DevOps–自動化建置應用程式環境與憑證管理

指導教授：李龍盛 老師 學生：林聖博、余貫瑒

國立嘉義大學資訊工程學系

摘要

在一般伺服器架設流程中，環境的安裝和配置通常繁瑣且容易出錯，這會使系統的復原與維護變得相對複雜。為了提升效率與減少出錯的風險，我們通過將環境容器化來架設環境並運行伺服器。但 Docker 在預設情況下並不直接支持使用 HTTPS。傳統上，常見的做法是將伺服器環境與憑證打包成一個 Docker image 還原運行。然而，這樣的方式導致憑證存放於容器內，管理和更新會變得不便。

本專題旨在解決如何在使用 Docker 運行環境的同時，能夠方便地管理憑證並確保透過 HTTPS 進行安全連線的問題。我們將運用 Docker 來建立多個不同的環境並管理網路應用程式伺服器，同時通過在 Docker 外部架設一個反向代理（Reverse Proxy），將流量從外部轉送至 Docker 容器內部。我們的解決方法是將憑證安裝在反向代理伺服器上，這樣就能夠在外部使用 HTTPS 協定進行安全連線，而無需將憑證直接打包進 Docker 容器中。

另外，我們還會設計自動化的建置腳本，使伺服器的建立與還原過程變得更加簡單。透過這種方式，無論是重新配置伺服器環境還是搭建全新的運行環境，都能夠迅速而穩定地完成，提升伺服器管理的效率與安全性，令開發人員能迅速建置與部屬環境並運行或以此專題所提供的方式做為參考自行延伸或進行自定義的設置，本專題已上傳至 GitHub：https://github.com/Spl6026/DevOps-Docker-ssl。

關鍵字：DevOps、Docker、Reverse Proxy、SSL

目錄

圖目錄

表目錄

第一章、 緒論

網路蓬勃的發展，現在對於在網路上的應用程式需求隨著科技的進步越來越大，紙本資料電子化、又或是將桌面應用程式 Web 化，這類的需求在未來只會越來越多，同時，對現在的社會而言，流行的更迭速度越來越快，並且每隔幾年對於網頁的技術就會出現一次大變革，如從以往只有靜態 HTML 演變到動態網頁、Ajax 用作網頁部分更新、為了在不同裝置能正常顯示而誕生的 RWD，也因為網頁能動態更新的緣故，Web 應用程式就出現了，之後更分成了前後端並各自發展出了許多方便開發的架構與框架可供選擇[1]，一般來說要建立一個最簡單的 Web 應用程式我們可以使用 LAMP 架構[2]，其主要建立在 Linux 系統底下，並透過 Apache 來處理前端需求，資料庫使用 MySQL 進行管理，並使用 PHP 來處理後端程序，並分別取其英文字首作為縮寫，裡面的每個部份也都能用其他提供相近功能的程式來替代，如 Apache → Nginx、MySQL → MariaDB、PHP → C#...，但大致上不會脫離架構，但經過時代的進步，這樣簡單區分出前後端的架構已經無法負荷現在的需求，隨著功能的增多，前後端的程式碼都會變得難以維護與管理，為了方便管理以及維護程式碼，於是框架就這樣誕生了。

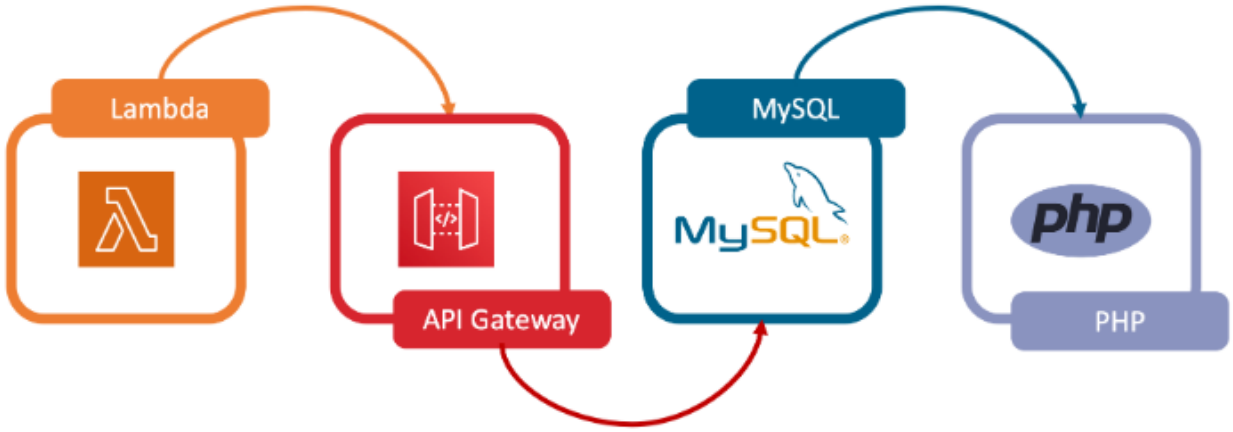

圖 1.1：LAMP 架構 (參考資料[3])

