目錄

[**一、虛擬化技術簡介**](#_xii8eefvola4) **1**

[**二、硬體虛擬化與作業系統層級虛擬化**](#_ewskh4mv7gm4) **1**

[硬體虛擬化](#_cqrpy451hhau) 1

[VMM (Hypervisor) 介紹](#_6tf3lgjxsadz) 1

[全虛擬化與半虛擬化](#_hkcup32a3pd8) 2

[作業系統層級虛擬化 (容器化)](#_8jsawn7hf8q0) 2

[**三、Qubes OS**](#_x48k1zfr3gfm) **2**

[Xen 基本介紹](#_gcge257bf7or) 3

[Qubes 架構概觀](#_tsr1447qxxld) 3

[Qubes Networking 簡介](#_nbz139sh2cmi) 6

[**四、Docker**](#_mzrgu6y7e3xd) **6**

[Docker 架構概觀](#_blwy7wmtbsdu) 7

[Docker Daemon](#_tdhgcr90dubp) 7

[Docker Client](#_dzsuzues4e2j) 8

[Docker Registries](#_jbow68dpwf8e) 8

[Docker Objects](#_unvrzs8s6pth) 8

[Images](#_a07oa1qyvg2n) 8

[Containers](#_1c8rs2bnakc1) 8

[Docker 核心技術](#_1qr0gj6dihz7) 9

[Namespace](#_268jjqrmf5z8) 9

[Control Group (cgroup)](#_ds7k68p5xxns) 10

[**參考資料**](#_b4ag1547pffs) **11**

# 一、虛擬化技術簡介

虛擬化技術 (Virtualization) 是將實體資源 (例如 CPU、記憶體或硬碟儲存空間) ，予以抽象、轉換後呈現出來並可供分割、組合為一個或多個電腦組態環境，使用戶可以比原本的組態更好的方式來應用這些硬體資源的一種技術。

虛擬化技術可分成多種類別，以虛擬的對象來分類的話，可分為硬體虛擬化和網路虛擬化等等。而本報告針對的是虛擬化技術類別中的硬體虛擬化 (hardware virtualization) 和作業系統層級虛擬化 (OS-level virtualization) 的部份。

# 二、硬體虛擬化與作業系統層級虛擬化

## 硬體虛擬化

硬體虛擬化 (又稱平台虛擬化) 是將一台實體機器虛擬化為一台或多台虛擬機器，每個虛擬機器都擁有自己的虛擬硬體 (包括 CPU、記憶體以及週邊設備等)，以便提供一個獨立的虛擬機器執行環境。通過虛擬化層的模擬，虛擬機器中的作業系統認為自己仍然是獨占一個機器在運行。每個虛擬機器中的作業系統可以完全不同，其執行環境也是完全獨立。這個虛擬化層就是 VMM (Virtual Machine Monitor)。

### VMM (Hypervisor) 介紹

VMM 又稱為 hypervisor，而 hypervisor 又分成兩種：type 1 (或稱 bare-metal hypervisor) 以及 type 2 (或稱 hosted hypervisor)。type 1 hypervisor 運行於硬體之上 (硬體必須能夠支援)，等同作業系統，直接操控硬體與管理虛擬機器，運作效能高於 type 2，例如 Microsoft Hyper-V、Xen (3.0 以後版本) 就屬於 type 1 hypervisor。type 2 hypervisor 運作於主機的作業系統之上，像是一般運作於作業系統的應用程式那樣，例如 VMware Workstation、VirtualBox 就屬於 type 2 hypervisor。兩種 hypervisor 的結構如下圖 (VM 代表 virtual machine)。



### 全虛擬化與半虛擬化

硬體虛擬化可以分為以下的類別：全虛擬化 (full virtualization) 與半虛擬化 (paravirtualization)。全虛擬化是利用 Intel VT 與 AMD-V 等 CPU 虛擬化支援功能，將 PC 硬體上運作的作業系統 (不用做任何修改) 直接拿到虛擬機器環境下執行的方法；半虛擬化則是客體作業系統 (guest OS) 需要知道自己在虛擬化環境下執行，kernel 與驅動程式必須修正。執行於兩種虛擬化類型的 hypervisor 之上的客體作業系統的比較如下圖。



