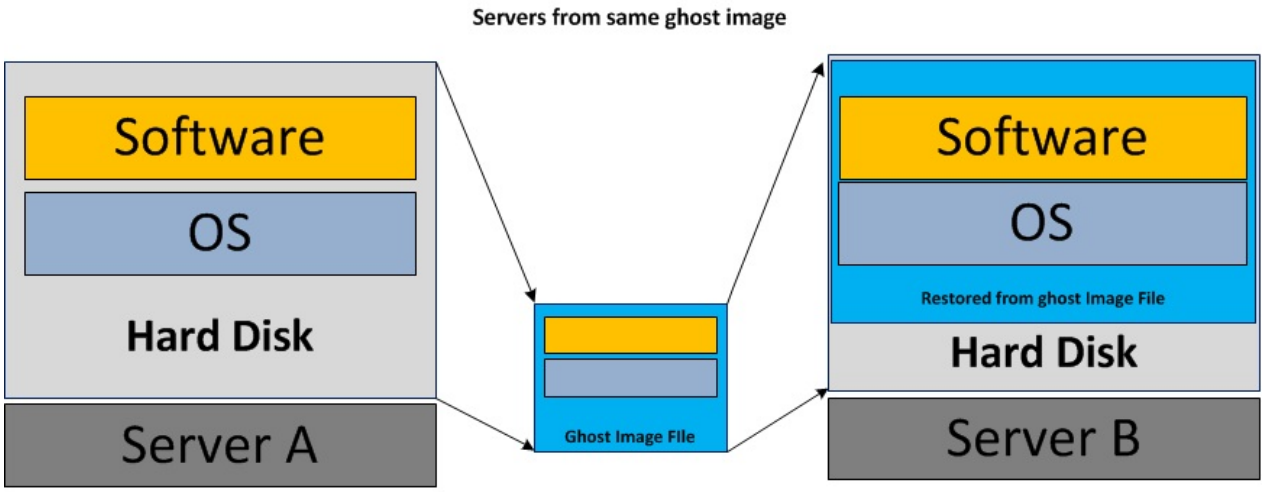
本章重點

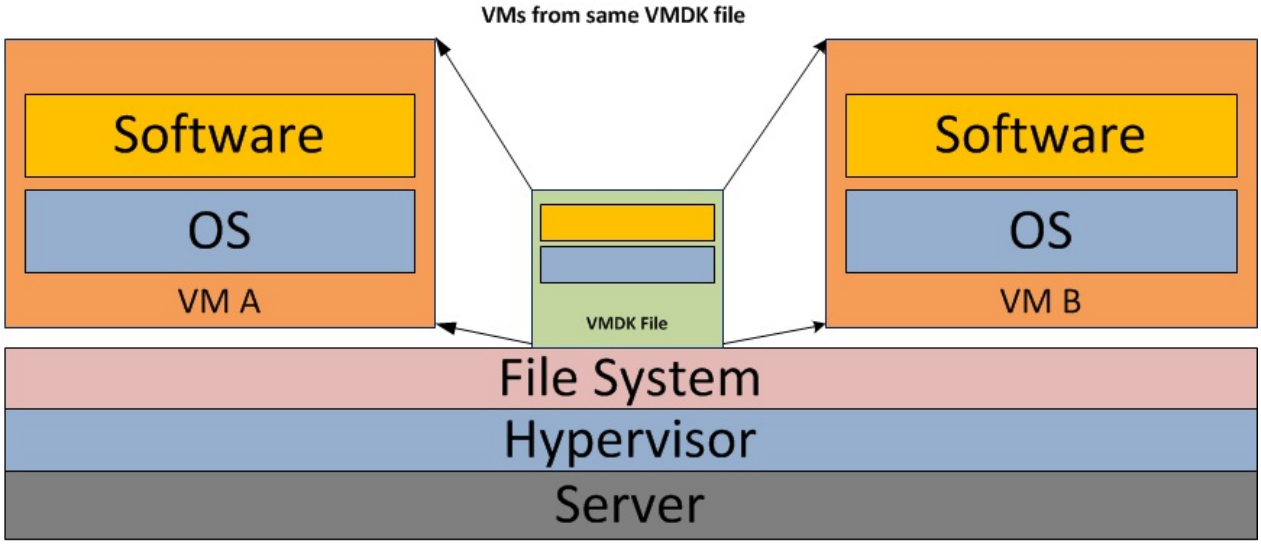
Docker映像檔相關指令

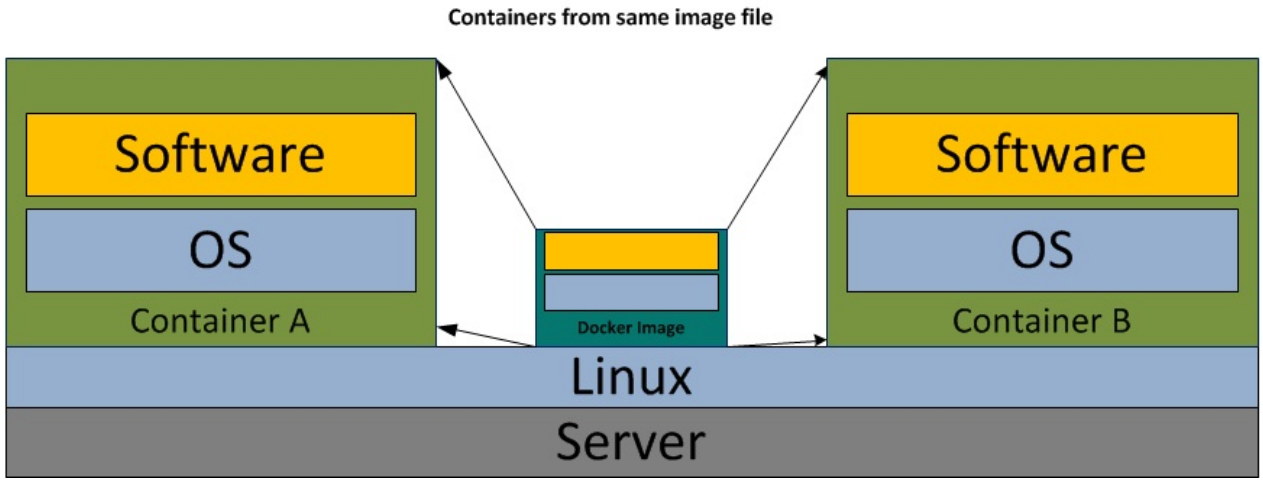
映像檔是用來填入Container的內容，而Container有了內容之後才「活了起來」。如果比較成傳統伺服器器，映像檔就像是安裝了了作業系統和應用程式的硬碟，然後做成 `Ghost` 的檔案。這個 `Ghost` 檔案可以還原到任何電腦上，新的電腦再開機提供服務。

比較於VM，映像檔就像是安裝好作業系統和應用程式的虛擬硬碟檔。任何VM都可以掛載這個虛擬硬碟檔，新建立的VM再開機提供服務。

由此可知，Container的映像檔當然也是安裝好作業系統和應用程式的「某種類型」的檔案，然後任何Container都可以把這個映像檔載入，Container開機提供服務。Docker中有許多指令用來處理映檔案。


Servers from same ghost image
Software
OS
Hard Disk
Server A
Ghost Image File
Software
OS
Restored from ghost Image File
Hard Disk
Server B


VMs from same VMDK file
Software
OS
VM A
VMDK File
Software
OS
VM B
File System
Hypervisor
Server


Containers from same image file
Software
OS
Container A
Docker Image
Software
OS
Container B
Linux
Server

映像檔基礎

映像檔是Docker的靈魂，任何Container都必須從一個映像檔來建立。標準的Docker映像檔包括了映像檔名稱，以及唯一的映像檔**id**。我們在這一小節先來看映像檔的基礎。

