中 原 大 學
資訊工程學系
專題實驗報告

# 基於 Docker 的 ROS 應用市集與管理系統
ROS Market place and Management System Based on Docker

指導老師：鍾武君教授

專題生：

10827225 蔡亞軒
10827232 魯霈恩
10827247 陳奕誠

中華民國 111 年 12 月 2 日

# 目錄

# 圖目錄



# 表格目錄

# 第一章　緒論

## 1.1　摘要

隨著科技發展，機器人在各領域的占比越來越重，針對多樣化的需求，各界也研發出了各式各樣的機器人。然而因為各功能間的環境設置差異，無法彈性地切換功能去面對不同情形，因此我們想藉由 Docker 結合 ROS（Robot Operation System－機器人作業系統），將功能偕同其環境設置一併容器化，讓各功能在互不影響的情形下作業，並透過視覺化的網頁讓使用者方便使用。

## 1.2　研究動機

現今機器人技術逐漸普及，網路上出現了各式各樣的開源資源，讓機器人可以做到各式各樣的任務，例如：SLAM（掃圖與導航技術）、圖像辨識、深度學習，這些技術已經應用在許多方面上。

然而，這些完善的技術需要繁瑣的設置過程、複雜的環境套件，當我們想將網路上的開源資源安裝至 ROS 機器人上時，這些資源不僅沒辦法快速地被安裝，更容易因為不同套件間的版本產生相容性問題，導致在設定上花費了大量的時間。

因此我們希望設計一套系統，讓用於 ROS 的開源資源包能夠透過容器化的方式，使各功能在使用上不會發生相容性問題。不僅擁有高度彈性，更能夠省去使用者花費在不同的機器上安裝相同功能的時間，甚至可以透過 Docker Compose 的技術使一台機器可以擁有複合型的功能，可用於處理複雜情形。如此一來，即便是單一一台機器也有著高度的彈性及擴充性，隨時能夠切換其定位，使用者也可以隨時進行調度。

## 1.3 研究目的

### 1.3.1 功能容器化

- 如何迅速移植功能至機器上
- 如何最簡化功能使用流程
- 如何將圖像、影像辨識結果導出容器，並針對該結果進行運用
- 如何將不同功能結合，使機器有複合型功能

### 1.3.2 網頁與設備的通訊

- 網頁前端與後端之間的通訊與資料傳遞
- 如何用 TCP protocol 使設備 server 端與 client 端之間傳遞資料。
- 網頁前端與設備 server 端之間如何通訊
- 使用者如何透過網頁監看使用狀況

# 第二章　研究架構與方法

## 2.1　研究架構

### 2.1.1　前端

1. Vue.js 做為前端的框架
2. Vue Router 用來分配頁面路由
3. Vuetify 為語意化元件的前端框架，用以繪製網頁。
4. Axios 與後端通過 API 溝通的 HTTP 請求庫，它可以用 Get / Post / Put / Delete 請求與後端 API 進行交互
5. Vuex 做為網站的全域狀態管理庫，用以記錄後端發送之 jwt token
6. Components（组件）共分為五個子組件 vue 頁面
7. Base64負責密碼訊息的加密與解密