## 作業系統層級虛擬化 (容器化)

而作業系統層級虛擬化又稱為容器化 (containerization)，主要依賴作業系統 kernel 虛擬化的支援，讓多個 user-space instances 得以在宿主機作業系統 (host OS) kernel 中各自獨立運行，每個承載 user-space instance 的資源空間就稱為容器 (容器亦可稱為虛擬引擎 (virtualization engine) )。宿主機 (host machine) kernel 本身也具備資源管理功能，盡量讓各個容器之間隔離運行，將其他軟體容器的造成的交互影響最小化。因此，宿主機中的運算、儲存、記憶、網路等資源，對不同容器中所執行的程序來說，就好像是自己專用的一樣，而容器中的程序只能看得到其所屬的容器被分配的資源。

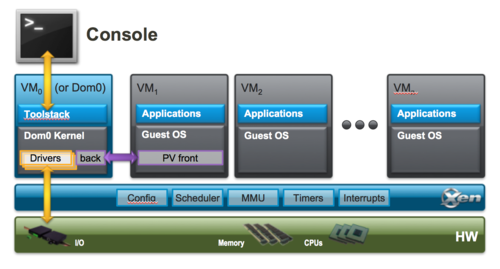
接下來的研究內容，將以 Qubes OS 和 Docker 作為研究標的 ( Qubes OS 為硬體虛擬化 (全虛擬化的部份) 的實作，而 Docker 則是容器化的實作)。

# 三、Qubes OS

Qubes OS 是一個以安全為重點的 (security-focused) 的桌面作業系統，其目標是透過隔離 (isolation) 來提供安全。而其開發團隊認為虛擬化 (virtualization) 是目前實際上最能達到強效隔離的效果，同時提供現有應用程式和驅動程式相容性。Qubes OS 是藉由 Xen 來達到虛擬化。在深入了解 Qubes OS 之前，首先必須先對 Xen 有所認知。

## Xen 基本介紹

Xen 的系統架構如下圖。Xen (Project) hypervisor 直接在硬體上執行並且負責管理 CPU、記憶體、interrupts，它是 bootloader 被執行後第一個載入的程式。而下圖中在 Xen hypervisor 之上有好幾個正在執行的虛擬機器，這些虛擬機器又稱為 domain (或是 guest)。在這些 domains 當中，domain 0 (最左邊的 domain) 擁有要求 hypervisor 對虛擬機器環境做出改變、進行控制的特權 (藉由 control stack (即圖中 Toolstack 的部份) 來建立、刪除以及設定其他 domain )，同時也含有系統中所有裝置的驅動程式。



(圖片來源：https://wiki.xenproject.org/wiki/Xen\_Project\_Software\_Overview)

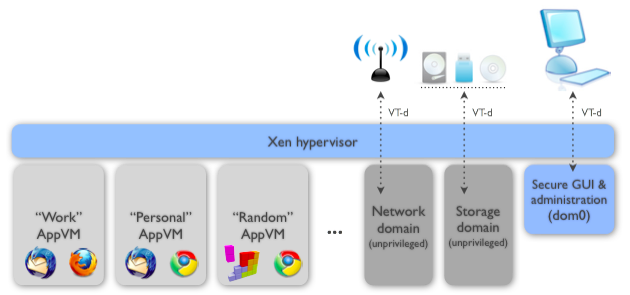
接著介紹 domain 有以下種類 (guest type)：

* PV guests
* HVM guests
* PVH guests

PV guest (Paravirtualized guest) 即是指採用半虛擬化技術所建立的虛擬機器，其客體作業系統 kernel 和 driver 必須修改過才能被 Xen hypervisor 察覺 (Xen-aware)；HVM guest 指的是採用全虛擬化技術所建立的虛擬機器，硬體必須支援虛擬化 (例如 Intel VT 或是 AMD-V)；PVH guest 則是採用修改過的客體作業系統以及有支援虛擬化的硬體來建立虛擬機器。

## Qubes 架構概觀

介紹完 Xen 的基本概念後，再來就是正式開始來看 Qubes OS。下圖為 Qubes OS 的系統架構。Xen hypervisor 上方是各種硬體及週邊，例如網路卡、儲存裝置和 I/O 設備等；而 Xen hypervisor 下方的即是運行中的各個虛擬機器 (在 Qubes 的術語中又稱為 qube)，其中又可區分為 AppVM、network domain、storage domain 和 dom0，後續將一一介紹。



