**可延展式與容錯性 WebAPI 安全管制機制之設計**

* **摘要**
  + 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，它們通常會透過 http 或 https 來溝通，並使用 JSON 或 XML 來傳遞資料，此種模式稱為 WebAPI。
* **研究動機與研究問題**

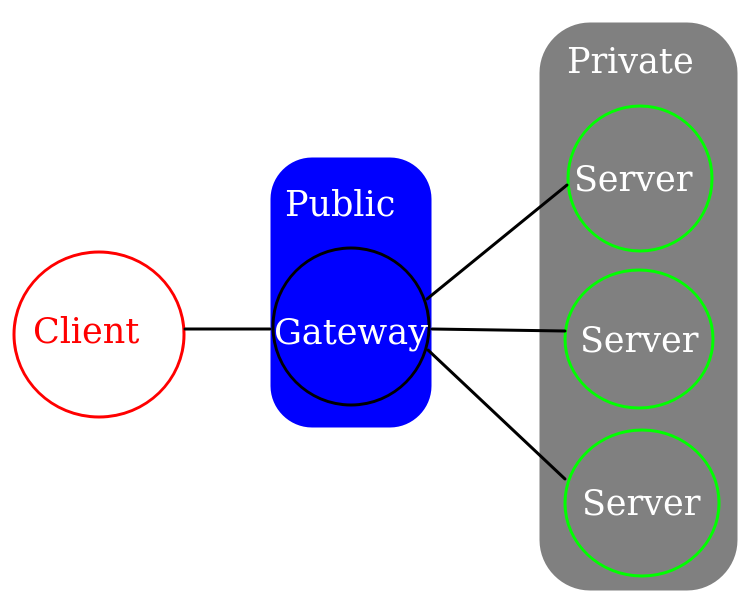
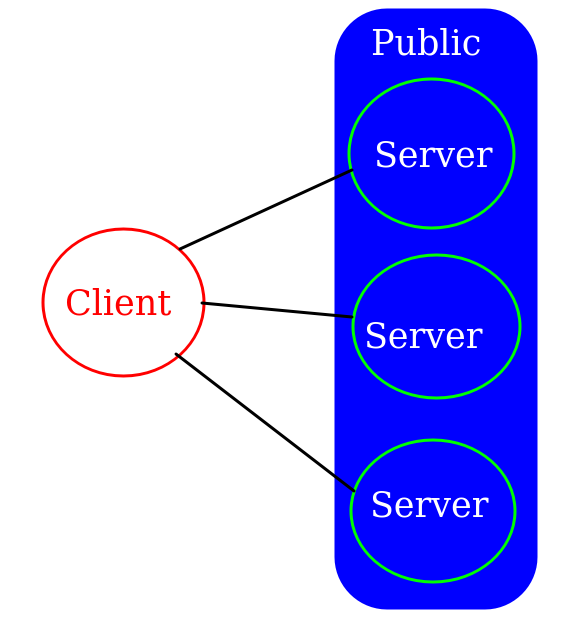
****

圖1、傳統 Web Server 圖2、WebAPI Gateway

* + 在使用網頁時，通常使用者 (client) 會需要去 WebAPI 服務端提取資料 (也就是 server 端)。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服務端。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，那這種架構極有可能會讓有心人士達成他們的目的。
  + 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，真正在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狀況。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。
  + 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。
* **實驗所需**
  + API Gateway
    - 此實驗將比較 Express Gateway [1], API Umbrella [2], Kong API Gateway [3] 這三個 API Gateway 的效能。
  + Docker

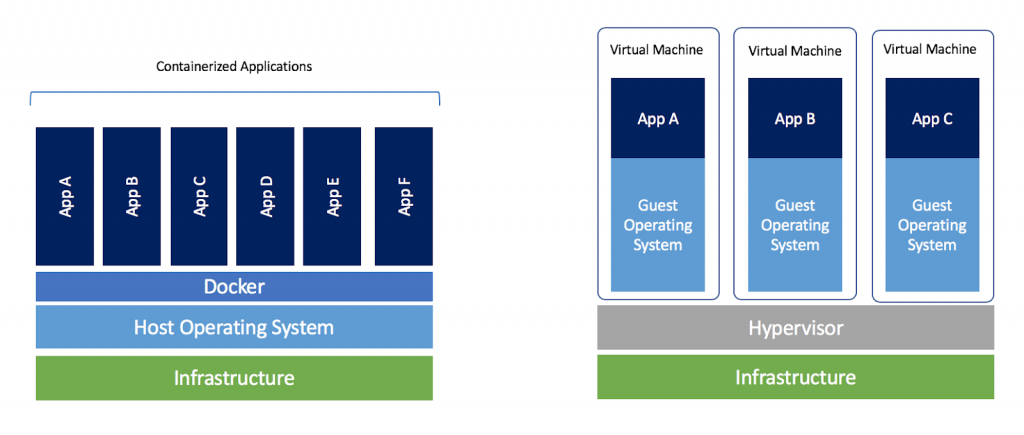


圖3、 Docker vs. VM [4]

* + - container，其原理與虛擬機（VM）類似，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 container的架構，右邊則為傳統的虛擬機。我們可以看到，右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。左邊的 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。
  + Kubernetes

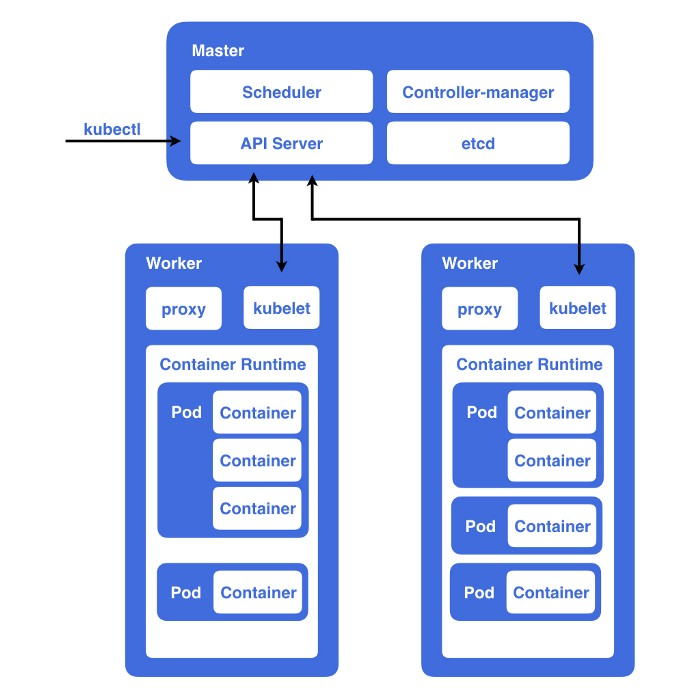


圖4、Kubernetes 架構 [5]

* + - Kubernetes 又稱為 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，有兩個主要腳色，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，來決定每個 Worker 要分配幾個 Pod。K8s 中還有一個物件為 Deployment，他的工作是保持 Pod 的數量，當 Pod 若因意外故障而少於預期的數量，Deployment 則會啟動新的 Pod，讓 Pod 維持在一定的數量。
  + k6
    - k6 為壓力測試的工具, 能夠使用 JavaScript 編寫自己定義的測試內容。
* **實驗步驟**
  + 先針對 API 的來源做壓力測試
  + 再來使用 API Gateway, 來轉傳 API, 並做壓力測試
    - 在VMware的虛擬機上直接安裝 API Gateway
    - 在 Docker 上運行單個 container 的 API Gateway
    - 在 K8s 部署 3 個 pod 在 2 個 Worker Node 上, 並透過 Service 的 LoadBalancer 來平分 request (API Umbrella 因為已經沒有在維護了, 新版的 Docker 已無法運行, 所以 K8s 的實驗不會有 API Umbrella)
    - 使用 k6 來做測試, 本實驗將針對 Load Testing, Stress Testing, Spike Testing, 並對以上三點不同的情境做測試。
* **實驗結果**

(綠色為虛擬使用者數量, 黃色為 request 所需的時間)

* + Load Testing

前 5 鐘從 0 個虛擬線性增加到 100 個虛擬使用者去 request, 接下來五分鐘維持在 100 個虛擬使用者, 最後五分鐘從 100 個虛擬使用者線性遞減到 0 個