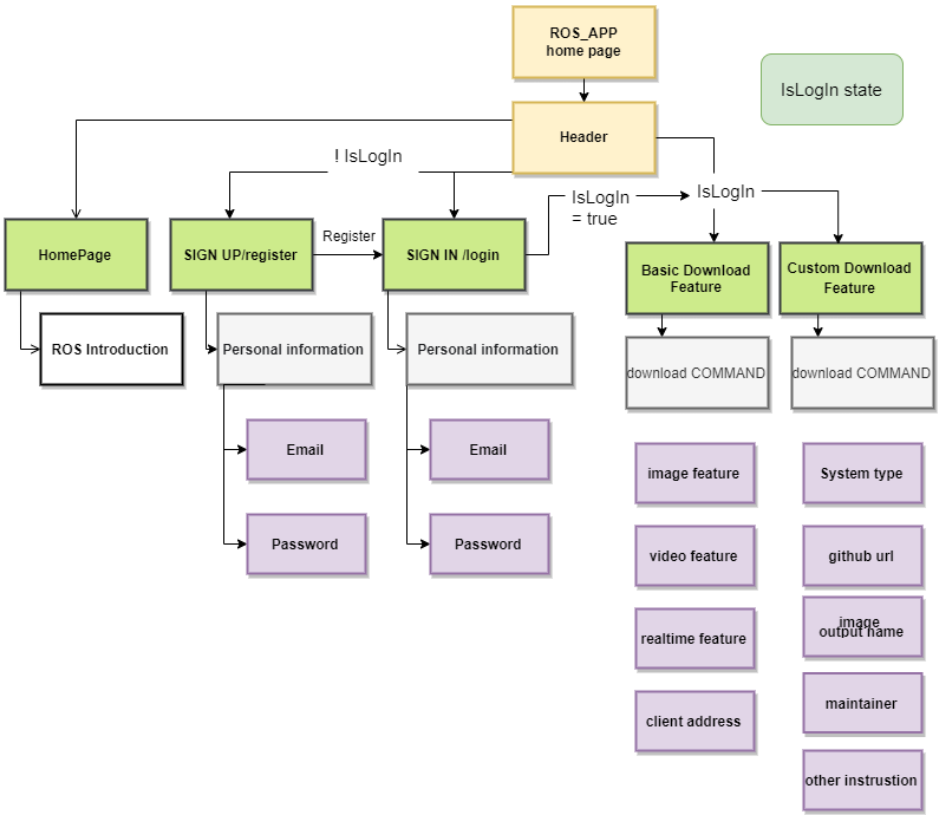


圖 1 前端架構

### 2.1.2 後端

1. node.js 做為後端環境
2. 使用 Express web 應用框架建置 Rest Api 以及配置跨域
3. Sequelize 是一種 ORM 框架，透過物件導向的概念來連線並操作資料庫
4. Sqlite 作為使用者帳號資料庫
5. Net 建置 TCP 連線
6. Middle-ware
7. jwt(*JSON Web Token*)： 驗證用戶 token 機制
8. Bluebird：封裝 Promise
9. Bcyrp：Hash 雜湊處理加密用戶密碼

| 序號 | URI | HTTP 方法 | 發送內容 | 結果 |
|---|---|---|---|---|
| 1 | download | GET | 空 | 響應式資料綁定所有連線設備列表 |
| | custom | GET | 空 | 響應式資料綁定所有連線設備列表 |
| 2 | download | POST | JSON 字符串 | TCP 通訊協定傳送下載指令置設備端 |
| 3 | custom | POST | JSON 字符串 | TCP 訊協定傳送建置 docker image 指令 |
| 4 | login | POST | JSON object | 獲取用戶登陸結果，以及 token 驗證 |
| 5. | register | POST | JSON object | 用戶註冊驗證政策，用戶登陸結果 |

表格 1 後端處理方式整理

### 2.1.3 設備端

網頁後端與設備端之間的訊息透過TCP socket進行傳輸，設備端接收訊息後會將此做為參數傳入client.py讓其進行後續處理，其中包含自動生成dockerfile並建置成image以及從docker hub下載image並啟動container至完成所有動作。