存放**Docker**映像檔的地方

存放Docker映像檔的地方稱之為一個Docker Registry。每個Docker Registry都有一個網址。當然每個Registry不一定只有一台伺服器，其存放方式屬於系統底層，這我們管不著。一般Docker Registry的來源有：

- 官方的**Docker Hub**：其上有數十萬個Docker映像檔，也是以github的概念建立而成的。我們在使用Docker時，預設的下載來源就是這裏。
- 非官方的公開**Docker Registry**：如果你要從這些地方下載，就必須在下載時指定完整的位址名稱，也可能需要該網站的認證金鑰，或是在Docker執行的設定檔中先設定好，稍後再談。
- 自建私有的**Docker Registry**：公司內部可以建立私有的Registry以保證不會用到來源不明的映像檔。我們在本書實作的地方會有完整架設步驟。

Docker Hub的個人映像檔名稱

Docker是公開的映像檔集散中心，上面有不同的使用者，建立不同的倉庫，其中放了不同的映像檔。上面粗體的字，即映像檔名稱的組成。標準的Docker Hub的個人映像檔名稱格式為：

```
<user name>/<repo name>:<tag name>
```

如

```
joshhu/webdemo:ubuntu14
```

說明如下：

- `user name`：使用者名稱。在Docker Hub上每個使用者都有一個獨立的名稱，這是Docker Hub上的最大單位。
- `repo name`：倉庫名稱。在Docker Hub上的每一個使用者，都可以建立自己的倉庫，倉庫中可以放多個映像檔。
- `tag name`：要分辨同一個倉庫中的不同映像檔，就要用`tag name`來區分。
  - 如果該倉庫中只有一個映像檔，則`tag name`可以省略。
  - 如果該倉庫中有多個映像檔，在沒有指定`tag name`時，以最新的一個為主。
  - 同一個映像檔可以有多個`tag name`，可看做是別名。可以從相同的映像檔ID看出來。


映像檔 ID用來分別映像檔，注意`ubuntu:14.10`和`ubuntu:utopic`、`ubuntu:utopic-20150319`三個映像檔 ID一樣，是同一個映像檔。

Docker Hub上大公司的官方映像檔

主要的Linux發佈商、平台供應商、資料庫供應商，服務供應商等，他們在Docker Hub上也有自己的官方映像檔。這些公司的映像檔格式和一般私人的不同，他們應該是較大牌，因此只有`repo name`和`tag name`，而沒有`user name`。格式如下：

```
apache:latest
ubuntu:trusty
```

大公司的官方映像檔`tag name`通常用來標記來自同一個repo的不同映像檔。例如 ubuntu 倉庫中有多個映像檔，通過 TAG 來區分發行版本，例如 12.10、13.04、14.04、14.10 等。但最常用的還是用官方名稱，如`trusy`、`precise`、`utopic`來作為TAG。

如果不指定`tag name`，直接就下載最新的版本，有些廠商的映像檔甚至連`tag name`都沒有。

此外，這種官方版本映像檔其使用方式、限制及tag的別名也和一般的映像檔完全一樣。

映像檔指令參數總類

Docker的映像檔指令是第二多的，僅次於對於容器處理的指令，下面簡單分類。

映像檔指令有`images, pull, load, save, push, rmi, tag, pull, build`幾個。我們就來看看。

列出本機映像檔`docker images`

如果你使用任何映像檔建立過Container，或是手動下載過映像檔，就會存放在本機中。此時可以列出所有的映像檔：

```
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             VIRTUAL S
joshhu/webdemo      latest              a3574f323972        3 days ago          243.2 MB
ubuntu              latest              d0955f21bf24        5 days ago          188.3 MB
nginx               latest              3f72b0ae3e59        5 days ago          93.43 MB
tutum/apache-php    latest              df6b22a47766        6 days ago          244.4 MB
```

可以看到列出的REPOSITORY就是`user name/repo name`，TAG就是`tag name`。當然還有其IMAGE ID以及下載時間和大小。`docker images`有幾個常用的參數，分別是：

- `-a`：列出完整的映像檔層次資訊。每個映像檔是由不同層次組成的，我們會在稍後說明。
- `-q`：只列出映像檔ID。這在做映像檔批次處理時很方便。
- `-tree`：官方文件已經沒有這個參數，但還是可以用。列出映像檔不同層次之間的樹狀關係。

```
$ docker images -a
REPOSITORY          TAG                 IMAGE ID            CREATED             VIRTUAL S
<none>              <none>              d11f03cd837d        3 days ago          243.2 MB
<none>              <none>              8abfc77aac61        3 days ago          243.2 MB
joshhu/webdemo      latest              a3574f323972        3 days ago          243.2 MB
<none>              <none>              d1dfd80f5ed9        3 days ago          243.2 MB
<none>              <none>              2d4173730925        3 days ago          243.2 MB
<none>              <none>              e8f612e3238f        3 days ago          243.2 MB
<none>              <none>              8be3dadcb43a        3 days ago          243.2 MB
<none>              <none>              45ad00454734        3 days ago          243.2 MB
<none>              <none>              1214be61bcaa        3 days ago          188.3 MB
ubuntu              latest              d0955f21bf24        5 days ago          188.3 MB
<none>              <none>              9fec74352904        5 days ago          188.3 MB
<none>              <none>              a1a958a24818        5 days ago          188.3 MB
<none>              <none>              f3c84ac3a053        5 days ago          188.1 MB
<none>              <none>              511136ea3c5a        21 months ago       0 B
```

```
$ docker images -q
a3574f323972
d0955f21bf24
```

```
$ docker images -tree
Warning: '-tree' is deprecated, it will be removed soon. See usage.
└─511136ea3c5a Virtual Size: 0 B
  └─f3c84ac3a053 Virtual Size: 188.1 MB
    └─a1a958a24818 Virtual Size: 188.3 MB
      └─9fec74352904 Virtual Size: 188.3 MB
        └─d0955f21bf24 Virtual Size: 188.3 MB Tags: ubuntu:latest
          └─1214be61bcaa Virtual Size: 188.3 MB
            └─45ad00454734 Virtual Size: 243.2 MB
              └─8be3dadcb43a Virtual Size: 243.2 MB
                └─2d4173730925 Virtual Size: 243.2 MB
                  └─e8f612e3238f Virtual Size: 243.2 MB
                    └─d11f03cd837d Virtual Size: 243.2 MB
                      └─8abfc77aac61 Virtual Size: 243.2 MB
                        └─d1dfd80f5ed9 Virtual Size: 243.2 MB
                          └─a3574f323972 Virtual Size: 243.2 MB Tags: joshhu/webdemo:late
```

下載映像檔 `docker pull`

從Docker Hub下載映像檔，沒有加任何Registry的位址時，就預設從官方的Registry下載（`registry.hub.docker.com`）。如：

```
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             VIRTUAL S
joshhu/webdemo      latest              a3574f323972        3 days ago          243.2 MB
$ docker pull ubuntu:latest
511136ea3c5a: Already exists
f3c84ac3a053: Already exists
a1a958a24818: Already exists
9fec74352904: Already exists
d0955f21bf24: Already exists
ubuntu:latest: The image you are pulling has been verified. Important: image verification

Status: Image is up to date for ubuntu:latest
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             VIRTUAL S
joshhu/webdemo      latest              a3574f323972        3 days ago          243.2 MB
ubuntu              latest              d0955f21bf24        5 days ago          188.3 MB
```

要將某一個倉庫的所有映像檔都下載回來，可使用`-a`參數。但這樣要小心，因為有可能會太大，下載需要很長時間。

```
$ docker pull -a joshhu/webdemo
c34b4129cfd8: Pulling image (u12) from joshhu/webdemo, endpoint: https://registry-1.docke
c34b4129cfd8: Download complete
a3574f323972: Download complete
511136ea3c5a: Download complete
f3c84ac3a053: Download complete
a1a958a24818: Download complete
9fec74352904: Download complete
.....
```

```
9654debb69ae: Download complete
Status: Downloaded newer image for joshhu/webdemo
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             VIRTUAL S
joshhu/webdemo      u12                 c34b4129cfd8        35 minutes ago      288 MB
joshhu/webdemo      latest              a3574f323972        3 days ago          243.2 MB
$
```