圖6、API 來源的反應時間

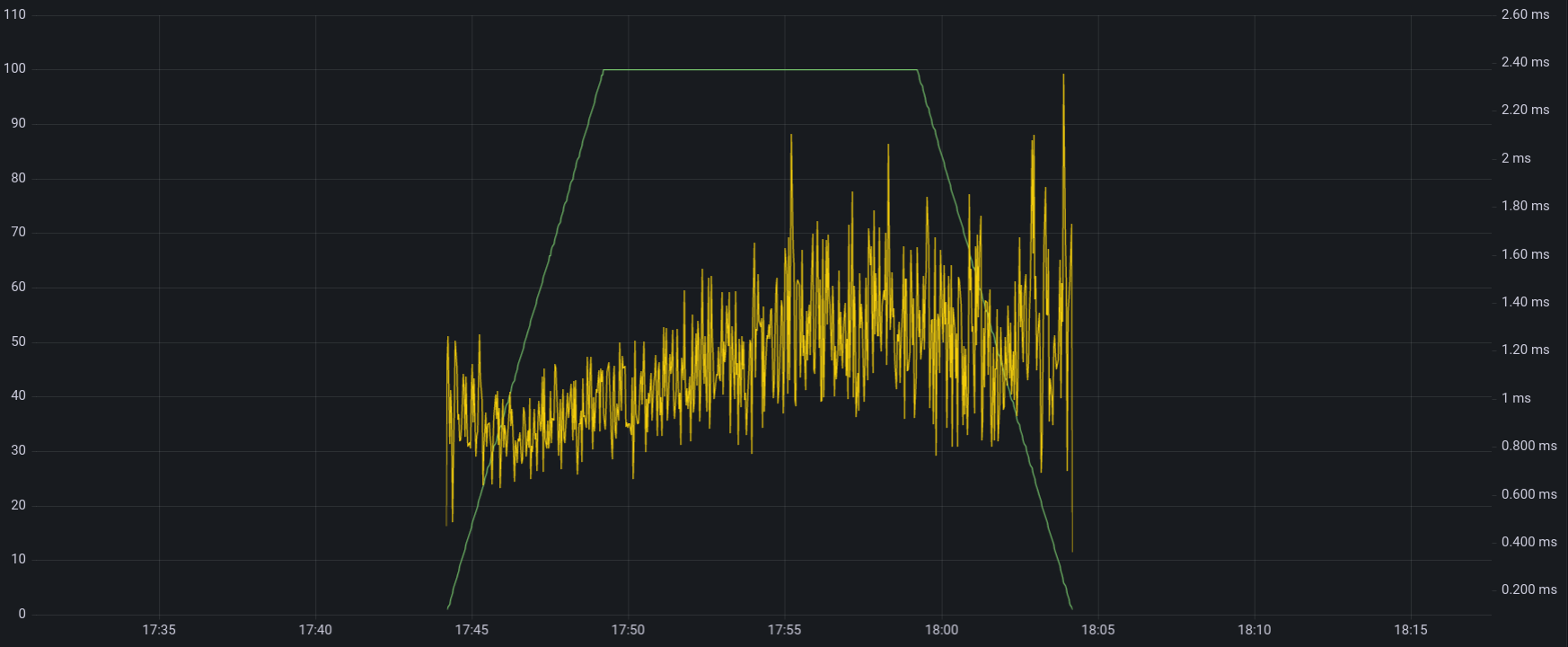


圖7-1、在 VMware 上的 Express Gateway 的反應時間

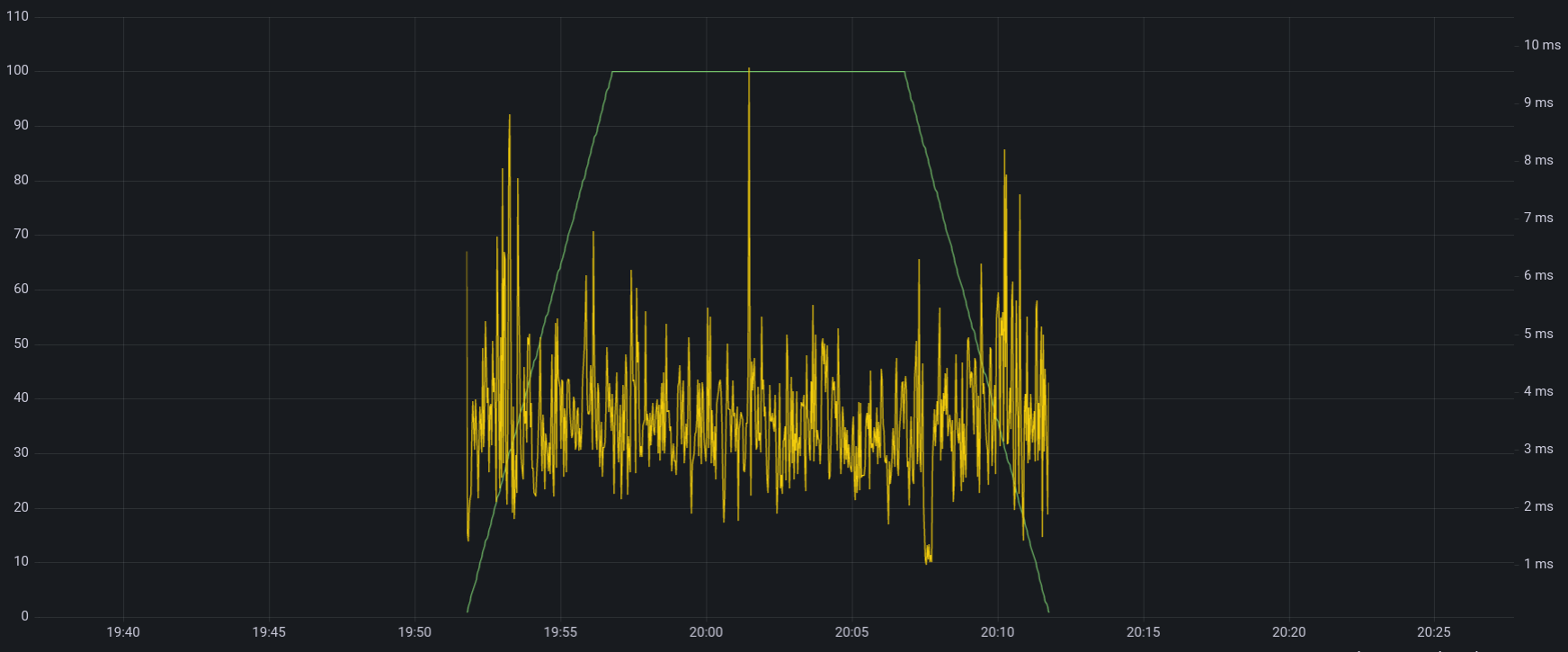


圖7-2、在 Docker 上的 Express Gateway 的反應時間

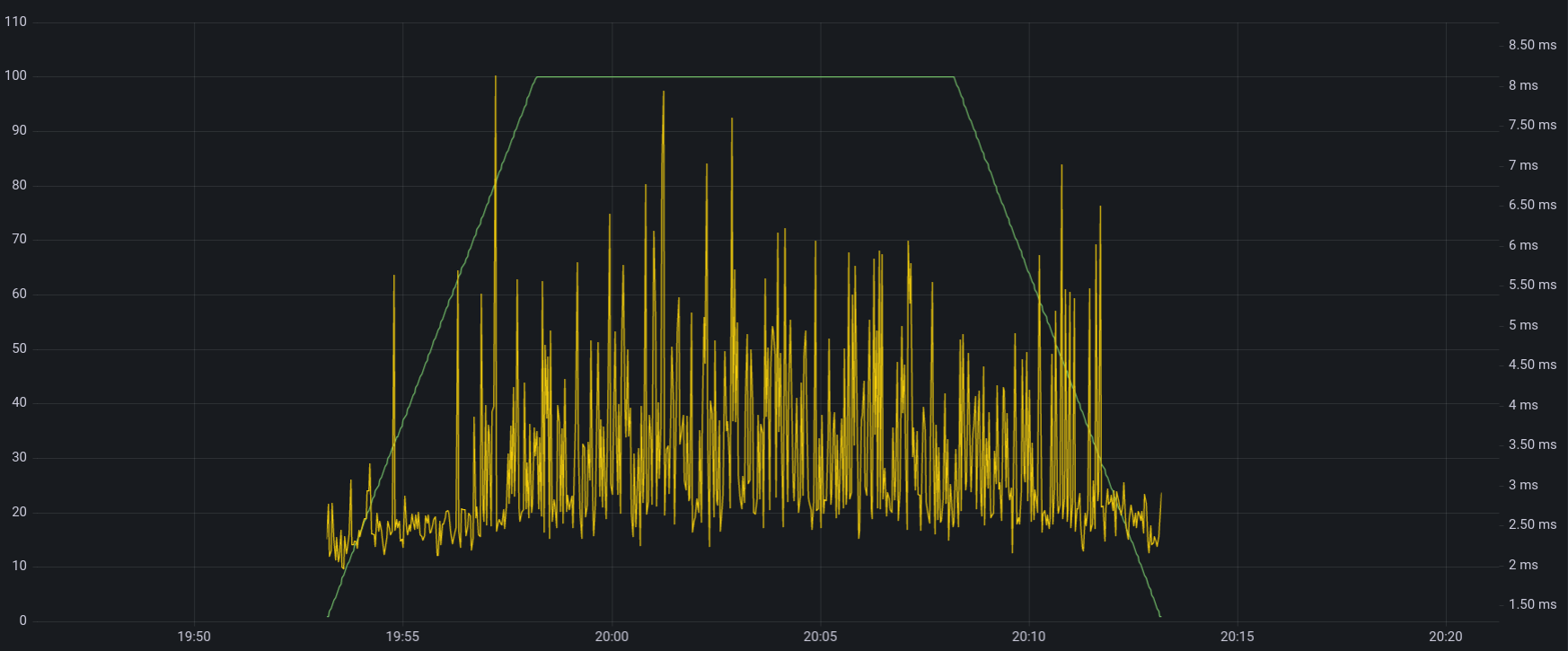


圖7-3、在 K8s 上的 Express Gateway 的反應時間



圖8-1、在 VMware 上的 API Umbrella 的反應時間



圖8-2、在 Docker 上的 API Umbrella 的反應時間

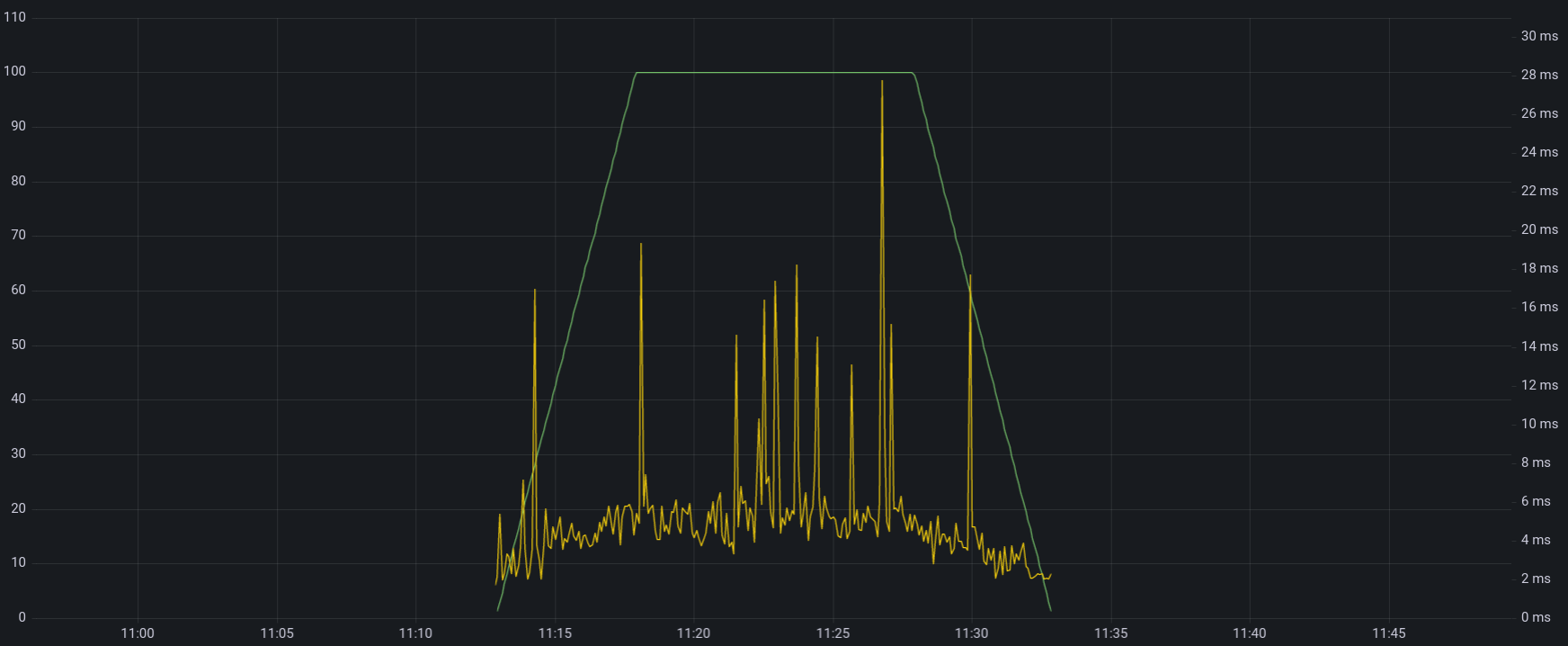