### 2.1.4 架構圖

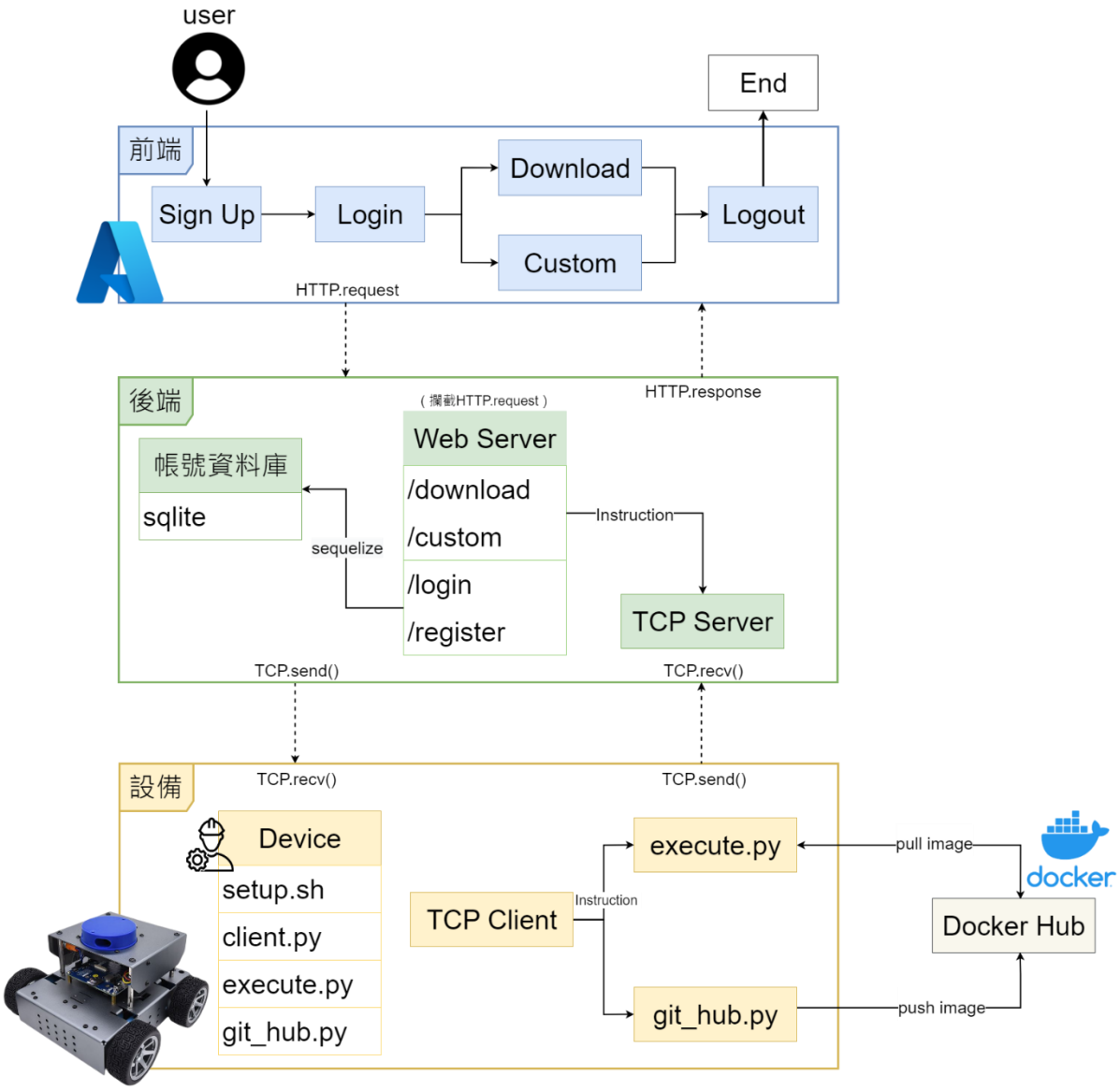


圖 2 架構圖

## 2.2 研究方法

### 2.2.1 容器化 Container

#### # 使用說明

我們使用的平台軟體是 Docker，透過 Dockerfile 或是 yaml 檔將功能所需的套件化為一層一層的 layer，由 layer 架構出 image 檔，最後再運行 image 檔生成該單一功能的 Container，亦或是運行 Docker-compose 根據 yaml 檔生成有複合功能的 Container。

#### # Docker容器

Docker 是使用 Go 語言所開發的一套平台軟體，他利用了 Linux 核心中的資源分離機制，以及 Linux 核心的命名空間(namespace)，建立獨立的容器(container)，使其能夠在單一 Linux 下實作，避免額外啟動一個虛擬機器的負擔。與虛擬機器的不同在於 Container 是將作業系統層虛擬化，而非像虛擬機器將硬體虛擬化的方式，因此有更高的可攜性，在伺服器運用方面也相對優秀。

我們選擇使用 Docker 正是因為看中其輕量化及高可攜性的特點，這不僅能幫助我們快速在不同平台、環境上得到相同的效果，同時也最大程度的減少了重複且繁瑣的安裝過程。

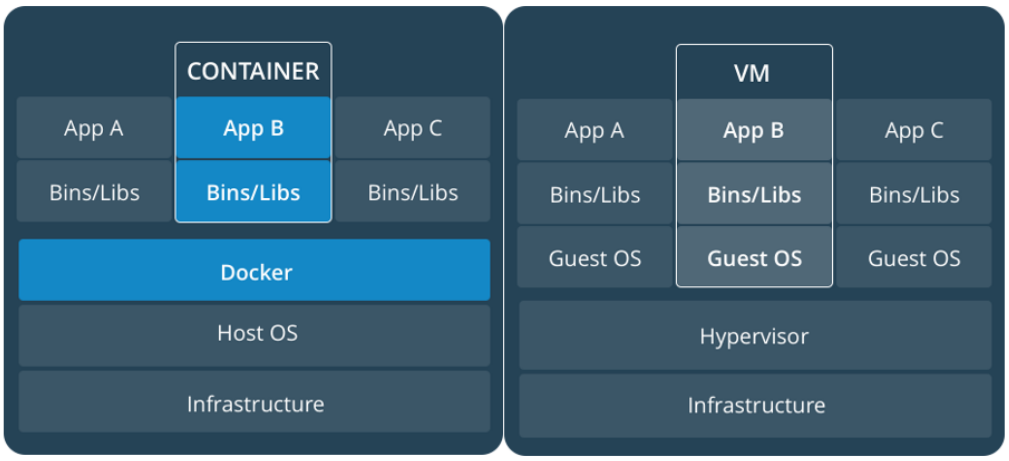


圖 3 虛擬機器與Container之間的差異

## # Dockerfile

Dockefile 為單一 container 的設定檔，裡面包含了該 container 的環境設定，以及所安裝的基礎套件，也可以提前規畫該 container 啟用時要執行的指令，將這些設定分層化後建置出 image 檔。

我們透過將啟動後需執行的步驟設定在 Dockerfile 中以達到後面全自動化的執行，所以當使用者在使用我們提供的功能時，除了輸入必需資訊外，不需要輸入任何一行指令即可輕鬆的直接獲取結果。

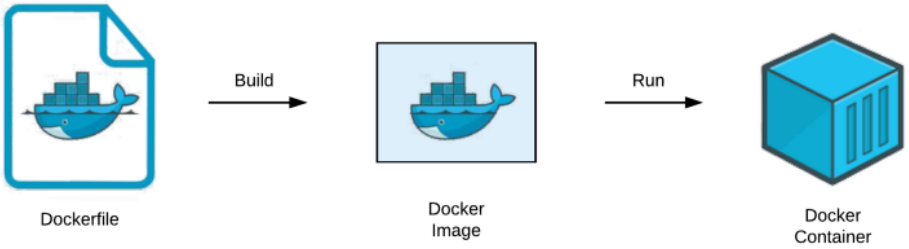


圖 4 Docker容器的架構流程圖

## # Docker Compose

Docker Compose 是 Docker 的擴充架構工具，用來定義以及執行多個 Docker 容器，我們可以透過撰寫 yaml 檔，組態應用程式需要的所有服務，並透過簡易命令，隨時切換已包含在內的不同服務而無須重新下載 image。