將映像檔存入/匯出電腦檔案格式 `docker save/load`

Docker的映像檔雖然名為檔案，但其格式十分複雜(本書後面章節會有說明)。你也不知道他存在哪，也不知道什麼格式，如果想要和其它人交換時，不想上傳到Docker Hub，不想自己架設私有Docker Registry，就可以用這兩個參數存成 `tarball` 格式及匯出。存入時別忘了加 `-o` 參數，要不然只會在顯示不會真的壓縮。

```
$ docker save -o webdemou12.tar joshhu/webdemo:u12
$ ls -al
total 294020
drwxrwxr-x  4 joshhu joshhu      4096 Mar 26 00:37 .
drwxr-xr-x 27 joshhu joshhu      4096 Mar 25 20:03 ..
drwxr-xr-x  3 joshhu joshhu      4096 Mar 22 08:25 test
drwxr-xr-x  3 joshhu joshhu      4096 Mar 25 23:52 u12
-rw-r--r--  1 root   root   301058560 Mar 26 00:37 webdemou12.tar
$
```

將 `tarball` 還原回映像檔格式，則用 `load` 。可輸入

```
$ docker load --input webdemou12.tar
```

下面是範例：

```
$ docker images -a
REPOSITORY          TAG                 IMAGE ID            CREATED             VIRTUAL S
$ ls -al webdemou12.tar
-rw-r--r-- 1 chtti chtti 301058560 Mar 26 00:41 webdemou12.tar
$ docker load --input webdemou12.tar
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             VIRTUAL S
joshhu/webdemo      u12                 c34b4129cfd8        47 minutes ago      288 MB
```

刪除映像檔 `docker rmi`

這個指令刪除本機中存放的映像檔。但如果有容器還在使用這個映像檔，則無法刪除。如果硬要刪除，可以下 `-f` 參數強迫刪除。

前面提到映像檔是以層次的方式來存放，因此一個映像檔會有多個層次。你可以下 `--no-prune=true` 這個

參數，只殺掉有 `tag name` 的映像檔。

以一個標準的映像檔來說，只會殺掉最上面一層，因為建立時的其它中間層次並不會有 `tag name`，這樣做的好處是可以留下許多映像檔共用的母層次。**這也是Docker映像檔指令中，唯一能處理到層次這個等級的參數了**

```
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             VIRTUAL S
joshhu/webdemo      u12                 c34b4129cfd8        2 hours ago         288 MB
```

```
$ docker images -a
REPOSITORY          TAG                 IMAGE ID            CREATED             VIRTUAL S
joshhu/webdemo      u12                 c34b4129cfd8        2 hours ago         288 MB
<none>              <none>              878616cacd7b        2 hours ago         288 MB
<none>              <none>              95325bba7e4e        2 hours ago         288 MB
<none>              <none>              f12848fa258d        2 hours ago         288 MB
<none>              <none>              9654debb69ae        2 hours ago         288 MB
<none>              <none>              dec7d1913547        2 hours ago         288 MB
<none>              <none>              05958e5f3b74        2 hours ago         288 MB
<none>              <none>              fe798fd9c738        2 hours ago         288 MB
<none>              <none>              afc5c5484606        2 hours ago         131.9 MB
<none>              <none>              9c5e4be642b7        5 days ago          131.9 MB
<none>              <none>              72dffce15bf2        5 days ago          131.9 MB
<none>              <none>              c41f0fc9131d        5 days ago          131.9 MB
<none>              <none>              ea806576238a        5 days ago          131.7 MB
<none>              <none>              511136ea3c5a        21 months ago       0 B
$ docker rmi --no-prune=true joshhu/webdemo:u12
Untagged: joshhu/webdemo:u12
Deleted: c34b4129cfd8bbf8a9b6a820cc7ae5e96dc10c58dda5916d8849d64b9c43db7b
$ docker images -a
REPOSITORY          TAG                 IMAGE ID            CREATED             VIRTUAL S
<none>              <none>              878616cacd7b        2 hours ago         288 MB
<none>              <none>              9654debb69ae        2 hours ago         288 MB
<none>              <none>              f12848fa258d        2 hours ago         288 MB
<none>              <none>              95325bba7e4e        2 hours ago         288 MB
<none>              <none>              05958e5f3b74        2 hours ago         288 MB
<none>              <none>              dec7d1913547        2 hours ago         288 MB
<none>              <none>              fe798fd9c738        2 hours ago         288 MB
<none>              <none>              afc5c5484606        2 hours ago         131.9 MB
<none>              <none>              9c5e4be642b7        5 days ago          131.9 MB
<none>              <none>              72dffce15bf2        5 days ago          131.9 MB
<none>              <none>              c41f0fc9131d        5 days ago          131.9 MB
<none>              <none>              ea806576238a        5 days ago          131.7 MB
<none>              <none>              511136ea3c5a        21 months ago       0 B
```

一次刪掉所有映像檔

可以配合Linux的批次指令來一次清乾淨所有的映像檔，輸入

```
docker rmi -f $(docker images -aq)
```

執行如下：

```
docker rmi -f $(docker images -aq)
Deleted: 9654debb69ae7a8ef27571e088e8779c89523c3654dd4d5f5ffa7862f097d012
Deleted: 95325bba7e4e26487d71b6e95d1574a235ebe993dd0708eade700f7f78b74e58
Deleted: f12848fa258de0af708f30eb135ad3a4ffcfb4391e219d266014116aa6fbfd27
Deleted: 878616cacd7b522581e5a41209293bef644381625405b19d6882c070314e1b49
Deleted: 05958e5f3b74cf5aaa788b79711e75f71b3e9b01a8da80a0fcbe294cb2c34d3c
Deleted: dec7d19135473395d025767bb00d0b2a8fdbff81426a1cf64b528e48f4b5b738
Deleted: fe798fd9c738d610261e5a87e9f8503fb04fe2c8c8c928399c15fe12d52d4aec
Deleted: afc5c5484606ebde6fd96908c958020d57bfdfc4e3dcc81d1d9551572916cbcf
Deleted: 9c5e4be642b799060baa07b826968128ff66d2ced86f6098f6d9f5845f2d35dd
Deleted: 72dffce15bf2df5afebfbe91515463a1629dc6a9233ede1346e80cd97da606ac
Deleted: c41f0fc9131d093dea7decaf641db6b455336a592409d5209832c6eebb749405
Deleted: ea806576238a90ec7ea482876f39d63427cbe1cf980a54488fbad1fbae8f9eef
Deleted: 511136ea3c5a64f264b78b5433614aec563103b4d4702f3ba7d4d2698e22c158
FATA[0000] Error: failed to remove one or more images
$ docker images -a
REPOSITORY          TAG                 IMAGE ID            CREATED             VIRTUAL S
```

替本機映像檔加標籤名稱 `docker tag`

前面提過，一個映像檔可以有很多不同的 `tag name` 。為了方便，我們常常會給同一個映像檔不同的 `tag name` 。

舉例來說，筆者在本機有一個 `joshhu/webdemo` 倉庫，其中有兩個映像檔，一個是Ubuntu 12+Apache/php，一個是Ubuntu14+Apache/php。由於不同Ubuntu版本安裝的Apache/php版本也不同，因此都放在webdemo這個repo下，必須使用 `tag name` 來區分。原來的Ubuntu12已經有一個 `tag name` 叫 `u12` 了，因此我要將另一個沒有 `tag name` 的加上標籤名稱(在建立時，系統自動將這個映像檔的 `tag name` 指定成 `latest`)。 原來沒的joshhu/webdemo的TAG是 `latest`

```
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             VIRTUAL S
joshhu/webdemo      u12                 c34b4129cfd8        12 hours ago        288 MB
joshhu/webdemo      latest              a3574f323972        3 days ago          243.2 MB
ubuntu              12.04               9c5e4be642b7        5 days ago          131.9 MB
ubuntu              12.04.5             9c5e4be642b7        5 days ago          131.9 MB
ubuntu              precise             9c5e4be642b7        5 days ago          131.9 MB
ubuntu
```