圖9-1、在 VMware 上的 Kong API Gateway 的反應時間



圖9-2、在 Docker 上的 Kong API Gateway 的反應時間

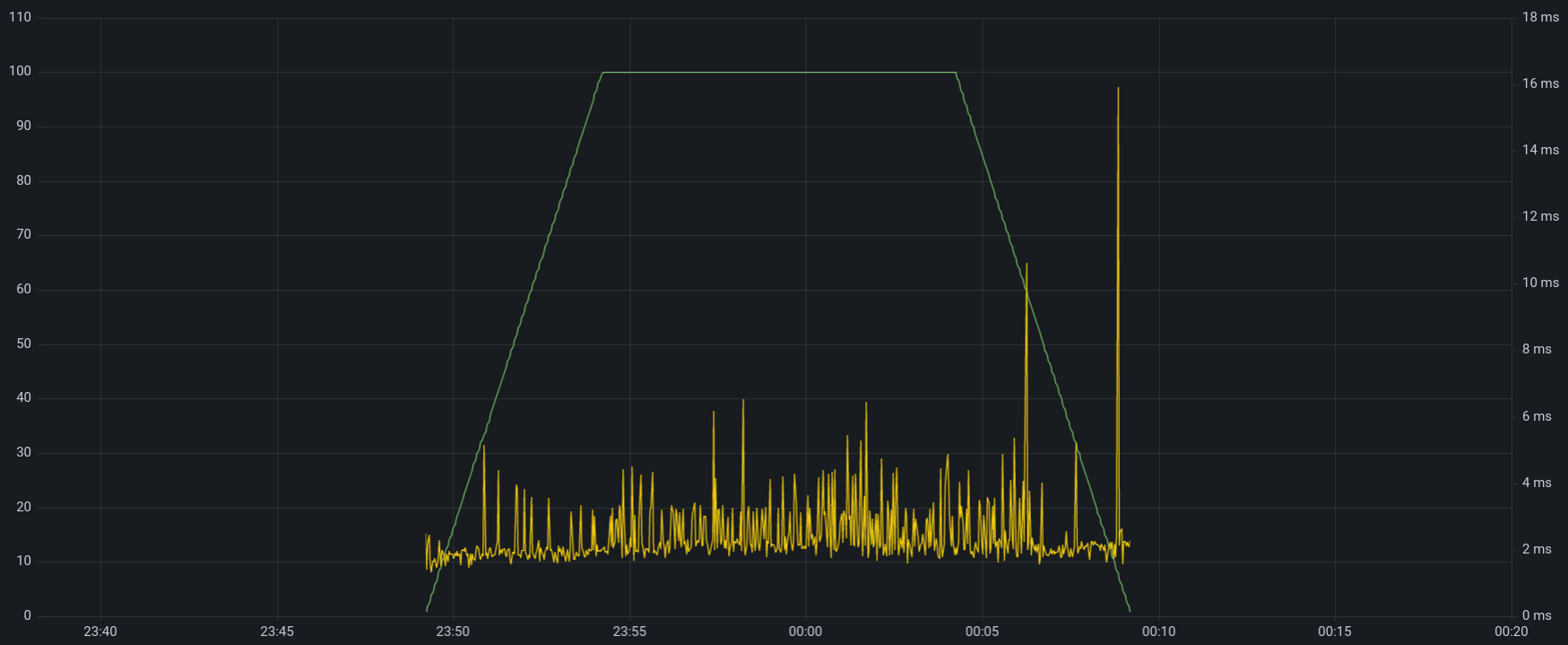


圖9-3、在 K8s 上的 Kong API Gateway 的反應



* + Stress Testing

前 2 分鐘從 0 個 virtual user 增加到 100, 再來 5 分鐘維持 100 個。再來兩分鐘從 100 增加到 200, 再來 5 分鐘維持 200 個。再來兩分鐘從 200 增加到 300, 再來 5 分鐘維持 300 個。再來兩分鐘從 300 增加到 400, 再來 5 分鐘維持 400 個。最後 10 分鐘從 400 降到 0 個。

