

資工二 110710502 楊淞元
期中報告 -- Docker 虛擬機研究

目錄

內容

目錄.....	2
虛擬機介紹.....	3
定義.....	4
Docker 簡介	5
Docker 三元素	10
docker 的使用方式.....	12
參考資料.....	17

虛擬機介紹

虛擬機器（英語：**virtual machine**），在電腦科學中的體系結構里，是指一種特殊的軟體，可以在電腦平台和終端用戶之間建立一種環境，而終端用戶則是基於虛擬機器這個軟體所建立的環境來操作其它軟體。虛擬機器（**VM**）是電腦系統的仿真器，通過軟體類比具有完整硬體系統功能的、執行在一個完全隔離環境中的完整電腦系統，能提供物理電腦的功能。

有不同種類的虛擬機器，每種虛擬機器具有不同的功能：

系統虛擬機器（也稱為全虛擬化虛擬機器）可代替物理電腦。它提供了執行整個作業系統所需的功能。虛擬機器監視器（**hypervisor**）共享和管理硬體，從而允許有相互隔離但存在於同一物理機器上的多個環境。現代虛擬機器監視器使用虛擬化專用硬體（主要是主機CPU）來進行硬體輔助虛擬化。

程式虛擬機器 被設計用來在與平台無關的環境中執行電腦程式。

定義

虛擬機器最初由波佩克與戈德堡定義為有效的、獨立的真實機器的副本。當前包括跟任何真實機器無關的虛擬機器。

虛擬機器根據它們的運用和與直接機器的相關性分為兩大類。「系統虛擬機器」提供一個可以執行完整作業系統的完整系統平台。「程式虛擬機器」則為執行單個電腦程式設計，這意味它支援單個行程。

虛擬機器的一個本質特點是執行在虛擬機器上的軟體被局限在虛擬機器提供的資源里，也就是說它不能超出虛擬世界。「作業系統層虛擬化」不提供完整作業系統環境，將母機核心分給多個獨立空間的應用程式，不同於系統虛擬機需要運行完整作業系統，也不像程式虛擬機器運行特定程式語言。

系統虛擬機器

例如：VirtualBox、VMware

程式虛擬機器

例如：Java 虛擬機（JVM）

作業系統層虛擬化

例如：Docker

Docker 簡介

Docker 的歷史

Docker 是一個開源專案，誕生於 2013 年初，最初是 dotCloud 公司內部的一個業餘專案。它基於 Google 公司推出的 Go 語言實作。

專案後來加入了 Linux 基金會，遵從了 Apache 2.0 協議，原始碼在 GitHub 上進行維護。

Docker 自開源後受到廣泛的關注和討論，以至於 dotCloud 公司後來都改名為 Docker Inc。Redhat 已經在其 RHEL6.5 中集中支援 Docker；

