打造自己的深度學習計算環境

hzliu123

Published
with GitBook

目錄

打造自己的深度學習計算環境

本文告訴您如何在有限預算內，從硬體採購、組裝、作業系統選擇、軟體安裝與設定，自行打造與NVIDIA DIGITS DEVBOX相當的深度學習計算環境。

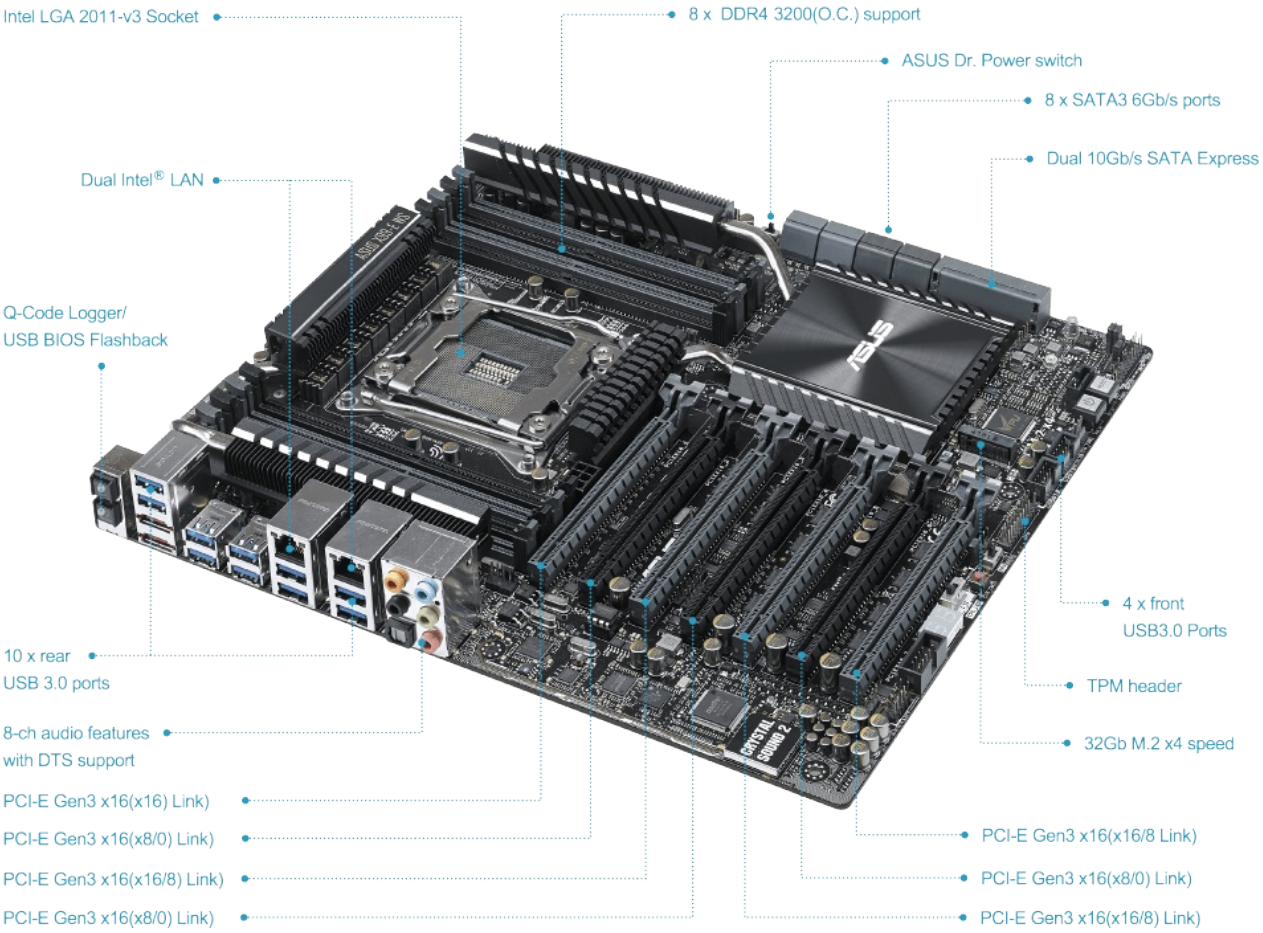
打造自己的深度學習計算環境

本文告訴您如何在有限預算內，從硬體採購、組裝、作業系統選擇、軟體安裝與設定，自行打造與NVIDIA DIGITS DEVBOX相當的深度學習計算環境。

硬體採購

參考NVIDIA DIGITS DEVBOX的設計 [1]，建購二台深度學習的高性能實驗環境。預算總額約15萬台幣。

主機板：ASUS X99-E WS(*2)，這塊主機板支援最多 PCIe 3.0 x16 四張顯示卡連結 (NVIDIA SLI)，可以提供良好的擴充性。如果有遠端管理的需求，ASUS X99-WS IPMI 也可以考慮。