圖10、API 來源的反應時間



圖11-1、在 VMware 上的 Express Gateway 的反應時間

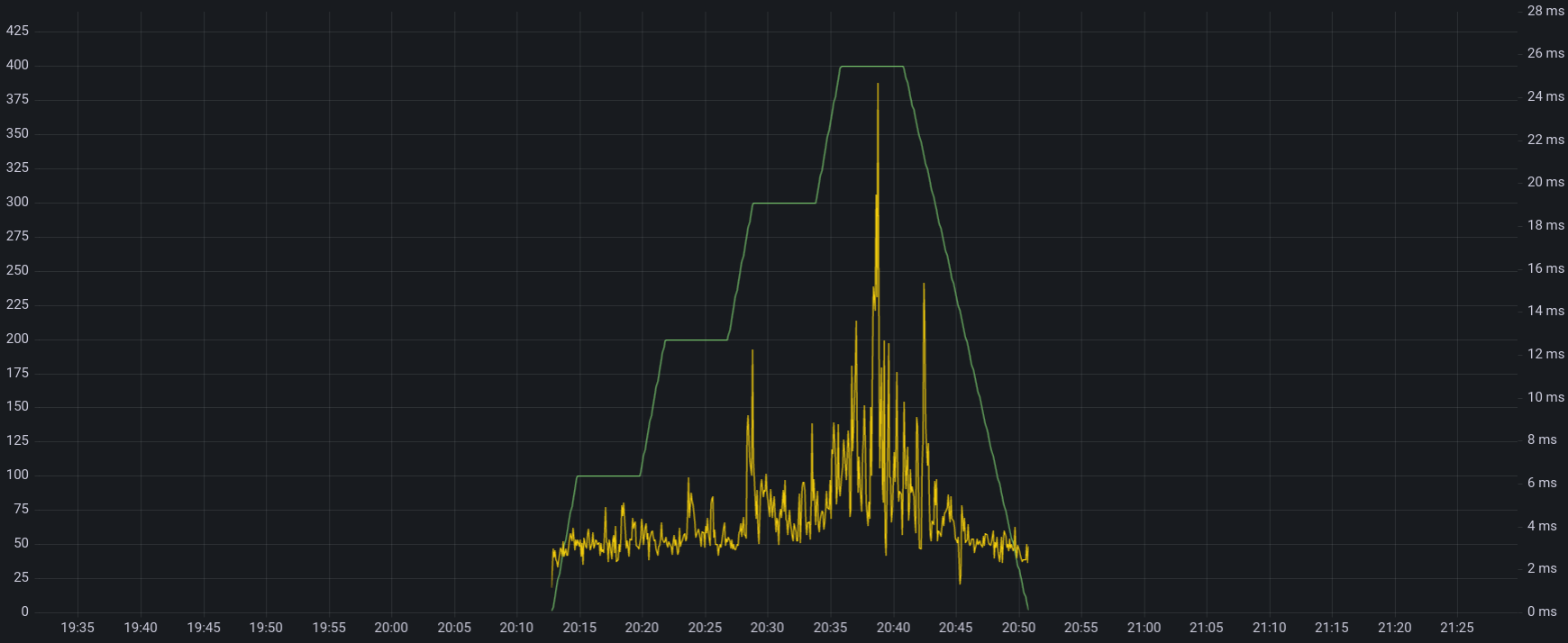


圖11-2、在 Docker 上的 Express Gateway 的反應時間

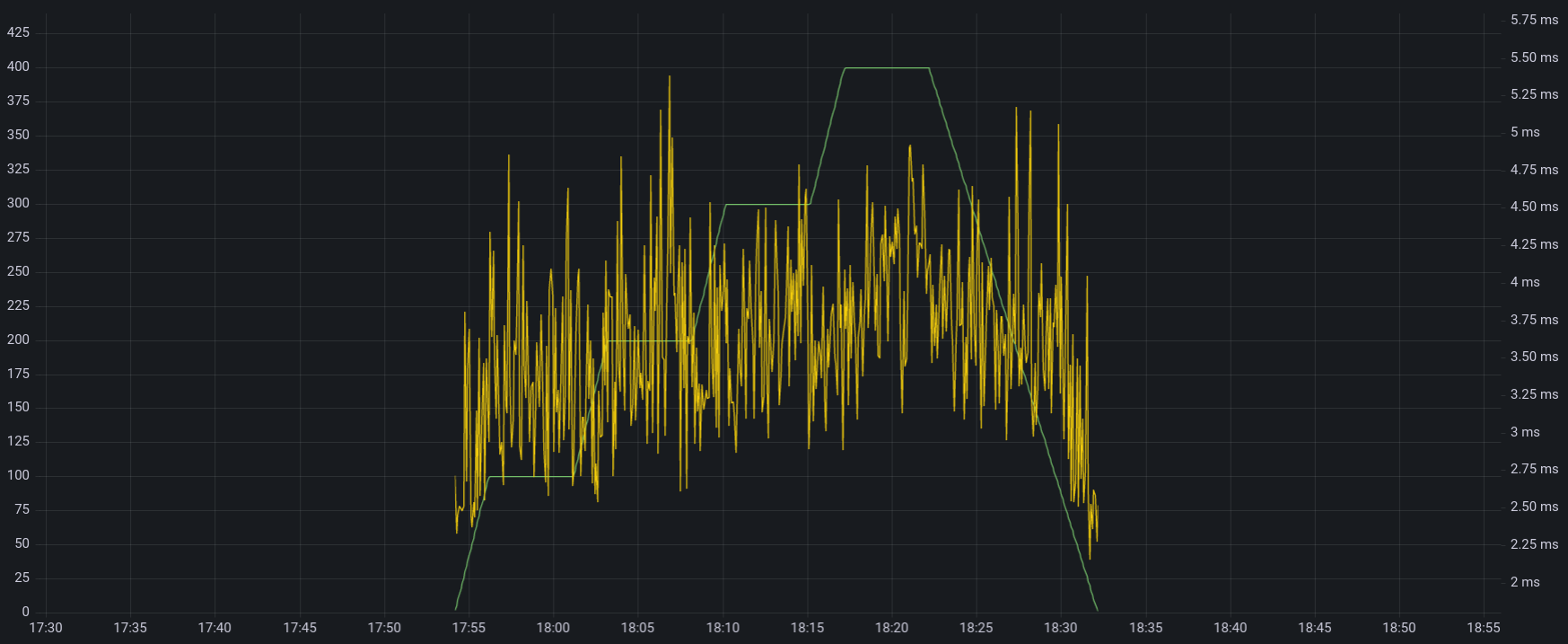


圖11-3、在 K8s 上的 Express Gateway 的反應時間

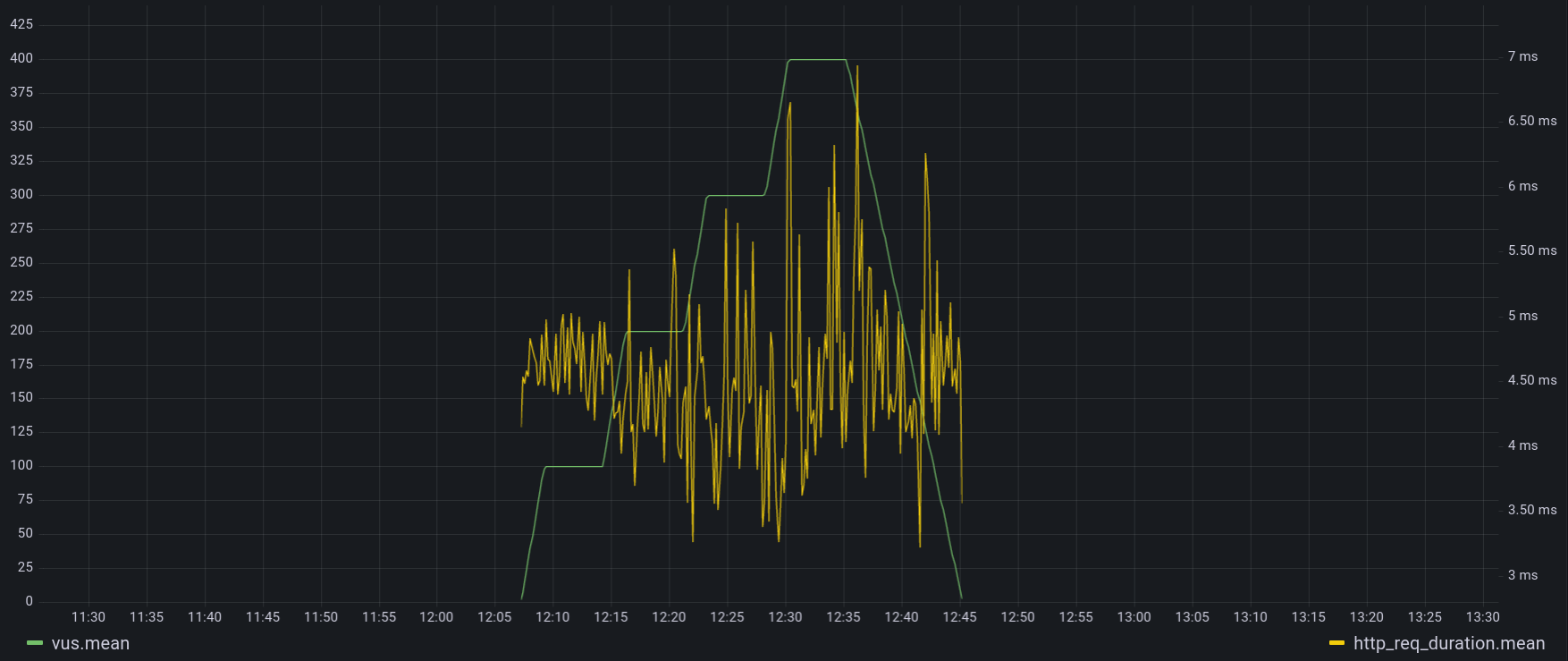


