

一、動機與目的：

本文的目的是在 Ubuntu 系統下使用 KVM 建立虛擬機器，再使用這些虛擬機器建構一個 Kubernetes 環境的手動安裝範例。

系統環境設定如下：

實體主機： k8s-host 192.168.100.90	虛擬主機： master1 192.168.100.91	虛擬主機： node1 192.168.100.92	虛擬主機： node2 192.168.100.93
KVM	Docker ETCD CA & Certificates API Server Front proxy Bootstrap Admin Controller manager Scheduler Adding Kubelet	Docker CA & Certificates Kubernetes CNI	Docker CA & Certificates Kubernetes CNI

1.1 在實體主機上安裝 Ubuntu 16.04 作業系統：

使用 ubuntu-16.04.3-desktop-amd64.iso 檔案安裝作業系統。

主機名稱：k8s-host

用戶帳號：user

1.2 建立 SSH 連線機制：

執行下列指令，安裝、啟用和查詢 SSH 服務：

```
$ sudo apt-get install openssh-server
$ sudo service ssh restart
$ sudo service ssh status
```

1.3 將用戶帳號設定為 sudo no password：

執行下列指令：

```
$ sudo visudo
```

將下列內容：

```
# Allow members of group sudo to execute any command
%sudo    ALL=(ALL:ALL) ALL
```

改為：

```
# Allow members of group sudo to execute any command
%sudo    ALL=(ALL:ALL) NOPASSWD:ALL
```

儲存並關閉檔案。

1.4 網路組態設定：

執行 ifconfig 指令查詢網路組態設定，如圖 1。

```
$ ifconfig
enp0s3  Link encap:Ethernet  HWaddr 08:00:27:e3:56:cf
        inet addr:10.10.1.174  Bcast:10.10.1.255  Mask:255.255.255.0
        inet6 addr: fe80::acfe:4970:3abc:311f/64 Scope:Link
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
        RX packets:12914 errors:0 dropped:1455 overruns:0 frame:0
        TX packets:75 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1000
        RX bytes:808444 (808.4 KB)  TX bytes:8574 (8.5 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:36 errors:0 dropped:0 overruns:0 frame:0
        TX packets:36 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1000
        RX bytes:2823 (2.8 KB)  TX bytes:2823 (2.8 KB)
```

圖 1 動態位址 k8s 主機網路組態設定

從圖 1 中，可以知道此主機的網卡名稱為 enp0s3。

接著編輯 /etc/network/interfaces 檔案，以設定主機的 IP 位址。

原始網路設定內容：

```
$ cat /etc/network/interfaces
```

```
# interfaces(5) file used by ifup(8) and ifdown(8)
auto lo
iface lo inet loopback
```

將 /etc/network/interfaces 內容改成下列文字：

```
# interfaces(5) file used by ifup(8) and ifdown(8)
auto lo
iface lo inet loopback

auto enp0s3
iface enp0s3 inet static
    address 192.168.100.90
    netmask 255.255.255.0
    gateway 192.168.100.254
    dns-nameservers 8.8.8.8 8.8.4.4
```

重啟作業系統：

```
$ sudo reboot
```

檢查新的網路組態設定：

```
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:e3:56:cf
          inet addr:10.10.1.90  Bcast:10.10.1.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fee3:56cf/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:2855 errors:0 dropped:2641 overruns:0 frame:0
          TX packets:75 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:184496 (184.4 KB)  TX bytes:8909 (8.9 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:8 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:396 (396.0 B)  TX bytes:396 (396.0 B)
```

圖 2 固定位址 k8s 主機網路組態設定

二、虛擬主機的安裝與設定：

2.1 KVM 安裝：

2.1.1 檢查是否支援硬體虛擬化技術

要在 Linux 系統中執行 KVM，必須先確認處理器是否支援硬體虛擬化技術。確認的方法是執行下列指令：

```
$ egrep -c '(vmx|svm)' /proc/cpuinfo
```

出現數字 0 表示 CPU 不支援硬體虛擬化技術，大於 0 則表示支援硬體虛擬化技術。

除了確認 CPU 是否支援支援硬體虛擬化技術外，同時必須確認 BIOS 中的硬體虛擬化技術功能是否已經啟用，如果未啟用，那麼還是無法執行硬體虛擬化技術。

如果想確認 CPU 是否為 64 位元，可以執行下列指令：

```
$ egrep -c 'lm' /proc/cpuinfo
```

出現 0 表示非 64 位元 CPU，大於 0 則表示是 64 位元 CPU。

如果想要知道 kernel 是否為 64 位元，可以執行下列指令：

```
$ sudo uname -m
```

出現 x86_64 表示是 64-bit 核心。出現 i386, i486, i586 or i686 則表示是 32-bit 的核心。

2.1.2 安裝 KVM

安裝 KVM：

```
$ sudo apt-get update && sudo apt-get upgrade
```

```
$ sudo apt-get install qemu-kvm libvirt-bin ubuntu-vm-builder bridge-utils
```

將使用者帳號加入 kvm 群組：

```
$ sudo adduser `id -un` kvm
```

```
Adding user `user' to group `kvm' ...
Adding user user to group kvm
Done.
```

確認使用者帳號已隸屬於 kvm 和 libvirt 群組：

```
$ groups user
```

```
user : user adm cdrom sudo dip plugdev lpadmin sambashare kvm libvirt
```

重新開機使群組設定立刻生效。

```
$ sudo reboot
```

KVM 安裝確認，如果出現下列訊息則表示成功。

```
$ kvm-ok
```

```
INFO: /dev/kvm exists  
KVM acceleration can be used
```

如果出現下列訊息，表示 CPU 不支援硬體虛擬化技術，但是還是可以執行 KVM，但是速度會變慢。

```
INFO: Your CPU does not support KVM extensions  
INFO: For more detailed results, you should run this as root  
HINT:  sudo /usr/sbin/kvm-ok
```

執行 virsh 指令，如果出現下列訊息則表示安裝成功：

```
$ virsh list --all
```

Id	Name	State

2.1.3 設定 KVM 的網路組態設定

因為安裝了 KVM，所以必須修改 /etc/network/interfaces 的內容讓網路可以正常運作，執行指令如下：

```
$ sudo nano /etc/network/interfaces
```

修改後內容如下：

```
# interfaces(5) file used by ifup(8) and ifdown(8)  
auto lo  
iface lo inet loopback  
  
auto enp0s3  
iface enp0s3 inet manual  
  
# Bridge  
auto br0  
iface br0 inet static  
    address 192.168.100.90  
    netmask 255.255.255.0
```

```
gateway 192.168.100.254
dns-nameservers 8.8.8.8 8.8.4.4
bridge_ports enp0s3
bridge_stp off
bridge_fd 0
bridge_maxwait 0
```

編輯完成後重啟主機，並執行 `ifconfig` 指令確定網路組態設定。

```
br0      Link encap:Ethernet  HWaddr 08:00:27:e3:56:cf
          inet addr:10.10.1.90  Bcast:10.10.1.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fee3:56cf/64  Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:16191 errors:0 dropped:14017 overruns:0 frame:0
          TX packets:121 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:879852 (879.8 KB)  TX bytes:16128 (16.1 KB)

enp0s3   Link encap:Ethernet  HWaddr 08:00:27:e3:56:cf
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:17868 errors:0 dropped:1234 overruns:0 frame:0
          TX packets:133 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1216140 (1.2 MB)  TX bytes:17489 (17.4 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:14 errors:0 dropped:0 overruns:0 frame:0
          TX packets:14 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:630 (630.0 B)  TX bytes:630 (630.0 B)

virbr0    Link encap:Ethernet  HWaddr 00:00:00:00:00:00
          inet addr:192.168.122.1  Bcast:192.168.122.255  Mask:255.255.255.0
          UP BROADCAST MULTICAST  MTU:1500  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:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

圖 3 安裝 KVM 後的 k8s 主機網路組態設定

2.2 建立虛擬主機

2.2.1 安裝 `virt-manager`：

使用下列指令安裝 `virt-manager` 套件：

```
$ sudo apt-get update && sudo apt-get upgrade  
$ sudo apt-get install -y virt-manager
```

2.2.2 下載 Ubuntu ISO 檔案：

下載桌機版 Ubuntu 16.04 的指令如下：

```
$ mkdir -p ~/kvm/images  
$ cd ~/kvm/images  
$ wget http://releases.ubuntu.com/16.04.4/ubuntu-16.04.4-desktop-amd64.iso
```

下載伺服器版 Ubuntu 16.04 的指令如下：

```
$ wget http://releases.ubuntu.com/16.04.4/ubuntu-16.04.4-server-amd64.iso
```

2.2.3 建立虛擬主機：

執行下列指令，將會出現 Virtual Machine Manager 視窗，如圖 4。

