

# 最完整的Docker聖經

---

## Docker原理圖解及全環境安裝

joshhu

Published  
with GitBook



# Table of Contents

---

1. 本書簡介
2. Docker的故事及原理
  - i. Docker的前世今生
    - i. 從dotCloud平台管理到Docker
    - ii. 旨在建立一個「更輕量化的Container」
  - ii. 一定要了解的Docker原理
    - i. Docker的最大特色
    - ii. Docker的元件— Linux核心部分
    - iii. Docker的元件— Docker核心部分
    - iv. 這些元件如何合作建立Docker環境
    - v. 看個實例
3. 全環境Docker的完整安裝
  - i. 安裝前說明
  - ii. 在Mac及Windows下安裝Docker
    - i. boot2docker簡介
    - ii. Mac OS下的boot2docker
    - iii. 在Windows下使用boot2docker
    - iv. 使用標準VM安裝
  - iii. 在Ubuntu Linux下安裝Docker
    - i. 使用本書所附的VM
    - ii. 手動安裝Docker
    - iii. 讓Docker更好用的工具
  - iv. 使用雲端專用CoreOS
    - i. CoreOS簡介
    - ii. 建立自動登入的CoreOS
    - iii. 更換CoreOS不安全的ssh key
    - iv. 生產環境vSphere CoreOS
    - v. 在公有雲端平台部署Docker

# 這本書在寫什麼？

---

為什麼Docker這麼紅？在2014年VMware舉辦的VMWorld上，竟然大家都不管VMware在搞什麼鬼，反而都在討論Docker！這個號稱取代傳統VM架構的輕量化容器，是怎麼操作的呢？

另外在Windows、Mac、Linux、VM，甚至是雲端平台上，要怎麼安裝Docker呢？如何做部署叢集Docker(Docker Cluster)的準備呢？這些重點都在本書中詳述。

- Docker的由來
- Docker的操作原理
- 在Windows/Mac OS下安裝Docker
- 在Linux下安裝Docker
- 使用本書Ubuntu的VM中的Docker
- 完整使用CoreOS VM中的Docker
- 一些好用工具
- 安裝時常見的問題解決



如果您有任何指教和建議，非常歡迎和我聯絡

[Follow @joshhu](#)

## 本章重點

---

在了解最接近VM的Container LXC之後，本章將介紹慢慢遠離系統底層(硬體、OS及VM)，而越來越接近應用程式的最新一代Container：Docker。

- Docker的創始公司及由來
- 容器就是輕量的VM
- Docker是輕量的容器
- Docker有哪些元件
- 這些元件如何一起工作

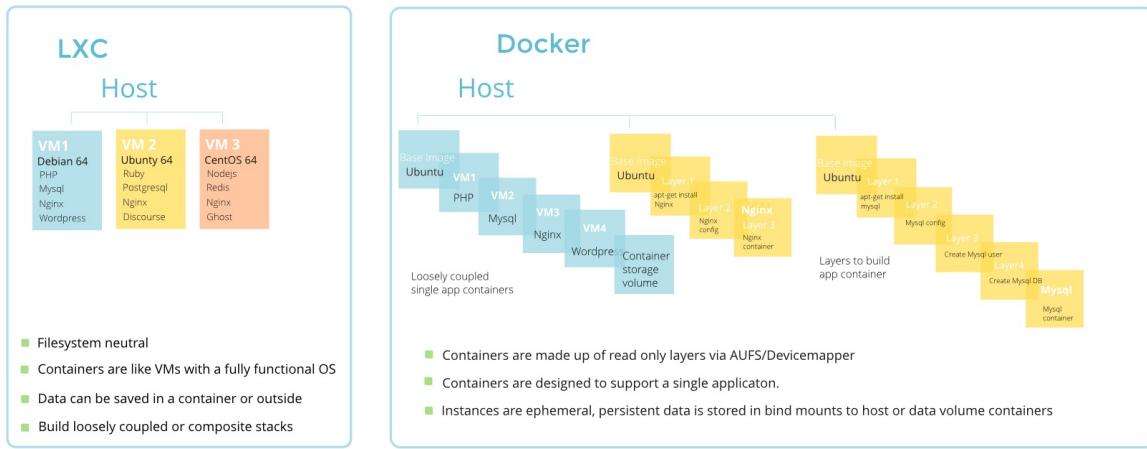
# Docker的前世今生

和Container的前輩LXC比較起來，Docker已擺脫LXC類似VM的笨重及「拖油瓶」的低效檔案系統，而朝著應用程式的輕量，以及彈性分層的檔案結構趨近。Docker在2013年出現之後，立即在業界引起轟動，到底這個號稱輕量級的虛擬化產品（如前文所言，Container不太被歸類於傳統虛擬化），是怎麼來的呢？

和LXC比起來，Docker使用aufs是非常正確的選擇

flockport

## Key differences between LXC and Docker



flockport.com

# 從dotCloud平台管理到Docker

Docker是由Docker.io這個公司所創建的產品，而Docker.io的前身稱之為dotCloud。dotCloud這個公司，就是不折不扣的平台供應商。其提供了類似IDC的服務，針對全世界的公司行號提供了Web、Application、Transaction、Database等服務。隨著網路上的服務越來越複雜，dotCloud也面臨了這些服務的最佳化、使用，以及資源分配的重大挑戰。

從前叫dotCloud，現在這個網址還在

The screenshot shows the homepage of dotCloud.com. At the top, there's a navigation bar with links for Log in, Blog, Docs, Status & Help, Sign up, and Log in. The main heading is "Introducing the next dotCloud" followed by the subtext "Take the new and improved dotCloud PaaS on Google Compute Engine for a test-drive." Below this are two buttons: "Sign up for a new account" and "Or sign in with your existing account". To the right, there's a green sidebar with the heading "Developer Cloud Platform" and the subtext "Build, deploy and scale apps – not infrastructure.", along with a "Sign up for free" button. The main content area below features a testimonial from JR Patten, Founder of Rafflecopter, and a "Learn more" link. At the bottom, there's a section titled "Build Your Ideal Stack" with icons for PHP, Node.js, Python, Ruby, and Java.

但在經營一段時間後，dotCloud針對其提供的服務進行深入的研究，赫然發現以VM為最小單位的服務隔離方式並不符合成本效益及客戶的需求。DotCloud本身當然擁有很強大的工程師們，在經過了極深層的剖析後，發現了幾件事：

## IDC業者在提供服務時的要點

- 以Linux平台為主
- 大部分的平台服務的使用均非極端狀況
- VM的粒度太大，造成資源浪費
- VM的Hypervisor必須模擬硬體，無法使用原生硬體的效能
- DevOp人員無法避免直接接觸Sysadmin的工作
- 客戶要求快速的啟動時間
- 客戶要求更彈性的服務組合
- 客戶要求更快速的服務昇級/更新時間

為了滿足這些條件，dotCloud的工程師們即開始研究出一個以Linux為主，不使用Hypervisor，但又能讓其上服務能獨立執行的環境。在2013年時，能滿足這些條件的技術最適合的Container，然而當時較流行的Container技術只有LXC，因此dotCloud的工程師，就以「包裝」一個「較好用」的LXC產品為主，用來滿足自己公司的需求，因此Docker，就在此背景下誕生了。

現在改名叫Docker，名氣比dotCloud大多了

 docker

What is Docker? Use Cases Try It! Install & Docs Browse Login

# Build, Ship and Run Any App, Anywhere

Docker - An open platform for distributed applications for developers and sysadmins.



Get started with Docker

- Build better apps
- Deploy apps faster
- Get the latest news

[Sign up for free](#) [Learn More](#)

## New: Docker Orchestration Tools

Assemble multi-container apps, run on any infrastructure.

# 旨在建立一個「更輕量化的Container」

Docker一開始就是dotCloud的內部專屬專案，建立在其自行開發的「Cloudlets」，專案之上。Docker的開發者是Solomon Hykes、Andrea Luzzardi、Francois-Xavier Bourlet、Jeff Lindsay等人。

dotCloud本身就是一個PaaS的業者，使用的客戶當然也以DevOps為主，他們也針對了PaaS客戶的需求來設計Docker，也達成了下面的目的：

- 不需要用到Hypervisor - 直接利用硬體效能
- Linux核心現有功能 - 保持不同distro的Linux相容性
- 放棄大量VM的sysadmin功能，如HA、FT等，這些工作交由系統層級實作
- 保留VM的操作概念，如CPU、記憶體分配、獨立的網路拓樸 – 符合目前操作習慣
- 拋棄VM檔案系統的概念，使用層次化的檔案系統 – 維持Container的輕便
- 使用主從架構，有Docker Daemon及Docker Client – 便於自訂開發
- 類似應用程式的執行方式 – 用「參數」代表不同的設定值

在內部使用了一陣之後，dotCloud的工程師們發現，這個專案如果只用在公司內部，實在是太可惜了，因此在2013年的7月23日，他們將原來Gluxter & Plaxo的CEO Ben Golub請來，並且進行商業化的動作，並且也讓Docker成為Open Source的專案，當然dotCloud在同年也改名為Docker Inc，因為在當時，Docker的名字響亮程度已經遠遠超過dotCloud了！

有關Docker的發展歷史以及版本演進，有興趣的讀者可以參考[維基百科上的Docker條目](#)。一台實體伺服器可部署的Docker Container數量成百上千(相對於VM的數十個)，目前也Docker已經被全世界所有的IDC所使用，在取代了傳統VM的架構之後，VM世界及Container世界之間的微妙變化將會如何發展，我們繼續看下去！

有興趣的讀者可以到維基百科上瞧瞧。



The screenshot shows the Wikipedia article for "Docker (software)". The page title is "Docker (software)" and it is categorized under "Article". The main content discusses Docker as an open-source project for automating application deployment. A sidebar on the right provides detailed information about Docker, including its original author (Solomon Hykes), developer (Docker, Inc.), initial release (March 13, 2013), stable release (version 1.5.0 on February 3, 2015), and other details like being written in Go, using Linux as the operating system, and running on x86-64 with modern Linux kernel. The page also includes a "Contents" sidebar and a "See also" section.

内幕消息 – 昔日親密戰友，今日反目成仇

在Docker剛出來時，獲得許多Linux高手的推崇，其中最大的支援者之一，就是叢集Linux CoreOS的創辦人Alex Polvi。然而

隨著Docker的商業化，Alex Polvi和Docker也漸行漸遠。Polvi甚至在和Gigaom的訪問上直指Docker的創辦人Solomon Hykes背棄原來Container的理念，並且Polvi的CoreOS也推出了自己Container Rocket。當然Solomon Hykes也不甘示弱在Twitter上回擊。兩人之間精彩的交鋒，可以參考

<https://gigaom.com/2014/12/02/why-coreos-just-fired-a-rocket-at-docker/>

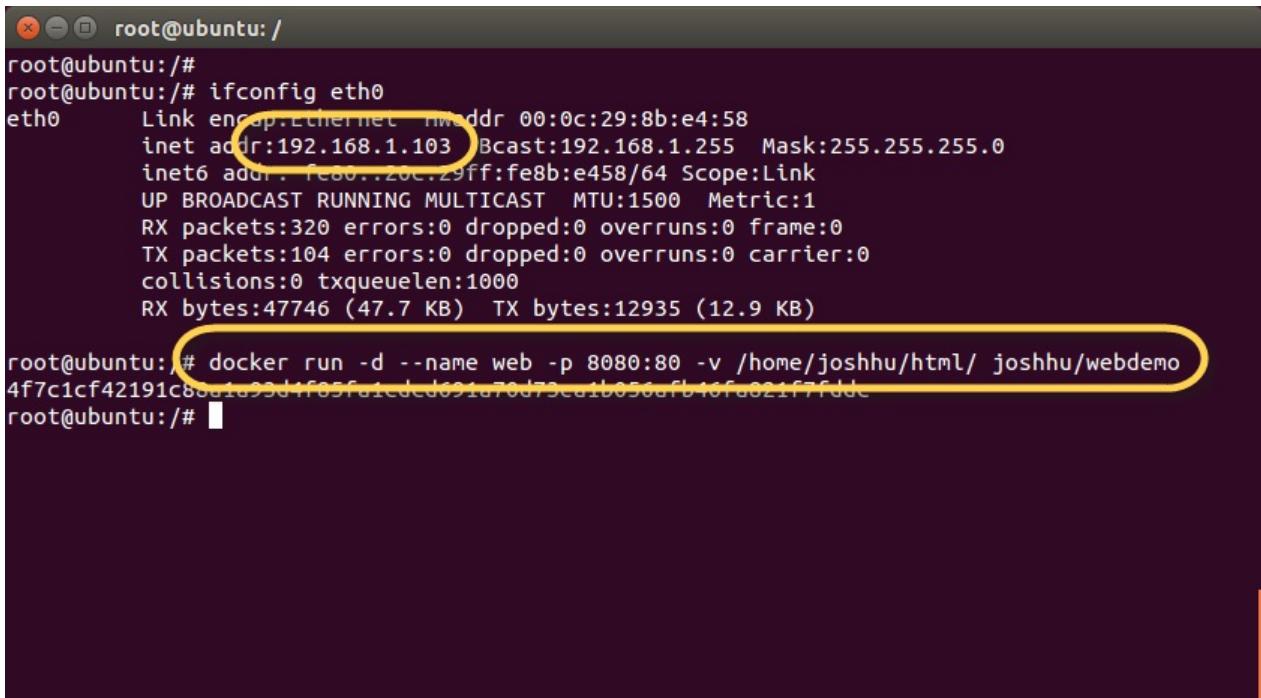
<http://www.ithome.com.tw/news/92769/>

<https://twitter.com/DShankar/timelines/539528446473289728/>

# 一定要了解Docker的原理

我們先來看一行標準的Docker指令：

```
docker run -d --name web -p 8080:80 -v /home/joshu/html:/html joshhu/webdemo
```



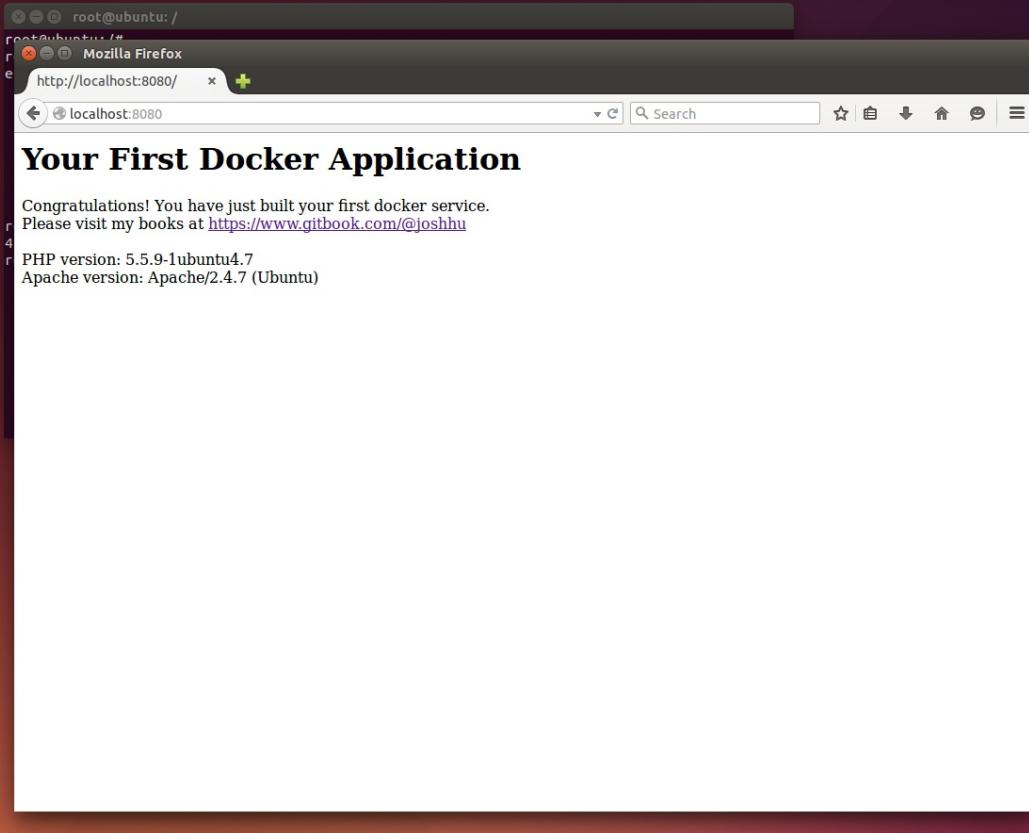
The screenshot shows a terminal window titled 'root@ubuntu: /'. The user runs the command 'ifconfig eth0' which outputs details about the network interface. Then, the user runs 'docker run -d --name web -p 8080:80 -v /home/joshu/html:/html joshhu/webdemo'. The entire command line is highlighted with a yellow oval.

```
root@ubuntu:/# ifconfig eth0
eth0      Link encap:Ethernet HWaddr 00:0c:29:8b:e4:58
          inet addr:192.168.1.103 Bcast:192.168.1.255 Mask:255.255.255.0
              inet6 addr: fe80::20c:29ff:fe8b:e458/64 Scope:Link
                  UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
                  RX packets:320 errors:0 dropped:0 overruns:0 frame:0
                  TX packets:104 errors:0 dropped:0 overruns:0 carrier:0
                  collisions:0 txqueuelen:1000
                  RX bytes:47746 (47.7 KB) TX bytes:12935 (12.9 KB)

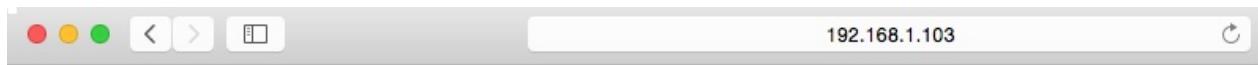
root@ubuntu:/# docker run -d --name web -p 8080:80 -v /home/joshu/html/ joshhu/webdemo
4f7c1cf42191c88a1a93d4f85f81cdcd691a70d73ca1b05cafb46fa821f7fdde
root@ubuntu:/#
```

就是這麼一行指令，就可以立即建立一個支援Apache/php的網頁服務，這個網頁的檔案存放在本機的/home/joshu/html目錄下，並且使用主機的8080埠，有夠簡單吧！

從本機看



從網路上其它電腦上看



## Your First Docker Application

Congratulations! You have just built your first docker service.  
Please visit my books at <https://www.gitbook.com/@joshhu>

PHP version: 5.5.9-1ubuntu4.7  
Apache version: Apache/2.4.7 (Ubuntu)

只要打入短短幾個英文，就可以立即建立一個類似VM的執行環境，稱之為Container。

Docker的Container充份利用了現有Linux核心功能，成功模擬出一個類似VM的概念，但和真正需要硬體、軟體、作業系統、應用程式程式配合的虛擬機Hypervisor比較起來，Container反而更輕量，更彈性。但Docker比起一般的Container來說，又更小更快速了！在這一小節我們就來看看Docker的原理。

# Docker的最大特色

---

首先是Docker的特色，這邊對這些技術名詞不了解沒關係，在接下來的章節中，會針對這些特色有更多的說明。

- 是一個**OS**等級虛擬化的產品
- 使用**Go**語言開發。
- 實作了**Linux Kernel**功能。
- Modules的功能來完成其「虛擬化」的「長相」。
- 在呼叫Linux的核心提供的虛擬化模組時，0.9版前還需依賴LXC、libvirt及systemd-nspawn的功能，0.9版之後，就預設使用自己開發的**libcontainer**來呼叫(也可向前相容)。
- 100%原生硬體的效能。
- VM等級的隔離及資源分配。
- 應用程式等級的輕量及方便。
- 由唯讀的多層次映像檔做為模板(用來產生Container)。
- 產生Container後，保持最上層可寫入(用來提供服務)。
- 服務完全可攜，可以讓Docker在不同的VM/機器上跑來跑去

在擁有了最後三點特色之後，Docker完全擺脫LXC的影子，讓Container再度進化，變成又小又輕又可攜，一下子就成為DevOps最愛的好物！

## 注意 - LXC和最大的區別是

相對於LXC，Docker是更輕量的Container，LXC較像**VM**，Docker較像應用程式。針對同一個應用，LXC仍需要建立Container、下載作業系統、安裝平台軟體、安裝應用程式。但對Docker來說，一行帶參數的指令就完成上述所有的動作。我們會在Docker的使用介紹章節中，套用LXC章節中完全一樣的訊息交流平台 Etherpad 為例子。

# Docker的元件 – Linux核心部分

---

一個能執行Docker的系統分為兩大部分：

- Linux的核心元件
- Docker相關元件

Docker使用的Linux核心模組功能包括下列各項：

- Namespace – 用來隔離不同Container的執行空間
- Cgroup – 用來分配硬體資源
- AUFS(chroot) – 用來建立不同Container的檔案系統
- SELinux – 用來確保Container的網路的安全
- Netlink – 用來讓不同Container之間的行程進行溝通
- Netfilter – 建立Container埠為基礎的網路防火牆封包過濾
- AppArmor – 保護Container的網路及執行安全
- Linux Bridge – 讓不同Container或不同主機上的Container能溝通

我們會在本書的其它章節中會完整實作上面的所有功能。

## 注意 – 不同版本Linux的核心編號

本書全書使用Ubuntu及CoreOS兩種Linux，是完全支援Docker所需要的Linux核心，如果你使用的是其它版本的Linux，請參考第三章 中有關Linux核心對Docker支援版本編號。

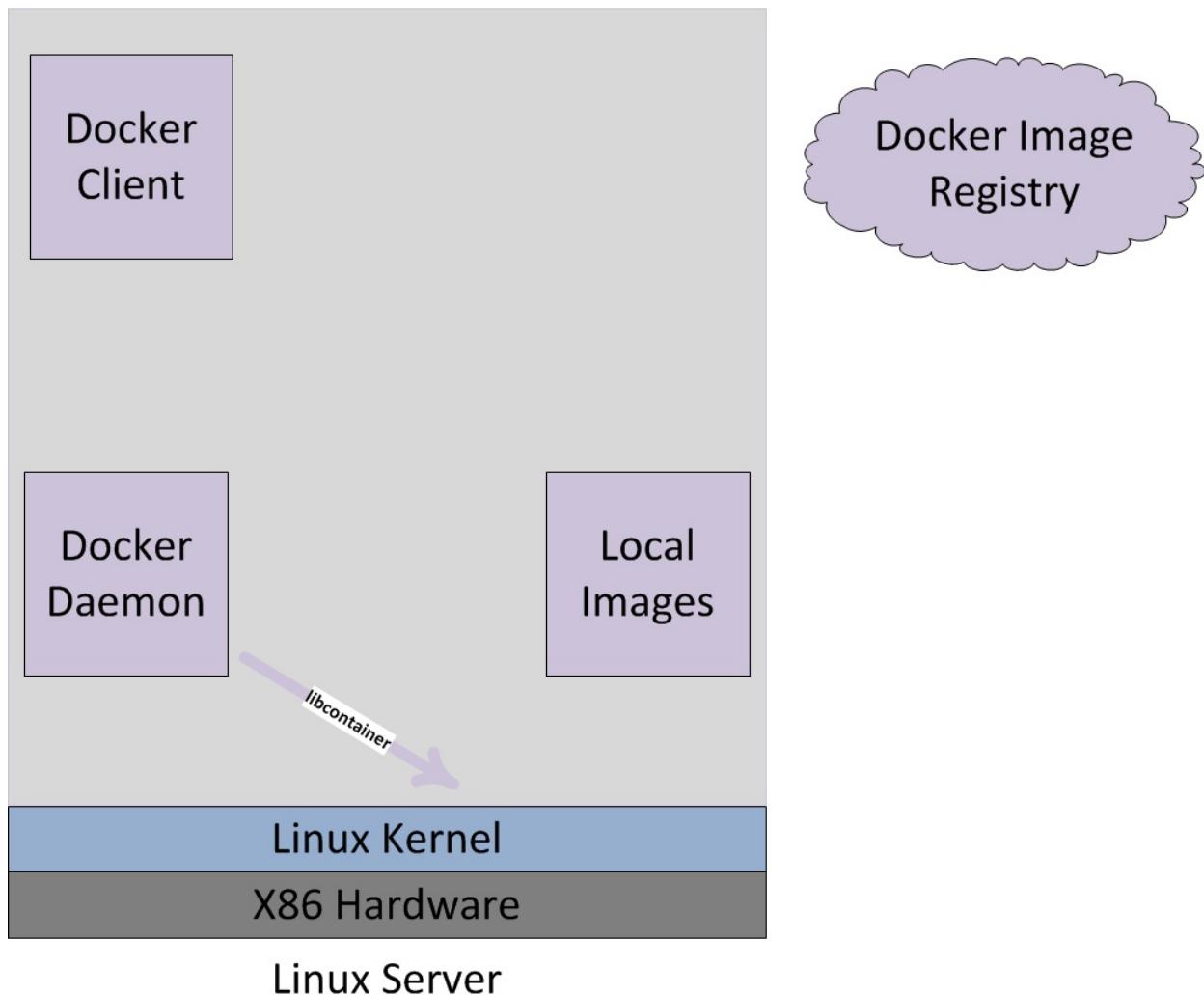
# Docker的元件 – Docker核心部分

系統要能執行Docker，除了前述的Linux核心元件外，就是Docker部分了。Docker必要條件分為本機的三個元件之外，真的靈魂其實是雲端上的映像檔資料庫。本書稍後會有映像檔資料庫的完整介紹

## Docker的執行元件

- Docker client – 呼叫Docker Daemon (本機或其它客戶端)
- Docker daemon – 執行Docker功能並用 (本機)
- libcontainer - 和Linux核心溝通的library (本機)
- Docker Image – 建立容器用的映像檔 (本機或雲端映像庫)

圖中紫色部分為Docker的核心元件



通常在安裝Docker時，就是同時安裝Docker客戶端、Docker daemon以及libcontainer，此時客戶端和daemon是在同一台電腦上的(使用unix:///var/unix.sock呼叫)。當然也可以讓Docker的客戶端和daemon不在同一台電腦(使用，但會有安全上的疑慮。此外你也可以利用Docker提供的API來撰寫自己的客戶端，預設的Docker客戶端介面就是我們安裝Docker時的輸入指令。

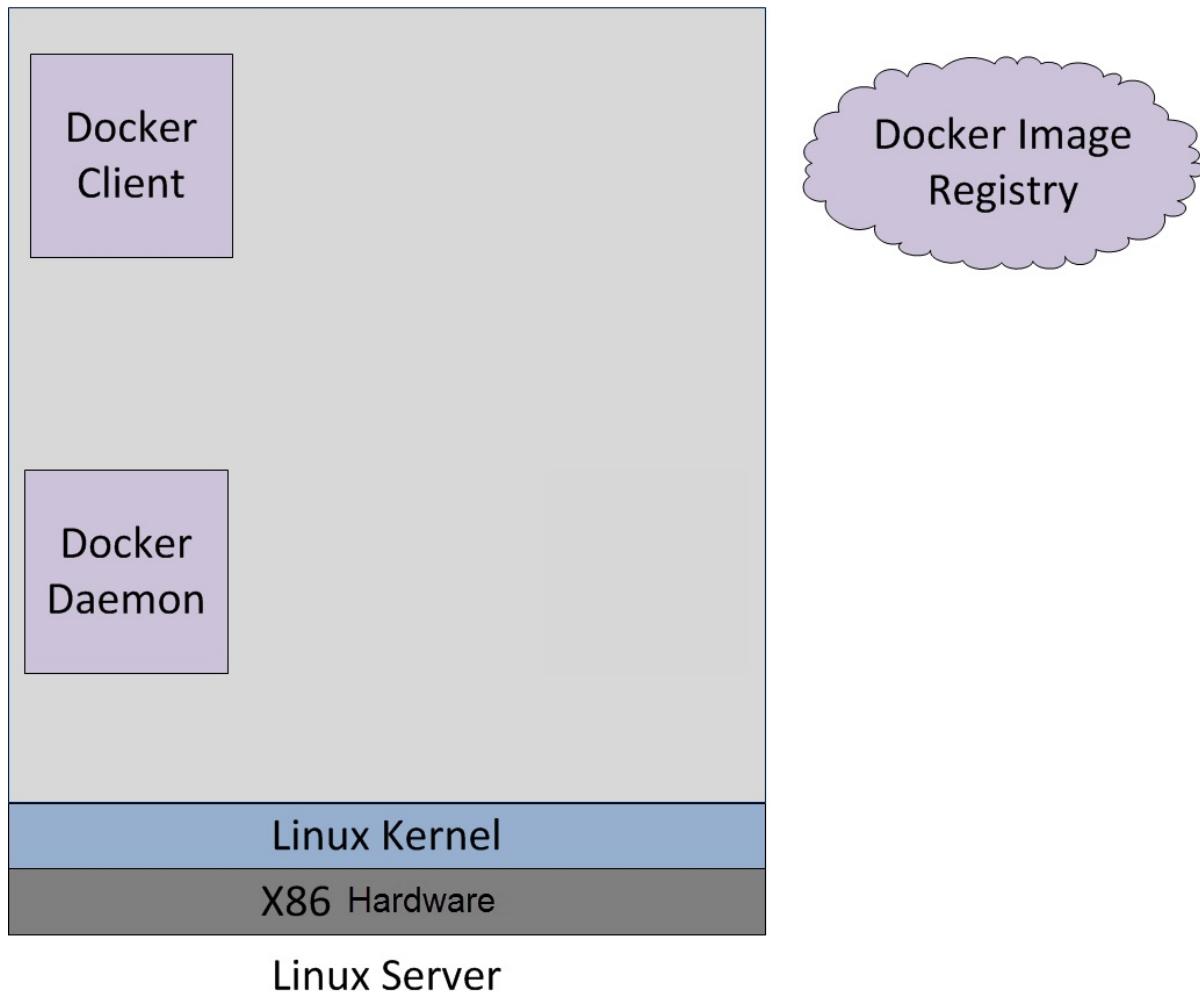
# 這些元件如何合作建立Docker環境

Docker的執行非常簡單，就是docker client呼叫Docker daemon，然後daemon透過 `libcontainer` 呼叫Linux的核心模組來完成建立容器的步驟，下面看圖說故事。