圖12-1、在 VMware 上的 API Umbrella 的反應時間

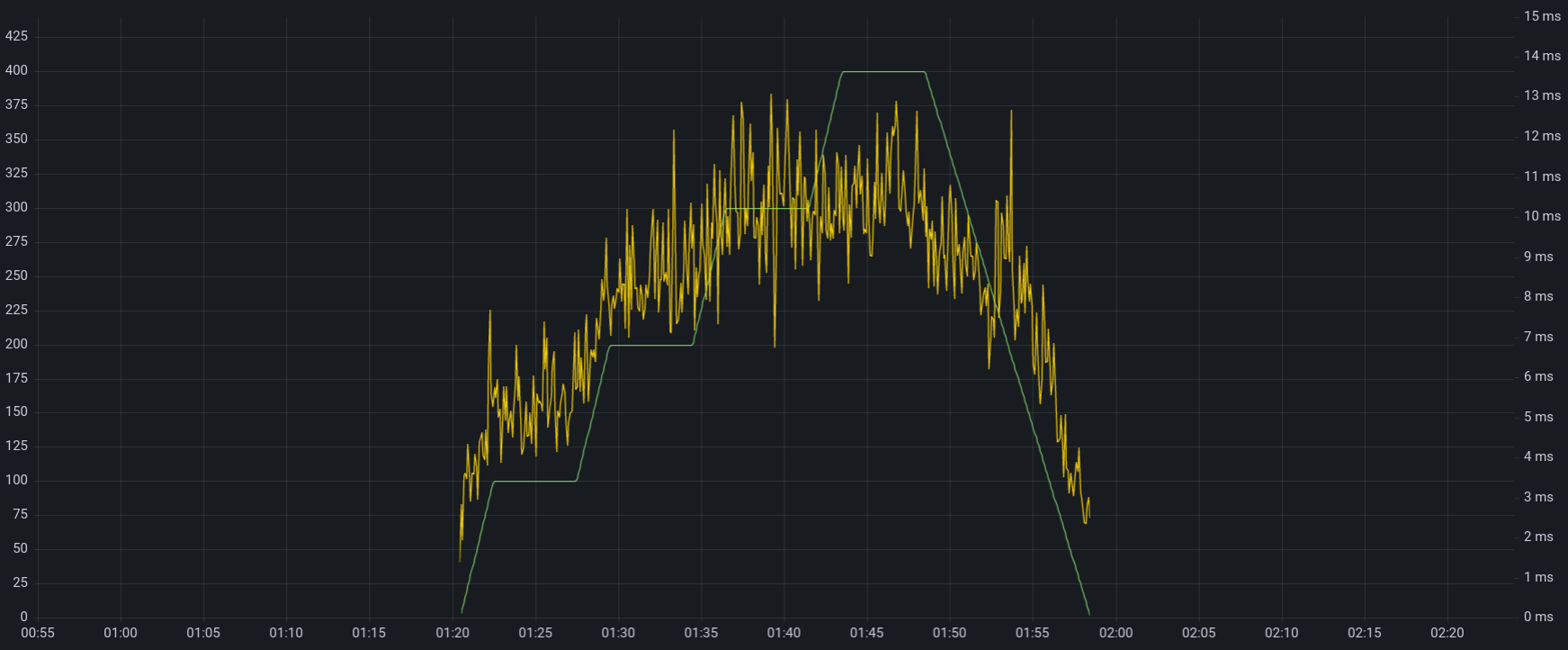


圖12-2、在 Docker 上的 API Umbrella 的反應時間

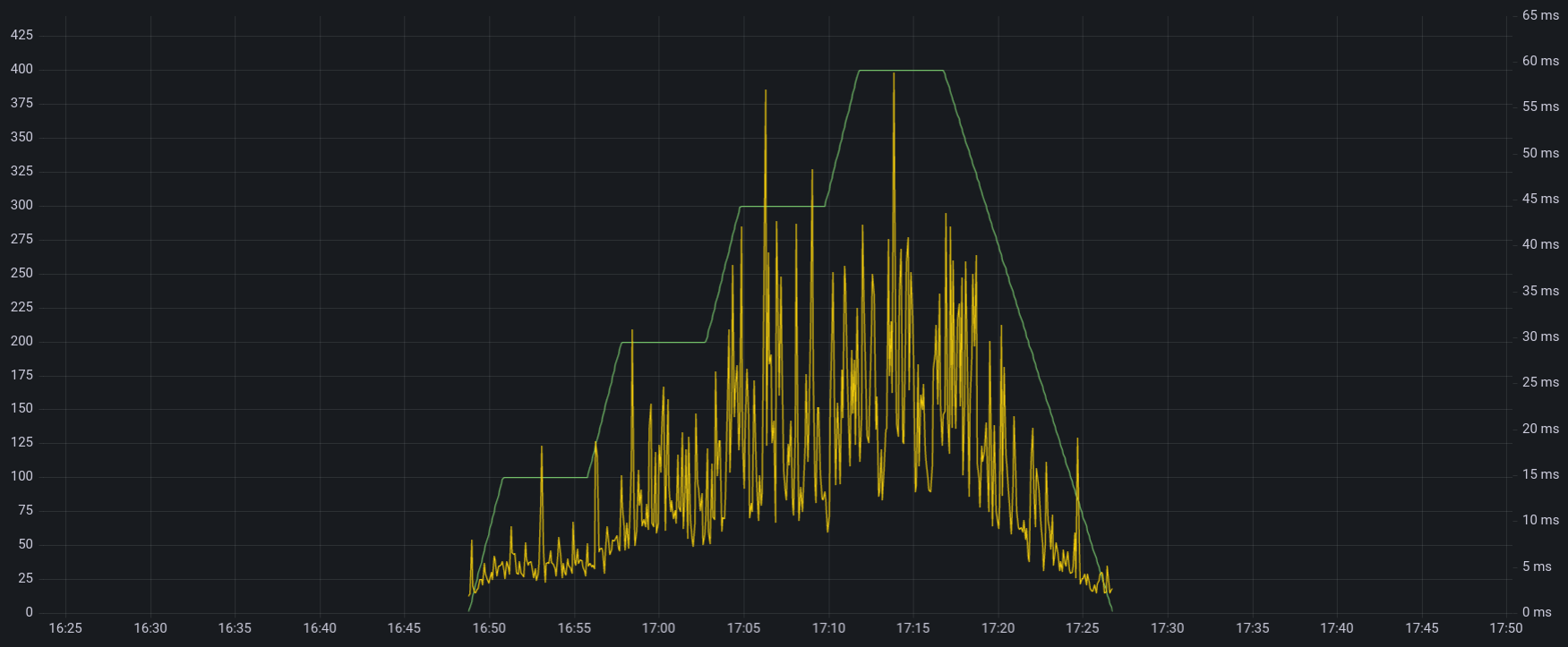


圖13-1、在 VMware 上的 Kong API Gateway 的反應時間



圖13-2、在 Docker 上的 Kong API Gateway 的反應時間