```
$ virt-manager
```

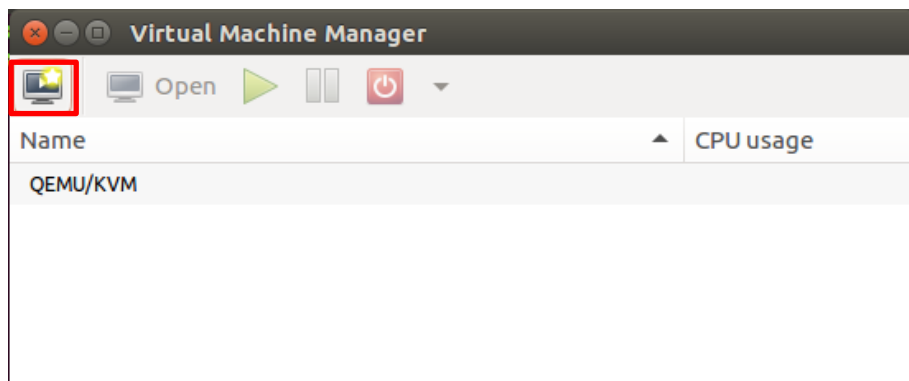


圖 4 virt-manager

在圖 4 中，按下左上角的 Create a new virtual machine 按鈕，就會出現 New VM 視窗，如圖 5。在 New VM 視窗中共有 5 個步驟要執行：

- A. 在圖 5 中，點選 Local install media (ISO image or CD ROM) 選項後，按下 Forward 按鈕，出現步驟 2 的視窗。如圖 6。

- B. 在圖 6 中，點選 Use ISO image 選項，並按下 Browse 按鈕後，出現 Choose Storage Volume 視窗，如圖 7。
- 在圖 7 中，按下 Browser Local 按鈕，就會出現 Locate ISO media 視窗，如圖 8。
- 在圖 8 中選擇剛剛下載的 Ubuntu ISO 檔案，按下 Open 按鈕，就會回到圖 6。在圖 6 中，按下 Forward 按鈕就會前進到 Step 3，如圖 9。
- C. 在圖 9 中，選擇記憶體大小與 CPU 數量後按下 Forward 按鈕，接著執行 Step 4，如圖 10。
- D. 在圖 10 中，設定虛擬主機的磁碟空間大小，確認後按下 Forward 按鈕，出現 Step 5 的視窗，如圖 11。在圖 11 中，將虛擬主機名稱設定為 master1，按下 Finish 按鈕後，就會開始安裝虛擬主機。
- E. 安裝程序和一般 Ubuntu 的安裝方法完全一樣。當 master1 虛擬主機安裝完畢後，重複上面的程序建立虛擬主機 node1 和 node2。

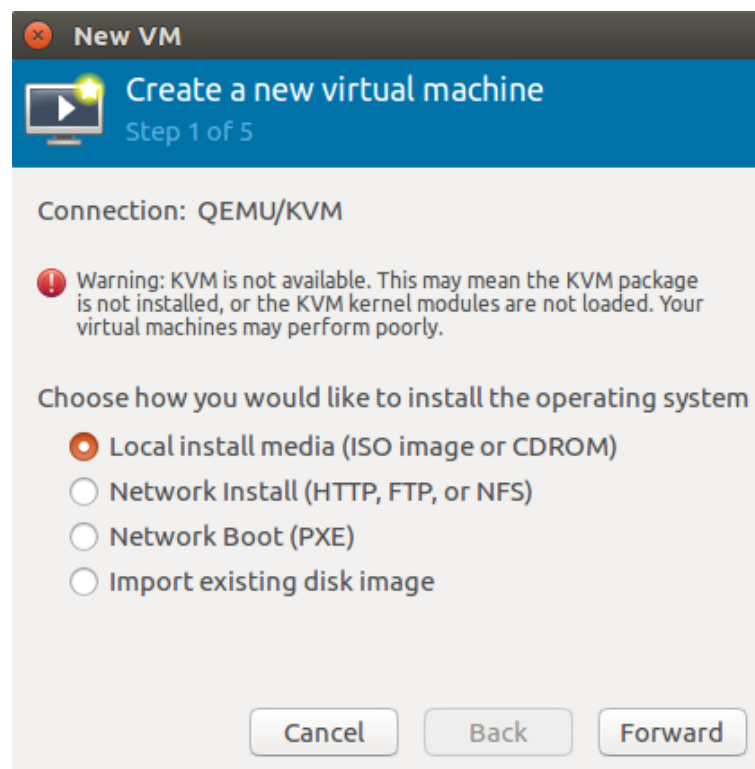


圖 5 Create a new virtual machine: Step 1

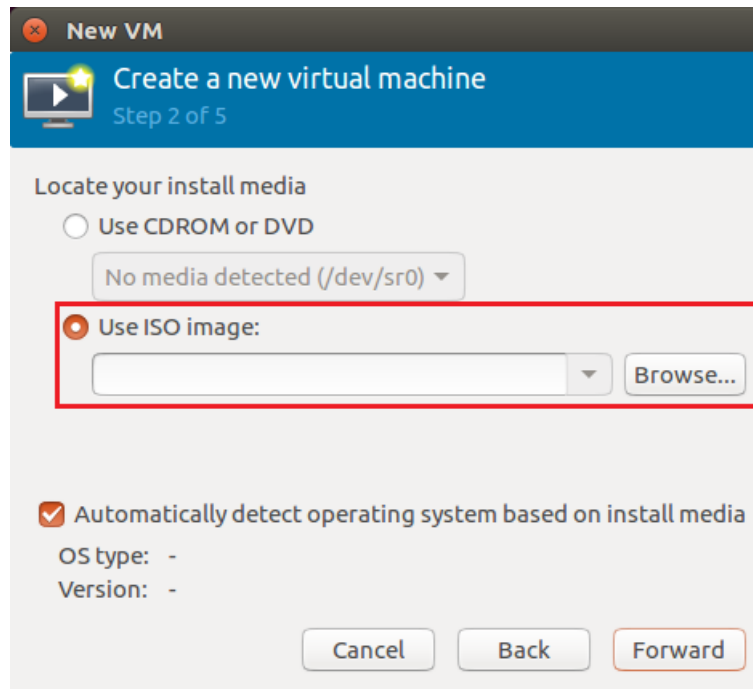


圖 6 Create a new virtual machine: Step 2

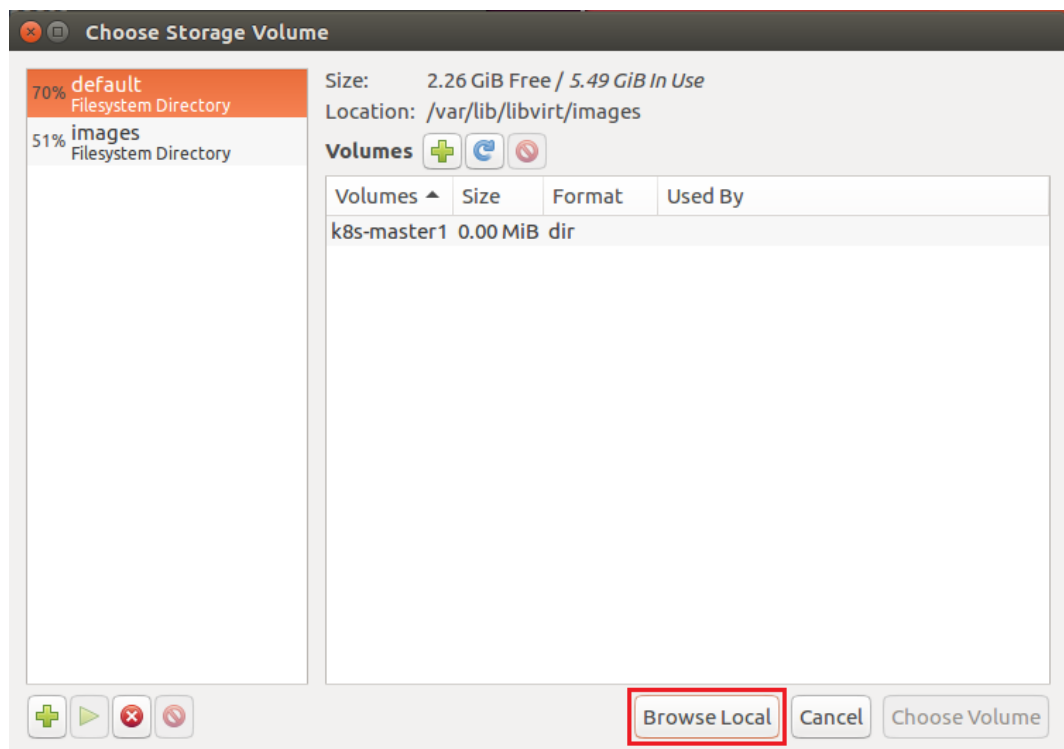


圖 7 Choose Storage Volume 視窗

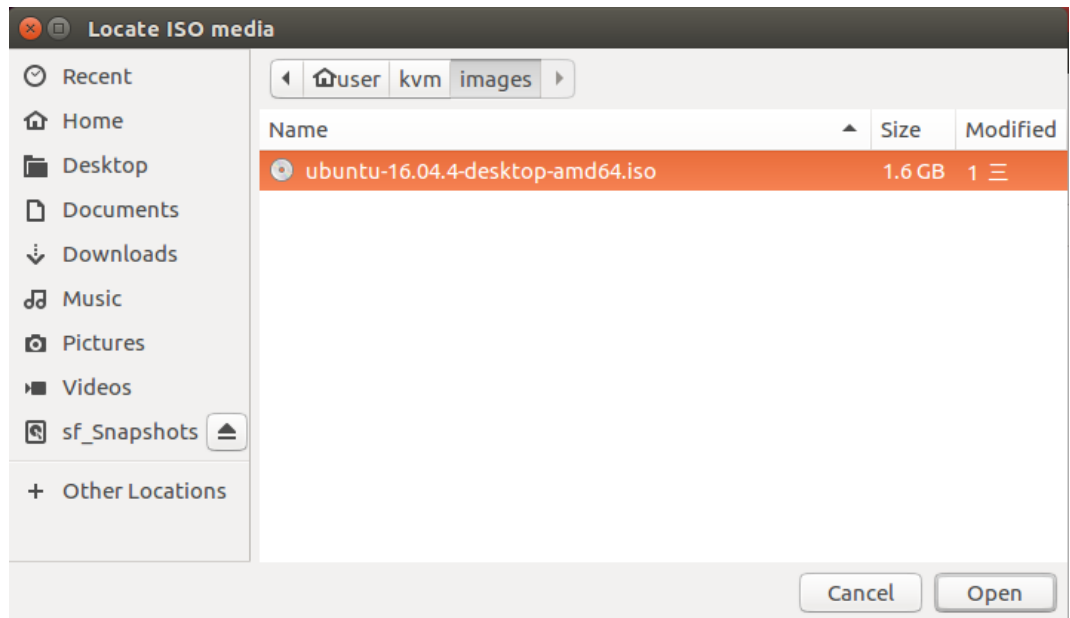


圖 8 Locate ISO media 視窗

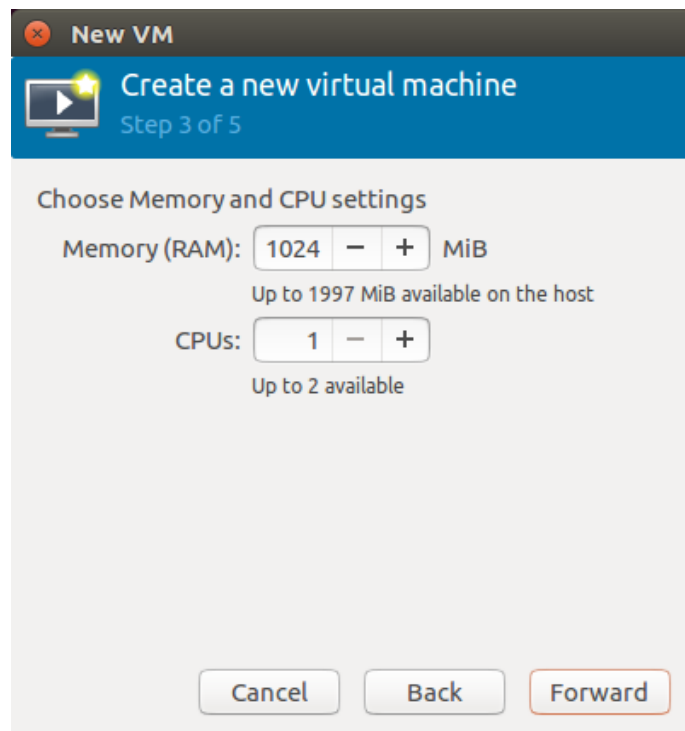


圖 9 Choose Memory and CPU settings 視窗

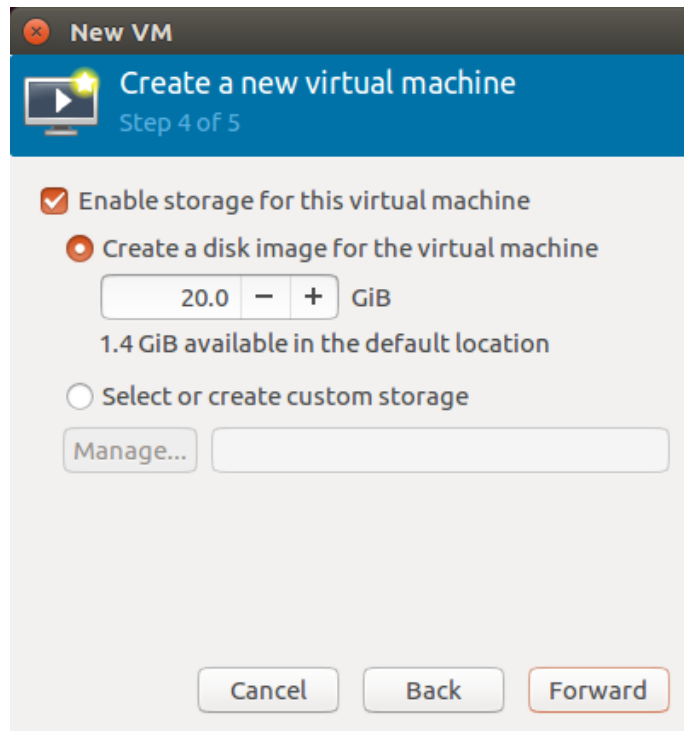


圖 10 Enable storage for this virtual machine

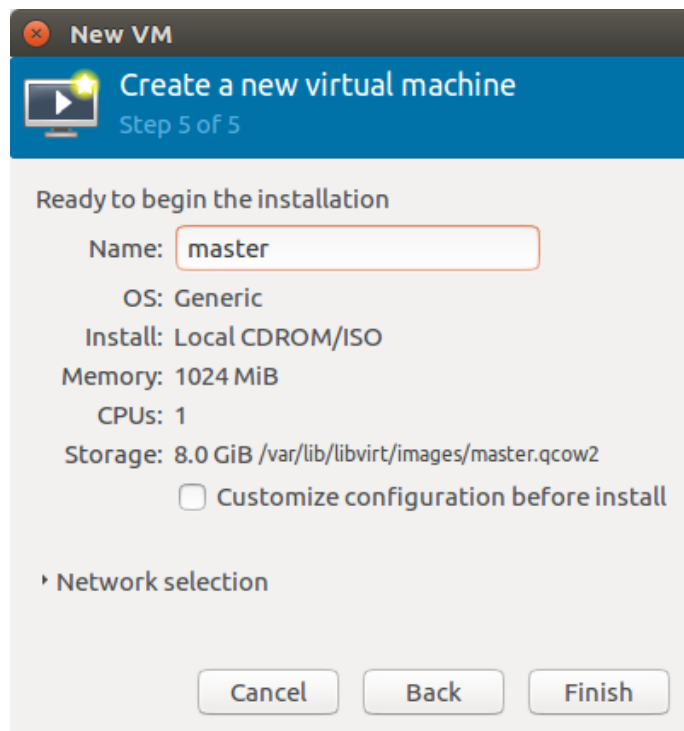


圖 11 Ready to be the installation

2.2.4 設定虛擬主機的網路組態設定：

當所有虛擬主機建置完畢後就設定網路組態，方法如下：

master1 虛擬主機的 /etc/network/interfaces：

```
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

source /etc/network/interfaces.d/*

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto ens3
iface ens3 inet static
    address 192.168.100.91
    netmask 255.255.255.0
    gateway 192.168.100.1
    dns-nameservers 8.8.8.8 8.8.4.4
```

node1 虛擬主機的 /etc/network/interfaces：