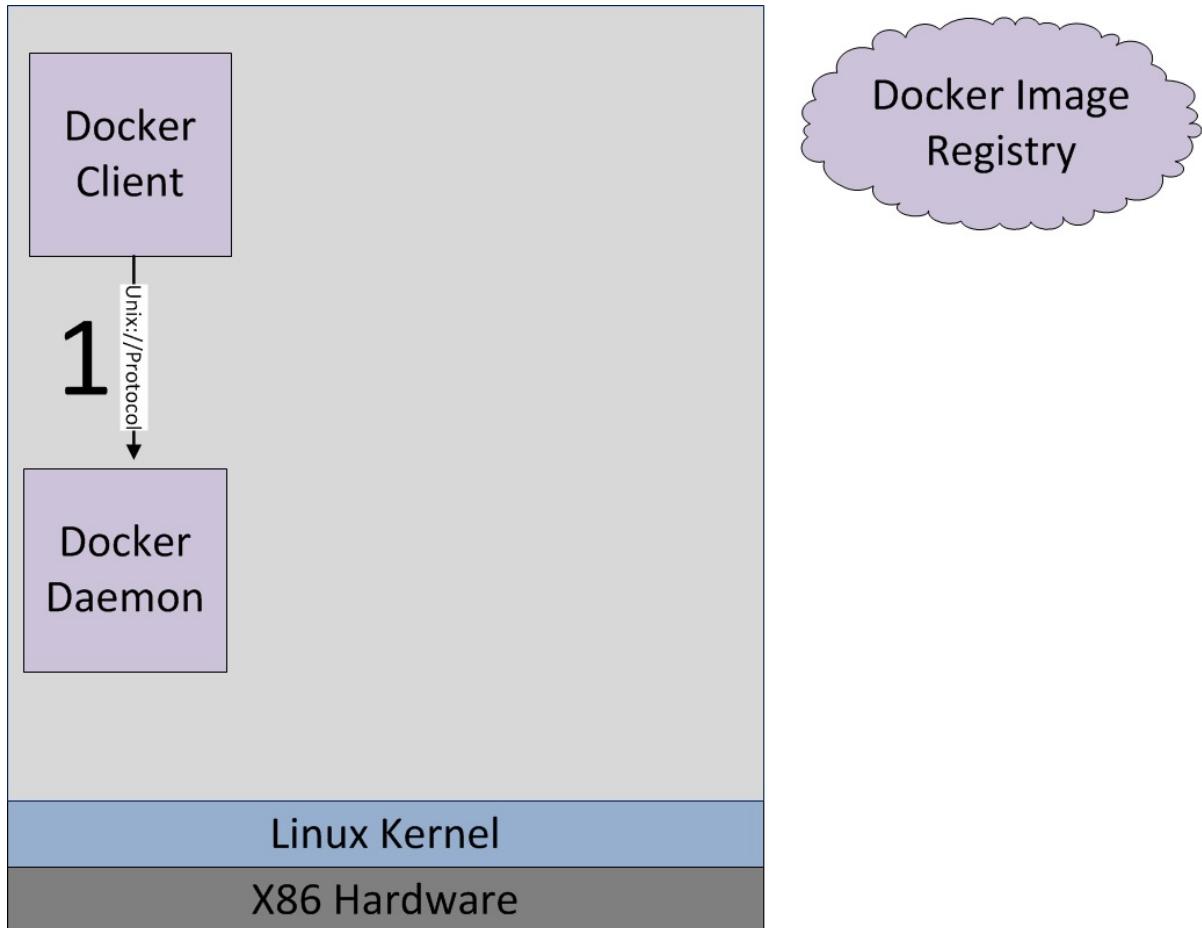
## Docker執行過程

- 首先當然要有安裝了Docker的Linux及硬體。

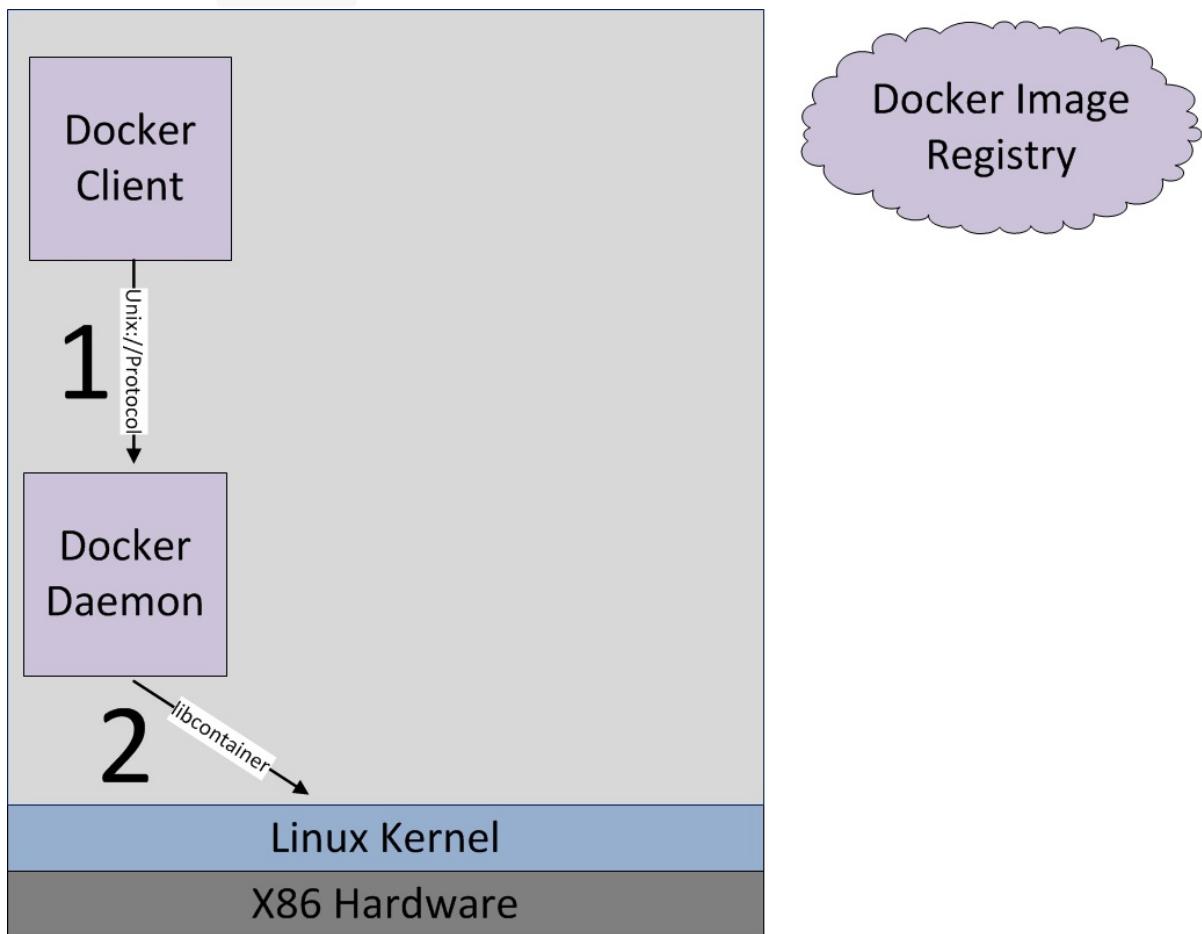
x86硬體，版本正確的Linux以及Docker



- 當準備建立Container時，輸入指令即執行docker client。呼叫Docker daemon，預設使用的通訊協定是 `unix:///var/run/docker.sock`。

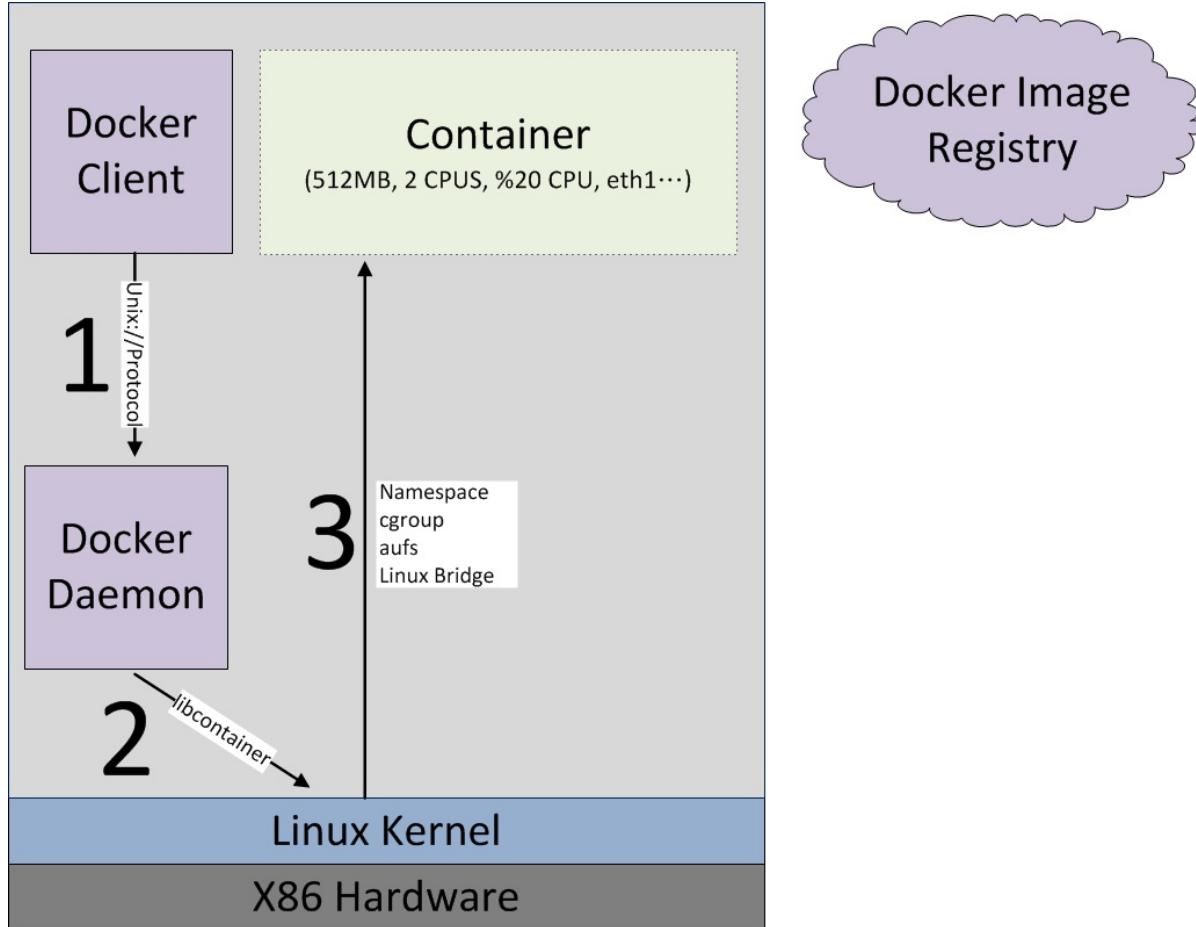


3. Docker daemon透過 `libcontainer` 要求Linux核心建立Container。

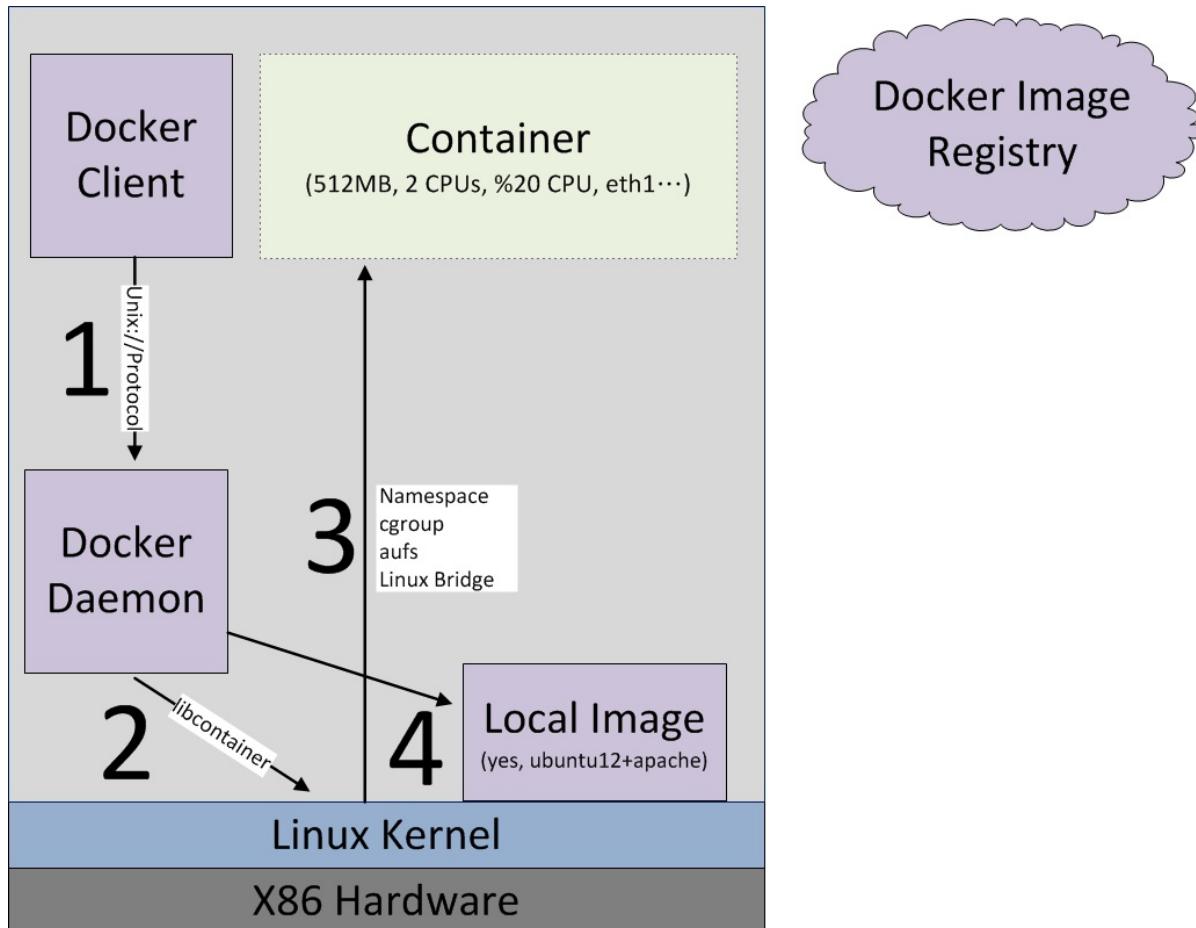


4. 此時Linux核心收到指示，即啟動核心的 namespace 建立一個獨立的空間，包括 pid、network、IPC、UTS、mount 等 namespace，daemon根據client的參數定義來分配CPU、記憶體或磁碟IO等。注意虛線，此時**Container**中沒有任何東

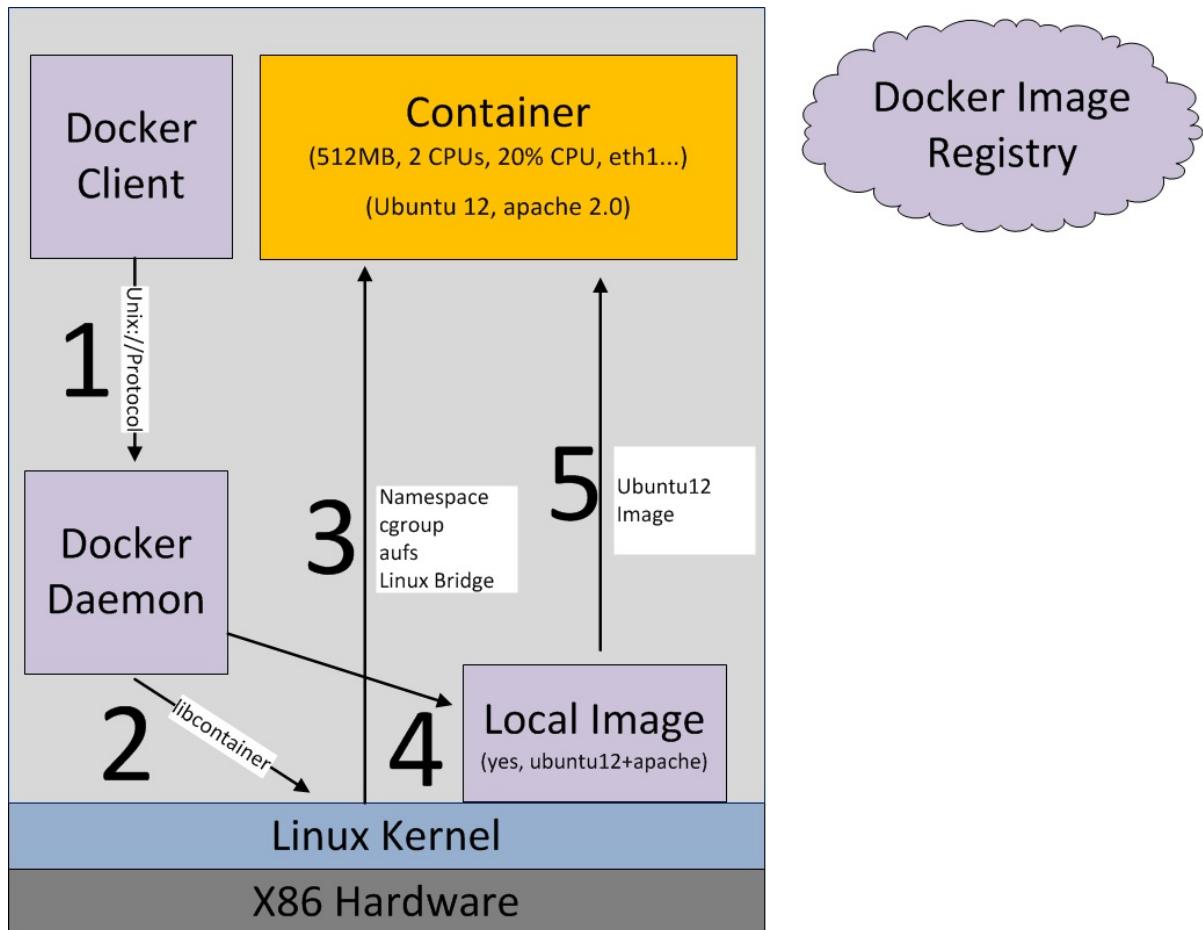
西！只有空殼沒有內容。Container的空殼建立完成，需要將真正的作業系統及應用程式放入這個空殼中。



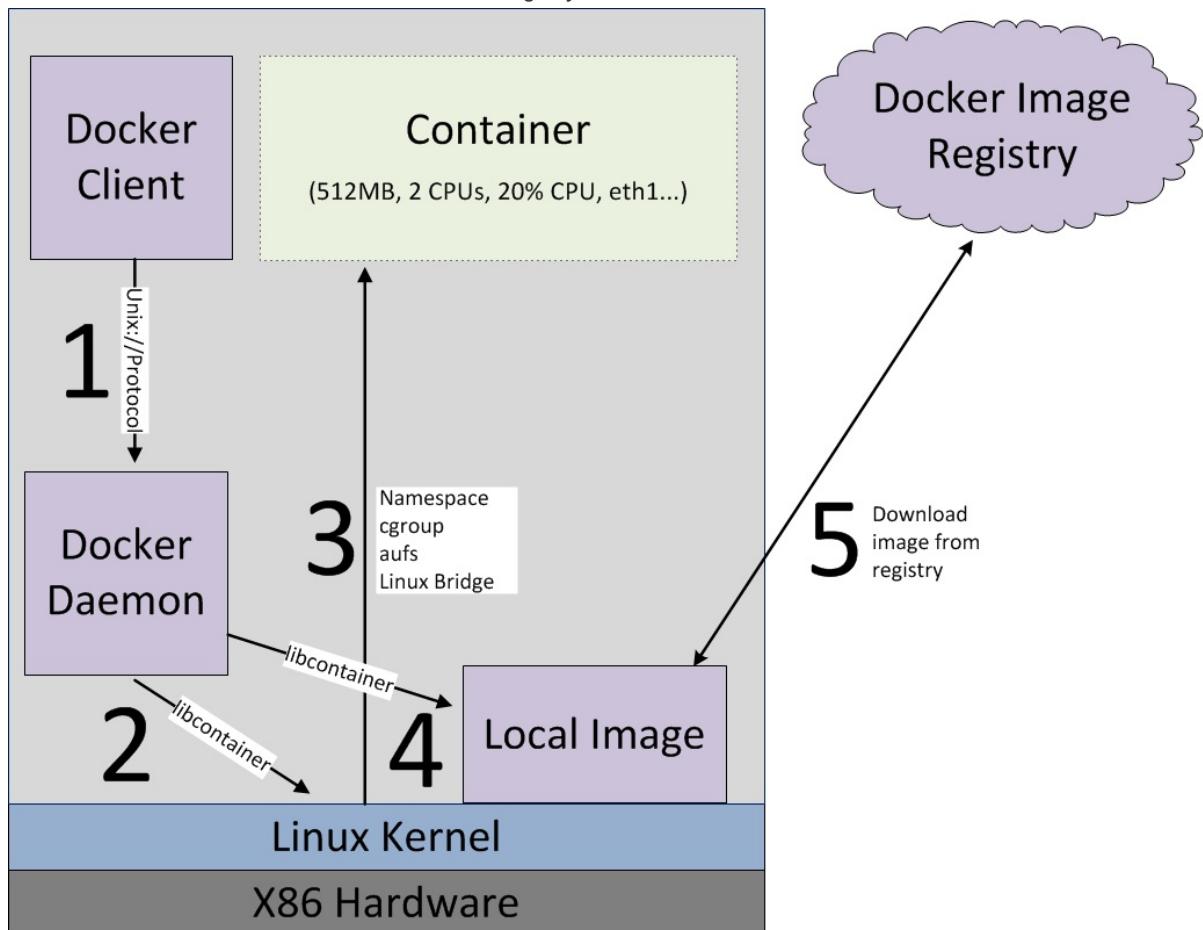
5. Damone檢查本機的現有映像檔列表，看要填入此Container空殼的映像檔之前有沒有下載過，



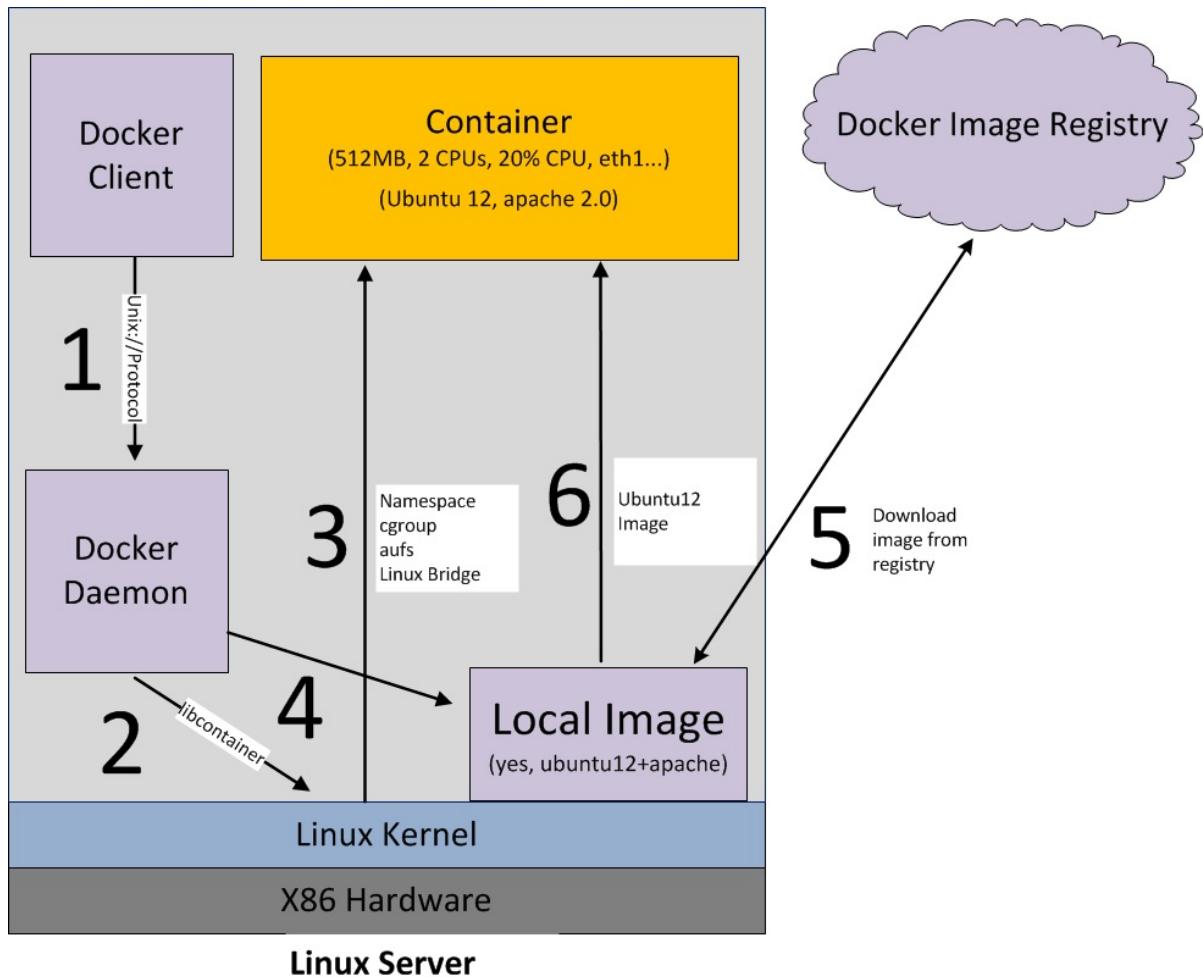
6. 有的話，直接從本機載入Container中，此時Container建立完成並啟動。



7. 若本機沒有此映像檔, daemon到預設的Docker Registry, 根據client的參數, 下載適當的映像檔。

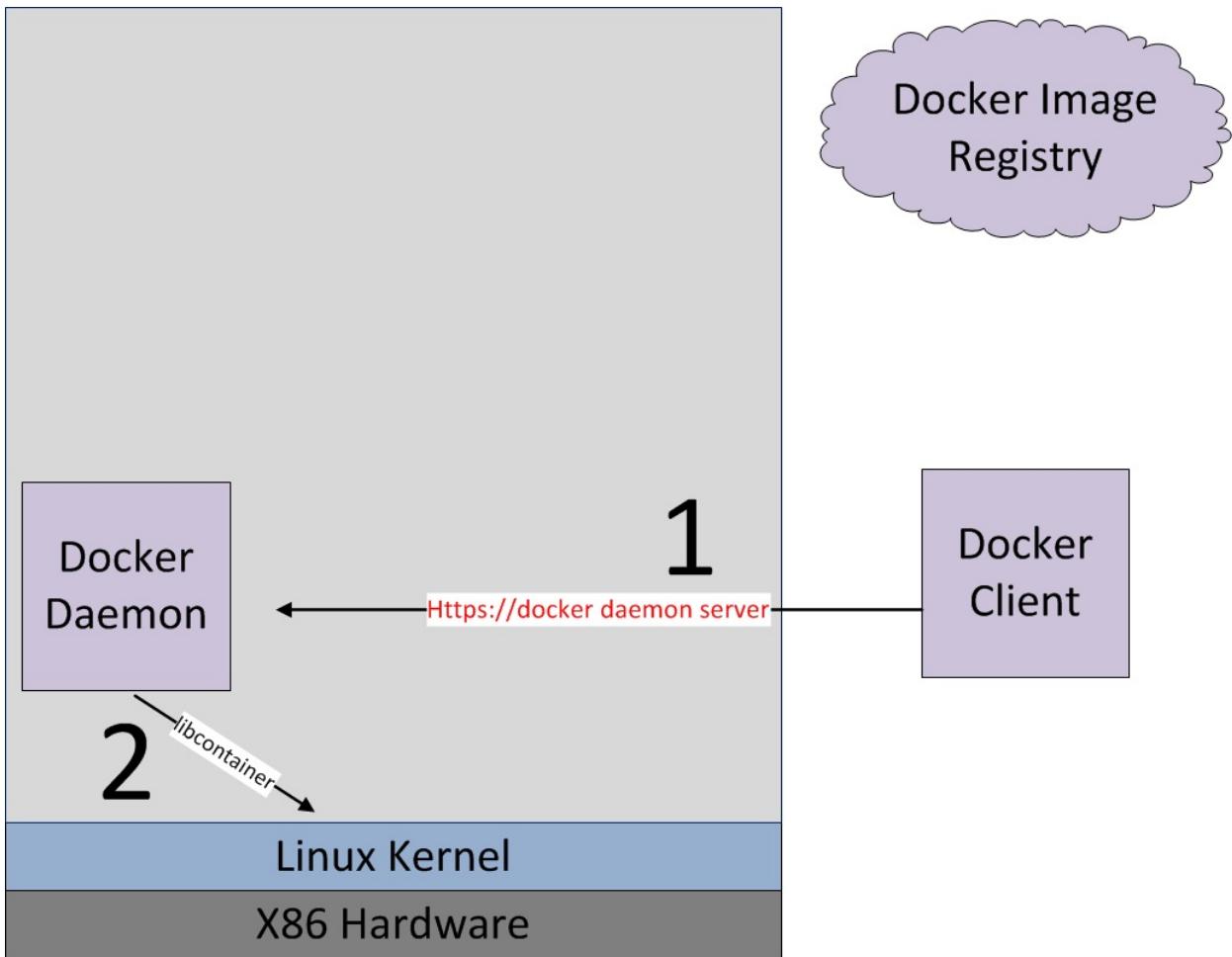


8. 下載回來即將此映像檔, 填入Container的空殼, 此時Container即啟動完成。



從上面的流程可以得知，Docker Container執行的調整方式就是我們下的參數。事實上，執行Docker時的參數，就是決定Linux核心建立Namespace以及設定網路、儲存的方法，這在本書稍後會有詳細說明。

注意 - Docker client可以在另一台主機此時和 Daemon的之間的連線可以用https取代unix://，但十分危險，除非你很確定安全性沒問題才要這麼做！



# 看個實例

我們就來看一個實際的例子來說明上一小節的Docker運行原理。

- 先確定這個Linux環境是可以執行Docker的。包括安裝了Docker，Linux的版本正確，以及有連上公網。

```
root@ubuntu:/# 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
root@ubuntu:/# lsb_release -r
Release: 14.10
root@ubuntu:/# ifconfig eth0
eth0      Link encap:Ethernet HWaddr 00:0c:29:8b:e4:58
          inet addr:192.168.1.103 Bcast:192.168.1.255 Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe8b:e458/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
          RX packets:10422 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3629 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:2519934 (2.5 MB) TX bytes:409113 (409.1 KB)

root@ubuntu:/#
```

- 輸入Docker的指令，即準備呼叫docker daemon：`docker run -d --name web -m 512m -p 8080:80 joshhu/webdemo`。這邊要注意的是，Docker的選項，全部以參數方式表現，如記憶體限制，名稱，通訊埠對應，映像檔名稱等。

```
root@ubuntu:/#
root@ubuntu:/#
root@ubuntu:/# docker run -d --name web -m 512m -p 8080:80 joshhu/webdemo
```

Diagram illustrating the Docker command components:

- docker client command**: Points to the first part of the command (`docker run`).
- name of the container**: Points to the `--name web` option.
- setting system resource(512M memory)**: Points to the `-m 512m` option.
- the image to fill up container**: Points to the `joshhu/webdemo` image name.