前端目前有三大主流框架，Angular、React、Vue，後端的框架則是五花八門，並且可能過個幾年又會有大變化。而現在對於網站的需要也從單純顯示資訊變成需要提供互動亦或是程式的功能，要使用這些框架前可能會需要建立環境，並且這些功能需要更新或新增時，就會需要對在線上的網頁伺服器做適度的變更，一般來說會先在本機端將程式開發完成後再上傳到運作中的伺服器，來完成改版的目的，勢必會需要停機時間來操作，若其中遇到錯誤又或者是程式和在本機端運作的結果不同，就會延長停機的時間，尤其是在套用框架的現今，每次重新傳檔都會是一段不短的時長，因此如何縮短建立環境的時間並減少錯誤以及進行程式功能更新、傳檔時間就會是一大重點。

表 1.1：常用的十種後端框架 (參考資料[4])

十種後端框架				
Laravel	Django	Spring Boot	Ruby on Rails	ASP.NET Core
Express.js	NestJS	Koa.js	Flask	Phoenix

因此我們參考 DevOps 的精神來改善 Web 應用程式的開發流程，運用 Docker 來將伺

服器的環境建置好，此處可選擇使用 Dockerfile 建立 image 或將目前開發用環境進行打包還原，使環境建立時間縮短，之後執行提供的自動建置腳本，此腳本會執行建立容器與配置 apache config，只需要短短幾個步驟就能將應用程式建立好並上線，也能根據負責不同部分各自進行開發，透過分工增加效率。

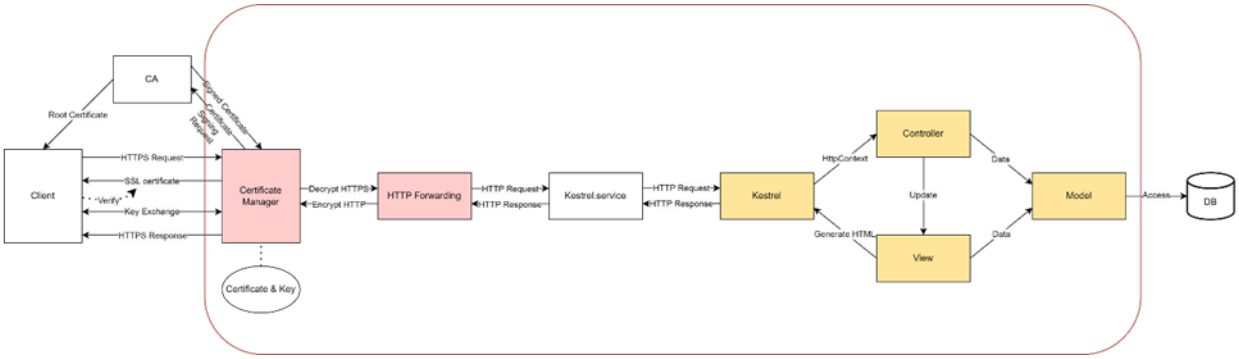

圖 1.2：傳統包進容器內的流程 以.NET 為例 (參考資料[5])

目前對於要在打包後的 Docker Image 新增、修改或管理憑證的流程並沒有一個好的解決方案，通常將整個環境打包時會連帶憑證一同被打包進 Image 中，以及對於在將程式從測試環境移至正式環境若環境有所差異，用現今的方法重新建置步驟過於繁瑣，且可能會遇到系統、軟體版本不一致、資料庫遷移、硬體配置不同...種種狀況，會耗費大量時間與精力。因此希望能透過本專題提供一套方法來使憑證能夠正常安裝並運行，並且不像以往存在於容器中不易管理，讓 Docker 也能擁有 HTTPS 的安全性，使運行環境能夠重現於任何環境中，不論是在測試或是正式運行時都只要透過一樣的程序設置便能確保程式的正常運作。

第二章、 文獻回顧與探討

2.1 DevOps [6]

DevOps 代表兩種服務 Dev(Development)和 Ops(Operation)之間互相溝通、配合的模式，在過去的 IT 文化當中，開發團隊與維運團隊之間並不會互相配合，且互相衝突，如此一來會造成交付或完成進度的延宕，有人意識到了這點[7]，並希望對於 Dev 和 Ops 之間的配合進行改進，並思考合作的方式，將兩個團隊整合，以更有效率的方式進行工作，DevOps 就這樣誕生了。