Google 也在其 PaaS 產品中廣泛應用。

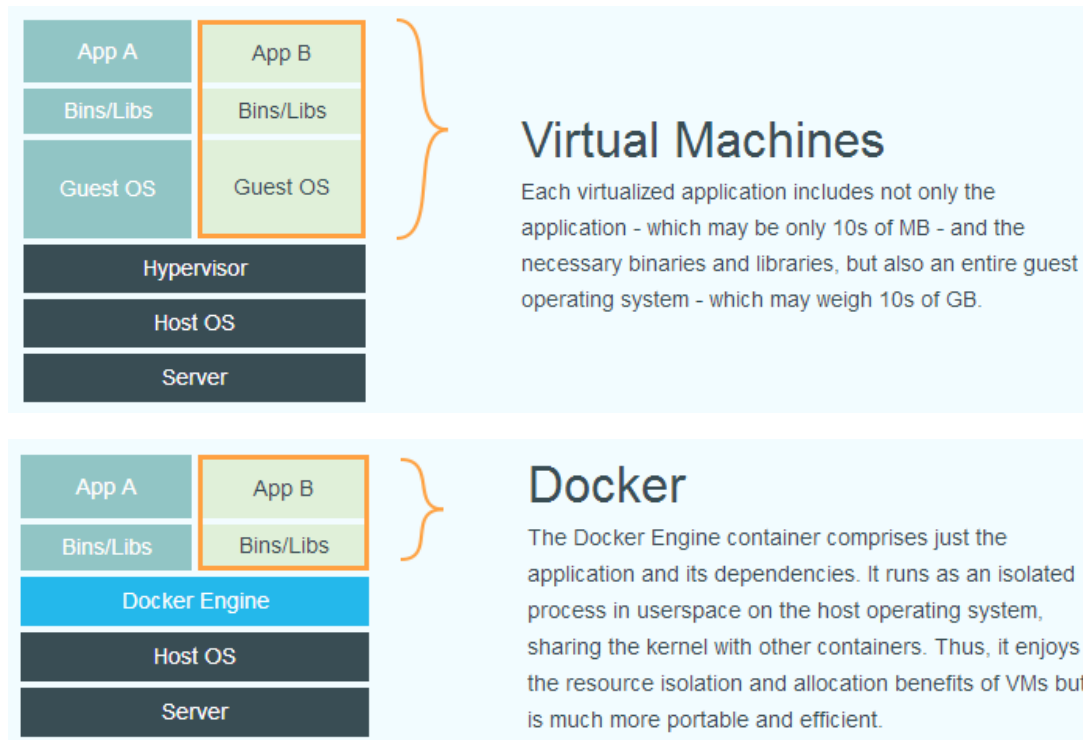
Docker 專案的目標是實作輕量級的作業系統虛擬化解決方案。

Docker 的基礎是 Linux 容器（LXC）等技術。

在 LXC 的基礎上 Docker 進行了進一步的封裝，讓使用者不需要去關心容器的管理，使得操作更為簡便。

使用者操作 Docker 的容器就像操作一個快速輕量級的虛擬機一樣簡單。

下面的圖片比較了 **Docker** 和傳統虛擬化方式的不同之處，可見容器是在作業系統層面上實作虛擬化，直接使用本地主機的作業系統，而傳統方式則是在硬體層面實作。



虛擬化

在了解 **Docker** 這項技術前，首先要先了解何謂虛擬化。

白話來說虛擬化要解決的問題就是：

「我寫好了一支程式，在我的電腦上可以正常運作，但搬到你的電腦上可能就爆掉」。

會有這樣的問題是因為：每台電腦的作業系統與硬體配置不盡相同，我的程式可能剛好只跟我電腦上的環境相容。而虛擬化要做的就是模擬出一個環境，讓程式可以在不同硬體上執行時，都以為自己在同一個環境中執行。

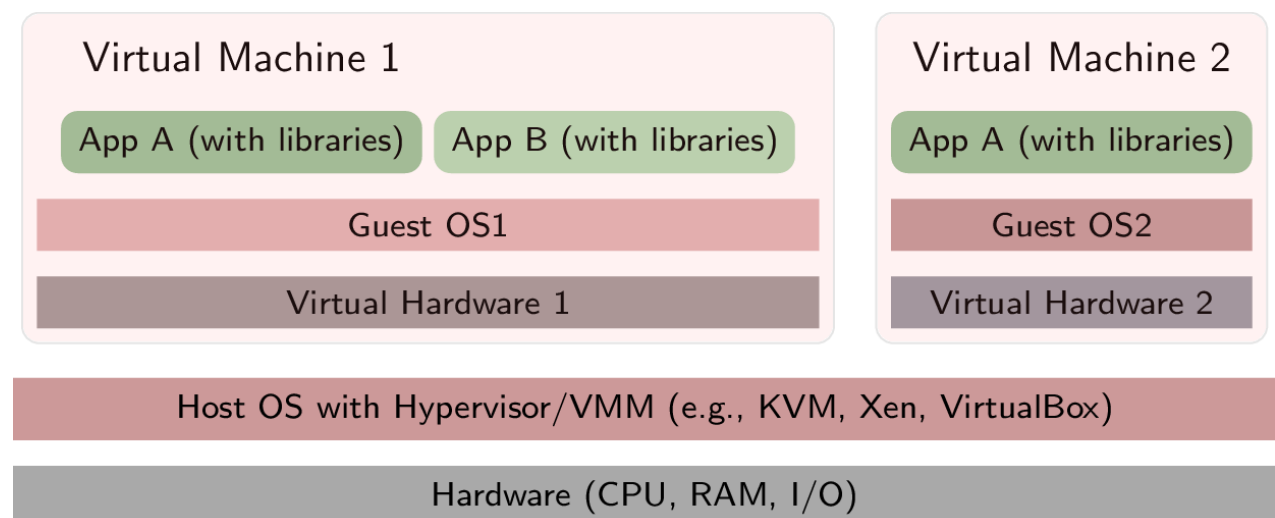
目前常見用來比較的虛擬化技術有兩種，一種是在系統層級的虛擬化技術，在這我們叫它稱虛擬機器（**Virtual machine**）。另外一種則是在作業系統層級，此稱容器（**Container**）。前者的代表如 **Virtual Box**，而後者如 **Docker**。