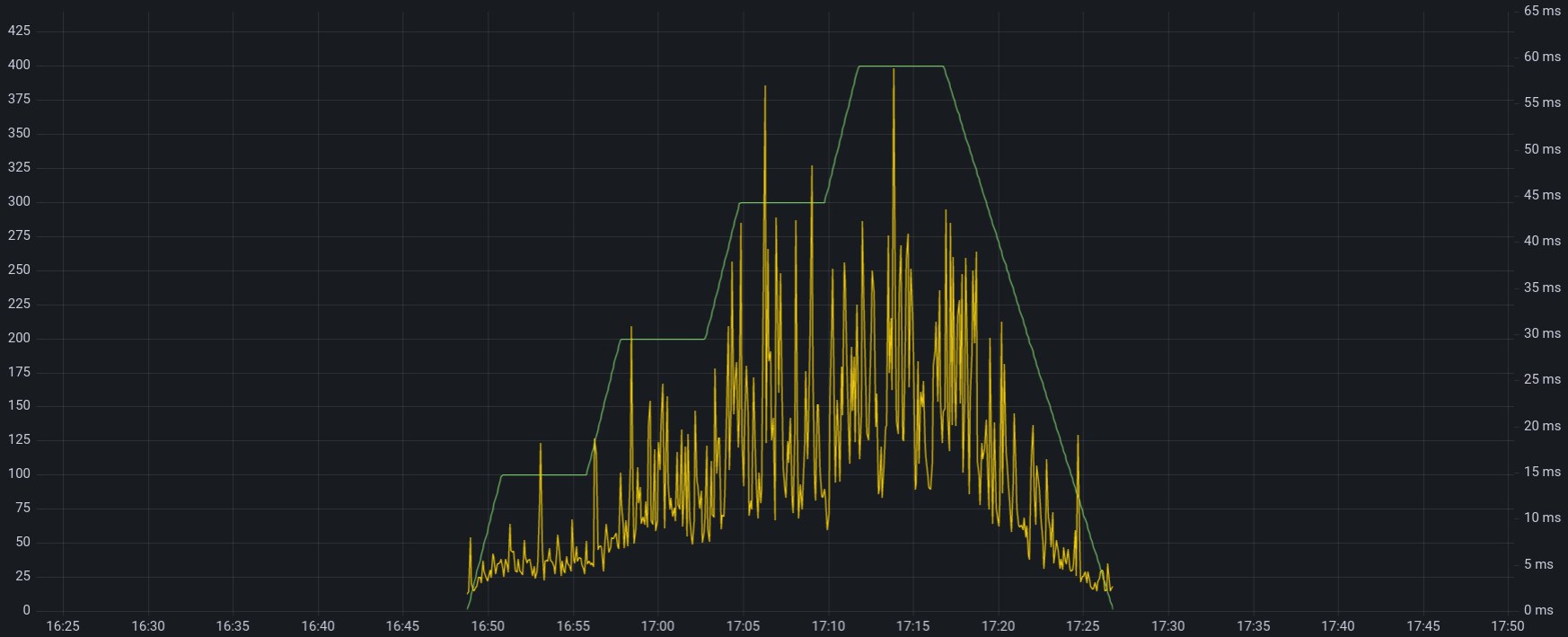


圖13-3、在 K8s 上的 Kong API Gateway 的反應時間



* + Spike Testing

前 10 秒鐘從 0 個 virtual user 增加到 100, 再來 1 分鐘維持 100 個。再來 10 秒鐘從 100 增加到 1400, 再來 3 分鐘維持 1400 個。再來 10 秒鐘從 1400 降到 100, 再來 3 分鐘維持 100 個。最後 10 秒鐘從 100 降到 0 個。