以這種模式進行開發通常會分為四個階段：規劃(Plan)、開發(Develop)、傳遞(Deliver)、維運(Operate)[8]，每個階段互相依賴，且會不斷循環以達成隊程式持續的改進。在規劃階段中，團隊應該對於即將開發的程式需求以及後續的擴充性進行設計與討論，並規劃出大致上開發的流程和開發失敗的應對措施，確保後續階段順利；開發階段會開始按照規劃的流程實際進行程式的開發，程式碼的編寫、測試是這個階段的主要目標，對於團隊協作在此處也會使用版控工具(如 git)讓功能分別交付給不同成員開發，並確保程式碼不會因開發進度不同而混亂，並經由持續整合/持續交付工具(CI/CD)令程式能及時部署至環境中運作與執行；在傳遞階段會對於開發完的程式進行部署並檢查其是否正確在環境中執行，發生錯誤則會進行復原，也包括對於環境的設定，確認開發環境與運行環境是否一致；而在維運階段主要是使程式在環境中運作正常且能穩定的執行，並持續監控，發現問題就及時修復與通知，也會記錄執行情況，若出錯以便觀察問題來源，並且做好備份來使發生錯誤時

能迅速回復。

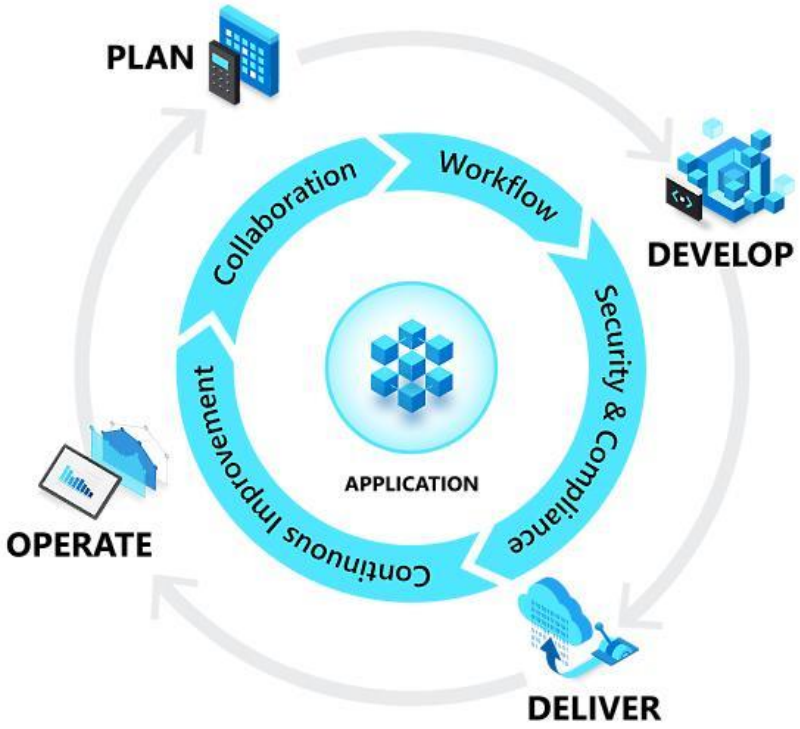

圖 2.1：DevOps 流程(參考資料[8])

本專題針對 DevOps 中的流程進行改善，引入 Docker 來使用，透過 Docker 來使環境能夠很好的被搬運以及運行程式，使程式能夠在各種環境中執行，降低發生錯誤和停機時間過長的情況，並且透過與 Reverse Proxy 的配合來使網址連入時能夠正常轉送進入程式中，使開發、傳遞、維運這三個階段能夠更加便捷，減少不同階段環境差異產生的問題。

2.2 Docker [9]

Docker 是一個開源的平台，會用於打包、部署環境，並使應用程式在環境中執行，會透過容器化的技術來實現，Docker 會建立一個 Image 為基本執行的系統，再在其之上

建立一到多個 Container，每個 Container 之間各自獨立運行，並且 Image 所建立的環境能夠在有安裝 Docker 的主機上很輕鬆的還原出來後建立 Container 即可執行應用程式，相比一般的 VM 會需要將整個系統虛擬化，Docker 這種方式會比較輕量化且快速，並且確保其能夠在不同的主機上運行結果一致。

Image 是 Container 的基礎，他會包含所需執行的應用程式應該有的依賴關係，其可以打包一個已經建立好的系統或是使用 Dockerfile 來還原環境在不同的設備，也可以透過 Docker Hub 來找有沒有符合需求的 Image 可供使用，並且 Image 會作為 Container 的模板，每個在這個 Image 下建立的 Container 都會執行在 Image 的環境之下，Container 內會放置要執行的應用程式，並且每個 Container 都可以單獨被啟動或停止，因此我們可以以此來運行很多建立在同一環境下的應用程式，並且能獨立被控制，也可以以此來達成在應用程式需要更新時，將新版建立在另一個 Container，確認執行無誤再切換執行為新版，以此減少網站停擺的時間。