虛擬機器 vs 容器

虛擬機器（以作業系統為中心）

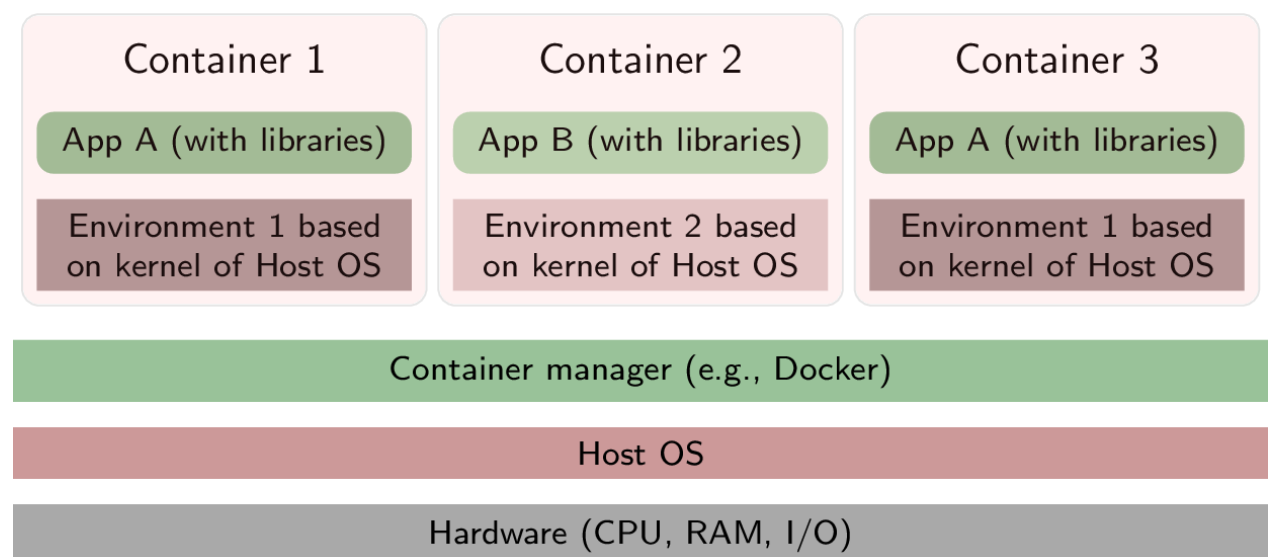
虛擬化的目標：將一個應用程式所需的執行環境打包起來，建立一個獨立環境，方便在不同的硬體中移動。

虛擬機器是在系統層上虛擬化，透過 **Hypervisor** 在目標的機器上提供可以執行一個或多個虛擬機器的平台。而這些虛擬機器可以執行完整的作業系統。簡單來說，**Hypervisor** 就是一個可以讓你在作業系統（**Host OS**）上面再裝一個作業系統（**Guest OS**），然後讓兩個作業系統彼此不會打架的平台。透過選擇不同的 **Guest OS**，虛擬機器的技術就可以確保只要我的程式在該 **Guest OS** 上可以正常運作，那放到你的電腦上跑時，可以不管你的 **Host OS** 是什麼，只要在你的 **Host OS** 上先裝上我的 **Guest OS**，我的程式就可以正常在你的電腦上運作。