加上TAG之後

```
$ docker tag joshhu/webdemo:latest joshhu/webdemo:u14
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             VIRTUAL S
joshhu/webdemo      u12                 c34b4129cfd8        12 hours ago        288 MB
joshhu/webdemo      latest              a3574f323972        3 days ago          243.2 MB
joshhu/webdemo      u14                 a3574f323972        3 days ago          243.2 MB
ubuntu              12.04               9c5e4be642b7        5 days ago          131.9 MB
```

```
ubuntu              12.04.5             9c5e4be642b7        5 days ago          131.9 MB
ubuntu              precise             9c5e4be642b7        5 days ago          131.9 MB
ubuntu              precise-20150320    9c5e4be642b7        5 days ago          131.9 MB
```

可以看出其中`latest`和`u14`兩個TAG是指到同一個映像檔ID。

從上圖也可以看到，來自官方的映像檔`tag name`通常用來標記來自同一個repo的不同映像檔。例如ubuntu 倉庫中有多個映像檔，用 TAG 來區分發行版本，如圖中，`9c5e4be642b7`映像檔擁有四個TAG，是同一份映像檔。

自建映像檔`docker build/hisotry`

`docker build`指令可以從現成的映像檔為基礎，自行建立全新的映像檔，而`docker history`則會列出製作的每一步過程。我們會在自建映像檔章節中討論完整實作。

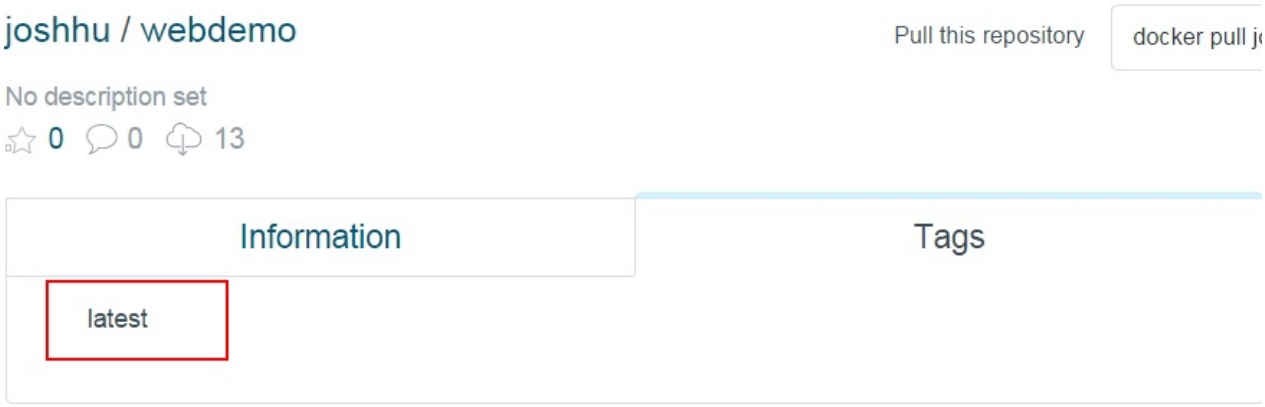
上傳映像檔`docker push`

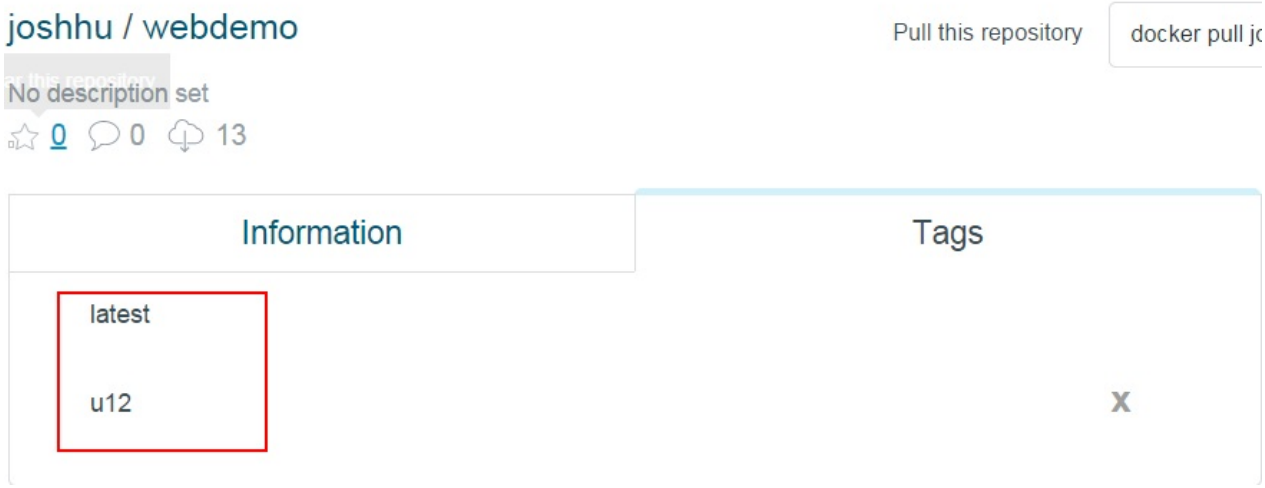
如果你自建立映像檔，可以上傳到官方/私有/非官方公開的Docker Registry上。就使用push這個指令。使用方法很簡單，如果你已經使用`docker login`，那就可以直接用標準的映像檔名稱上傳，如果你還沒有`login`，會提示你輸入帳號、密碼及電子郵件。

`docker push <username>/<repo name>:<Tag name>`

```
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             VIRTUAL S
joshhu/webdemo      u12                 c34b4129cfd8        12 hours ago        288 MB
joshhu/webdemo      latest              a3574f323972        3 days ago          243.2 MB
joshhu/webdemo      u14                 a3574f323972        3 days ago          243.2 MB
ubuntu              12.04               9c5e4be642b7        5 days ago          131.9 MB
ubuntu              12.04.5             9c5e4be642b7        5 days ago          131.9 MB
ubuntu              precise             9c5e4be642b7        5 days ago          131.9 MB
ubuntu              precise-20150320    9c5e4be642b7        5 days ago          131.9 MB
$
$ docker push joshhu/webdemo:u12
The push refers to a repository [joshhu/webdemo] (len: 1)
Sending image list
Pushing repository joshhu/webdemo (1 tags)
511136ea3c5a: Image already pushed, skipping
ea806576238a: Image already pushed, skipping
c41f0fc9131d: Image already pushed, skipping
72dffce15bf2: Image already pushed, skipping
9c5e4be642b7: Image already pushed, skipping
afc5c5484606: Image already pushed, skipping
fe798fd9c738: Image already pushed, skipping
dec7d1913547: Image already pushed, skipping
05958e5f3b74: Image already pushed, skipping
878616cacd7b: Image already pushed, skipping
f12848fa258d: Image already pushed, skipping
95325bba7e4e: Image already pushed, skipping
9654debb69ae: Image already pushed, skipping
c34b4129cfd8: Image already pushed, skipping
```