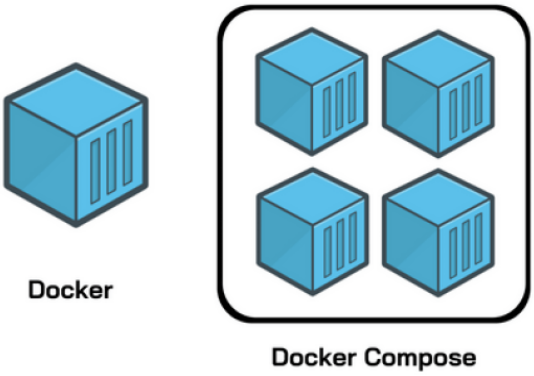


圖 5 單一Docker容器與Docker-Compose的不同

### 2.2.2　視覺化使用者介面

#### # 使用說明

一般而言，操作 container 需要在 terminal 中輸入指令才能執行或是傳遞資料，這與我們想要達成的方便性有所差異，因此我們透過以 Vue 語言撰寫的網頁前端讓使用者能夠在網頁上只需幾個簡單的按鍵，就能對設備進行操作。為了在網頁開發流程中降低前後端混合開發的複雜度以及減少後端伺服器的壓力，同時使人數乘載提高、頁面渲染加速、增加用戶體驗感，我們用 Node.js 架構後端以達到前後端分離的效果。

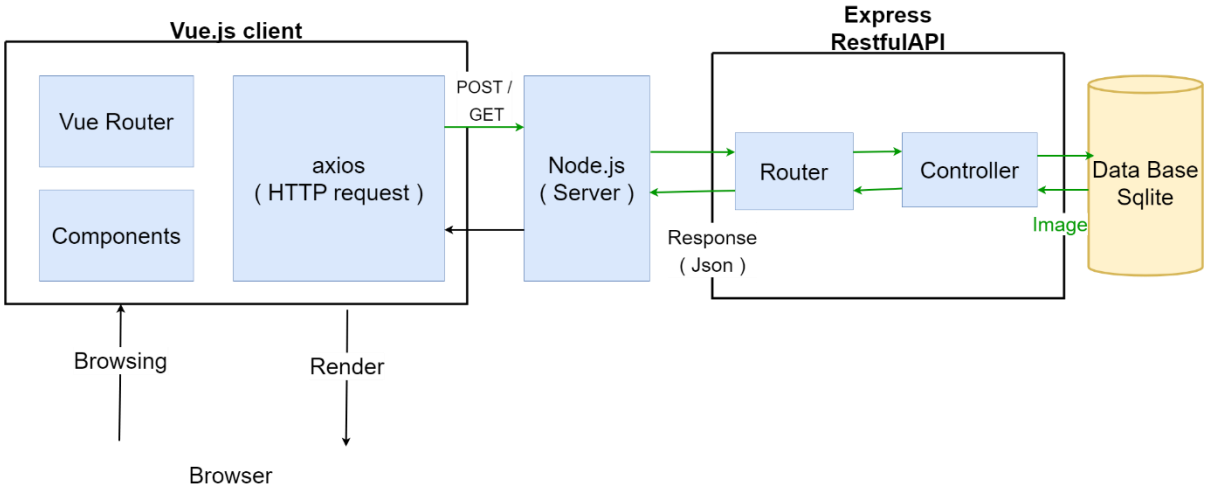


圖 6 全端架構圖

#### # 前後端分離

在此系統中使用到「前後端分離—Client Side Rendering（CSR）」的概念，在 SPA 模式下開發，伺服器只是純粹做一些邏輯與資料處理，透過 Web API 將 JSON 資料傳給瀏覽器，主要的實作方式會分兩個階段：

第一階段是瀏覽器先跟伺服器取得小量的 HTML，待瀏覽器顯現後使用 axios 一種建立在 Ajax 上的套件利用 BOM 所提供的 xmlhttprequest 發出第二次請求，取得更多的資料，並透過 JavaScript 把資料轉換成 HTML 放進 DOM 產生完整介面。

### 2.2.3 設備事前設置

#### # 使用說明

雖然我們大幅減少了使用者進入機器的操作，但還是有事前設置的必要性。負責事前設置的人員只須先進入機器系統，進行簡易操作後即完成設置。設置內容包含安裝必需套件、進行自動連線的設置及啟用。

#### # 設置過程

設置人員需要先從github下載設置檔案，並執行其中的bash檔以完成必須的設置。以下為所需的檔案及用途：

1. Setup.sh

   自動安裝系統內部可能缺少之套件、設定日後開機自動執行的服務 start_connect.service 內容，其內容包含工作目錄、執行的檔案路徑、是否自動重啟以及自動重啟的時間間距。

```
[Unit]
Description= connect to server when device starts every time
[Service]
WorkingDirectory=/home/chiz/exe
ExecStart=/usr/bin/python3 /home/chiz/exe/client.py
Restart=always
RestartSec=3s
```

圖 7 start_connect.service內容

2. client.py

   接收 server 端傳遞過來的訊息，並且透過訊息內容去決定下一步是執行 git_hub.py 或是 execute.py。

3. git_hub.py

   透過收到的訊息生成 dockerfile 並建置成 image，之後再上傳至 docker hub 供使用者使用。

4. execute.py

   透過訊息可以讓設備端自動從 docker hub 下載對應的 image 並執行。

# 第三章 現有功能容器簡述與自定義功能容器化

## 3.1 現有功能容器

### 3.1.1物件偵測

#### 3.1.1.1 選擇YOLO的原因

網路上有許多開源的圖像辨識模型供使用，而我們選擇的是建立在 openCV 基礎上的 YOLO v3。有別於其他模型，YOLO 只需對圖片進行一次 CNN 即可判斷物體類別與位置，大幅提升辨識效率的同時甚至可以達到即時辨識。除此之外，YOLO 的準確度也比其他模型高許多，因此最終我們決定使用 YOLO 作為辨識的主要模型。