容器（以應用程式為中心）

容器化的目標：改善虛擬機器因為需要裝 Guest OS 導致啟動慢、佔較大記憶體的問題。容器是在作業系統層上虛擬化，透過 Container Manager 直接將一個應用程式所需的程式碼、函式庫打包，建立資源控管機制隔離各個容器，並分配 Host OS 上的系統資源。透過容器，應用程式不需要再另外安裝作業系統（Guest OS）也可以執行。



Docker 三元素

使用 Docker 時最重要的三個元素：映像檔、容器、倉庫。

用一個簡單的比喻來解釋，如果映像檔是一個做蛋糕的模具，容器則是用該模具烤出來的蛋糕，而倉庫是用來集中放置模具們的收納櫃。接下來讓我們搭配這個比喻深入一點解釋這三個概念：

映像檔 Image

Docker 映像檔是一個模板，用來重複產生容器實體。例如：一個映像檔裡可以包含一個完整的 MySQL 服務、一個 Golang 的編譯環境、或是一個 Ubuntu 作業系統。

透過 Docker 映像檔，我們可以快速的產生可以執行應用程式的容器。而 Docker 映像檔可以透過撰寫由命令行構成的 Dockerfile 輕鬆建立，或甚至可以從公開的地方下載已經做好的映像檔來使用。舉例來說，如果我今天想要一個 node.js 的執行環境跑我寫好的程式，我可以直接到上 DockerHub 找到相對應的 node.js 映像檔，而不需要自己想辦法打包一個執行環境。

容器 Container

就像是用蛋糕模具烤出來的蛋糕本體，容器是用映像檔建立出來的執行實例。它可以被啟動、開始、停止、刪除。每個容器都是相互

隔離、保證安全的平台。

可以把容器看做是一個執行的應用程式加上執行它的簡易版 **Linux** 環境（包括 **root** 使用者權限、程式空間、使用者空間和網路空間等）。

另外要注意的是，**Docker** 映像檔是唯讀（**read-only**）的，而容器在啟動的時候會建立一層可以被修改的可寫層作為最上層，讓容器的功能可以再擴充。這點在下面的實例會有更多補充。

倉庫 **Repository**

倉庫（**Repository**）是集中存放映像檔檔案的場所，也可以想像成存放蛋糕模具的大本營。倉庫註冊伺服器（**Registry**）上則存放著多個倉庫。

最大的公開倉庫註冊伺服器是上面提到過的 **Docker Hub**，存放了數量龐大的映像檔供使用者下載，我們可以輕鬆在上面找到各式各樣現成實用的映像檔。

而 **Docker** 倉庫註冊伺服器的概念就跟 **Github** 類似，你可以在上面建立多個倉庫，然後透過 **push**、**pull** 的方式上傳、存取。

docker 的使用方式

安裝前需知

你需要有一台 64bit 的伺服器資源，大部分的 image 都是使用 64 位元的作業系統環境開始，也是 Docker 預設支援。

那台伺服器在這篇中假設是 Linux，選用 Linux 為 Host 是因為這種情境下才是 Docker 最原始標準配備，其它環境(Mac, Windows)都是透過作業系統容器(VM)來模擬。