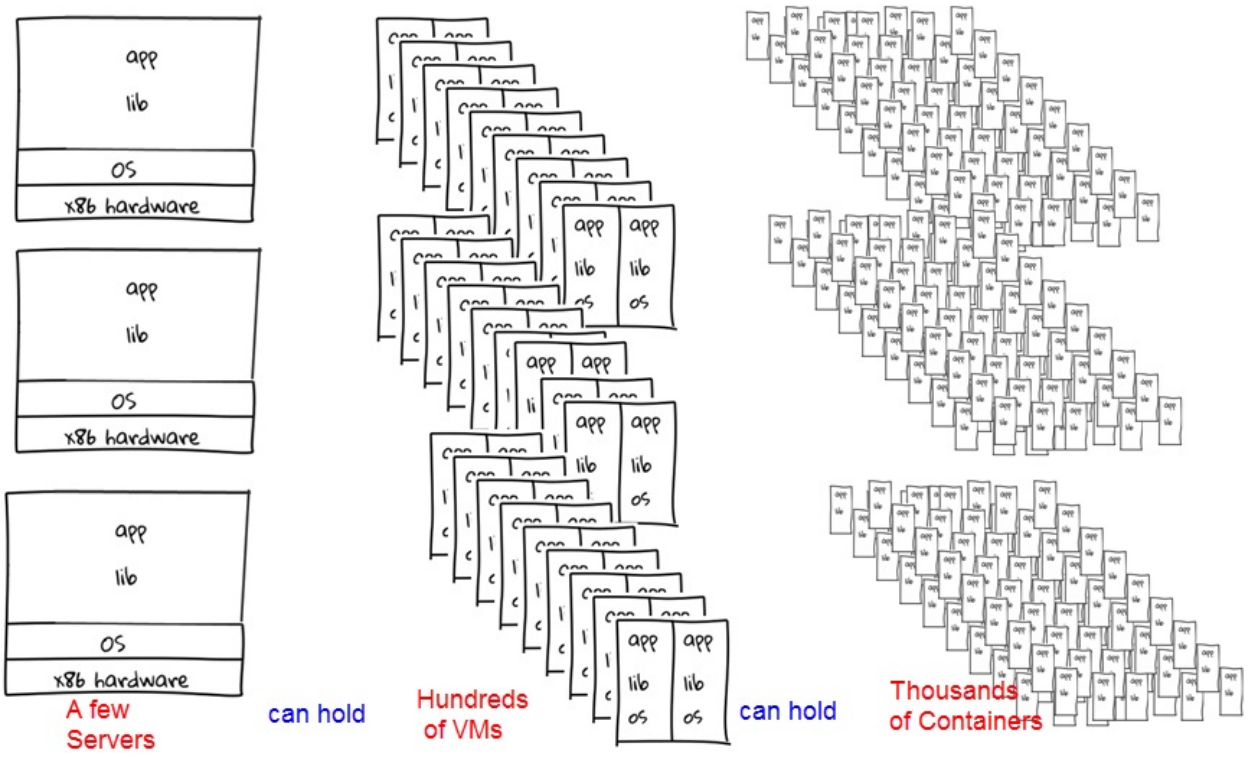
```
Pushing tag for rev [c34b4129cfd8] on {https://cdn-registry-1.docker.io/v1/repositories/j
$
```





本章重點

前面說了這麼多廢話，終於到了Docker最重要的部分 - Container了。Container是一個極輕量的「類」VM，而Docker又是更輕量的Container，建議讀者到本書前面的章節來看看Container和VM之間的比較，更要看看同為Container的Docker以及LXC之間的比較。