#### 3.1.1.2 圖像辨識

使用者透過網頁輸入欲辨識的圖片存放之目錄絕對路徑，只需傳入一些參數經由 container 執行，即可達到圖片辨識的目的，無須自行安裝即可獲得辨識結果。若使用者想獨立存放辨識結果，我們也提供使用者輸入欲儲存辨識結果的目錄位置。

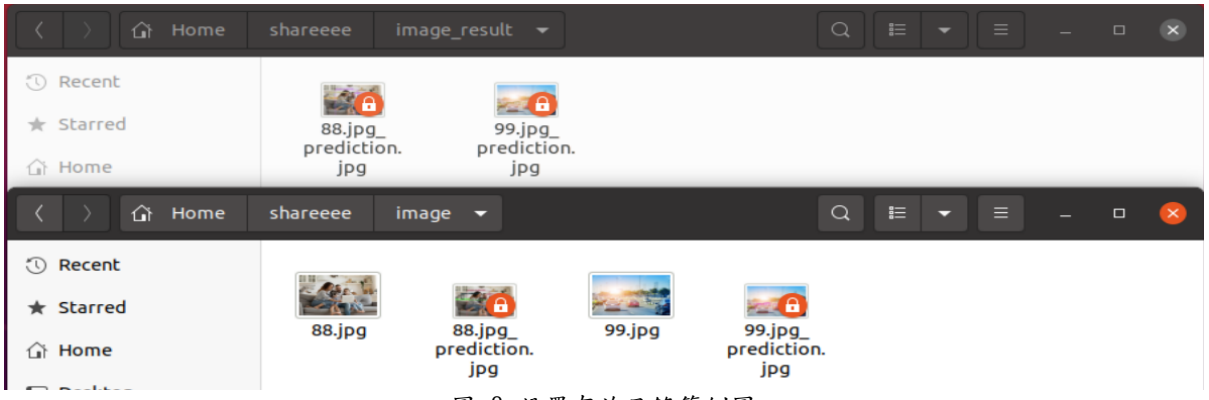


圖 8 設置存放目錄範例圖

#### 3.1.1.3 影像辨識

使用者透過網頁輸入欲辨識的影片存放之目錄絕對路徑，和圖像辨識相同，只需傳入一些參數經由 container 執行，即可達到辨識的目的，依然無須自行安裝環境即可獲得辨識結果。若使用者想獨立存放辨識結果，我們也有提供使用者輸入儲存結果的目錄位置。

#### 3.1.1.4 即時影像辨識

使用者透過網頁輸入欲掛載的 camera，即可透過 container 執行即時影像辨識。

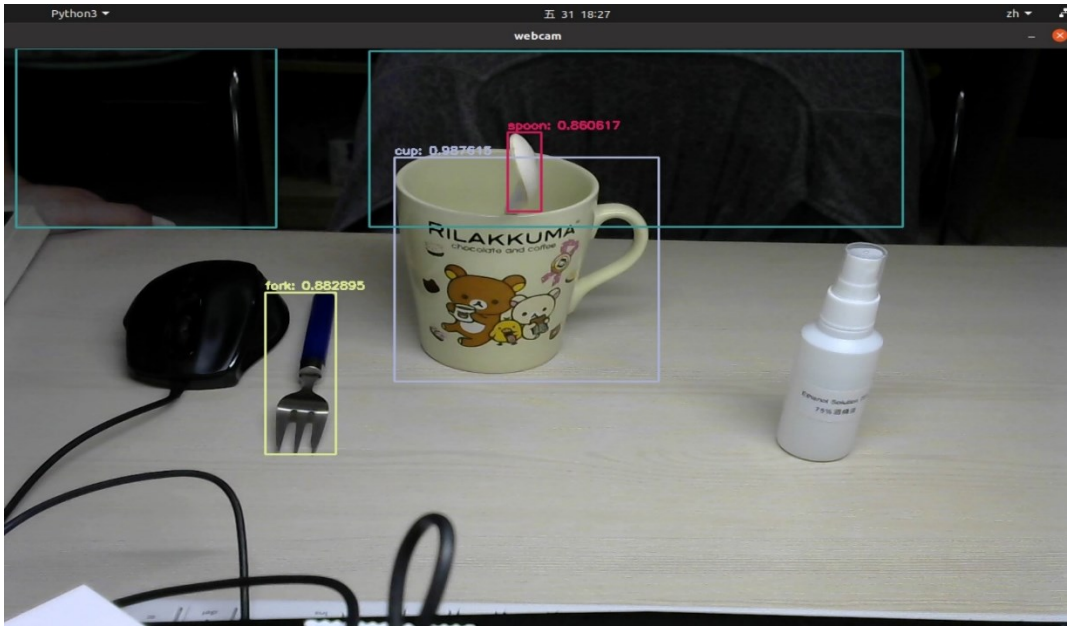


圖 9 即時影像辨識成功截圖

## 3.2 自定義功能容器化

### 3.2.1 使用說明

因為網路上有許多開源資源，以深度學習模型來說，光是根據不同情況所訓練的模型就非常多，要將所有情況會使用到的功能容器化是不太實際的，因此我們想讓使用者可以透過我們的網頁，將自己所設計的應用功能容器化，透過此種方式讓使用者可以更便於使用，而不用等待我們在系統上提供對應的功能才可使用。使用者需要輸入一些必須的參數供建置成 image，例如：基層系統名稱、輸出的 image 名稱以及欲輸入的指令內容等。