Linux kernel 的版本需大於 3.10 。

最後是面對指令視窗有勇氣按下鍵盤控制和不找到答案不放棄的心。

大部分：Linux 的版本很多是因為它開放原始碼，而造就各路英雄將這個偉大的工具移植在不同運算平台上，所以並非絕對。

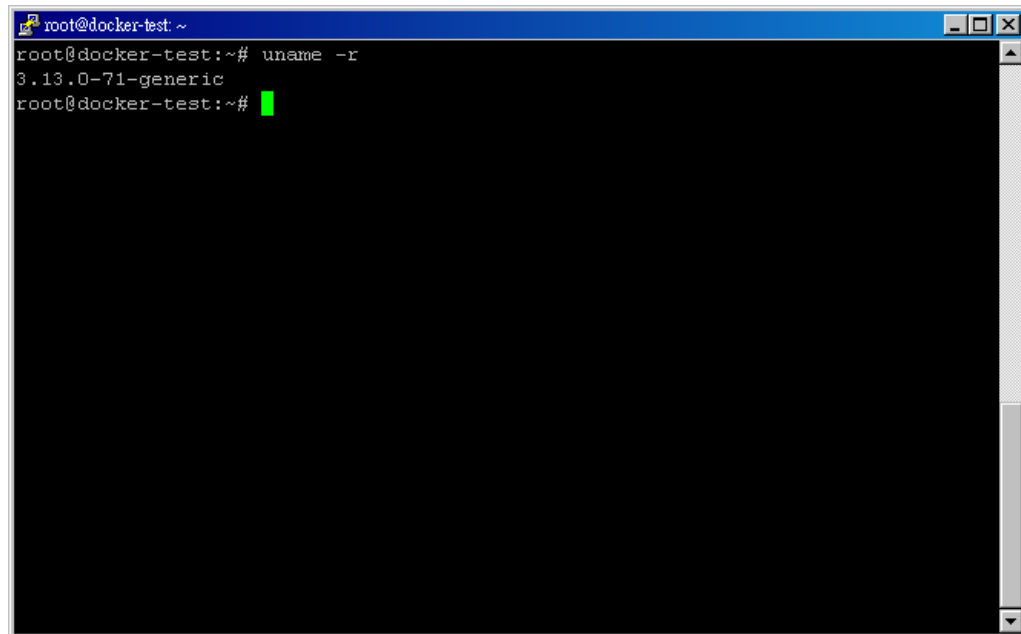
Image：Docker 在執行時需要一個基底的環境，我們將這個環境的資源稱做 Image，與作業系統安裝時的映像檔有著差不多的概念，而由 image 建立起的容器稱為 Container。

Host：執行 Docker 或是 VMs 這兩種容器"軟體"時會需要一個作業系統來管理資源，那個作業系統就稱為 Host。

開始安裝吧！

Linux 的版本眾多，而下面選用 Ubuntu 14.04.3 版來做示範

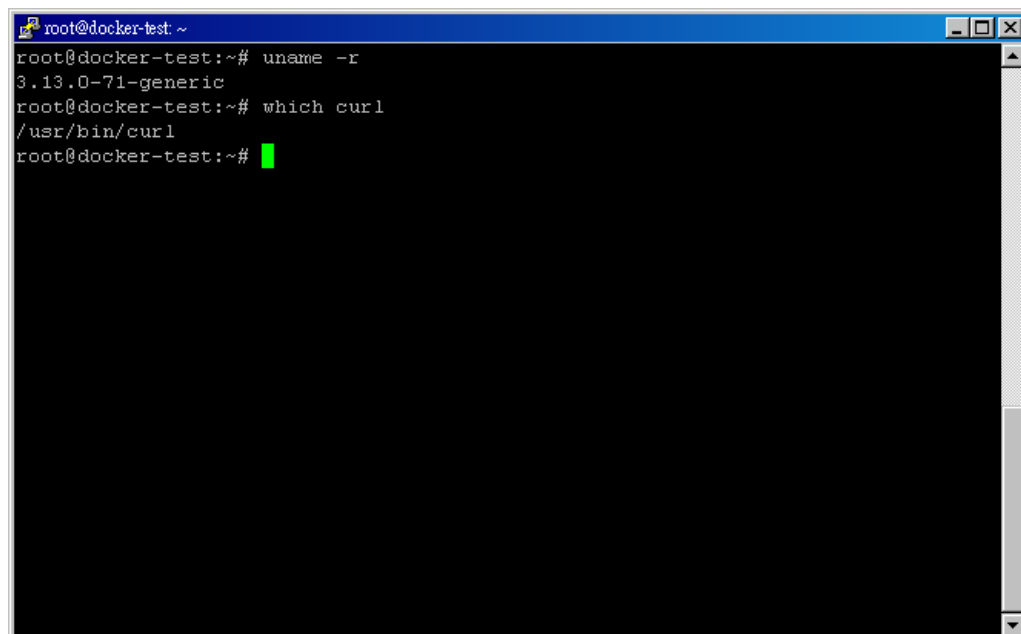
首先，檢查環境是否滿足 Docker 的需求



```
root@doker-test: ~  
root@doker-test:~# uname -r  
3.13.0-71-generic  
root@doker-test:~#
```

