# 可延展式與容錯性 WebAPI 安全管制機制設計

洪胤勛　吳坤熹
國立暨南國際大學 資訊工程學系[1]
{s108321019,solomon}@ncnu.edu.tw

摘要

本研究將探討傳統的 Web Server 與 WebAPI Gateway 之間的差異，並對這兩者的運作模式做效能上的分析。WebAPI Gateway 能減少公有 IP 位址暴露的數量，但同也會有單點故障的風險。因此本研究會使用虛擬化 container 的技術，讓 API Gateway 跑在多個 container 上，並使用 Kubernetes 來分擔 container 的壓力和確保 container 的正常運作。

## I. 前言

在一個分工的時代，工程師也要懂得如何分工。在軟體工程裡面，有個專有名詞叫做 API（Application Programming Interface）。它是一個溝通的介面，旨在讓工程師們專心在他們的任務，而不需要憂慮其他層面的問題。比如說一個人用電腦時，他只要知道鍵盤、滑鼠可以輸入資料，螢幕可以輸出結果，但他並不需要去知道電腦是如何運作的。如果是網頁的開發，通常會分為前端和後端，前端負責設計使用者介面，後端負責資料的儲存，前後端之間通常會透過 HTTP 或 HTTPS 來溝通，並使用 JSON 或 XML 格式來傳遞資料，此種模式稱為 WebAPI。

在使用網頁時，通常使用者（client）會需要去 WebAPI 服務端（也就是 server）提取資料。當使用者需要多筆資料，而那些資料可能是由不同的服務端所提供的，那架構就會像圖1。但這架構會造成一個大問題，每個服務端都必須要有一個公用的 IP 地址，也就表示所有人都可以直接連到伺服務器。尤其在 IoT 的佈建當中，許多的感測器是由外包廠商負責建置，再透過公開的4G 網路傳回伺服器。IoT 設備由4G 取得的是浮動的 IP 位址，因此實務上伺服端都必須擁有固定的公開 IP 位址，以讓 IoT 設備傳回資料。如果有心人士想竊取伺服器的資料，或是想讓伺服器癱瘓，這種架構極有可能會讓有心人士藉由伺服器的公開 IP 位址達成他們的目的。

於是後來有了圖2這樣的 Gateway 架構。Gateway 扮演了使用者連到服務端間的中繼站，使用者需要的資料都間接透過 Gateway 取得，服務端要給使用者的資料也都透過 Gateway 傳送。公用的 IP 地址只需要 Gateway 這台電腦擁有，真正提供服務的服務端可以只使用私有 IP 地址。這解決了剛剛所提到的問題，服務端不再需要暴露在所有人都可以直接連線的公開網路中。除了上述的好處之外，Gateway 還能讓服務端的管理者清楚掌握當前有哪些電腦正在對外提供服務，並把沒在提供服務的電腦給關機，減少了被入侵的機會。

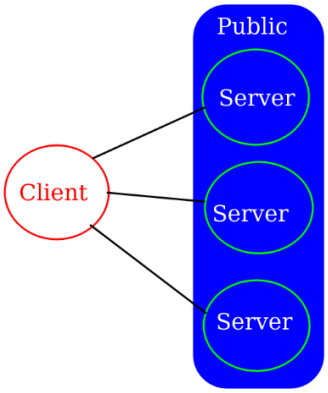


圖1. 傳統 Web Server

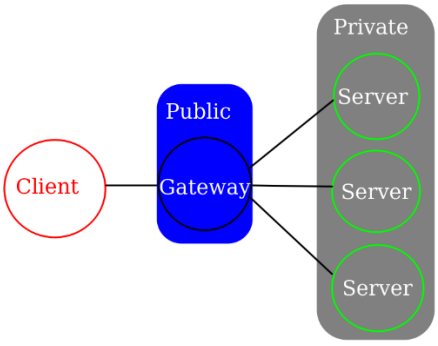


圖2. WebAPI Gateway

## II. 研究問題

雖然上述的架構解決了把服務端暴露在公用 IP 的環境，但相對地，也讓風險全部都承擔在 Gateway 上。萬一，Gateway 停止運作，那麼使用者也沒辦法正常存取服務。針對此問題，勢必要有個自動化的機制，讓停止

[1] 本研究成果感謝國立暨南國際大學與埔基醫療財團法人埔里基督教醫院產學合作之「埔暨計畫」(111-PuChi-AIR-006) 經費贊助

運作的 Gateway 能夠自動重啟。因此本研究將使用 Gateway 與 Kubernetes 搭配，讓因意外而停止運作的 Gateway 自動重啟。

網路上有許多常見的 API Gateway 開源項目，本研究將對以下三個 API Gateway：Express Gateway，API Umbrella，Kong API Gateway 進行效能上的分析與比較。