GPU：GeForce GTX TITAN X 12GB(*2) ，Maxwell 系列省電且效能更好，單卡耗電約 250 瓦。因爲預算關係，只能買二張卡，每台電腦各一，未來有需求時再加購。

CPU：Intel Core i7-5930K(*2)，6核心，如選購其他型號，需注意 CPU 是否支援 40 PCIe 通道。耗電約 140 瓦。

CPU散熱：Corsair Hydro H60(*2)，單風扇水冷系統。

記憶體：32GB(每台)，DDR4-2133 Micron 16GB2(2)。記憶體採購需參考主機板廠商的QVL清單，以免買到相容性不佳的記憶體。

SSD：EZLINK CV-6 512G SATA3 (*2) 作爲系統碟。M.2 Kingston HyperX Predator PCIe HPM2280P2/240G 作爲 SSD cache，它的讀取速度達1400MB/s，這能改善資料從硬碟讀寫時的速度。

硬碟：Seagate 3TB 3.5 吋 SATA3 (ST3000VN0001) 企業級 NAS 碟 3個，組成 RAID5 陣列，可用空間 6 TB。

機殼：Thermaltake Core X9 平躺式機殼 (*2)，最下層放置電源供應器及硬碟，上層放置主機板、CPU、GPU，分層管理可以避免硬碟受到GPU散發的高熱影響。上層具備足夠的空間可讓風流順暢，用不到的 2.5/3.5 吋硬碟架可以移除來換取機箱空間，進一步提升散熱效能。前

置 200mm 超大靜音風扇，後置 120mm 標準風扇。上方及側面支援多組風扇支架，未來增購 GPU 時也有彈性再加強散熱。

電源供應器：振華 Leadex 850W (*2)，通過 80PLUS 認證，在各種負載下都有90%左右的效率。經費考量下，我們沒有選擇 1600W，但 850W 有空間讓我們再加一張GPU卡。