Container概念

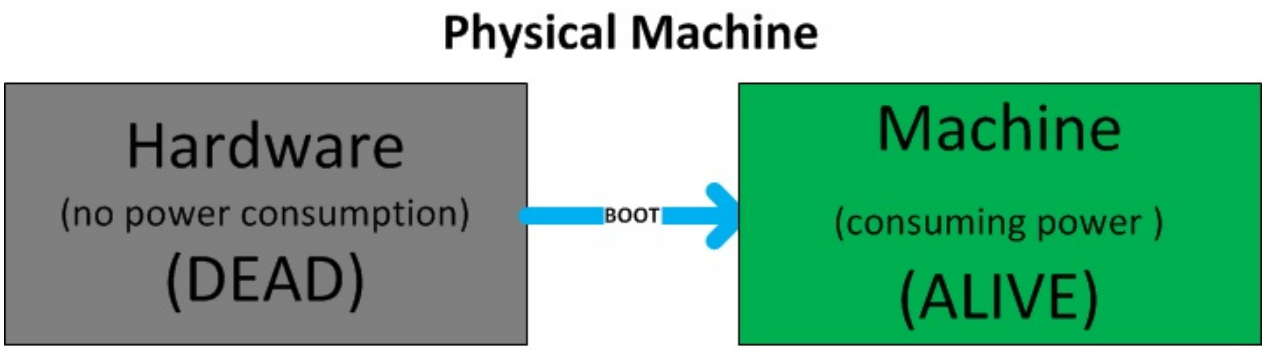
Container是Docker的靈魂

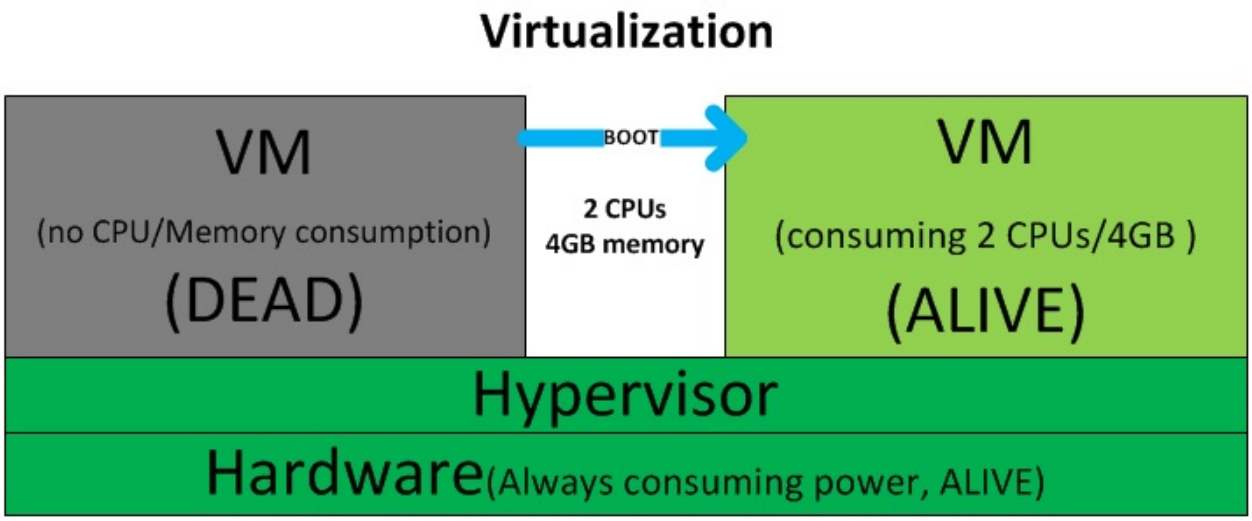
Container的生命週期

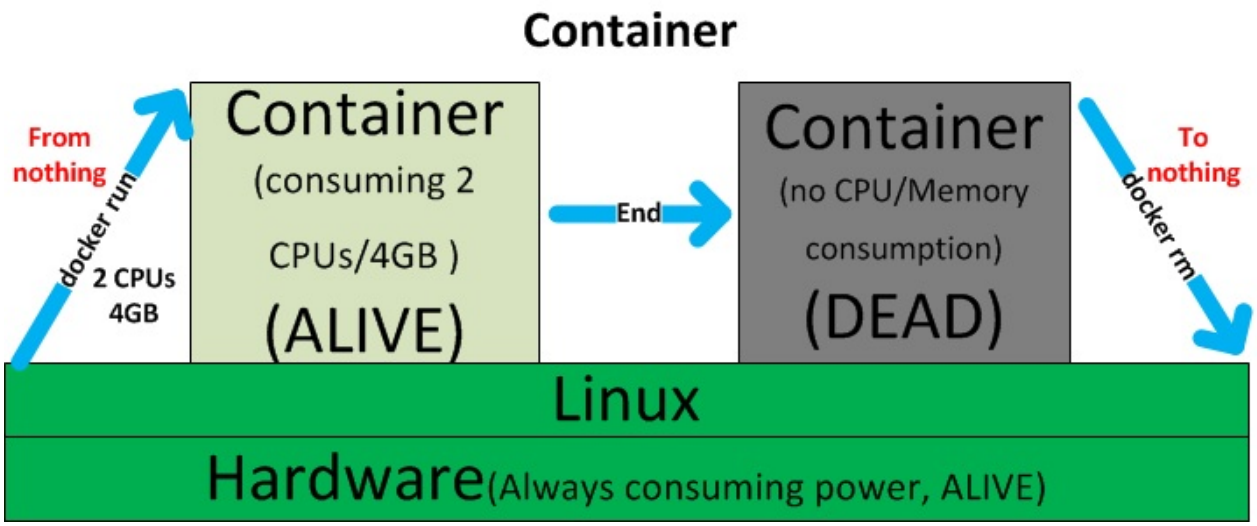
針對一個服務或軟體的宿主而言，伺服器、VM和Container的分別如下：

- 對伺服器來說，活著，就是電源打開，記憶體載入作業系統或軟體，並且開始消耗CPU，記憶體等系統資源的的電腦，沒有開機的電腦就是死的電腦。
- 對虛擬化平台來說，活著的VM，就是一個被Hypervisor分配到資源(CPU，記憶體)，並且開始真正消耗實體伺服器資源的VM，，因為VM本身並沒有所謂真正的開機關機，一個沒有消耗實體機器資源的VM，就是死的VM。
- 對Container來說，沒有執行`docker run/create`之前，**Container**根本不存在，更沒有所謂活著或死掉的問題。執行`docker run/create`之後，Container才會誕生，開始消耗系統資源(由Linux核心提供)，就是活著。執行完畢的Container，就是死的Container，但是並沒有消失，還是存在的。要執行`docker rm <container>`指令後，Container才會消失。

有關Container的存在，死活問題，我們會在本章稍後說明。


Physical Machine
Hardware
(no power consumption)
(DEAD)
BOOT
Machine
(consuming power)
(ALIVE)


Virtualization
VM
(no CPU/Memory consumption)
(DEAD)
BOOT
2 CPUs
4GB memory
VM
(consuming 2 CPUs/4GB)
(ALIVE)
Hypervisor
Hardware(Always consuming power, ALIVE)


Container
From
nothing
docker run
2 CPUs
4GB
Container
(consuming 2
CPUs/4GB)
(ALIVE)
End
Container
(no CPU/Memory
consumption)
(DEAD)
To
nothing
docker rm
Linux
Hardware(Always consuming power, ALIVE)

Container指令基礎

Docker有關Container的指令是最多樣的，而光一個 `docker run` 就有非常非多的參數。我們在這邊先將最基本的Container指令列出，讓讀者們能先能感受一下Docker的強大。

這邊列出Docker有關Container指令的分類方便讀者查詢，粗體表示常用指令，會有講解說明。

Container和映像檔之間的操作

- commit：將Container的改變存入映像檔
- export：將Container存成快照
- import：從Container快照恢復成映像檔

Container執行時的操作

- **create**：建立**Container**並執行指令
- **run**：同**create**
- kill：刪除執行中的Container，但**Container**還是存在，只是死了。
- **rm**：刪除**Container(**停止或運行中都行**)**，Container就從這世上消失了
- pause：暫停執行中的Container，仍暫有記憶體停，服務不中斷
- unpause：恢復暫停中的Container
- stop：停止執行中的Container，但不暫有記憶體，服務中斷
- start：啟動停止中的Container
- restart：重新啟動Container
- wait：讓Container暫停直到Container停止為止
- rename：更名Container

Container的狀態

- **inspect**：檢查**Container**的狀態**(**非常常用**)**
- **stats**：查看**Container**的**CPU**、記憶體及網路使用
- **port**：查看**Container**的通訊埠使用
- **ps**：查看**Container**使用狀態
- **top**：查看**Container**在主系統中的記憶體使用
- **dip**：查看**Container**的**IP**
- **dpid**：查看**Container**的**pid**

Container執行時的操作

- **attach**：連接**Container**的標準輸出輸入端
- **exec**：在外部向**Container**內執行指令
- **denter**：進入**Container**
- logs：將Container內的輸出顯示到螢幕上

Container和主系統之間的操作

- cp：複製Container內的檔案到主系統
- diff：列出兩個Container之間檔案系統差異
- events：列出某個時間點之前或之後的事件

和**Linux**的指令配合使用

Docker的執行常常和Linux的內部指令配合使用，如`grep`、`awk`、`xargs`等，會在需要的時候提及。下面的例子就是列出所有Container的狀態(`docker stats`預設只會列出指定Container的狀態)。

```
docker ps | awk 'NR>1 {print $NF}'| xargs docker stats
```

```
$ docker stats $(docker ps | awk 'NR>1 {print $NF}')
CONTAINER           CPU %               MEM USAGE/LIMIT     MEM %               NET I/O
web12               0.00%               8.742 MiB/128 MiB   6.83%               131.2 KiB
web14               0.01%               38.43 MiB/128 MiB   30.02%              181.6 KiB
```

快速建立你的第一個Docker服務

最簡單的Docker Container程式

最常執行的Docker指令就是`run`，我們就拿前幾章提到多次的範例來說明：

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

就是在標準輸出`stdout`列出`hello world`。

Docker建立的網頁服務

接下來使用一個沒有執行指令，而是單純啟動Container。其實這個Container並不是沒有執行程式，而是預先設定好的啟動程式已經設定在這個Container的來源映像檔`joshhu/webdemo`中了。

```
$ docker run -d --name web -p 8080:80 joshhu/webdemo
```

這個指令可以建立一個標準的`apache/php`的網頁服務，指令及參數說明如下：

- `run`：標準的Docker建立Container並執行指令
- `-d`：`run`指令的無數值參數，背景執行。
- `--name web`：`run`指令的文字參數，指定這個Container的名字為`web`。
- `-p 8080:80`：`run`指令的數值參數，把主機的`8080`通訊埠所有流量轉發到`web`這個Container的`80`通訊埠。
- `joshhu/webdemo`：`run`指令的文字參數，使用`joshhu/webdemo`來填入`web`這個Container。 Docker用指令加參數就可以完成99%的動作，非常方便。

建立並啟動Container：`docker run/create`

讀者一定很希望快速建立一個Docker服務。我們就來看看你的第一個Docker應用程式，官方使用Hello world!，我們也不例外：`docker run busybox echo "hello world"`

```
$ docker run busybox echo "hello world"
Unable to find image 'busybox:latest' locally
511136ea3c5a: Pull complete
df7546f9f060: Pull complete
ea13149945cb: Pull complete
4986bf8c1536: Pull complete
busybox:latest: The image you are pulling has been verified. Important: image verificatio

Status: Downloaded newer image for busybox:latest
hello world
```

這就是標準的 `docker run` 指令，後面先接著映像檔名稱 `busybox`，然後再在由 `busybox` 這個映像檔所建立容器中，執行 `echo "hello world"` 這個指令。如果我們再執行一次，由於已經下載映像檔到到本機了，因此就會直接顯示結果

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

短短一個指令，其實已經經過了Container的建立、執行、停止等步驟了。

- `docker run`：Container建立並執行。
- `busybox`：建立這個Container的映像檔。
- `echo "hello world"`：Container中執行 `echo "hello world"` 指令，並且將結果輸出到Container所代表的標準輸出上。
- 執行完之後，Container就停止，但還是存在**!!**。