```
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

source /etc/network/interfaces.d/*

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto ens3
iface ens3 inet static
    address 192.168.100.92
    netmask 255.255.255.0
    gateway 192.168.100.1
    dns-nameservers 8.8.8.8 8.8.4.4
```

node2 虛擬主機的 /etc/network/interfaces：

```
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

source /etc/network/interfaces.d/*

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto ens3
iface ens3 inet static
```

```
address 192.168.100.93
netmask 255.255.255.0
gateway 192.168.100.1
dns-nameservers 8.8.8.8 8.8.4.4
```

在所有虛擬主機的 /etc/hosts 加入下列文字：

```
192.168.100.91 master1
192.168.100.92 node1
192.168.100.93 node2
```

所有虛擬主機的網路組態都設定完畢後，就全部重新開機。

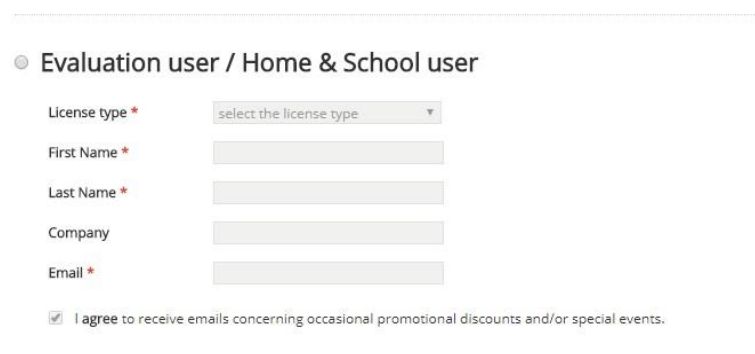
2.3 建立 Win7 和 Ubuntu 間的 SSH 憑證登錄機制：

考慮到管理人員可能需要從 Win7 使用 SSH 登錄到各個 Ubuntu 系統，以方便管理和安裝軟體，所以提供在 Win7 環境下使用 Xshell 軟體和一組金鑰憑證登錄所有 Ubuntu 主機的方法，包含了實體主機 (k8s) 和虛擬主機 (master1, node1 & node2)。

2.3.1 建立 Win7 和實體主機的 SSH 密碼登錄：

在 Win7 安裝 Xshell 軟體。

Xshell 5 的下載網址是 https://www.netsarang.com/download/download_form.html?code=522，如圖 12。填寫基本資料後按下 Submit 按鈕，官網就會把下載連結寄到登記的 email 信箱，點擊 email 中的連結即可下載。



● Evaluation user / Home & School user

License type *

First Name *

Last Name *

Company

Email *

☒ I agree to receive emails concerning occasional promotional discounts and/or special events.

圖 12 Xshell 5 下載頁面

下載完成且安裝完畢後，點擊 Xshell 的圖示，會出現 Xshell 的視窗畫面，如圖 13。在圖

13 中，按下右上角的「建立新工作階段」，就會出現「新增工作階段屬性」視窗。輸入名稱 (N) 和主機 (H) 資訊，其中，名稱 (N) 可以任意填寫，例如 k8s-host。主機 (H) 則填寫實體主機的 IP，也就是 192.168.100.90。

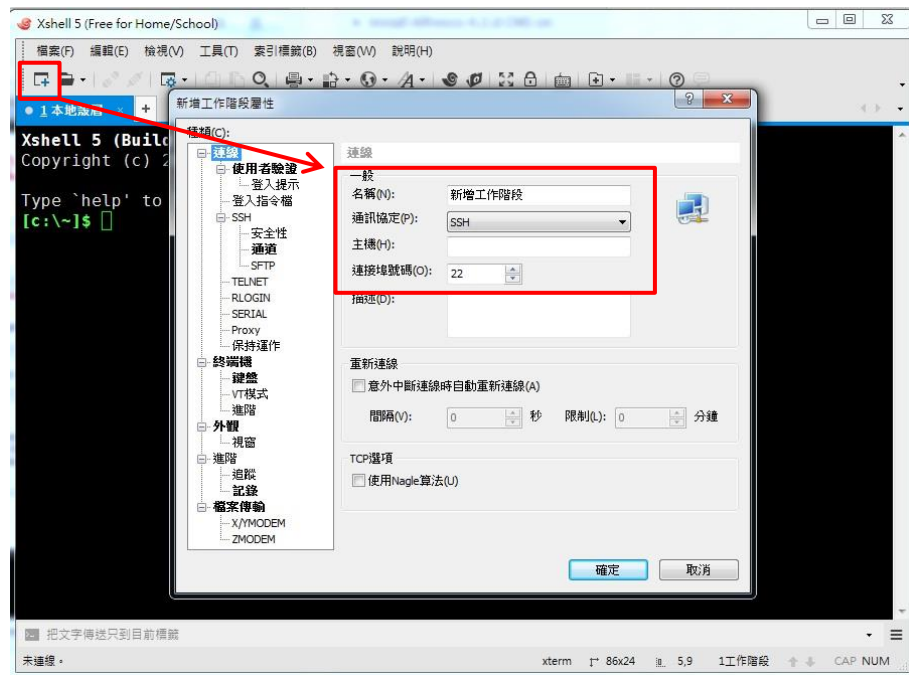


圖 13 Xshell 新增工作階段畫面

接下來，點選左邊樹狀結構中的「使用者驗證」。右邊視窗中的方法 (M) 選擇 'Password' 項目，使用者名稱 (U) 和密碼 (P) 分別填寫實體主機中的帳號名稱和密碼，再按下「確定」按鈕後完成設定。最後，開啟「工作階段」視窗，點選剛剛建立的連線名稱，例如 k8s-host，再按下「確定」按鈕，就可以完成連線。

2.3.2 建立 Win7 和實體主機的 SSH 憑證登錄：

首先，在實體主機 k8s-host 中產生金鑰憑證：

```
$ cd ~/
$ ssh-keygen
$ ls ~/.ssh/
```

會看到 id_rsa 和 id_rsa.pub 檔案。

在實體主機 k8s-host 的 /etc/hosts 檔案，加上下列文字：

```
192.168.100.90 k8s-host
```

接著，將憑證傳送到實體主機 k8s-host。

```
$ ssh-copy-id -f user@k8s-host
```

確認憑證已經建立。

```
$ ls ~/.ssh/authorized_keys  
/home/user/.ssh/authorized_keys
```

在 /etc/ssh/sshd_config 中設定公鑰憑證檔案的路徑和名稱。

編輯 /etc/ssh/sshd_config：

```
$ sudo nano /etc/ssh/sshd_config
```

找到下列文字：

```
# AuthorizedKeysFile      %h/.ssh/authorized_keys
```

移除前面的 '#' 註解符號，存檔後離開。完成後內容如下：

```
RSAAuthentication yes  
PubkeyAuthentication yes  
AuthorizedKeysFile      %h/.ssh/authorized_keys
```

重啟 SSH 服務。

```
$ sudo service ssh restart
```

查看憑證密鑰的內容。

```
$ cat ~/.ssh/id_rsa  
-----BEGIN RSA PRIVATE KEY-----  
MIIEpAIBAAKCAQEA1R0IB1M+MXccF+KMTcQWjoXGVJ6azfnPBmiyKGwgcmhR5dj0  
...  
qLH1/hXV7WiA2PFyPdC8H4HIRjK2i9GlmCB+Us3Ys+3j1aewkZQdYw==  
-----END RSA PRIVATE KEY-----
```

將憑證密鑰內容複製到 Win7 儲存成文字檔，例如 id_rsa_k8s-host.pem。

在 Win7 使用 Xshell 軟體視窗中工具列的 工具(T) → 使用者金鑰管理(U) 功能開啟「使用者金鑰」視窗，如圖 14。在圖 14 中，點選「匯入(I)」按鈕，在「開啟舊檔」視窗中，選擇剛剛建立的 id_rsa_k8s-host.pem 檔案，就會自動匯入。按下「關閉」後結束視窗。

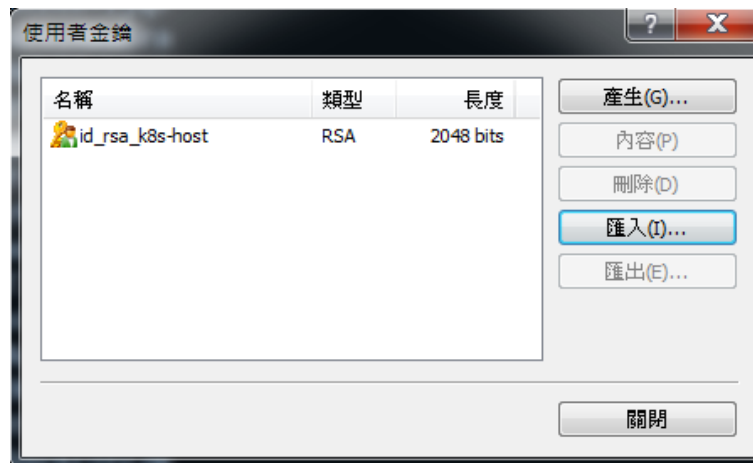


圖 14 Xshell 使用者金鑰視窗

接下來，建立新的工作階段。在名稱 (N) 中輸入 k8s-host，並將主機 (H) 的內容設定為 192.168.100.90。接著，點選「使用者驗證」選項，將方法 (M) 改為 'Public Key'，使用者名稱輸入 user，使用者金鑰 (K) 選擇 id_rsa_k8s-host，如圖 15。按下「確定」按鈕後關閉視窗。

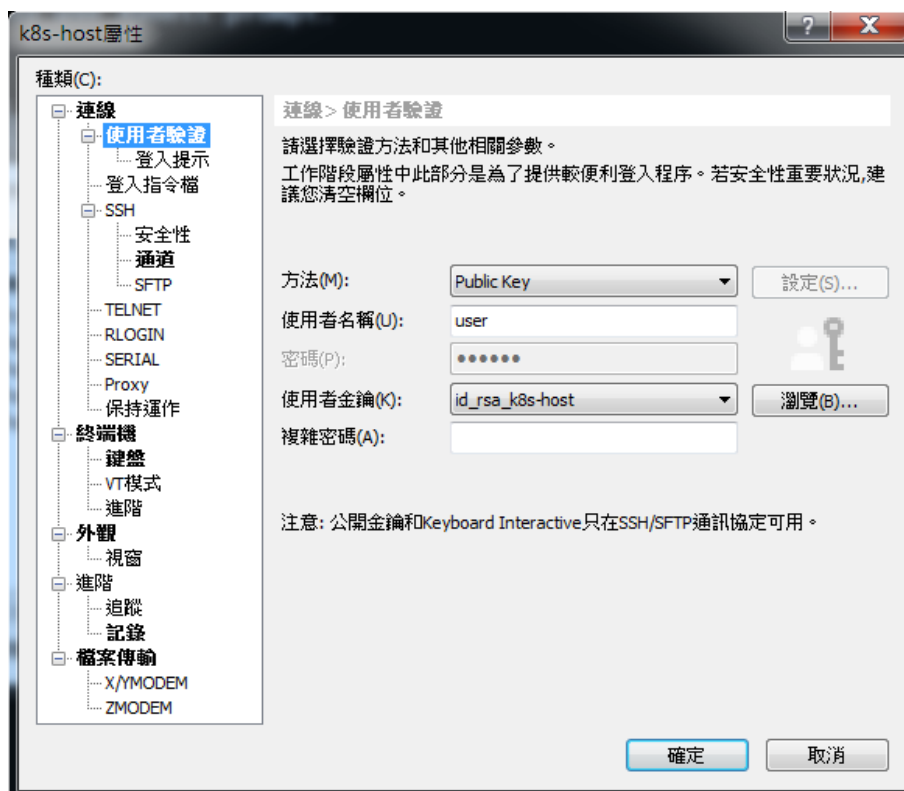


圖 15 使用 Public key 方法連線登錄

接下來，從 Xshell 工具列左上角第二個圖形「開啟(Alt+O)」功能，開啟「工作階段」視

窗。在「工作階段」視窗中點選 k8s-host 名稱，按下「連線」按鈕，就會開始連線。當 SSH 憑證登錄成功連線時，出現畫面如圖 16。

```
Xshell 5 (Build 1339)
Copyright (c) 2002-2017 NetSarang Computer, Inc. All rights reserved.

Type `help' to learn how to use Xshell prompt.
[c:\~]$

Connecting to 192.168.100.90:22...
Connection established.
To escape to local shell, press 'Ctrl+Alt+J'.