網路：預算考量下，原來考慮的 Infiniband 網路就沒列入了。ASUS X99-E WS 主機板內建雙埠 1000Mb ethernet 可以利用。

硬體組裝注意事項

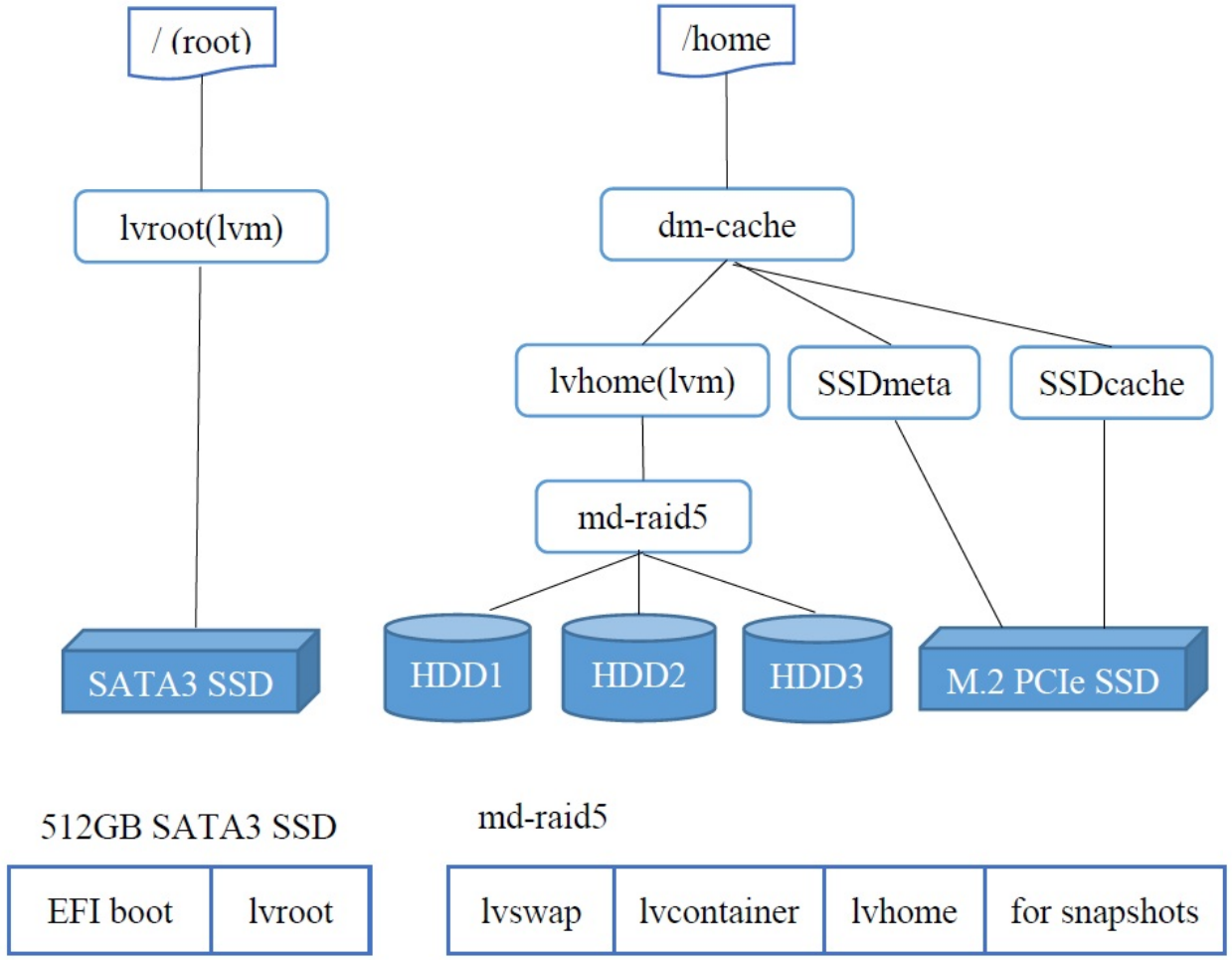
如果 M.2 插槽有安裝 SSD，NVIDIA TITAN X 必須避開 PCIEX16_4 插槽，因爲 M.2 SSD 插槽與 PCIEX16_4 共享頻寬 [2]。

作業系統選擇

一開始我們便選定使用 Ubuntu，主要是它的操作環境，多數人都已熟悉，且 NVIDIA devbox 也是使用 Ubuntu。曾經考慮是否使用 16.04，但 NVIDIA CUDA 尚未正式支援 16.04，不希望一些 bug 影響到系統的穩定性，所以還是選擇 14.04。待日後確定CUDA 支援16.04 再升級。16.04 有不少新功能，其中較令人期待的是 ZFS [16] 檔案系統的支援；它能同時滿足資料卷的管理、磁碟陣列的容錯及資料夾快照備份等需求。在 14.04，我們會先以 ext4 + lvm + md 來達成這些需求。

這系統主要是提供多人遠端使用，不需要圖形介面的環境，因此選擇沒有圖形介面的 Ubuntu 14.04.4 Server。在安裝完後第一次開機會沒有畫面，原因是open source nvidia driver 支援的 GPU 卡有限，需在 GRUB 裡加上 kernel 的參數 nomodeset。

儲存系統規畫



最底層 3 個硬碟，組成一個 R5 的磁碟陣列，即使壞掉一個硬碟，資料也不會遺失。但會有相當一個硬碟的空間儲存 parity，實際可用空間只 3TB+3TB=6TB。md-raid5 的 6TB 會與 EZLINK SATA3 SSD 的 512GB，再組成一個 lvm volume group。將 SSD 空間與硬碟空間劃分在一起的原因是希望在 SATA3 SSD 上的 volume 也能被 lvm snapshot 備份到硬碟上。lvhome 規劃約2.5T空間，剩餘大部份的空間留做未來彈性使用；除了做爲 snapshot 空間，也可以在 lvhome 空間不夠時分給它。SATA3 SSD 上除了 EFI boot partition，所有空間都分給 lvroot。

