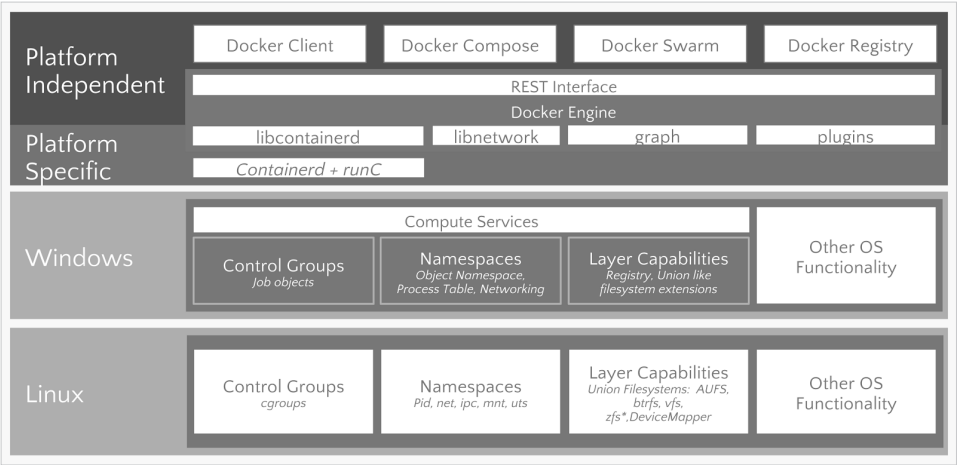


▲ Docker 平台以及被 Docker 使用的核心功能

◉ Namespaces（命名空間）

命名空間是容器的建構方塊。它有許多種類，每一種都可以把應用程式和其他應用程式隔離開來。這些命名空間由複製系統呼叫（clone system call）的方式建立。你也可以附加到現有的命名空間中。其中一些 Docker 所使用到的命名空間將在以下的小節中說明。

PID 命名空間

PID 命名空間允許每一個容器有它自己的處理程序編號（process numbering）。每一個 PID 規範它自己的處理程序階層結構。父命名空間（parent namespace）可以看到子命名空間（children namesapce），也可以變更它們，但是子命名空間就無法看到也不可以變更父命名空間。

假設有兩層結構，在上層中我們看到在子命名空間中執行的處理程序有著不同的 PID。因此，一個在子命名空間中執行的程序會有兩個不同的 PID：其中一個在子命名空間中，而另外一個則是在父命名空間中。例如，如果在 `container.sh` 容器中執行一個程式，我們就能在主機上看到這支程式。

```
bash-4.3# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
    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
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
269: eth0: <BROADCAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:ac:11:00:0b brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.11/16 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 2001:db8:1::242:ac11:b/64 scope global
       valid_lft forever preferred_lft forever
    inet6 fe80::42:acff:fe11:b/64 scope link
       valid_lft forever preferred_lft forever
bash-4.3#
```

而在主機端，看起來會像是以下這個樣子：

```
$ ip a
```

```
bash-4.3# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
    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
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
269: eth0: <BROADCAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:ac:11:00:0b brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.11/16 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 2001:db8:1::242:ac11:b/64 scope global
       valid_lft forever preferred_lft forever
    inet6 fe80::42:acff:fe11:b/64 scope link
       valid_lft forever preferred_lft forever
bash-4.3#
```

而且，每一個 net 命名空間都有它自己的路由表以及防火牆規則。

IPC 命名空間

IPC（**inter-process communication**）命名空間提供 semaphore、message queues、以及 shared memory segments。這些技術現在不是那麼常用了，但還是有些程式需要它。

如果 PIC 資源在某個容器中被建立，而要在另外一個容器中用掉，那麼這個執行在第一個容器中的應用程式就會失敗。因為 IPC 命名空間的關係，

◉ 可參閱

你可以瀏覽「*Get Started with Docker for Mac*」以協助學習關於應用程式以及如何最佳使用它的引導。這份文件可以在下列網址找到：

`https://docs.docker.com/docker-for-mac/`

提取映像檔並執行容器

我將先以這個來自於下一章的訣竅簡單介紹一些概念。在本章接下來的章節中才會完整解說這些主題。現在，我們要做的是提取一個映像檔（pulling an image）並且執行它。你將會在這個訣竅中開始認識 Docker 的架構以及它的一些元件。