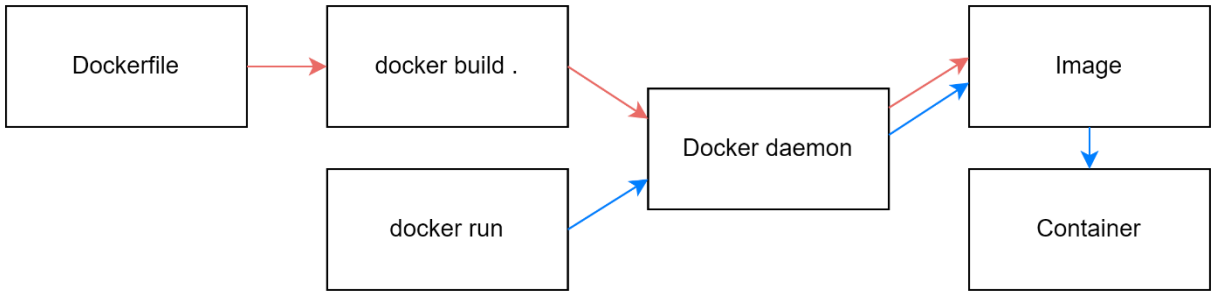

圖 2.2：Docker 建置流程

本專題運用 Docker 令開發應用程式時能夠有效率的建立環境，我們在建立前後端時，分別對前端與後端各自的環境建立不同的 Image，並在 Image 中運用 Container 讓伺服器能夠一次運行多個前後端，並且易於控制，在需要移轉到其他設備時也能藉由

Docker 將環境打包為一個 Image 的特性，來讓運行環境能夠在任何設備中易於還原，從而降低重新設置環境所會造成的時間浪費以及錯誤發生的機會。

2.3 Reverse Proxy

在本專題中，我們希望能夠使 Docker 內部的伺服器與外部溝通時不只是使用 HTTP 協定來進行連線；HTTP 是網頁與用戶端之間的一種通訊協定，通常會透過 port80 來傳輸網頁的服務，若只使用 HTTP，其在傳輸時的訊息會直接傳送，也就是不經過加密的方式傳輸，因此在過程中任何人都可以從中擷取資料。而 HTTPS 會在 HTTP 的基礎上以加密形式進行傳輸，因此相較於 HTTP 會提升安全性，並且在現今的網站上大多採用 HTTPS，HTTP 則會被網頁瀏覽器標示為不安全的網站。

要使用 HTTPS 就會需要對伺服器新增由第三方機構核發的憑證，一般情況下，要管理憑證可以直接進到目錄中使用相關工具進行管理，但運用 Docker 後會將建置好的運行環境整個打包，導致憑證也一起被包在裡面，憑證會有有效期限的問題，暨時會需要特別進到 Docker 內部進行設定，並且會需要特別設定對外開放的 Port，對於後續維護與使用會有難度，於是我們採用 Reverse Proxy 的方式，在外部就透過反向代理伺服器建立 HTTPS 的連線後再連進 Docker，以此來令憑證不會被建立在內部。

Reverse Proxy 是一種伺服器配置，中文稱作反向代理伺服器，其主要功能是接收來自客戶端的請求，然後轉發這些請求到不同的伺服器，主要可以提供負載平衡、加速 web 伺服器(快取)、安全和隱私性[10]，而其中這次計畫主要會應用到的是安全性的部分，我

們設計了一個 Reverse Proxy 存在在我們的架構當中，其會將外面傳送至伺服器的請求先以 HTTPS 傳送至 Reverse Proxy，再經由其轉送給我們的 AP Server，以此來避免在 Docker 中操作憑證，並且具備安全性，同時也可透過這個 Reverse Proxy，使容器之間透過呼叫 API 的方式進行溝通，透過這個方法來調用在 Image 中不同環境執行的後端 API 與前端伺服器。Reverse Proxy 流程架構如下：