顯示 3.13.0-71-generic 有大於 3.10 可以放心下一步

檢測有無安裝需要的工具： curl



```
root@doker-test: ~  
root@doker-test:~# uname -r  
3.13.0-71-generic  
root@doker-test:~# which curl  
/usr/bin/curl  
root@doker-test:~#
```

如果無顯示結果，請執行 `sudo apt-get install curl`

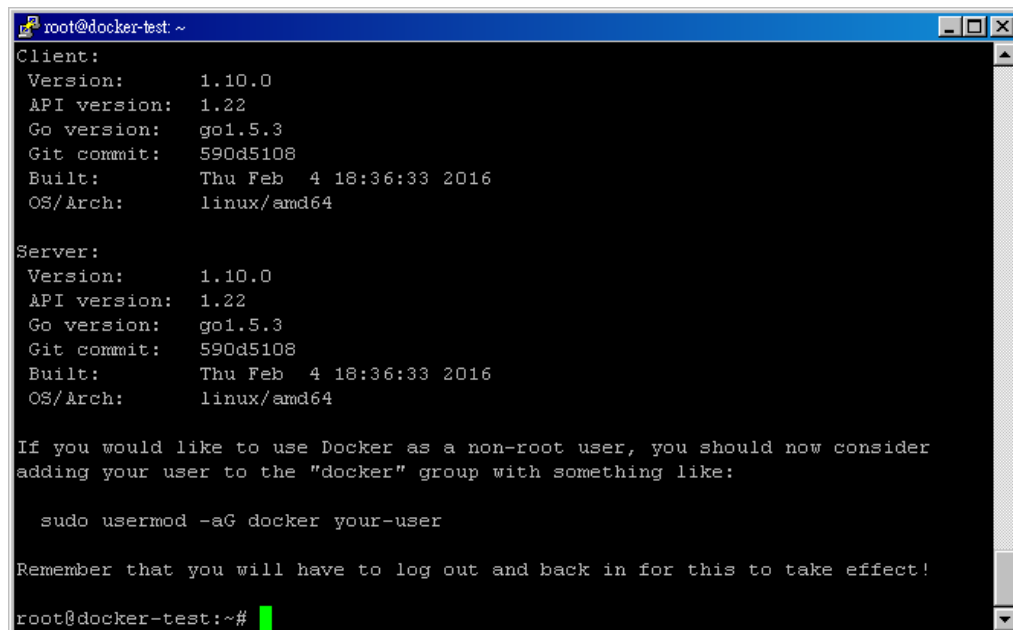
更新套件管理工具

```
root@docker-test: ~  
root@docker-test:~# uname -r  
3.13.0-71-generic  
root@docker-test:~# which curl  
/usr/bin/curl  
root@docker-test:~# apt-get update  
Ign http://mirrors.digitalocean.com trusty InRelease  
Get:1 http://mirrors.digitalocean.com trusty-updates InRelease [64.4 kB]  
Get:2 http://security.ubuntu.com trusty-security InRelease [64.4 kB]  
61% [1 InRelease gpgv 64.4 kB] [Waiting for headers] [2 InRelease 14.2 kB/64.4
```

完成更新後如圖輸入：`curl -fsSL https://get.docker.com/ | sh` 指令

```
Get:48 http://mirrors.digitalocean.com trusty/restricted Translation-en [3,457 B]  
]  
Hit http://mirrors.digitalocean.com trusty/universe Translation-en  
Get:49 http://security.ubuntu.com trusty-security/universe 1386 Packages [123 kB]  
]  
Ign http://mirrors.digitalocean.com trusty/main Translation-en_US  
Get:50 http://security.ubuntu.com trusty-security/main Translation-en [228 kB]  
Ign http://mirrors.digitalocean.com trusty/multiverse Translation-en_US  
Ign http://mirrors.digitalocean.com trusty/restricted Translation-en_US  
Ign http://mirrors.digitalocean.com trusty/universe Translation-en_US  
Get:51 http://security.ubuntu.com trusty-security/universe Translation-en [72.1  
kB]  
Fetched 5,490 kB in 13s (399 kB/s)  
Reading package lists... Done  
root@docker-test:~# curl -fsSL https://get.docker.com/ | sh  
apparmor is enabled in the kernel and apparmor utils were already installed  
+ sh -c apt-key adv --keyserver hkp://p80.pool.sks-keyservers.net:80 --recv-keys  
58118E89F3A912897C070ADB76221572C52609D  
Executing: gpg --ignore-time-conflict --no-options --no-default-keyring --homedir  
/tmp/tmp.AjQ400MUSx --no-auto-check-trustdb --trust-model always --keyring /et  
c/apt/trusted.gpg --primary-keyring /etc/apt/trusted.gpg --keyserver hkp://p80.p  
ool.sks-keyservers.net:80 --recv-keys 58118E89F3A912897C070ADB76221572C52609D  
gpg: requesting key 2C52609D from hkp server p80.pool.sks-keyservers.net
```

