

# 面向 AI 計算的輕量級 GPU 雲端計算平台實作

<sup>1</sup>盧宇程、<sup>2</sup>林至偉、<sup>3</sup>林子敬、<sup>4</sup>李家安、<sup>5</sup>黃伯靜、<sup>6</sup>盧鴻復、<sup>7</sup>陳政宇

<sup>167</sup>國家高速網路與計算中心、<sup>2</sup>國立臺灣科技大學、<sup>3</sup>私立東海大學、<sup>45</sup>國立交通大學

<sup>1</sup>a12125577@gmail.com, <sup>2</sup>z0794658132@gmail.com, <sup>3</sup>ginginging0000@gmail.com,

<sup>4</sup>calee@cs.nctu.edu.tw, <sup>5</sup>humble.iim06g@nctu.edu.tw, {<sup>6</sup>1203065, <sup>7</sup>1603014}@narlabs.org.tw

## 摘要

受惠於 GPU 可大量平行之計算特性，以 GPU 進行 Machine Learning、Deep Learning 或 AI 的訓練計算加速，已是目前的主流方法。然而受限於 GPU 此類裝置於電腦系統的獨佔特性、以及面對各式 AI 計算軟體堆疊環境之複雜度，如何將一具有多 GPU 伺服器系統以多人共享、快速部署、簡易使用之原則提供給組織或團隊共用，為本文主要欲解決之問題。我們採用現有受歡迎之輕量化容器虛擬化技術，搭配前端使用者入口，後端 API server、資料庫、帳號認證、儲存服務等子系統，並考慮基本的資安要求，實作一個可提供 GPU 容器 AI 計算服務之輕量化雲端平台。因使用開放原始碼軟體設計與實作，可以做為有意搭建 GPU 私有雲、並提供 AI 訓練計算的個人或團體之參考。

**關鍵詞：**GPU, 雲端運算, Docker, 高效能運算, 人工智慧, 容器化

## Abstract

Benefit from parallelized and large scale GPU computing approach, modern algorithms, like machine learning, deep learning and artificial intelligent related tasks, can be accelerated by GPGPU (General-purpose computing on graphics processing units), and this has been a state-of-art approach. However, due to limitation of using GPGPU, for example the exclusive hardware binding and domain dependent software development stack, it is very difficult to share computing resources with others. In this study, we try to build a light-weight GPU resource-governing framework, which can share sources for multiple users and groups, fast deployment, easily use as well as management intuitively. Within this framework, users can use different kind of GPU-enhanced container images to facilitate their development, and make sure all data stay in secure (comparing to existing container management tools which lack of data protection mechanism). All necessary functions for A.I. computing service are provided in this framework, including user login portal, application interface server, account services, storage service design and monitoring-management portal. Proposed framework would be a reference built for whom intent to construct an on premise site.

**Keywords:** GPU, cloud computing, Docker, High-Performance Computing, Artificial Intelligence, Container

## 1. 前言

隨著人工智慧、自動車、虛擬實境等的興起，特別是最近幾年人工智慧深度學習在各領域的應用，GPU 在人工智慧應用計算中逐漸嶄露頭角，GPU 本身設計為適合以分散式運算來加速大量單一的計算工作，在許多深度學習、機器學習等人工智慧應用是需要大量平行運算的能力來訓練模型的前提下，以 GPU 作為基礎開發環境需求日益增加，GPU 正是 AI 人工智慧發展的加速引擎。開發的軟體堆疊架構也日趨複雜且高度相依。新型態的大量平行計算硬體(如 GPU)，使用上具獨佔性，使得虛擬化及雲端化此類型計算資源深具挑戰。自2017年，科技部推動人工智慧相關政策開始，產、官、學、研各界對機器學習、深度學習與人工智慧等相關應用需求大幅提高，然而計算資源治理方式確一直無法跟上產業需求腳步。