Welcome to Ubuntu 16.04.4 LTS (GNU/Linux 4.13.0-38-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

0 個套件可以更新。
0 個屬於安全性更新

Last login: Thu Apr  5 23:37:01 2018 from 192.168.100.3
```

圖 16 SSH 憑證登錄成功畫面

2.3.3 設定所有虛擬主機的 SSH 憑證登錄：

在實體主機 k8s-host 的/etc/hosts 檔案，加上下列文字：

```
192.168.100.91 master1
192.168.100.92 node1
192.168.100.93 node2
```

將公開金鑰憑證複製到各虛擬主機。

```
$ ssh-copy-id -i ~/.ssh/id_rsa.pub user@master1
$ ssh-copy-id -i ~/.ssh/id_rsa.pub user@node1
$ ssh-copy-id -i ~/.ssh/id_rsa.pub user@node2
```

到所有虛擬主機執行下列指令，確認公鑰憑證傳送成功。

```
$ ls ~/.ssh/authorized_keys
```

如果 authorized_keys 檔案存在則表示成功。

依 2.3.2 節的 Xshell 設定 Public key 憑證登錄建立程序來設定個虛擬主機的工作階段。完成後的「工作階段」視窗如圖 17。

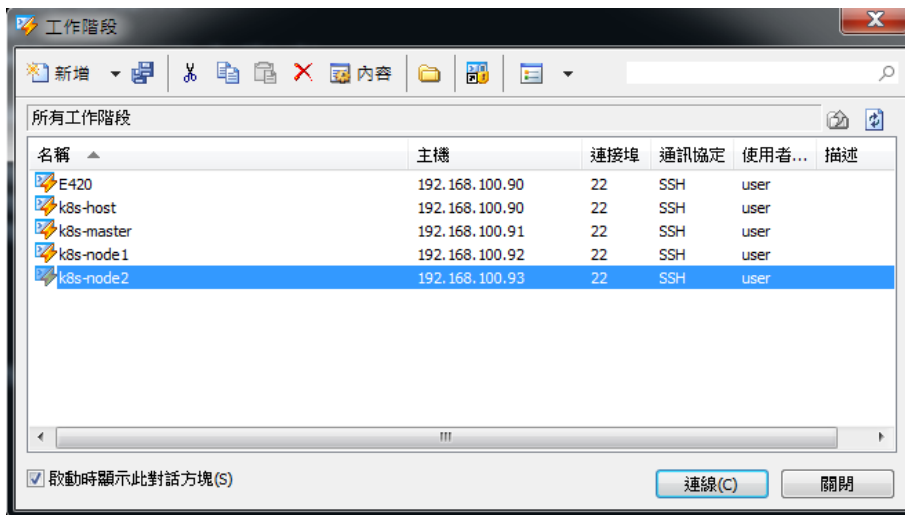


圖 17 虛擬主機 SSH 憑證登錄工作階段建立完成

2.4 建立虛擬機之間 root 的無密碼 SSH 連線環境：

在建置 Kubernetes 時，將會有許多認證金鑰和組態設定檔需要從 master1 傳遞到 nodes，其中一種方法是使用無密碼 SSH 傳遞這些檔案，但由於有些金鑰的權限為 root 擁有，所以需先建立虛擬主機之間 root 帳號的無密碼 SSH 憑證登錄環境，以利後續傳遞 root 權限的金鑰和組態設定檔時使用。值得特別注意的是，所有的虛擬主機，包含 master1 和 nodes，在執行下列的指令操作之前都必須先下達 `sudo -s` (或 `sudo -i`) 指令。其原因有二個，第一是許多 kubernetes 套件和環境的安裝與設定需要 root 權限。第二是 user 帳號的 `/home/user/.ssh/authorized_keys` 檔案已經使用於 Win7 和 Ubuntu 間的 SSH 憑證登錄使用，如果再在虛擬主機之間使用 user 帳號來執行無密碼 SSH 連線，容易使兩組 SSH 憑證金鑰相互覆蓋，增加管理困難，甚至無法連線。故在虛擬主機之間是採取 root 帳號的無密碼 SSH 憑證登錄機制。

2.4.1 在 master1 端建立 SSH 登錄憑證：

因為此組金鑰是供虛擬主機間 SSH 憑證登錄使用，所以將金鑰組的名稱設為 k8s。

為了方便後續操作，在所有虛擬主機環境下先切換成 root 帳號。並以 root 身分來執行下面所有指令：

```
$ sudo -i
```

建立 SSH 連線用的金鑰：

```
$ mkdir ~/.ssh
$ cd ~/.ssh
$ ssh-keygen -f k8s
```

```
Generating public/private rsa key pair.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in k8s.
Your public key has been saved in k8s.pub.
The key fingerprint is:
SHA256:UvzMWGhxQ9PanrSIVGvhtjBKZi81D+cg772smkVUZU0 root@master1
The key's randomart image is:
+---[RSA 2048]---+
|      ..*oooE  |
|      . =0+.  |
|      *o.=     |
|      =+0*B o  |
|      +.BS&+= o |
|      oo+ = +  |
|      o..      |
|      o...     |
|      o...o.   |
+-----[SHA256]-----+
```

```
$ ls ~/.ssh/k8s*
```

執行完畢後在 /root/.ssh/ 目錄下會有 k8s 和 k8s.pub 兩個檔案。

2.4.2 在 node 端允許 root 使用密碼登入：

在 node 端編輯 /etc/ssh/sshd_config。

```
$ nano /etc/ssh/sshd_config
```

找到 #Authentication，將底下的 PermitRootLogin prohibit-password 註解起來。

```
# Authentication:
LoginGraceTime 120
```

```
#PermitRootLogin prohibit-password
StrictModes yes
```

加上下列文字：

```
PasswordAuthentication yes
PermitRootLogin yes
```

儲存並離開 `/etc/ssh/sshd_config`。

重啟 SSH 服務：

```
$ service ssh restart
```

設定 root 帳號的密碼：

```
$ passwd
```

2.4.3 將 master1 憑證傳給 node 端：

在 master1 端執行下列指令：

```
$ ssh-copy-id -i ~/.ssh/k8s root@node1
$ ssh-copy-id -i ~/.ssh/k8s root@node2
```

測試 SSH 連線：

```
$ ssh -i k8s root@node1
$ ssh -i k8s root@node2
```

2.4.4 在 node 端禁用密碼認證：

在 node 端執行下列指令：

```
$ passwd -l root
passwd: password expiry information changed.
```

編輯 `/etc/ssh/sshd_config`，解除密碼認證並允許 root 遠端登錄：

```
$ nano /etc/ssh/sshd_config
```

將下列文字註解或移除：

```
PasswordAuthentication yes  
PermitRootLogin yes
```

增加下列文字：

```
PasswordAuthentication no  
PermitRootLogin without-password
```

重啟 SSH 服務：

```
$ sudo service ssh restart
```

2.4.5 在 master1 端執行憑證登錄測試：

執行下列指令：

```
$ ssh -i ~/.ssh/k8s root@node1  
Welcome to Ubuntu 16.04.4 LTS (GNU/Linux 4.4.0-116-generic x86_64)  
  
* Documentation:  https://help.ubuntu.com  
* Management:    https://landscape.canonical.com  
* Support:        https://ubuntu.com/advantage  
  
7 packages can be updated.  
7 updates are security updates.  
  
Last login: Fri Apr  6 13:09:56 2018 from 192.168.100.91  
測試 node2 的 SSH 憑證登錄。  
$ ssh -i ~/.ssh/k8s root@node2
```

三、Kubernetes 安裝：

Kubernetes 提供了許多雲端平台與作業系統的安裝方式，本章將以全手動安裝方式來部署 Kubernetes v1.8.x 版本，主要是學習與了解 Kubernetes 建置流程。

3.1 預先準備資訊：

本次安裝中，所有的虛擬主機都是使用 Ubuntu 16.04 Server 版本。

3.1.1 虛擬主機的設定如下：

IP Address	Role	CPU	RAM	Disk
192.168.100.91	master1	2	2048 MB	10.0 GB
192.168.100.92	node1	2	2048 MB	10.0 GB
192.168.100.93	node2	2	2048 MB	10.0 GB

其中，master1 為主要控制節點也是部署節點，node 為應用程式工作節點。

以下所有的指令操作，都是以 root 帳號執行的，所以如果是用一般帳號登錄虛擬主機時，請先執行 sudo 指令：

```
$ sudo -i
```

在開始安裝前要確認以下事項都已經準備完成：

3.1.2 關閉防火牆

確認防火牆 (ufw) 已關閉。

```
$ sudo ufw disable
```

3.1.3 在所有虛擬主機的 /etc/hosts 加入網域名稱解析

```
$ nano /etc/hosts
```

加入下列內容：

```
192.168.100.91 master1
192.168.100.92 node1
192.168.100.93 node2
```

3.1.4 安裝 Docker

在所有虛擬主機安裝 Docker，並啟動 Docker 服務。

安裝 Docker 套件

```
$ curl -fsSL "https://get.docker.com/" | sh
```

將 Docker 設定成開機自動啟動，且立即啟動 Docker 服務。

```
$ systemctl enable docker && systemctl start docker
```

檢查 Docker 服務的狀態。

```
$ systemctl status docker
```

```
• docker.service - Docker Application Container Engine
   Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset:
   enabled)
   Active: active (running) since Fri 2018-04-06 13:36:21 CST; 1min 43s ago
     Docs: https://docs.docker.com
  Main PID: 3641 (dockerd)
    CGroup: /system.slice/docker.service
            └─3641 /usr/bin/dockerd -H fd://
               └─3649 docker-containerd --config
                 /var/run/docker/containerd/containerd.toml
```

檢查安裝的 Docker 版本。

```
$ docker -v
```

```
Docker version 18.03.0-ce, build 0520e24
```

編輯/lib/systemd/system/docker.service。

```
$ nano /lib/systemd/system/docker.service
```

找到 ExecStart=...，在上面加上：

```
ExecStartPost=/sbin/iptables -A FORWARD -s 0.0.0.0/0 -j ACCEPT
```

儲存並離開檔案。

完成後內容如下：

```
[Service]
Type=notify
# the default is not to use systemd for cgroups because the delegate issues still
# exists and systemd currently does not support the cgroup feature set required
# for containers run by docker
ExecStartPost=/sbin/iptables -A FORWARD -s 0.0.0.0/0 -j ACCEPT
ExecStart=/usr/bin/dockerd -H fd://
ExecReload=/bin/kill -s HUP $MAINPID
```

重啟 docker 服務並確認服務狀態：

```
$ systemctl daemon-reload && systemctl restart docker
$ systemctl status docker
```

3.1.5 設定 k8s 參數：

在所有虛擬主機設定/etc/sysctl.d/k8s.conf 的系統參數。

```
$ cat <<EOF > /etc/sysctl.d/k8s.conf
net.ipv4.ip_forward = 1
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
EOF
```

啟動 k8s 組態設定。

```
$ sysctl -p /etc/sysctl.d/k8s.conf
net.ipv4.ip_forward = 1
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
```

3.1.6 安裝 CFSSL：

在 master1 安裝 CFSSL 工具，在建立 TLS certificates 時會用到。

```
$ export CFSSL_URL="https://pkg.cfssl.org/R1.2"
$ wget "${CFSSL_URL}/cfssl_linux-amd64" -O /usr/local/bin/cfssl
$ wget "${CFSSL_URL}/cfssljson_linux-amd64" -O /usr/local/bin/cfssljson
$ chmod +x /usr/local/bin/cfssl /usr/local/bin/cfssljson
```


3.2 安裝 ETCD

在開始安裝 Kubernetes (K8S) 之前，需要先將一些必要系統建置完成，其中 ETCD 就是 K8S 最重要的一環，K8S 會將大部分資訊儲存於 ETCD 上，來提供給其他節點索取，以確保整個叢集運作與溝通正常。

3.2.1 建立叢集 CA 與 Certificates

在這一節，將會需要產生 client 與 server 的各元件 certificates，並且替 K8S admin user 產生 client 證書。

首先在 master1 建立 /etc/etcd/ssl 資料夾，然後進入目錄完成以下操作。

```
$ mkdir -p /etc/etcd/ssl && cd /etc/etcd/ssl
$ export PKI_URL="https://kairen.github.io/files/manual-v1.8/pki"
```

下載 ca-config.json 與 etcd-ca-csr.json 檔案，並從 CSR json 產生 CA 金鑰與 Certificate：

```
$ wget "${PKI_URL}/ca-config.json" "${PKI_URL}/etcd-ca-csr.json"
$ cfssl gencert -initca etcd-ca-csr.json | cfssljson -bare etcd-ca
$ ls etcd-ca*.pem
etcd-ca-key.pem etcd-ca.pem
```

下載 etcd-csr.json 檔案。

```
$ wget "${PKI_URL}/etcd-csr.json"
```

依據 master1 的 IP 位址，修改 etcd-csr.json 的 hosts。