## III. 實驗所需

### A. API Gateway

1. 此實驗將比較 Express Gateway [1]，API Umbrella [2]，Kong API Gateway [3]這三個 API Gateway 的效能。
2. 在建置 API Gateway 時所使用的平台，將分別使用實體主機、VM 虛擬機、以及 Docker 的容器（container）技術。

### B. Docker

相較於虛擬機[4]（VM），Docker [5]的 container 是個類似虛擬的技術，但卻比虛擬機更為輕巧，更能夠快速地建立起來。傳統的虛擬機著重在將硬體設備給虛擬化，而 container 則著重在將作業系統給虛擬化。如圖3所示，左邊為傳統的虛擬機，右邊則為 container 的架構。我們可以看到，左邊的虛擬機可以把一台電腦的硬體資源分配給多台虛擬機，但每台虛擬機都需要有作業系統在上面執行。右邊的 container 一樣是把硬體資源分配給 container，但值得注意的是，每個 container 並不需要再架起一個作業系統，這也是為甚麼 container 比起虛擬機更為輕量，更能夠快速建立起來的原因。

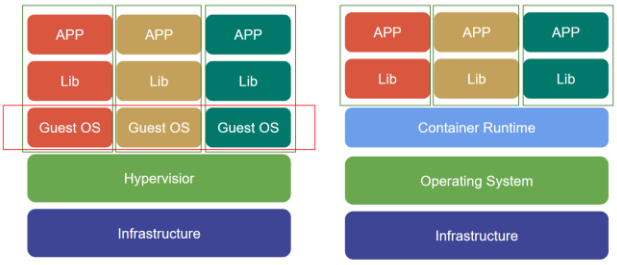


圖3. VM vs. Container

### C. Kubernetes

Kubernetes [6]簡稱為 K8s，由 Google 設計出來，現在已屬於 Cloud Native Computing Foundation。K8s 中，一個基本單位為 Pod，Pod 裡面可以有一個或多個 container，但通常只會有一個。在運行 K8s 時，有兩個主要角色，如圖4所示，為 Master 和 Worker。Master 和一個或多個 Worker 組合起來就稱為 Cluster。使用者會在 Master 端下達指令，告訴 K8s 我們要架起服務的需求共要幾個 Pod。Master 則會根據 Worker 的狀態，來決定如何把所有的 Pod 分配到各個 Worker 上。K8s 中還有一個物件稱為 Deployment，它的工作是保持 Pod 的數量；當有 Pod 若因意外故障，Deployment 則會自動啟動新的 Pod，讓 Pod 維持在一定的數量。

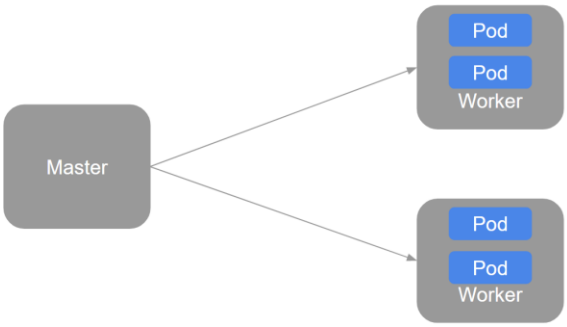


圖4. Kubernetes 架構

### D. K6

K6 [7]為壓力測試的工具，它能夠使用 CLI 或 JavaScript 編寫自己定義的測試內容，例如要有多少個虛擬使用者去做請求，還有這段測試要執行多久。

K6執行完時，會輸出請求傳送時間、等待時間、接收時間等等，之後可以選擇多種輸出方式，如圖5。預設會在終端機有摘要輸出，（平均值，最大值，最小值……），也能決定是否要把這些摘要輸出給存起來。除了上述的摘要輸出，K6也能在測試時，用 time-series 的方式把每個時間點的結果給存起來，而這裡又分為 stream 的方式（每紀錄一筆就輸出到資料庫）或是結束時再把整個結果寫入到檔案裡。