開始進行安裝



```
root@docker-test: ~
Client:
Version:      1.10.0
API version:  1.22
Go version:   go1.5.3
Git commit:   590d5108
Built:        Thu Feb  4 18:36:33 2016
OS/Arch:      linux/amd64

Server:
Version:      1.10.0
API version:  1.22
Go version:   go1.5.3
Git commit:   590d5108
Built:        Thu Feb  4 18:36:33 2016
OS/Arch:      linux/amd64

If you would like to use Docker as a non-root user, you should now consider
adding your user to the "docker" group with something like:

    sudo usermod -aG docker your-user

Remember that you will have to log out and back in for this to take effect!

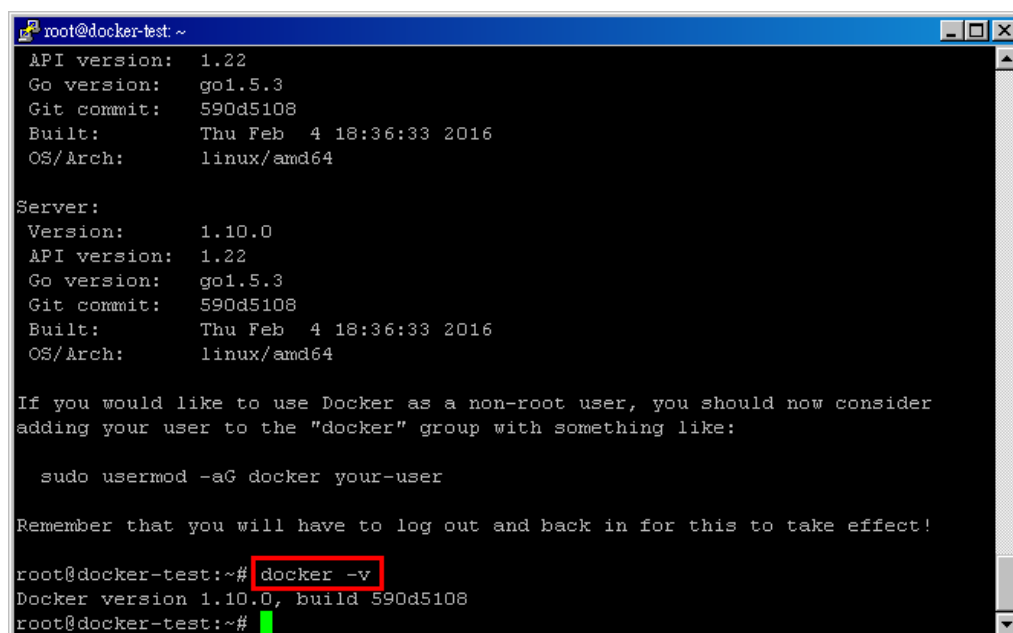
root@docker-test:~#
```

