

崑山科技大學

資訊工程系

111學年度專題製作報告

可依使用者需求動態配置的雲端資安學習
服務平台

Dynamic Configurable Cloud-based Cybersecurity
Learning Service Platform on User Needs

學 生：羅煥璋、施治宇、胡凱竣

指導老師：曾龍

中華民國 112 年 04 月

可依使用者需求動態配置的雲端資安學習 服務平台

Dynamic Configurable Cloud-based Cybersecurity Learning Service Platform on User Needs

學 生：	羅竣瑋	Student:	LUO JUN-WEI
	施治宇		SHIH JHIH-YU
	胡凱竣		HU KAI-JUN
指導老師：	曾龍	Advisor:	TSENG LUNG

崑山科技大學

資訊工程系

111學年度專題報告

A Report

Submitted to Department of Information Engineering

Kun Shan University

in Partial Fulfillment of the Requirements

for the Degree of Bachelor

in

Information Engineering

June 2023

Tainan, Taiwan, Republic of China

中華民國 112 年 4 月

可依使用者需求動態配置的雲端資安學習
服務平台

學 生

胡凱竣 4080E001

指導老師： (簽章)

可依使用者需求動態配置的雲端資安學習服務平台

學 生：羅竣瑋

指導老師：曾龍

施治宇

胡凱竣

崑山科技大學資訊工程系

摘 要

我們選擇在 Kubernetes (K8S) 上部署 CTFd 開源平台，並將現有的 CTFd 平台整合到 K8S 中，以實現動態配置題目的功能。透過 K8S 的自動化調度和容器管理功能，我們能夠有效地管理解題平台的運作，實現題目的動態啟動與關閉，並在使用者需要時快速配置所需的題目範圍。

Kubernetes 的強大功能體現在其分散式架構和自我修復能力上。它能夠自動平衡伺服器資源負載，根據使用者需求調整容器，確保平台的高可用性和可靠性。此外，K8S 提供了彈性擴展的能力，使我們能夠根據流量需求調整容器數量，有效地利用資源並避免不必要的資源浪費。

因此，我們的主要目標是利用上述所講的 K8S 獨有功能性，在動態部署和配置題目的過程中提供高效且可彈性擴展的解題平台，以滿足不同使用者的需求並確保最佳的資源利用。

同時，我們還將整合普羅米修斯 (Prometheus) 作為監控和警報系統，以進一步增強我們的解題平台的可靠性和可用性。

DYNAMIC CONFIGURABLE CLOUD-BASED CYBERSECURITY LEARNING SERVICE PLATFORM ON USER NEEDS

Student: LUO JUN-WEI

Advisor: TSENG LUNG

SHIH JHIH-YU

HU KAI-JUN

Department of Information Engineering
Kun Shan University

Abstract

We have chosen to deploy the CTFd open-source platform on Kubernetes (K8S) and integrate our existing CTFd platform into K8S to achieve the functionality of dynamically configuring challenges. Through the automation and container management capabilities of K8S, we can effectively manage the operation of our challenge platform, enabling the dynamic startup and shutdown of challenges, and quickly configuring the desired range of challenges when needed by users.

The strength of Kubernetes lies in its distributed architecture and self-healing capabilities. It can automatically balance server resource loads and adjust containers based on user demands, ensuring high availability and reliability of the platform. Additionally, K8S provides the ability for elastic scaling, allowing us to adjust the number of containers based on traffic requirements, effectively utilizing resources and avoiding unnecessary resource waste.

Therefore, our main objective is to leverage the unique functionalities of K8S mentioned above to provide an efficient and flexible scalable challenge platform for dynamic deployment and configuration of challenges. This enables us to meet the needs of different users and ensure optimal resource utilization.

Furthermore, we will integrate Prometheus as a monitoring and alerting system to further enhance the reliability and availability of our challenge platform.

誌 謝

本專題能如期順利地完成，首先要感謝我們的偉大恩師曾龍教授，教授潛心學術研究，有幸追隨恩師求學，使的我們在學習與態度上得到了許多的啟發，專業知識和經驗為我們提供了寶貴的指引，使我們能夠在正確的軌道上前進，您的鼓勵和建議激勵著我們不斷進步，受益匪淺，我們在此由衷的感謝教授專業且細心的指導，在此向恩師致上我們最崇高的敬意與無限的感激。

在此，也要感謝我們的班級導師邱正毅助理教授，感謝他的悉心指導與關照，老師時常提醒我們學校的文件繳交時程與應準備的文件，還能適時的給予建議與方向，由衷的感謝老師的細心關照。

感謝組員們過去一年對專題的貢獻，大家通力合作、共同努力，才使我們能夠順利完成並呈現出現在的專題成果，同時，我們要感謝崑山科技大學的支持，學校提供了良好的研究環境與資源，讓我們能夠順利地深入研究。

最後，我們要感謝在這一路上給予協助與支持的朋友們，你們的支持和鼓勵使我們克服了许多困難和挑戰，你們的意見和建議為我們的研究提供了新的方向與新奇的點子，感謝你們在我們需要幫助時總是伸出援手，給予協助，並與我們共同追求卓越。

目錄

圖目錄	V
表格目錄	1
第一章 緒論	2
1.1 研究背景	2
1.2 研究動機與目的	3
1.3 研究方法與系統概述	5
1.4 專題內容架構概述	6
第二章 相關研究	7
2.1 Google Cloud Platform	7
2.2 Kubernetes	8
2.3 Container	10
2.4 CTFd (Capture The Flag Framework)	11
2.5 CTFd-owl	12
2.6 Prometheus	12
2.7 Line Notify	13
第三章 專題系統架構	14
3.1 系統核心架構	14
3.2 監控端架構	15
3.3 題目動態部署架構	16
第四章 系統實現	17
4.1 關鍵核心技術	17
4.2 系統成品	45
第五章 結論	49
參考文獻	50

圖目錄

圖1 CTFd 網站示意圖	11
圖2 系統核心架構部署圖 (DEPLOYMENT DIAGRAM)	14
圖3 監控端架構圖	15
圖4 題目動態部署架構圖	16
圖5 網頁樹狀圖	38
圖6 註冊介面	45
圖7 登入介面	45
圖8 題目選擇介面	46
圖9 題目動態部署示意圖1	46
圖10 監控端圖表監控畫面1	48
圖11 監控端數據監控畫面1	48

表格目錄

表1 傳統配置與動態配置差異表1	4
表2 傳統配置與動態配置差異表2	9
表3 STORAGECLASS 配置文件	18
表4 DB-PVC 配置文件	19
表5 CTFD 平台 LOG 用的 PVC 配置文件	20
表6 CTFD 平台存放上傳物件用的 PVC 配置文件	21
表7 K8S SECRET 配置文件	23
表8 MYSQL 配置文件	24
表9 MYSQL 的 SERVICE 配置文件	26
表10 PHPMYADMIN 的配置文件	27
表11 CTFD 解題平台 CONTAINERS 的配置文件	29
表12 CTFD 解題平台 SECURITYCONTEXT 的配置文件	30
表13 CTFD 解題平台 VOLUMES 的配置文件	31
表14 CTFD 解題平台 SERVICE 的配置文件	33
表15 動態部署核心關鍵程式碼	36
表16 監控端網頁主模板程式碼	40
表17 監控端主頁面程式碼	41
表18 監控端發送請求至PROMETHEUS程式碼	43
表19 告警端程式碼	44

第一章 緒論

1.1 研究背景

隨著資訊安全意識的提升和網絡攻擊風險的增加，CTF（Capture The Flag）比賽作為一種測試和增強安全技能的方式是有效地並且受到廣泛應用。CTFd 是一個被廣泛使用的開源平台[6]，它提供了一個用於舉辦 CTF 比賽的網頁框架和功能。

然而，隨著 CTF 比賽的複雜化和多樣化，傳統的 CTFd 平台在動態部署和配置題目方面存在著一些挑戰。傳統方式下，題目需要手動配置和部署，這可能導致配置錯誤、效率低和伺服器資源浪費的問題。同時，傳統部署方式難以滿足使用者對於即時啟動和題目關閉的需求，尤其在大規模 CTF 比賽中，解題題目佔據了伺服器大部分的資源，所有參賽者都使用同一個解題資源，容易造成資源互搶問題。

為了克服這些挑戰，本專題研究選擇在 Kubernetes（K8S）上部署 CTFd 開源平台並進行整合，以實現動態配置題目的功能。Kubernetes 作為一個強大的容器管理平台，提供了自動化調節、容器管理和分散式架構等功能，能夠有效地管理解題題目與平台的運作。

通過將 CTFd 平台整合到 Kubernetes 中，我們能夠利用 Kubernetes 的自動化調度和容器管理功能，實現題目的動態啟動和關閉，並在使用者需要時快速配置所需的題目範圍。Kubernetes 的自我修復能力可以確保平台的高可用性和可靠性，並通過自動平衡伺服器資源負載，確保 CTFd 平台的穩定性。

此外，Kubernetes 提供了彈性擴展的能力，使我們能夠根據使用者數量需求調整容器數量，有效地利用資源並避免不必要的資源浪費。這種高效且可彈性擴展的解題平台能夠滿足不同使用者的需求，也能讓不同網頁管理者更好進行管理，確保最佳的資源利用，提高 CTF 比賽的效率和可靠性。

隨著數位化時代的來臨，現代應用程式和系統的複雜性不斷增加，同時也帶來了更高的運營挑戰。我們為了確保應用程式和系統平台的可靠性、可用性和效能，監控和警報系統變得至關重要，普羅米修斯（Prometheus）作為一個開源監控系統，迅速崛起並成為廣受歡迎的解決方案。

在這樣的背景下，普羅米修斯（Prometheus）作為一個開源監控系統，迅速崛起並成為廣受歡迎的解決方案。普羅米修斯提供了全面且靈活的監控能力，能夠收集、儲存和分析各種指標數據，包括容器、主機和應用程式層面的監控資料。

我們在 Kubernetes 上部署 CTFd 平台並整合現有的功能，我們能夠實現動態配置題目的目標，並提供高效且可彈性擴展的解題環境。

利用普羅米修斯在監控和警報方面的強大功能，結合 Kubernetes 的自動化和容器管理特性，提升解題平台的可靠性、可用性和效能。通過對關鍵指標數據的監控和即時警報，我們能夠實時掌握平台的狀態並快速響應任何潛在的問題，從而提供更好的使用者體驗和服務品質。

這項專題有助於改進傳統 CTFd 平台的靜態部署限制，提供更靈活、高效的解題環境，從而促進資訊安全技能的培養和測試。

1.2 研究動機與目的

CTF（Capture The Flag）比賽在資訊安全領域中被廣泛使用，它不僅可以促進學習資訊安全的人員提升技能，還能夠測驗專業人士對於資訊安全領域的理解程度。然而，傳統的 CTF 平台在部署和配置題目方面存在限制，無法有效地滿足不斷增長的使用者需求和變化的資源負載。