查看Container：`docker ps`

我們可以使用 `docker ps` 指令來查看執行中的**Container**，但由於這個Container已經執行完畢，因此用 `docker ps` 是無法看到。此時要加 `-a` 參數，列出所有的Container，包括已經執行結束的(即死掉的**Container**)。 `docker ps -a`

```
$ docker run busybox echo "hello world"
hello world
$ docker run busybox echo "hello world"
hello world
$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
$ docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
743b1d2c496c        busybox:latest      "echo 'hello world'"   18 seconds ago      Exited
16d0080388ed        busybox:latest      "echo 'hello world'"   24 seconds ago      Exited
```

上面可以看出，使用沒有參數的 `docker ps` 時，只會列出正在執行的Container，但 `echo` 指令執行完即結束，因此只能用 `docker ps -a` 列出已經執行完，死掉但還沒消失的Container。其STATUS是 `Exited (0) 18 seconds ago`，表示18秒前執行完成。那怎麼樣讓一個Container能一直執行下去，不要死掉或消失呢？

執行中的Container

永遠執行的ontainer

在Docker不加參數的情況下，要讓Container一直執行下去，就是執行一個無止盡的程式，如`ping localhost`。我們就來看看執行結果：

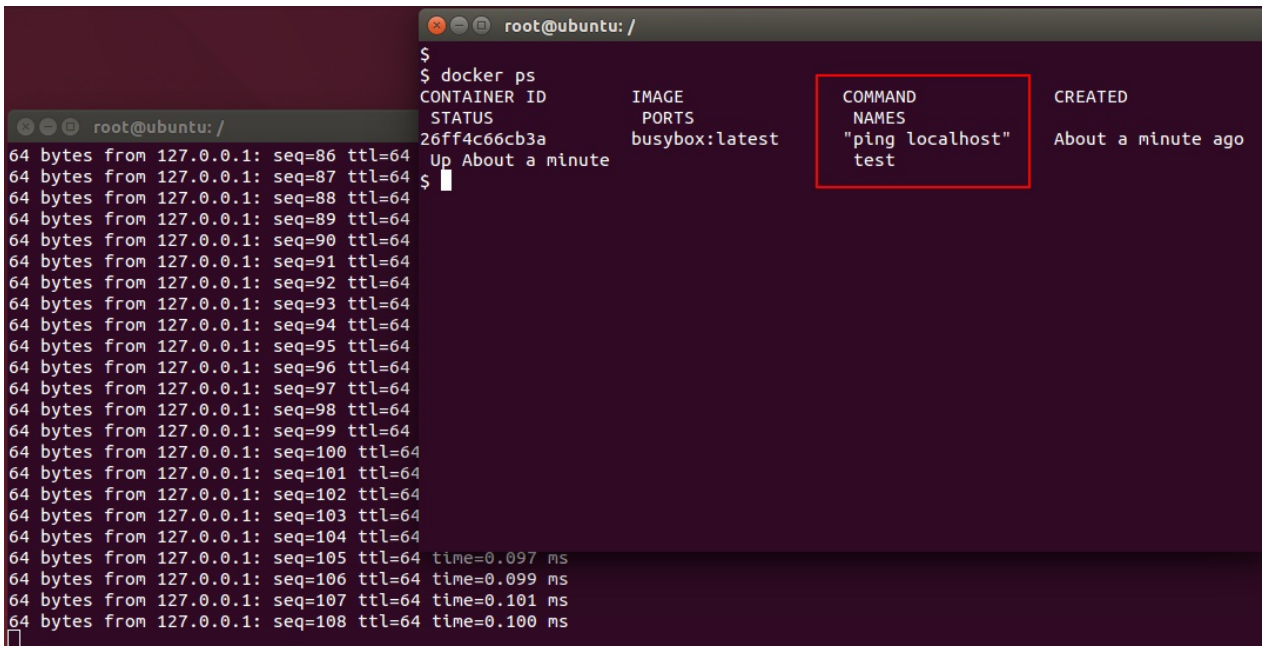
1. 執行一個不會結束的程式，如`ping localhost`：

   ```
   docker run --name test busybox ping localhost
   ```

   這次我們加了一個新的參數`--name test`，表示把這個執行的Container取名為`test`，方便以後的處理。

   ```
   $ docker run --name test busybox ping localhost
   PING localhost (127.0.0.1): 56 data bytes
   64 bytes from 127.0.0.1: seq=0 ttl=64 time=0.041 ms
   64 bytes from 127.0.0.1: seq=1 ttl=64 time=0.071 ms
   64 bytes from 127.0.0.1: seq=2 ttl=64 time=0.065 ms
   64 bytes from 127.0.0.1: seq=3 ttl=64 time=0.039 ms
   64 bytes from 127.0.0.1: seq=4 ttl=64 time=0.100 ms
   64 bytes from 127.0.0.1: seq=5 ttl=64 time=0.100 ms
   64 bytes from 127.0.0.1: seq=6 ttl=64 time=0.087 ms
   64 bytes from 127.0.0.1: seq=7 ttl=64 time=0.094 ms
   64 bytes from 127.0.0.1: seq=8 ttl=64 time=0.054 ms
   .....(無止無盡的執行)
   ```

2. 此時再開啟一個終端視窗，並且執行`docker ps`，就可以看到正在執行中的Container，這就是一個標準的「活」的Container



事實上，要讓Container能持續執行下去，除了Container本來去執行一個永遠不停止的命令之外，我們當然

也可以使用 `docker run` 本身附帶的參數來進行。

給Container標準輸出輸入裝置

Container可被視為一台獨立的電腦，因此當然可以有鍵盤及螢幕。但這邊所謂的鍵盤螢幕，就是Linux的輸入裝置 `stdin` 及輸出裝置 `stdout` 。在Docker中的指令如下：

- `-t` ： `attach` 時Container的螢幕會接到原來的螢幕上。
- `-i` ： `attach` 時鍵盤輸入會被Container接手

如果你想讓Container擁有這兩樣，在執行 `docker run` 時別忘加上參數 `-i` 或 `-t` ，如果兩個同時都加，這個Container就具備了標準的輸入(即你目前使用的鍵盤)和輸出(標準的Linux輸出，即目前操作的終端視窗)。

注意，Docker下的參數是可以合併的，因此 `docker run -t -i` 可以直接寫成 `docker run -it` 。

```
$ docker run -it --name test busybox
/ # ls
bin      etc      lib      linuxrc  mnt      proc     run      sys      usr
dev      home     lib64    media    opt      root     sbin     tmp      var
/ # exit
$
```

讀者可以發現當你執行上述指令時，Linux的提示符號從 `$` 變成了 `#` ，表示你已經從原來的Linux主機，進入了這個Container的內部操作介面了。

此時執行的 `ls` 指令顯示的是**Container內**的檔案系統列表，而非主系統的檔案系統列表。從另一個終端命令視窗進去執行 `docker ps` ，也可以看到這個Container正在執行中。當然就像任何Linux一樣，當你在Container中輸入 `exit` 之後，就會離開這個Container，此Container就會停止運行，回到外部的Linux系統。

讓Container在背景執行

大部分的Docker應用程式都是以服務方式執行，換句話說，我們不太需要進入這個Container的命令提示符號操作，此時可以在 `docker run` 後輸入 `-d` 參數，表示是Detached模式，或稱之為"Daemonized"的方式執行。此時不會進入Container，但Container照道理應該在背景執行了，可以用 `docker ps` 查看。

```
$ docker run -d busybox
e8a30f345b260893cbda28cd9ef087def15080d756e974f21789ac2389306631
$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
```

讀者應該會發現，背景並沒有這個Container的執行？不是已經下了 `-d` 參數了嗎？我們就來看看原因。

我們在本機執行 `echo "hello world"` 時，執行完畢即離開 `echo` 這個指令，這個指令是不是在背景執行並不重要，通常執行完就離開。

Docker也是一樣。就算下了 `-d` 參數，但由 `busybox` 映像檔所產生的Container，在啟動時，並沒有一個長期駐留記憶體的服務，因此也是執行完就離開這個Container，因此不會常駐在主系統的記憶體中，當然就看不到了。