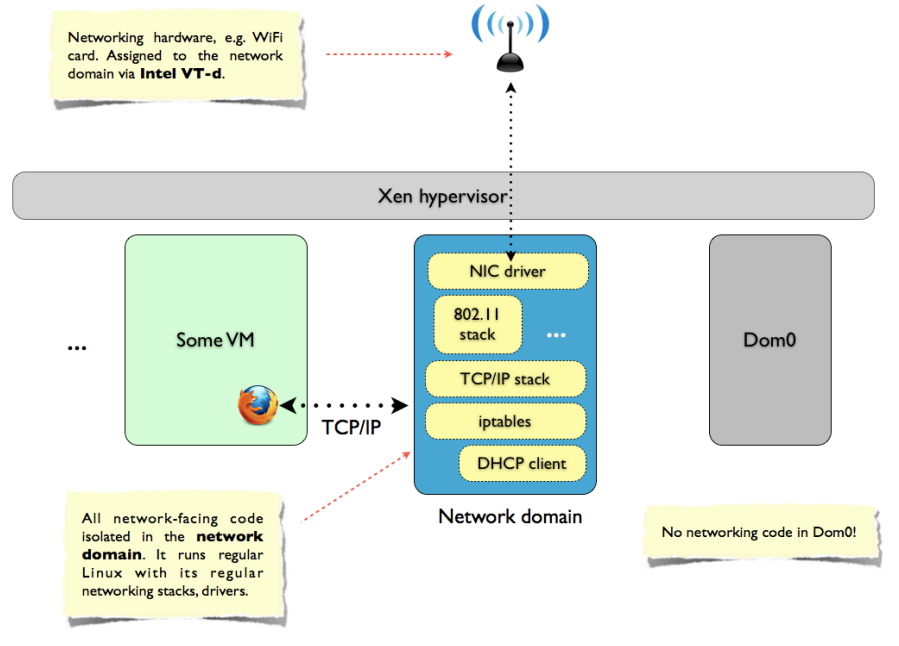
(圖片來源：https://www.qubes-os.org/doc/architecture/)

AppVM 是指用來執行 user 所需的應用程式的虛擬機器，所有的應用程式都在 AppVM 中執行。

在開始介紹 dom0 前，首先介紹一些 Qubes 中的術語。Qubes 中的 dom0 與 Xen 中的 domain 0 意義上相同，都能管理其他 domain 且可對硬體做直接存取。在 Qubes 中的 domain，其意義是「由某些 qubes 為了特定安全需求所組成」的應用服務。舉例來說，若今天有一個 e-mail domain 提供寄送 PGP 加密的電子郵件的功能，那此 domain 就是由 e-mail qube 和 Split GPG qube 組成；不過 network domain 指的就是 Qubes OS 中負責網路的 qube。因此 domain 是屬於哪一種定義，必須根據文章的上下文進行判讀。

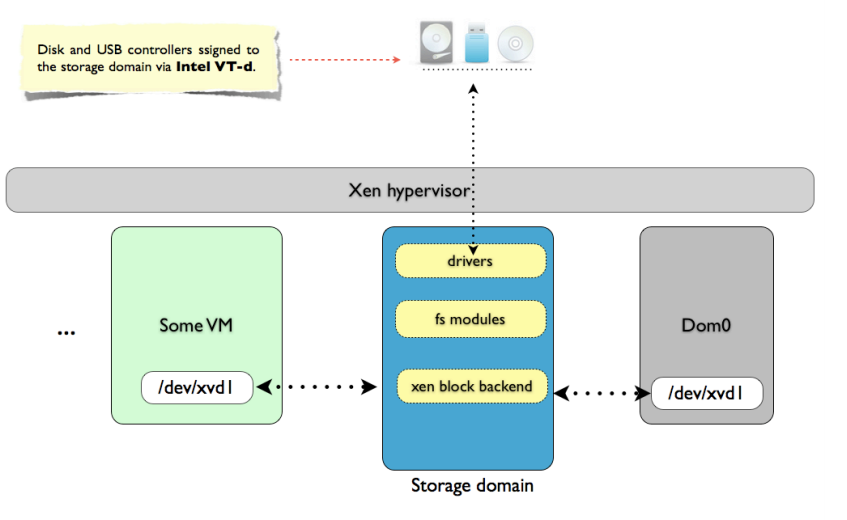
dom0 負責管理其他 domain，有跟 hypervisor 幾乎一樣的特權 (privilege)，其中為了讓受到攻擊表面 (attack surface) 減至最小，dom0 不包含 networking 相關的程式碼，但有包含與各個虛擬機器溝通的介面，如 GUI 和 xenstore daemon (含有系統中的 registry，像是哪個 backend device 位於哪裡等等的資訊，可用於系統管理)。除了前述與虛擬機器溝通的介面外，dom0 中就沒有其他的溝通介面，也因此沒有任何 backend drivers，所以 network 和 storage 功能另外移出到其他虛擬機器中 (network domain 和 storage domain)。