因此，本研究的動機在於改進傳統 CTF 平台的靜態部署限制，提供高效且可彈性擴展的解題平台，以滿足不同使用者的需求並確保最佳的資源利用。傳統配置與動態配置差異表如下：

	傳統靜態部署方式	動態部署方式
部署時間	長	短
人工處理	需要	不用
部署難度	難	簡易
資源浪費	高	低
技術成分	高	低

表1 傳統配置與動態配置差異表1

本研究的目的是在於 Kubernetes (K8S) 上部署 CTFd 平台並整合現有的功能，以實現動態配置題目的功能。具體目標如下：

1. 提供解題平台題目的動態配置能力：利用 Kubernetes 的自動化調度和容器管理功能，實現題目容器的動態啟動與關閉，並在使用者需要時快速配置所需的題目範圍。這將提高解題平台的靈活性和效率。
2. 實現平台的可擴展性：利用 Kubernetes 的彈性擴展能力，根據使用者數量需求動態調整容器數量，有效地利用資源並避免不必要的資源浪費。這將提高解題平台的可靠性和可用性。
3. 簡化平台管理：利用 Kubernetes 的自動化調度和容器管理功能，簡化題目配置和平台管理的工作量。透過Kubernetes的指令和工具，輕鬆進行容器的部署、更新和管理，提高平台管理的效率。

此外，本研究的目的還在於探索 Kubernetes 在 CTF 比賽中的應用，並為資訊安全領域的學術和實踐提供一個新的解題平台設計和架構。

最後，為了確保應用程式和系統的可靠性、可用性和效能，監控和警報系統變得至關重要，普羅米修斯 (Prometheus) 作為一個開源監控系統，迅速崛起並成為廣受歡迎的解決方案。普羅米修斯提供了全面且靈活的監控能力，能夠收集、儲存和分析各種指標數據，包括容器、主機和應用程式層面的監控資料。

1.3 研究方法與系統概述

我們首先對現有的 CTFd 平台進行了以下需求分析：

(1) 了解使用者與平台維護者的需求

對使用者（參賽者）而言，若在競賽時遇到伺服器回應時間不佳，或競賽題目出錯無法連接，必定會影響賽事進行，也會產生公平性問題。

對平台維護者而言，開始競賽前必須先手動啟動所有挑戰題目，若競賽題目很多將會花費更多時間。另一個問題是，題目全部需要手動配置和部署，這可能導致配置錯誤、效率低和伺服器資源浪費的問題發生，管控不易，若競賽過程中發生大量題目下線，人工處理將會花費更多時間。

綜合上述問題，發生問題的主要原因在於以下幾點：

1. 多人使用同一道挑戰題目資源，無法做到分流管控，人數一多最終伺服器會回應過慢，甚至出現嚴重錯誤以至於無法連接。
2. 管理者難以維護、管理挑戰題目，造成許多人力資源、時間浪費以及技術成本偏高。

根據上述了解使用者與平台維護者的需求分析，最終確定了動態配置題目的功能需求必要性。

同時，我們研究了如何最佳化的動態配置一個挑戰容器（Container）以及解題平台（網站），最終得知 Kubernetes（K8S）的特性以及優勢，確定了它適合作為我們的容器管理和自動化調節平台。

此外，為了確保能更加平台執行的穩定性與安全性，我們需要有一個系統能夠監控解題平台，供管理者查看平台狀況，並在需要時發出警報的功能，為了實現上述功能我們所選擇的是 Prometheus 開源的系統監控和警報的工具包，以此達到系統監控的功能，而警報功能採用了 Line Notify 提供的現有功能，來達到即時通知平台管理者的功能。

最後，此專案進行了資源分配與分工，主要功能拆分成了以下三點：

1. 解題平台與挑戰题目的動態部署功能
2. 解題平台的監控系統
3. 監控系統的即時警報

1.4 專題內容架構概述

本專題將會採用 Google Cloud Platform (GCP) 上所提供的 Google Kubernetes Engine (GKE) 服務進行架設，章節架構共分為以下五個：

第一章：緒論

說明研究背景、動機與目的、研究方法與系統概述、專題內容概述。

第二章：相關研究

說明並探討本專題之技術研究，包含但不限於：

GCP、K8s、Container、CTFd、CTF-owl、Prometheus、Line Notify。

第三章：專題系統架構

說明本專題設計之系統架構、資料與平台服務細節。

第四章：系統實現

說明本專題實作時所採用的關鍵技術部分。

第五章：結論

說明本專題之結論。

第二章 相關研究

2.1 Google Cloud Platform

Google Cloud Platform (GCP) 是一個由 Google 提供的雲端運算平台，提供了豐富的雲端服務和工具，包括計算、儲存、資料庫、應用程式開發、人工智慧和機器學習等。在GCP平台上幾乎解決掉了所有的基礎設施建置的過程，只需要手動啟動所需要的服務就可以馬上使用，GCP也對於近幾年相當流行的容器化技術也有提供相關的服務，像是 GKE、Google Cloud Run、Google Container Registry、Google Cloud Build 等等的服務，都是對於容器化所提供的服務，所有服務都在雲端上，只要有網路就能隨時隨地存取，不受地理位置因素影響。

在進行專題研究前，我們進行了廣泛的調查和比較不同的雲端平台，包括Google Cloud Platform (GCP)、Amazon Web Services (AWS) 和Microsoft Azure等。這些平台都具有強大的雲端計算能力，我們最終選擇 Google Cloud Platform (GCP) 作為基礎研究平台的考量原因和優勢如下：

1. 較廣的服務範圍和功能：GCP 提供了廣泛且全面的雲端計算服務和功能，包括計算、儲存、網路、資料庫、機器學習、人工智慧等領域。這使得使用GCP能夠滿足各種不同的應用需求與場景。
2. 全球通用性佳：GCP 支援全球各地，能夠提供高可用性和可靠的服務，這使得用戶可以在全球範圍內都能隨時隨地部署應用，並為用戶提供低延遲和高速度的服務。
3. 安全性：GCP 具有高等級別的安全性措施，提供了許多安全功能，包括身份和訪問管理、防火牆、流量管制等等。

4. 簡易性：GCP 提供直觀且易於操作的使用者介面與管理功能，能透過一個按鍵建置所需要的設備與服務，不需手動設定底層參數，能快速建置的所需服務與功能。
5. 較低廉的價格：我們發現 GCP 所提供的功能服務在價格上較其他平台來的低，較能節省成本支出。

另一個值得補充的比較優勢是 GCP 相對於亞馬遜服務（AWS）在資源使用上的彈性和便利性：

6. 彈性的資源使用：GCP 提供了彈性的資源使用方案，使用者可以根據實際需求彈性調整資源的使用量與規模。使用者可以根據負載情況增加或減少雲端所使用的資源量，而不需要事先申請和預配置資源。

相比之下，GCP 有較高的資源使用方案，GCP 在資源使用上更加靈活和便捷，無需像 AWS 限制部份資源，需要使用者提交申請並等待客服回應開通更多所需雲端資源。

2.2 Kubernetes

Kubernetes (K8s) 是一個由 Google 設計的可移植、可擴展的開源容器化（containerized）應用程式管理系統，用於自動化部署、擴展和管理容器化應用程序。它提供了「跨主機叢集的自動部署、擴充以及執行應用程式容器的平台」，並支援一系列容器工具，包括 Docker 等，K8s 還提供了強大的調節和管理功能，可以有效地管理多個容器應用程式，實現高可用性和可靠性。

Kubernetes 的可移植性表示可以在不同的環境中執行，這使得應用程式能夠在不同的基礎架構上執行，具有強大的靈活性和可擴展性。

現在有許多服務背後都是採用分散式系統的架構，而 Kubernetes 就是現在許多服務背後採用的分散式系統架構，它使用容器來封裝和隔離微服務，並提供統一的管理和調節機制。

所以一個微服務通常會用一個容器包裝起來，若有很多微服務就會需要使用到可以一次管理多個容器（Container）或微服務的工具，Kubernetes 就是用於自動部署、擴充和管理「容器化應用程式」的開源系統，可自動部署、管理和擴充應用程式，Kubernetes 讓應用程式的部署和擴展變得自動化和可靠，運用上能夠藉由 Kubernetes 來管理多個容器（Container）。

此外，Kubernetes 也可自動管理服務探索、納入負載平衡、追蹤資源配置，以及根據計算使用率進行調整，還可檢查個別 POD 的健康狀態，並透過自動重啟，使應用程式能自我修復進行回滾（roll back）機制。

若使用早期傳統方式來架設此 CTFd 解題平台[6]，會需要使用人工方式先行將平台與每一個題目都啟動，耗費非常多時間，這時將會長期占用系統資源，且部署不易，一旦有狀況就必須人工介入處理，拖延比賽時間。

本專題採用自動部署的方式在 Kubernetes 上執行，僅需使用者點一個按鈕，題目、平台就會由硬碟中被拉起，直到正常啟動完成為止，每一個使用者將會擁有一個獨立的題目容器（Container）空間，且每一個相同的題目標誌（flag）都不相同，使用傳統配置與動態配置的差別比較如下：

	傳統靜態部署方式	動態部署方式
部署時間	長	短
人工處理	需要	不用
部署難度	難	簡易
資源浪費	高	低
技術成分	高	低

表2 傳統配置與動態配置差異表2

2.3 Container

dotCloud 公司在 2013 年時推出雲端容器技術 Docker，Docker 的主要目標是實作出輕量級的作業系統層虛擬化解決方案，在雲端容器技術 Docker 中，是利用 docker image 來建立一個容器（Container），一個 docker image 可以重覆創建多個容器（Container）實例。

容器（Container）提供了能將應用程式或服務從原先執行的環境中獨立抽取出來的技術，這種技術能夠輕鬆的、一致的在各種不同環境或平台中進行部署、執行，管理者可將注意力集中到部署與管理上，而不用擔心特定軟體版本對應的特定環境或設定上的細節。

使用容器（Container）化技術幾乎能在任意環境執行，例如在 Linux、Windows 或 Mac 作業系統上，這種技術被稱之為「微服務」，此類技術被廣泛應用於 Docker 映像檔格式，藉此強化了可攜性。

本專題將採用容器化技術，將 CTFd [6]與每道資安題目（挑戰）包裝成一個容器（Container），部署在 Kubernetes 上，透過此技術能達到以下幾個優點：快速部署、隔離性、可攜性、資源利用率和彈性擴展等。

