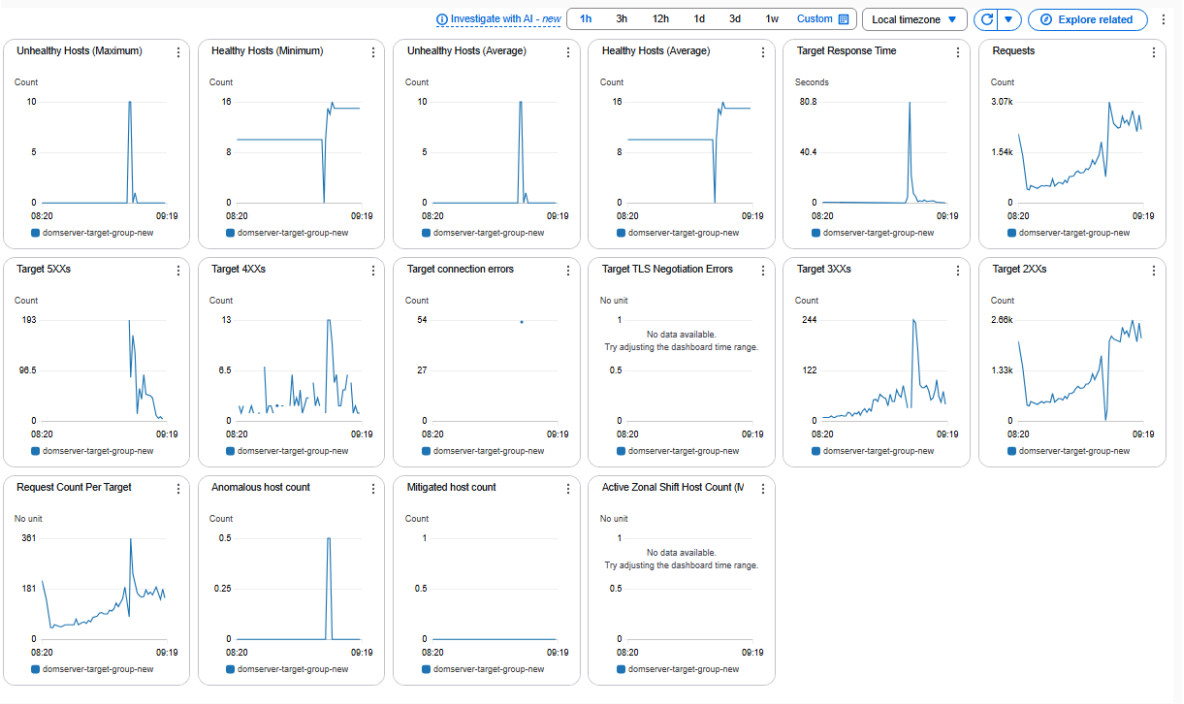
一、事件背景

在賽事開始前，為確保 DOMjudge 網站服務能承受預期流量，我們預先在 AWS EC2的環境中啟動了 5 台 medium 規格的伺服器，並在此之上開啟 10個 DOMjudge 網站容器。