安裝 mdadm，它是 Linux software RAID 的管理程式。下面指令會將三個硬碟組成一個 R5 的磁碟陣列。

```
mdadm --create /dev/md0 --level=5 --raid-devices=3 /dev/sd[a-c]
```

LVM [12] 的操作主要在三個物件上：PV (Physical Volume)、VG (Volume Group)、LV (Logical Volume)。PV 代表可用的實體磁碟，VG 是這些實體磁碟的總合。而 LV 是邏輯磁碟，它的儲存空間是從某個 VG 配置而來。LV 的空間可依實際需求，配合檔案系統做長大或縮小。

因爲 root filesystem 是存放在 lvm 裡的 lvroot volume，以下 LVM 操作須在安裝 Ubuntu 前先完成。

```
pvcreate /dev/md0     # 將md0空間設爲 PV
```

使用 parted 將 512GB SATA3 SSD (/dev/sdd) 分割成 2 塊，分別給 EFI boot 及 lvm。

```
pvcreate /dev/sdd2    # 將 lvroot 空間設爲 PV
vgcreate vg0 /dev/md0 # 創造一個 VG vg0, 並將 md0 & sdd2 納入
vgextend vg0 /dev/sdd2

# 從 vg0 創造出 lvroot, 但它必須使用 sdd2 100% 的空間
lvcreate -n lvroot -l 100%PVS vg0 /dev/sdd2

lvcreate -n lvswap -L 32G vg0
lvcreate -n lvhome -L 2500G vg0
lvcreate -n lvcontainer -L 500G vg0
```

我們有二台機器一台稱 gc1，一台稱 gc2，以上架構爲 gc1 的規畫，gc2 沒有自己的硬碟，會透過 NFS 存取 gc1:/home 及 gc1:/container 的資料。

SSD 快取

SSD 快取可以縮短讀寫硬碟的等待時間。SSD 不像硬碟需要移動磁頭才能存取到資料；將需要做大量磁頭移動的資料，儲存在 SSD 上操作，可以有效減少資料讀寫時間，加速程式系統運行。

Linux kernel 支援二種 SSD 快取：bcache 及 dm-cache [11]。這二種快取的效能相當，但 bcache 不方便使用在已存在的檔案系統。lvm-cache [10] 採用 dm-cache，操作 lvm 指令就能簡單設定好 SSD 快取，但 Ubuntu 到 14.04 不支援 lvm-cache。這邊採直接使用 dm-cache 方式。

dm-cache 需要 3 個子裝置來組成一個有快取的虛擬裝置：ssdcache-metadata, ssdcache-blocks 及要被加速的裝置 (這邊是 lvhome)。使用 dmsetup (device-mapper) 將 M.2 PCIe 240GB SSD 儲存空間分割給 ssdcache-meta 及 ssdcache-blocks。下面是空間大小的計算 (dmsetup 參數都是以 sector 爲單位)：

```
cache block size: 512KB (配合 md chunk size 512KB)
ssdcache size: 468862128 sectors (M.2 PCIe SSD 的大小)
ssdcache-metadata size: 512000 sectors (~= 1/1000 of ssdcache size)
ssdcache-blocks size: 468350128 sectors
lvhome size: 5242880000 sectors
```

使用 dmsetup [7] 來切割與設定 dm-cache (/dev/sde = M2. PCIe SSD)

```
# 建立 ssdcache-metadata，它位於 /sde 前 512000 sectors
dmsetup create ssdcache-metadata --table '0 512000 linear /dev/sde 0'
# 將 ssdcache-metadata 內容清除爲 0
dd if=/dev/zero of=/dev/mapper/ssdcache-metadata

# 建立ssdcache-blocks，它位於/sde 第 512000 sector 起以後的所有空間
dmsetup create ssdcache-blocks --table '0 468350128 linear /dev/sde 512000'

# 建立 lvhome-cached，cache block size = 1024 sectors (512KB)，cache policy = writeback
dmsetup create lvhome-cached --table '0 5242880000 cache /dev/mapper/ssdcache-metadata /d
```

接著就可以把 lvhome-cached mount 起來用用看，沒有問題再更新 /etc/fstab，讓 lvhome-cached 在開機時可以被自動 mount 起來。我們還會需要設定 upstart init script 在開機時自動執行上列 dmsetup 設定，並在關機時將尚未寫入硬碟的資料寫入 [6]。