- 按下Enter之後，即使用 `unix://var/run/docker.sock` 呼叫docker daemon。由於 `joshhu/webdemo` 這個映像檔已經存在了，因此就直接用此映像檔，填入名為 `web` 的Container中。

```
root@ubuntu:/#
root@ubuntu:/# docker images
REPOSITORY          TAG        IMAGE ID       CREATED             VIRTUAL SIZE
joshhu/webdemo     latest     a3574f323972   42 hours ago    243.2 MB
6fcfc8e566e59f1f2e1eacb8f76c4183b63d06ac5216e4588394578b1aee8196c
```

Annotations for the Docker command output:

- show local images**: Points to the `docker images` command.
- image already exists**: Points to the `joshhu/webdemo` entry in the image list.
- container id**: Points to the `a3574f323972` ID of the existing container.
- image name**: Points to the `joshhu/webdemo` image name.

- 可以輸入 `showmem` 看一下記憶體的使用情況。

```

root@ubuntu:~# docker run -d --name web -m 512m -p 8080:80 joshhu/webdemo
a6570c9af94901f180f50fbff4c192bab93117ec448b45924e6cbd8afe2237c0
root@ubuntu:~#
root@ubuntu:~# showmem
web 12MB 512MB
root@ubuntu:~#

```

Memory used, allowed

5. 如果這個 `joshhu/webdemo` 影像檔沒有在本機，就會先去下載，下載回來後，再填入空的Container `web` 中。

REPOSITORY	TAG	IMAGE ID	CREATED
VIRTUAL SIZE			
root@ubuntu:~# docker run -d --name web -m 512m -p 8080:80 joshhu/webdemo			
Unable to find image 'joshhu/webdemo:latest' locally			
Pulling repository joshhu/webdemo			
a3574f323972: Pulling image (latest) from joshhu/webdemo, endpoint: https://regi			
a3574f323972: Download complete			
511136ea3c5a: Download complete			
f3c84ac3a053: Download complete			
a1a958a24818: Download complete			
9fec74352904: Download complete			
d0955f21bf24: Download complete			
1214be61bcaa: Download complete			
45ad00454734: Download complete			
8be3dadcb43a: Download complete			
2d4173730925: Download complete			
e8f612e3238f: Download complete			
d11f03cd837d: Download complete			
8abfc77aac61: Download complete			
d1dfd80f5ed9: Download complete			
Status: Downloaded newer image for joshhu/webdemo:latest			
9738c5c00c07ef0534267e657e609460bf360147bf73970799c374e28bcb7441			

#### 注意 - Container的ID及名稱

在Docker執行時，如果你沒有使用 `--name <名稱>` 的參數，Docker會主動幫這個Container取一個好玩的名稱。而不管你有沒有幫這個Container命名，Docker一定會產生一個全世界獨一無二的Container id。

```

root@ubuntu:~#
root@ubuntu:~# docker run -d joshhu/webdemo No name assigned
root@ubuntu:~# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS
69a0c2851abd       joshhu/webdemo:latest   "/start.sh"        3 seconds ago      Up 2 seconds
          80/tcp
root@ubuntu:~# System generated name

```

## 本章重點

---

Docker是一個原生的Linux產物，只要安裝了正確版本的Linux，就可以直接使用Docker。但大部分的我們不太可能生活在純Linux的環境，事實上，大部分的電腦使用及操作都是在Windows或Mac OS之下，因此我們在這一章，就來看看最完整的Docker執行環境及操作環境的架設。

- 最常使用的工具
- 最常使用的虛擬機軟體
- Mac及Windows下安裝及簡單使用boot2docker(不推薦)
- 在標準VM中安裝Docker
- 在Ubuntu下安裝Docker
- 使用現成安裝好Docker的VM
- 雲端專用Linux CoreOS的VM安裝
- CoreOS安裝後的校調

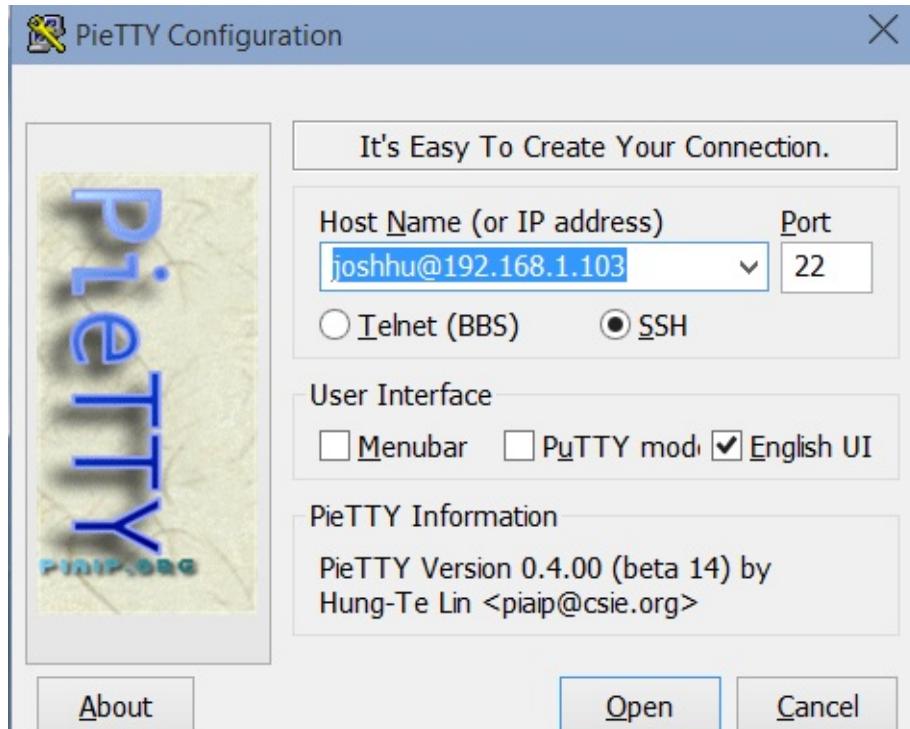
# 安裝前說明

Docker支援絕大部分的Linux，只要該Linux的核心版本編號夠新。另外在Mac OS及Windows下則無法直接支援Docker，必須使用特殊方法，或是使用VM，我們就來看看。

## 注意：本書使用環境說明

本書操作主要使用原生的Linux終端視窗，有關圖型環境操作，則主要使用Mac OS及Windows 10。在Windows/Mac的虛擬機方面，本書使用VirtualBox及VMware Workstation/Fusion，讀者只要有上述的作業系統及虛擬化環境的任一組合即可。

也可以使用Windows下的SSH工具，PieTTY最好用



[VirtualBox下載](#)

[VMware Workstation下載](#)

[VMware Fusion下載](#)

# 在Mac及Windows下安裝Docker

---

Docker所產生出來的Container只能是Linux，因此Docker只能運行/安裝在純Linux環境下，Docker支援絕大部分的Linux，只要該Linux的核心版本編號夠新。

為了方便起見，本書一律使用Docker指定的Ubuntu 14.04以後的Linux，以及CoreOS，這兩個是原生就支援Docker的Linux版本。

Mac OS及Windows下則無法直接原生使用Docker，必須使用Linux的VM才行。使用VM的方法分為兩種，一種是直接自行架設VM(如VMware Workstation/Fusion、VirtualBox)，再自安裝Linux的VM，然後再在這個VM中安裝Docker。

另一個使用VM的方式，則是使用官方的 boot2docker，雖然也是用VM+Linux的方式，但會一次幫你完成上面的所有動作，並且使用類似Windows Shell指令及Mac OS的終端視窗來操作。

---

## 注意 -Docker部署在實體伺服器 vs 部署在VM中

雖然本書不斷強調原生硬體的效能遠遠超過VM的模擬硬體效能，將Docker直接安裝在實體伺服器的Linux中，一定比先安裝VM Linux，再在此VM中執行Docker要快上許多。但企業考量的重點除了效能之外，更有部署的便利性、現存系統的穩定度、整個系統的HA、FT、DRS高可用性等重點，筆者建議還是將Docker部署在VM中，犧牲一點點效能，換來的是更方便的部署、更穩定的環境以及更彈性的設定，這是絕對值得的。

---

# boot2docker簡介

---

Boot2docker是一個專門在Mac及Windows下使用Docker的套件，包括了：

- 一個VirtualBox程式
- VirtualBox格式的極小Linux VM
- 位於該VM中的Docker程式
- Boot2Docker管理工具。

除了Docker本身之外，boot2docker元件中的VirtualBox VM非常小，完全不佔磁碟空間只存在記憶體中，大小僅為24MB，開機時間只要5秒。此外，Boot2Docker管理工具本身就是一個輕量級的Linux VM，專門用來在Mac OS中執行Docker Daemon。

# Mac OS下的boot2docker

由於Mac OS是一個類UNIX的封閉系統，因此在操作上雖然很像Linux，但核心是使用蘋果自己的系統。要使用Docker也必須使用boot2docker的Mac版，操作較Windows版多幾個步驟，我們就來看看。

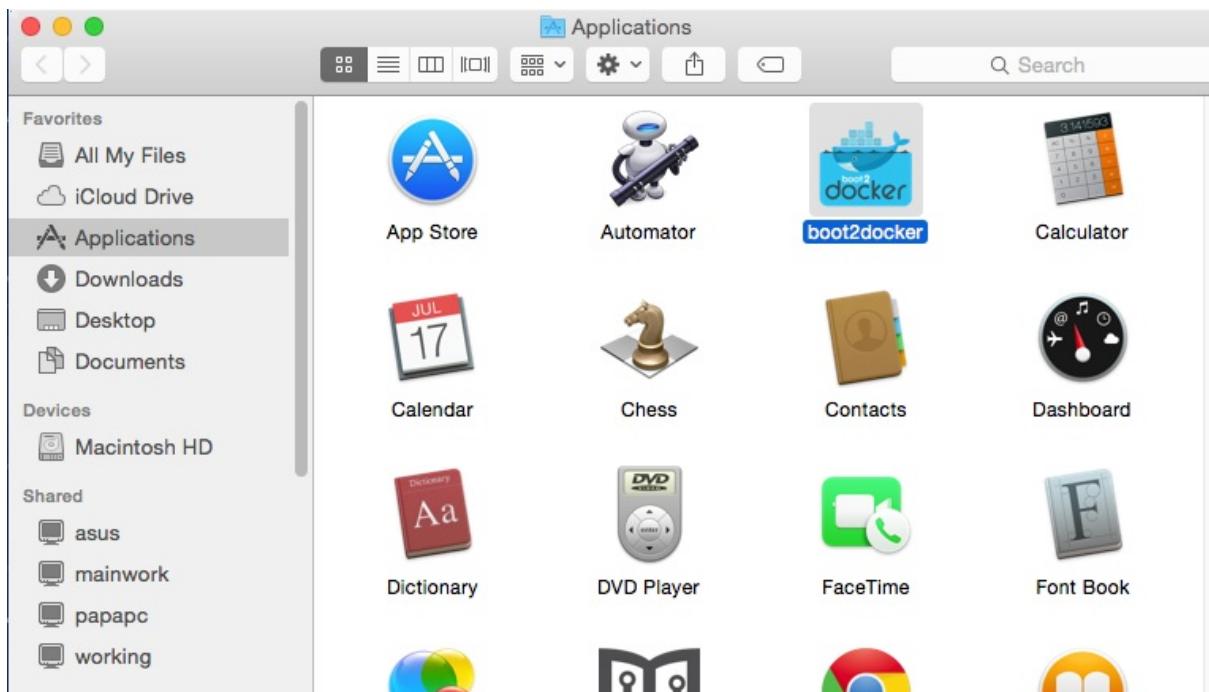
## 安裝boot2docker

Mac OS下的boot2docker安裝和Windows差不多，我們就來看看。

1. 確定Mac OS的版本在10.6以上。
2. 到<https://github.com/boot2docker/osx-installer/releases/tag/v1.5.0>下載Boot2Docker-1.5.0.pkg。
3. 下載後直接執行該檔案，此時會將Boot2Docker放到Mac OS的Applications的資料夾下，而將docker及boot2docker放到`/usr/local/bin`的資料夾下(和Linux一樣)。



4. 照著一般Mac OS的程式安裝即可(其實就是安裝VirtualBox，一直Next即可)。
5. 安裝完畢後，會在「應用程式」的資料夾出現boot2docker的圖示。



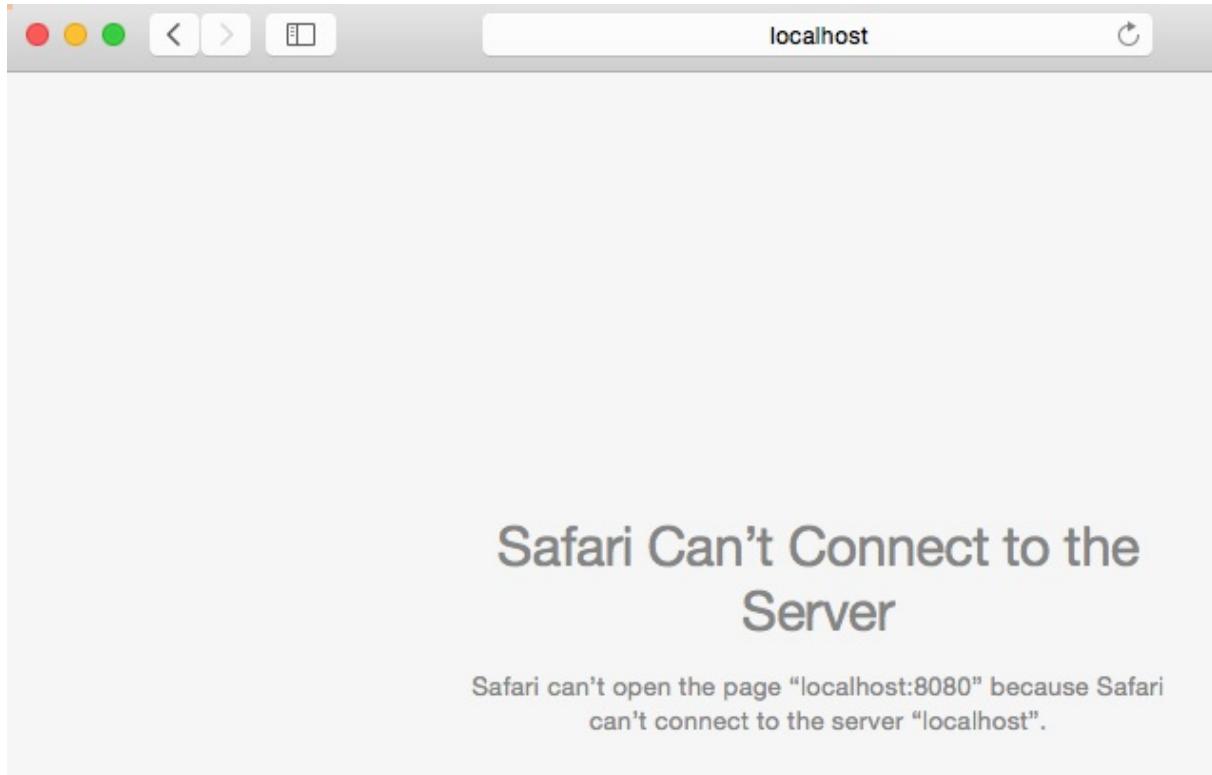
6. 直接執行，第一次會出現如圖的視窗，要等一下。

```
Last login: Mon Mar 23 13:58:19 on ttys000
Joshi-Mac:~ joshhu$ bash
bash-3.2$ unset DYLD_LIBRARY_PATH ; unset LD_LIBRARY_PATH
bash-3.2$ mkdir -p ~/.boot2docker
if [ ! -f ~/.boot2docker/boot2docker.iso ]; then cp /usr/local/share/boot2docker
/boot2docker.iso ~/.boot2docker/ ; fi
/usr/local/bin/boot2docker init
/usr/local/bin/boot2docker up
$(/usr/local/bin/boot2docker shellinit)
docker version
bash-3.2$ if [ ! -f ~/.boot2docker/boot2docker.iso ]; then cp /usr/local/share/b
oot2docker/boot2docker.iso ~/.boot2docker/ ; fi
bash-3.2$ /usr/local/bin/boot2docker init
Virtual machine boot2docker-vm already exists
bash-3.2$ /usr/local/bin/boot2docker up
Waiting for VM and Docker daemon to start...
...
```

7. 跑完之後，輸入 docker run -d --name web -p 8080:80 joshhu/webdemo，出現下圖的畫面時表示安裝成功。

```
bash-3.2$ docker run -d --name web -p 8080:80 joshhu/webdemo
Unable to find image 'joshhu/webdemo:latest' locally
Pulling repository joshhu/webdemo
a3574f323972: Pulling image (latest) from joshhu/webdemo, endpoint: https://regi
a3574f323972: Download complete
511136ea3c5a: Download complete
f3c84ac3a053: Download complete
a1a958a24818: Download complete
9fec74352904: Download complete
d0955f21bf24: Download complete
1214be61bcaa: Download complete
45ad00454734: Download complete
8be3dadcb43a: Download complete
2d4173730925: Download complete
e8f612e3238f: Download complete
d11f03cd837d: Download complete
8abfc77aac61: Download complete
d1dfd80f5ed9: Download complete
Status: Downloaded newer image for joshhu/webdemo:latest
92d8ec3e8e0e0a70b486d80605eba70e66c7c3ada45b67602101675a79c233bb
bash-3.2$
```

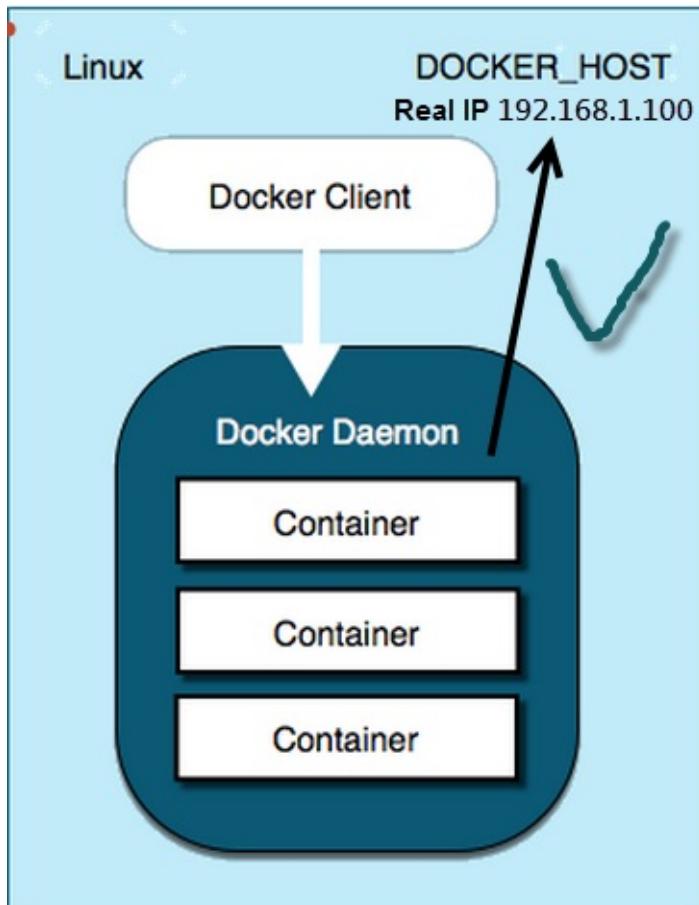
8. 但這個服務成功了嗎？照道理來說，應該是這台Mac的IP位址的8080埠對應到Docker中Container的80埠，我們試著輸入<http://localhost:8080/>，應該可以看到服務，但是並沒有。



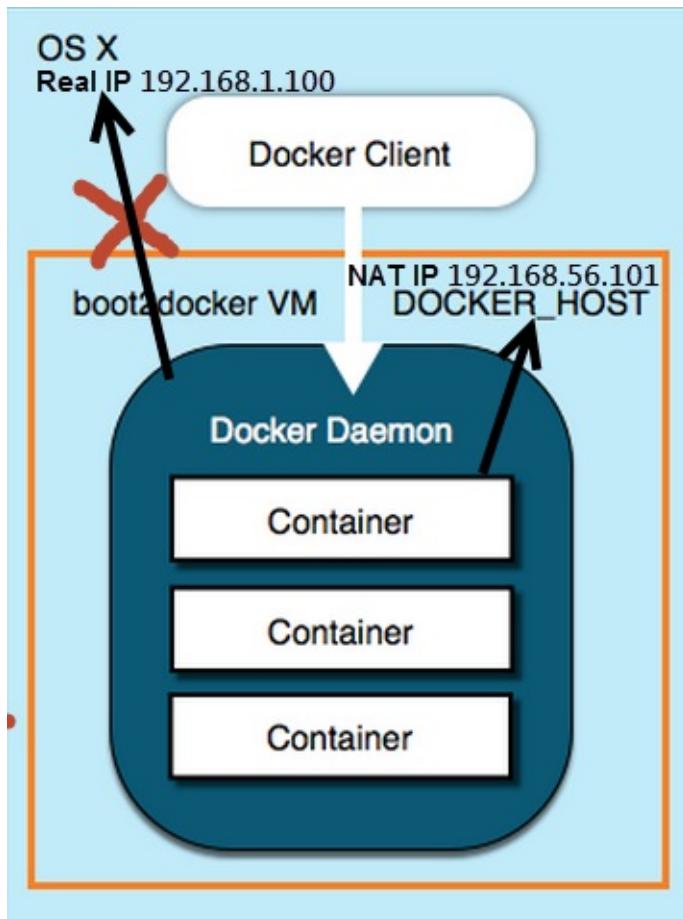
## boot2docker和標準docker的差別

在一般的Linux下使用Docker時，docker daemon是運作在該Linux的主機上，但是在boot2docker中，docker daemon是運作在Windows/Mac OS主機下的一個VM中，因此無法直接使用本機的資源(如IP位址、磁碟對應等)。

這是標準Linux



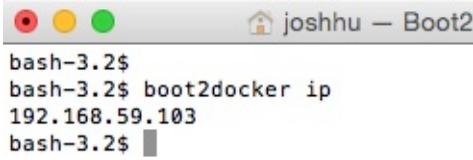
這是boot2docker



舉例來說，使用Linux主機上的Docker時，你可以設定該Linux主機的IP位置和Docker Container的對應，但在boot2docker中，Windows/Mac OS主機的IP位置是完全無法對應到Docker的Container中，只能讓VirtualBox VM的IP位置和Docker的Container對應。

## 使用Mac OS在VirtualBox網段的IP才行

1. 在正常的Linux Docker中，可以直接存取主機的IP以及對應的埠，就可以看到Apache啟動。
2. 但在Mac OS的boot2docker之下無法看到網頁，這是因為boot2docker是在一個VirtualBox的VM中執行的，只能先取得Mac的VirtualBox NAT IP。輸入 `boot2docker ip` 獲得Container執行VM的主機IP。



```
bash-3.2$ boot2docker ip
192.168.59.103
bash-3.2$
```

3. 此時再在Mac OS的瀏覽器中輸入剛才獲得的VM IP即對應埠，才能看到測試網站的產生。



PHP version: 5.5.9-1ubuntu4.7  
Apache version: Apache/2.4.7 (Ubuntu)

Boot2docker只是把一個VirtualBox的VM隱藏在記憶體中，因此並無法完全發揮正常Docker的功能，本書不推薦。另外有關Docker的IP位址對應在後面章節有詳細的說明。

# 在Windows下使用boot2docker

要安裝Windows下的boot2docker，必須確定你的電腦CPU有支援VT-X，但這並不是Docker需要的，而是VirtualBox需要的，可使用CPU-Z來檢查。另外也必須到BIOS中確定VT-X有開啟，如果都OK的話，即可直接安裝。

## 在Windows下安裝boot2docker

1. 先到<https://github.com/boot2docker/windows-installer/releases>下載 boot2docker的安裝檔docker-install.exe。
2. 下載回來後直接執行，不更改預設值按「Next」即可，這邊就略過。
3. 在安裝完畢後，桌面上會出現一個boot2docker Start的圖示，直接點擊執行。

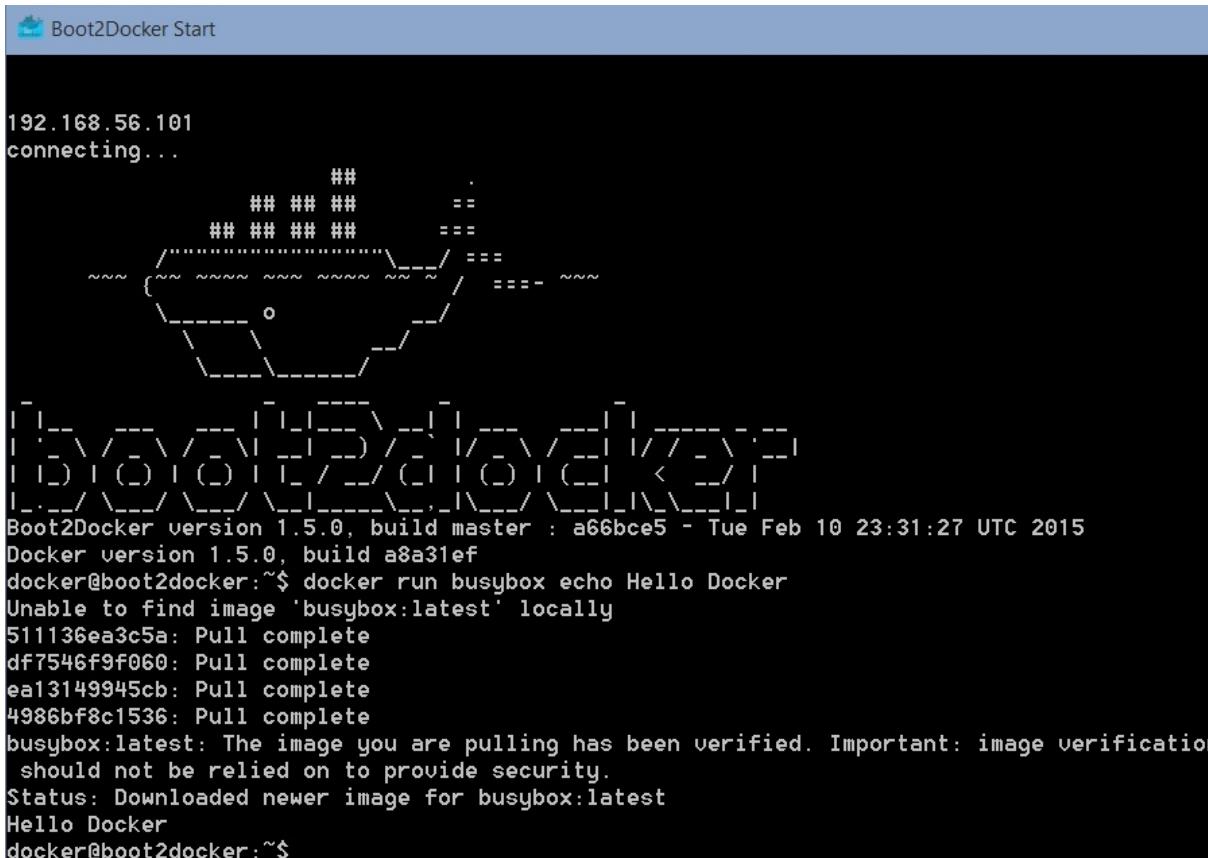


4. 此時會出現如下圖的畫面，這時是將精簡的Linux VM載入記憶體，如果中間有任何詢問視窗，都選擇「Y」即可。

```
Boot2Docker Start

copying initial boot2docker.iso (run "boot2docker.exe download" to update)
initializing...
Generating public/private rsa key pair.
Your identification has been saved in c:\Users\joshhu\.ssh\id_boot2docker.
Your public key has been saved in c:\Users\joshhu\.ssh\id_boot2docker.pub.
The key fingerprint is:
3f:50:96:35:68:56:b5:bd:28:e5:78:4c:3c:5f:fc:e8 joshhu@WIN-KLC2LK02157
The key's randomart image is:
+--[ RSA 2048]----+
|   o+.. |
| +o.. + |
| o+ = .+|
| o * +.+|
| S o = .o.|
| o o. |
| o E |
| . |
+
```