◉ 備妥

請先把 Docker 安裝到你的系統中。

◉ 如何做

要提取一個映像檔並且執行容器，請依照以下的步驟進行：

1. 請使用以下的指令提取一個映像檔：

```
$ docker image pull alpine
```

2. 使用以下的指令可以列出已存在的映像檔：

```
$ docker image ls
```

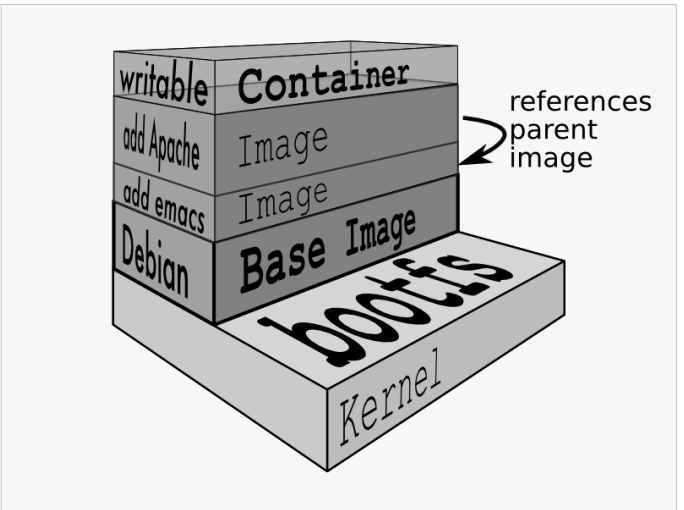
```
$ docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
hello-world         latest              2cb0d9787c4d        2 weeks ago         1.85kB
alpine              latest              11cd0b38bc3c        3 weeks ago         4.41MB
$
```

所以在我們的例子中，Docker client 傳送一個要求給在本地機器上執行的 daemon，而它會連線到公用的 Docker registry 並且下載這個映像檔。一旦映像檔被下載之後，就可以執行它了。

◉ 補充資訊

讓我們來探討一些在前面的訣竅中遇到的關鍵字：

- **Images（映像檔）**：Docker 映像檔是唯讀的樣板，它們在執行之後就成為容器。實現這個想法的概念是以一個作為基礎的映像檔再層層往上疊。例如，我們可以有一個 Alpine 或是 Ubuntu 的基礎映像檔（base image），然後在其上安裝一些套件或做一些修改以建立一個新的層（layer），如此基礎映像檔和新的層就可以把它們結合一起視為一個新的映像檔。舉個例子，在下面這張圖中，Debian 是基礎映像檔，Emacs 和 Apache 則是兩個被加在上面的層。它們具有高可攜性，而且可以很輕易地被共用：



「層（layers）」被逐次地疊加在基礎映像檔上方以建立一個單一聚合檔案系統。

◉ 備妥

請確定 Docker daemon 在主機上正常運作，而且可以被 Docker client 順利地連上。

◉ 如何做

請執行以下任一行指令以列出所有的映像檔：

```
$ docker image ls
$ docker images
```

```
$ docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ubuntu              latest              ccc7a11d65b1        2 weeks ago         120MB
centos              centos7             328edcd84f1b        3 weeks ago         193MB
alpine              edge                6ab1c97283af        4 weeks ago         3.95MB
nginx               <none>              b8efb18f159b        5 weeks ago         107MB
alpine              3.6                 7328f6f8b418        2 months ago        3.96MB
alpine              latest              7328f6f8b418        2 months ago        3.96MB
alpine              3.5                 074d602a59d7        2 months ago        3.99MB
alpine              3.4                 f64255f97787        2 months ago        4.81MB
alpine              3.3                 606fed0878ec        2 months ago        4.81MB
alpine              3.2                 39be345c901f        2 months ago        5.26MB
alpine              3.1                 00772ebf9244        2 months ago        5.04MB
alpine              2.7                 93f518ec2c41        19 months ago       4.71MB
alpine              2.6                 e738dfbe7a10        19 months ago       4.5MB
```