/etc/init/ssd-cache.conf

```
description "Configure dmcache for SSD cache"
start on virtual-filesystems
task

script
    dmsetup create ssdcache-metadata --table '0 512000 linear /dev/sde 0'
    dmsetup create ssdcache-blocks --table '0 468350128 linear /dev/sde 512000'
    LVHOME_SIZE=`blockdev --getsz /dev/mapper/vg0-lvhome`
    dmsetup create lvhome-cached --table "0 $LVHOME_SIZE cache /dev/mapper/ssdcache-metad
    dmsetup resume lvhome-cached
end script
```

/etc/init/ssd-cache-wait.conf

```
description "Suspend dmcache for SSD cache"
start on mounted MOUNTPOINT=/home

script
    inotifywait -e unmount $MOUNTPOINT
    dmsetup suspend lvhome-cached
    dmsetup remove lvhome-cached
    dmsetup remove ssd-blocks
    dmsetup remove ssd-metadata
end script
```

備份計畫

週期性的備份計畫能避免意外時失去寶貴的資料。針對 Ubuntu 系統 (lvroot) 及 /home (lvhome)，排程在 crontab 裡，每週自動產生新的 lvm snapshot，並刪除過舊的 snapshot。snapshot 只有在舊資料要被覆寫時，才會做備份 (copy-on-write)，這樣的備份方式既省時又經濟。

但目前 Linux kernel dm-snapshot 的實作沒有效率，snapshot 間並沒有共享備份資料。一個資料覆寫，如與 3 個 snapshot 有關，就會變成要多寫 3 倍的的資料量。同一個 volume 上，不要有太多的 snapshot，因爲數量多時，會嚴重影響寫入硬碟的速度。我們計畫在 16.04 ，以 ZFS 取代 lvm & md。

以 root 身份執行 crontab -e，下面的例子是每週對 lvroot 做 snapshot，系統最多只留一份 lvroot 的 snapshot。

```
@weekly /root/bin/snapshot.sh weekly 1 vg0 lvroot 20G
```

snapshot.sh 在這裡下載：http://bellot.net/cyril/contribs/snapshots-scripts。

網路環境

如果有分散式的計算需求，快速的網路環境可大幅縮短深度學習的計算時間。我們會在未來有需求時，再增購 infiniband 設備。這裡，因 X99-E WS 有雙 ethernet port，我們直接使用一條網路線將二台電腦連接起來。二台電腦間的溝通會直接透過這條線。

NFS 網路檔案系統

爲方便檔案管理與使用，gc1 同時也是檔案伺服器。它的 /home 及 /container 會透過 NFS 分享給 gc2。gc1 上需安裝 nfs-kernel-server package，並在 /etc/exports 裡將上述二個目錄 export 出來讓 gc2 使用。

```
/home      gc2.dt42.io(rw,no_subtree_check,no_root_squash,async)
/container gc2.dt42.io(rw,no_subtree_check,no_root_squash,async)
```

async 參數能改善透過 NFS 寫入大量小檔案時，速度過慢的問題 [5]。它允許 NFS server 在未實際將檔案寫入硬碟前，就回報 NFS client 已完成。no_root_squash 允許 gc2 以 root 身份存取 gc1 exports。

gc2 是 NFS client，必須在 gc2 上安裝 nfs-common package 及修改 /etc/fstab 如下：

```
gc1.dt42.io:/home       /home       nfs    defaults    0    0
gc1.dt42.io:/container  /container  nfs    defaults    0    0
```

LDAP帳號集中管理

集中管理使用者帳號，可以減少管理上的負擔與增進系統安全。曾經考慮 NIS，但 NIS 多年前就不再發展。考慮未來管理上的需求，我們採用 OpenLDAP。

LDAP (Lightweight Directory Access Protocol) 是一種 Internet 上通訊協定，用來存取以 X.500 爲基礎的目錄服務，而 OpenLDAP 是開放原始碼的 LDAP 實作，它提供 LDAP server、函式庫以及各種操作工具。