從技術面來看，為了能提供使用者能多人共享、快速部署、簡易使用之原則，虛擬化軟體及計算資源治理工具的發展也漸興火熱。從早期的 VMware、OpenStack 虛擬機服務架構，到新型態 Docker、Kubernetes 等容器化管理平台，無一不標榜能快速佈建及搭載 GPGPU 硬體環境。在虛擬機技術方面，由於透過 hypervisor 承載的 Guest 作業系統太浪費計算資源，且運行啟動及釋放資源時間耗時，讓使用者快速開發的使用經驗大降降低，從而目前雲端服務提供商(如 AWS, Azure 等)則傾向以容器化提供計算資源，更進一步稱此類型服務為無主機型(Server-less)服務。容器化的計算資源提供方式雖然為現下主流，但在資源的安全控管上確是十分的缺乏，包括執行緒(process)、檔案系統(filesystem)、設備區隔(device segmentation)、行程間通訊(IPC, Inter-Process Communication)、網路(network)及資源的限制等管理原則[11]。另一方面容器化管理平台工具，雖然已有平台治理工具 Mesos 及 Kubernetes 等，但此類型的工具各別有不同的問題，但一般都有共享檔案系統的存取權限(UID & GID)上問題、主機根目錄(Root Filesystem)的容量問題、使用者軟體架構複雜及安裝管理上需要專業知識。

有鑑於上述容器資源治理軟體缺點，本文提出輕量級 GPU 雲端計算容器的治理架構並進行實作。提出的架構中，搭配前端使用者入口，後端 API server、資料庫、帳號認證、儲存服務等子系統，並考慮基本資安要求。在實作上，我們除設計一個可提供 GPU 容器 AI 計算服務之輕量化雲端

平台之外，並提供原始碼於 Github 中<sup>1</sup>做為有意搭建 GPU 私有雲、並提供 AI 訓練計算的個人或團體之建置使用或參考架構。

## 2. 架構設計

從使用者角度來說，將計算資源交付給使用者的流程決定了使用者的觀感。因此設計上，我們採用使用者流程導向設計本平台架構。

### 2.1 使用者流程

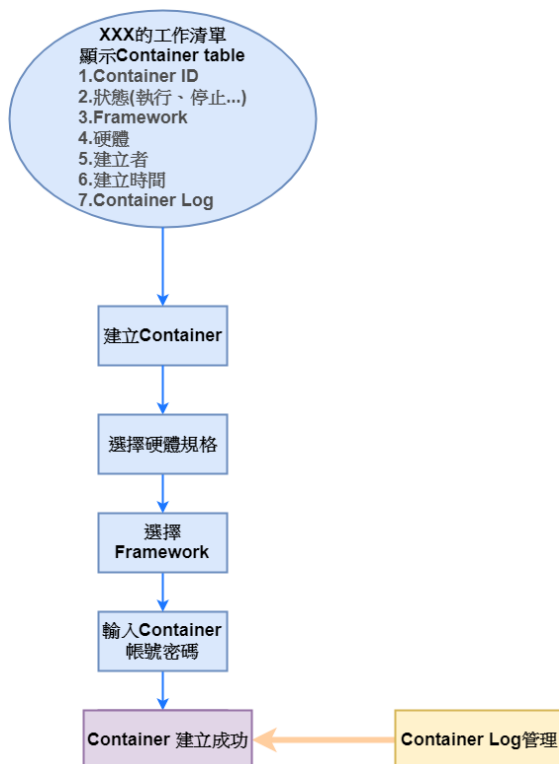


圖 1. 使用者流程概念圖

1. 登入平台 — 本平台為一個大規模共享的高速運算環境，提供 AI 人工智慧等機器學習、深度學習與大數據分析所需之設備。
2. 建立 Container — 透過 Docker RESTful API 建立容器，並將 Framework 建立於容器中
3. 選擇硬體規格 — 經由硬體配置管理讓使用者可自行選取所需之硬體規格，包含 CPU、GPU、RAM 等。
4. 選擇機器學習深度學習之 Framework — 將 Framework 包在 Docker image 裡面 - 平台提供多種 Framework：Digits、Caffe、Pytorch、Tensorflow、Mxnet 等等供使用者選擇。
5. 輸入容器帳號密碼 — 利用容器帳號密碼來保護使用者資料安全。
6. 建立成功 — 建立以後，使用 Container Log 去監控使用者使用容器的情形，例如：CPU、GPU 用量等等 - 未來將以 Log 資料為基準，運用大數據分析去建立使用者使用習慣模型，以供後續系統管理、資源管理做為參考。