network domain 負責執行與 protocol stacks (例如 802.11、TCP/IP 等) 和網路卡驅動程式的程式碼，並利用 Intel VT-d 等技術來直接存取網路硬體設備 (如下圖)。會將這些程式碼獨立出來在一個虛擬機器執行的原因，是為了即使因為某些漏洞，例如網路卡驅動程式中的 bug，導致被駭客取得 network domain 的控制權的情況下，不致於讓整個系統被攻陷。本章節的最後會更進一步介紹 Qubes networking。



(來源：https://www.qubes-os.org/attachment/wiki/QubesArchitecture/arch-spec-0.3.pdf)

storage domain 中有 Xen block backend 可被用來輸出 (export) 其他的虛擬機器 (包含 dom0) virtual block device，並透過 Intel VT-d 技術來存取硬碟或是其他可移除的儲存裝置 (如下圖所示)。將 file system sharing code、backends 以及 disk drivers 移出 dom0 到 storage domain 中，可減少系統的攻擊表面。



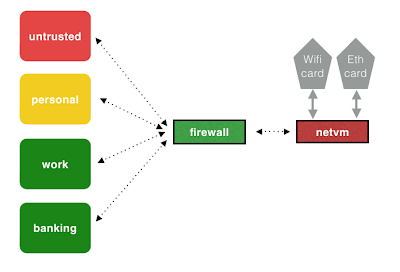
(來源：https://www.qubes-os.org/attachment/wiki/QubesArchitecture/arch-spec-0.3.pdf)

## Qubes Networking 簡介

系統的 networking 在資安防護上是常常是重點項目，若後續要進行這方面的研究，那麼對 Qubes networking 有基本的認識是必須的。

下圖是預設的 Qubes networking topology。從 networking 的觀點來看，Qubes 中的虛擬機器可以分為三種：Net VMs，AppVMs 以及 Proxy VMs。Net VM (如下圖中的 netvm) 有 networking 裝置 (例如 WiFi/ethernet card) 被指派給它，且包含一個 Xen network backend，該 backend 提供 networking 的能力給其他連結到此 Net VM 的其他虛擬機器。AppVM 指的就是那些連結到 Net VM 以取得連網能力的虛擬機器。而 Proxy VM 則是 Net VM 和 AppVM 的結合，對 Net VM 來說，它是需要連網能力的 AppVM；而對連到它的 AppVMs 來說，它又決定這些 AppVMs 能不能連到 Net VM (如下圖中的 firewall)。Proxy VM 可以根據設定的 firewall rules 來管理那些連到它的虛擬機器的網路連線。

此外，連到同一個 Net/Proxy VM 的 AppVMs，這些 AppVMs 之間的 traffic 基本上預設是無法流通的 (被阻擋)。



(圖片來源：http://theinvisiblethings.blogspot.com/2011/09/playing-with-qubes-networking-for-fun.html)

# 四、Docker

Docker 是一個開源容器運行引擎與容器資源抽象化的工具，讓開發人員或系統管理員能創建、打包、運行容器中的應用程式。開發者能用 Docker 分層打包其應用程序和會用到的以及依存的函式庫或程序包，成為所謂的 Docker image，並能夠發布在任何其他具備 Docker engine 的環境，因此可讓應用服務輕易地跨實體機或虛擬機以及在不同 Linux 發行版的主機上運行。

