**可延展式與容錯性WebAPI安全管制機制之設計**

洪胤勛 吳坤熹

 國立暨南國際大學 資訊工程學系

{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。

1. 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，而不需要管理其他層面的問題。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Gateway圖2這架構。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用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知道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

1. 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，讓停止運作的Gateway能夠自動重啟。因此本研究將使用Gateway與Kubernetes搭配，讓停止運作的Gateway再次重啟。

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

1. 實驗所需
2. API Gateway
3. 此實驗將比較Express Gateway[1]，API Umbrella[2]，Kong API Gateway[3]這三個API Gateway的效能。
4. 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。
5. 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

1. 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若因意外故障而低於預期的數量，Deployment則會啟動新的Pod，讓Pod維持在一定的數量。



圖4. Kubernetes架構

1. 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

1. 實驗步驟
2. 針對API的來源做壓力測試



圖6. API來源測試

1. 使用API Gateway，來轉傳API，並做壓力測試



圖7. Gateway

1. 把API Gateway部署在K8s上



圖8. Gateway在K8s上

1. 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. 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

1. 結論

Express Gateway和Kong API Gateway在請求時間的比較，明顯示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，所以效能會低一點，但不至於太差。且Container的優勢就在於能快速安裝且啟動，裝壞了就重新另起一個，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。

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."

[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," 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.