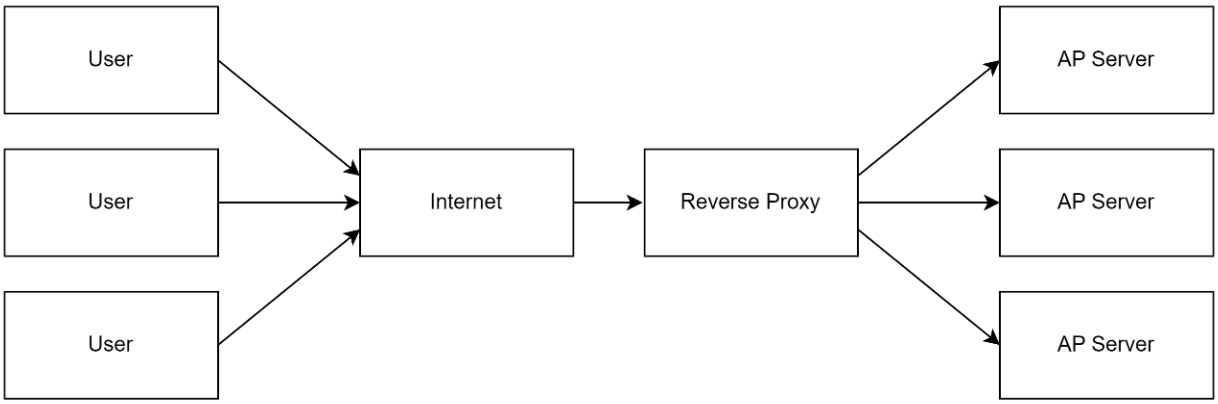

圖 2.3：反向代理伺服器的流程

第三章、 研究方法與成果討論

3.1 研究方法

我們會先建立一個網路應用程式，這個應用程式具備分離的前後端，並且透過 API 的方式進行溝通，分別對前端與後端運行的環境建立各自的 Docker Image，並使應用程式能透過 Volume 的映射，分別放置進去對應 Image 中的 Container，使程式執行在建立的環境之下，對於前端與後端間 API 互相的請求則藉由 Reverse Proxy 來做溝通，並確保後端能正確存取資料庫，以這種方式來達成網路應用程式的三層式架構，並且前端與後端就能建置各自的環境，並透過不同的設置執行各自的框架。而接收外部請求則由 Reverse Proxy 先經由 HTTPS 接收後，再轉發至前端的網頁中。在理想的情況下，開發團隊只需要將開發後的應用程式根據對應的路徑放入環境中即可執行，而維運團隊只需要確保環境正確被還原。

我們的研究步驟如下：

步驟一：了解背景知識

1.1 了解前後端架構與差異

1.2 對環境有基本認識

步驟二：建置基本網路應用程式

2.1 嘗試建立前端伺服器

2.2 建立後端 AP 伺服器與資料庫

2.3 測試其基本操作

步驟三：研究 Docker 與其建置流程

3.1 理解 Docker 的概念

3.2 Image、Container、Volume 之間的建置

3.3 設計與了解環境的需求與架構

3.4 打包與還原環境的實作

步驟四：研究 Reverse Proxy 的知識與建立

4.1 了解 Reverse Proxy 的運作邏輯

4.2 以 SSL 加密進行轉送

4.3 對憑證進行管理

4.4 實作 Reverse Proxy

步驟五：DevOps 開發流程的改善

5.1 了解 DevOps 基本流程

5.2 實際手動建立並了解改善需求

5.3 實作流程部署

步驟六：系統測試與錯誤排除

6.1 對改進後的流程進行測試與操作，確保無誤，或找出錯誤加以改進

6.2 進行實際運作，來了解使用是否順暢，並進行調整

步驟七：結果與報告書撰寫

3.2 系統架構

主要可透過圖 3.1 與圖 3.2 了解系統大致架構，用戶透過 DNS 存取網站，而用戶欲存取的子網域會經由 Reverse Proxy 存取到對應的 Container 中，並且憑證存在於 Reverse Proxy 中，達成我們所希望的將憑證隔離出容器的目的。