值得探討的是，容器化技術所面臨的挑戰，如安全性、網路通訊等問題，駭客是否會從一個容器（Container）中找到漏洞對外或對系統內部進行攻擊，簡單且粗暴的解決方案就是不要使用 root 權限啟動一個容器（Container），並且在容器（Container）中不要安裝或允許任何其他不使用的相關套件和指令，以確保容器化環境的穩定性和安全性。

此外，Docker Hub 網站提供了開發者上傳或下載 docker image 的使用空間，可以將開發項目上傳至 Docker Hub 網站供人下載，也可以設置為私人倉庫，利於管理開發版本。

2.4 CTFd (Capture The Flag Framework)

CTFd 作為一個開源的 Capture The Flag 框架[6]，用於舉辦和管理 CTF 比賽，參賽者需要運用自己所會的資安技能來解決一系列與資安議題相關的題目(挑戰)，以此來獲取旗幟 (flags)，旗幟 (flags) 即是所謂的答案 (Answer)，並在比賽中競爭獲得分數。競賽主題大致分為以下幾個類型：PWN (漏洞利用)、Reverse (逆向工程)、Web (網絡應用)、Crypto (加密學)、Network (網絡安全)、Misc (綜合題目) 等等。CTFd 提供了一個用戶界面和豐富的管理功能，包括題目管理、用戶註冊和登錄、分數追蹤等，並且可以使用自定義套件和主題完善平台功能。我們將在我們的研究中使用 CTFd 作為基礎平台，並將其部署到 Kubernetes 上。CTFd 示意圖如下：

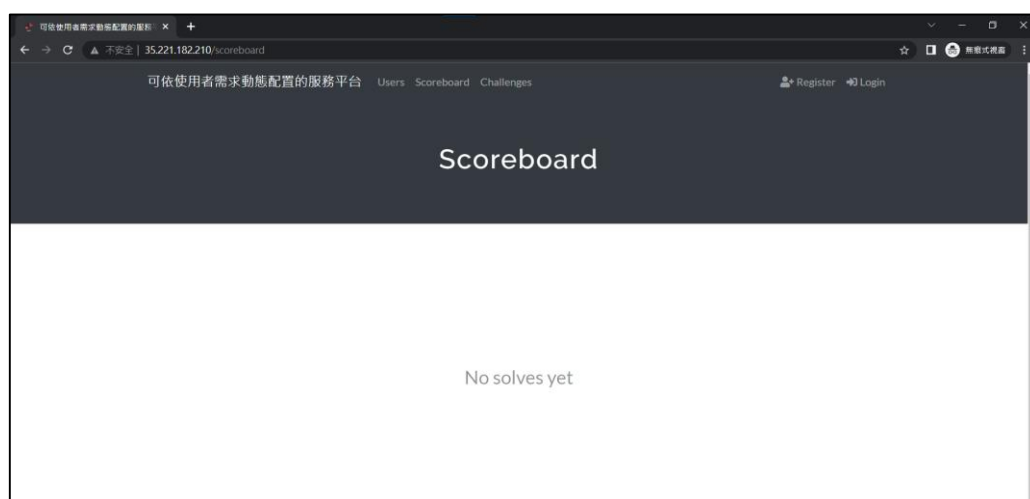


圖1 CTFd 網站示意圖

CTFd 是採用 Python 語言開發撰寫的，並且搭配了 Flask 作為網頁開發框架。使用 Python 語言撰寫具備簡潔、易讀和易於維護的特點。Flask 網頁開發框架提供了簡單而強大的功能，用於處理 HTTP 請求、路由管理等，使用 Flask 可以使開發過程更加高效和容易維護。

在我們的研究中，我們選擇使用 CTFd 作為基礎平台，並將其部署到 Kubernetes 上。這樣做的目的是為了利用 Kubernetes 的自動化部署機制，提供更靈活、可擴展和高效的 CTFd 平台。

在 Kubernetes 上部署 CTFd 的重點如下：

1. 自動擴展：Kubernetes 的擴展性使得我們可以根據使用者人數和資源需求自動調節 CTFd 容器的數量。這樣，我們可以確保使用期間提供穩定的性能，同時節省資源成本。
2. 容器化管理：我們將 CTFd 框架容器化，以便我們可以更容易地部署、更新和管理整個平台。Kubernetes 提供了強大的容器管理功能，例如：滾動更新、容錯恢復和自我修復等，這能使 CTFd 平台在運作時更加穩定且可靠。
3. 可監控性：Kubernetes 提供了豐富的監控和日誌記錄功能，使我們能夠更好地追蹤和分析 CTFd 平台的執行狀態，從而進行故障排查和性能優化。

2.5 CTFd-owl

CTFd-owl 是一個開源的 CTFd 框架擴充套件[11]，旨在實現 Docker compose 指令動態配置題目的能力。它提供了自動化的題目配置和管理功能，使得用戶可以根據需要時提供動態啟動和關閉題目的能力。

在我們的專題中，我們將借鑒 CTFd-owl 的設計思想和方法，並將其整合到我們在 Kubernetes 上部署的 CTFd 平台中，我們將使用 Kubernetes 的自動部署特性和工具來實現自動化的題目配置和管理功能，而不是使用原先的 Docker Compose 指令。

在一個 Kubernetes 集群中，每個節點（Node）上都有一個 kubelet 行程（Process），這個行程（Process）會與 Kubernetes API Server 進行通訊，向 Kubernetes API Server 發送請求，並接收 Kubernetes API Server 的回應，我們可以使用 Kubernetes API 更方便地動態管理和配置 CTFd 的題目。

2.6 Prometheus

在 Kubernetes 上部署 CTFd 開源平台後，需要有一個能夠監控系統在需要時發出警報的功能，為了實現上述功能我們所選擇的是 Prometheus 這個開

源的系統監控和警報的工具包，Prometheus 是一整套監控系統，包括資料的收集、資料儲存、警報，它提供了能夠靈活使用的查詢語言以及一個本地的時間序列資料庫，能夠協助我們達成儲存和查詢大量的時間序列數據的目的。

此外，它也能依照我們的需求設定指標來進行警報，並且還提供了許多方式來通知警報，包括電子郵件、Slack 和 Line 等，而與其它的監控套件例如 Zabbix 比較起來，Prometheus 和 Zabbix 都是開源的監控工具，Zabbix 更適合於物理機/虛擬機的監控，而 **Prometheus 更適合容器的監控**，且 Prometheus 只有一個核心 server 組件，一條命令便可以啟動，相比而言，其他系統配置相對麻煩，並且 Prometheus 在開源社群上也十分活躍。系統每隔一兩週就會有一個小版本的更新，且 Prometheus 與 Kubernetes 原生就能夠順利的協作，因此我們能夠有足夠多的技術支援與能夠確保 Kubernetes 上的順利運作，綜上所述，我們選擇使用 Prometheus 來實現監控功能。

2.7 Line Notify

Line Notify 是 Line 平台的通知服務，是一個簡易型的通知系統不需要特別去創建一個機器人，並且也支援文字、圖片、影片、聲音等等的訊息，只需要使用 API 和 Web 介面就可以輕鬆的管理及發送訊息，也可以透過第三方程式來進行連動像是 Github、IFTTT、Mackerel 等等的程式，來把可以傳送的訊息變得更加豐富。

我們之所以選用 Line Notify 而沒有使用 Line 的 Bot 是因為我們可能只需要簡單的文字跟圖片提醒也不用花太多時間去撰寫程式碼，所以我們後來決定使用 Line Notify 的簡易型通知系統，只需要呼叫 API 即可傳輸文字或是圖片，現今的監控或是通知軟體也開始支援 Line Notify 的 API，讓使用者們只要輸入金鑰，就可以呼叫 Line Notify 來當作一個報警系統。

第三章 專題系統架構

3.1 系統核心架構

本系統的主要建構於 GCP 的 Kubernetes 上，CTFd 伺服器、資料庫、資料庫管理系統以及所有挑戰題目也都建於 Kubernetes 中，分別獨立由不同 POD 進行管理。

本系統主要由一個 External IP 進行對外服務，使用者或管理者可由此 IP 連線至 CTFd 伺服器，我們將每一個微服務都獨立一個 Pod 進行管理，CTFd 伺服器、DB server、phpadmin 以及每一道 Challenge 都獨立一個 Pod 來進行管控，其中 phpadmin 是為方便管理者管理資料庫所建置的。

此外，管理員可透過 GCP cloud shell 進行控制。核心架構部署圖如下：

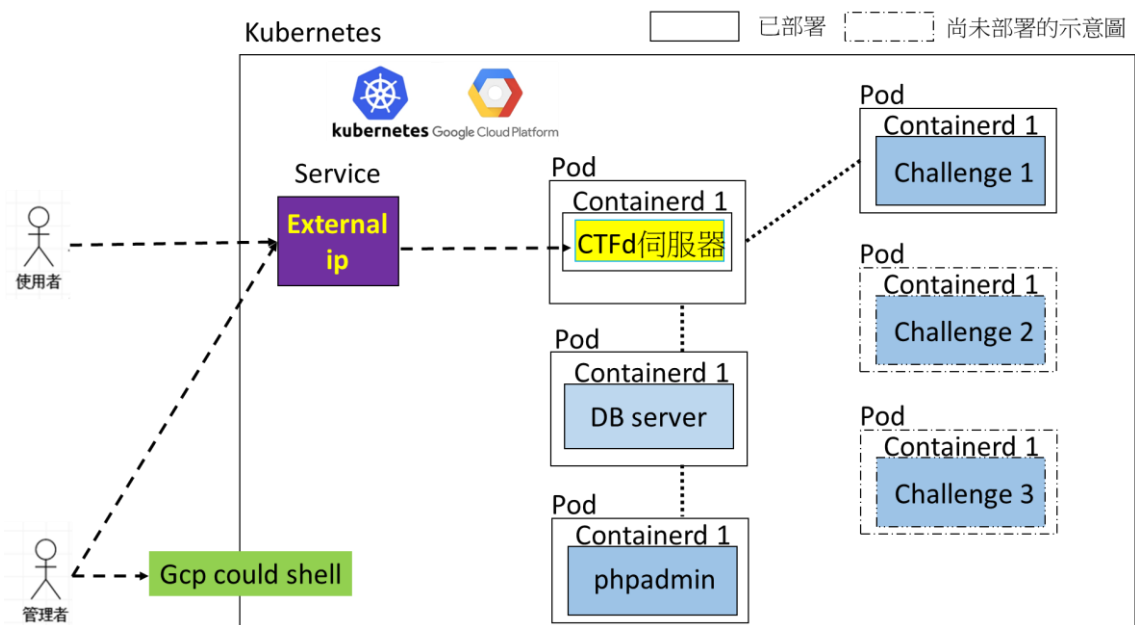


圖2 系統核心架構部署圖 (Deployment Diagram)