圖15、API 來源的反應時間



圖16-1、在 VMware 上的 Express Gateway 的反應時間



圖16-2、在 Docker 上的 Express Gateway 的反應時間

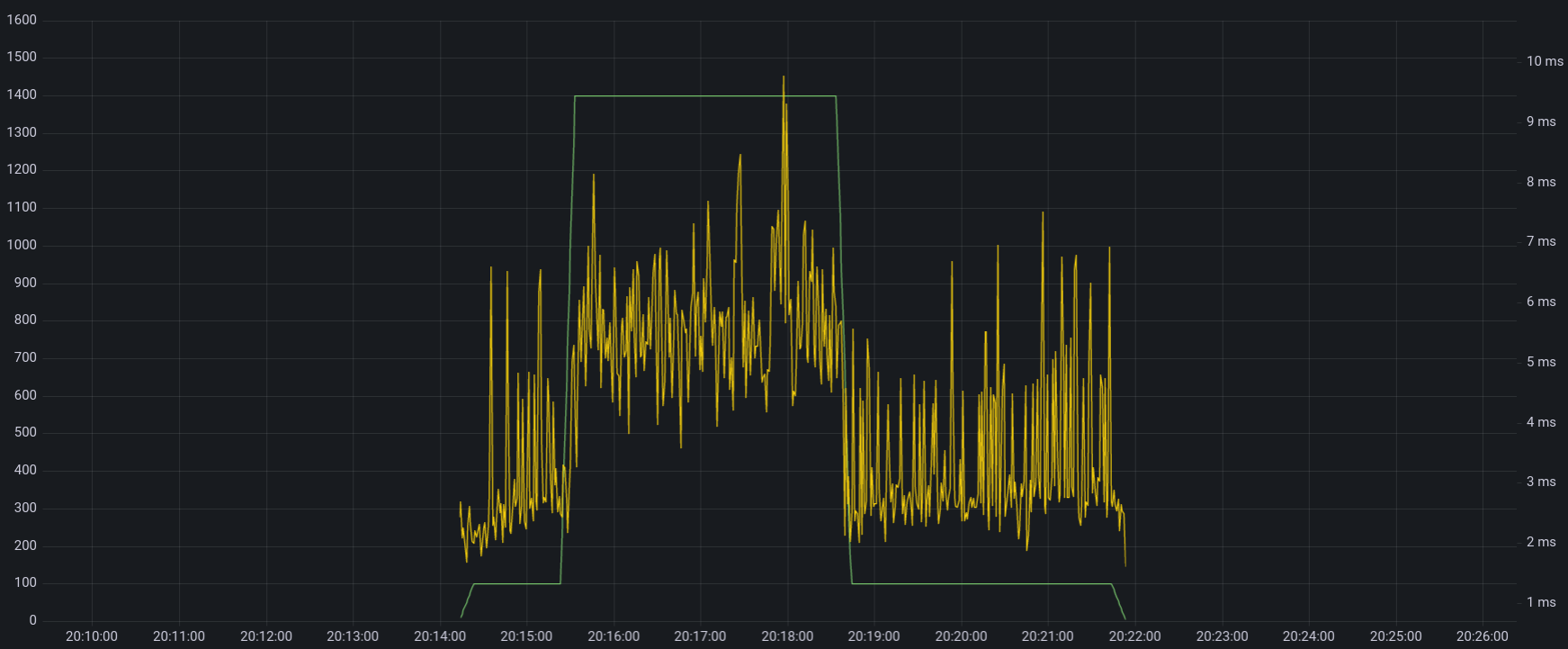


圖16-3、在 K8s 上的 Express Gateway 的反應時間

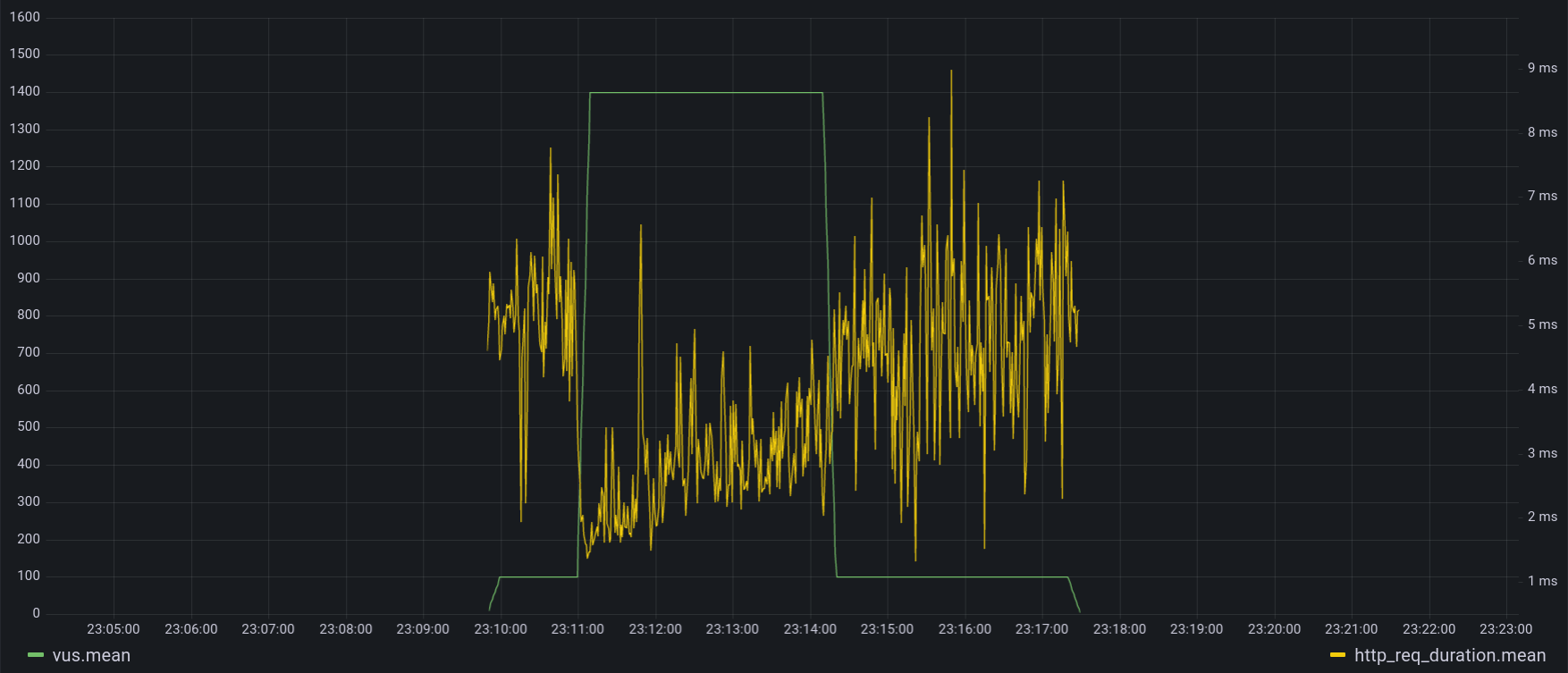


圖17-1、在 VMware 上的 API Umbrella 的反應時間



圖17-2、在 Docker 上的 API Umbrella 的反應時間

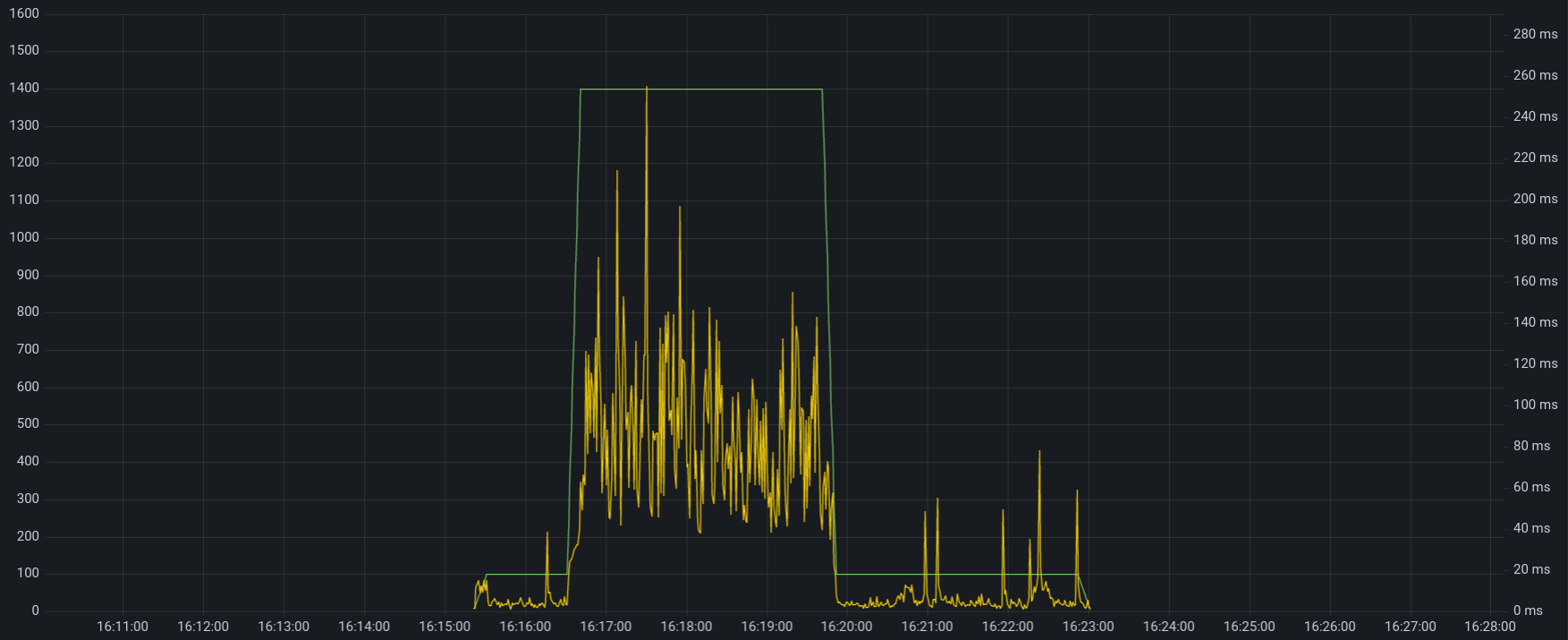


圖18-1、在 VMware 上的 Kong API Gateway 的反應時間

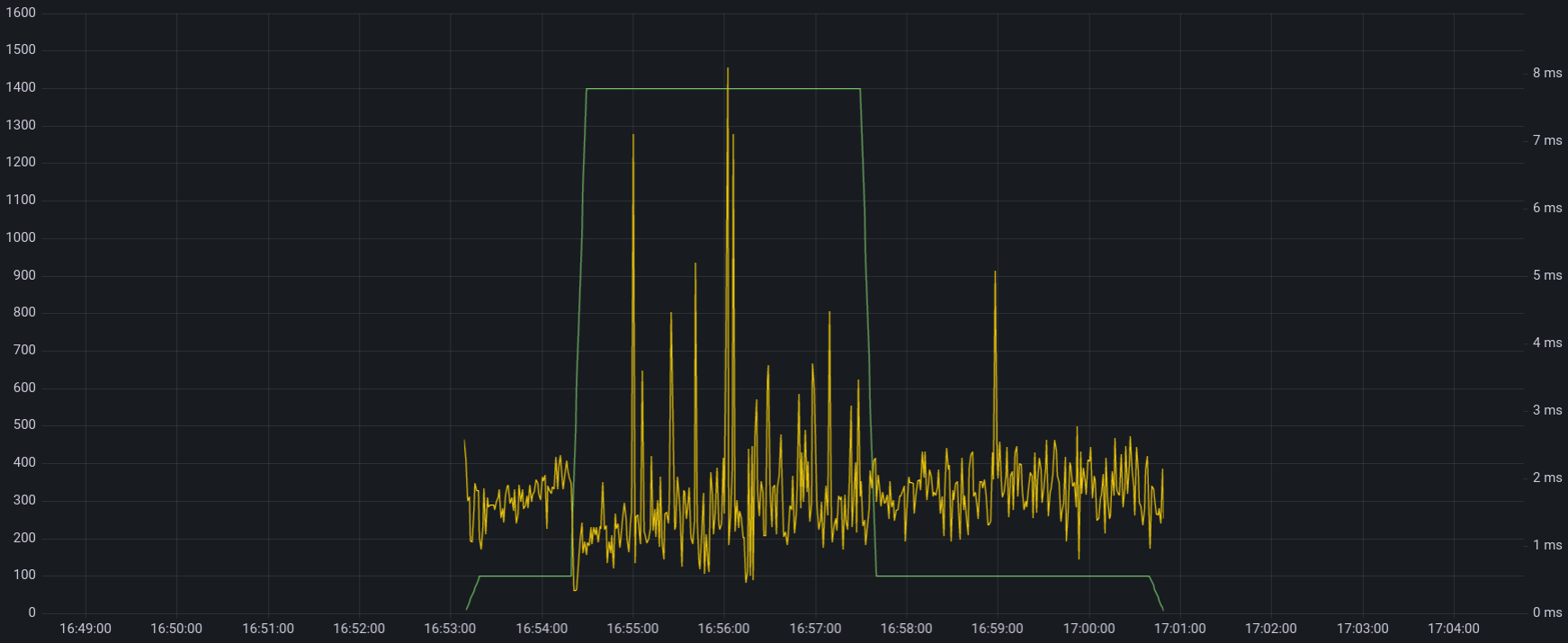


圖18-2、在 Docker 上的 Kong API Gateway 的反應時間

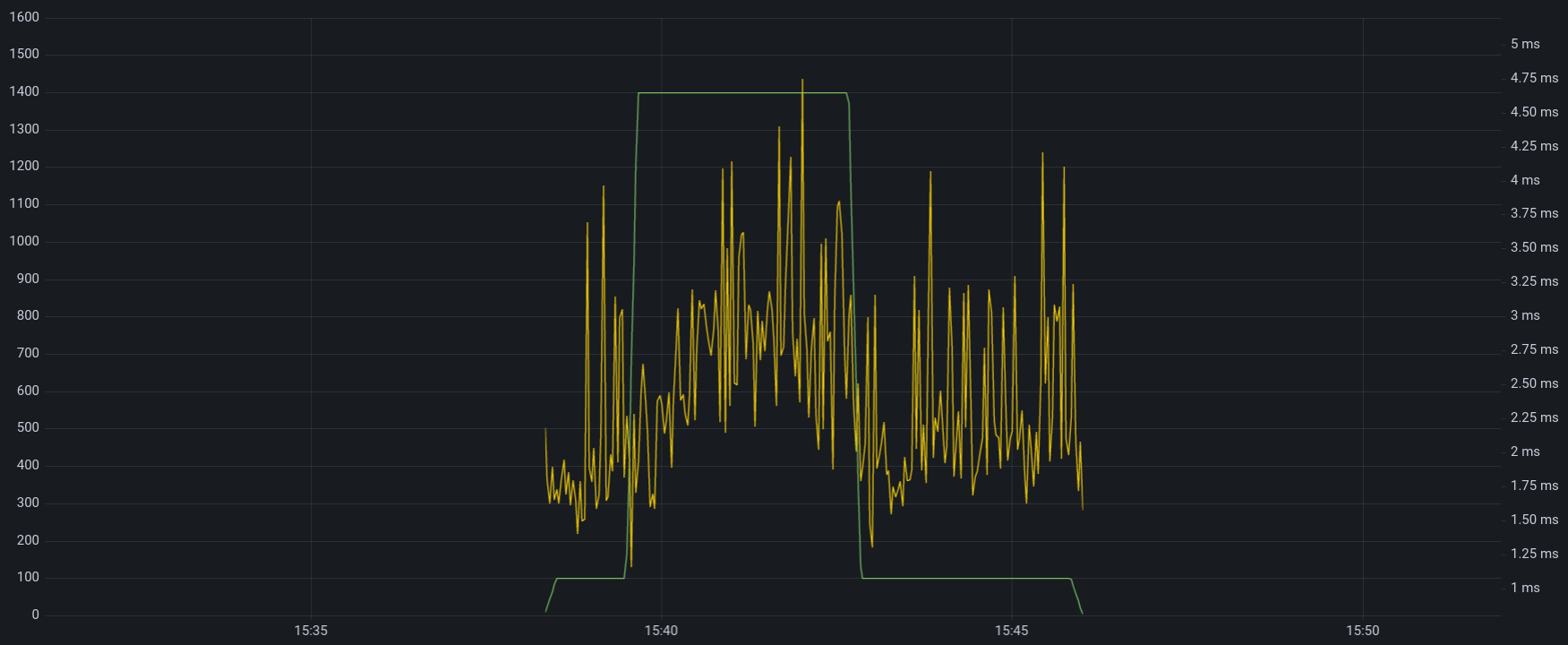


圖18-3、在 K8s 上的 Kong API Gateway 的反應時間



* **結論**
  + API Umbrella 在實驗中, 相較於其他兩個 API Gateway, 效能相對地較差, 且已經不怎麼維護了
  + Express Gateway 和 Kong API Gateway 效能幾乎是差不多的, 不過在社群的活躍度上, Kong API Gateway 明顯略勝一籌。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 Web GUI 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 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。