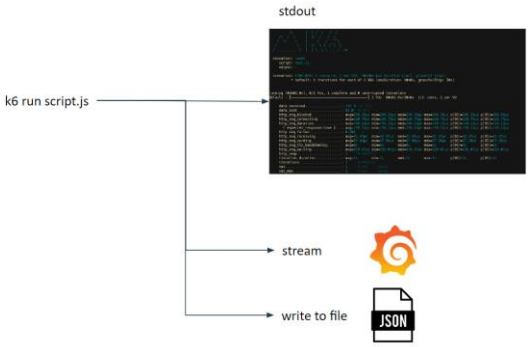


圖5. K6輸出方式

## IV. 實驗步驟

### A. 直接針對 API 的來源做壓力測試

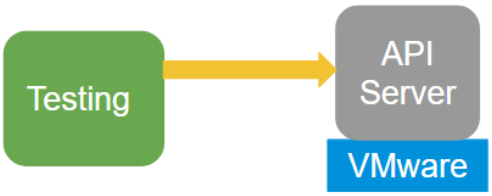


圖6. API 來源測試

*B.* 使用 API Gateway，來轉傳 API，並做壓力測試

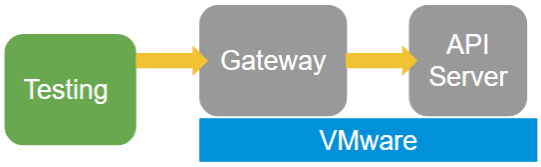


圖7. Gateway

*C.* 把 API Gateway 部署在 K8s 上

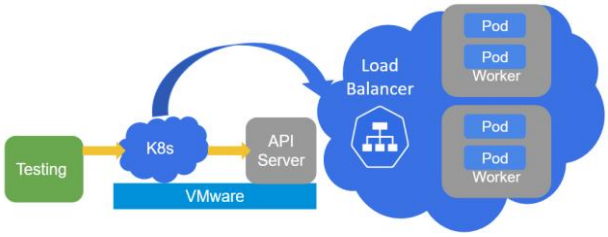


圖8. Gateway 在 K8s 上

## V. 實驗結果

底下實驗圖中，橫軸代表時間線，縱軸代表虛擬使用者的數量；藍線代表直接對 API 來源的請求時間，紅線代表透過在 VMware 上的 API Gateway 的請求時間，黃線則是代表在 K8s 的 API Gateway 的請求時間。

圖9和圖12的黃線分別是 Express Gateway 和 Kong API Gateway 在 K8s 使用一個 Pod 執行的結果；圖10和圖13的黃線分別是 Express Gateway 和 Kong API Gateway 在 K8s 使用兩個 Pod 執行，並且兩個 Pod 在相同主機的結果；圖11和圖14分別是 Express Gateway 和 Kong API Gateway 在 K8s 使用兩個 Pod 執行，並且兩個 Pod 在不同主機的結果。