◉ 如何辦到的

Docker client 可以和 Docker engine 溝通且取得那些已經被下載（提取過的）到 Docker 主機上映像檔列表。

◉ 補充資訊

所有名稱相同但是具不同標記的映像檔都被下載。在這裡要特別留意的一個有趣的地方是，它們有相同的名稱但是卻有不一樣的標記。此外，有兩個的 `IMAGE ID` 都是 `7328f6f8b418`，但是卻有不同的標記。

- 容器目前的狀態
- 此容器對外的連接埠
- 此容器的名稱

◉ 補充資訊

如果想要列出包括執行中以及停止中的容器，請加上「-a」這個選項：

```
$ docker container ls -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS                      PORTS               NAMES
bae274ee1e60        nginx:alpine        "nginx -g 'daemon ..."   4 minutes ago       Up 4 minutes                80/tcp              awesome_perlman
38c34651c8ca        node:alpine         "node"                   5 minutes ago       Exited (0) 5 minutes ago                        nifty_curran
1a9d18cef22d        ubuntu              "/bin/bash"              7 minutes ago       Up 7 minutes                                    gracious_kirch
```

如果只想要列出所有容器的 ID，可以使用「-aq」選項：

```
$ docker container ls -aq
bae274ee1e60
38c34651c8ca
1a9d18cef22d
```

要顯示上一個建立的容器，包括沒有在執行的容器，請執行以下的指令：

```
$ docker container ls -l
```

如果在 ps 指令中使用「--filter/-f」參數，可以列出使用特定標籤的容器。你可以參閱本章的「**為容器建立標籤及過濾容器**」。

◉ 可參閱

要查詢 docker container ls 的使用說明，可輸入如下：

```
$ docker container ls --help
```

你可以在 Docker 的網站上看到相關的說明文件：

https://docs.docker.com/engine/reference/commandline/container_ls/

移除所有停止中的容器

我們可以使用一個指令就移除所有停止中的容器，在這個訣竅中將會建立一堆處於停止狀態的容器，然後再把它們全部刪除。

◉ 備妥

請確定 Docker daemon 1.13（或更新的版本）在主機上執行，而且可以被 Docker client 順利地連接上。你也需要一些處於停止狀態或是正在執行中的容器，這樣才有容器可以移除。

◉ 如何做

請使用以下的指令：

```
$ docker container prune [OPTIONS]
```

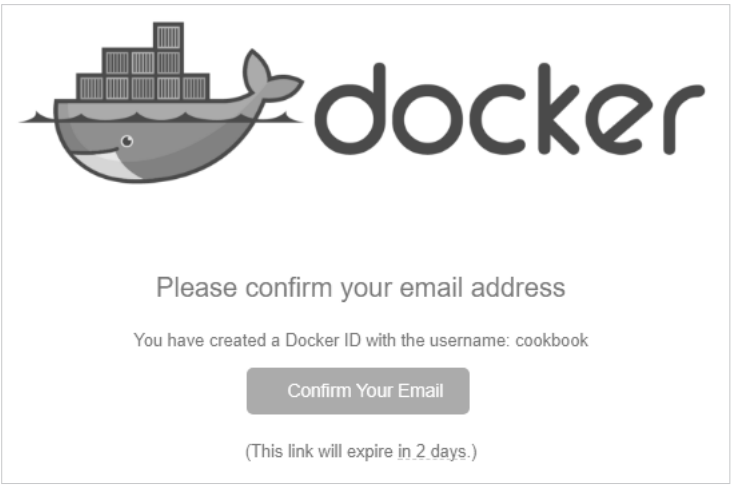
首先，建立容器，接著使用以下的指令進行移除：

```
$ docker container create --name c1 ubuntu /bin/bash
$ docker container run --name c2 ubuntu /bin/bash
$ docker container prune
```