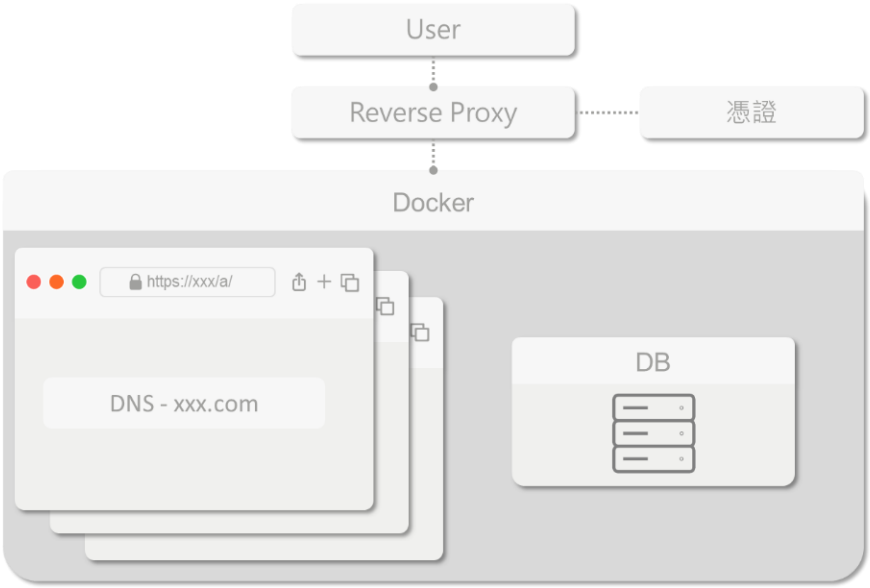

圖 3.1：系統運作示意圖

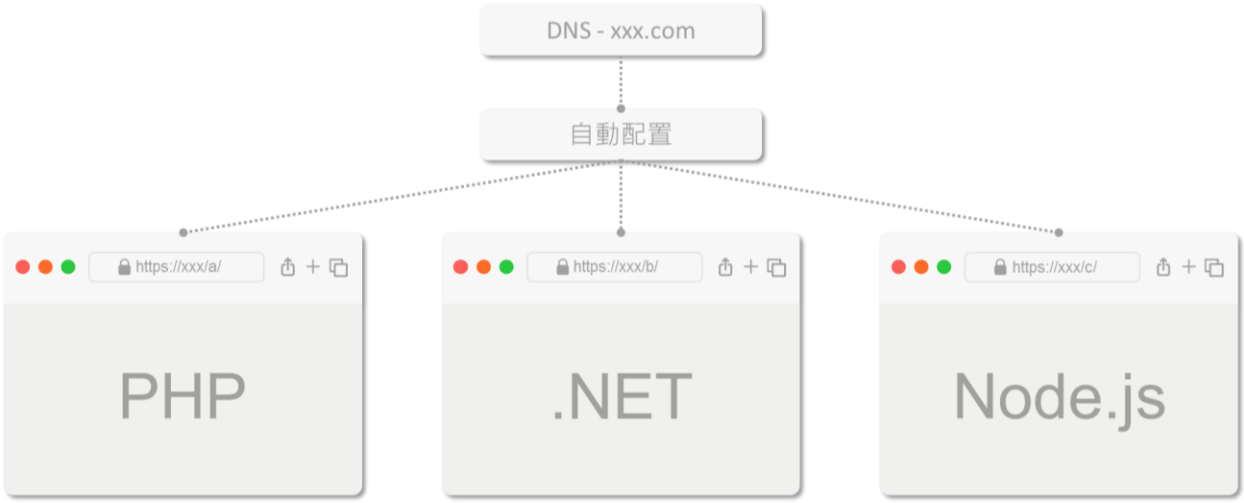

圖 3.2：系統建置示意圖

3.2.1 檔案結構

圖 3.3 為此系統的基本構造，內含 create.sh 與 remove.sh 並隨附上 Dockerfile 範例與 apache2 範例配置檔，其中 Dockerfile 會將 app 中的程式存放進容器中供容器運作使用，docker build 後再接續配合使用 create.sh 與 remove.sh 即可完成自動建置。

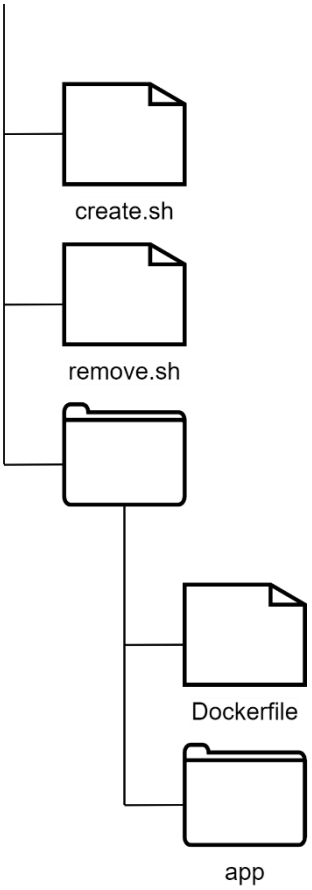

圖 3.3：系統資料夾結構示意圖

3.2.2 Web Server (Reverse Proxy)

圖 3.4 是我們系統流程的架構，粉紅色底是 Web Server，也就是作為 Reverse Proxy 使用的地方，會運用 apache2 進行架設，其會接受 HTTPS 的需求並將需求透過 HTTP Forwarding 到 AP Server 中，可參考圖 3.5 配置。

3.2.3 AP Server (Container)

參考圖 3.4，黃色底部分作為 AP Server 使用，紅色圈起部分則各自為一個容器，因此以此示意圖而言總共有 1 個 Web Server 和 3 個由 Docker Container 所運行的 AP Server，如有需要也能新增容器用作資料庫使用。