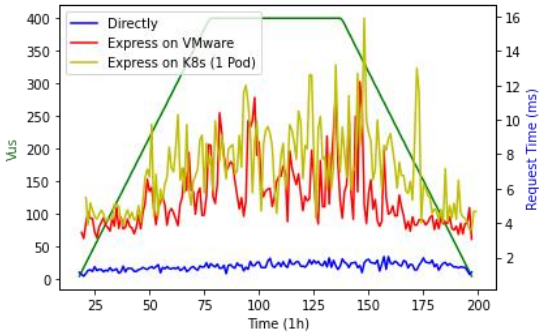


圖9. Express 比較圖

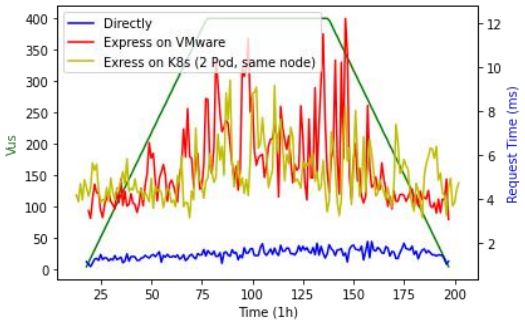


圖10. Express 比較圖

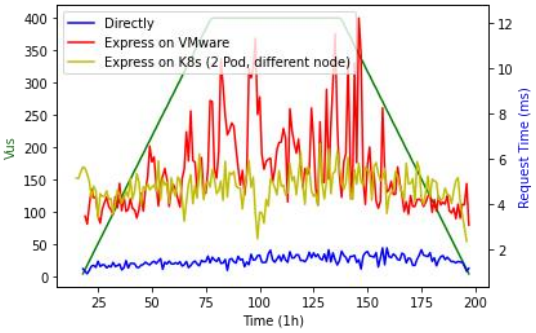


圖11. Express 比較圖

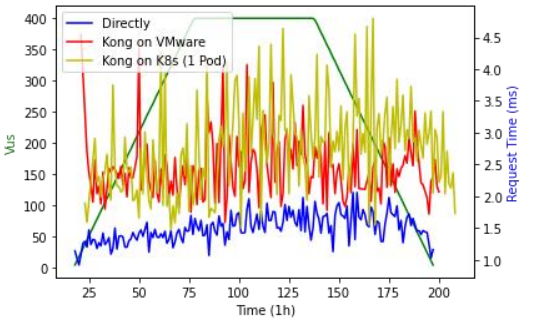


圖12. Kong 比較圖

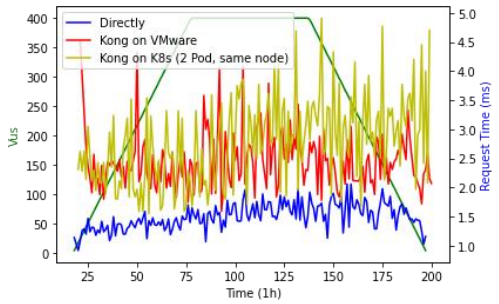


圖13. Kong 比較圖

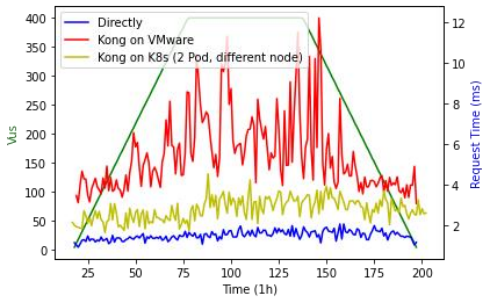


圖14. Kong 比較圖

## VI. 結論

Express Gateway 和 Kong API Gateway 在請求時間的比較，明顯是 Kong 比較好，在社群的活躍度上，Kong API Gateway 也是勝過 Express Gateway。且在設定 API Gateway 方面，Express Gateway 只能透過修改設定檔來配置，Kong API Gateway 可以透過發送 HTTP/HTTPS request 來作到 CRUD（Create，Read，Update，Delete）。且還有第三方插件，能讓 Kong API Gateway 在網頁上做設定。

在裸機上的 API Gateway 效能最好，因為所需要 network hopping 數最少，且可以直接使用電腦上的硬體資源。而 Container 則需要透過 Linux 的 cgroup [8]來分配， 且不是100%的分配，所以效能略低，但不至於太差。且 Container 的優勢就在於能快速安裝且啟動，安裝設定若希望調整，就可以重新另創一個新的容器，不用像裸機一樣需要擔心是否會搞亂環境。

在 K8s 的比較上，單個 Pod 的 Request 時間比裝在 VMware 上來的慢。當 Pod 提升到兩個的時候，當這兩個 Pod 都在相同的 Node 上，Request 時間已經與在 VMware 上差不多；當兩個 Pod 在不相同的 Node 上，可以看到 Request 時間已經相比在 VMware 上較低。

綜合以上實驗，我們可以看出，利用 Kubernetes 的自動重啟功能，可以讓 API Gateway 的運作更穩健，故障重啟的時間更加縮短。

## 參考文獻

[1] M. Bawane, I. Gawande, V. Joshi, R. Nikam, and S. A. Bachwani, "A Review on Technologies used in MERN stack," *International Journal for Research in Applied Science & Engineering Technology (IJRASET)*, 2022.

[2] A. Gámez Díaz, P. Fernández Montes, and A. Ruiz Cortés, "Towards SLA-driven API gateways," *XI JORNADAS DE CIENCIA E INGENIERÍA DE SERVICIOS*, 2015.

[3] R. Xu, W. Jin, and D. Kim, "Microservice security agent based on API gateway in edge computing," *Sensors*, vol. 19, no. 22, p. 4905, 2019.

[4] R. P. Goldberg, "Survey of virtual machine research," *Computer*, vol. 7, no. 6, pp. 34-45, 1974.

[5] C. Anderson, "Docker [software engineering]," *IEEE Software*, vol. 32, no. 3, pp. 102-c3, 2015.

[6] D. Bernstein, "Containers and cloud: From lxc to docker to kubernetes," *IEEE cloud computing*, vol. 1, no. 3, pp. 81-84, 2014.

[7] V. Seifermann, "Application performance monitoring in microservice-based systems," Bachelor, Institute of Software Technology Reliable Software Systems, University of Stuttgart, 2017.

[8] M. G. Xavier, M. V. Neves, F. D. Rossi, T. C. Ferreto, T. Lange, and C. A. De Rose, "Performance evaluation of container-based virtualization for high performance computing environments," in *2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, 2013: IEEE, pp. 233-240.