這份文件完稿時，Ubuntu 16.04 已發行，如果你使用 16.04，建議改用 FreeIPA。FreeIPA 是另一種開放原始碼的實作，相對於 OpenLDAP，它安裝簡便、容易使用。管理者的操作介面除了傳統的命令列外，還有 Web UI。使用者也能透過 Web UI，查詢或更改個人資料及密碼。

在安裝 OpenLDAP 前，有些小提醒。我們的機器是遠端管理，當 LDAP 或 NFS 出了問題，很可能造成機器無法登入。務必保留一個可執行 **sudo** 的本地帳號，且其家目錄不在 **NFS** 上，供管理者緊急救援使用。

安裝前須先更改 /etc/hosts 中 127.0.1.1 的名稱對應，因爲 slapd 會用這個決定 DN suffix(eg. dc=dt42,dc=io)

```
127.0.1.1    gc1.dt42.io    gc1
```

安裝 slapd (LDAP server)及 ldap-utils (LDAP 資料庫的操作工具)，過程中會要求輸入 rootDN 管理者 (cn=admin,dc=dt42,dc=io)的密碼。接著編寫一 slapindex.ldif 內容如下，以修正 slapd database index 的問題。

```
dn: olcDatabase={1}hdb,cn=config
changetype: modify
add: olcDbIndex
olcDbIndex: uid,cn,uidNumber,gidNumber,memberUid,uniqueMember eq
```

```
ldapmodify -QY EXTERNAL -H ldapi:/// -f slapindex.ldif
```

這樣 LDAP server 就算安裝完成。

安裝 ldapscripts，這些 scripts 包裝 ldap-utils 裡的指令，使之更容易使用。參考下列設定，修改 /etc/ldapscripts/ldapscripts.conf。

```
SUFFIX="dc=dt42,dc=io"
BINDDN="cn=admin,dc=dt42,dc=io"
BINDPWDFILE="/etc/ldapscripts/ldapscripts.passwd"
RECORDPASSWORDS="yes"
PASSWORDFILE="/var/log/ldapscripts_passwd.log"
CREATEHOMES="yes"
```

將 rootDN 的密碼寫入 /etc/ldapscripts/ldapscripts.passwd 裡。此檔內不允許有換行符號 [8] (不可使用 vim 去修改)，需用指令更新：

```
sh -c “echo -n ‘secret’ > /etc/ldapscripts/ldapscripts.passwd”
chmod 400 /etc/ldapscripts/ldapscripts.passwd
```

這樣 ldapscripts 就可以使用了。

使用 ldapinit 建立存放帳號資訊的樹狀結構。

```
ldapinit -s
```

開始建立帳號：

```
ldapaddgroup dt42                # 建立 dt42 group
ldapadduser joseph dt42          # 增加一個 user:joseph,group=dt42
ldapaddusertogroup joseph docker # 將 joseph 加入其他需要的 group
```

設定 Linux 使用 LDAP 認證

還需要安裝 libnss-ldap 在會使用 LDAP 認證的 Linux client 上。安裝過程中會問 LDAP server 的設定，如果答錯了，可以用下面指令重來：

```
dpkg-reconfigure ldap-auth-config
```

設定結果都會存在 /etc/ldap.conf 裡。接著設定 NSS (名稱服務切換器) 使它切換到 LDAP。

```
auth-client-config -t nss -p lac_ldap
```

最後更新 PAM，選取 LDAP，選單上其他認證方式，如會用到也可以選取。這樣剛才新增的帳號就能登入了。

```
pam-auth-update
```

Ubuntu 14.04 有 bug，使用者無法用熟悉的 passwd 指令更改自己的密碼。編輯 /etc/pam.d/common-password，將 use_authtok 這個關鍵字移除就可以解決 [9]。

爲了避免單一 LDAP server 停止運作，造成所有的機器都無法登入，我們在 gc1 & gc2 上都安裝了 slapd，並讓這二台機器彼此同步更新它們的 LDAP 資料庫。當 LDAP 認證無法在時限內從 gc1 上取得時，gc2 可以當備援。另外，我們也開啓 TLS 加密，讓 LDAP 認證資訊的傳送能更安全。

LDAP 資料庫複製備援與 TLS 的相關設定，有興趣的讀者可以參考 [4]。

Docker 安裝

機器學習常會使用到 docker，但讓使用者使用 docker，等同於把 root 權限交給使用者 [13]。但目前 docker daemon 在 AppArmor/LXD 下運作尚有許多問題 [14][15]，無法使用 AppArmor/LXD 來保護系統。使用者在使用如 bind mount 等功能時，須小心以免損壞系統檔案，特別是測試來路不明的 docker image。