### 2.2 系統架構

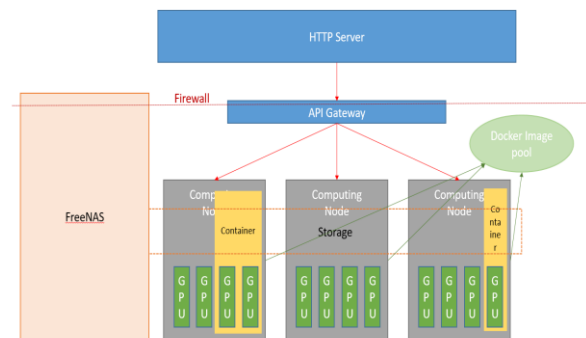


圖 2. 系統架構圖

在上圖2的系統架構中，我們將系統分為三部份實作，將 Computing module、Storage module 與使用者介面切開，統一由 API Gateway 負責資源監控與調度，以達到安全性、可擴增、使用者體驗最佳化，分述如下：

#### (一) 使用者介面

使用者介面部分，使用 PHP 與 MySQL 串接儲存前端的各類資料與分配，因屬於小型網站，故不需要前後端分離的前端框架。

在使用者資料部分，設計使用者 ID 作為唯一識別，將使用者 ID 對應到系統的 UID，方便儲存空間 mounting 與權限控管，並將登入之帳號密碼建一份於 NAS 儲存空間。使用者需要任何的資料儲存時，可以直接用註冊時申請的帳號密碼，將資料上傳到儲存空間。

#### (二) 計算節點

計算節點為各個實體運算資源。在計算節點收到 API Gateway 的指令後(多為 Docker API)，做出相對應動作並回覆 API Gateway，而 HTTP Server 再根據 API Gateway 收到的資訊，進而回傳至 Client 端。

計算節點也同時負責 Container 的維護、GPU/CPU / memory 資源監控分配、使用者權限管理、使用者資源調配、容器映像檔 (Container Image) 更新等工作，利用 Crontab 做容器映像檔的例行更新。

#### (三) 儲存空間

使用 FreeNAS 作為儲存空間，以 NFS protocol mount 空間至 Computing Node，同時提供 FTP protocol 與 samba protocol 提供使用者上傳資料。在使用者註冊帳號後，就會在 FreeNAS 建立出相對應的帳號提供使用者儲存資料，同時會針對使用者的付費方案而做出 Quota 的容量限制。

在資料掛載進 Computing Node 後，系統透過 docker volume 指令，將其掛載進容器的儲存空間資料夾，並使存放位置指向該位置，讓使用者不需額外進行繁瑣動作，便可直接存取資料。

<sup>1</sup> 本項研究內容之原始碼提供於 <https://github.com/TW-NCHC/InternProject2018>。

#### (四) API Gateway

API Gateway 負責做出系統監控、API 指令呼叫、與 Web Server 回覆等動作。

### 3. 系統實作與實例

#### 3.1 Larevel (PHP)

在多人使用的系統下，每位使用者能看見所有已被建立容器，但只有自己建立的容器才可透過刪除按鈕進行刪除。透過在建立容器時，在資料庫欄位寫入該使用者的 UID (使用者識別碼，User ID)，讓此容器綁定使用者，要進行刪除時，只有資料庫中 UID 欄位與當下登入使用者一致的容器才可執行刪除作業。

為達到上述系統架構實作，需使用具有身分驗證的帳號密碼系統，同時也要具備 MVC (Model-View-Controller) 架構。在實作中，為了達到高可擴充性及系統穩定性，及確保所設計的系統能運行在高效率運作的環境，我們參考 YU[1] 及 Olanrewaju 等人[2]建議使用 Larevel 框架搭配 PHP，以便快速地呈現使用者帳號控管系統。