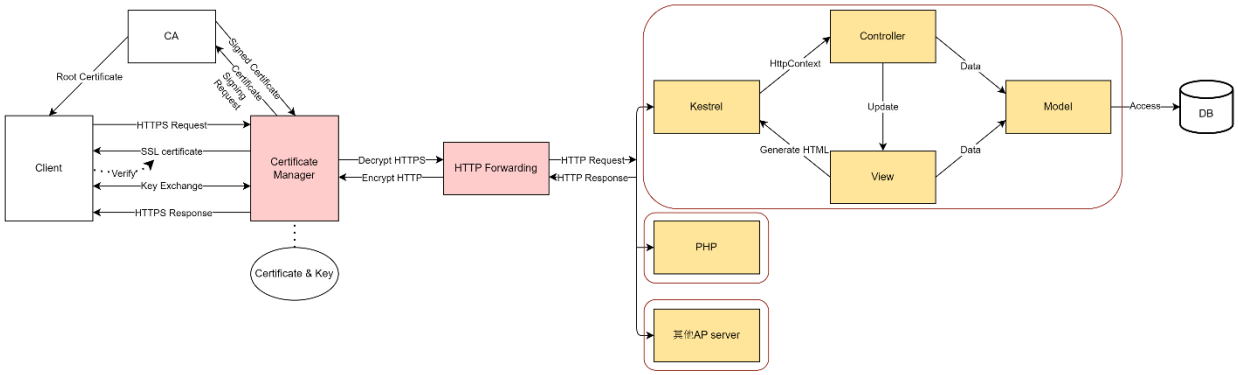

圖 3.4：系統流程架構圖

```
<VirtualHost *:80>

        ServerName spl.sytes.net

        ServerAlias spl.sytes.net

#       <Location />
#       Order allow,deny
#       Deny from all
#       </Location>

#        Redirect permanent / https://spl.sytes.net/

</VirtualHost>

<VirtualHost *:443>

        ServerName spl.sytes.net

        ServerAlias spl.sytes.net

        SSLEngine on

        SSLCertificateFile /etc/letsencrypt/live/spl.sytes.net/fullchain.pem

        SSLCertificateKeyFile /etc/letsencrypt/live/spl.sytes.net/privkey.pem

        RewriteEngine on

</VirtualHost>

<Proxy *>

        Order allow,deny

        Allow from all

</Proxy>

ProxyPreserveHost On

ProxyRequests Off
```

圖 3.5：apache2 配置示意圖

3.3 自動建置

自動建置作為完成我們專題的最後一塊拼圖，其會運用腳本來自動進行容器的建置以及 apache2 的配置，並分為 create 與 remove 兩個腳本進行使用，腳本使用的指令請依照 sudo <sh> <subsite> <port> <image id> <ex>的方式進行使用。

3.3.1 create.sh

參考圖 3.6，create.sh 作為建立容器的腳本，其主要會運行 docker run 與編輯 apache2

相關指令，其會根據輸入的值進行對應操作，用戶只需輸入其希望的子網域名稱與想建立容器的 port 並將 image id 輸入即可自動建立與配置好容器，使伺服器上線使用，而 ex，即外加項可自由選填，如透過--link 可連接其他容器，其使用方法為 sudo create.sh <subsite> <port> <image id> <ex>，假設欲建立一子域名為 php 將其建立在 port 9527 上，image id 為 12345678，並且其需連接至另一建立在容器中的資料庫 mariadb，其指令應為 sudo create.sh php 9527 12345678 '--link mariadb -e PMA_HOST="mariadb"'。

```
#!/bin/bash

subsite="$1"

port="$2"

image="$3"

extra=" $4 "

sed -i "/<\/VirtualHost>/i \
        RewriteCond %{REQUEST_URI} ^/${subsite}$ \n\
        RewriteRule ^(.*)$ /${subsite}/ [R=301,L]" /etc/apache2/sites-enabled/project.conf

sed -i "\$aProxyPass /${subsite}/ http://localhost:${port}/" /etc/apache2/sites-enabled/project.conf

sed -i "\$aProxyPassReverse /${subsite}/ http://localhost:${port}/" /etc/apache2/sites-enabled/project.conf

docker run --name "${subsite}_${port}" -idt -p ${port}:80 ${extra} ${image}

service apache2 reload
~
```