# 第四章 網頁與設備之間的通訊概述

## 4.1 設備 server 端與 client 端之間的通訊(ROS 系統)

### 4.1.1 ROS 系統簡介

ROS（Robot Operating System）為機器人作業系統，但它並不像傳統的 OS 是可以直接在電腦硬體上執行。ROS 需依附在 Linux 的環境上，其主要的功能是負責機器人各個元件之間的溝通，同時也提供大量的機器人開發程式庫供使用者使用。

### 4.1.2 通訊方式

如圖所示，當我們在 ROS 中撰寫各種應用功能時，皆須透過「MasterNode」來管理整個系統的運作。

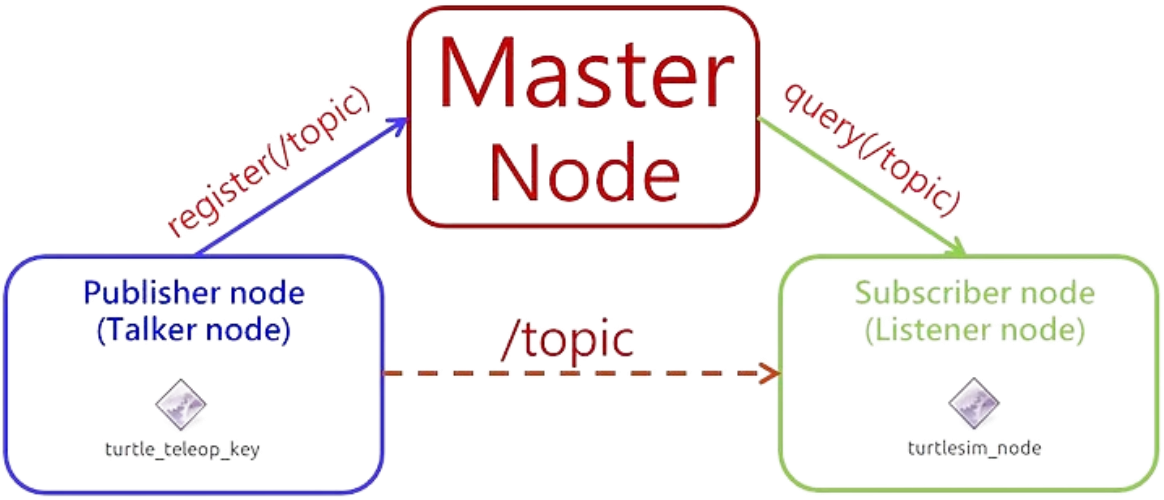


圖 10 ROS 通訊系統

Publisher Node 會發送一個 Topic 到 Master Node，然後 Subscriber Node 再從 Master Node 訂閱 Topic。Topic 為該資料的名稱，而該資料的內容(資料型態)為 Message。

## 4.2 網頁前後端的通訊

### 4.2.1 Axios 前後端交互的異步請求

Axios做為基於Promise的請求庫可應用於「瀏覽器環境」與「服務器」環境中，比起較笨重的jQuery，我們選擇使用更輕量的Axios套件。由於axios預設為接口異步，傳入api處理資料的function中的callback會被丟到工作佇列（Task Queue）中，JavaScript 會先把執行堆疊（Call Stack）全部執行完才把被丟到工作佇列的程式抓出來執行。造成數據不同步返回的問題，而 async/await 是一種建立在Promise之上的編寫異步或非阻塞代碼的新方法。

所以從語義上就很好理解async用於聲明一個函數是異步的，await則是可以暫停非同步函式的運行（中止Promise的運行），直到非同步進入resolve或reject。當接收到回傳值後，繼續非同步函式的運行。為了同步化返回數據，使用async-await對所有api做同步化處理。

### # Axios 特性

1. 可以在瀏覽器中發送 XML Http Requests
2. 可以在 node.js 發送 http 請求
3. 支持 Promise API
4. 攔截請求和響應
5. 轉換請求數據和響應數據
6. 能夠取消請求
7. 自動轉換 JSON 數據
8. 客戶端支持保護安全免受 XSRF 攻擊

以處理/register api post 為例，當 async register function 被呼叫，它會回傳一個 Promise。當 await User create 會暫停此 async function 的執行，待等待傳遞至表達式的 Promise 的解析回傳解析值，該 Promise 狀態會是 resolved 且帶有回傳值。當 async function 拋出一個例外或某個值，該 Promise 狀態會是 rejected 且帶有拋出的值，並使用 try catch 做錯誤處理。

```
async register (req, res) {
  try {
    const user = await User.create(req.body)
    res.send(user.toJSON())
  } catch (err) {
    res.status(400).send({
      error: 'This email account is already use.'
    })
    // email allready exists
  }
```