5. 當出現下圖的畫面，表示載入成功。
6. 輸入 `docker run busybox echo Hello Docker`，如果出現下圖的畫面時，表示安裝成功。



```

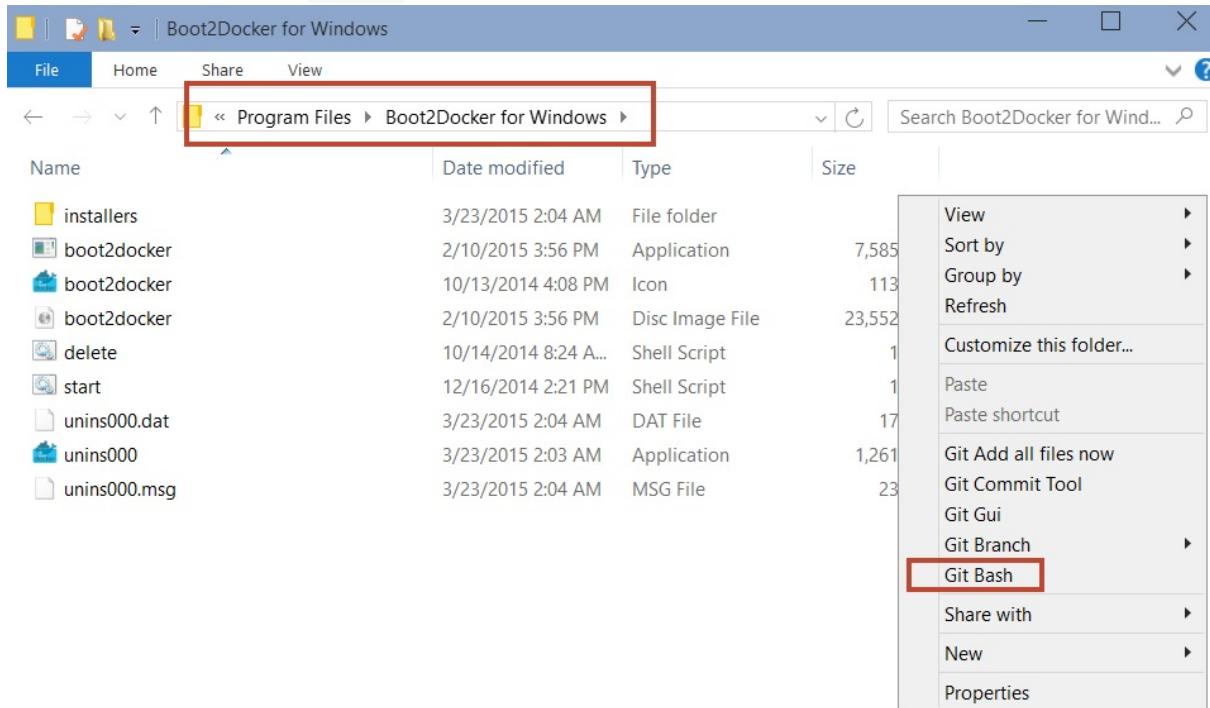
192.168.56.101
connecting...
##          ##
## ## ##    ==
## ## ## ==-
~~~ {~~~ ~~~ ~~~ ~~~ } / ==- ~~~
   \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ 
Boot2Docker version 1.5.0, build master : a66bce5 - Tue Feb 10 23:31:27 UTC 2015
Docker version 1.5.0, build a8a31ef
docker@boot2docker:~$ docker run busybox echo Hello Docker
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 verification
should not be relied on to provide security.
Status: Downloaded newer image for busybox:latest
Hello Docker
docker@boot2docker:~$
```

## 如果無法執行

在某些64位元的Windows 7/8/10系統下，boot2docker會一直無法啟動VirtualBox，此時你只要去VirtualBox的官方安裝重新安裝最新版的VirtualBox後即可。

另外如果按下boot2docker start圖示一直會跳出記事本或是無法進去，可以用下列方便解決。

1. 從Windows的檔案總管進入boot2docker的安裝目錄，預設為 C:\Program Files\Boot2Docker for Windows\。
2. 在該目錄下按滑鼠右鍵，選擇 Git bash。



3. 出現bash視窗時，輸入 ./start.sh 即可。

```
MINGW32:/C/Program Files/Boot2Docker for Windows  
initializing...  
Virtual machine boot2docker-vm already exists  
  
starting...  
Waiting for VM and Docker daemon to start...  
.o  
Started.  
Writing c:\Users\joshhu\.boot2docker\certs\boot2docker-vm\ca.pem  
Writing c:\Users\joshhu\.boot2docker\certs\boot2docker-vm\cert.pem  
Writing c:\Users\joshhu\.boot2docker\certs\boot2docker-vm\key.pem  
Docker client does not run on Windows for now. Please use  
"c:\Program Files\Boot2Docker for Windows\boot2docker.exe" ssh  
to SSH into the VM instead.  
  
192.168.56.101  
connecting...  
##  
## ## ## .  
## ## ## ## ==  
~~~ {----/ ===  
~~~~ ~~~~ ~~~~ ~~~~ ~~~~ / ---- ~~~  
~~~~ {----/ ===  
~~~~ ~~~~ ~~~~ ~~~~ ~~~~ / ---- ~~~  
~~~~ {----/ ===  
~~~~ ~~~~ ~~~~ ~~~~ ~~~~ / ---- ~~~  
~~~~ {----/ ===  
~~~~ ~~~~ ~~~~ ~~~~ ~~~~ / ---- ~~~  
~~~~ {----/ ===  
~~~~ ~~~~ ~~~~ ~~~~ ~~~~ / ---- ~~~  
~~~~ {----/ ===  
~~~~ ~~~~ ~~~~ ~~~~ ~~~~ / ---- ~~~  
~~~~ {----/ ===  
~~~~ ~~~~ ~~~~ ~~~~ ~~~~ / ---- ~~~  
~~~~ {----/ ===  
~~~~ ~~~~ ~~~~ ~~~~ ~~~~ / ---- ~~~  
~~~~ {----/ ===  
~~~~ ~~~~ ~~~~ ~~~~ ~~~~ / ---- ~~~  
~~~~ {----/ ===  
~~~~ ~~~~ ~~~~ ~~~~ ~~~~ / ---- ~~~
```

本書的重點不在boot2docker，也不推薦使用，如果一定要在Windows/Mac下使用Docker，還是安裝一個Hypervisor，再下載本書專用的VM，或自行安裝Ubuntu，再正式安裝Docker比較方便。

# 在Mac OS/Windows中的VM安裝Docker

---

另一個方法，即循正規安裝，在Mac OS或Windows下先安裝習慣用的Hypervisor，推薦的有：

## 自行在VM中安裝Ubuntu

讀者可以先去下載下列幾個虛擬化的Hypervisor，如：

- VirtualBox(Mac OS/Windows/Linux，本書推薦)
- VMware Workstation (Windows/Linux)
- VMware Fusion (Mac OS)

然後再自行安裝或下載Ubuntu Linux ISO檔案，先安裝好Linux，然後再安裝Docker，這個方法彈性大，同時也方便示範後面章節所提的Docker叢集，由於正規方式和在Linux下安裝Docker完全一樣，讀者直接參考Linux的安裝部分即可。

Ubuntu ISO下載：<http://www.ubuntu.com/download/server>

---

## 注意 – Hypervisor的選擇， Hosted或是Bare Metal

在本書的示範章節，都會使用Hosted的Hypervisor，這是大部分測試環境最方便使用的。在測試完後需要上生產環境時，才會應用到Bare Metal的企業版Hypervisor，主要是為了快速大量部署，本書稍後會有專門的章節介紹。

---

# 在Ubuntu Linux下安裝Docker

---

時至今日大部分的Linux版本都可正常執行Docker，只要該Linux核心的版本編號夠新。如果你已經習慣於某一個版本的Linux，可以參考該Linux廠商的官方安裝方式，本書的教學示範全部使用Ubuntu Linux。

另外一個對Docker支援較強的Linux即為CoreOS。通常CoreOS本身會發佈VM格式，因此只要到CoreOS的官網下載對應Hypervisor的VM即可使用。本小節就針對這兩個Linux的版本來安裝Docker，另外也會在本章最後一小節列出安裝Docker時常見的問題。

---

## 注意 –Docker部署在實體伺服器 vs 部署在VM中

本書使用對Docker支援最好的Ubuntu作為主要版本，叢集部署則使用CoreOS。其它版本的Linux當然對Docker的支援也完全沒問題，但除了版本不夠新的問題之外，許多工具(如 `nsenter` )也不一定有安裝，不是使用這兩個版本Linux的讀者必須自行安裝這些工具。

---

# 使用本書所附的VM

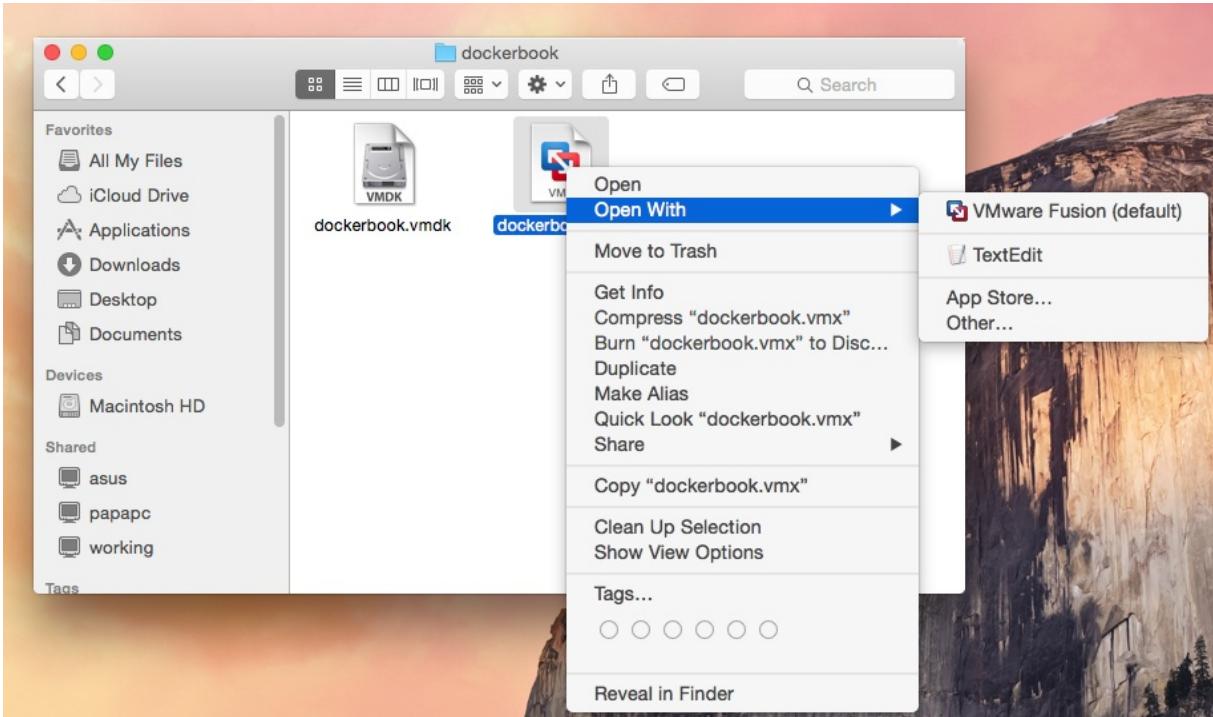
使用Docker最快的方式，就是直接使用本書所建立好的VM，直接到本書的VM下點下載。是一個zip檔。

本書的Docker VM下載點

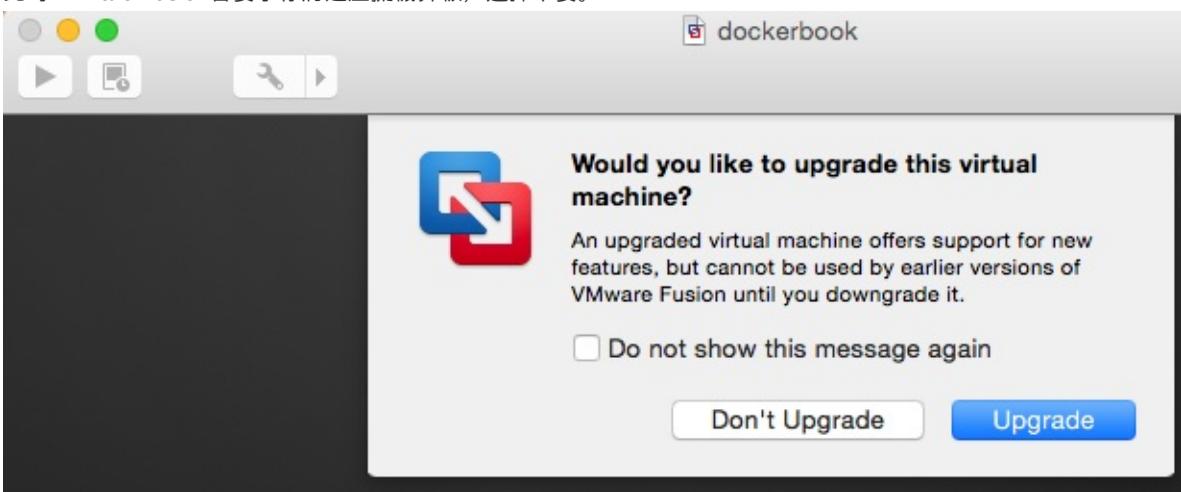
## 使用VMware Workstation/Fusion產品開啟

下載 dockerbook.zip 後解壓，會有 dockerbook.vmx 及 dockerbook.vmdk 兩個檔案，如果你使用的是VMware桌面系列的產品(VMware Workstation/Fusion)，直接開啟 dockerbook.vmx 即可。以下用Mac OS下的VMware Fusion示範：

1. 開啟這個 dockerbook.vmx



2. 此時VMware Fusion會要求你將這虛擬機昇級，選擇不要。

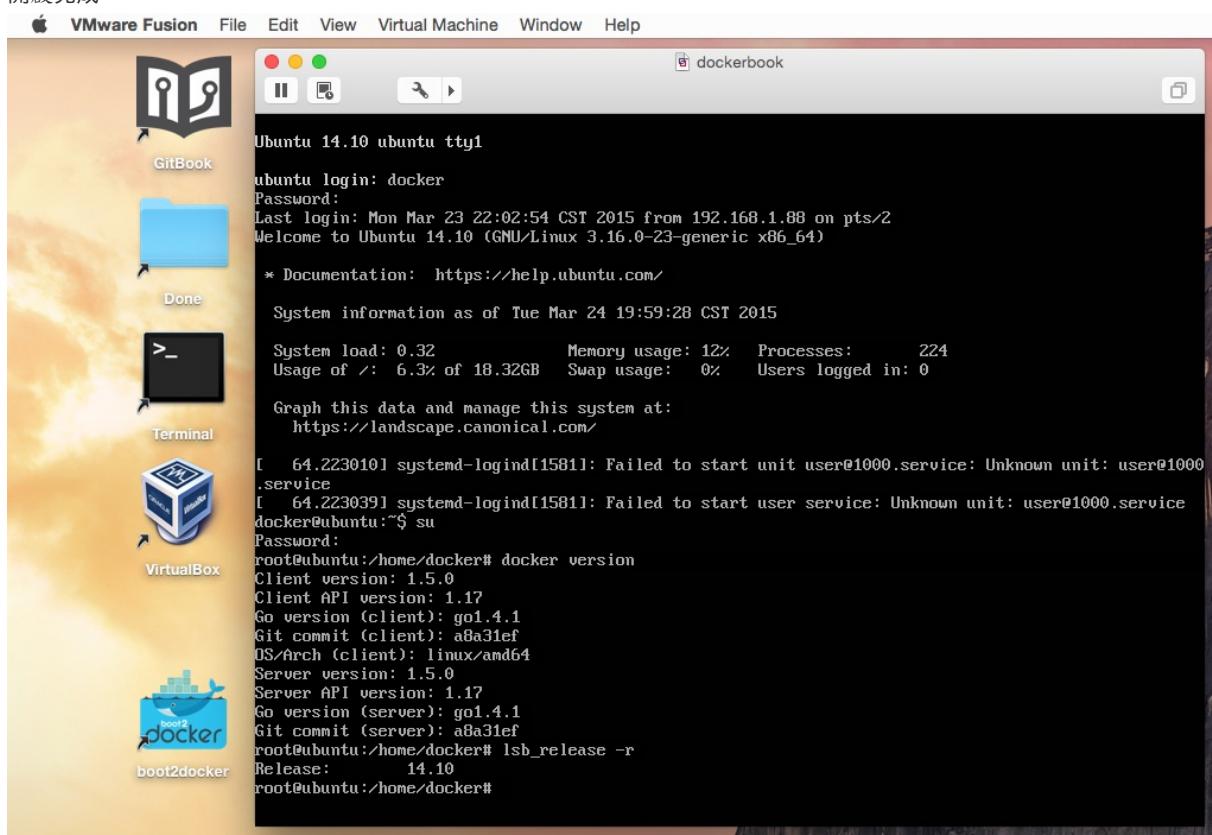


3. 這邊選擇 I Copied It。



4. 其它都是 ok 過了就行。

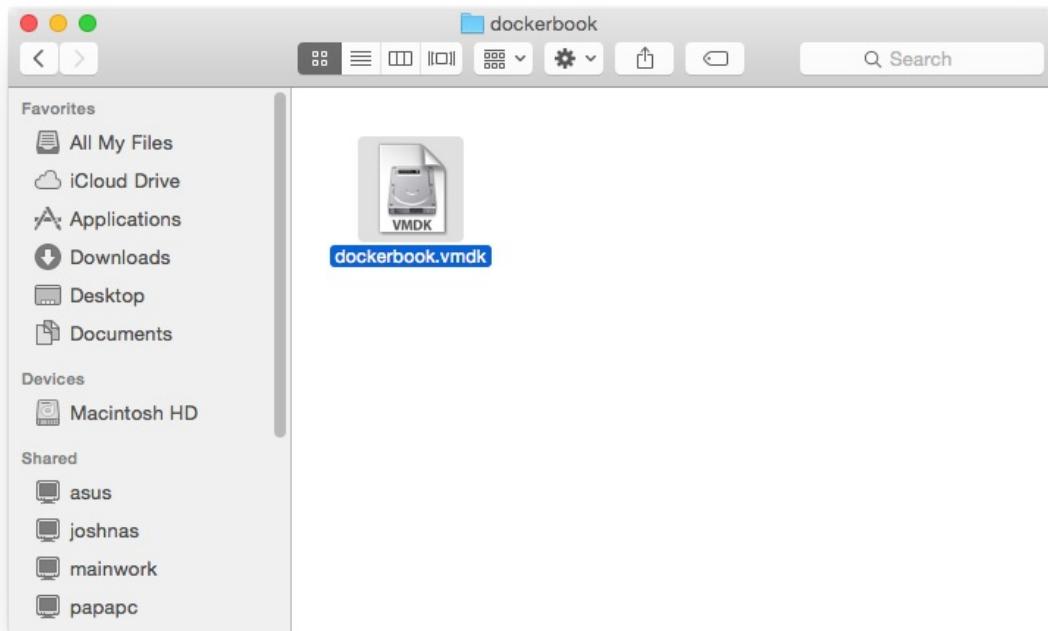
5. 開啟完成



## 在VirtualBox中使用本書所附的VM安裝Docker

如果你使用的是VirtualBOX，則必須建立一個新的VM，再將下載回來解壓縮的虛擬碟碟的 `dockerbook.vmdk` 附加成一個VirtualBox的虛擬磁碟，以下為Mac OS下的步驟：

1. 先將本書所附的 `dockerbook.vmdk` 放到使用者Home目錄下的 `VirtualBox VMs/dockerbook` 下。



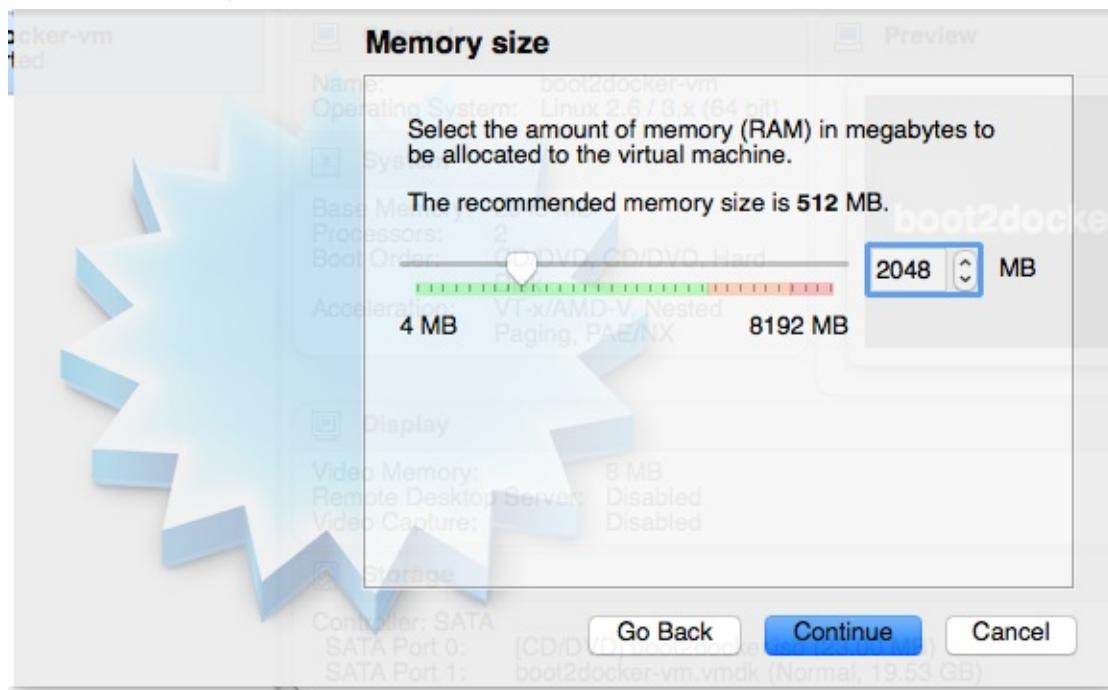
2. 開啟VirtualBox，選擇 new。



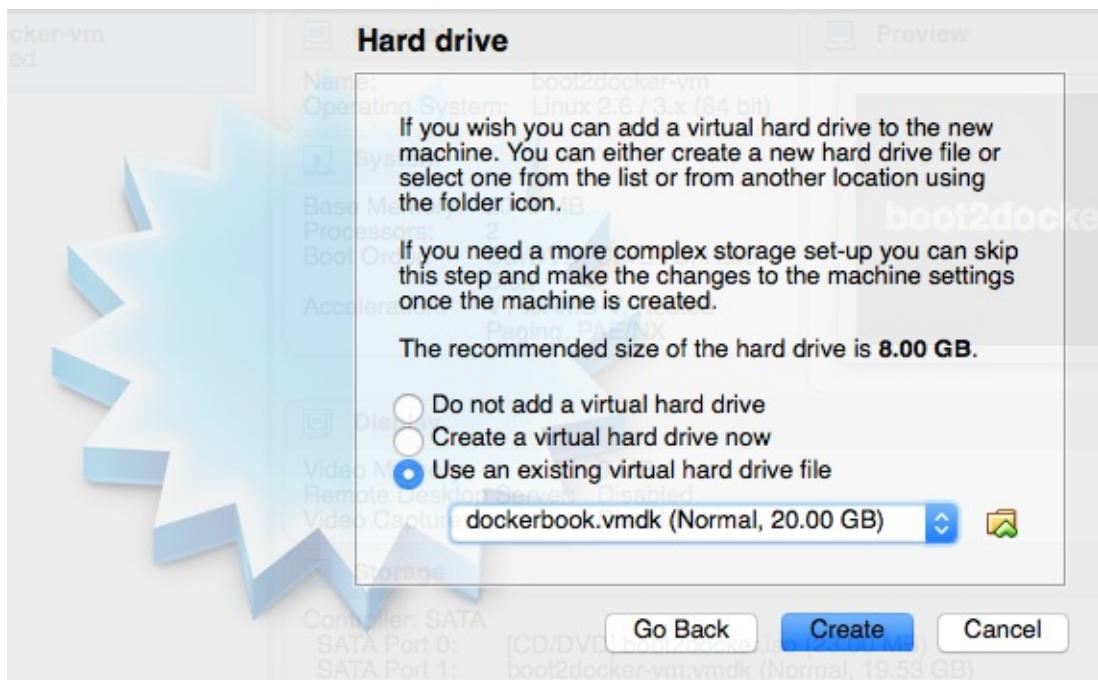
3. 輸入這個VM的名字，作業系統以及種類，如圖所示。



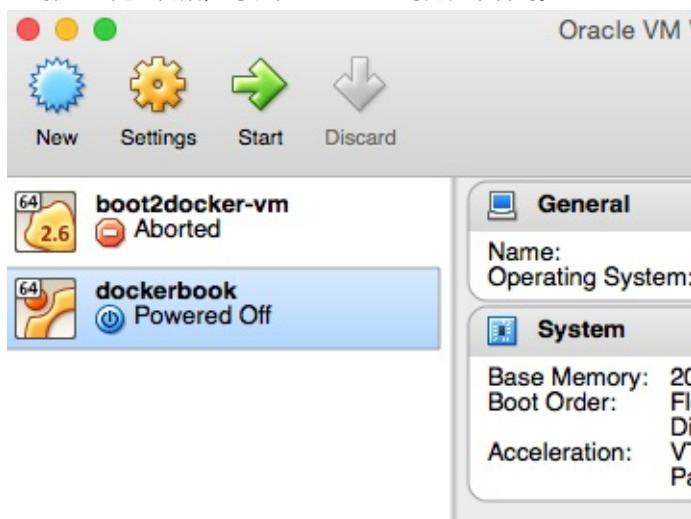
4. 這個VM的記憶體大小，一般我們會給2GB。



5. 這邊要選擇 use an existing virtual hard drive file，並且選擇剛才的那個 dockerbook.vmdk，然後按下 create。



6. 此時該VM建立完成，可以在VirtualBox的列表中看到。



7. 開啟這台VM，並且可以進入該VM的主控台，進入後，帳號為 docker，密碼為 dockerbook，並且輸入 su 取得root權限，密碼也是 dockerbook 。
8. 輸入 docker run -d --name web -p 8080:80 joshhu/webdemo 。

```
root@ubuntu:/home/docker# You have the Auto capture keyboard option turned on. This will cause the Virtual Machine to automatically capture
root@ubuntu:/home/docker# The Virtual Machine reports that the guest OS supports mouse pointer integration. This means that you do not need
root@ubuntu:/home/docker#
root@ubuntu:/home/docker#
root@ubuntu:/home/docker#
root@ubuntu:/home/docker#
root@ubuntu:/home/docker# docker run -d --name web -p 8080:80 joshhu/webdemo
Unable to find image 'joshhu/webdemo:latest' locally
Pulling repository joshhu/webdemo
a3574f323972: Pulling image (latest) from joshhu/webdemo, endpoint: https://regi
a3574f323972: Pulling dependent layers
511136ea3c5a: Download complete
f3c84ac3a053: Pulling metadata
-
```

9. 看一下主機的IP：`ifconfig`。
10. 進入瀏覽器，並且輸入`http://192.168.1.113:8080`。出現下圖的畫面表示安裝成功。

Congratulations! You have just built your first docker service.  
Please visit my books at <https://www.gitbook.com/@joshu>

PHP version: 5.5.9-1ubuntu4.7  
Apache version: Apache/2.4.7 (Ubuntu)