```
$ cat etcd-csr.json
{
  "CN": "etcd",
  "hosts": ["127.0.0.1", "192.168.100.91"],
  "key": {
    "algo": "rsa",
    "size": 2048
  },
  "names": [{
    "C": "TW",
    "ST": "Taipei",
```

```
"L":"Taipei",
"O":"etcd",
"OU":"Etcd Security"
}]
}
```

產生 Etcd certificate 證書：

```
$ cfssl gencert \
  -ca=etcd-ca.pem \
  -ca-key=etcd-ca-key.pem \
  -config=ca-config.json \
  -profile=kubernetes \
  etcd-csr.json | cfssljson -bare etcd
```

```
$ ls etcd*.pem
etcd-ca-key.pem etcd-ca.pem etcd-key.pem etcd.pem
```

完成後刪除不必要檔案：

```
$ rm -rf *.json
```

確認/etc/etcd/ssl 有以下檔案：

```
$ ls /etc/etcd/ssl
etcd-ca.csr etcd-ca-key.pem etcd-ca.pem etcd.csr etcd-key.pem etcd.pem
```

3.2.2 Etcd 安裝與設定

首先在 master1 下載 ETCD，並解壓縮放到 /opt 底下與安裝：

```
$ export ETCD_URL="https://github.com/coreos/etcd/releases/download"
$ cd && wget -q0- --show-progress "${ETCD_URL}/v3.2.9/etcd-v3.2.9-linux-amd64.t
  ar.gz" | tar -zx
$ mv etcd-v3.2.9-linux-amd64/etcd* /usr/local/bin/ && rm -rf etcd-v3.2.9-linux-a
  md64
```

完成後新建 Etcd Group 與 User，並建立 Etcd 設定檔目錄：

```
$ groupadd etcd && useradd -c "Etcd user" -g etcd -s /sbin/nologin -r etcd
```

下載 ETCD 相關檔案：

```
$ export ETCD_CONF_URL="https://kaiaren.github.io/files/manual-v1.8/master"
$ wget "${ETCD_CONF_URL}/etcd.conf" -O /etc/etcd/etcd.conf
$ wget "${ETCD_CONF_URL}/etcd.service" -O /lib/systemd/system/etcd.service
```

將 etcd.conf 中的 172.16.35.12 以 master1 的 IP 位址取代。

```
# [cluster]
ETCD_ADVERTISE_CLIENT_URLS=https://192.168.100.91:2379
ETCD_INITIAL_ADVERTISE_PEER_URLS=https://192.168.100.91:2380
ETCD_INITIAL_CLUSTER=master1=https://192.168.100.91:2380
```

建立 var 存放資訊，然後啟動 ETCD 服務：

```
$ mkdir -p /var/lib/etcd && chown etcd:etcd -R /var/lib/etcd /etc/etcd
$ systemctl enable etcd.service && systemctl start etcd.service
```

確認 ETCD 服務已啟動。

```
$ systemctl status etcd
```

驗證 ETCD：

```
$ export CA="/etc/etcd/ssl"
$ ETCDCTL_API=3 etcdctl \
  --cacert=${CA}/etcd-ca.pem \
  --cert=${CA}/etcd.pem \
  --key=${CA}/etcd-key.pem \
  --endpoints="https://192.168.100.91:2379" \
  endpoint health
https://192.168.100.91:2379 is healthy: successfully committed proposal: took
= 4.282902ms
```

3.3 Kubernetes Master：

Master 是 Kubernetes 的大總管，主要建置 apiserver、Controller manager 與 Scheduler 來元件管理所有 Node。本節的目的是下載 Kubernetes 並安裝至 master1 上，然後產生相關 TLS Cert 與 CA 金鑰，提供給叢集元件認證使用。

3.3.1 下載 Kubernetes：

首先，從網路取得所有需要的執行檔案：

下載 Kubernetes：

```
$ export KUBE_URL="https://storage.googleapis.com/kubernetes-release/release/v1.8.6/bin/linux/amd64"
$ wget "${KUBE_URL}/kubelet" -O /usr/local/bin/kubelet
$ wget "${KUBE_URL}/kubectl" -O /usr/local/bin/kubectl
$ chmod +x /usr/local/bin/kubelet /usr/local/bin/kubectl
```

下載 CNI：

```
$ mkdir -p /opt/cni/bin && cd /opt/cni/bin
$ export CNI_URL="https://github.com/containernetworking/plugins/releases/download"
$ wget -qO- --show-progress "${CNI_URL}/v0.6.0/cni-plugins-amd64-v0.6.0.tgz" | tar -zx
```

3.3.2 建立叢集 CA 與 Certificates：

在此節，將會需要產生 client 與 server 的各元件 certificates，並且替 Kubernetes admin user 產生 client 證書。

在 master1 建立 pki 資料夾，然後進入目錄完成以下操作。

```
$ mkdir -p /etc/kubernetes/pki && cd /etc/kubernetes/pki
$ export PKI_URL="https://kaiaren.github.io/files/manual-v1.8/pki"
$ export KUBE_APISERVER="https://192.168.100.91:6443"
```

下載 ca-config.json 與 ca-csr.json 檔案，並產生 CA 金鑰：

```
$ wget "${PKI_URL}/ca-config.json" "${PKI_URL}/ca-csr.json"
$ cfssl gencert -initca ca-csr.json | cfssljson -bare ca
$ ls ca*.pem
ca-key.pem  ca.pem
```

3.3.3 API server certificate

下載 apiserver-csr.json 檔案，並產生 kube-apiserver certificate 證書：

如果 master1 的 IP 位置不同，需要修改第 6 列 hostname 的 IP 位址。

```
$ wget "${PKI_URL}/apiserver-csr.json"
$ cfssl gencert \
  -ca=ca.pem \
  -ca-key=ca-key.pem \
  -config=ca-config.json \
  -hostname=10.96.0.1,192.168.100.91,127.0.0.1,kubernetes.default \
  -profile=kubernetes \
  apiserver-csr.json | cfssljson -bare apiserver
```

```
$ ls apiserver*.pem
apiserver-key.pem  apiserver.pem
```

3.3.4 Front proxy certificate

下載 front-proxy-ca-csr.json 檔案，並產生 Front proxy CA 金鑰。

Front proxy 主要是用在 API aggregator 上：

```
$ wget "${PKI_URL}/front-proxy-ca-csr.json"
$ cfssl gencert -initca front-proxy-ca-csr.json | cfssljson -bare front-proxy-ca
```

```
$ ls front-proxy-ca*.pem
front-proxy-ca-key.pem  front-proxy-ca.pem
```

下載 front-proxy-client-csr.json 檔案，並產生 front-proxy-client 證書：

```
$ wget "${PKI_URL}/front-proxy-client-csr.json"
$ cfssl gencert \
  -ca=front-proxy-ca.pem \
  -ca-key=front-proxy-ca-key.pem \
  -config=ca-config.json \
  -profile=kubernetes \
  front-proxy-client-csr.json | cfssljson -bare front-proxy-client
$ ls front-proxy-client*.pem
front-proxy-client-key.pem  front-proxy-client.pem
```

3.3.5 Bootstrap Token

由於手動建立 CA 的方式在每次簽證時都需要綁定 Node IP，程序太過繁雜，只適合少量機器佈署。因此這裡使用 TLS Bootstrapping 方式進行授權，由 apiserver 自動給符合條件的 Node 發送證書來授權加入叢集。

主要做法是 kubelet 啟動時，向 kube-apiserver 傳送 TLS Bootstrapping 請求，而 kube-apiserver 驗證 kubelet 請求的 token 是否與設定的一樣，若一樣就自動產生 kubelet 證書與金鑰。具體作法可以參考 [TLS bootstrapping](#)。

首先，建立一個變數來產生 BOOTSTRAP_TOKEN，並建立 bootstrap.conf 的 kubeconfig 檔：

```
$ export BOOTSTRAP_TOKEN=$(head -c 16 /dev/urandom | od -An -t x | tr -d ' ')\n$ cat <<EOF > /etc/kubernetes/token.csv\n${BOOTSTRAP_TOKEN},kubelet-bootstrap,10001,"system:kubelet-bootstrap"\nEOF
```

bootstrap set-cluster

```
$ kubectl config set-cluster kubernetes \n  --certificate-authority=ca.pem \n  --embed-certs=true \n  --server=${KUBE_APISERVER} \n  --kubeconfig=../bootstrap.conf
```

bootstrap set-credentials

```
$ kubectl config set-credentials kubelet-bootstrap \n  --token=${BOOTSTRAP_TOKEN} \n  --kubeconfig=../bootstrap.conf
```

bootstrap set-context

```
$ kubectl config set-context default \n  --cluster=kubernetes \n  --user=kubelet-bootstrap \n  --kubeconfig=../bootstrap.conf
```

bootstrap set default context

```
$ kubectl config use-context default --kubeconfig=../bootstrap.conf
```

如果想使用 CA 方式來認證，可以參考 [Kubelet certificate](#)。

3.3.6 Admin certificate

下載 admin-csr.json 檔案，並產生 admin certificate 證書：

```
$ wget "${PKI_URL}/admin-csr.json"
$ cfssl gencert \
  -ca=ca.pem \
  -ca-key=ca-key.pem \
  -config=ca-config.json \
  -profile=kubernetes \
  admin-csr.json | cfssljson -bare admin
```

```
$ ls admin*.pem
admin-key.pem  admin.pem
```

接著使用下面指令產生名為 admin.conf 的 kubeconfig 檔：

```
# admin set-cluster
```

```
$ kubectl config set-cluster kubernetes \
  --certificate-authority=ca.pem \
  --embed-certs=true \
  --server=${KUBE_APISERVER} \
  --kubeconfig=../admin.conf
```

```
# admin set-credentials
```

```
$ kubectl config set-credentials kubernetes-admin \
  --client-certificate=admin.pem \
  --client-key=admin-key.pem \
  --embed-certs=true \
  --kubeconfig=../admin.conf
```

```
# admin set-context
```

```
$ kubectl config set-context kubernetes-admin@kubernetes \
  --cluster=kubernetes \
  --user=kubernetes-admin \
```

```
--kubeconfig=../admin.conf
```

```
# admin set default context
```

```
$ kubectl config use-context kubernetes-admin@kubernetes \
  --kubeconfig=../admin.conf
```

3.3.7 Controller manager certificate

下載 manager-csr.json 檔案，並產生 kube-controller-manager certificate 證書：

```
$ wget "${PKI_URL}/manager-csr.json"
$ cfssl gencert \
  -ca=ca.pem \
  -ca-key=ca-key.pem \
  -config=ca-config.json \
  -profile=kubernetes \
  manager-csr.json | cfssljson -bare controller-manager
```

```
$ ls controller-manager*.pem
```

```
controller-manager-key.pem controller-manager.pem
```

接著使用下面指令產生名為 controller-manager.conf 的 kubeconfig 檔：

```
# controller-manager set-cluster
```

```
$ kubectl config set-cluster kubernetes \
  --certificate-authority=ca.pem \
  --embed-certs=true \
  --server=${KUBE_APISERVER} \
  --kubeconfig=../controller-manager.conf
```

```
# controller-manager set-credentials
```

```
$ kubectl config set-credentials system:kube-controller-manager \
  --client-certificate=controller-manager.pem \
  --client-key=controller-manager-key.pem \
  --embed-certs=true \
  --kubeconfig=../controller-manager.conf
```

```
# controller-manager set-context
```

```
$ kubectl config set-context system:kube-controller-manager@kubernetes \
```



```
--cluster=kubernetes \  
--user=system:kube-controller-manager \  
--kubeconfig=../controller-manager.conf
```

controller-manager set default context

```
$ kubectl config use-context system:kube-controller-manager@kubernetes \  
--kubeconfig=../controller-manager.conf
```

3.3.8 Scheduler certificate

下載 scheduler-csr.json 檔案，並產生 kube-scheduler certificate 證書：

```
$ wget "${PKI_URL}/scheduler-csr.json"  
$ cfssl gencert \  
-ca=ca.pem \  
-ca-key=ca-key.pem \  
-config=ca-config.json \  
-profile=kubernetes \  
scheduler-csr.json | cfssljson -bare scheduler
```

```
$ ls scheduler*.pem
```

```
scheduler-key.pem scheduler.pem
```

接著使用下面指令產生名為 scheduler.conf 的 kubeconfig 檔：

scheduler set-cluster

```
$ kubectl config set-cluster kubernetes \  
--certificate-authority=ca.pem \  
--embed-certs=true \  
--server=${KUBE_APISERVER} \  
--kubeconfig=../scheduler.conf
```

scheduler set-credentials