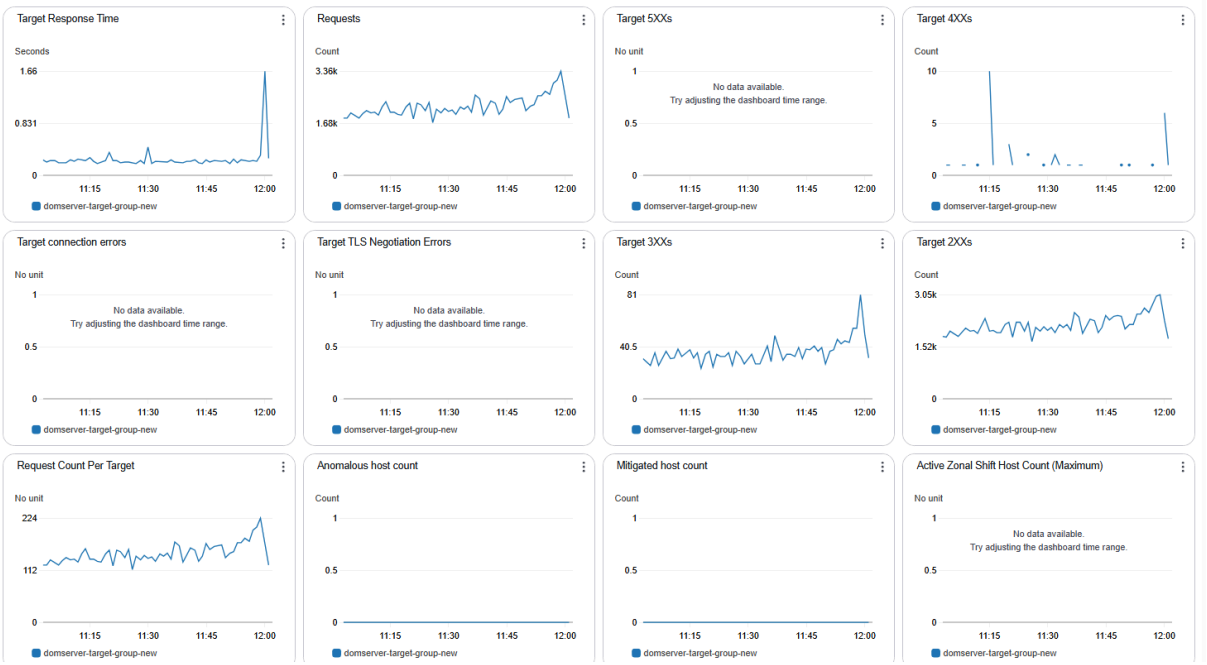
二、事件經過與處置



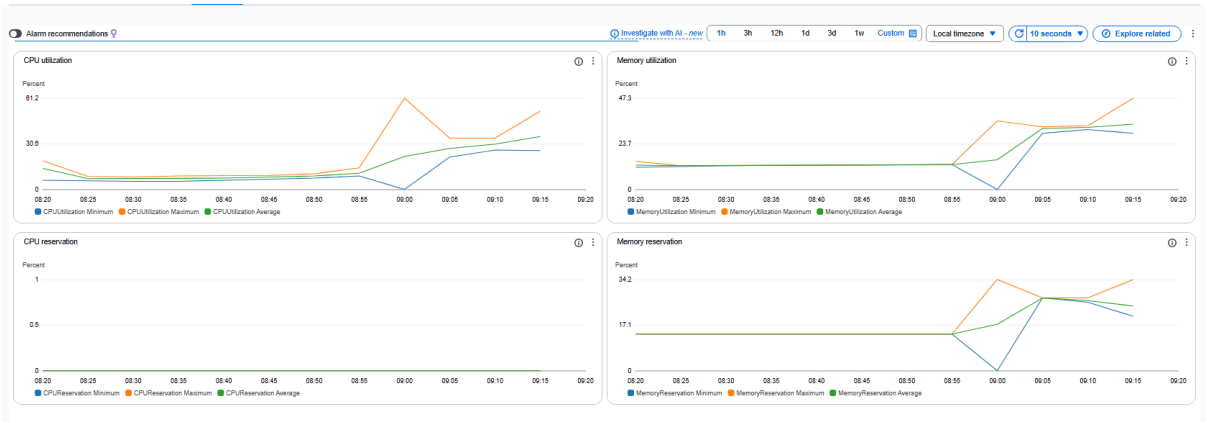
圖一、賽事期間網站流量與效能監控數據

1. 事件發生：於賽事正式開始的瞬間，系統監控數據顯示，裁判系統的回應時間開始急遽上升。使用者也幾乎同時開始回報有因伺服器未能及時回應的 504 Gateway Timeout 錯誤。
2. 即時監控與處置：團隊立即檢視即時監控指標，發現當前流量已超過 10 個容器的承載極限，且負載持續攀升。判斷瓶頸出現在 DOMjudge 前端伺服器，無法有效處理湧入的請求。團隊立即執行緊急擴容，增開了 5 個 DOMjudge 容器，使容器總數達到 15 個，藉以分散並消化龐大的流量，使 DOMjudge 於9:05分左右恢復正常。

三、調查結果與分析



圖二、賽事期間HTTP請求回應狀態碼分佈



圖三、DOMjudge 前端伺服器CPU與記憶體使用率變化

事後數據分析顯示，網頁伺服器是本次效能瓶頸的關鍵。詳細數據如下：

- 開賽初期（**10台容器時**）：我們紀錄到開賽瞬間的最大請求量高達 **3074 reqs/min**。在當時由 10 個容器組成的叢集中，單一容器需要處理高達 **361 reqs/min** 的請求。此負載量已遠超過其穩定處理能力。
- 擴容後：在緊急增開 5 個容器後，容器總數達到 15 台。此時，儘管總體請求量降至 **2277 reqs/min**，但單一容器的平均處理量顯著下降至 **151 reqs/min**，降幅超過一倍，顯示擴容措施有效地緩解了單機壓力。
- **504 Timeout** 原因：在事件高峰期，伺服器同時等待回應的請求數一度飆升至 **1200** 筆。由於前端伺服器無法在預設的超時時間內完成處理並回傳回應，最終導致大量的 504 Timeout 錯誤產生。

- 根本原因分析：雖然我們確認了瓶頸在於網頁伺服器，但本次事件的根本原因（Root Cause）仍無法精確確定，尚不清楚是伺服器的 CPU、記憶體或與後端資料庫的連線池達到上限，導致服務無力處理。

四、結論與未來建議

本次事件證明，現有的網站容器配置在應對開賽時瞬間爆發的高流量時仍顯不足。根據本次經驗，我們保守估計，單一網站容器的穩定負載上限約為 每分鐘 **200** 次請求。

為了避免再次發生類似事件，我們提出以下兩點建議供未來舉辦時參考：

1. 提前進行壓力測試：在比賽開始前，提前進行全面的壓力測試。這能提早發現系統在極限負載下的潛在瓶頸，並進行相應的最佳化。
2. 啟動自動擴展策略（**Auto Scaling Policies**）：建議設定並啟用 AWS 的自動擴展策略。透過監控如 CPU 使用率或請求佇列長度等關鍵指標，當指標超過預設閾值時，系統將能自動新增容器來應對突發流量。這將大幅減少人工反應時間，從而確保服務的穩定性和可用性。