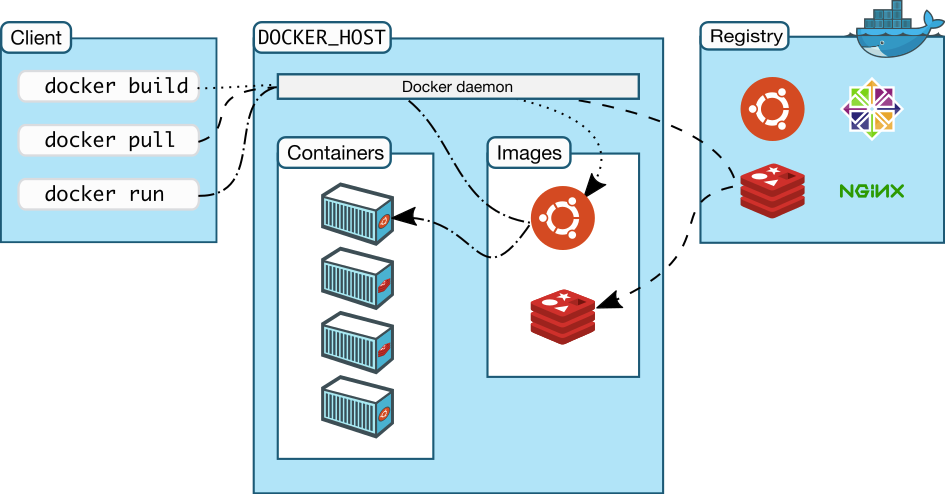
由於 Docker 是容器技術的一種實作，因此具備前述的「輕量化」運行的特質。從下圖中的左圖 (Docker container) 和右圖 (傳統虛擬機器) 可比較出來，因為容器是直接分配宿主機作業系統的資源來運行，因此能以 Docker engine 將實體資源抽象化後直接分配給應用程序；另外因容器內外共用相同的 Linux kernel，因此可省去客體作業系統。以下將介紹 Docker 的架構概觀以及其核心技術。



(圖片來源：https://docs.docker.com/get-started/#containers-and-virtual-machines)

## Docker 架構概觀

Docker 使用 client-server 架構。Docker client 讓使用者可以與 Docker daemon 進行通訊，而 Docker daemon 主要負責建立、執行和分佈 (distribute) Docker containers。Docker client 和 Docker daemon 可以在同一個機器上運行，也可以是兩者不在同一個系統上，client 連線到遠端的 daemon。Docker client 和 daemon 之間的溝通是使用 REST API，透過 Unix sockets 或 network interface 進行通訊。以下將進一步詳細介紹架構圖中主要的部份。



(圖片來源：https://docs.docker.com/engine/docker-overview/#docker-architecture)

### Docker Daemon

Docker daemon 負責監聽 Docker API requests 以及管理 Docker objects (如 images、containers 和 networks)。

### Docker Client

Docker client 是使用者與 Docker 互動的主要方式。當使用者下達像是 "docker run" 的命令時，client 會將命令送給 Docker daemon，daemon 會負責完成執行該命令。

### Docker Registries

Docker registry 存放 Docker images。Docker Hub 和 Docker Cloud 屬於公開的 Docker registry，任何人都可以使用；另外 Docker 要找欠缺的 images 來用時，預設是到 Docker Hub 來找。甚至使用者也可以有屬於自己、私人的 registry。

當使用者下達如 "docker pull" 或是 "docker run" 的命令，需要用到的 images 就會從設定的 registry 抓下來；而若是下達 "docker push" 的命令，使用者欲上傳的 image 就會存放到設定的 registry 上。

### Docker Objects

以下僅介紹兩個主要的 objects：images 和 containers。

#### Images

image 指的是用來建立 Docker container 的唯讀樣板 (read-only template)。可在基於另一個 image 的情況下，加上其他的客製化來建立所需的 image。舉例來說，今天需要一個提供 web server 功能的 image，除了基於 Ubuntu image 之外，另外還要加上像是 Apache web server 和自己需要的應用服務，以及相關的組態設定，才能做出所需的 image。