```
$ kubectl config set-credentials system:kube-scheduler \  
--client-certificate=scheduler.pem \  
--client-key=scheduler-key.pem \  
--embed-certs=true \  
--kubeconfig=../scheduler.conf
```

```
# scheduler set-context
```

```
$ kubectl config set-context system:kube-scheduler@kubernetes \
  --cluster=kubernetes \
  --user=system:kube-scheduler \
  --kubeconfig=../scheduler.conf
```

```
# scheduler set default context
```

```
$ kubectl config use-context system:kube-scheduler@kubernetes \
  --kubeconfig=../scheduler.conf
```

3.3.9 Kubelet master certificate

下載 kubelet-csr.json 檔案，並產生 master node certificate 證書：

```
$ wget "${PKI_URL}/kubelet-csr.json"
```

這裡的 master1 是 kubernetes 的管理主機的網域名稱，要依設定而改變。

```
$ sed -i 's/${NODE}/master1/g' kubelet-csr.json
```

底下 hostname 中的 IP 位址要隨著 master1 的 IP 位址而改變。

```
$ cfssl gencert \
  -ca=ca.pem \
  -ca-key=ca-key.pem \
  -config=ca-config.json \
  -hostname=master1,192.168.100.91 \
  -profile=kubernetes \
  kubelet-csr.json | cfssljson -bare kubelet
```

```
$ ls kubelet*.pem
```

```
kubelet-key.pem kubelet.pem
```

接著使用下面指令產生名為 kubelet.conf 的 kubeconfig 檔：

```
# kubelet set-cluster
```

```
$ kubectl config set-cluster kubernetes \
  --certificate-authority=ca.pem \
  --embed-certs=true \
```

```
--server=${KUBE_APISERVER} \  
--kubeconfig=../kubelet.conf
```

kubelet set-credentials

```
$ kubectl config set-credentials system:node:master1 \  
  --client-certificate=kubelet.pem \  
  --client-key=kubelet-key.pem \  
  --embed-certs=true \  
  --kubeconfig=../kubelet.conf
```

kubelet set-context

```
$ kubectl config set-context system:node:master1@kubernetes \  
  --cluster=kubernetes \  
  --user=system:node:master1 \  
  --kubeconfig=../kubelet.conf
```

kubelet set default context

```
$ kubectl config use-context system:node:master1@kubernetes \  
  --kubeconfig=../kubelet.conf
```

3.3.10 Service account key

Service account 不是透過 CA 進行認證，因此不需透過 CA 來做 Service account key 的檢查，這邊建立一組 Private 與 Public 金鑰提供給 Service account key 使用：

```
$ openssl genrsa -out sa.key 2048  
$ openssl rsa -in sa.key -pubout -out sa.pub  
$ ls sa.*  
sa.key sa.pub
```

完成後刪除不必要檔案：

```
$ rm -rf *.json *.csr
```

確認/etc/kubernetes 與/etc/kubernetes/pki 有以下檔案：

```
$ ls /etc/kubernetes/  
admin.conf bootstrap.conf controller-manager.conf kubelet.conf pki
```

```
scheduler.conf token.csv
```

```
$ ls /etc/kubernetes/pki
```

```
admin-key.pem      ca.pem             front-proxy-client-key.pem  sa.pub
admin.pem          controller-manager-key.pem  front-proxy-client.pem     scheduler-key.pem
apiserver-key.pem  controller-manager.pem     kubelet-key.pem            scheduler.pem
apiserver.pem      front-proxy-ca-key.pem     kubelet.pem
ca-key.pem         front-proxy-ca.pem        sa.key
```

3.4 Kubernetes 核心元件安裝：

首先下載 Kubernetes 核心元件 YAML 檔案，這邊我們不透過 Binary 方案來建立 Master 核心元件，而是利用 Kubernetes Static Pod 來達成，因此需下載所有核心元件的 Static Pod 檔案到 /etc/kubernetes/manifests 目錄：

3.4.1 安裝 apiserver

```
$ export CORE_URL="https://kaiaren.github.io/files/manual-v1.8/master"
$ mkdir -p /etc/kubernetes/manifests && cd /etc/kubernetes/manifests
$ for FILE in apiserver manager scheduler; do
    wget "${CORE_URL}/${FILE}.yaml.conf" -O ${FILE}.yaml
done
```

查看下載的 yaml 檔案：

```
$ ls *.yaml
```

```
apiserver.yaml  manager.yaml  scheduler.yaml
```

將 apiserver.yaml 中的 172.16.35.12 改為 master1 的 IP 位址，總共有兩處。

```
- --advertise-address=192.168.100.91
- --service-cluster-ip-range=10.96.0.0/12
- --service-node-port-range=30000-32767
- --etcd-servers=https://192.168.100.91:2379
```

apiserver 中的 NodeRestriction 請參考 [Using Node Authorization](#)。

3.4.2 安裝 ETCD

產生一個用來加密 ETCD 的 Key：

```
$ head -c 32 /dev/urandom | base64
EM65ajavmNA25B+x+dad9C0nt+JUzm7y+mDzF/8ruk8=
```

在/etc/kubernetes/目錄下，建立 encryption.yml 的加密 YAML 檔案：

```
$ cat <<EOF > /etc/kubernetes/encryption.yml
kind: EncryptionConfig
apiVersion: v1
resources:
  - resources:
      - secrets
    providers:
      - aescbc:
          keys:
            - name: key1
              secret: EM65ajavmNA25B+x+dad9C0nt+JUzm7y+mDzF/8ruk8=
      - identity: {}
EOF
```

Etcd 資料加密可參考這篇 [Encrypting data at rest](#)。

3.4.3 安裝 Auditing

在/etc/kubernetes/目錄下，建立 audit-policy.yml 的進階稽核策略 YAML 檔：

```
$ cat <<EOF > /etc/kubernetes/audit-policy.yml
apiVersion: audit.k8s.io/v1beta1
kind: Policy
rules:
  - level: Metadata
EOF
```

Audit Policy 請參考這篇 [Auditing](#)。

3.4.4 安裝 kubelet

下載 kubelet.service 相關檔案來管理 kubelet：

```
$ export KUBELET_URL="https://kaiaren.github.io/files/manual-v1.8/master"
$ mkdir -p /etc/systemd/system/kubelet.service.d
$ wget "${KUBELET_URL}/kubelet.service" -O /lib/systemd/system/kubelet.service
$ wget "${KUBELET_URL}/10-kubelet.conf" -O /etc/systemd/system/kubelet.service.d/10-kubelet.conf
```

若 cluster-dns 或 cluster-domain 有改變的話，需要修改 10-kubelet.conf。

最後建立 var 存放資訊，然後啟動 kubelet 服務：

```
$ mkdir -p /var/lib/kubelet /var/log/kubernetes
$ systemctl enable kubelet.service && systemctl start kubelet.service
$ systemctl status kubelet.service
```

完成後會需要一段時間來下載映像檔與啟動元件，可以利用該指令來監看：

```
$ watch netstat -ntlp
```

若看到下面資訊表示服務正常啟動，若發生問題可以用 docker cli 來查看。

```
Every 2.0s: netstat -ntlp                               Fri Apr  6 16:00:53 2018

Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      1003/sshd
tcp        0      0 127.0.0.1:10248         0.0.0.0:*               LISTEN      1932/kubelet
tcp        0      0 127.0.0.1:10251         0.0.0.0:*               LISTEN      2237/kube-scheduler
tcp        0      0 127.0.0.1:10252         0.0.0.0:*               LISTEN      2303/kube-controller
tcp6       0      0 :::22                   :::*                     LISTEN      1003/sshd
tcp6       0      0 :::4194                  :::*                     LISTEN      1932/kubelet
tcp6       0      0 :::10250                  :::*                     LISTEN      1932/kubelet
tcp6       0      0 :::6443                   :::*                     LISTEN      2365/kube-apiserver
tcp6       0      0 :::2379                   :::*                     LISTEN      1391/etcd
tcp6       0      0 :::2380                   :::*                     LISTEN      1391/etcd
tcp6       0      0 :::10255                  :::*                     LISTEN      1932/kubelet
```

完成後，複製 admin kubeconfig 檔案，並透過指令來驗證：

```
$ cp /etc/kubernetes/admin.conf ~/.kube/config
$ kubectl get cs
```

NAME	STATUS	MESSAGE	ERROR
scheduler	Healthy	ok	
controller-manager	Healthy	ok	
etcd-0	Healthy	{"health": "true"}	

```
$ kubectl get node
```

NAME	STATUS	ROLES	AGE	VERSION
master1	NotReady	master	4m	v1.8.6

```
$ kubectl -n kube-system get po
```

NAME	READY	STATUS	RESTARTS	AGE
kube-apiserver-master1	1/1	Running	0	3m
kube-controller-manager-master1	1/1	Running	0	3m
kube-scheduler-master1	1/1	Running	0	3m

```
root@master1:/etc/kubernetes/manifests#
```

確認服務能夠執行 logs 等指令：

```
$ kubectl -n kube-system logs -f kube-scheduler-master1
```

```
Error from server (Forbidden): Forbidden (user=kube-apiserver, verb=get, resource=nodes, subresource=proxy) ( pods/log kube-scheduler-master1)
```

這邊出現 Forbidden 問題是屬於正常結果，因為 kube-apiserver user 並沒有 nodes 的資源權限。

由於上述權限問題，必需建立一個 apiserver-to-kubelet-rbac.yml 來定義權限，以執行 logs、exec 等指令：

```
$ cd /etc/kubernetes/
$ export URL="https://kaiaren.github.io/files/manual-v1.8/master"
$ wget "${URL}/apiserver-to-kubelet-rbac.yml.conf" -O apiserver-to-kubelet-rbac.yml
```

```
$ kubectl apply -f apiserver-to-kubelet-rbac.yml
```

```
clusterrole "system:kube-apiserver-to-kubelet" created
clusterrolebinding "system:kube-apiserver" created
```

測試 logs

```
$ kubectl -n kube-system logs -f kube-scheduler-master1
```

```
...
I0406 08:00:37.301498      1 leaderelection.go:184] successfully acquired lease
kube-system/kube-scheduler
...
```

3.5 Kubernetes Node 安裝

Node 是主要執行容器實例的節點，可視為工作節點。在此節會下載 Kubernetes binary 檔，並建立 node 的 certificate 來提供給節點註冊認證用。Kubernetes 使用 Node Authorizer 來提供 Authorization mode，這種授權模式會替 Kubelet 生成 API request。

3.5.1 複製 CA 與 Cert：

從 master1 將需要的 ca 與 cert 複製到 Node 節點上：

```
$ for NODE in node1 node2; do
> # Create directory
> ssh -i ${K8S_SSH_KEY} ${NODE} "mkdir -p /etc/kubernetes/pki/"
> ssh -i ${K8S_SSH_KEY} ${NODE} "mkdir -p /etc/etcd/ssl/"
> # Etcd ca and cert
> for FILE in etcd-ca.pem etcd.pem etcd-key.pem; do
> scp -i ${K8S_SSH_KEY} /etc/etcd/ssl/${FILE} ${NODE}:/etc/etcd/ssl/${FILE}
> done
> # Kubernetes ca and cert
> for FILE in pki/ca.pem pki/ca-key.pem bootstrap.conf; do
> scp -i ${K8S_SSH_KEY} /etc/kubernetes/${FILE} ${NODE}:/etc/kubernetes/${FILE}
> done
> done
```

查詢是否有成功複製到 Node 端：

```
$ ssh -i ${K8S_SSH_KEY} ${NODE} "ls /etc/etcd/ssl/"
```

```
etcd-ca.pem
etcd-key.pem
etcd.pem
```

```
$ ssh -i ${K8S_SSH_KEY} ${NODE} "ls /etc/kubernetes/"
```

```
bootstrap.conf
pki
```

```
$ ssh -i ${K8S_SSH_KEY} ${NODE} "ls /etc/kubernetes/pki/"
```

```
ca-key.pem
ca.pem
```


3.5.2 在 Node 們下載 Kubernetes 元件：

在 node 端執行下列指令，從網路下載所有需要的執行檔案：

Download Kubernetes

```
$ export KUBE_URL="https://storage.googleapis.com/kubernetes-release/release/v1.8.6/bin/linux/amd64"
$ wget "${KUBE_URL}/kubelet" -O /usr/local/bin/kubelet
$ chmod +x /usr/local/bin/kubelet
```

Download CNI

```
$ mkdir -p /opt/cni/bin && cd /opt/cni/bin
$ export CNI_URL="https://github.com/containernetworking/plugins/releases/download"
$ wget -qO- --show-progress "${CNI_URL}/v0.6.0/cni-plugins-amd64-v0.6.0.tgz" | tar -zx
```

3.5.3 設定 Kubernetes node

接著下載 Kubernetes 相關檔案，包含 drop-in file、systemd service 檔案等：