```
root@ubuntu:/#
```

```
dockerbook [Running]
collisions:0 txqueuelen:0
RX bytes:536 (536.0 B) TX bytes:648 (648.0 B)

eth0      Link encap:Ethernet HWaddr 08:00:27:f8:b8:90
          inet addr:192.168.1.113  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe00::400:27ff:fe8:b890/64 Scope:Link
             UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
             RX packets:146 errors:0 dropped:0 overruns:0 frame:0
             TX packets:32 errors:0 dropped:0 overruns:0 carrier:0
             collisions:0 txqueuelen:1000
             RX bytes:22369 (22.3 KB) TX bytes:3188 (3.1 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
             UP LOOPBACK RUNNING MTU:65536 Metric:1
             RX packets:0 errors:0 dropped:0 overruns:0 frame:0
             TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
             collisions:0 txqueuelen:0
             RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)

veth8e763ad  Link encap:Ethernet HWaddr fa:e0:76:75:96:1b
            inet6 addr: fe80::f8e0:76ff:fe75:961b/64 Scope:Link
               UP BROADCAST RUNNING MTU:1500 Metric:1
               RX packets:8 errors:0 dropped:0 overruns:0 frame:0
               TX packets:16 errors:0 dropped:0 overruns:0 carrier:0
               collisions:0 txqueuelen:0
               RX bytes:648 (648.0 B) TX bytes:1296 (1.2 KB)
```

注意 - 使用ssh連入較方便

一般我們建立了VM，並不會在Hypervisor的主控台下操作，因為不但不方便，也無法使用複製貼上功能。通常將安裝好Docker的VM開啟後，會使用ssh進入該VM，筆者習慣使用Linux的標準終端視窗，您也可以使用Windows上的PuTTY或Mac本身的終端視窗。

```
x docker@ubuntu: ~
joshhu@ubuntu:~$ 
joshhu@ubuntu:~$ 
joshhu@ubuntu:~$ ssh docker@192.168.1.113
docker@192.168.1.113's password:
Welcome to Ubuntu 14.10 (GNU/Linux 3.16.0-23-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

 System information as of Tue Mar 24 00:50:44 CST 2015

 System load:  0.0          Processes:      93
 Usage of /:   7.8% of 18.32GB  Users logged in:    1
 Memory usage: 5%           IP address for eth0:  192.168.1.113
 Swap usage:   0%           IP address for docker0: 172.17.42.1

 Graph this data and manage this system at:
 https://landscape.canonical.com/

Last login: Tue Mar 24 00:34:03 2015
docker@ubuntu:~$
```

# 手動安裝Docker

如果你已經有了現成使用中的Linux，不想在虛擬機中安裝Docker，視使用的Linux distro，Docker的安裝方法也不同。在 Docker的官網上有列出所有Linux下的安裝指南，這邊就不多作說明，但官網指出，Docker最適合的環境就是Ubuntu，安裝起來也是最簡單的。讀者這邊要確定的，就是不要使用傳統的 `apt-get install docker.io` 來安裝，因為這樣就無法獲得最新的Docker版本了。

這邊有完整的Docker安裝，幾乎Linux全包了

The screenshot shows a web browser displaying the Docker documentation at [docs.docker.com](https://docs.docker.com). The page is titled "Installation". On the left, there's a sidebar with "Reference" and "Contributor Guide" sections. The main content area lists various platforms for Docker installation, each preceded by a blue circular bullet point:

- Ubuntu
- Mac OS X
- Microsoft Windows
- Amazon EC2
- Arch Linux
- Binaries
- CentOS
- CRUX Linux
- Debian
- Fedora
- FrugalWare
- Google Cloud Platform
- Gentoo
- IBM Softlayer
- Rackspace Cloud
- Red Hat Enterprise Linux
- Oracle Linux
- SUSE
- Docker Compose

## 注意 - 本書預設使用root帳號進行操作

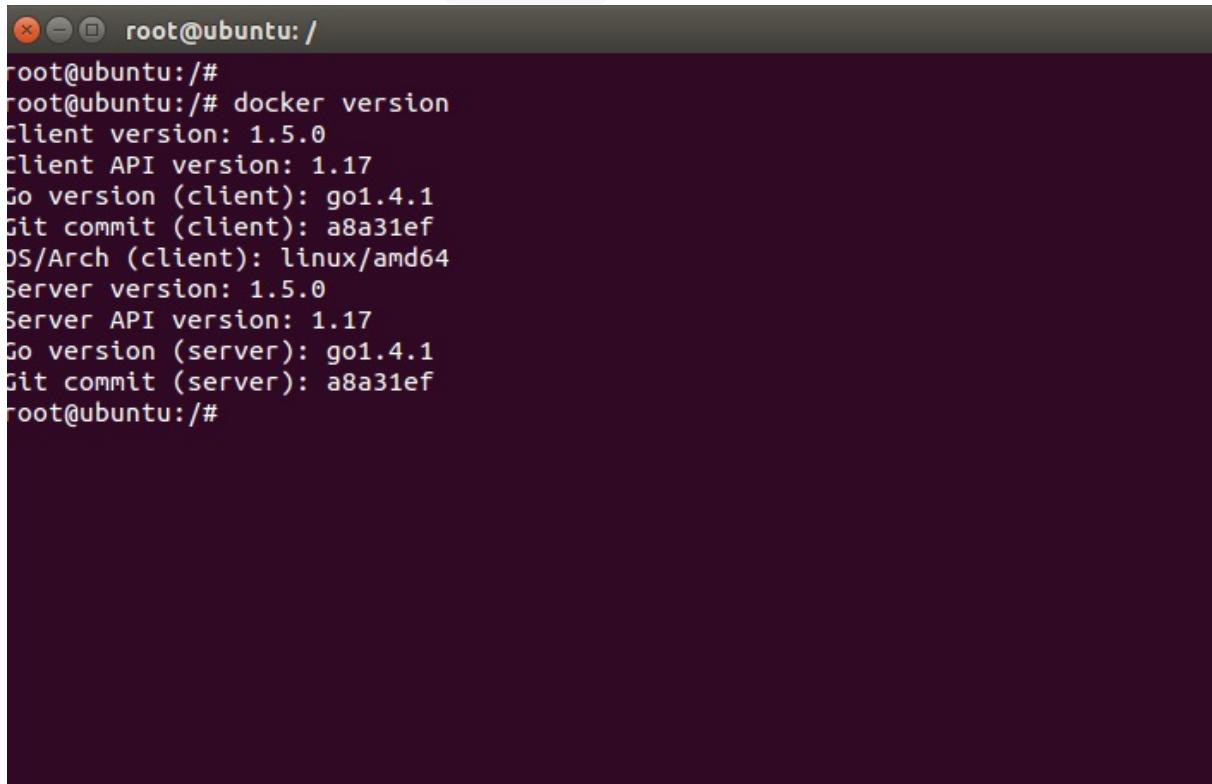
本書的操作均以root帳號為基礎，因此別忘了啟用電腦的root帳號，如果你的Ubuntu還沒有root帳號，請使用有sudo權限的帳號，在Linux的終端機中輸入：`sudo passwd root`，之後會要求你輸入目前使用帳號的密碼，以及指定給root的密碼(兩次)。輸入完之後，你可以輸入 `su`，之後輸入root的密碼後，就可以進入root身份了。

## 安裝Docker

1. 先更新Ubuntu，輸入 `apt-get update`。
2. 再輸入 `apt-get upgrade`。
3. 在Ubuntu的終端機視窗中，輸入

```
curl -sSL https://get.docker.com/ubuntu/ | sudo sh
```

- 如果沒有錯誤訊息，在安裝完畢之後，輸入 docker version，應就會出現下圖的畫面。

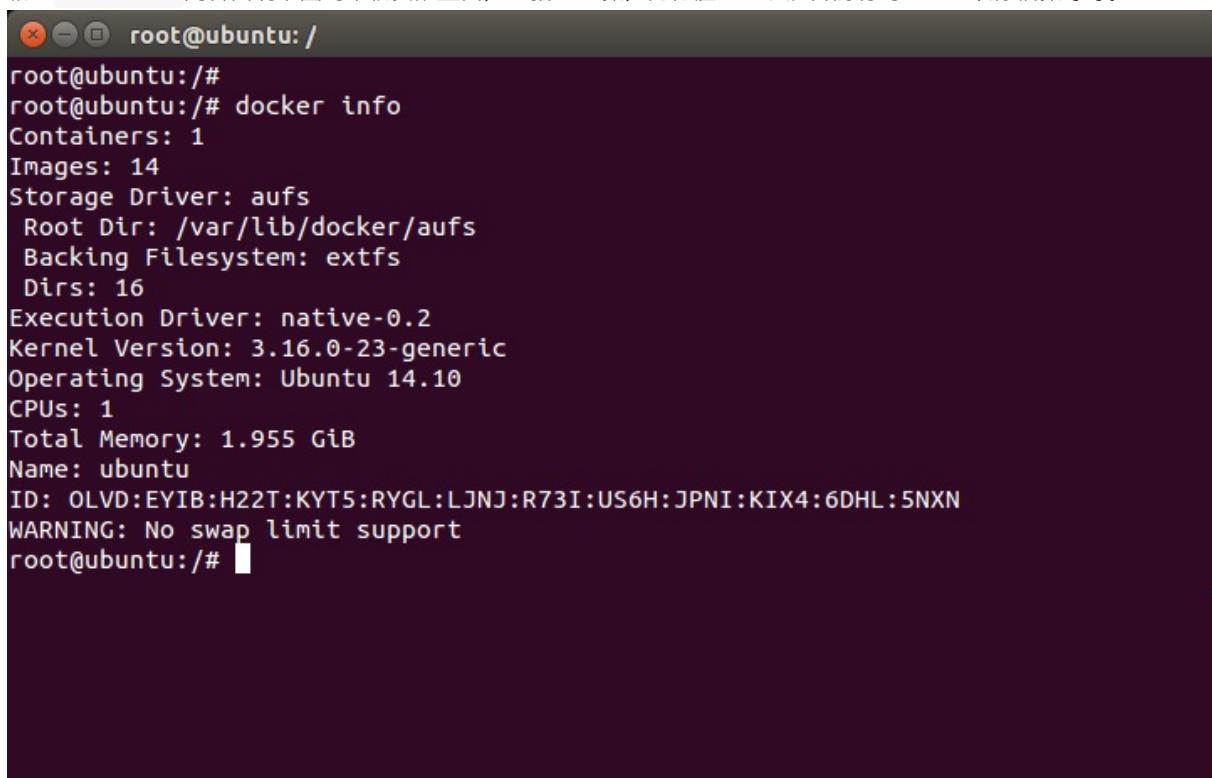


```
root@ubuntu:/# docker version
Client version: 1.5.0
Client API version: 1.17
Go version (client): go1.4.1
Git commit (client): a8a31ef
OS/Arch (client): linux/amd64
Server version: 1.5.0
Server API version: 1.17
Go version (server): go1.4.1
Git commit (server): a8a31ef
root@ubuntu:/#
```

接下來我們就執行幾個Docker的指令來驗證Docker的安裝成功。以下的指令讀者目前還不需要了解其意義，本書稍後都會有完整的解釋。

## 檢驗Docker是否安裝成功

- 輸入 docker info 則會出現下圖的系統資訊畫面，包括CPU數，記憶體大小以及目前有的docker映像檔數等等。



```
root@ubuntu:/# docker info
Containers: 1
Images: 14
Storage Driver: aufs
 Root Dir: /var/lib/docker/aufs
 Backing Filesystem: extfs
 Dirs: 16
Execution Driver: native-0.2
Kernel Version: 3.16.0-23-generic
Operating System: Ubuntu 14.10
CPUs: 1
Total Memory: 1.955 GiB
Name: ubuntu
ID: OLVD:EYIB:H22T:KYT5:RYGL:LJNJ:R73I:US6H:JPNI:KIX4:6DHL:5NXN
WARNING: No swap limit support
root@ubuntu:/#
```

- 輸入 ifconfig docker0 則應該出現如圖的畫面，表示Docker的網路部分也安裝成功。

```
root@ubuntu:/#  
root@ubuntu:/# ifconfig docker0  
docker0 Link encap:Ethernet HWaddr 56:84:7a:fe:97:99  
      inet addr:172.17.42.1 Bcast:0.0.0.0 Mask:255.255.0.0  
      inet6 addr: fe80::5484:7aff:fefe:9799/64 Scope:Link  
        UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1  
        RX packets:19 errors:0 dropped:0 overruns:0 frame:0  
        TX packets:21 errors:0 dropped:0 overruns:0 carrier:0  
        collisions:0 txqueuelen:0  
          RX bytes:2117 (2.1 KB) TX bytes:2152 (2.1 KB)  
  
root@ubuntu:/#
```

3. 輸入 `docker run -d --name web -p 8080:80 joshhu/webdemo`。如果出現了下圖的畫面，表示Docker已經安裝成功了。

```
root@ubuntu:/#  
root@ubuntu:/# docker run -d --name web -p 8080:80 joshhu/webdemo  
Unable to find image 'joshhu/webdemo:latest' locally  
Pulling repository joshhu/webdemo  
a3574f323972: Pulling image (latest) from joshhu/webdemo, endpoint: https://regi  
a3574f323972: Download complete  
511136ea3c5a: Download complete  
f3c84ac3a053: Download complete  
a1a958a24818: Download complete  
9fec74352904: Download complete  
d0955f21bf24: Download complete  
1214be61bcaa: Download complete  
45ad00454734: Download complete  
8be3dadcb43a: Download complete  
2d4173730925: Download complete  
e8f612e3238f: Download complete  
d11f03cd837d: Download complete  
8abfc77aac61: Download complete  
d1dfd80f5ed9: Download complete  
Status: Downloaded newer image for joshhu/webdemo:latest  
6b4add584b35a16ac1e82ea151aae72217cf59cf154f9a92d7faa846d929c30d  
root@ubuntu:/#
```

# 讓Docker更好用的工具

本節介紹讓Docker的操作更方便的軟體，包括Windows下的PieTTY及一些Linux下的Docker專用scripts檔案。

## 方便Docker操作的小工具

- [PieTTY](#)(Windows) : Windows下的SSH軟體，以PuTTY為基礎但更好用。
- [WinSCP](#)(Windows) : Windows和Linux之間傳送檔案的工具。
- Screen(Linux Terminal) : 多個登入TTY之間的切換工具。

## 進入Docker Container的好用scripts

Docker的Container被看成虛擬機的一種，我們常要進入Docker建立的VM中進行作業。此外也常需獲得執行中Container的資訊。雖然Docker提供了 `docker inspect`，但需要配合正規表示法以及json的文字處理。為了方便，筆者把這些使用 `docker inspect` 的語法簡化成一些指令，分別為：

- denter : 進入容器中，如：`denter web`，即會直接進入這個容器內。
- dip : 取得執行容器的IP，如：`dip web`。
- dpid : 取得容器在宿主Linux下的pid，如`dpid web`，取得該容器的pid。

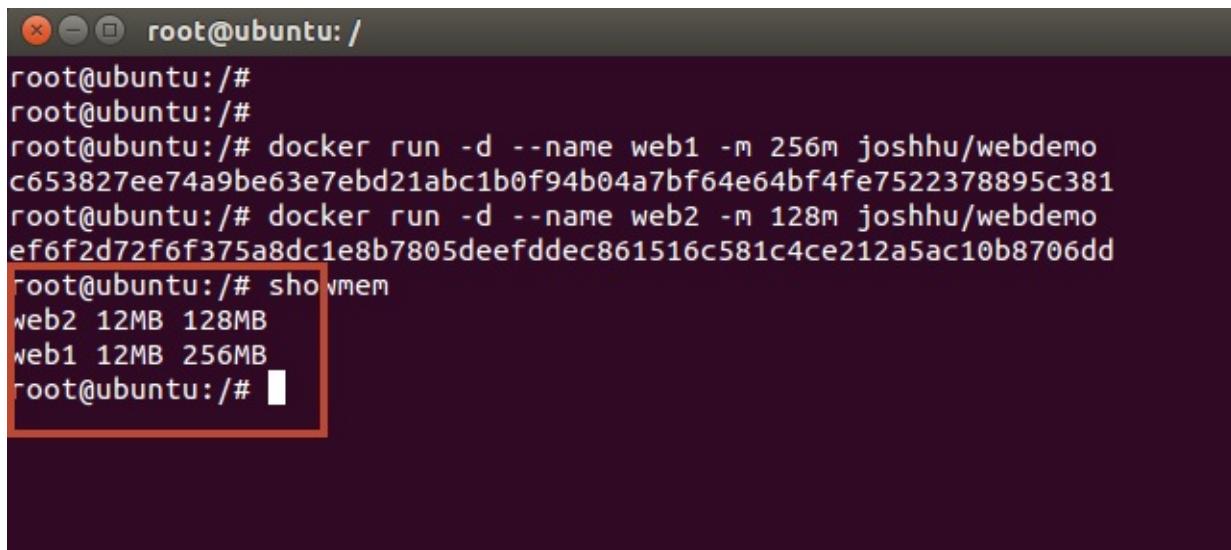
讀者可以下載這些指令，然後導入Linux下使用者的.bashrc設定檔。注意，不同的Linux使用者都需要重新執行一次這兩行指令。執行完畢之後先用 `exit` 登出系統，然後再登入就可以使用了。輸入的指令如下(請以root身份執行)：

```
$ wget -P ~ https://github.com/joshhu/docker/raw/master/docker_scripts/.bashrc_docker;
$ echo "[ -f ~/.bashrc_docker ] && . ~/.bashrc_docker" >> ~/.bashrc; source ~/.bashrc
$ exit
```

```
root@ubuntu:/home/joshhu# dip web1
172.17.0.5
root@ubuntu:/home/joshhu# dpid web1
3723
root@ubuntu:/home/joshhu# denter web1
root@c653827ee74a:~# uname -r
3.16.0-31-generic
root@c653827ee74a:~# exit
logout
root@ubuntu:/home/joshhu#
```

另一個工具則是顯示目前執行中的Container，佔用了多少系統記憶體，以及一開始宣告多少記憶體的值，這個工具稱之為 `showmem`。安裝方法一樣簡單，請以root身份執行，且此script只適用於Ubuntu：

```
$ wget -P ~ https://github.com/joschu/docker/raw/master/docker_scripts/showmem
$ chmod +x ~/showmem
$ mv ~/showmem /usr/bin
$ showmem
```



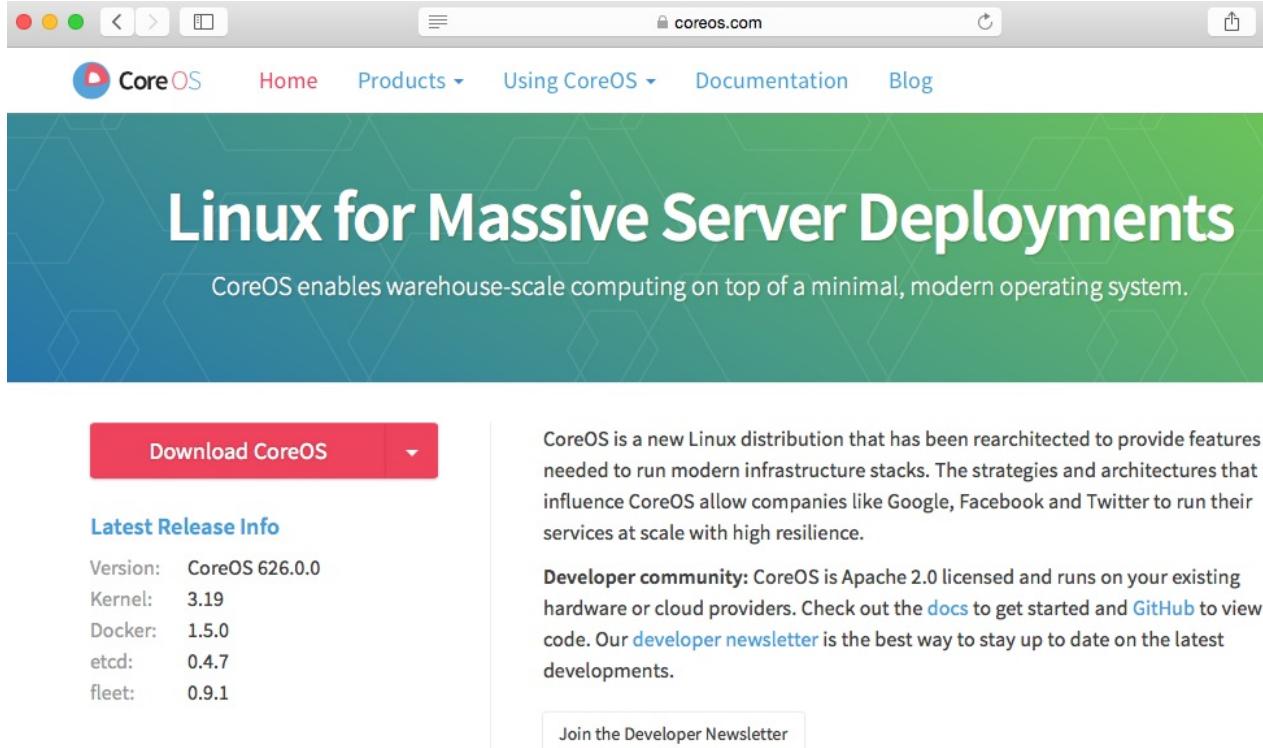
A screenshot of a terminal window titled "root@ubuntu: /". The terminal shows the following command sequence:

```
root@ubuntu:/#
root@ubuntu:/#
root@ubuntu:/# docker run -d --name web1 -m 256m joshhu/webdemo
c653827ee74a9be63e7ebd21abc1b0f94b04a7bf64e64bf4fe7522378895c381
root@ubuntu:/# docker run -d --name web2 -m 128m joshhu/webdemo
ef6f2d72f6f375a8dc1e8b7805deefddc861516c581c4ce212a5ac10b8706dd
root@ubuntu:/# showmem
web2 12MB 128MB
web1 12MB 256MB
root@ubuntu:/#
```

The output of the "showmem" command is highlighted with a red rectangular box.

# 使用雲端專用CoreOS

在雲端上要執行Docker主要的目的之一就是為了Cluster。Docker本身可以和不同的私有雲平台整合，如vSphere 或是 OpenStack。但在這種私有雲平台上，使用Ubuntu就不太方便了。前面我們會提到最適合Cluster用的Linux版本為CoreOS，這一節我們就直接從CoreOS的網站下載已經安裝好的Docker VM並部署到vSphere中。



The screenshot shows the official CoreOS website at [coreos.com](https://coreos.com). The header includes the CoreOS logo, navigation links for Home, Products, Using CoreOS, Documentation, and Blog, and a search bar. The main banner features the text "Linux for Massive Server Deployments" and "CoreOS enables warehouse-scale computing on top of a minimal, modern operating system." Below the banner, there's a "Download CoreOS" button and a "Latest Release Info" section listing software versions: Version: CoreOS 626.0.0, Kernel: 3.19, Docker: 1.5.0, etcd: 0.4.7, and fleet: 0.9.1. To the right, there's a detailed description of CoreOS and a "Developer community" section.

CoreOS is a new Linux distribution that has been rearchitected to provide features needed to run modern infrastructure stacks. The strategies and architectures that influence CoreOS allow companies like Google, Facebook and Twitter to run their services at scale with high resilience.

**Developer community:** CoreOS is Apache 2.0 licensed and runs on your existing hardware or cloud providers. Check out the [docs](#) to get started and [GitHub](#) to view code. Our [developer newsletter](#) is the best way to stay up to date on the latest developments.

[Join the Developer Newsletter](#)

# CoreOS簡介

CoreOS是一個叢集專用的Linux，在Docker的初期，也是和Docker合作最密切的Linux版本，但隨著Docker的商業化及日漸走紅，CoreOS也漸行漸遠。雖然CoreOS也提供了自己格式的Container Rocket，但目前成熟度還有待加強。

CoreOS支援非常多安裝方式，包括常見的VM

[Download CoreOS](#) ▾

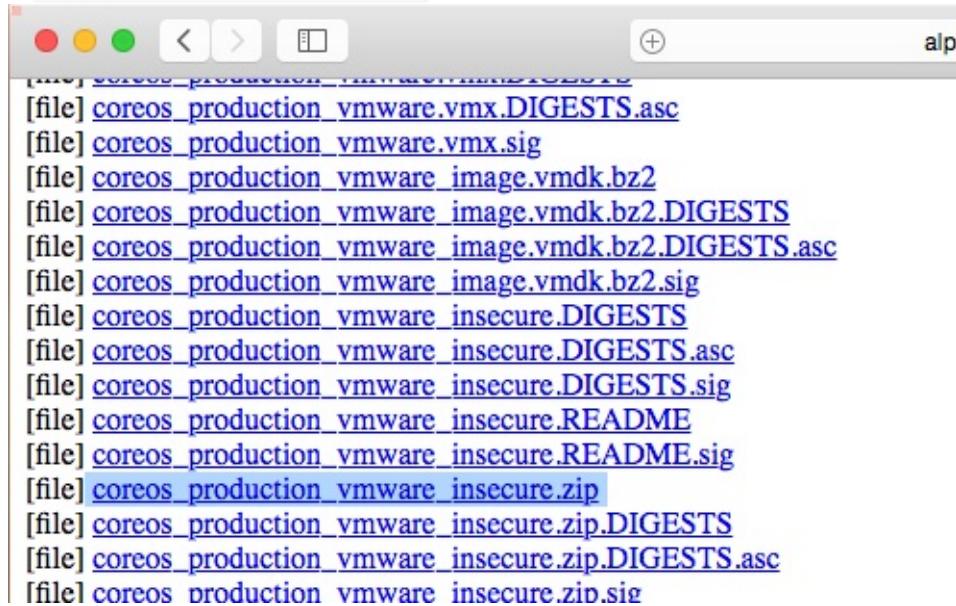
Bare Metal  
[PXE, iPXE, Install to Disk](#)

Cloud Providers  
[EC2, DigitalOcean, GCE, Rackspace](#),  
[Azure, Brightbox](#)

Virtualization Platforms  
[Vagrant, VMware, QEMU, Openstack](#),  
[Eucalyptus, ISO Image](#)

儘管如此，對Docker叢集支援最好的Linux仍然還是CoreOS，CoreOS也提供了VMware的VM方便部署Docker。針對同一個版本編號的CoreOS有兩個不同應用的VM，分別為：

- 針對測試用的(**如VMware Workstation**)VM：使用ssh無帳號密碼登入，主要用於本機使用，其檔案名稱為 `coreos_production_vmware_insecure.zip`。



- 針對生產環境用的(**vSphere ESXi**)VM，有預先安裝VMware Tools，其中沒有建立任何使用者帳號及密碼，主要用於叢集使用，其檔案名稱為 `coreos_production_vmware_image.vmdk.bz2` 的 `vmdk` 檔案，以及 `coreos_production_vmware.vmx` 的虛擬機檔案。



The screenshot shows a web browser window with the URL `alpha.release.core-os.net`. The page displays a list of file links, many of which are highlighted with red boxes. The listed files include:

- [file] [coreos\\_production\\_vmware.DIGESTS.asc](#)
- [file] [coreos\\_production\\_vmware.DIGESTS.sig](#)
- [file] [coreos\\_production\\_vmware.vmx](#)
- [file] [coreos\\_production\\_vmware.vmx.DIGESTS](#)
- [file] [coreos\\_production\\_vmware.vmx.DIGESTS.asc](#)
- [file] [coreos\\_production\\_vmware.vmx.sig](#)
- [file] [coreos\\_production\\_vmware\\_image.vmdk.bz2](#)
- [file] [coreos\\_production\\_vmware\\_image.vmdk.bz2.DIGESTS](#)
- [file] [coreos\\_production\\_vmware\\_image.vmdk.bz2.DIGESTS.asc](#)
- [file] [coreos\\_production\\_vmware\\_image.vmdk.bz2.sig](#)
- [file] [coreos\\_production\\_vmware\\_insecure.DIGESTS](#)
- [file] [coreos\\_production\\_vmware\\_insecure.DIGESTS.asc](#)
- [file] [coreos\\_production\\_vmware\\_insecure.DIGESTS.sig](#)
- [file] [coreos\\_production\\_vmware\\_insecure README](#)
- [file] [coreos\\_production\\_vmware\\_insecure README.sig](#)
- [file] [coreos\\_production\\_vmware\\_insecure.zip](#)
- [file] [coreos\\_production\\_vmware\\_insecure.zip.DIGESTS](#)
- [file] [coreos\\_production\\_vmware\\_insecure.zip.DIGESTS.asc](#)
- [file] [coreos\\_production\\_vmware\\_insecure.zip.sig](#)

這兩個CoreOS的VM在使用上不儘相同，從名字就可看出不同。`coreos_production_vmware_insecure` 使用公開的ssh key，因此拿來部署生產環境相當危險，在正式使用前必須更換ssh key。

`coreos_production_vmware` 因為安全性較佳，並且有預設安裝 `open-vm-tools`，適合用在雲端及生產環境，但由於沒有預設使用者帳號密碼，因此使用上必須經過設定。設定方法有：

- 建立帳號密碼
- 使用雲端 config-drive 方式

本小節會介紹第一種方式，第二種方式則留在本書後半部有關雲端部署Docker時再細談。

# 建立自動登入的CoreOS

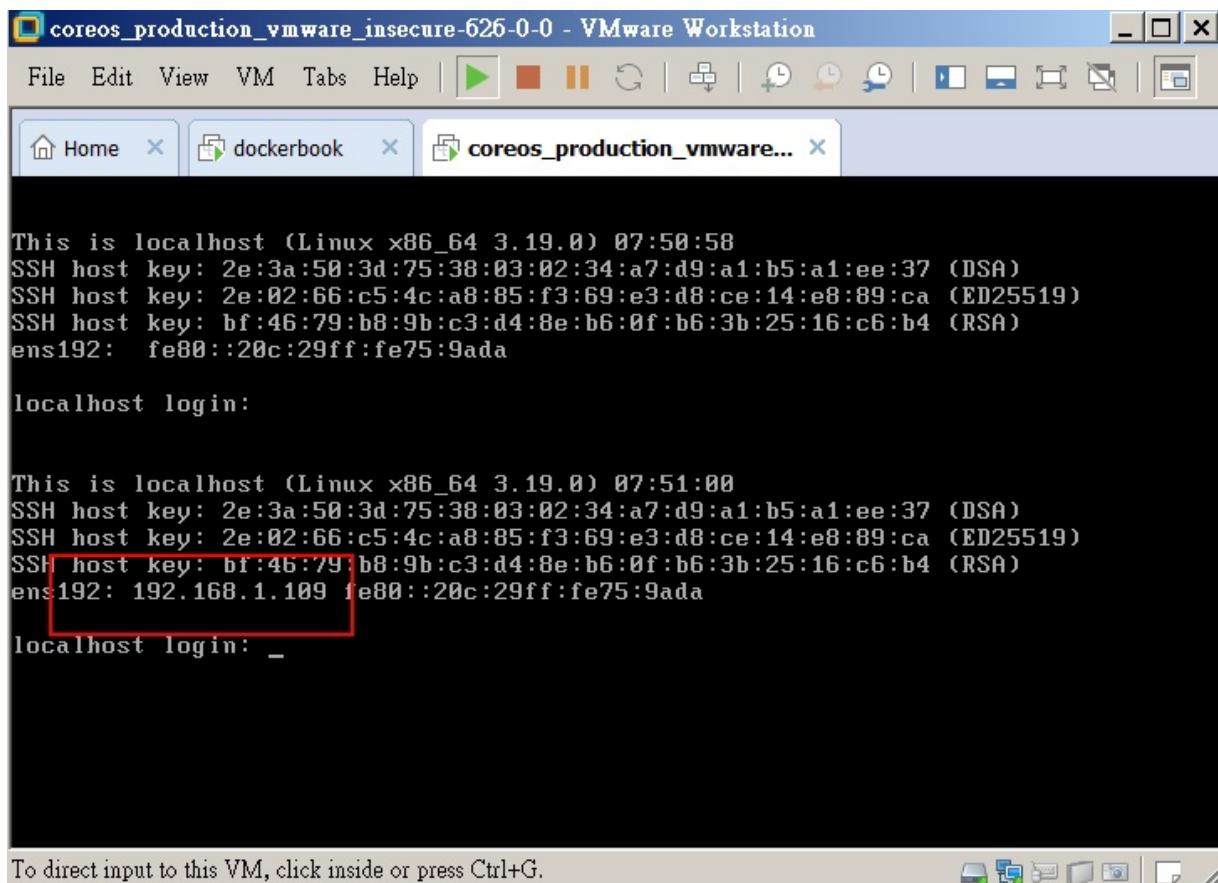
本小節將說明使用SSH的Private key進行登入的CoreOS VM，其中已經安裝好最新版本的Docker了。

## 啟動CoreOS的VM並獲得IP

- 前往<http://alpha.release.core-os.net/amd64-usr/current/index.html>，並且下載coreos\_production\_vmware\_insecure.zip。
- 下載回來後解壓可以看到標準VMware格式的 vmx 及 vmdk，及 insecure\_ssh\_key，放入一個固定的地方，如 D:\VM\Coreos 中。



- 直接使用VMware Workstation開啟這個虛擬機，並且將此VM啟動。
- 進入登入界面時，直接按下enter，就會出現IP位置。

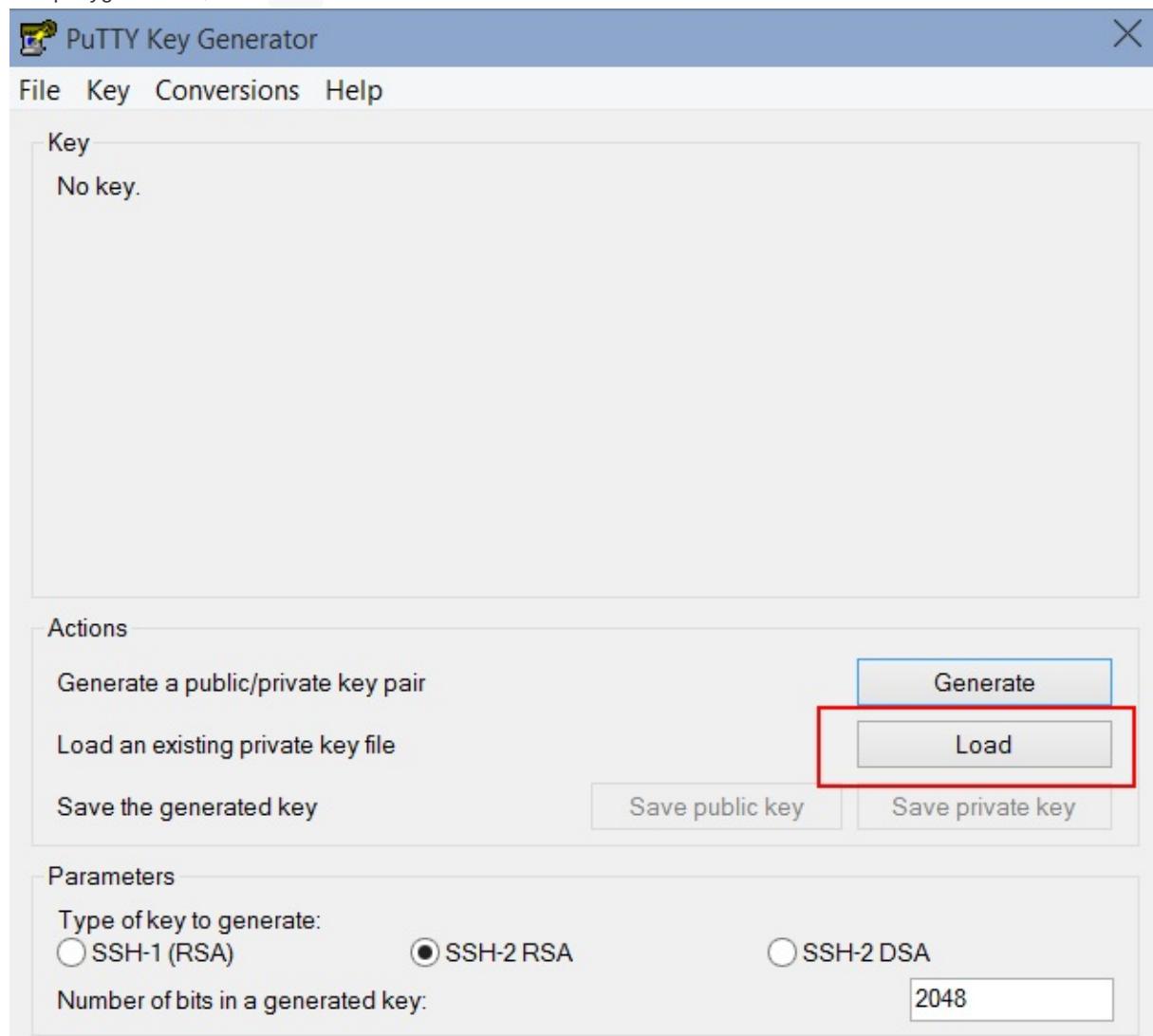


## Windows下使用puttygen轉換insecure\_ssh\_key

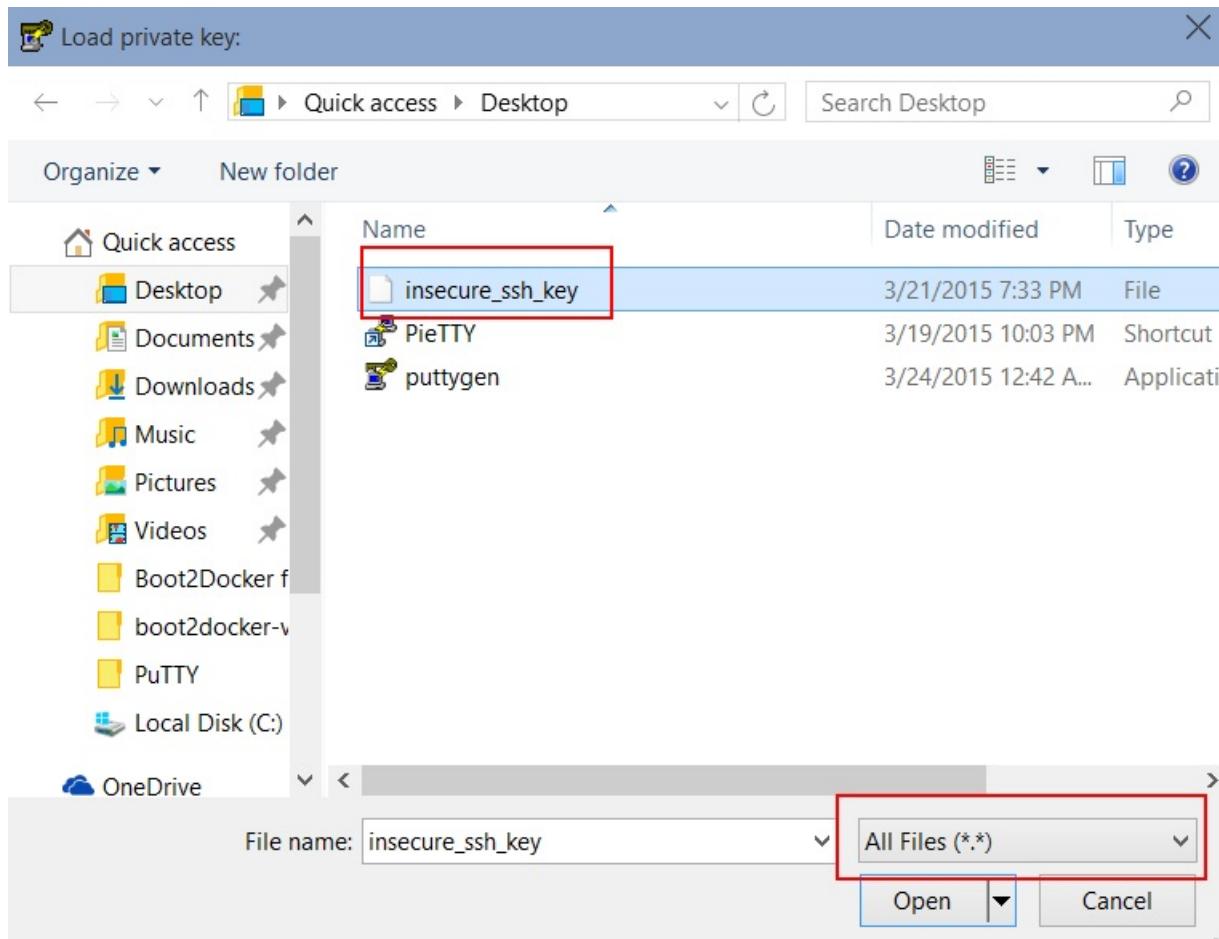
Linux/Mac客戶端直接進入終端命令列操作ssh沒問題，這邊特別嘉惠Windows下的使用者，看如何PieTTY套用ssh key。先下載puttygen.exe吧

先下載Windows專用的puttygen.exe

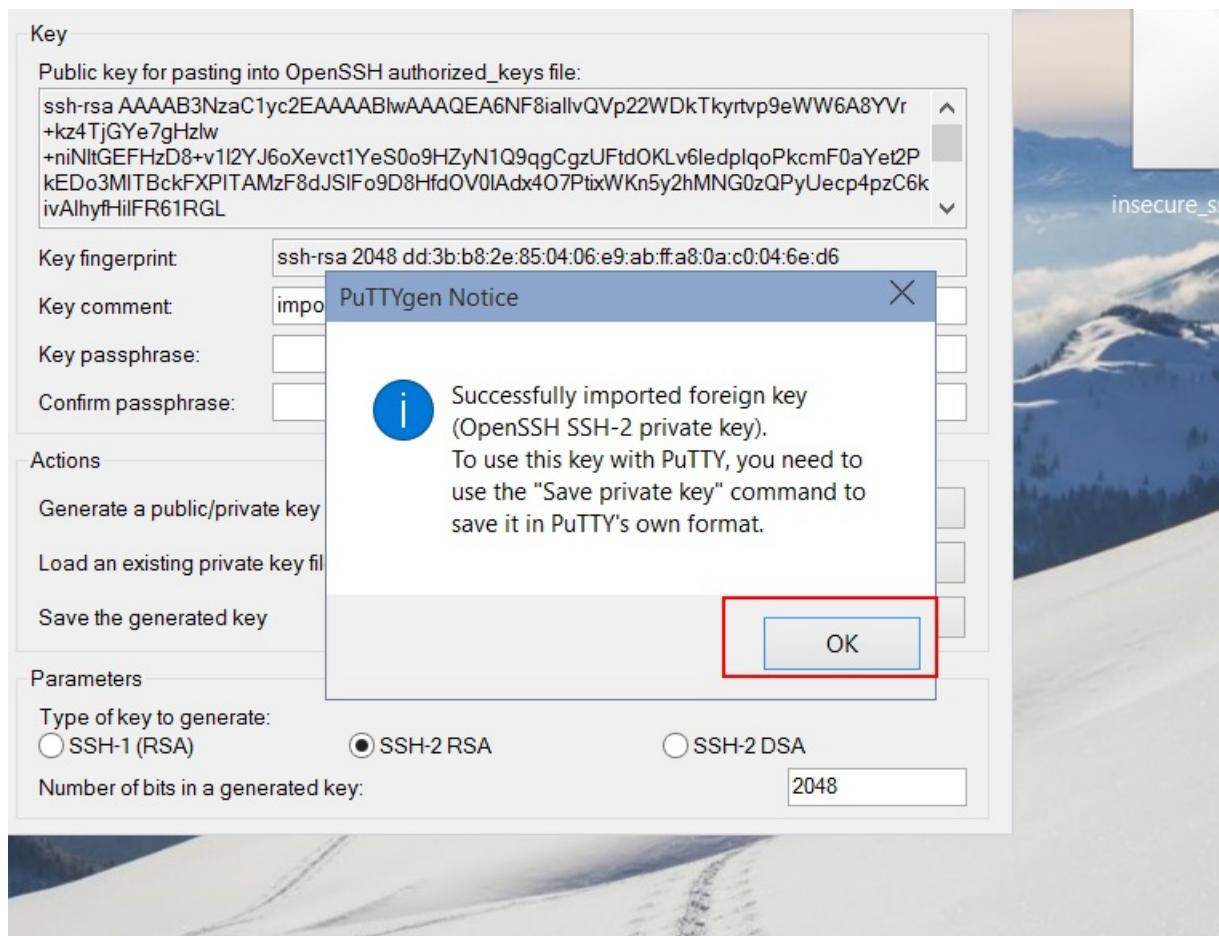
1. 進入puttygen的介面，選擇 Load。



2. 選擇CoreOS資料夾的 `insecure_ssh_key`。



3. 載入成功後，按下 Save private key，將這個key輸出到 \coreos\ 的同目錄下，出現圖中的視窗時直接按Enter，檔名自己取即可。



Key

Public key for pasting into OpenSSH authorized\_keys file:

```
ssh-rsa AAAAB3NzaC1yc2EAAAQEA6NF8iallvQVp22WDkTkyrtvp9eWW6A8YVr  
+kz4TjGYe7gHzlw  
+niNltGEFHzD8+v1l2YJ6oXevct1YeS0o9HZyN1Q9qgCgzUFtdOKLv6ledplqoPkcmF0aYet2P  
kEDo3MITBckFXPITAMzF8dJSIFo9D8HfdOV0lAdx4O7PtxWKn5y2hMNG0zQPyUecp4pzC6k  
ivAlhyfHilFR61RGL
```

Key fingerprint: ssh-rsa 2048 dd:3b:b8:2e:85:04:06:e9:ab:ff:a8:0a:c0:04:6e:d6

Key comment: imported-openssh-key

Key passphrase:

Confirm passphrase:

Actions

Generate a public/private key pair

Load an existing private key file

Save the generated key

Parameters

Type of key to generate:  
 SSH-1 (RSA)  SSH-2 RSA  SSH-2 DSA

Number of bits in a generated key: 2048

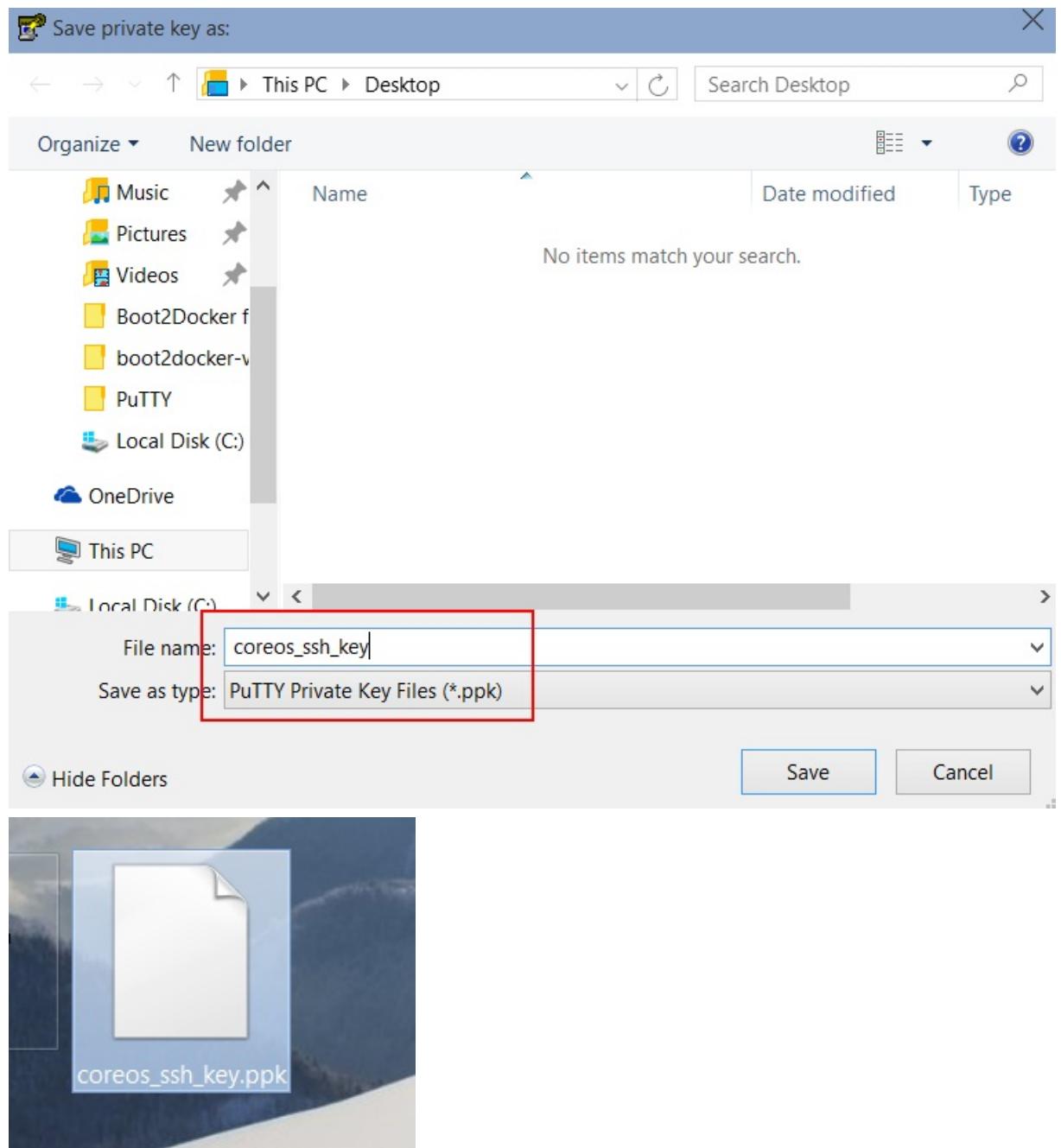


PuTTYgen Warning X



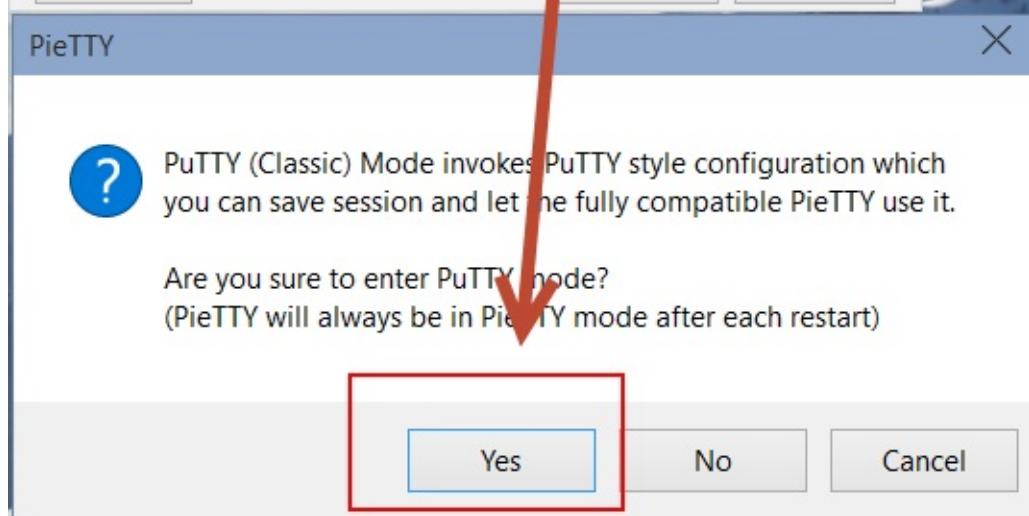
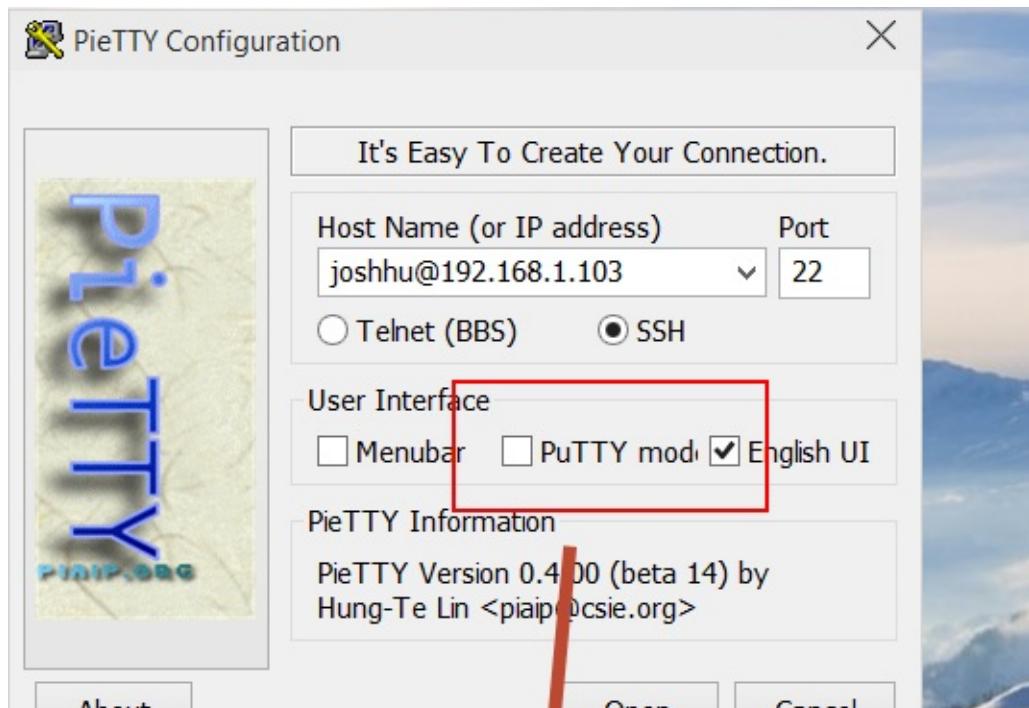
Are you sure you want to save this key  
without a passphrase to protect it?

4. 此時可以看到這個CoreOS專用的Private key已經建立了。

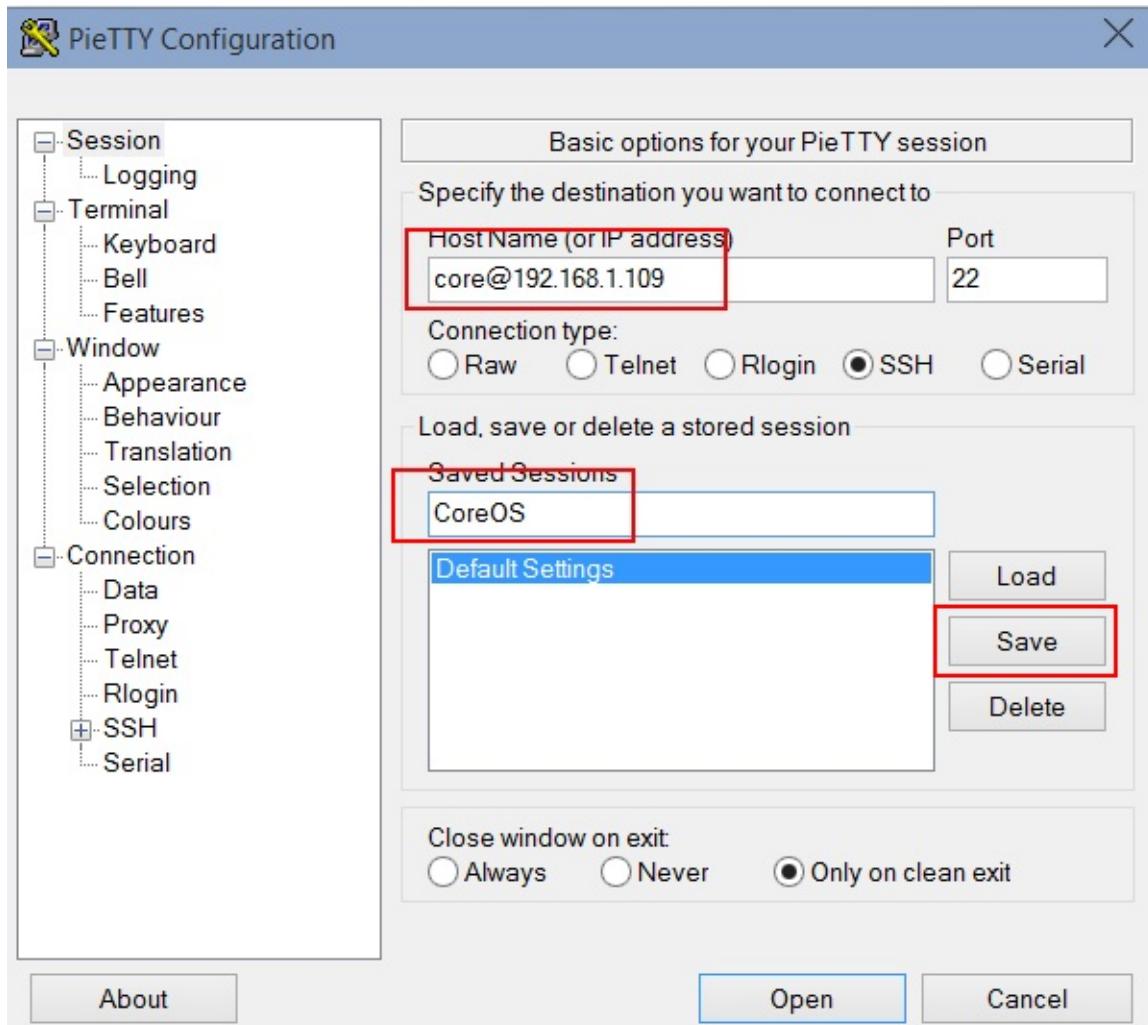


## 使用Private key無帳號密碼ssh登入CoreOS

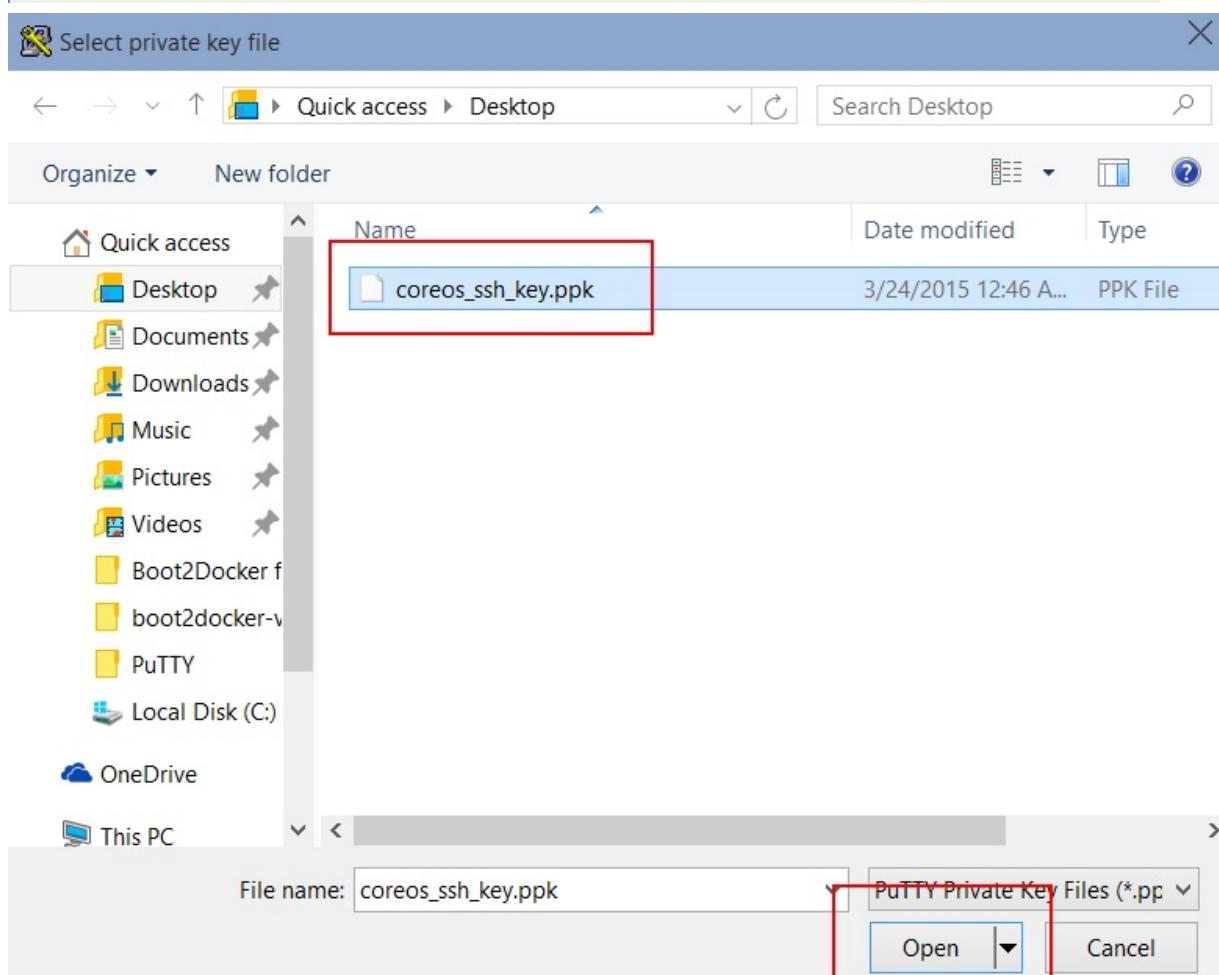
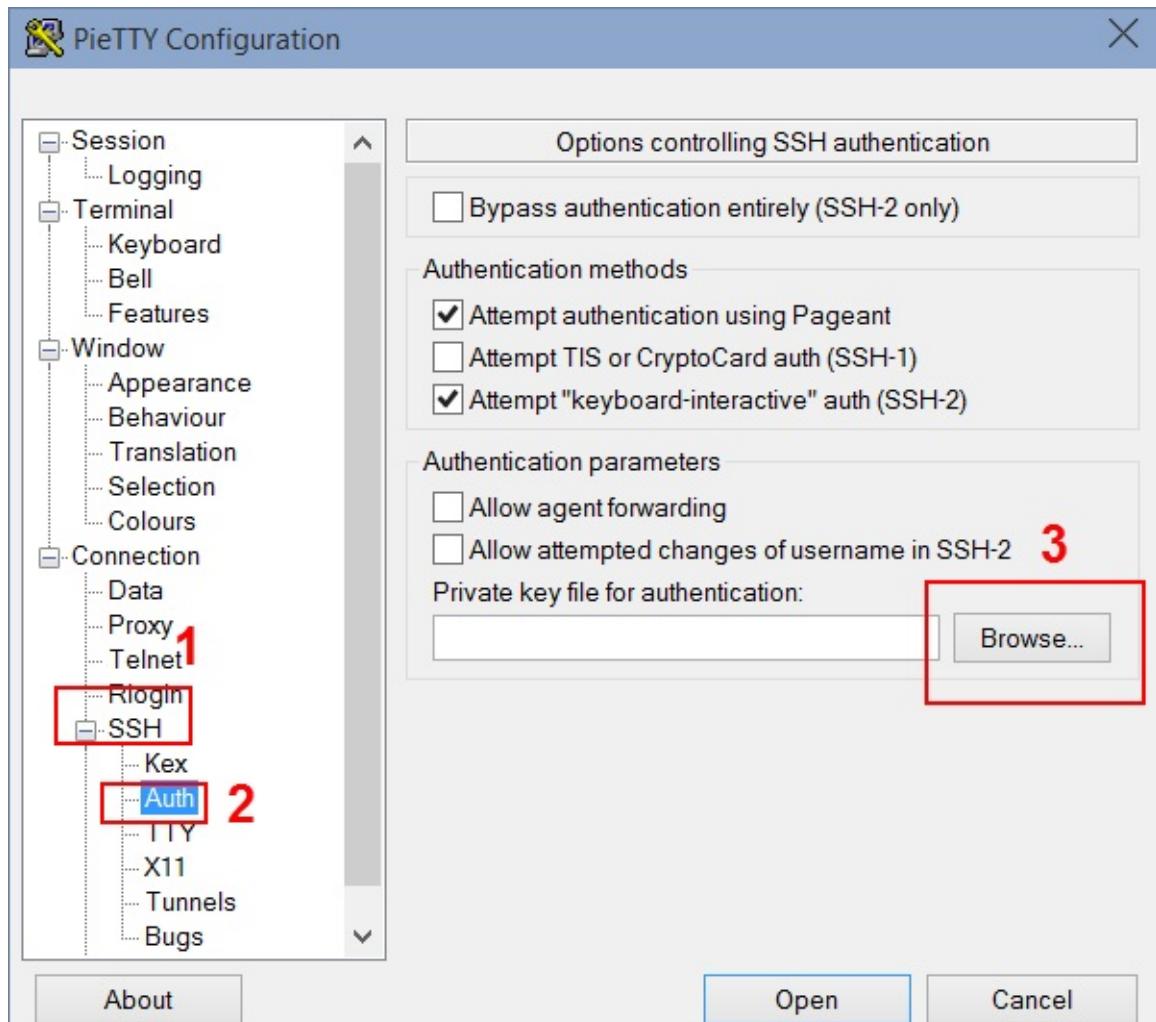
- 首先進入PuTTY或PieTTY的介面，選擇 PuTTY 模式。



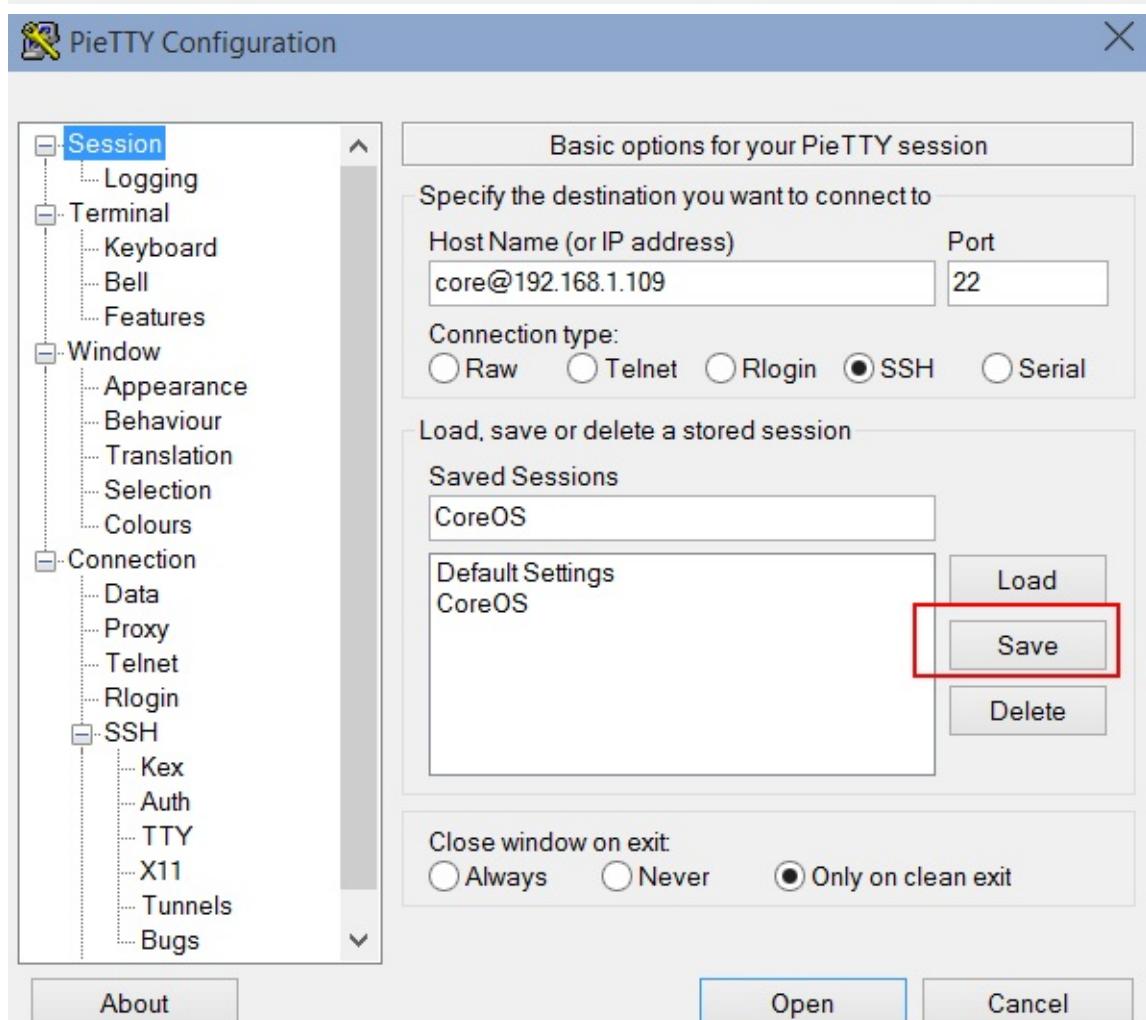
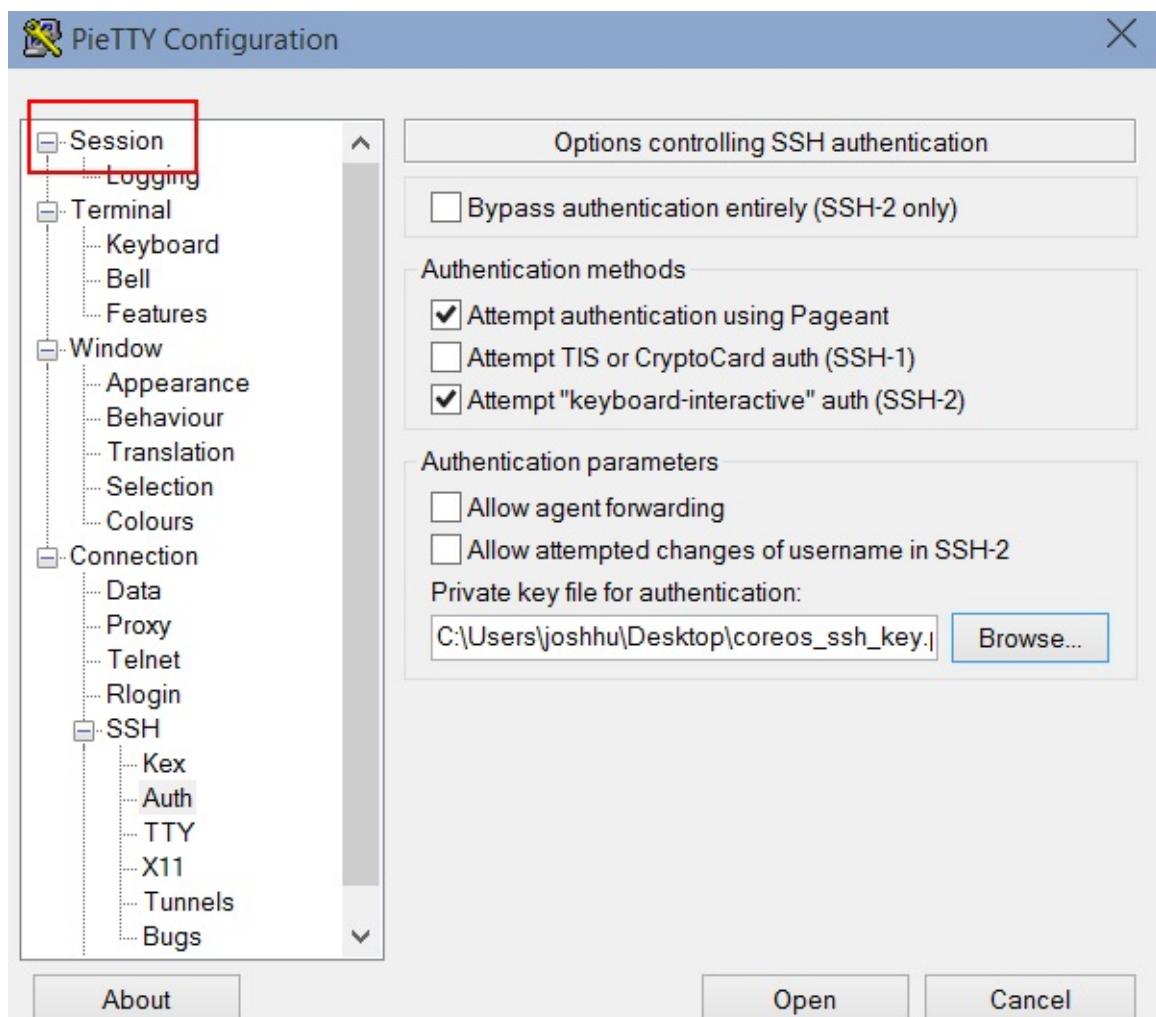
- 輸入剛才開機CoreOS的IP，並且將這個連線存檔為CoreOS，最方便的還是使用標準ssh格式，如 core@192.168.1.108，將使用者名稱放在前面。



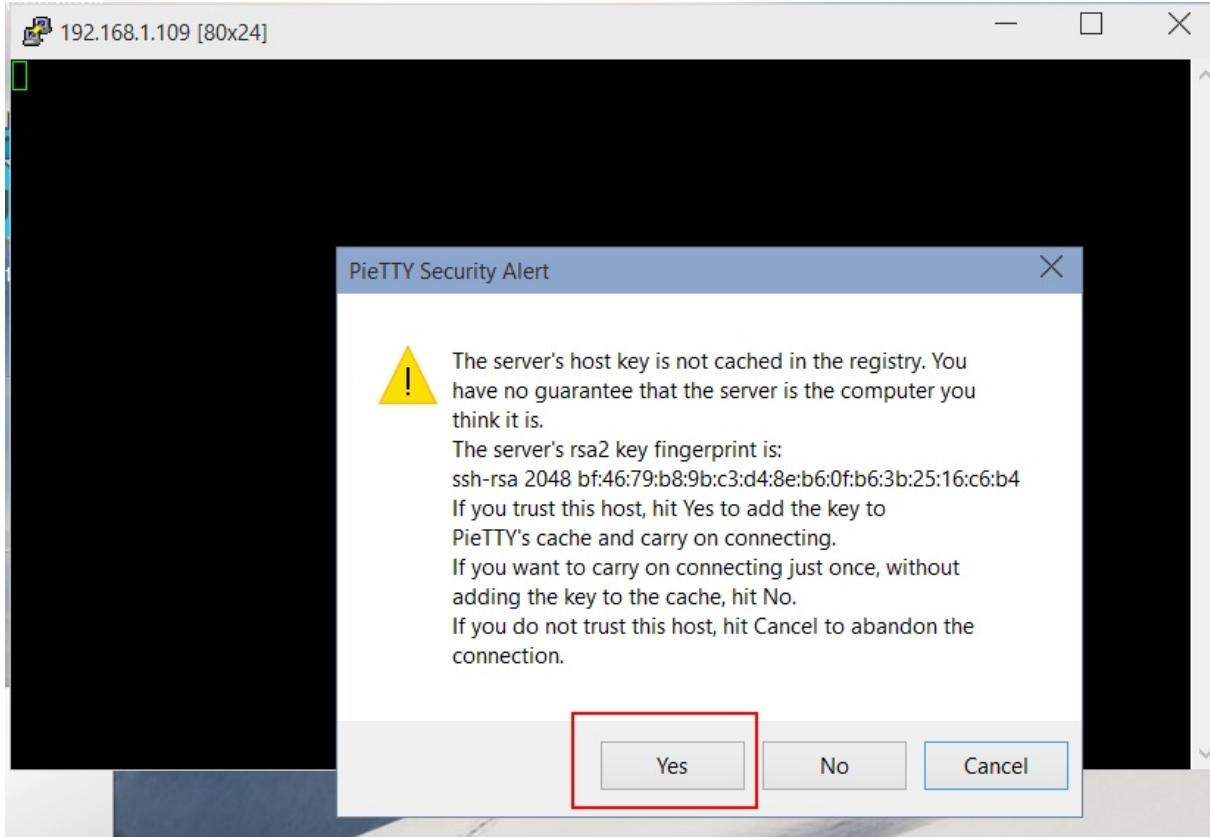
3. 選擇 SSH/Auth 的地方，將剛才產生的key選過來。



4. 選擇 Session，並且別忘了按下 save。



5. 此時按下 open，就可以直接ssh入這個CoreOS而不需要帳號密碼了。第一次連線時會出現圖中的視窗，直接按 Yes。



6. core 這個使用者已經是root權限，因此可以直接使用，不需要 sudo 輸。入 docker version，可以看到docker的版本，而輸入 docker run -d --name web -p 8080:80 joshhu/webdmo，可以直接執行。

```
Using username "core".
Authenticating with public key "imported-openssh-key"
Last login: Tue Mar 24 08:04:47 2015 from 192.168.1.110
CoreOS alpha (626.0.0)
core@localhost ~ $
```

7. 如果看到圖中的網頁畫面，表示這個CoreOS也部署完成了。

```
core@localhost:~ [80x24]
Using username "core".
Authenticating with public key "imported-openssh-key"
Last login: Tue Mar 24 08:07:15 2015 from 192.168.1.110
CoreOS alpha (626.0.0)
core@localhost ~ $ docker run -d --name web -p 8080:80 joshhu/webdemo
Unable to find image 'joshhu/webdemo:latest' locally
Pulling repository joshhu/webdemo
a3574f323972: Pulling image (latest) from joshhu/webdemo, endpoint: https://reqi
a3574f323972: Download complete
511136ea3c5a: Download complete
f3c84ac3a053: Download complete
a1a958a24818: Download complete
9fec74352904: Download complete
d0955f21bf24: Download complete
1214be61bcaa: Download complete
45ad00454734: Download complete
8be3dadcb43a: Download complete
2d4173730925: Download complete
e8f612e3238f: Download complete
d11f03cd837d: Download complete
8abfc77aac61: Download complete
d1dfd80f5ed9: Download complete
Status: Downloaded newer image for joshhu/webdemo:latest
c98a6e540c33becf19bdc10494bcf1d3c8df90fa3517b8325f9d7a0f07e38a47
core@localhost ~ $
```

```
core@localhost:~ [80x24]
Authenticating with public key "imported-openssh-key"
Last login: Tue Mar 24 08:06:46 2015 from 192.168.1.110
CoreOS alpha (626.0.0)
core@localhost ~ $ 
core@localhost ~ $ 
core@localhost ~ $ 
core@localhost ~ $ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS
0fda6c2fbc1b      joshhu/webdemo:latest   "/start.sh"        About a minute ago
go    Up About a minute   0.0.0.0:8080->80/tcp    web
core@localhost ~ $ ifconfig
docker0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500

```

The browser window shows the URL <http://192.168.1.109:8080/>.

## Your First Docker Application

Congratulations! You have just built your first docker service.  
Please visit my books at <https://www.gitbook.com/@joshhu>

PHP version: 5.5.9-1ubuntu4.7  
Apache version: Apache/2.4.7 (Ubuntu)

8. 如果你也想將這個VM部署到vSphere下，請參考下一小節有關轉換到OVF的步驟。

### Mac/Linux下直接進入

在Mac/Linux的終端機視窗中，與 `insecure_ssh_key` 的同目錄下，直接輸入 `ssh -i insecure_ssh_key core@<CoreOS的IP位置>`

如 `ssh -i insecure_ssh_key core@192.168.1.109`

```
joshhu@ubuntu:~$ ssh -i insecure_ssh_key core@192.168.1.109
Last login: Tue Mar 24 08:09:11 2015 from 192.168.1.110
CoreOS alpha (626.0.0)
core@localhost ~ $
```

## CoreOS的docker scripts加入

CoreOS VM中的Docker，由於所有使用者的設定檔都連結到 /usr/share/skel/.bashrc，要先將core這個使用者.bashrc的連結移動到獨立自主的.bashrc，再執行上面的指令，先用core這個使用者名稱ssh連入這個VM，再輸入下列指令：

```
cp ~/.bashrc ~/.bashrc.my
rm ~/.bashrc
mv ~/.bashrc.my ~/.bashrc
wget -P ~ https://github.com/joschu/docker/raw/master/docker_scripts/.bashrc_docker;
echo "[ -f ~/bashrc_docker ] && . ~/bashrc_docker" >> ~/.bashrc; source ~/.bashrc;
wget -P ~ https://github.com/joschu/docker/raw/master/docker_scripts/showmem.coreos;
mv ~/showmem.coreos ~/showmem;
chmod +x ~/showmem;
```

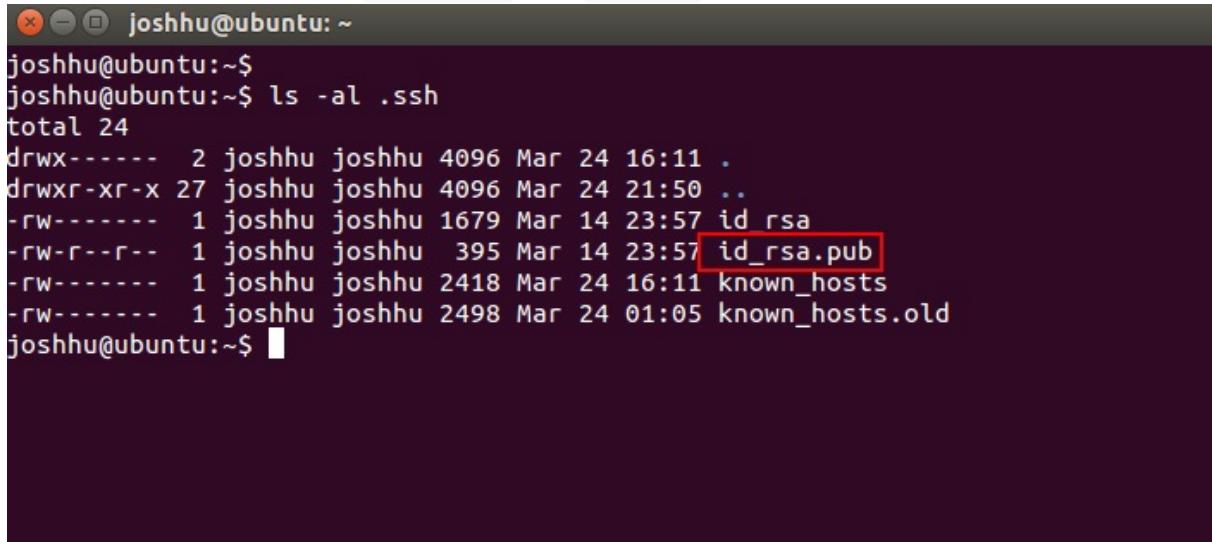
```
core@localhost ~ $
core@localhost ~ $ dip web
172.17.0.5
core@localhost ~ $ dpid web
2683
core@localhost ~ $ denter web
root@38eaa03202cf:~# exit
logout
core@localhost ~ $ ./showmem
web 12MB 128MB
core@localhost ~ $
```

# 更換CoreOS不安全的ssh key

coreos\_production\_vmware\_insecure 的登入方式使用公開的ssh key，任何一個下載同一份 zip 檔案的人，都可以利用相同的 ssh key 登入，十分危險。本小節說明如何把這不安全的ssh key刪除掉，並換成使用者自己安全的ssh key。使用Ubuntu Linux作為ssh的客戶端示範。

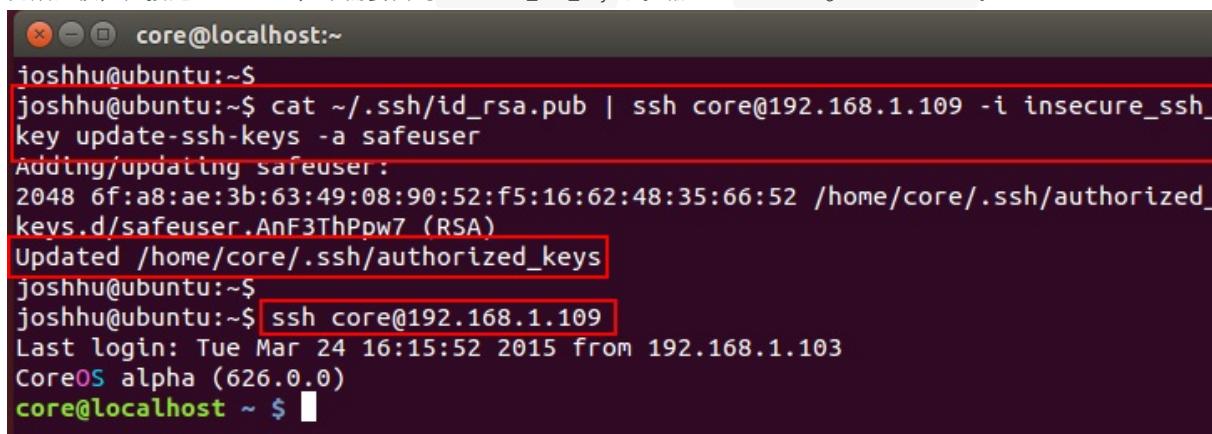
## 換成安全的ssh key

- 先確定你擁有自己的ssh key，輸入 `ls ~/.ssh -al`，要有 `id_rsa.pub` 這個檔案。



```
joshhu@ubuntu:~$ ls -al .ssh
total 24
drwx----- 2 joshhu joshhu 4096 Mar 24 16:11 .
drwxr-xr-x 27 joshhu joshhu 4096 Mar 24 21:50 ..
-rw----- 1 joshhu joshhu 1679 Mar 14 23:57 id_rsa
-rw-r--r-- 1 joshhu joshhu 395 Mar 14 23:57 id_rsa.pub
-rw----- 1 joshhu joshhu 2418 Mar 24 16:11 known_hosts
-rw----- 1 joshhu joshhu 2498 Mar 24 01:05 known_hosts.old
joshhu@ubuntu:~$
```

- 將自己的ssh key加入CoreOS主機中安全ssh key的清單中。CoreOS中有一個更新ssh key的專門script檔 `update-ssh-keys`，用來把自己安全的ssh key更新到CoreOS的主機ssh key允許清單中。假如CoreOS的IP是192.168.1.109，則輸入：`cat ~/.ssh/id_rsa.pub | ssh core@192.168.1.109 -i insecure_ssh_key update-ssh-keys -a safeuser`
- 完成之後，直接進入CoreOS，不需要舊的 `insecure_ssh_key` 了。輸入：`ssh core@192.168.1.109`。



```
joshhu@ubuntu:~$ cat ~/.ssh/id_rsa.pub | ssh core@192.168.1.109 -i insecure_ssh_key update-ssh-keys -a safeuser
Adding/updating safeuser:
2048 6f:a8:ae:3b:63:49:08:90:52:f5:16:62:48:35:66:52 /home/core/.ssh/authorized_keys.d/safeuser.AnF3ThPpw7 (RSA)
Updated /home/core/.ssh/authorized_keys
joshhu@ubuntu:~$ ssh core@192.168.1.109
Last login: Tue Mar 24 16:15:52 2015 from 192.168.1.103
CoreOS alpha (626.0.0)
core@localhost ~ $
```

- 進入CoreOS之後查看目前的key，包括了舊有不安全的，和剛才加入安全的Key。

```
core@localhost:~/ssh/authorized_keys.d
core@localhost ~ $ cd .ssh
core@localhost ~/ssh $ ls -al
total 20
drwx----- 3 core core 4096 Mar 24 16:17 .
drwxr-xr-x 3 core core 4096 Mar 24 14:15 ..
-rw----- 1 core core 849 Mar 24 16:17 authorized_keys
drwx----- 2 core core 4096 Mar 24 16:17 authorized_keys.d
-rw-r--r-- 1 core core 95 Mar 24 12:00 known_hosts
core@localhost ~/ssh $
core@localhost ~/ssh $ cd authorized_keys.d
core@localhost ~/ssh/authorized_keys.d $
core@localhost ~/ssh/authorized_keys.d $ ls -al
total 16
drwx----- 2 core core 4096 Mar 24 16:17 .
drwx----- 3 core core 4096 Mar 24 16:17 ..
-rw----- 1 core core 409 Mar 24 08:03 coreos-cloudinit
-rw----- 1 core core 395 Mar 24 16:17 safeuser
core@localhost ~/ssh/authorized_keys.d $
```

5. 把舊的不安全的key移除，輸入 update-ssh-keys -D coreos-cloudinit 即可。

```
core@localhost:~/ssh/authorized_keys.d
drwx----- 2 core core 4096 Mar 24 16:17 .
drwx----- 3 core core 4096 Mar 24 16:17 ..
-rw----- 1 core core 409 Mar 24 08:03 coreos-cloudinit
-rw----- 1 core core 395 Mar 24 16:17 safeuser
core@localhost ~/ssh/authorized_keys.d $ update-ssh-keys -D coreos-cloudinit
Disabling coreos-cloudinit:
2048 dd:3b:b8:2e:85:04:06:e9:ab:ff:a8:0a:c0:04:6e:d6 /home/core/.ssh/authorized_
keys.d/coreos-cloudinit (RSA)
Updated /home/core/.ssh/authorized_keys
core@localhost ~/ssh/authorized_keys.d $
core@localhost ~/ssh/authorized_keys.d $ ls
coreos-cloudinit safeuser
core@localhost ~/ssh/authorized_keys.d $ ls -al Size 0
total 12
drwx----- 2 core core 4096 Mar 24 16:17 .
drwx----- 3 core core 4096 Mar 24 16:27 ..
-rw----- 1 core core 0 Mar 24 16:27 coreos-cloudinit
-rw----- 1 core core 395 Mar 24 16:17 safeuser
core@localhost ~/ssh/authorized_keys.d $
```

6. 此時用原來舊的KEY已經進不去了。