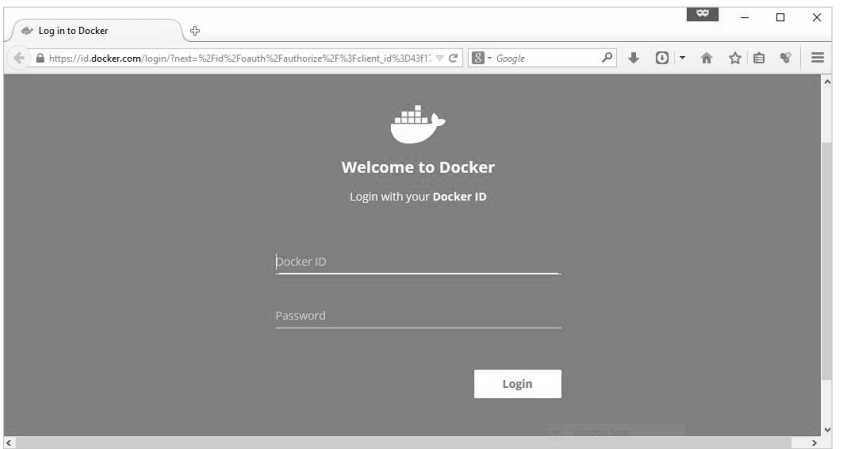
```
$ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
63f1f7de6adc        ubuntu              "/bin/bash"         24 hours ago        Up 24 hours                             nervous_swirles
$ docker container create --name c1 ubuntu /bin/bash
35b2e0b876bfe44ad0625e7f2265c205079eafb92364aa4fe83631d43d20a24b
$ docker container run --name c2 ubuntu /bin/bash
$ docker container ls -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS                      PORTS               NAMES
20a439fd9ad6        ubuntu              "/bin/bash"         10 seconds ago      Exited (0) 10 seconds ago                       c2
35b2e0b876bf        ubuntu              "/bin/bash"         24 seconds ago      Created                                         c1
63f1f7de6adc        ubuntu              "/bin/bash"         24 hours ago        Up 24 hours                                     nervous_swirles
$ docker container prune
WARNING! This will remove all stopped containers.
Are you sure you want to continue? [y/N] y
Deleted Containers:
20a439fd9ad61a74c6da1f17b40bc527803804f58024f67a2ed81e79e5b88775
35b2e0b876bfe44ad0625e7f2265c205079eafb92364aa4fe83631d43d20a24b

Total reclaimed space: 0B
$ docker container ls -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
63f1f7de6adc        ubuntu              "/bin/bash"         24 hours ago        Up 24 hours                             nervous_swirles
```

4. 由上圖可見，帳號還沒有被啟用。為了啟用帳號，需要到你的電子郵件帳號收件匣中找到並開啟電子郵件確認信，然後點擊「**Confirm Your Email**」按鈕，如下圖：



5. 一旦確認了電子郵件，你將會被引導到一個「**Welocme to Docker**」的歡迎畫面，如下圖所示：



現在，已經成功建立以及啟用你的 Docker Hub 帳號了。

```
$ docker login [OPTIONS] [SERVER]
$ docker logout [SERVER]
```

在預設的情況下，包括 `docker login` 以及 `docker logout` 指令都是以 `https://hub.docker.com` 作為預設的 registry，這當然也是可以變更的設定。

接著我們將更進一步地說明這個程序，如下圖所示：

- 登入預設的 Docker registry
- 登入在 `https://about.gitlab.com/` 中的 registry
- 讀出已存在的登入資訊
- 從所有的 registry 中登出

```
$ 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: cookbook
Password:
Login Succeeded
$ docker info | egrep ^\(Username\|Registry\)
Username: cookbook
Registry: https://index.docker.io/v1/
$ docker login registry.gitlab.com
Username: sjeeva@gmail.com
Password:
Login Succeeded
$ cat ~/.docker/config.json
{
        "auths": {
                "https://index.docker.io/v1/": {
                        "auth": "Y29va2Jvb2s6c29zZWN1cmU="
                },
                "registry.gitlab.com": {
                        "auth": "c2plZXZhQGdtYWlsLmNvbTpzb3NlY3VyZQ=="
                }
        }
}$
$ echo -n Y29va2Jvb2s6c29zZWN1cmU= | base64 -d
cookbook:sosecure$
$ docker logout
Removing login credentials for https://index.docker.io/v1/
$ docker logout
Not logged in to https://index.docker.io/v1/
$ docker logout registry.gitlab.com
Removing login credentials for registry.gitlab.com
$ cat ~/.docker/config.json
{
        "auths": {}
}$
```