```
$ export KUBELET_URL="https://kairn.github.io/files/manual-v1.8/node"
$ mkdir -p /etc/systemd/system/kubelet.service.d
$ wget "${KUBELET_URL}/kubelet.service" -O /lib/systemd/system/kubelet.service
$ wget "${KUBELET_URL}/10-kubelet.conf" -O /etc/systemd/system/kubelet.service.d/10-kubelet.conf
```

若 cluster-dns 或 cluster-domain 有改變的話，需要修改 10-kubelet.conf。

接著在所有 node 建立 var 存放資訊，然後啟動 kubelet 服務：

```
$ mkdir -p /var/lib/kubelet /var/log/kubernetes /etc/kubernetes/manifests
$ systemctl enable kubelet.service && systemctl start kubelet.service
```

3.5.4 授權 Kubernetes Node

當所有節點都完成後，在 master1 節點建立一個 ClusterRoleBinding：

```
$ kubectl create clusterrolebinding kubelet-bootstrap \
  --clusterrole=system:node-bootstrapper \
  --user=kubelet-bootstrap
clusterrolebinding "kubelet-bootstrap" created
```

在 master 透過簡單指令驗證，會看到節點處於 pending：

```
$ kubectl get csr
```

NAME	AGE	REQUESTOR	CONDITION
node-csr-auKTe9C3_rA2HRed803lqS1H2qHdLfivAZpm0Pohzcx	40s	kubelet-bootstrap	Pending
node-csr-cv7H4tk3f0NoM7iU9QozSYUVEJtBwGnaQ0g9S-mXr0E	37s	kubelet-bootstrap	Pending

透過 kubectl 來允許節點加入叢集：

```
$ kubectl get csr | awk '/Pending/ {print $1}' | xargs kubectl certificate approve
```

```
certificatesigningrequest "node-csr-auKTe9C3_rA2HRed803lqS1H2qHdLfivAZpm0Pohzcx" approved
certificatesigningrequest "node-csr-cv7H4tk3f0NoM7iU9QozSYUVEJtBwGnaQ0g9S-mXr0E" approved
```

```
$ kubectl get csr
```

NAME	AGE	REQUESTOR	CONDITION
node-csr-auKTe9C3_rA2HRed803lqS1H2qHdLfivAZpm0Pohzcx	3m	kubelet-bootstrap	Approved, Issued
node-csr-cv7H4tk3f0NoM7iU9QozSYUVEJtBwGnaQ0g9S-mXr0E	3m	kubelet-bootstrap	Approved, Issued

```
$ kubectl get no
```

NAME	STATUS	ROLES	AGE	VERSION
master1	NotReady	master	42m	v1.8.6
node1	NotReady	node	1m	v1.8.6
node2	NotReady	node	1m	v1.8.6

3.6 Kubernetes Core Addons 部署

當完成上面所有步驟後，可以安裝一些非常重要跟好用的套件，例如 Kube-dns 與 Kube-proxy。

3.6.1 Kube-proxy addon

[Kube-proxy](#) 是實現 Service 的關鍵元件，kube-proxy 會在每台節點上執行，然後監聽 API Server 的 Service 與 Endpoint 資源物件的改變，然後來依據變化執行 iptables 來實現網路的路由。這邊會需要建議一個 DaemonSet 來執行，並且建立一些需要的 certificate。

首先在 master1 下載 kube-proxy-csr.json 檔案，並產生 kube-proxy certificate 證書：

```
$ export PKI_URL="https://kairen.github.io/files/manual-v1.8/pki"
$ cd /etc/kubernetes/pki
$ wget "${PKI_URL}/kube-proxy-csr.json" "${PKI_URL}/ca-config.json"
$ cfssl gencert \
  -ca=ca.pem \
  -ca-key=ca-key.pem \
  -config=ca-config.json \
  -profile=kubernetes \
  kube-proxy-csr.json | cfssljson -bare kube-proxy
```

```
$ ls kube-proxy*.pem
kube-proxy-key.pem  kube-proxy.pem
```

接著使用下面指令產生名為 kube-proxy.conf 的 kubeconfig 檔：

kube-proxy set-cluster

```
$ kubectl config set-cluster kubernetes \
  --certificate-authority=ca.pem \
  --embed-certs=true \
  --server="https://192.168.100.91:6443" \
  --kubeconfig=../kube-proxy.conf
```

kube-proxy set-credentials

```
$ kubectl config set-credentials system:kube-proxy \
  --client-key=kube-proxy-key.pem \
  --client-certificate=kube-proxy.pem \
  --embed-certs=true \
  --kubeconfig=../kube-proxy.conf
```

kube-proxy set-context

```
$ kubectl config set-context system:kube-proxy@kubernetes \
  --cluster=kubernetes \
  --user=system:kube-proxy \
  --kubeconfig=../kube-proxy.conf
```

kube-proxy set default context

```
$ kubectl config use-context system:kube-proxy@kubernetes \
  --kubeconfig=../kube-proxy.conf
```

完成後刪除不必要檔案：

```
$ rm -rf *.json *.csr
```

確認/etc/kubernetes 有以下檔案：

```
$ ls /etc/kubernetes/
```

admin.conf	bootstrap.conf	kubelet.conf	pki
apiserver-to-kubelet-rbac.yml	controller-manager.conf	kube-proxy.conf	scheduler.conf
audit-policy.yml	encryption.yml	manifests	token.csv

在 master1 將 kube-proxy 相關檔案複製到 Node 節點上：

```
$ for NODE in node1 node2; do
> echo "--- $NODE ---"
> for FILE in pki/kube-proxy.pem pki/kube-proxy-key.pem kube-proxy.conf; do
> scp -i ${K8S_SSH_KEY} /etc/kubernetes/${FILE} ${NODE}:/etc/kubernetes/${FILE}
> done
> done
```

```
--- node1 ---
kube-proxy.pem                100% 1456    1.4KB/s   00:00
kube-proxy-key.pem            100% 1679    1.6KB/s   00:00
kube-proxy.conf               100% 6484    6.3KB/s   00:00
--- node2 ---
kube-proxy.pem                100% 1456    1.4KB/s   00:00
kube-proxy-key.pem            100% 1679    1.6KB/s   00:00
kube-proxy.conf               100% 6484    6.3KB/s   00:00
```

完成後，在 master1 透過 kubectl 來建立 kube-proxy daemon：

```
$ export ADDON_URL="https://kairan.github.io/files/manual-v1.8/addon"
$ mkdir -p /etc/kubernetes/addons && cd /etc/kubernetes/addons
$ wget "${ADDON_URL}/kube-proxy.yml.conf" -O kube-proxy.yml
```

```
$ kubectl apply -f kube-proxy.yml
serviceaccount "kube-proxy" created
daemonset "kube-proxy" created
```

```
$ kubectl -n kube-system get po -l k8s-app=kube-proxy
```

NAME	READY	STATUS	RESTARTS	AGE
kube-proxy-9smgm	1/1	Running	0	32s
kube-proxy-llq5w	1/1	Running	0	32s
kube-proxy-rmqwl	1/1	Running	0	32s

3.6.2 Kube-dns addon

[Kube DNS](#) 是 Kubernetes 叢集內部 Pod 之間互相溝通的重要 Addon，它允許 Pod 可以透過 Domain Name 方式來連接 Service，其主要由 Kube DNS 與 Sky DNS 組合而成，透過 Kube DNS 監聽 Service 與 Endpoint 變化，來提供給 Sky DNS 資訊，已更新解析位址。

只需要在 master1 透過 kubectl 來建立 kube-dns deployment 即可：

```
$ export ADDON_URL="https://kaiaren.github.io/files/manual-v1.8/addon"
$ wget "${ADDON_URL}/kube-dns.yml.conf" -O kube-dns.yml
$ kubectl apply -f kube-dns.yml
```

```
serviceaccount "kube-dns" created
service "kube-dns" created
deployment "kube-dns" created
```

```
$ kubectl -n kube-system get po -l k8s-app=kube-dns
```

NAME	READY	STATUS	RESTARTS	AGE
kube-dns-6cb549f55f-59bjf	0/3	Pending	0	11s

3.7 Calico Network 安裝與設定

Calico 是一款純 Layer 3 的資料中心網路方案（不需要 Overlay 網路），好處是已與各種雲原生平台有良好的整合，而 Calico 在每一個節點利用 Linux Kernel 實現高效的 vRouter 來負責資料的轉發，而當資料中心複雜度增加時，可以用 BGP route reflector 來達成。

3.7.1 建立 Calico policy controller

首先在 master1 透過 kubectl 建立 Calico policy controller：

```
$ export CALICO_CONF_URL="https://kaiaren.github.io/files/manual-v1.8/network"
$ wget "${CALICO_CONF_URL}/calico-controller.yml.conf" -O calico-controller.yml
```

依 master1 的 IP 位址修改 calico-controller.yml 中 ETCD_ENDPOINTS 的 value。

```
$ nano calico-controller.yml
```

```
env:
  - name: ETCD_ENDPOINTS
    value: "https://192.168.100.91:2379"
  - name: ETCD_CA_CERT_FILE
    value: "/etc/etcd/ssl/etcd-ca.pem"
```

```
$ kubectl apply -f calico-controller.yml
```

```
clusterrolebinding "calico-kube-controllers" created
clusterrole "calico-kube-controllers" created
serviceaccount "calico-kube-controllers" created
deployment "calico-policy-controller" created
```

```
$ kubectl -n kube-system get po -l k8s-app=calico-policy
```

NAME	READY	STATUS	RESTARTS	AGE
calico-policy-controller-76bf9574d4-qbnqd	0/1	Pending	0	3m

3.7.2 安裝 Calico：

在 master1 下載 Calico CLI 工具：

```
$ wget https://github.com/projectcalico/calicoctl/releases/download/v1.6.1/calicoctl
$ chmod +x calicoctl && mv calicoctl /usr/local/bin/
```

在所有節點下載 Calico，並執行以下步驟：

```
$ export CALICO_URL="https://github.com/projectcalico/cni-plugin/releases/download/v1.11.0"
$ wget -N -P /opt/cni/bin ${CALICO_URL}/calico
$ wget -N -P /opt/cni/bin ${CALICO_URL}/calico-ipam
$ chmod +x /opt/cni/bin/calico /opt/cni/bin/calico-ipam
```

在所有節點下載 CNI plugins 設定檔，以及 calico-node.service：

```
$ mkdir -p /etc/cni/net.d
$ export CALICO_CONF_URL="https://kaiaren.github.io/files/manual-v1.8/network"
$ wget "${CALICO_CONF_URL}/10-calico.conf" -O /etc/cni/net.d/10-calico.conf
```

將所有節點的 /etc/cni/net.d/10-calico.conf 中的 etcd_endpoints 的值改為 master1 的 IP 位址。修改後內容如下：

```
"name": "calico-k8s-network",
"cniVersion": "0.1.0",
"type": "calico",
"etcd_endpoints": "https://192.168.100.91:2379",
"etcd_ca_cert_file": "/etc/etcd/ssl/etcd-ca.pem",
```

```
$ wget "${CALICO_CONF_URL}/calico-node.service" -O /lib/systemd/system/calico-node.service
```

將所有節點的 /lib/systemd/system/calico-node.service 中的 ETCD_ENDPOINTS 的 172.16.35.12 改為 master1 的 IP 位址。以及將 IP_AUTODETECTION_METHOD 和 IP6_AUTODETECTION_METHOD 的值改為虛擬主機的網路名稱，在此範例中為 ens3。

```
[Service]
User=root
PermissionsStartOnly=true
ExecStart=/usr/bin/docker run --net=host --privileged --name=calico-node \
-e ETCD_ENDPOINTS=https://192.168.100.91:2379 \
-e ETCD_CA_CERT_FILE=/etc/etcd/ssl/etcd-ca.pem \
-e ETCD_CERT_FILE=/etc/etcd/ssl/etcd.pem \
-e ETCD_KEY_FILE=/etc/etcd/ssl/etcd-key.pem \
-e NODENAME=${HOSTNAME} \
-e IP= \
-e NO_DEFAULT_POOLS= \
-e AS= \
-e CALICO_LIBNETWORK_ENABLED=true \
-e IP6= \
-e CALICO_NETWORKING_BACKEND=bird \
-e FELIX_DEFAULTENDPOINTTOHOSTACTION=ACCEPT \
-e FELIX_HEALTHENABLED=true \
-e CALICO_IPV4POOL_CIDR=10.244.0.0/16 \
-e CALICO_IPV4POOL_IPIP=always \
-e IP_AUTODETECTION_METHOD=interface=ens3 \
-e IP6_AUTODETECTION_METHOD=interface=ens3 \
-v /etc/etcd/ssl:/etc/etcd/ssl \
```

3.7.3 啟動 Calico

然後在所有節點啟動 Calico-node：