#### 3.2 Database (MySQL):

在實現登入登出系統中，資料庫管理是不可或缺的部分。為了滿足開發多人多工的開發環境，我們選用 UNIX/ Linux 為基礎的作業系統。而平台中所需的資料庫管理平台則參考顏意珍[3]所建議使用的免費、開源 MySQL 資料庫，由於 MySQL 可架構在 Unix/Linux 為基礎的作業系統上，故可使用 PAM\_MySQL[4]模組存放使用者資訊，讓系統在管理上更加便利。

#### 3.3 NAT (Network Address Translate)

為了能將計算資源透過網路傳遞給使用者，在實作架構中我們以一固定 IP 透過 NAT 及埠轉譯 (port forwarding) 方式，將分配外部連線埠號對應到虛擬機中容器所指派的服務埠 (Port) 上，如下圖所示：

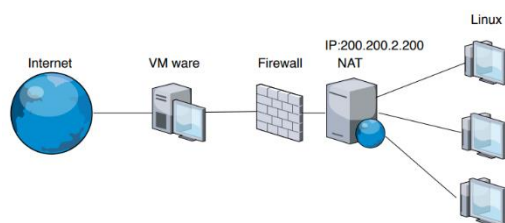


圖 3. NAT 概念架構圖

在上圖概念架構圖中，我們使用 VMware 所提供的防火牆及 NAT 管理軟體，設定服務端節點主機 (IP: 200.200.2.200) 上的埠號，並分別對應到內部虛擬網路中 Linux 計算節點機群上運行容器裡的服務埠號 (Port: 80)。此一管理機制在提出的框架中，有對應的設定功能。

#### 3.4 Docker (容器管理軟體)及 API Gateway

為了達到輕量級計算資源管理，且讓使用者所需的資源能快速啟動及耗資源低的治理目標，以達成可分享、快速之計算資源，我們參考李信賢[5]建議，以 Nvidia 官方支援的 Docker 技術作為容器管理元件，計算 Docker 服務與虛擬機技術，差別在於軟體堆疊上的複雜度，參考 Bemstein[10]提供的下圖：

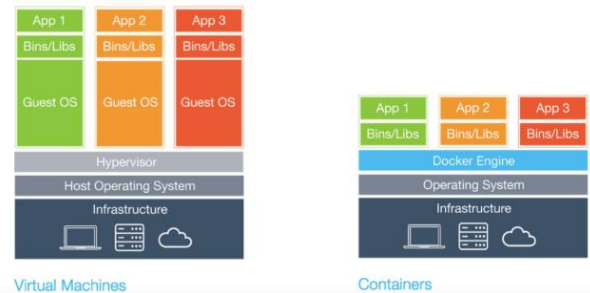


圖 4. 虛擬機與容器技術軟體堆疊差異

在上圖中，虛擬機技術 (上圖左方) 透過 Hypervisor 提供 Guest OS 虛擬化硬體資源環境，相對容器技術 (上圖右方) 中使用 Docker 引擎提供 Bins (編譯後二元檔)/Libs (函式庫) 來說，容器則捨棄 Guest OS 層級直接使用 Host 作業系統所提供的虛擬化工具，因此在運作上，相對於虛擬機更能提高運作效能。此外 Docker 在 Linux 核心中的資源分離機制，如控制群組 (Cgroups)，以及 Linux 核心命名空間 (namespace)，來建立獨立的容器，能讓不同使用者各自擁有獨立的虛擬化設施。另一方面，Docker 在管理容器上提供主流應用服務支援的 RESTful API，能針對容器釋放資源及回收更有彈性。

為能將 Docker 所提供的 Restful API 包括成為本系統服務之特色，我們參考 Chen 等人建議[6]，重新將 Docker RESTful API 透過前述 Laravel 平台，將其整合成 API 服務工具，即可透過 API 服務工具對容器進行快速建立、啟動、刪除等處理。