當 registry 上找不到所需的 image 時，這時只能自己建立。建立 image 需要用到 Dockerfile，其中的指令定義了建立和執行的步驟。在 Dockerfile 中的每道指令都會建立 image 中的一個分層 (layer)；當 Dockerfile 被修改以及重新建立 (rebuild) image 後，只有變動的那些分層才需要重新建立。這也是為什麼 image 相比於其他虛擬化技術可以這麼輕量化 (lightweight) 的其中一個原因。

#### Containers

一個 container 其實就是一個 image 的可執行的實體 (runnable instance)。使用者可透過 Docker API 或是 CLI 來操作 container (例如建立、啟動、停止或是刪除 container)。

以下使用 "docker run" 命令為例子進行建立 container 的解說，下面的例子執行一個 Ubuntu container，並且附著 (attach) 到本地端的 command-line session 並執行 bash。

$ docker run -i -t ubuntu /bin/bash

當執行上述命令時，會發生以下情況 (假設你正在使用預設的 registry 設定)：

1. 如果本地端沒有 Ubuntu image，Docker 會從預設的 registry 拉下來本地端，彷彿你手動下命令 "docker pull" 一樣。
2. Docker 創見一個新的 container，就像手動下 "docker container create" 命令一樣。
3. Docker 配置了一個可讀寫的 file system 給 container，用以作為 container 中的最上層的分層。這麼一來可讓執行中的 container 可以在它自己的 local file system 建立或是修改檔案或是目錄。
4. Docker 建立了一個 network interface 用來將 container 連結到預設的 network (雖然使用者沒有指明任何的網路選擇設定)，這包括了分派一個 IP address 給 container。預設上，container 可以透過宿主機 (host machine) 的 network connection 來連到外部網路。
5. Docker 啟動 (start) container 並執行 /bin/bash。由於 -i 和 -t 這兩個 flag 的關係，使用者可以使用鍵盤打字輸入，同時輸出會顯示在 terminal 上。
6. 當使用者輸入 "exit" 用以終止 /bin/bash 命令時，container 會停止但不會被移除。使用者可以在重新啟動或是移除 container。

## Docker 核心技術

namespace 和 cgroup (control group) 是 Linux kernel 中提供的技術，可用來作隔離 以及資源限制之用。以下將分別介紹 namespace 和 cgroup。

### Namespace

Linux kernel 為了隔離容器，提供了以下幾種 namespace 和其對應的系統呼叫 (system call) 的引數來達成目的。

|  |  |  |
| --- | --- | --- |
| Namespace | 系統呼叫的帶入引數 | 隔離內容 |
| pid  (process id) | CLONE\_NEWPID | process ID |
| net  (networking) | CLONE\_NEWNET | 網路裝置，network stack 和 port 等 |
| ipc  (inter-process communication) | CLONE\_NEWIPC | semaphore，message queue 和 shared memory |
| mnt  (mount) | CLONE\_NEWNS | 掛載點 (filesystem) |
| uts  (Unix time-sharing system) | CLONE\_NEWUTS | host name 和 domain name |
| user | CLONE\_NEWUSER | user 及 user group |

clone() 系統呼叫屬於 namespace API，可用來建立一個隔離的 namespace 的 process。clone() 中第三個引數指定要分割出 namespace 的 flag (上面表格中的「系統呼叫的帶入引數」欄)，指定後呼叫 clone()，即可分割 namespace。

比方說，若分割了 pid namespace，新建立的 pid namespace 裡，process 的 PID 就會從 1 開始編號。新的 PID 1 process 所 fork() 出來的 process 也會被關在這個新的 pid namespace 之內，與其他 pid namespace 隔離。

同樣地，pid namespace 以外的 namespace 也可以進行分割，只要 clone() 第三個引數記得設定想要分割的 namespace 對應的 flag 即可。