主要的作業都係透過網頁的方式來提供服務，服務平台（CTFd 伺服器）的型態分為一般使用端以及管理端，二者皆須透過瀏覽器，並透過 TCP 的方式與伺服器連線。

3.2 監控端架構

監控端系統如圖3所示，系統主要提供「管理者」一個Prometheus開源系統監控Kubernetes目前狀態（如Nodes、Pods、CPU usage…）的網頁介面，讓管理者不用還要透過連線至Kubernetes透過kubectl等指令的方式來檢查集群的所有狀態，並且當問題或異常產生時告警元件將會觸發並透過網路發送請求的方式來傳送告警訊息至Line Notify。

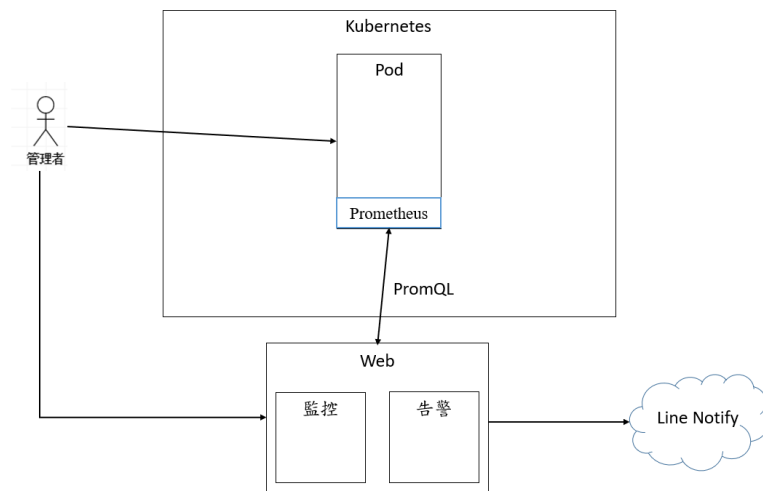


圖3 監控端架構圖

3.3 題目動態部署架構

CTFd 伺服器的容器 (Container) 中已包含 Kubernetes API 程式碼，當用戶成功登入 CTFd 解題平台觸發 API 時 (按下啟動題目按鈕，下面架構圖中的第三點)，CTFd 伺服器所在的 Pod 將與 Kubernetes API Server 進行互動 (下面架構圖中的第四點)，以部署新的 Challenge Pod (下面架構圖中的第五點)，即包含新挑戰題目的容器 (Container)，這樣使用者就能通過 CTFd 伺服器和 Pod 之間的 Service 訪問 Challenge Pod 的服務了。

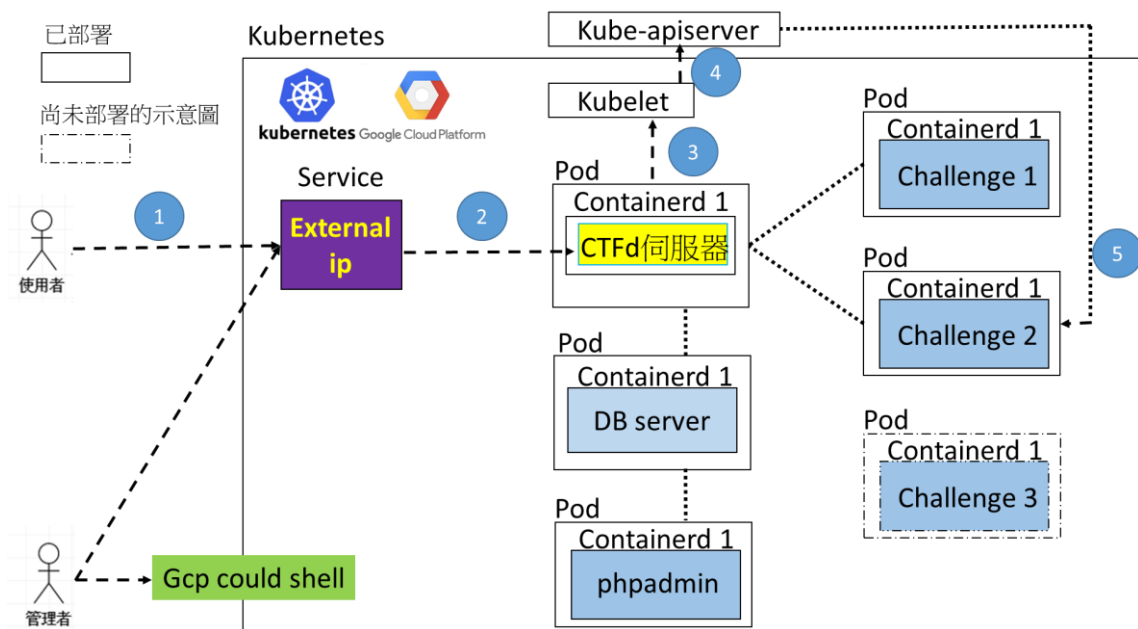


圖4 題目動態部署架構圖

此架構主要提供與自動部署題目的解決方案，當使用者依自身需求選擇題目後，k8s 系統將依使用者所選取之挑戰題目自動建立一個相應的 CTF 題目。本作業需要管理員事先先設定好挑戰題目類型的相應 Docker Image，以利自動部署該類型題目。

第四章 系統實現

4.1 關鍵核心技術

4.1.1 CTFd 部署技術

在此章節中，我們將逐一介紹每個部署細節，分別是：

1. **StorageClass 和 PVC 配置**：我們需要定義 StorageClass，這是在定義儲存設備和訪問模型。接著需要配置 PersistentVolumeClaim (PVC)，PVC 則是應用程式對儲存的需求描述，包括容量大小和訪問模式等等。
2. **Secret 配置**：有時需要儲存敏感資料，例如通用的帳號與密碼，為了安全地儲存這些數據，我們使用 Kubernetes 的 Secret 功能，此功能可以以 Base64 加密[13]的方式儲存這些敏感數據，並且只有擁有權限訪問的容器才能讀取。
3. **MySQL 配置**：我們需要配置一個儲存資料的數據庫，用來儲存使用者資訊、解題資料、解題分數、使用者帳號密碼等等。
4. **phpMyAdmin 配置**：我們選擇使用 phpMyAdmin 配置一個網頁微服務，讓管理者好進行資料庫的管理，phpMyAdmin 提供了良好且簡單易懂的 UI，方便管理者操作。
5. **CTFd 解題平台配置**：我們最重要的目的是在 Kubernetes 上配置 CTFd 解題平台，在這個小節中將介紹部署配置的詳細細節。

一、Kubernetes 中的 StorageClass 和 PVC 配置

1. StorageClass

StorageClass 是 Kubernetes 中定義儲存類型的方式之一。它允許我們定義不同的儲存設置，如儲存提供商、儲存類型和回收策略等。以下是一個 StorageClass 配置：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: sc-retain
provisioner: kubernetes.io/gce-pd
parameters:
  type: pd-standard
  fstype: ext4
reclaimPolicy: Retain
volumeBindingMode: Immediate
```

表3 StorageClass 配置文件

在上述配置文件中，我們創建了一個名為 "sc-retain" 的 StorageClass。這個 StorageClass 使用了 kubernetes.io/gce-pd 作為提供商，表示它將使用 Google Compute Engine Persistent Disk（GCE PD）來提供持久化儲存。

parameters 部分允許我們指定其他配置參數。在此示例中，我們設置了 type 為 pd-standard，表示使用標準的 GCE PD 類型。fstype 被設置為 ext4，指定了文件系統類型為 ext4。

reclaimPolicy 屬性指定了當 PVC 與 PV 解除綁定時的回收策略。在此示例中，我們設置為 Retain，這意味著 PV 的數據將保留，並不會自動刪除。

volumeBindingMode 屬性指定了 PV 的綁定模式。在此示例中，我們設置為 Immediate，表示 PVC 將立即綁定到可用的 PV，而不需要等待。

2. PVC (PersistentVolumeClaim)

PVC 是 Kubernetes 中用於申請持久化儲存的對象。Kubernetes 根據這個配置文件自動分配一個 PV (PersistentVolume)。以下是一個 Database 的 PVC 配置：

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: db-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 50Gi
  storageClassName: sc-retain
```

表4 db-pvc 配置文件

metadata 部分指定了 PVC 的元數據，其中 name 為 "db-pvc"。

spec 部分是 PVC 的具體配置。accessModes 定義了 PVC 的存取模式，這裡設置為 "ReadWriteOnce"，表示該 PVC 只能以單讀寫方式進行存取。

resources 部分定義了 PVC 的資源需求。在這個配置文件中，我們設置了 storage 為 "50Gi"，表示這個 PVC 需要至少 50GB 的儲存空間。

storageClassName 指定了 PVC 所使用的 StorageClass 的名稱，這裡設置為 "sc-retain"，以便根據該 StorageClass 申請對應的 PV。

通過以上配置，當 Kubernetes 接收到 "db-pvc" 的請求後，它將根據 "sc-retain" 的 StorageClass 自動分配一個 PV，並將其與 PVC 進行綁定，從而為應用程序提供所需的持久化儲存空間。

接下來配置 CTFd 解題平台寫 log 用的 PVC：

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  creationTimestamp: null
  labels:
    ctf: ctf-pv
    app: ctf-pv-logs
  name: ctf-pv-logs
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 100Mi
status: { }
```

表5 CTFd 平台 log 用的 PVC 配置文件

metadata 指定了 PersistentVolumeClaim (PVC) 的元數據，其中名稱為 "ctf-pv-logs"。這個 PVC 的目的是為了提供 CTFd 平台寫 log 的持久化儲存空間。

spec 部分是 PVC 的具體配置。存取模式 (accessModes) 被設置為 "ReadWriteOnce"，表示該 PVC 只能以單一讀寫模式進行存取。

資源需求 (resources) 部分定義了 PVC 所需的儲存資源。在這個配置文件中，我們設置了儲存空間 (storage) 為 "100Mi"，表示這個 PVC 需要至少 100MiB 的儲存空間。

該配置文件中並未指定 `storageClassName`，表示未指定特定的 `StorageClass`。這意味著 Kubernetes 將使用預設的 `StorageClass` 來滿足這個 PVC 的需求。

通過以上配置，當 Kubernetes 接收到 "ctf-pv-logs" 的請求後，它將自動分配一個符合需求的 `PersistentVolume` (持久化卷)，並將其與 PVC 進行綁定，以提供所需的持久化儲存空間。這樣，應用程式就能夠使用該 PVC 來保存並存取日誌資訊等重要數據。

接下來配置 CTFd 解題平台用來存放上傳物件的 PVC：

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  creationTimestamp: null
  labels:
    ctf: ctf-pv
    app: ctf-pv-uploads
  name: ctf-pv-uploads
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 500Mi
status: { }
```