#### 3.5 Reversed Proxy:

為能提高 Docker API 之資訊安全等級，防止容器遭未經授權的外部所改動，我們以 Reversed Proxy 阻擋外部資源連線。其中，Reversed Proxy 可將特定的 Web Server 獨立區分，再透過 Reverse Proxy 連接實際的 Web Server，達到保護實際 Web Server 的功能[7]。概念以下圖表示：

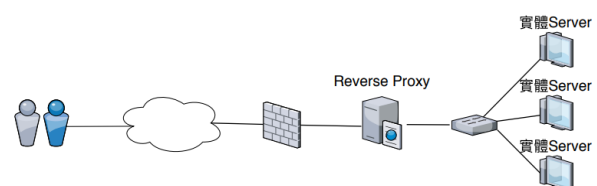


圖 5. 資安防護下的容器管理架構

在上圖中，實體 Server 為虛擬網路中的各計算節點，且已將 Docker API 開啟可供計算容器佈署使用。本平台架構中的 Reversed Proxy 為統一管理各計算節點之管理節點，依不同使用者需求佈



署容器至實體 Server 中。使用者則透過網際網路存取本平台服務，經授權後可將容器佈署至計算節點中。

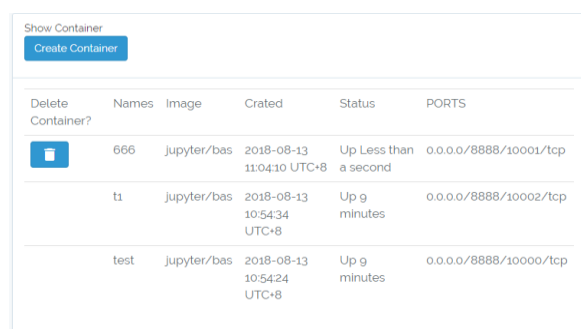
為防止有心人士透過 Docker RESTful API 惡意改寫本平台管理之容器，故需架設 Reverse Proxy 進行內部保護。在實作上，本平台為能滿足以最少計算資源維持在高效能的服務架構，參考 Reese[8]建議以 Nginx 進行實作，並且提供相關的管理介面。

### 3.6 FreeNAS (儲存服務)

由於本系統為多人使用架構，故在建立使用者帳號時，需同時建立該使用者的目錄，並直接掛載工作目錄至容器，除了讓使用者無需準備運算時所需儲存空間之外，亦能確保儲存資料可自動建立備援資料。本平台參考[9][10]所建議使用之 FreeNAS 進行建置網路掛載儲存空間 (NAS, Network Attached Storage)，並使用 NFS (Network File System)自動在容器開啟前進行掛載。除此之外，FreeNAS 針對所有檔案設定存取控制清單 (Access Control List) 權限，可與前述之 PAM\_MySQL 整合，確保使用者資料存取的安全性。

### 3.7 使用者介面及開發介面

綜合上述各項系統功能，使用者可透過網際網路登入本平台，並依 AI 人工智慧演算法實際需求配置所需要的計算資源 (CPU、GPU、Memory、儲存空間大小、網路等等)，透過平台建置的 API Gateway 進行權限控管與認證後，取得容器計算資源。平台中，容器管理依使用者權限進行監控，如下圖



Delete Container?	Names	Image	Crated	Status	PORTS
	666	jupyter/bas	2018-08-13 11:04:10 UTC+8	Up Less than a second	0.0.0.0/8888/10001/tcp
	ti	jupyter/bas	2018-08-13 10:54:34 UTC+8	Up 9 minutes	0.0.0.0/8888/10002/tcp
	test	jupyter/bas	2018-08-13 10:54:24 UTC+8	Up 9 minutes	0.0.0.0/8888/10000/tcp

圖 6. 容器開啟及管理介面

在上圖中，使用者可點擊左上角「Create Container」按鈕進行建立容器。底層 API 會依使用者所需之計算資源及容器映像檔(Container Image)，自動部署到計算節點中，並將所需要網路服務埠 (Port) 資訊提供給使用者存取。