### Control Group (cgroup)

cgroup 是將任意 processes 包成群組一併管理的 Linux kernel 功能。cgroup 本身是提供 processes 群組化功能介面的架構，之後再利用這個架構實作 I/O 與記憶體配置控制等具體的資源管理功能。這些具體的資源管理功能稱為 cgroup subsystem 或 controller。

cgroup subsystem 包含控制記憶體配置的 memory controller、控制 process scheduling 的 CPU controller 等等。執行中的 kernel 可使用的 cgroup subsystem 可透過 /proc/cgroups 進行確認。

cgroup 管理 groups 與設定各 subsystems 的使用者介面，是透過 cgroup 虛擬檔案系統 (例如 procfs 或 sysfs) 提供。使用 cgroup 時需要先掛載 cgroup 檔案系統，下面以掛載 debug 這個 subsystem 作為例子。

# mount -t cgroup -o debug cgroup /cgroup

掛載後在掛載點下會見到以下檔案，這些是 cgroup 提供的特殊檔案。

# ls /cgroup

cgroup.event\_control debug.current\_css\_set debug.taskcount

cgroup.procs debug.current\_css\_set\_cg\_links notify\_on\_release

debug.cgroup\_css\_links debug.current\_css\_set\_refcount release\_agent

debug.cgroup\_refcount debug.releasable tasks

這些檔案分為唯讀以及可讀寫兩種類型，唯讀的檔案向使用者提供資訊，而可讀寫的檔案在寫入新值時會改變 cgroup 與 cgroup subsystem 的設定。tasks 是這些檔案中最重要的，裡面紀錄著屬於 /cgroup 這個群組的 threads 的 thread IDs (TIDs)。

cgroup 最基本的用法，也就是建立、刪除群組，以及把 thread 加入群組的方法。實際使用 cgroup 的時候，是在把 thread 加入群組之後，對群組內的特殊檔案寫入設定值，以控制系統的行為。

# 參考資料

書籍

* 《虛擬化技術原理與實現》，廣小明，胡杰，陳龍，郭京等著，2012
* 《Linux Kernel Hacks：改善效能、提昇開發效率及節能的技巧與工具》，池田宗廣、大岩尚宏等著，2014
* 《Docker 技術入門與實踐》，第 2 版，楊保華，戴王劍，曹亞倉著，2017

論文與文章

* [Susanta Nanda, Tzi-cker Chiueh, “A Survey on Virtualization Technologies”, 2005](http://www.computing.dcu.ie/~ray/teaching/CA485/notes/survey_virtualization_technologies.pdf)
* [容器虛擬化技術發展，陳詠翰，工研院資通所第 167 期 2016 年 9 月號](https://ictjournal.itri.org.tw/WebTools/FilesDownload.ashx?Siteid=654246032665636316&MmmID=654304432122064271&fd=Messagess_NFiles&RD=2&Pname=%E8%B6%A8%E5%8B%A2_%E9%99%B3%E8%A9%A0%E7%BF%B0.pdf&MSID=1)

網頁

* 維基百科
  + [Virtualization](https://en.wikipedia.org/wiki/Virtualization)
  + [Hardware virtualization](https://en.wikipedia.org/wiki/Hardware_virtualization)
  + [Operating-system-level virtualization](https://en.wikipedia.org/wiki/Operating-system-level_virtualization)
* [輕量虛擬化改寫 IT 歷史，Docker 容器技術細說從頭](http://www.netadmin.com.tw/article_content.aspx?sn=1707030003)
* [Documentation of Qubes OS](https://www.qubes-os.org/doc/)
  + [FAQ](https://www.qubes-os.org/faq/)
  + [Glossary of Qubes Terminology](https://www.qubes-os.org/doc/glossary/)
  + [Architecture Spec v0.3](https://www.qubes-os.org/attachment/wiki/QubesArchitecture/arch-spec-0.3.pdf)
  + [Introducing the Next Generation Qubes Core Stack](https://www.qubes-os.org/news/2017/10/03/core3/)
* [Playing with Qubes Networking for Fun and Profit](http://theinvisiblethings.blogspot.com/2011/09/playing-with-qubes-networking-for-fun.html)
* [Xen Project Software Overview](https://wiki.xenproject.org/wiki/Xen_Project_Software_Overview)
* Docker Documentation
  + [Get started with Docker](https://docs.docker.com/get-started/)
* [Docker 背後的內核知識─ Namespace 資源隔離](http://www.sel.zju.edu.cn/?p=556)
* [Docker 背後的內核知識─ cgroups 資源限制](http://www.sel.zju.edu.cn/?p=573)