表6 CTFd 平台存放上傳物件用的 PVC 配置文件

`metadata` 部分指定了 `PersistentVolumeClaim` (PVC) 的元數據，其中名稱為 "ctfd-pv-uploads"。這個 PVC 的目的是為了提供用來存放上傳物件的持久式儲存空間。

spec 部分是 PVC 的具體配置。存取模式 (accessModes) 被設置為 "ReadWriteOnce"，表示該 PVC 只能以單一讀寫模式進行存取。

資源需求 (resources) 部分定義了 PVC 所需的儲存資源。在這個配置文件中，我們設置了儲存空間 (storage) 為 "500Mi"，表示這個 PVC 需要至少 500MiB 的儲存空間。

該配置文件中並未指定 storageClassName，表示未指定特定的 StorageClass。這意味著 Kubernetes 將使用預設的 StorageClass 來滿足這個 PVC 的需求。

通過以上配置，當 Kubernetes 接收到 "ctfd-pv-uploads" 的請求後，它將自動分配一個符合需求的 PersistentVolume (持久化卷)，並將其與 PVC 進行綁定，以提供所需的持久化儲存空間。這樣，應用程式就能夠使用該 PVC 來保存並存取上傳的檔案或其他相關資源。

二、Kubernetes 中的 Secret 配置

1. 配置 Secret

Secret 在 Kubernetes 中用於儲存敏感的資訊，例如密碼、金鑰等。以下是 Secret 的配置：

```
apiVersion: v1
kind: Secret
metadata:
  name: db-secret
type: Opaque
data:
  root: <base64 here>
  root-password: <base64 here>
```

表7 K8S Secret 配置文件

metadata 部分指定了 Secret 的元數據，其中 name 為 "db-secret"。

type 屬性指定了 Secret 的類型，這裡設置為 "Opaque"，表示這是一個通用的 Secret，沒有特定的數據格式要求。

data 部分定義了 Secret 中儲存的數據。在這個配置文件中，我們有兩個鍵值，分別是 root、root-password。root 是一個 Base64 編碼後的值，同樣地，root-password 也是一個 Base64 編碼後的值。

三、Kubernetes 中的 MySQL 與 phpMyAdmin 配置

1. 配置 MySQL

```
kind: StatefulSet
metadata:
  name: mysql-db
spec:
  selector:
    matchLabels:
      db: ctfd
  serviceName: "ctfd-db"
  replicas: 1
  template:
    metadata:
      labels:
        db: ctfd
    spec:
      containers:
        - name: mysql
          image: mysql:8.0.21
          ports:
            - containerPort: 3306
          volumeMounts:
            - name: db-pvc-volume
              mountPath: /var/lib/mysql
          env:
            - name: MYSQL_ROOT_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: mysql-pass
                  key: password
      volumes:
        - name: db-pvc-volume
          persistentVolumeClaim:
            claimName: db-pvc
```

表8 MySQL 配置文件

metadata 部分指定了 StatefulSet 物件的元數據，名稱為 "mysql-db"。這個 StatefulSet 的目的是管理一個 MySQL 數據庫。

spec 部分定義了 StatefulSet 的具體配置。selector 用於選擇一個符合標籤 db: ctfdb 的 Pod。serviceName 被設置為 "ctfd-db"，用於定義 StatefulSet 所關聯的服務名稱。

replicas 被設置為 1，表示只需要一個 Pod 副本。

template 部分定義了 Pod 的模板，包含了 Pod 的元數據和規格。

在 Pod 規格中，terminationGracePeriodSeconds 被設置為 10，表示終止該 Pod 前的等待時間為 10 秒。

containers 部分定義了 Pod 中執行的容器。這裡使用名為 mysql 的容器，使用 mysql:8.0.21 Image。容器使用 3306 端口作為 MySQL 數據庫的通訊端口。

volumeMounts 定義了容器中的儲存掛載。在這個配置文件中，我們將一個名為 db-pvc-volume 的持久化卷掛載到容器的 /var/lib/mysql 目錄，用於持久化儲存 MySQL 數據庫文件。

env 部分定義了容器的環境變數。在這裡，我們定義了一個名為 MYSQL_ROOT_PASSWORD 的環境變數，它的值來自於名為 mysql-pass 的密碼型態的 Secret 的 password 鍵。

volumes 部分定義了 Pod 中的儲存卷。我們定義了一個名為 db-pvc-volume 的持久化卷，並聲明它依賴於名為 db-pvc 的 PersistentVolumeClaim (PVC)。

通過以上配置，StatefulSet 將根據模板創建一個帶有 MySQL 數據庫的 Pod，並將持久化卷 db-pvc-volume 掛載到容器中的 /var/lib/mysql 目錄，從而實現數據的持久化儲存和使用密碼保護的 MySQL 數據庫。

2. 配置用於 MySQL 的 Service

```
apiVersion: v1
kind: Service
metadata:
  name: db-service
  labels:
    app: ctfd
spec:
  ports:
    - port: 3306
      name: mysql-db-port
  clusterIP: None
  selector:
    db: ctfd
```

表9 MySQL 的 Service 配置文件

這個配置文件定義了一個名為 "db-service" 的 Service。

Service 是 Kubernetes 中提供對內部或外部訪問的功能。

在 metadata 部分，我們指定了 Service 的名稱為 "db-service"，並添加了一個標籤 "app: ctfd"。

在 spec 部分，我們定義了 Service 的端口配置。這裡設置了一個名為 "mysql-db-port" 的端口，對應的容器端口是 3306。

clusterIP 被設置為 "None"，將不會分配 Cluster IP，它只能在集群內部通過 Pod 的 IP 地址進行訪問。

Selector 部分定義了用於選擇目標 Pod 的標籤。這裡的選擇器是 "db: ctfd"，它將選擇帶有 "db: ctfd" 標籤的 Pod 作為 Service 的目標。

透過以上配置，我們創建了一個名為 "db-service" 的 Service，它通過選擇器將流量導向到具有標籤 "db: ctfd" 的 Pod，並將流量導向該 Pod 的 3306 端口，從而實現對該 Pod 內的 MySQL 數據庫的訪問。

3. 配置 phpMyAdmin

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: phpmyadmin
  labels:
    app: phpmyadmin
spec:
  replicas: 1
  selector:
    matchLabels:
      app: phpmyadmin
  template:
    metadata:
      labels:
        app: phpmyadmin
    spec:
      containers:
        - name: phpmyadmin
          image: phpmyadmin
          ports:
            - containerPort: 80
          env:
            - name: PMA_HOST
              value: "db-service"
```

表10 phpMyAdmin 的配置文件

metadata 指定了 Deployment 的名稱為 "phpmyadmin"，並添加了一個標籤 "app: phpmyadmin"。

spec 我們設置了副本數為 1，表示只執行一個 Pod 副本。

selector 定義了用於選擇一個目標 Pod 的標籤。這裡使用的 selector 是："app: phpmyadmin"，選擇具有相同標籤的 Pod 作為 Deployment 的目標。

template 我們定義了 Pod 的模板。模板中的 metadata 部分添加了標籤 "app: phpmyadmin"。

在 spec 的 containers 部分，我們定義了一個名為 "phpmyadmin" 的容器，使用了 phpmyadmin Image。容器使用了 80 端口。

容器中的環境變數設置了一個名為 "PMA_HOST" 的變數，其值為 "db-service"，這表示 phpmyadmin 將使用名為 "db-service" 的 Service 來連接到 MySQL 數據庫。

透過以上配置，Deployment 將創建一個名為 "phpmyadmin" 的 Pod，並將該 Pod 的流量導向 80 端口。

這個部署後的 Pod 將使用環境變數 "PMA_HOST" 中指定的 "db-service" Service 作為連接 MySQL 數據庫的主機。這樣可以在 Kubernetes 集群中執行 phpMyAdmin 網頁微服務，方便管理者用於管理 MySQL 數據庫。

四、Kubernetes 中的 CTFd 解题平台配置

1. 配置 CTFd 解题平台

```
containers:
- env:
  - name: ACCESS_LOG
    value: '-'
  - name: DATABASE_URL
    value: mysql+pymysql://root:<password here>@db-service/ctfd2
  - name: ERROR_LOG
    value: '-'
  - name: LOG_FOLDER
    value: /var/log/CTFd
  - name: REDIS_URL
    value: redis://ctfd-redis-cache:6379
  - name: REVERSE_PROXY
    value: "true"
  - name: UPLOAD_FOLDER
    value: /var/uploads
  - name: WORKERS
    value: "1"
  - name: PROBLEM_DOCKER_RUN_FOLDER
    value: /home/docker
image: <your name here>/<image name here>:<image version here>
imagePullPolicy: ""
name: ctf
resources: {}
volumeMounts:
- mountPath: /var/log/CTFd
  name: ctf-pv-logs
- mountPath: /var/uploads
  name: ctf-pv-uploads
- mountPath: /var/run/docker.sock
  name: docker-sock-volume
```

表11 CTFd 解题平台 Containers 的配置文件

DATABASE_URL：指定 CTFd 使用的數據庫連接 URL。這裡使用的是 MySQL，並提供了用戶名、密碼、數據庫主機和端口等相關資訊。

UPLOAD_FOLDER：指定上傳文件的儲存路徑。這裡將上傳的文件儲存在容器的 /var/uploads 路徑下。

LOG_FOLDER：指定日誌文件的儲存路徑。這裡將日誌文件儲存在容器的 /var/log/CTFd 路徑下。

PROBLEM_DOCKER_RUN_FOLDER：指定問題 Docker image 執行時的文件夾路徑。這裡將問題 Docker 鏡像執行時的文件夾路徑設置為 /home/docker。

WORKERS：指定 CTFd 解題平台使用的工作行程（Process）數量。這裡預設設置為 1。

image：指定要使用的 CTFd image 的名稱和版本。可以自定義映像名稱和版本，例如 ctfd/ctfd:latest。

volumeMounts：指定容器與 Persistent Volume Claim (PVC) 之間的掛載關係。這裡包括將 PVC 中的日誌文件掛載到容器的 /var/log/CTFd 路徑下，將 PVC 中的上傳文件掛載到容器的 /var/uploads 路徑下，以及將主機의 Docker socket 掛載到容器的 /var/run/docker.sock 路徑下，以便容器可以與主機의 Docker 進行溝通。

接著設定在 Kubernetes 中的 Security Context，用於定義容器執行時的安全屬性和限制。

<pre>securityContext: runAsUser: 0</pre>

表12 CTFd 解題平台 SecurityContext 的配置文件

runAsUser 屬性用於指定容器執行時的使用者身份。我們在這裡設置為 0 容器將以 root 使用者身份執行。

接著配置 CTFd 的重啟策略和 CTFd 所使用到的 volumes：