圖 3.6：create.sh

3.3.2 remove.sh

參考圖 3.7，remove.sh 作為刪除容器的腳本，其會將透過 create.sh 所建立的容器與編輯的配置自動刪除，其使用方法為 sudo remove.sh <subsite> <port>，若要刪除上面作為範例由 create.sh 建立的子網域 php，其指令應為 sudo remove.sh php 9527。

```
#!/bin/bash

subsite="$1"

port="$2"

sed -i "/RewriteCond %{REQUEST_URI} ^\/${subsite}/d" /etc/apache2/sites-enabled/project.conf

sed -i "/RewriteRule ^(.*)$ \/${subsite}\/ \[R=301,L\]/d" /etc/apache2/sites-enabled/project.conf

sed -i "/ProxyPass \/${subsite}\//d" /etc/apache2/sites-enabled/project.conf

sed -i "/ProxyPassReverse \/${subsite}\//d" /etc/apache2/sites-enabled/project.conf

docker stop "${subsite}_${port}"

docker remove "${subsite}_${port}"

service apache2 reload
```

圖 3.7：remove.sh

3.4 系統安裝流程

可參考本專題的 GitHub(https://github.com/Spl6026/DevOps-Docker-ssl)，首先先自行安裝 docker、apache2 以及準備好的憑證，憑證根據圖 3.5 範例方式安裝進 Web Server 後，進行 docker build 或直接輸入 image id，並搭配欲設定的子網域名稱與 port 以及自行設定的外加項運行自動建置腳本，腳本即會將伺服器架設完畢。

圖 3.8 顯示了有關系統運作的實際狀況，其中 php_9528 與 transch_9527 為腳本所製作的容器，搭配其他容器的 mariadb 與 phpmyadmin，構成一個完整的伺服器運行環境。

```
itlab@itlab-pr2024:~$ sudo docker ps -a
CONTAINER ID   IMAGE         COMMAND                  CREATED        STATUS        PORTS
      NAMES
1b8018c2fafe   f35f          "docker-php-entrypoi…"   6 hours ago    Up 6 hours    0.0.0.0:9528->80/tcp, [::]:9528->80/tc
p     php_9528
2aac413ce175   57da          "dotnet run --projec…"   33 hours ago   Up 33 hours   0.0.0.0:9527->80/tcp, [::]:9527->80/tc
p     transch_9527
14159a52c1fd   phpmyadmin    "/docker-entrypoint.…"   39 hours ago   Up 39 hours   0.0.0.0:8080->80/tcp, [::]:8080->80/tc
p     phpmyadmin
a6495182a3f8   mariadb       "docker-entrypoint.s…"   39 hours ago   Up 39 hours   0.0.0.0:3306->3306/tcp, :::3306->3306/
tcp   mariadb
```

圖 3.8：系統容器實際運作示意圖

第四章、 結論

我們期待這套流程能讓開發和維運的使用者在建置和維護 Web 應用程式的時候能夠加速工作進行，省去時間的不必要花費，並對兩者有不同的影響：

1. 開發人員：人員在開發時能夠使用這套流程來使自己在建立應用程式的時候不需顧及環境的設置，能夠將自己本機端使用的環境完整還原到任何地方，如此一來可以最大程度的專注在開發功能上，功能開發完成就放進負責維運的人員所指定的目錄就能正確執行。
2. 維運人員：維運人員能夠減少跟開發人員之間的溝通的衝突，其能透過這個系統來還原開發人員所提供的 Image，並將放置應用程式的目錄設定好，確認憑證的設置後，等待功能開發完畢，後續就是確保應用程式正常運行即可。

希望能透過這種方式令 Docker 支援 HTTPS 的目標以及前端與後端之間的溝通，使伺服器能夠快速被建立，並保證系統運作正常且方便管理。

參考文獻

[1] STEFANOVA, Ramona. Exploring the Latest Front-End Development Trends. 2024.

[2] LEE, James; WARE, Brent. Open Source Web Development with LAMP: Using Linux, Apache, MySQL, Perl, and PHP. Addison-Wesley Professional, 2003.

[3] What is a LAMP stack? https://aws.amazon.com/what-is/lamp-stack

[4] Top 10 Backend Frameworks in 2024 https://www.turing.com/resources/backend-frameworks

[5] When to use Kestrel with a reverse proxy https://learn.microsoft.com/en-us/aspnet/core/fundamentals/servers/kestrel/when-to-use-a-reverse-proxy

[6] EBERT, Christof, et al. DevOps. IEEE software, 2016, 33.3: 94-100.

[7] Velocity 09: John Allspaw and Paul Hammond, "10+ Deploys Pe https://www.youtube.com/watch?v=LdOe18KhtT4

[8] What is DevOps? https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-devops

[9] MERKEL, Dirk, et al. Docker: lightweight linux containers for consistent development and deployment. Linux j, 2014, 239.2: 2.

[10] What Is a Reverse Proxy Server? https://www.nginx.com/resources/glossary/reverse-proxy-server