為了讓使用者更加便利，以 Jupyter Notebook 為例，平台先將 Jupyter Notebook 預載入容器映像檔中，供使用者在瀏覽器介面中直接進行程式開發。由於 Jupyter Notebook 整合多媒體資訊呈現的開發工具，不僅能以圖表方式呈現科學計算後的結果，讓使用者更快速的理解，內建的終端機功

能，也讓使用者可對作業系統層直接下達指令。由於 Jupyter Notebook 的存取防護以預設的亂碼 (Token) 做為金鑰，本平台將其改為使用者自定義密碼，以確保存取金鑰不會遺失。

## 4. 結論與討論

隨著多元且即時人工智慧應用漸受矚目，尤其以深度學習的蓬勃發展，開發者對於基於 GPU 為主的計算資源日益要求，但 GPU 等計算硬體及軟體堆疊的治理也因容器化架構讓服務提供更具挑戰。為了能滿足不同使用者需求及易於架構、建置與治理的目標，本研究提出輕量級的面向 AI 計算的雲端計算平台，透過 Nvidia 官方支援的 Docker 工具又架構中帳號、資安、監控、儲存及網路等等工具的功能設計，將使用者所需的計算資源透過網際網路提供給使用者以 Jupyter Notebook 存取。本研究的設計理念及基本功能架構亦說明在上述內容中，且所有系統元件皆基於開放原始碼軟體設計與實作，可以做為有意搭建 GPU 私有雲、並提供 AI 訓練計算的個人或團體之重要參考。

本平台架構雖然從使用者角度切入設計，且仔細避免各系統層面可能遇到的概念架構盲點，然而在功能設計上仍有不足，包括仍無法加入支援人工智慧計算加速的儲存服務(如高速平行檔案系統)及跨節點的大規模訓練環境。此外提供使用者運行自建容器映像檔亦是客製化計算服務的重點，而在目前架構中仍尚未設計。然而上述限制都已列入未來開發項目，相關技術規格會於設計完成後說明。

## 參考文獻

- [1] H. R. YU, "Design and implementation of web based on Laravel framework," Atlantis Press, Advances in Computer Science Research, pp. 301–304, Jan. 2015.
- [2] R. F. Olanrewaju, T. Islam, N. Ali, "An empirical study of the evolution of php mvc framework," In: Advanced Computer and Communication Engineering Technology, pp. 399–410, 2015.
- [3] 顏意珍, "開放式軟體 MySQL 及其資料處理效能分析," 國立交通大學機械工程系, 未發表碩士論文, 民國92年。
- [4] PAM – MySQL PAM module backed by MySQL 2015 URL <http://pam-mysql.sourceforge.net/>, last visited 2018-0815.
- [5] 李信賢, "以 Docker 容器技術進行分散式就地計算架構之設計與實作," 國立高雄第一科技大學資訊管理系, 未發表碩士論文, 民國106年。
- [6] X. J. Chen, Z. Ji, Y. Fan, and Y. Zhan "Restful API Architecture Based on Laravel Framework," In: Journal of Physics: Conference Series. IOP Publishing, pp. 1–6, 2017.
- [7] 蘇建郡, 謝仁瑋, "利用 Reverse Proxy 建構安全性單一入口平台," Conference on Computer Science and Information Engineering Applications, pp. 123–127, 2009.
- [8] W. Reese, "Nginx: the high-performance web server and reverse proxy," Linux Journal, 2008.173: 2, 2008.
- [9] 蘇介民, "Ceph 虛擬儲存叢集," 國立中興大學應用數學系, 未發表碩士論文, 民國104年。
- [10] D. Bernstein, "Containers and cloud: From lxc to docker to kubernetes," IEEE Cloud Computing, (3), pp. 81–84, 2014.
- [11] T. Bui, "Analysis of Docker Security," CoRR, vol. abs/1501.02967, Available: <http://arxiv.org/abs/1501.02967>, 2015.