```
chtti@ubuntu14: ~
chtti@ubuntu14:~$ ssh -i insecure_ssh_key core@192.168.1.109
The authenticity of host '192.168.1.109 (192.168.1.109)' can't be established.
ED25519 key fingerprint is ec:c1:b2:09:ae:51:24:94:3e:37:f1:de:4d:51:0e:00.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.1.109' (ED25519) to the list of known hosts.
core@192.168.1.109's password:
```

password needed

7. 如果你的CoreOS在重新開機後IP變了，你可能會進不去了，會出現如下的畫面

```
joshhu@ubuntu:~$ ssh core@192.168.1.109
@@@@@@@@@@@0000000000000000000000000000000000000000000000000000000000000000000000
@     WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!      @
@@@@@@@@@@@0000000000000000000000000000000000000000000000000000000000000000000000
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that a host key has just been changed.
The fingerprint for the ED25519 key sent by the remote host is
5e:a8:6e:7f:36:21:8c:1d:24:c7:37:0d:df:96:44:ae.
Please contact your system administrator.
Add correct host key in /home/joshhu/.ssh/known_hosts to get rid of this message.
Offending ECDSA key in /home/joshhu/.ssh/known_hosts:1
remove with: ssh-keygen -f "/home/joshhu/.ssh/known_hosts" -R 192.168.1.109
ED25519 host key for 192.168.1.109 has changed and you have requested strict checking.
Host key verification failed.
```

此時只要輸入 `ssh-keygen -f "/home/joshhu/.ssh/known_hosts" -R 192.168.1.109` 來更新ssh key，就可以正常登入了。

# 生產環境vSphere部署CoreOS

如果你要將CoreOS部署到vSphere上，由於vSphere需要對其上的每一個VM都進行嚴格的控管，因此所有在vSphere下的VM都必須安裝VMware Tools。如此一來，你就必須下載已經預先安裝好VMware Tools的CoreOS的VMware Image。

下載網址為

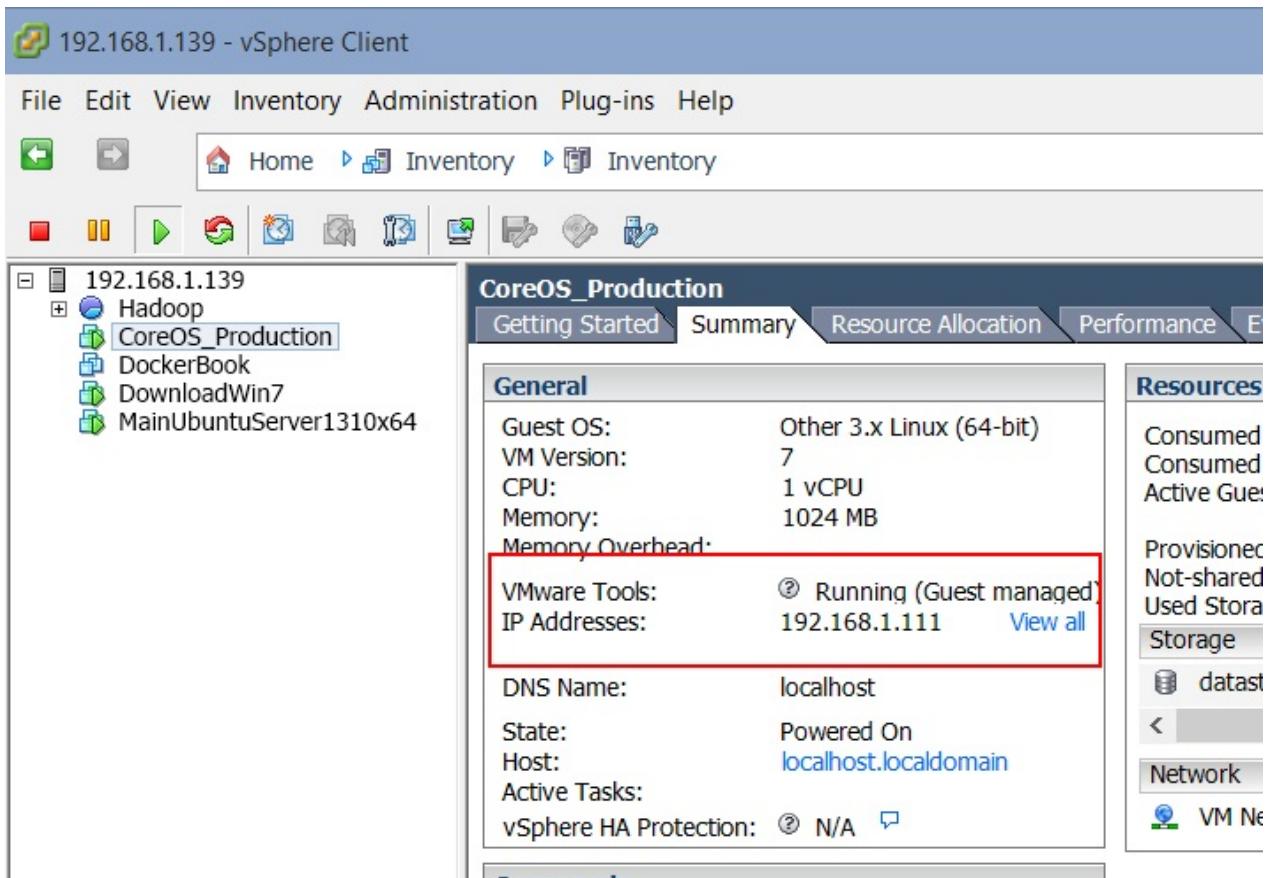
[coreos\\_production\\_vmware\\_image.vmdk.bz2](#)(要先解壓)  
[coreos\\_production\\_vmware.vmx](#)

下載回來有兩個檔案，一個是 `vmx`，另一個則是壓縮後的 `vmdk`。使用前必須先將這個VM轉換成OVF格式，Windows/Mac下使用VMware Workstation/Fusion內建程式，Linux下則可以至VMware的官網下載OVF Tool來轉換，下載網址為

<https://developercenter.vmware.com/web/dp/tool/ovf/3.5.2>

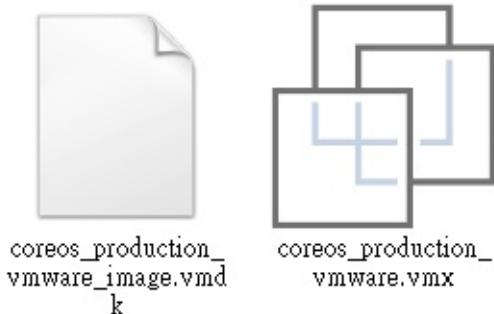
此版本的CoreOS雖預先安裝了vSphere的VMware Tools，但預設沒有使用者帳號密碼，如果要使用，必須透過雲端工具或設定開機參數，下面就是完整的部署步驟。

預先安裝了VMware Tools才能在開機後回報IP

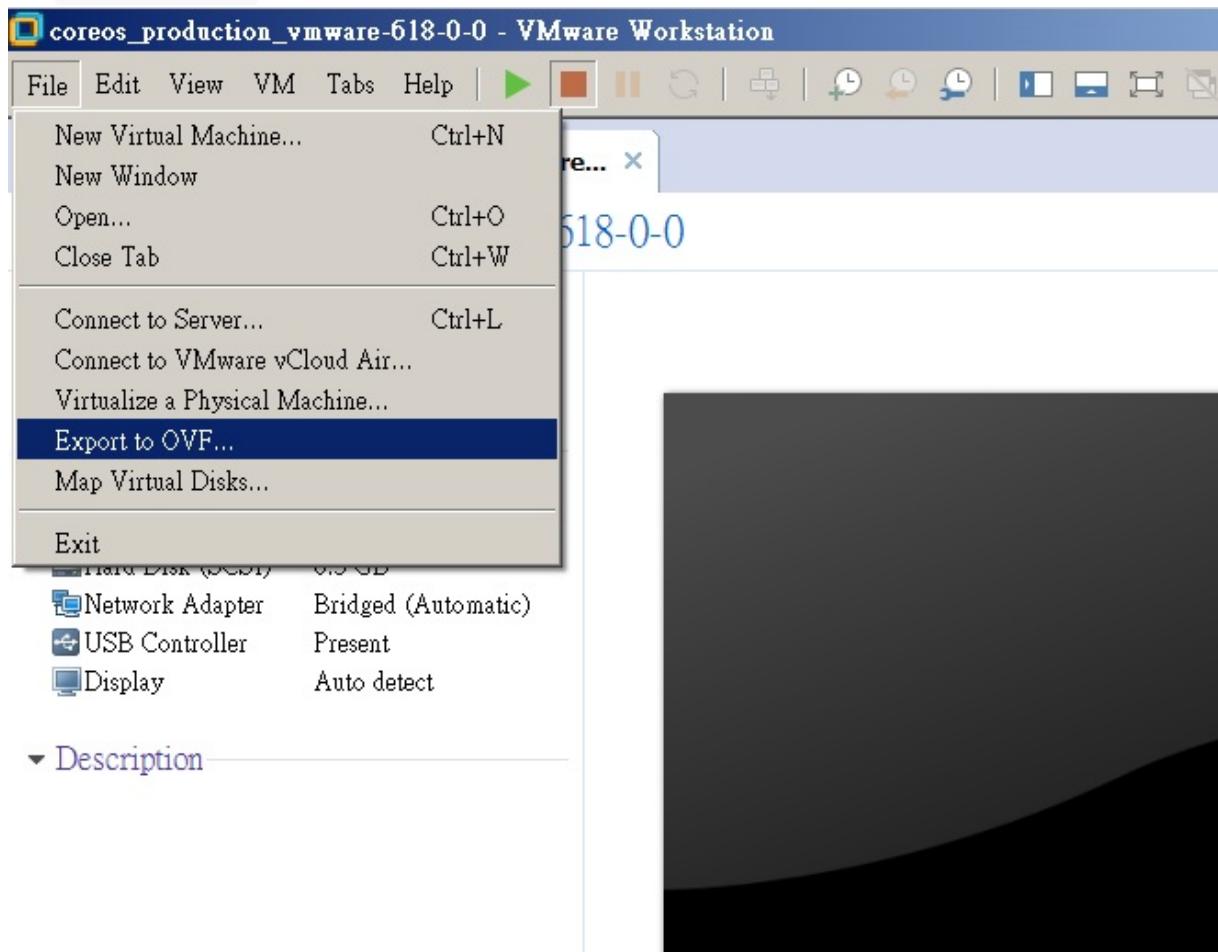


## 在vSphere下部署雲端版的CoreOS

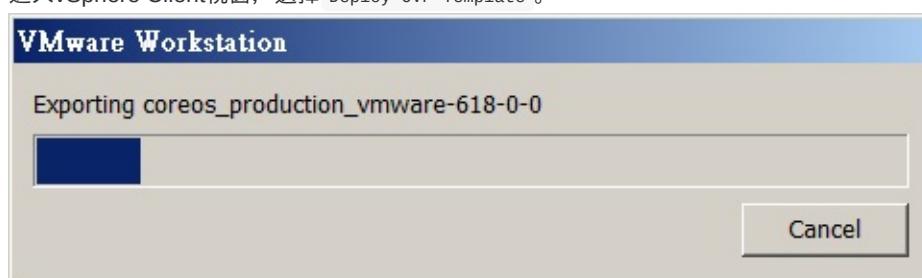
1. 將下載回來的 `bz2` 檔案解壓，內容是一個 `vmdk` 檔案。



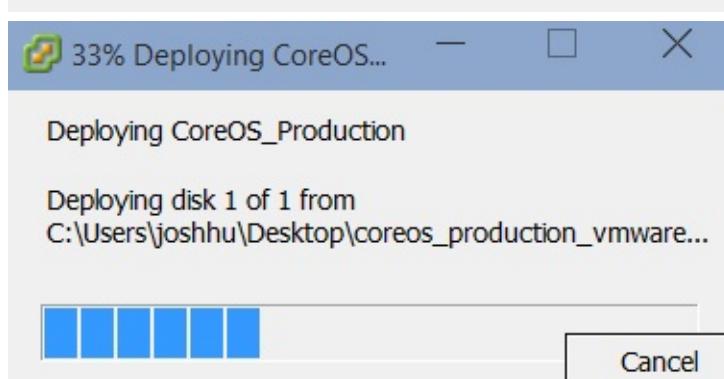
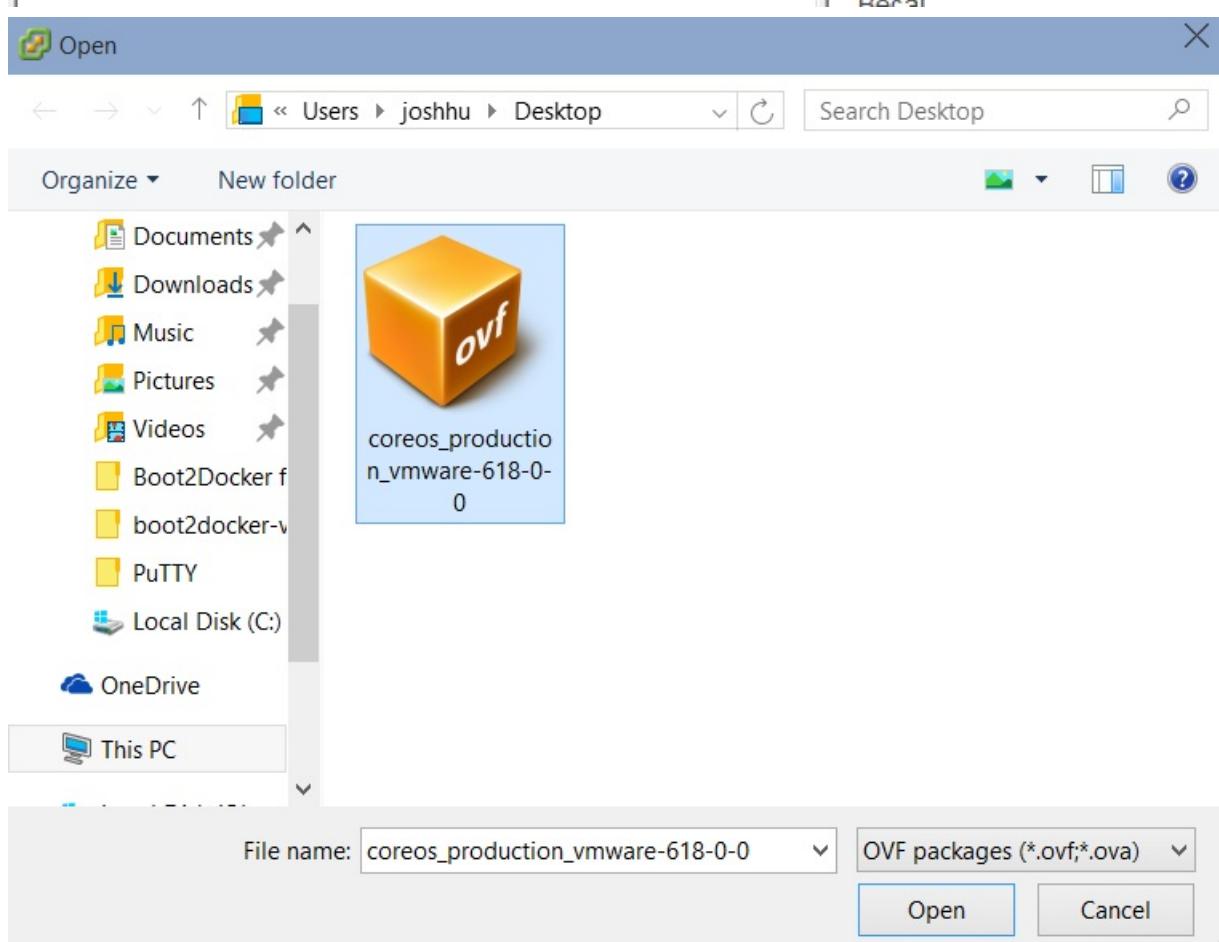
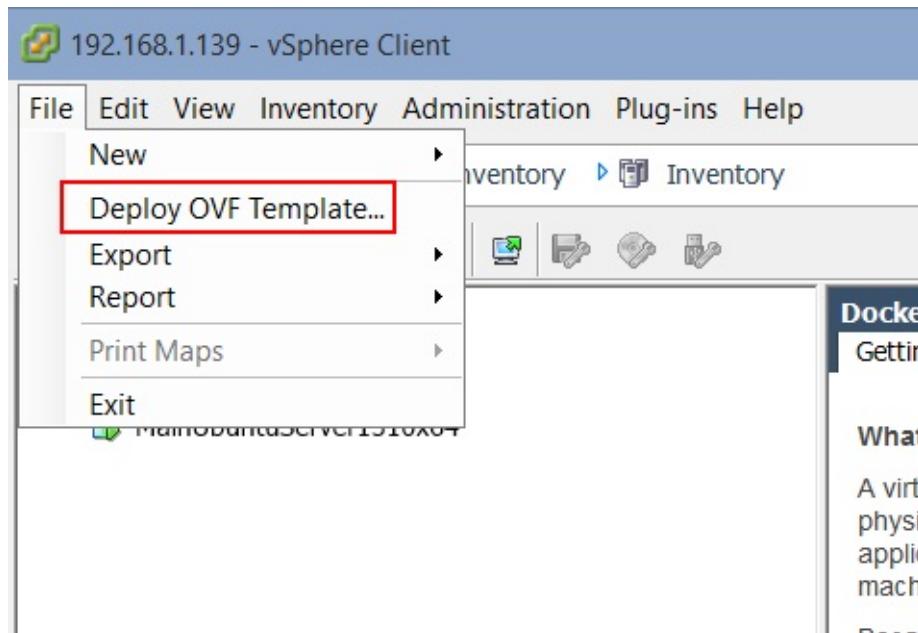
2. 使用VMware Workstation打開這個 vmx 。
3. 選擇 File/Export to OVF , 並且儲存在確定位置。



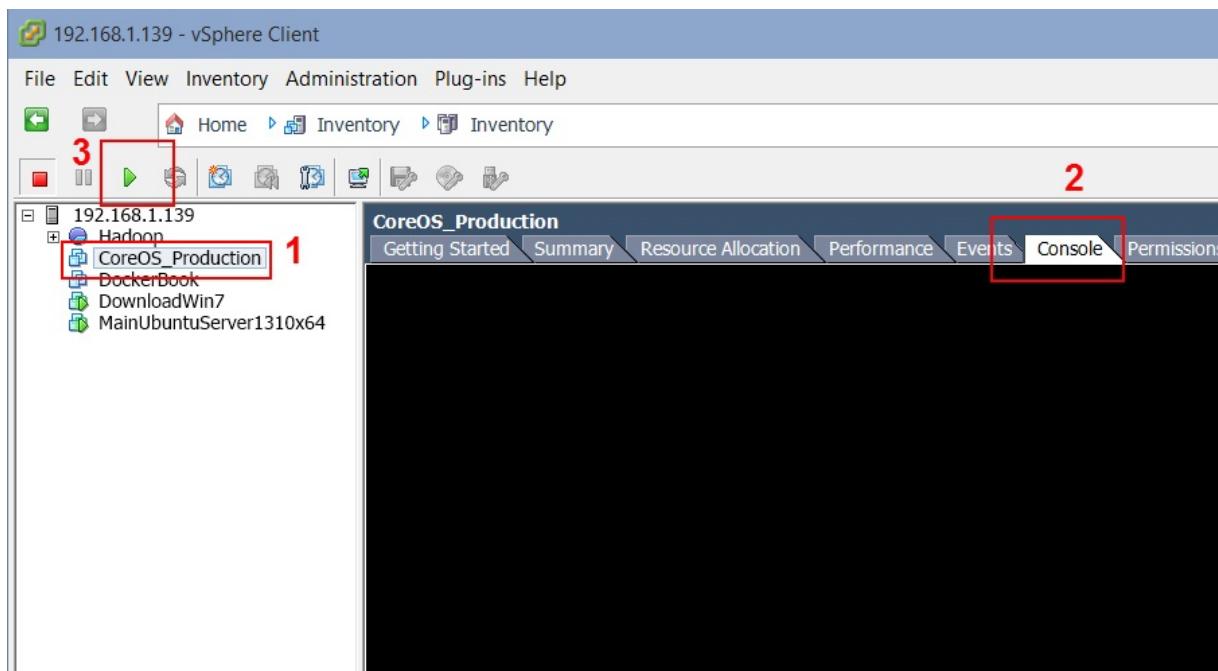
4. 進入vSphere Client視窗，選擇 Deploy OVF Template 。



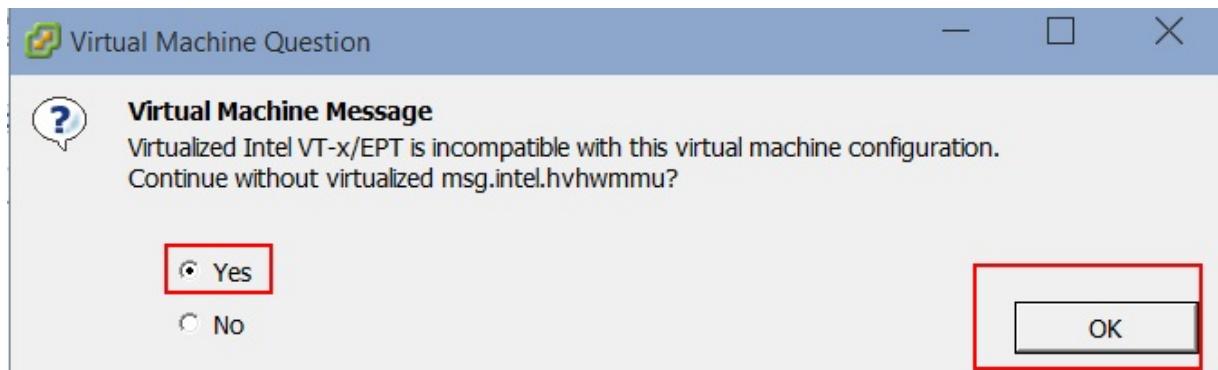
5. 此時會出現匯入視窗，一直按「下一步」即可，最後會在vSphere的列表中出現該VM，直接啟動即可。



6. 可以看到這個VM已經預設安裝好VMware Tools了。

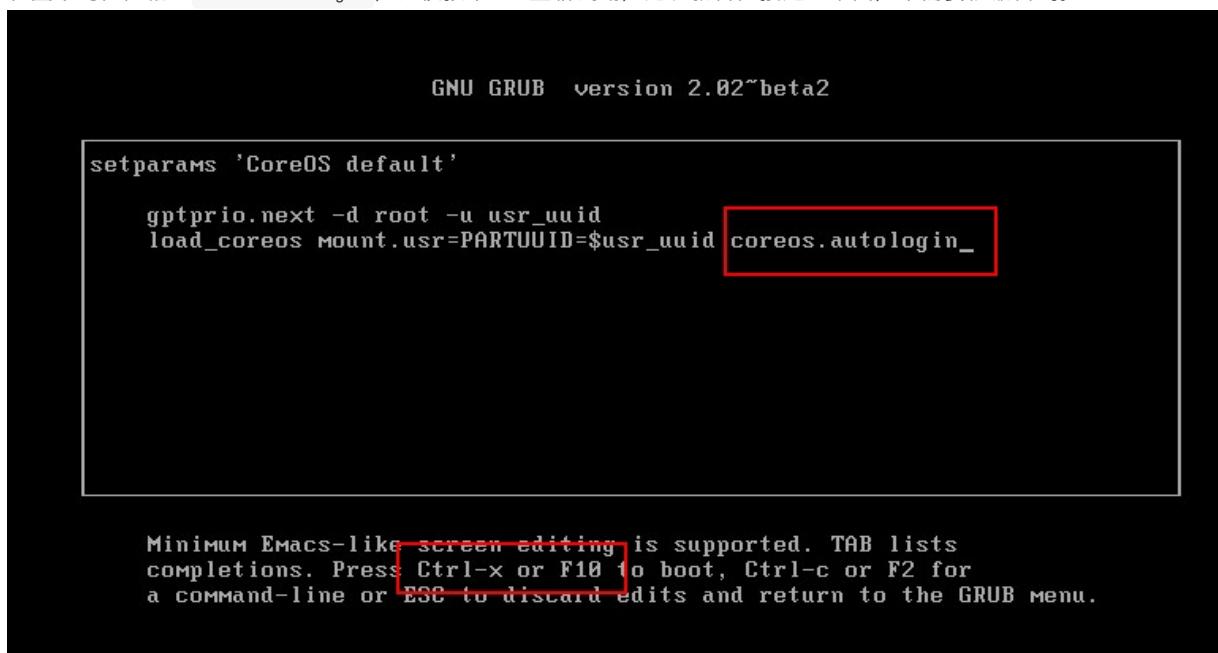


7. 啟動這個VM，並且快速進入主控介面。



8. 在還沒完全啟動前，請快速按下 e 鍵，直到啟動選項出現為止。

9. 在圖中的位置輸入 coreos.autologin，之後按下F10重新啟動，這次就會直接進入桌面，不需要帳號密碼。



## CoreOS\_Production

Getting Started Summary Resource Allocation Performance Events Console Permissions

```
This is localhost (Linux x86_64 3.19.0) 15:00:51
SSH host key: 30:35:c1:3a:eb:56:7f:14:7a:40:bb:84:f6:6d:ae:f0 (DSA)
SSH host key: 71:90:9b:66:b1:38:98:ec:74:b6:7a:4b:56:30:ed:1f (ED25519)
SSH host key: b0:20:11:ef:65:ec:70:b6:00:a7:2a:d2:21:fa:13:b3 (RSA)
ens192: fe80::20c:29ff:fef0:d64d

localhost login: core (automatic login)
CoreOS alpha (618.0.0)
[ 4.698329] sda9: WRITE SAME failed. Manually zeroing.
core@localhost ~ $ _
```

10. 在這個VMware中輸入 sudo passwd core 來建立帳號密碼，下次才可正常登入，你也可以輸入 sudo passwd root 來建立root的帳號密碼方便操作docker。

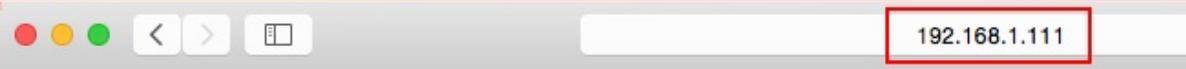
```
core@localhost ~ $ 
core@localhost ~ $ sudo passwd core
Changing password for core
Enter the new password (minimum of 5 characters)
Please use a combination of upper and lower case letters and numbers.
New password:
Re-enter new password:
passwd: password changed.
core@localhost ~ $ _
```

11. 可以從 docker version 或 docker info 查看。

```
core@localhost ~ $ 
core@localhost ~ $ docker version
Client version: 1.5.0
Client API version: 1.17
Go version (client): go1.3.3
Git commit (client): a8a31ef-dirty
OS/Arch (client): linux/amd64
[ 80.917688] bridge: automatic filtering via arp/ip/ip6tables has been deprecated. Update your scripts to load br_netfilter if you need this.
[ 80.924953] IPv6: ADDRCONF(NETDEV_UP): docker0: link is not ready
[ 80.938799] ip_tables: (C) 2000-2006 Netfilter Core Team
[ 80.960203] nf_conntrack version 0.5.0 (7983 buckets, 31932 max)
Server version: 1.5.0
Server API version: 1.17
Go version (server): go1.3.3
Git commit (server): a8a31ef-dirty
core@localhost ~ $ _
```

12. 一樣輸入 docker run -d --name web -p 8080:80 joshhu/webdemo 來查看安裝是否成功。

```
web
core@localhost ~ $ docker run -d --name web -p 8080:80 joshhu/webdemo
75b3a0fbaefb9ca4a8ef68c48105a1f48de4e61f21772aadb3a08881d47cfbf
[ 323.465273] device vethae0bc49 entered promiscuous mode
[ 323.467382] IPv6: ADDRCONF(NETDEV_UP): vethae0bc49: link is not ready
[ 323.481334] eth0: renamed from veth58934cf
[ 323.482878] IPv6: ADDRCONF(NETDEV_CHANGE): vethae0bc49: link becomes ready
[ 323.484307] docker0: port 1(vethae0bc49) entered forwarding state
[ 323.485421] docker0: port 1(vethae0bc49) entered forwarding state
core@localhost ~ $ [ 338.531597] docker0: port 1(vethae0bc49) entered forwardin
g state
```



Congratulations! You have just built your first docker service.  
Please visit my books at <https://www.gitbook.com/@joshhu>

PHP version: 5.5.9-1ubuntu4.7  
Apache version: Apache/2.4.7 (Ubuntu)

# 在公有雲端平台上安裝Docker

---

目前CoreOS中的Docker支援大部分的雲端平台，你可以到<http://alpha.release.core-os.net/amd64-usr/current/>查看所有可佈署的VM檔案，主要的有：

- 大部分的Hypervisor(VMware, VirtualBox, Xen, QEMU, Hyper-V等)
- Microsoft Azure
- BrightBox
- CloudSigma
- CloudStack
- DigitalOcean
- exoScale
- OpenStack/RackSpace
- Vagrant
- Amazon

當然大部分的IDC也可部署，如筆者自己使用的GoDaddy就可在其Ubuntu 14.04的VPS中直接安裝Docker使用，但安裝時要注意使用VPS的Linux版本，再照著不同Linux進行安裝。

有關常見雲端平台的安裝，我們會在本書稍後關於Docker叢集部署的章節有詳細說明。