```
restartPolicy: Always
serviceAccountName: ""
automountServiceAccountToken: true
volumes:
- name: ctf-pv-logs
  persistentVolumeClaim:
    claimName: ctf-pv-logs
- name: ctf-d-pv-uploads
  persistentVolumeClaim:
    claimName: ctf-d-pv-uploads
- name: docker-sock-volume
  hostPath:
    # location on host
    path: /var/run/docker.sock
    # this field is optional
    type: Socket
- name: homedocker
  hostPath:
    path: /home/docker
```

表13 CTFd 解題平台 volumes 的配置文件

`restartPolicy: Always`: 此屬性指定容器的重啟策略為「始終重啟」。這表示如果容器在任何情況下終止，Kubernetes 將自動重新啟動該容器，以確保容器持續運行。

`serviceAccountName: ""`: 此屬性指定要使用的服務帳戶的名稱。在這裡，空字符串表示該容器將使用默認的服務帳戶。服務帳戶用於授予容器在 Kubernetes 集群中執行的操作的權限。

`automountServiceAccountToken: true`: 此屬性指定是否自動掛載服務帳戶的 token 到容器內部。在這裡，設置為 `true` 表示啟用自動掛載，容器將自動獲取和使用服務帳戶的身份驗證 token，以便與 Kubernetes API 進行溝通。

`volumes`: 這是一個包含多個卷（volumes）的清單，每個卷都用於特定的儲存位置或資源掛載到容器中。

`ctf-pv-logs` 和 `ctfd-pv-uploads`: 這些是 PVC（PersistentVolumeClaim）的名稱，它們與容器關聯，用於掛載持久卷。PVC 允許容器使用持久化的儲存空間，以保留數據的持久性。

`docker-sock-volume`: 這是一個主機路徑卷（hostPath volume），將主機上的 `/var/run/docker.sock` 文件掛載到容器內部。這樣容器就可以與主機上執行的 Docker 行程（Process）進行溝通。

`homedocker`: 將主機上的 `/home/docker` 路徑掛載到容器內部。

2. 配置用於 CTFd 解題平台的 Service

```
apiVersion: v1
kind: Service
metadata:
  creationTimestamp: null
  labels:
    app: ctf
  name: ctf
spec:
  ports:
    - name: ui
      port: 8000
      protocol: TCP
      targetPort: 8000
  selector:
    app: ctf
  type: ClusterIP
status: {}
```

表14 CTFd 解題平台 Service 的配置文件

kind: Service: 這個屬性指定了該配置文件描述的 Kubernetes 物件類型為「服務」。

metadata: 這是對服務的元數據（metadata）進行配置的部分，如創建時間戳和標籤等。

creationTimestamp: null: 此屬性表示服務的創建時間戳為空。一旦服務被創建，此時間戳將被自動填充。

labels: 此屬性定義了標籤（labels）的鍵值對，用於識別和組織服務。

name: ctf: 這個屬性指定了服務的名稱為「ctf」。這是在 Kubernetes 中唯一標識該服務的名稱。

spec: 這個部分定義了服務的規範（specification），包括網絡端口和類型等。

ports: 這是一個包含端口配置的清單，用於定義服務的網絡端口。

name: 這個屬性指定了端口的名稱為「ui」。

port: 這個屬性指定了服務的端口為 8000。

protocol: 這個屬性指定了使用的網絡協議為 TCP。

targetPort: 這個屬性指定了服務將流量轉發到容器內部的目標端口。

selector: 這個屬性指定了用於選擇要關聯到該服務的 Pod 的標籤。

app: ctfd: 這個標籤選擇器表示只有帶有標籤 app=ctfd 的 Pod 才會與此服務關聯。

type: ClusterIP: 這個屬性指定了服務的類型為 ClusterIP，能在 Kubernetes 集群內部進行內部通信，並分配服務（Service）一個集群 IP。

4.1.2 動態部署核心技術

我們借鑒 GitHub 上的開源項目 CTFd-owl[11]，它是一個開源的 CTFd 框架擴充套件，旨在實現 Docker compose 指令動態配置題目的能力。它提供了自動化的題目配置和管理功能，使得用戶可以根據需要時提供動態啟動和關閉題目的能力。

在我們的專題中，我們將借鑒 CTFd-owl 的設計思想和方法，並將其整合到我們在 Kubernetes 上部署的 CTFd 平台中，我們使用 Kubernetes 的自動部署特性和工具來實現自動化的題目配置和管理功能，而不是使用原先的 Docker Compose 指令。

在一個 Kubernetes 集群中，每個節點（Node）上都有一個 kubelet 行程（Process），這個行程（Process）會與 Kubernetes API Server 進行通訊，向 Kubernetes API Server 發送請求，並接收 Kubernetes API Server 的回應，我們可以使用 Kubernetes API 更方便地動態管理和配置 CTFd 的題目。

當用戶成功登入 CTFd 解題平台觸發所寫的 API 時（按下啟動題目按鈕），CTFd 伺服器所在的 Pod 將與 Kubernetes API Server 進行互動，以部署新的 Challenge Pod，即包含新挑戰題目的容器（Container），這樣使用者就能通過 CTFd 伺服器和 Pod 之間的 Service 訪問 Challenge Pod 的服務了。

關鍵程式碼如下：

```
from kubernetes import client

# 創建 API 客戶端對象
v1 = client.CoreV1Api()

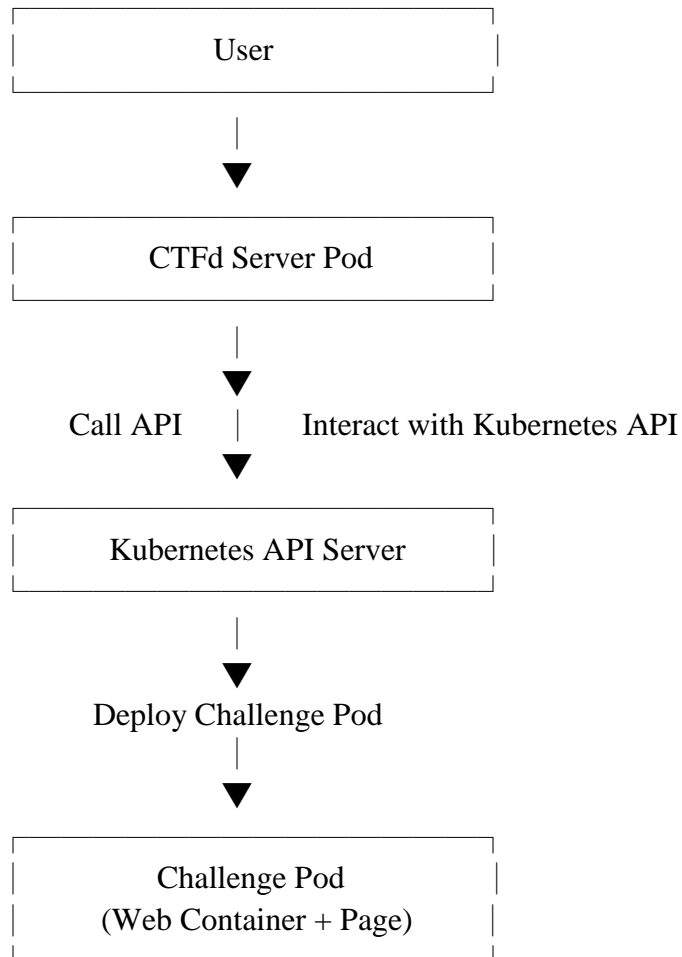
# 創建 Pod 的描述物件
pod_manifest = {
    "apiVersion": "v1",
    "kind": "Pod",
    # Pod 的其他配置...
}

# 呼叫 Kubernetes API 創建 Pod
api_response = v1.create_namespaced_pod(body=pod_manifest,
namespace="default")

print("New Pod created with name:", api_response.metadata.name)
```

表15 動態部署核心關鍵程式碼

執行流程如下：



上述執行流程描述了 CTFd Server Pod 包含了 CTFd 容器 (Container) 和 API 程式碼，當用戶觸發API時，CTFd Server Pod 將與 Kubernetes API Server 進行互動，以部署 Challenge Pod，即包含新頁面的 Web 容器 (Container)。

這麼一來，CTFd Server Pod 就能夠通過 Service 訪問 Challenge Pod 的服務，使用者能直接通過 CTFd Server 連線到 Challenge Pod 提供的服務。

4.1.3 監控與及時示警核心技術

監控端使用 Prometheus 的查詢語言 PromQL 來進行監控端的網頁製，使用 Python 編寫的輕量級網頁框架 Flask 和用來檢查是否有成功連線到 Prometheus 的 python 套件 prometheus-api-client，並且使用到 Bootstrap5.2.2 和 jQuery3.7.0。這兩個版本現階段尚未有安全漏洞，所以使用這兩個套件能較安全的進行網頁的簡單渲染和傳送請求。

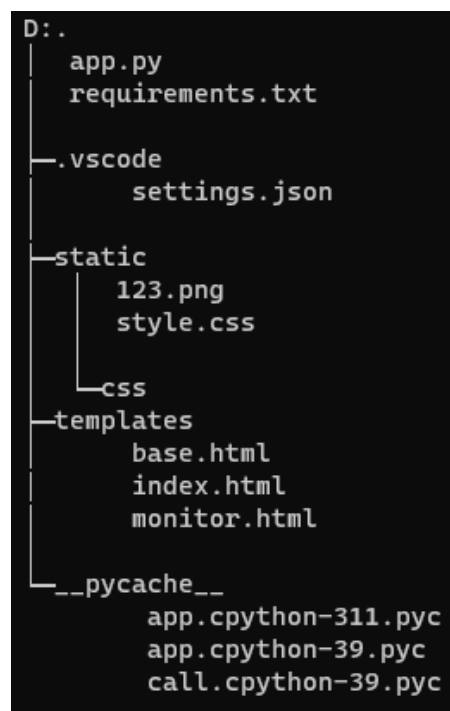


圖5 網頁樹狀圖

監控端也使用了 Jinja2 的版面繼承，讓每一個版面上的外觀都是固定的，也使用了 jQuery 的 ajax 來進行發送請求的動作，並且搭配 PromQL 的查詢語法來進行抓取資料，並將抓取到的 JSON 檔案資料進行處理，從中得到我們需要的資料，並且回傳到網頁的畫面上。程式碼如表16。

表16程式碼為網頁主版面，主要提供其他網頁頁面可以有一樣的畫面，讓網頁上方的 nav 可以使用到 Jinja2 的版面繼承套用到每一個頁面上，第五至十四行為使用 CDN 的方式將 jQuery 和 Bootstrap 新增到網頁中，讓每一個

網頁界面都可以使用到這兩個套件，程式碼中的「{% block content %}{% endblock %}」必須為空值，其他網頁才能把程式碼寫入到 block content 中。