圖 11 處理 /register api pos程式碼

## 4.3 網頁後端與設備 server 端的通訊

### 4.3.1 使用說明

為了傳輸前端傳送之建置功能request指令至設備端，我們在伺服器端建立TCP server與設備端運行之TCP client連接，傳輸功能安裝請求。為了使axios請求得以連接至運行之TCP server端並實現，我們使用Node.js的net模塊創建基於TCP或IPC server端與client的異步網絡API。

| 序號 | 方法 |
|---|---|
| 1 | net.createServer([options][, connectionListener]) |
| 2 | net.connect(options[, connectionListener]) |
| 3 | server.listen(port[, callback]) |
| 4 | server.close([callback]) |
| 5 | Socket.end |
| 6 | error |
| 7 | close |

表格 2 前端主要應用函數

### 4.3.2 伺服器端建置 TCP server

透過引入net 模塊，使用net.createServer([options][,connectionListener])創建一個 TCP 服務器。參數connectionListener自動為connection事件創建監聽器。使用server.listen通過指定host和port的連接，監聽TCP server運行之端口，當新連接創建後會被觸發connection，並在已建立之佇列socketList中存取連接client之socket訊息用以在資料傳輸時辨認。socket是net.Socket實例器。使用close與error監聽client socket之連接情形，在錯誤發生與當socket完全關閉時觸發。參數had_error是布林值，它表示是否因為傳輸錯誤導致socket關閉。

```
const net = require('net')
console.log('wait for connection...')
const sock = net.createServer()

sock.on('listening', () => {
  console.log(`Server端已開啟，IP位置為：${host}:${port}`)
})
sock.listen(port, host)
const requestIp = require('request-ip')

function socketConnection (client) {
  const clientIp = requestIp.getClientIp(client)
  return clientIp
}
sock.on('connection', socket => {
  // Put this new client in the list
  const ip = socketConnection(socket)
  console.log(ip)
  socket.setEncoding('utf8')
  console.log('Connection' + socket.remoteAddress + ':' + socket.remotePort)
  socketList.push(socket)
  socket.on('error', function (err) { // 斷開連結
    console.log('err' + err.message)
    const index = socketList.indexOf(socket)
    if (index > -1) {
      socketList.splice(index, 1)
    }
  })
  socket.on('end', function () { // 斷開連結
    console.log('end')
    socket.destroy()
```

圖 12 創建 TCP server之程式碼

### 4.3.3 觸發 TCP server 傳輸

在伺服器監聽到客戶端向http server發送過來的download url request，透過路由導向對應HTTP請求方法，存取傳送之下載指令至msg參數、設備地址至client參數，遍歷socketList佇列，找尋對應之socket物件，若存在則使用socket.write()傳送下載指令至設備端，若不存在則直接response錯誤訊息，同時使用try catch作錯誤處理。

```
app.post('/download', async (req, res) => {
  const msg = req.body.string // msg是前端傳給後端的下載指令
  const client = req.body.client // msg是前端傳給後端的device 訊息
  try {
  // console.log('DATA ' + client + ' : ' + msg)
  // sock.write('You said ''' + data + ''') // 傳給device的
  socketList.forEach(function(clients){
    const eclient = `${clients.remoteAddress}\u00A0${clients.remotePort}`
    if( eclient === client){
      //可以通过端口号来区分是谁说的话
      cclients.write(`${msg}\u00A0${client}`)
    }else {
      res.status(400).send(`無此設備`);
    }
  })
  } catch (err) {
    res.send({ error: 'receive error' })
    console.log('exception: ' + err)
  }
})
```

圖 13 使用 TCP server 傳輸之程式碼

# 第五章　實驗結果與討論

## 5.1　實驗過程

### 5.1.1　實驗環境

系統：Ubuntu20.04（虛擬機）、Windows10家用版

使用軟體：docker

程式語言：python

### 5.1.2　研究過程

#### # 容器化—無法顯示GUI介面

以即時辨識為例，當我們啟動鏡頭的時候需要一個GUI介面來顯示鏡頭擷取到的成果，但是容器內不支援，因此我們後來選擇在容器啟動的時候掛載X11，以達到GUI介面的功能

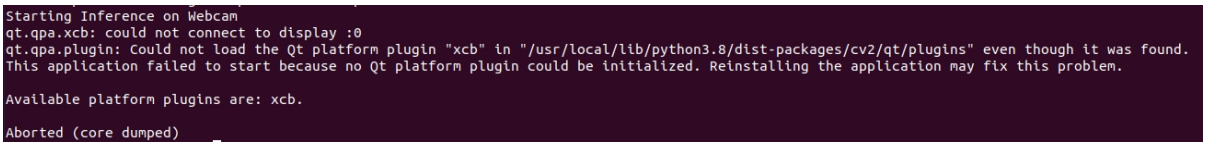


圖 14 容器內無法啟用鏡頭之錯誤訊息

## # 網頁—雲服務器跨域問題

開發環境中由於在本機進行通訊，無跨域問題產生。但在分離部屬至azure app services上後因為網頁在不同域名下傳遞資料的時候，不管是透過傳統XML Http Request，都會遵循同源政策(Same Origin Policy)，「同源」指的是同個域名底下的資源，因為只能存取相同來源的資料，所以那些跨域的請求就會被阻擋掉。

Access to XMLHttpRequest at `' https://backend20221130.azurewebsites.net/ '` from origin ' https://demo20221130.azurewebsites.net/ login' has been blocked by CORS policy: Response to preflight request doesn't pass access control check: No 'Access-Control-Allow-Origin' header is present on the requested resource.