完成後的畫面，如果未來管理的使用者不是 root (建議)，可以使用

`sudo usermod -aG docker HELLOSANTA` 指令指派，記得替換

HELLOSANTA 為你的使用者名稱哦！

接著再輸入 `docker -v` 確認正確執行。



```
root@docker-test: ~
API version:  1.22
Go version:   go1.5.3
Git commit:   590d5108
Built:        Thu Feb  4 18:36:33 2016
OS/Arch:      linux/amd64

Server:
Version:      1.10.0
API version:  1.22
Go version:   go1.5.3
Git commit:   590d5108
Built:        Thu Feb  4 18:36:33 2016
OS/Arch:      linux/amd64

If you would like to use Docker as a non-root user, you should now consider
adding your user to the "docker" group with something like:

    sudo usermod -aG docker your-user

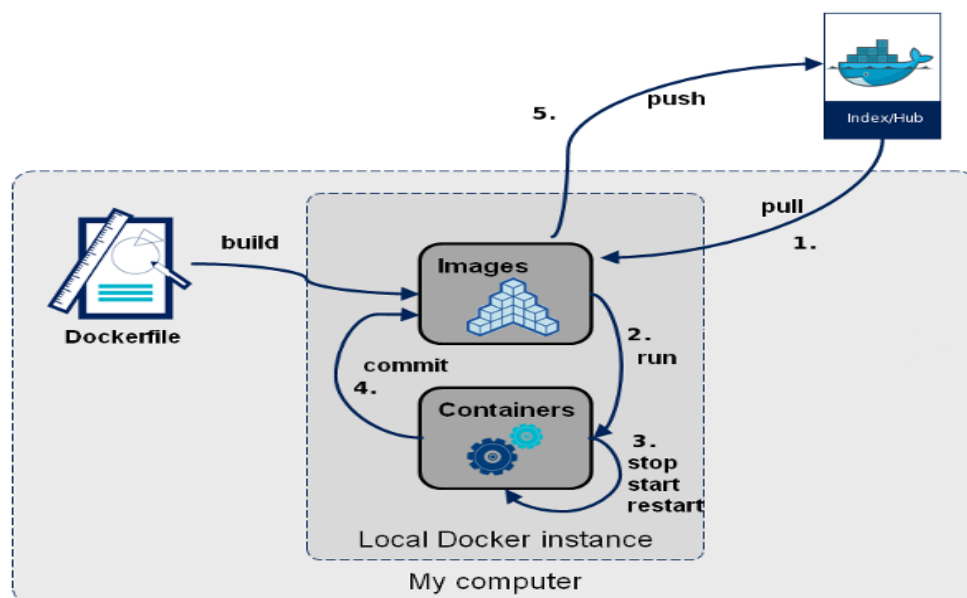
Remember that you will have to log out and back in for this to take effect!

root@docker-test:~# docker -v
Docker version 1.10.0, build 590d5108
root@docker-test:~#
```

來個 Hello World 開啟容器的世界吧～

```
root@docker-test: ~  
root@docker-test:~# docker run hello-world  
Unable to find image 'hello-world:latest' locally  
latest: Pulling from library/hello-world  
  
03f4658f8b78: Pull complete  
a3ed95caeb02: Pull complete  
Digest: sha256:8be990ef2aeb16dcb9271ddfe2610fa6658d13f6dfb8bc72074cc1ca36966a7  
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.  
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 Hub account:  
https://hub.docker.com  
  
For more examples and ideas, visit:  
https://docs.docker.com/userguide/  
  
root@docker-test:~#
```

輸入： docker run hello-world



圖示為使用這套 Docker 的流程，不論是自己建立 image (透過 Dockerfile)或是用方便的 DockerHub 直接下指令抓，都很彈性，有了社群力量，這股趨勢正旺呢！

參考資料

<https://medium.com/unorthodox-paranoid/docker-tutorial-101-c3808b899ac6>
https://philipzheng.gitbook.io/docker_practice/introduction/what
<https://blog.hellosanta.com.tw/%E7%B6%B2%E7%AB%99%E8%A8%AD%E8%A8%88/%E4%BC%BA%E6%9C%8D%E5%99%A8/%E6%95%99%E4%BD%A0%E4%B8%80%E6%AC%A1%E5%AD%B8%E6%9C%83%E5%AE%89%E8%A3%9D-docker-%E9%96%8B%E5%A7%8B%E7%8E%A9%E8%BD%89-container%C2%A0%E5%AE%B9%E5%99%A8%E4%B8%96%E7%95%8C>

謝謝觀賞