```
<!DOCTYPE html>
<html>
<head>
  <title>監控畫面</title>
  <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.2/dist/css/bootstrap.min.css"
  integrity="sha384-
Zenh87qX5JnK2Jl0vWa8Ck2rdkQ2Bzep5IDxbcnCeuOxjzrPF/et3URy9Bv1WT
Ri" crossorigin="anonymous">
  <script src="https://code.jquery.com/jquery-3.7.0.min.js" integrity="sha256-
2PmVv0kuTBOenSvLm6bvfbSSHrUJ+3A7x6P5Ebd07/g="
crossorigin="anonymous"></script>
  <link rel="stylesheet" type="text/css"
  href="{{ url_for('static', filename='style.css') }}">
  <script type="text/javascript" src="{{ url_for('static',
filename='check.js') }}"></script>
  <meta http-equiv="Content-Type" content="text/html" ; charset="uft-8">

<body>
  <nav class="navbar navbar-expand-xl navbar-light" style="background-color:
rgb(81, 174, 236);">
    <div class="container-fluid">
      <a class="navbar-brand text-light" style="font-weight: bold;font-
family:'Inconsolata', monospace;font-size:30px;" href="/">Kubernetes
Monitor</a>
      <button class="navbar-toggler" type="button" data-bs-toggle="collapse"
```

```

        data-bs-target="#navbarSupportedContent" aria-
controls="navbarSupportedContent" aria-expanded="false"
        aria-label="Toggle navigation">
        <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarSupportedContent">
        <ul class="navbar-nav me-auto mb-2 mb-lg-0">
            <li class="nav-item">
                <a class="nav-link text-light" style="font-weight: bold;font-
family:'Inconsolata', monospace;font-size:16px;"
href="{ { url_for('check_connect') } }">Monitor</a>
            </li>
            <li class="nav-item">
                { % block nav% } { % endblock % }
            </li>
        </ul>
    </div>
</div>
</nav>
{ % block content % } { % endblock % }
</body>
</html>

```

表16 監控端網頁主模板程式碼

表17為監控端主頁面的程式碼，因為使用了Jinja2的版面繼承所以程式碼會看起來相當精簡，第一行程式碼就是將表16主模板的程式碼套入其中，「{% block content %}{% endblock content %}」包起來中的程式碼將會寫入到表16中的「{% block content %}{% endblock %}」內。

```
{% extends 'base.html' %}
{% block content %}
    <p class="title">GCP Monitor Web Site Using Prometheus PromQL</p>
    
{% endblock content %}
```

表17 監控端主頁面程式碼

表18為監控端的主要程式碼之一，利用setInterval()這個Javascript的Function來完成到每四千毫秒會執行一次請求，我們也利用了jQuery中的\$.ajax來對Prometheus發送PromQL查詢語法的請求，並且在請求設定中設定為「GET」，從Prometheus儲存的數據中抓取資料，並且設定「JSON」的檔案格式，並將抓取到的json資料利用陣列的方式將我們需要的數值抓取出來，利用parseFloat()這個Function來把我們抓到的字串轉成浮點數，並進行四捨五入讓數值不會有過多的值，資料處理完過後會把值送到我們所寫的辨別警示Function中，最後將數值利用文字的方式回傳到相對應的Id中。

```
<script>
    setInterval(function() {
        $.ajax({url:'http://104.155.194.183:9090/api/v1/
query?query=(sum(node_memory_MemTotal_bytes)-
sum(node_memory_MemAvailable_bytes))/sum(node_memory_MemTot
al_bytes)',
            type:'GET',
            dataType:'json',
            success: function(data) {
                var mem_usage =
data['data']['result']['0']['value']['1']
                //console.log(result)
                mem_usage =
parseFloat(mem_usage).toFixed(4)*100 + "%"
                check_mem(mem_usage)
                $('#node_mem_usage').text(mem_usage)
            },
            error: function(xhr, status, error) {
                $('#node_mem_usage').text(error)
                // 錯誤回傳
            }
        });
    });
</script>
```

```

    }
    });
},4000);
</script>

```

表18 監控端發送請求至Prometheus程式碼

最後，將 Line Notify 和 IFTTT 進行結合，讓我們可以使 ajax 來發送請求到 IFTTT 來呼叫 Line Notify 來傳送訊息。

我們撰寫了Javascript的Function來進行數值的判斷是否有超過設定值，將我們處理過的數據值傳道Function內，利用if else的判別方式來進行判別，如果超過設定值就會使用AJAX的發送請求，並且寫入要回傳道Line Notify中的文字，如果成功就會在console端顯示傳送成功的字樣。

寫了一個相關程式碼如表19：

```

function check_cpu(value){
    if(value >= 4.96){
        console.log(value+" is over 4.96")
        $.ajax({
            url:'https://maker.ifttt.com/trigger/Line/with/key/W-
h3F6QbZzZ5LpfYHomd2?value1=CPU Usage'+ '&value2='+ "CPU_is_over:4.96%",
            type:'GET',
            headers: {
                'Access-Control-Allow-Origin': '*'
            },
            success: function(data) {
                console.log(data.text)
            },
            error: function(xhr, status, error) {
                console.log(error)
                // 錯誤回傳
            }
        })
    }
    else{
        console.log(value+" is not over 4.96")
    }
};
function check_mem(mem_value){

```