輸入下列指令安裝 docker：

```
apt-get install curl
curl -fsSL https://get.docker.com/ | sh
```

測試 docker 是否成功安裝：

```
docker run hello-world
```

CUDA toolkit 安裝

CUDA 是個運用 GPU 做平行計算的環境。到下列網址下載網路安裝的 meta package: https://developer.nvidia.com/cuda-downloads。

輸入以下指令安裝 CUDA，在安裝完畢後重新開機：

```
dpkg -i cuda-repo-ubuntu1404_7.5-18_amd64.deb
apt-get update
apt-get install cuda
```

設定環境參數，將以下內容寫入 /etc/profile.d/ 或 ~/.profile

```
export PATH=/usr/local/cuda-7.5/bin:$PATH
export LD_LIBRARY_PATH=/usr/local/cuda-7.5/lib64:$LD_LIBRARY_PATH
```

輸入以下指令，安裝 CUDA 範例程式到 ~/NVIDIA_CUDA-7.5_Samples/：

```
cuda-install-samples-7.5.sh ~/
```

在 NVIDIA_CUDA-7.5_Samples/ 目錄下執行 make，編譯所有的範例程式。

在子目錄下找到 deviceQuery 這支程式，它可以測試是否 CUDA 正確安裝、可正常運作。CUDA 更進一步的資訊，請參閱 [3]。

記憶體頻寬測試

CUDA 範例程式裡，有一支程式 bandwidthTest 可以測試 GPU 與系統間資料的傳輸速度。PCIe 3.0 x16 的理論值接近 16GB/s。但下面數據顯示，GPU 與系統記憶體之間的傳輸速度，理論值的一半都不到，只有 6GB/s。

```
[CUDA Bandwidth Test] - Starting...
Running on...

 Device 0: GeForce GTX TITAN X
 Quick Mode

 Host to Device Bandwidth, 1 Device(s)
 PINNED Memory Transfers
   Transfer Size (Bytes)        Bandwidth(MB/s)
   33554432                     6339.4

 Device to Host Bandwidth, 1 Device(s)
 PINNED Memory Transfers
   Transfer Size (Bytes)        Bandwidth(MB/s)
   33554432                     6462.6

 Device to Device Bandwidth, 1 Device(s)
 PINNED Memory Transfers
   Transfer Size (Bytes)        Bandwidth(MB/s)
   33554432                     247301.2

Result = PASS
```

進一步以 lspci 分析，發現NVIDIA TITAN X 並沒有使用到 16 通道，只有使用到 8 個 (Width x8)。原因是這張卡被安裝在 PCIEX16_4 上。在調整安裝位置之後，host to device bandwidth 的數值可提升一倍到 12GB。

```
09:00.0 VGA compatible controller: NVIDIA Corporation GM200 [GeForce GTX TITAN X] (rev a1
…
    LnkCap: Port #16, Speed 8GT/s, Width x16, ASPM not supported, Exit Latency L0s <512ns
            ClockPM+ Surprise- LLActRep- BwNot-
    LnkCtl: ASPM Disabled; RCB 64 bytes Disabled- CommClk-
            ExtSynch- ClockPM+ AutWidDis- BWInt- AutBWInt-
    LnkSta: Speed 2.5GT/s, Width x8, TrErr- Train- SlotClk+ DLActive- BWMgmt- ABWMgmt-
```

參考資料

1. NVIDIA DIGITS DevBox
2. ASUS X99E-WS User Guide
3. CUDA Toolkit Documentation v7.5
4. Ubuntu Documentation - OpenLDAP Server
5. Linux man pages - exports(5)
6. SSD Caching Using dm-cache Tutorial
7. Kernel documentation - dm-cache

8. ldapscripts doesn't use /etc/ldap.secret correctly (LP: #1090328)
9. Cannot change password on ldap client (LP: #329067)
10. Linux man pages — lvmcache(7)
11. bcache and/vs. LVM cache
12. Ubuntu wiki - LVM
13. Using the docker command to root the host
14. AppArmor for Docker daemon
15. Get docker to run inside LXD
16. Ubuntu wiki - ZFS