如果映像檔有一個以上的標記，在移除映像檔之前需要先移除這些標記。不過，另外一種方式是在 docker image rm 後面使用「-f」或是「--force」參數來強制一併移除所有的標記。

以下是 docker image rm 指令的語法：

```
docker image rm [OPTIONS] IMAGE [IMAGE...]
```

在這個訣竅中將會對一個映像檔建立多個標記，然後展示如何去移除它們。

◉ 備妥

在 Docker 主機上應該要有一個或多個 Docker 映像檔而且是可用的。

◉ 如何做

請依照以下的步驟進行：

1. 先選用其中一個已存在的映像檔，在上面新增數個標記，如下圖：

```
$ docker image tag centos:latest centos:tag1
$ docker image tag centos:latest centos:tag2
$ docker image tag centos:latest centos:tag3
$ docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
cookbook/myapache2  latest              061058607f39        2 weeks ago         258MB
myapache2           latest              061058607f39        2 weeks ago         258MB
ubuntu              latest              ccc7a11d65b1        6 weeks ago         120MB
centos              latest              328edcd84f1b        7 weeks ago         193MB
centos              tag1                328edcd84f1b        7 weeks ago         193MB
centos              tag2                328edcd84f1b        7 weeks ago         193MB
centos              tag3                328edcd84f1b        7 weeks ago         193MB
alpine              latest              7328f6f8b418        2 months ago        3.97MB
```

現在，我們已經選擇了映像檔 ID 是 328edcd84f1b 這個 centos 映像檔，而且加上了另外 3 個標記：tag1、tag2、以及 tag3。所有的標記都是同一個 ID 是 328edcd84f1b 的那個映像檔。

很明顯地，這個映像檔小多了，這個 Docker 映像檔只比可執行的 demo 多了 20 個位元組而已。

◉ 如何辦到的

Docker 建置系統直觀地瞭解這個在 FROM 指令中的保留映像檔名稱 scratch，然後開始綁定一個不加上任何額外資料層的基礎映像檔。所以，在這個訣竅中，Docker 建置系統只有綁定靜態連結的可執行檔 demo 以及這個映像檔的中繼資料。

◉ 補充資訊

就如同之前提到的，scratch 映像檔並不會增加任何額外的資料層到映像檔。docker image history 指令這時候就可以派上用場，在此可以用來列出映像檔的所有層，如下所示：

```
$ docker image history scratch-demo
IMAGE               CREATED             CREATED BY                                      SIZE
e4c65195b92a        16 hours ago        /bin/sh -c #(nop)  CMD ["/demo"]                0B
4c5ae0859fca        16 hours ago        /bin/sh -c #(nop) ADD file:583335d88ee487f...   949kB
```

正如同我們所看到的，scratch 這個基礎映像檔並沒有被加上任何額外的 layer。

◉ 可參閱

- 在 Docker Hub 上可以找到詳細的說明文件：

  https://hub.docker.com/_/scratch/

- 以下說明建立基礎映像檔的另外一種方式：

  https://docs.docker.com/articles/baseimages/

◉ 備妥

在開始之前，我們需要一個或多個 Docker 映像檔在 Docker daemon 的主機上執行。我們也需要確保 Graphviz 已經被安裝好了。

◉ 如何做

執行 nate/dockviz 容器並提供 images --dot 作為命令列參數，以及把輸出用管線轉到 dot（Graphviz）指令中，以下列指令產生出映像檔階層結構：

```
$ docker run --rm \
            -v /var/run/docker.sock:/var/run/docker.sock \
            nate/dockviz \
            images --dot | dot -Tpng -o images-graph.png
```

底下這張圖是在 Docker 主機上的映像檔之圖形化檢視：