```

        console.log(float_mem_value);
        if (float_mem_value >= 26){
            console.log(float_mem_value+" is over 27")
            $.ajax({
                url:'https://maker.ifttt.com/trigger/Line/with/key/W-
h3F6QbZzZ5LpfYHomd2?value1='+ "Memery_Us-
age"+"&value2='+ "Memery_is_Over_Using:27%",
                type:'GET',
                headers: {
                    'Access-Control-Allow-Origin': '*'
                },
                success: function(data) {
                    console.log(data.text)
                },
                error: function(xhr, status, error) {
                    console.log(error)
                    // 錯誤回傳
                }
            })
        }
    }
}

```

表19 告警端程式碼

4.2 系統成品

4.2.1. 使用者端

本專題所設計之解題平台係以現有的知名開源專案 CTFD [6]為基準打造設計，下面將分別呈現前章節所講之功能。

本系統需在 K8S（Kubernetes）系統上才能正常執行，且本系統係以 Google Cloud（GCP，Google Cloud Platform）平台為例，架設在 Google 雲端的 K8S 上，本系統執行的系統需求須具備可連線之環境，必須具備至少一個實體 IP（Public）可對外與內之網際網路連線能力。

在使用者（參賽者）連線進入網頁伺服器時，需要先選擇登入或註冊按鈕，選擇註冊按鈕需填寫相關資料後完成註冊，接著進行登入的動作。

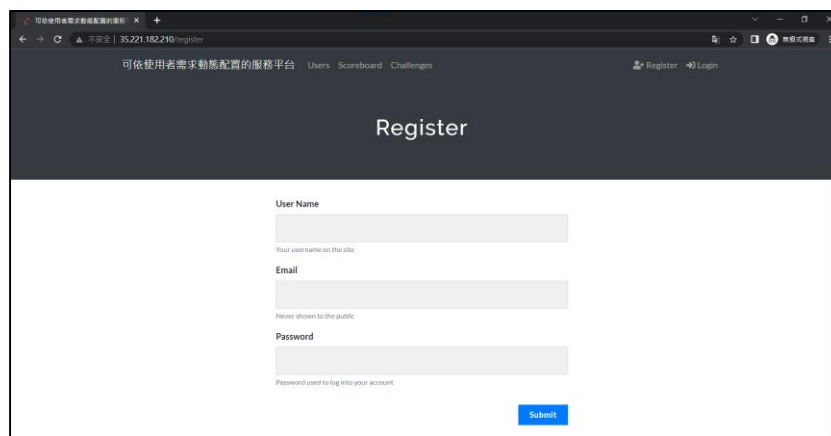
A screenshot of a web browser showing the 'Register' page of a platform. The page has a dark header with the title 'Register' in white. Below the header, there are three input fields: 'User Name' with a subtext 'Your username on the site', 'Email' with a subtext 'Never shown to the public', and 'Password' with a subtext 'Password used to log into your account'. A blue 'Submit' button is located at the bottom right of the form.

圖6 註冊介面

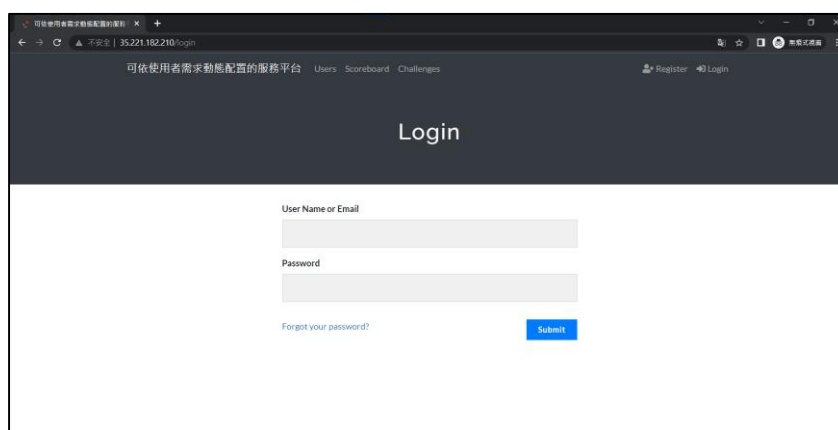
A screenshot of a web browser showing the 'Login' page of a platform. The page has a dark header with the title 'Login' in white. Below the header, there are two input fields: 'User Name or Email' and 'Password'. A blue 'Submit' button is located at the bottom right of the form. There is also a link 'Forgot your password?' below the password field.

圖7 登入介面

使用者（參賽者）登入完畢後可以依照自己學習進度選擇挑戰題目類型。

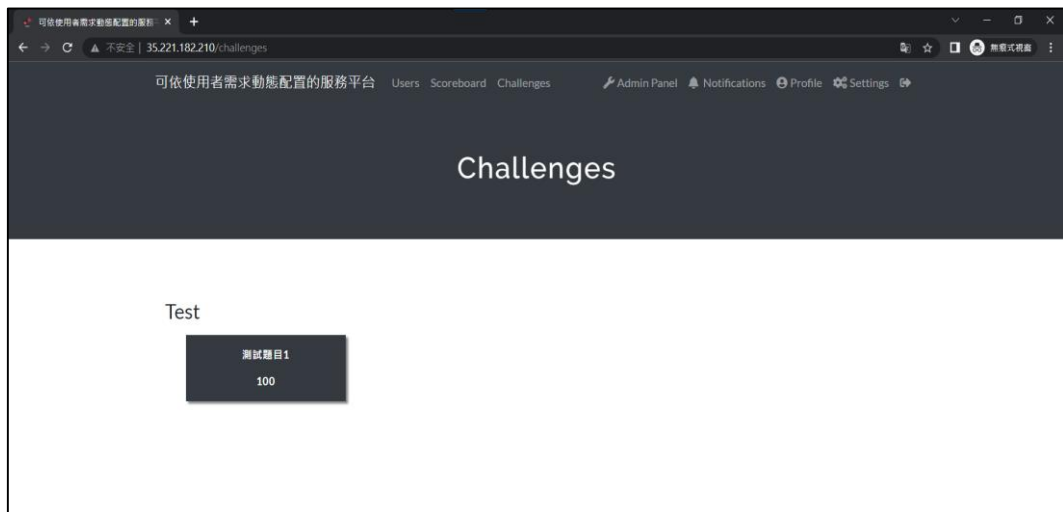


圖8 題目選擇介面

使用者（參賽者）可以在平台上選擇要解題的題目，只有在使用者選擇後要挑戰的題目後，系統才會將所選擇的題目啟動，這個過程可能需要一點時間（數十秒），等待題目啟動完畢使用者即可開始解題。

題目啟動時會在網頁上進行提示，若發生錯誤會顯示錯誤訊息供使用者聯絡管理員回報，若無人連結題目進行解題，預設五分鐘後系統將會自動釋放資源並關閉題目。

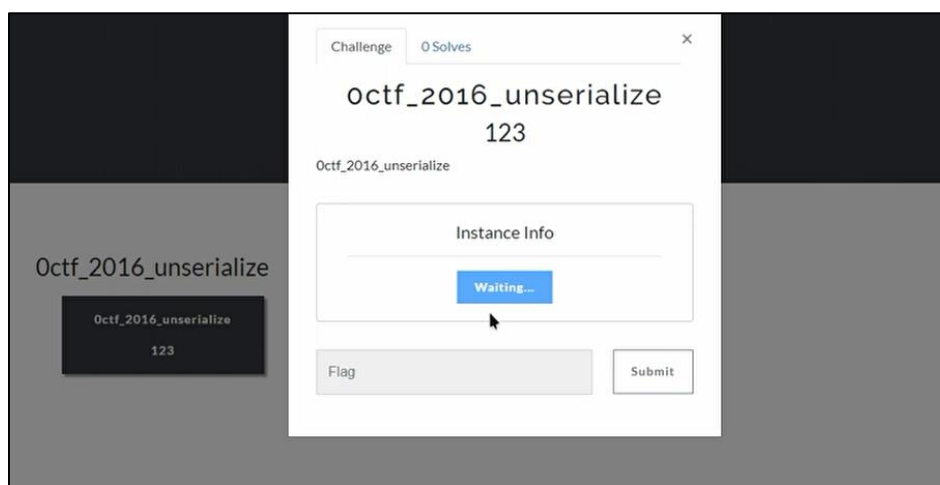


圖9 題目動態部署示意圖1

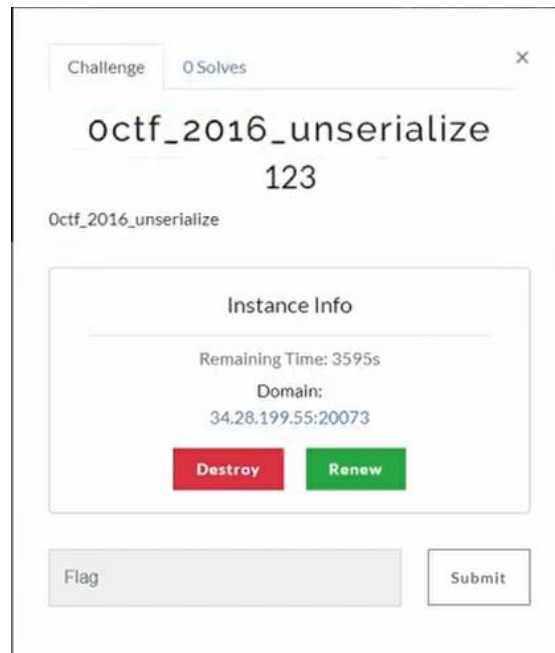


圖10 題目動態部署示意圖2

依照目前的設計來說，任何動態部署後的挑戰題目標誌（flag）都不會相同，同一道題目在不同時間下部署都會有不同的標誌（flag），這個機制用來防止讓線上的參賽者互相抄襲標誌（flag）。

在平台上方有記分板（Scoreboard）選項，所有參賽者都可以由此查看其他參賽者的解題情況、排名、分數等。

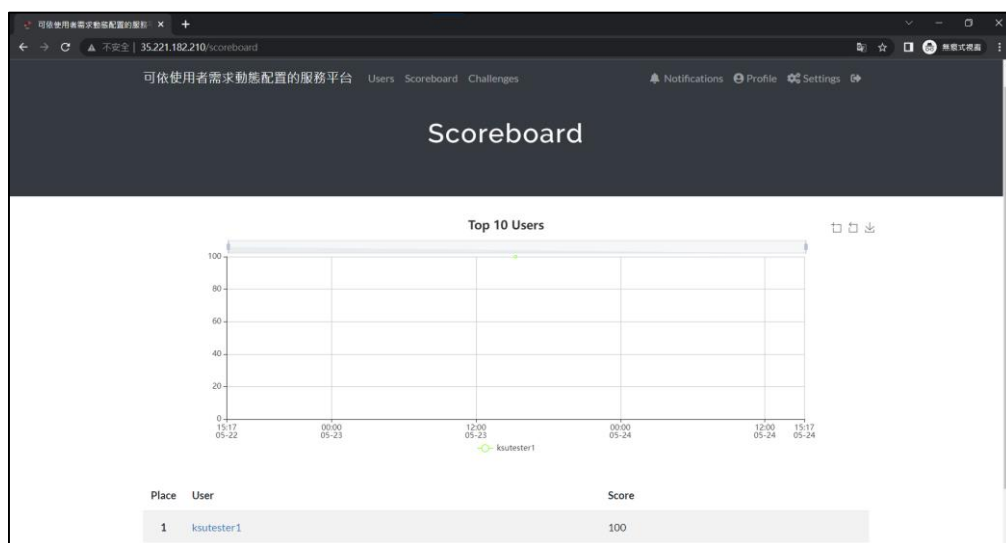


圖11 記分板（Scoreboard）

4.2.2. 系統監控端

進入系統監控頁面能查看目前平台所使用之系統資源，包含 CPU 使用率、RAM 使用率、RAM 可用剩餘數量、網路流量（上傳/下載）等等各種資訊，且能設定系統通知管理者。

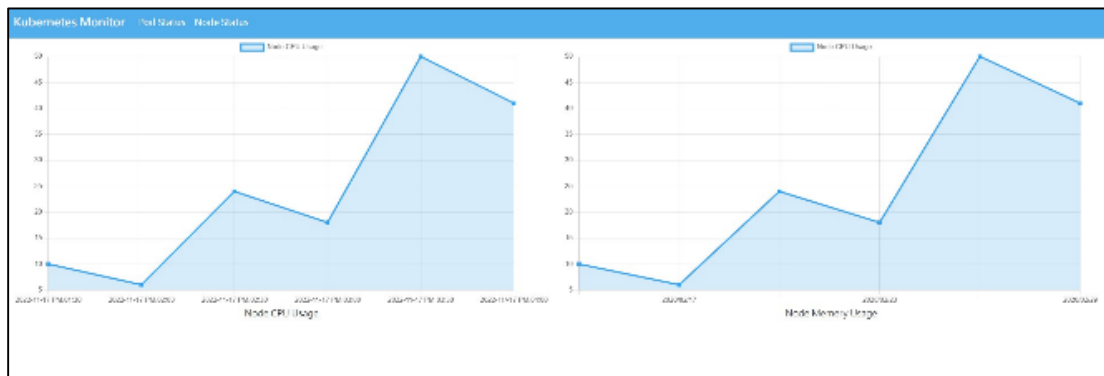


圖10 監控端圖表監控畫面1

Kubernetes Monitor Monitor						
Connect						
	Pod_Count	Node_Count	Node_CPU_Core	cluster_cpu_usage	Node_Mem	Node_Mem_Usage
Value	32	3	6	4.97%	12.07GB	27.48%

圖11 監控端數據監控畫面1

第五章 結論

在我們專題成員的努力下，借鑑了師長們的指導與建議，吸收了前人們的智慧和經驗，我們成功實現了我們原先期望的功能。我們希望這次的專題能為後續想要開發相關功能和技術的人們提供一些幫助。

關於系統未來的發展方向，目前我們的系統主要提供題目給使用者，而解題的環境，例如虛擬機，仍需要使用者自行處理。然而，我們期望能夠透過網路容器的特性和功能，直接提供一個統一設定和重置解題環境的方式給使用者，由管理單位負責。這將對教學有很大的幫助，同時減輕了老師和學生需要處理的前置作業。

總結而言，由於近年來教育開始向線上教學發展，我們的目標是提供一個方便使用者和管理者的解決方案，同時減少效能和資源浪費。我們透過整合現有技術，提供一種使線上學習更加便捷和彈性的方法。

參考文獻

中文部份

- [1]. 李文強, " Docker+Kubernetes 應用開發與快速上雲", 機械機械工業出版社
- [2]. microsoft azure, " 什麼是 Kubernetes ? ", <https://azure.microsoft.com/zh-tw/topic/what-is-kubernetes/#beyond-kubernetes>
- [3]. IBM Cloud 學習中心, "何謂 Kubernetes ? "
<https://azure.microsoft.com/zh-tw/resources/cloud-computing-dictionary/what-is-kubernetes/#overview>
- [4]. Kubernetes Examples <https://github.com/kubernetes/examples>
- [5]. Kubernetes Documentation - Service
<https://kubernetes.io/docs/concepts/services-networking/service/>
- [6]. CTFD 社群之100餘位貢獻者
<https://github.com/CTFd/CTFd/graphs/contributors> ,
<https://github.com/CTFd/CTFd>
- [7]. CTFd-Whale <https://github.com/glzjin/CTFd-Whale>
- [8]. ITHome Kubernetes 30天學習筆記
<https://ithelp.ithome.com.tw/articles/10192401>
- [9]. 【從題目中學習 k8s】 - 【Day5】K8s 中的 resource object(二) - Service
<https://ithelp.ithome.com.tw/articles/10235548?sc=hot>
- [10]. Frpc <https://github.com/fatedier/frp>
- [11]. CTFd-owl <https://github.com/BIT-NSC/CTFd-owl>
- [12]. Kubernetes <https://zh.wikipedia.org/zh-tw/Kubernetes>
- [13]. Base64 <https://zh.wikipedia.org/zh-tw/Base64>