  + K8s 相對於 Docker 來說, 又多了一層 network hopping, 因為多了 Service 在 request 傳到 Pod 之前。但因為有了 Service 來幫我們 load balancing, 所以每個 Pod 才能平分負擔。在負擔較大的情況下, 我們可以明顯看到 K8s 的表現相對比單一個 Docker Container 好。
* **參考文獻**

[1] “ExpressGateway/express-gateway: A microservices API Gateway built on top of Express.js.” n.d. GitHub. Accessed March 29, 2022. https://github.com/ExpressGateway/express-gateway.

[2] “NREL/api-umbrella: Open source API management platform.” n.d. GitHub. Accessed March 29, 2022. https://github.com/NREL/api-umbrella.

[3] “Kong/kong: 🦍 The Cloud-Native API Gateway.” n.d. GitHub. Accessed March 29, 2022. https://github.com/Kong/kong.

[4] “一起幫忙解決難題，拯救IT 人的一天.” n.d. iT 邦幫忙. Accessed March 23, 2022. https://ithelp.ithome.com.tw/articles/10250841.

[5] “Kubernetes 基礎教學（一）原理介紹. Kubernetes 如何運作？什麼是Pod？什麼是Node？Master… | by Cheng-Wei Hu | 胡程維| Medium.” n.d. 胡程維. Accessed March 29, 2022. https://cwhu.medium.com/kubernetes-basic-concept-tutorial-e033e3504ec0.