由此可知，要讓Container能活著提供服務的前題，就是該**Container**有一個駐留在記憶體中的服務，下列幾個就是符合這個前題的條件：

1. 該Container在產生並啟動時，來源的映像檔就有一個開機自動執行常駐的服務，而且這個Container被我們使用 `-d` 參數丟入背景。本書稍後章節會有完整實作。

   ```
   $ docker run -d joshhu/webdemo
   docker ps
   CONTAINER ID        IMAGE                   COMMAND             CREATED             S
   964951e51154        joshhu/webdemo:latest   "/start.sh"         9 seconds ago       U
   ```

   上面的例子，可以看到這個Container在啟動時，執行了一個 `start.sh`，這個檔案就是讓Container一直執行的服務，在建立映像檔時就建立好了。

2. 手動對Container執行一個不停止的服務並把Container丟入背景執行，如 `docker run -d busybox ping localhost`。

   ```
   $ docker run -d busybox ping localhost
   d794699780ce1ae907782809887d56ee5d21e6d52d50caea13e6052d37afcb48
   $ docker ps
   CONTAINER ID        IMAGE               COMMAND             CREATED             STATU
   d794699780ce        busybox:latest      "ping localhost"    2 seconds ago       Up 2
   ```

3. 用 `-d` 參數將Container放入背景，並且讓他保持基本輸入或輸出的能力。換句話說，就是不但要加入參數 `-d`，同時也要有參數 `-t` 或 `-i`，因此加上 `-dt` 或 `-di` 或 `-idt` 均可。

   ```
   $ docker run -dt busybox
   aa94ab648f25384c40e8c1e8fbbe3945472a7220f6fa20753765f6bb1499cc69
   $ docker ps
   CONTAINER ID        IMAGE               COMMAND             CREATED             STATU
   aa94ab648f25        busybox:latest      "/bin/sh"           2 seconds ago       Up 2
   ```

Container內部及外部的執行

進入執行中Container內部

如果Container在背景常駐，我們當然就可以進入Container進行操作，以下是最常見的使用：

denter

`denter`不是Docker的指令，而是我們在前面章節匯入的一個script，只要輸入`denter <CONTAINER NAME>`即可。

```
$ docker run --name web joshhu/webdemo
chown: cannot access '/app': No such file or directory
AH00558: apache2: Could not reliably determine the server's fully qualified domain name,
^C$ docker rm -f web
web
```

在上面的範例中，首先我們執行一個Web服務，其預設的常駐程式是`apache2`。但第一次執行沒有用`-d`參數放入背景，因此所有的訊息會輸出到前景的stdout中，我們使用`Ctrl-C`強迫中斷。

```
$ docker run -d --name web joshhu/webdemo
0941e3055a30352688290746f0a175df1f6bcbf447038cd4cf3e3a43cba1f658
$ denter web
root@0941e3055a30:~# exit
logout
$ docker ps
CONTAINER ID        IMAGE                   COMMAND             CREATED             STATU
0941e3055a30        joshhu/webdemo:latest   "/start.sh"         2 minutes ago       Up 2
$
```

第二次執行有使用`-d`參數，因此這個Container的Web服務被放入背景，輸入`denter web`就以進入這個Container中操作，`exit`離開後，在`docker ps`中也可以看到這個Container在繼續執行。

docker attach

`docker attach`是用來「監管」Container用的。換句話說，在使用`docker attach`一個Container之後，就會進入這個Container的操作終端命令列，但視你之前執行`docker run`的參數，離開這個Container時，會不會中止Container的執行。

- 使用`-d`或`-id`參數：離開Container時該Container停止。
- 使用`-td`參數：離開Container時該Container繼續在背景。

```
root@ubuntu:/# docker run -id --name test busybox ping localhost
45abb8bc99cdd91c1c6a46b6a3d1477dc0d7f0c5612b63dcbe50b2a0c5badbc7
root@ubuntu:/# docker attach test
64 bytes from 127.0.0.1: seq=6 ttl=64 time=0.095 ms
```

```
64 bytes from 127.0.0.1: seq=7 ttl=64 time=0.095 ms
64 bytes from 127.0.0.1: seq=8 ttl=64 time=0.102 ms
^C
--- localhost ping statistics ---
9 packets transmitted, 9 packets received, 0% packet loss
round-trip min/avg/max = 0.051/0.092/0.111 ms
root@ubuntu:/# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATU
```

```
$ docker run -td --name test busybox ping localhost
d8c3687f46a2c26e809c556a7c810a36a73959bfcb8636fb874c9da969b2ee43
$ docker attach test
64 bytes from 127.0.0.1: seq=7 ttl=64 time=0.063 ms
64 bytes from 127.0.0.1: seq=8 ttl=64 time=0.096 ms
64 bytes from 127.0.0.1: seq=9 ttl=64 time=0.097 ms
64 bytes from 127.0.0.1: seq=10 ttl=64 time=0.093 ms
^C$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATU
d8c3687f46a2        busybox:latest      "ping localhost"    14 seconds ago      Up 13
```

在外部執行**Container內**的程式

我們可以從外部執行Container內部的程式，或直接觀看Container內的輸出。有兩個指令：

docker logs

這個指令直接讀出Container內的輸出到主機的螢幕上。舉例來說，我們執行一個不斷列出系統時間的無窮迴圈： 先在前景執行：

```
$ docker run --name printtime busybox /bin/sh -c "while true;do date;sleep 1;done"
Fri Mar 27 12:24:15 UTC 2015
Fri Mar 27 12:24:16 UTC 2015
Fri Mar 27 12:24:17 UTC 2015
Fri Mar 27 12:24:18 UTC 2015
^C$
$
```

丟到背景執行，再用 `docker logs` 在前景列出執行結果：

```
$ docker run -d --name printtime busybox /bin/sh -c "while true;do date;sleep 1;done"
5a80b74ff2fb1c0596880c2b6d3f12a1db9a27777721326fa8d296fcfab9da53
$ docker logs printtime
Fri Mar 27 12:22:25 UTC 2015
Fri Mar 27 12:22:26 UTC 2015
Fri Mar 27 12:22:27 UTC 2015
Fri Mar 27 12:22:28 UTC 2015
Fri Mar 27 12:22:29 UTC 2015
Fri Mar 27 12:22:30 UTC 2015
Fri Mar 27 12:22:31 UTC 2015
Fri Mar 27 12:22:32 UTC 2015
```

```
$
```

docker exec

這個指令很簡單，就是在外部向執行中的Container內部下指令，此時會呼叫Container內部的shell程式來執行你下的指令，而不管你之前啟動Docker時給這個Container的指令，兩個沒有關係。

```
$ docker run -d --name printtime busybox /bin/sh -c "while true;do date;sleep 1;done"
f01ceebc8330893b741a5c3323bae506b7e62d242dc8c76e4d127010281b2c9b
$ docker logs printtime
Fri Mar 27 12:34:59 UTC 2015
Fri Mar 27 12:35:00 UTC 2015
Fri Mar 27 12:35:01 UTC 2015
Fri Mar 27 12:35:02 UTC 2015
Fri Mar 27 12:35:03 UTC 2015
Fri Mar 27 12:35:04 UTC 2015
$ docker exec printtime ps
PID   USER     COMMAND
    1 root     /bin/sh -c while true;do date;sleep 1;done
   52 root     sleep 1
   53 root     ps
```

就像上例一樣，我們雖然在執行 `docker run` 時讓這個Container執行一個無限印出系統時間的指令，但使用 `docker exec` 時，也是在Container內另外啟動一個 `/bin/sh` 的shell來執行 `ps` 。

ContainerOps

RunAsVM

Docker執行時參數