```
$ systemctl enable calico-node.service && systemctl start calico-node.service
```

在 master1 啟動 Calico-node 時會出現下面訊息，在 node 則不會出現任何訊息：

```
Created symlink from /etc/systemd/system/multi-user.target.wants/calico-node.service to /lib/systemd/system/calico-node.service.
```

在 master1 查看 Calico nodes：

```
$ cat <<EOF > ~/calico-rc
export ETCD_ENDPOINTS="https://192.168.100.91:2379"
export ETCD_CA_CERT_FILE="/etc/etcd/ssl/etcd-ca.pem"
export ETCD_CERT_FILE="/etc/etcd/ssl/etcd.pem"
export ETCD_KEY_FILE="/etc/etcd/ssl/etcd-key.pem"
EOF
```

```
$ . ~/calico-rc
$ calicoctl get node -o wide
```

NAME	ASN	IPV4	IPV6
master1	(64512)	192.168.100.91/24	
node1	(64512)	192.168.100.92/24	
node2	(64512)	192.168.100.93/24	

查看 pending 的 pod 是否已執行：

```
$ kubectl -n kube-system get po
```

NAME	READY	STATUS	RESTARTS	AGE
calico-policy-controller-76bf9574d4-qbnqd	1/1	Running	0	46m
kube-apiserver-master1	1/1	Running	0	2h
kube-controller-manager-master1	1/1	Running	0	2h
kube-dns-6cb549f55f-59bjf	3/3	Running	0	59m
kube-proxy-9smgm	1/1	Running	0	1h
kube-proxy-1lq5w	1/1	Running	0	1h
kube-proxy-rmqwl	1/1	Running	0	1h
kube-scheduler-master1	1/1	Running	0	2h

最後，如果想省事，可以直接用 [Standard Hosted](#) 方式安裝。

3.8 Kubernetes Extra Addons 部署

本節說明如何部署一些官方常用的 Addons，如 Dashboard、Heapster 等。

3.8.1 Dashboard addon

[Dashboard](#) 是 Kubernetes 社區官方開發的儀表板，有了儀表板後管理者就能夠透過 Web-based 方式來管理 Kubernetes 叢集，除了提升管理方便，也讓資源視覺化，讓人更直覺看見系統資訊的呈現結果。

在 master1 透過 kubectl 來建立 kubernetes dashboard 即可：

```
$ kubectl apply -f https://raw.githubusercontent.com/kubernetes/dashboard/master/src/deploy/recommended/kubernetes-dashboard.yaml
```

```
secret "kubernetes-dashboard-certs" created
serviceaccount "kubernetes-dashboard" created
role "kubernetes-dashboard-minimal" created
rolebinding "kubernetes-dashboard-minimal" created
deployment "kubernetes-dashboard" created
service "kubernetes-dashboard" created
```

```
$ kubectl -n kube-system get po,svc -l k8s-app=kubernetes-dashboard
```

NAME	READY	STATUS	RESTARTS	AGE
po/kubernetes-dashboard-5569448c6d-2w6mq	1/1	Running	0	1m

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
svc/kubernetes-dashboard	ClusterIP	10.98.144.35	<none>	443/TCP	1m

這邊會額外建立一個名稱為 open-api Cluster Role Binding，這僅作為方便測試時使用，在一般情況下不要開啟，不然就會直接被存取所有 API：

```
$ cat <<EOF | kubectl create -f -
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: open-api
  namespace: ""
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
```

```
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: system:anonymous
EOF
```

P.S：管理者可以針對特定使用者來開放 API 存取權限，但這邊方便使用直接綁在 cluster-admin cluster role。

完成後，就可以透過瀏覽器存取 [Dashboard](#)。

在 1.7 版本以後的 Dashboard 將不再提供所有權限，因此需要建立一個 service account 來綁定 cluster-admin role：

```
$ kubectl -n kube-system create sa dashboard
serviceaccount "dashboard" created
```

```
$ kubectl create clusterrolebinding dashboard --clusterrole cluster-admin --serviceaccount=kube-system:dashboard
clusterrolebinding "dashboard" created
```

```
$ SECRET=$(kubectl -n kube-system get sa dashboard -o yaml | awk '/dashboard-token/ {print $3}')
```

```
$ kubectl -n kube-system describe secrets ${SECRET} | awk '/token:/{print $2}'
```

```
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJrdWJlcm5ldGVzL3NlcnZpY2VhY2NvdW50Iiwia3ViZXJuZXRlcy5pby9zZXJ2aWNLWYWNjb3VudC9uYW1lc3BhY2UiOiJrdWJlLXN5c3RlbSI...
```

複製此 token，然後貼到 [Kubernetes dashboard](#)。

3.8.2 Heapster addon

[Heapster](#) 是 Kubernetes 社區維護的容器叢集監控與效能分析工具。Heapster 會從 Kubernetes apiserver 取得所有 Node 資訊，然後再透過這些 Node 來取得 kubelet 上的資料，最後再將所有收集到資料送到 Heapster 的後台儲存 InfluxDB，最後利用 Grafana 來抓取 InfluxDB 的資料源來進行視覺化。

在 master1 透過 kubectl 來建立 kubernetes monitor 即可：

```
$ export ADDON_URL="https://kaiaren.github.io/files/manual-v1.8/addon"
$ wget ${ADDON_URL}/kube-monitor.yml.conf -O kube-monitor.yml
$ kubectl apply -f kube-monitor.yml
```

```
serviceaccount "heapster-sa" created
clusterrolebinding "heapster-binding" created
role "system:pod-nanny" created
rolebinding "heapster-binding" created
service "heapster" created
service "monitoring-grafana" created
service "monitoring-influxdb" created
deployment "heapster" created
deployment "influxdb-grafana" created
```

```
$ kubectl -n kube-system get po,svc
```

NAME	READY	STATUS	RESTARTS	AGE
po/calico-policy-controller-76bf9574d4-qbnqd	1/1	Running	0	2h
po/heapster-74fb5c8cdc-zxrg8	4/4	Running	0	1m
po/influxdb-grafana-55bd7df44-hl7lv	2/2	Running	0	1m
po/kube-apiserver-master1	1/1	Running	0	3h
po/kube-controller-manager-master1	1/1	Running	0	3h
po/kube-dns-6cb549f55f-59bjf	3/3	Running	0	2h
po/kube-proxy-9smgm	1/1	Running	0	2h
po/kube-proxy-llq5w	1/1	Running	0	2h
po/kube-proxy-rmqwl	1/1	Running	0	2h
po/kube-scheduler-master1	1/1	Running	0	3h
po/kubernetes-dashboard-5569448c6d-2w6mq	1/1	Running	0	13m

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
svc/heapster	ClusterIP	10.100.144.51	<none>	80/TCP	1m
svc/kube-dns	ClusterIP	10.96.0.10	<none>	53/UDP,53/TCP	2h
svc/kubernetes-dashboard	ClusterIP	10.98.144.35	<none>	443/TCP	13m
svc/monitoring-grafana	ClusterIP	10.96.100.17	<none>	80/TCP	1m
svc/monitoring-influxdb	ClusterIP	10.100.136.239	<none>	8083/TCP,8086/TCP	1m

完成後，就可以透過瀏覽器存取 [Grafana Dashboard](#)。

3.8.3 簡單部署 Nginx 服務

Kubernetes 可以選擇使用指令直接建立應用程式與服務，或者撰寫 YAML 與 JSON 檔案來描述部署應用程式的配置，以下將建立一個簡單的 Nginx 服務：

```
$ kubectl run nginx --image=nginx --port=80
```

```
deployment "nginx" created
```

```
$ kubectl expose deploy nginx --port=80 --type=LoadBalancer --external-ip=192.168.100.91
```

```
service "nginx" exposed
```

```
$ kubectl get svc,po
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
svc/kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP	3h
svc/nginx	LoadBalancer	10.101.25.10	192.168.100.91	80:31438/TCP	16s

NAME	READY	STATUS	RESTARTS	AGE
po/nginx-7cbc4b4d9c-w4zp8	1/1	Running	0	1m

這邊 type 可以選擇 NodePort 與 LoadBalancer，兩者的差異在於 NodePort 只映射 Host port 到 Container port，而 LoadBalancer 則繼承 NodePort 額外多出映射 Host target port 到 Container port。

確認沒問題後即可在瀏覽器存取 <http://192.168.100.91>。瀏覽器畫面如下：

Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

3.8.4 擴展服務數量

若叢集 node 節點增加了，而想讓 Nginx 服務提供可靠性的話，可以透過以下方式來擴展服務的副本：

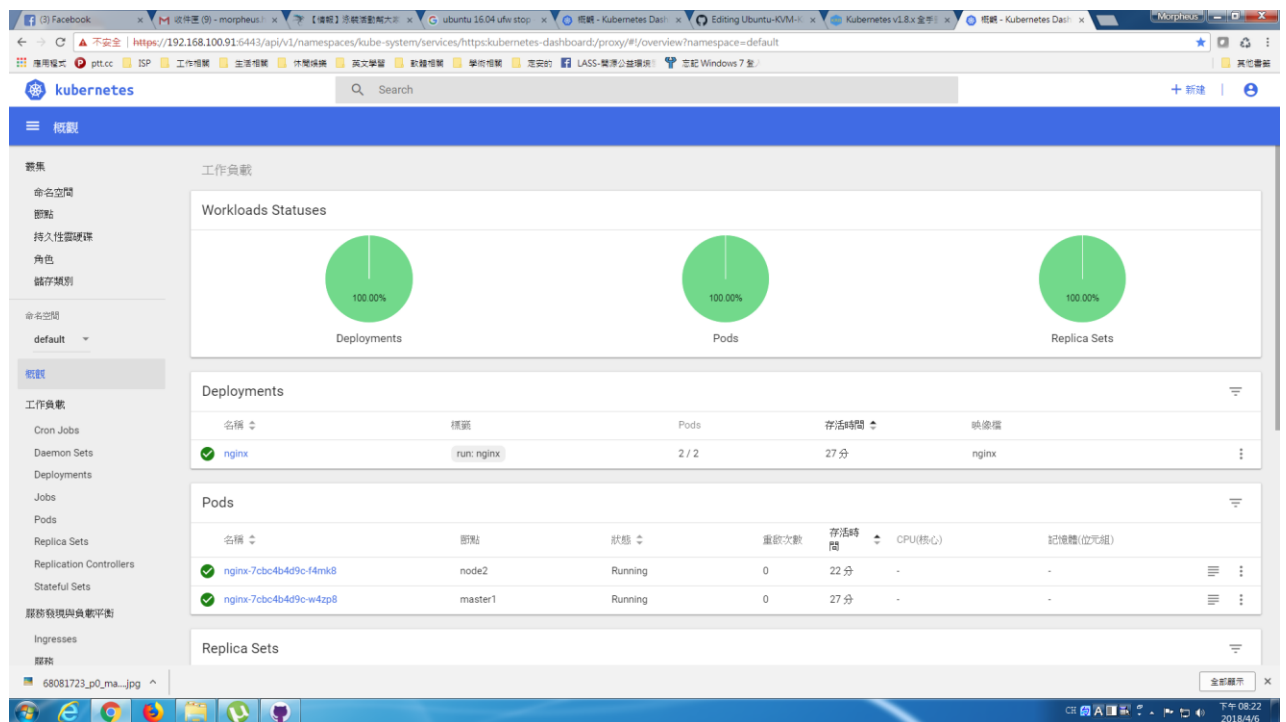
```
$ kubectl scale deploy nginx --replicas=2
```

```
deployment "nginx" scaled
```

```
$ kubectl get pods -o wide
```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
nginx-7cbc4b4d9c-f4mk8	1/1	Running	0	21s	10.244.104.4	node2
nginx-7cbc4b4d9c-w4zp8	1/1	Running	0	5m	10.244.137.65	master1

3.9 Kubernetes Dashboard 畫面：



參考文獻：

- [1] KVM Installation, online: <https://help.ubuntu.com/community/KVM/Installation>
- [2] 架設 Linux KVM 虛擬化主機, online: <http://www.lijyyh.com/2015/12/linux-kvm-set-up-linux-kvm.html>
- [3] Kubernetes v1.8.x 全手動苦工安裝教學, online: <https://kairn.github.io/2017/10/27/kubernetes/deploy/manual-v1.8/>
- [4] KVM and Libvirt on Ubuntu 16.04, online: <https://www.packet.net/developers/guides/kvm-and-libvirt/>