圖 15 跨域產生之錯誤訊息

在處理跨域問題時我們修改後端 Response Header，添加前端至origin url後即允許跨域。

```
const corsOptions = {
  origin: ['https://demo20221130.azurewebsites.net', 'http://localhost:3000'],
  credentials: true, // access-control-allow-credentials:true
  optionSuccessStatus: 200
}

app.use(cors((corsOptions)))

app.use(function (req, res, next) {
  res.header('Access-Control-Allow-Origin', 'https://demo20221130.azurewebsites.net')
  // 允许的header类型
  res.header('Access-Control-Allow-Headers', 'Origin, Methods, Content-Type, Authorization')
  // 跨域允许的请求方式
  res.header('Access-Control-Allow-Methods', 'DELETE,PUT,POST,GET,OPTIONS')
  res.header('Access-Control-Allow-Credentials', true)
  if (req.method === 'options') {
    res.send(200)
  } else {
    next()
  }
})
```

圖 16 修改後程式碼

## 5.2 實驗結果

### 5.2.1 網頁端

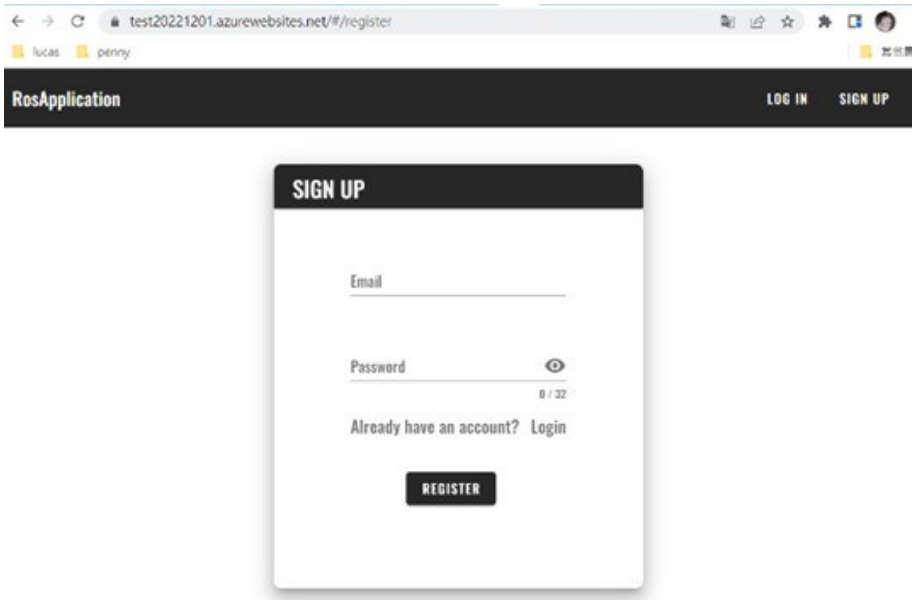


圖 17 sign up 頁面 Component

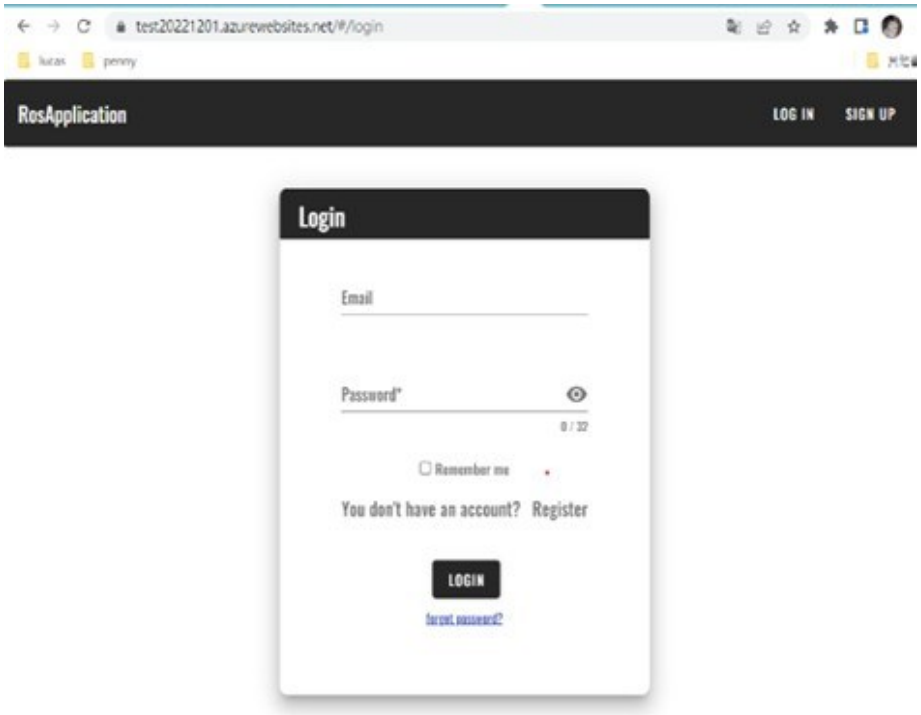


圖 18 Login 頁面 Component

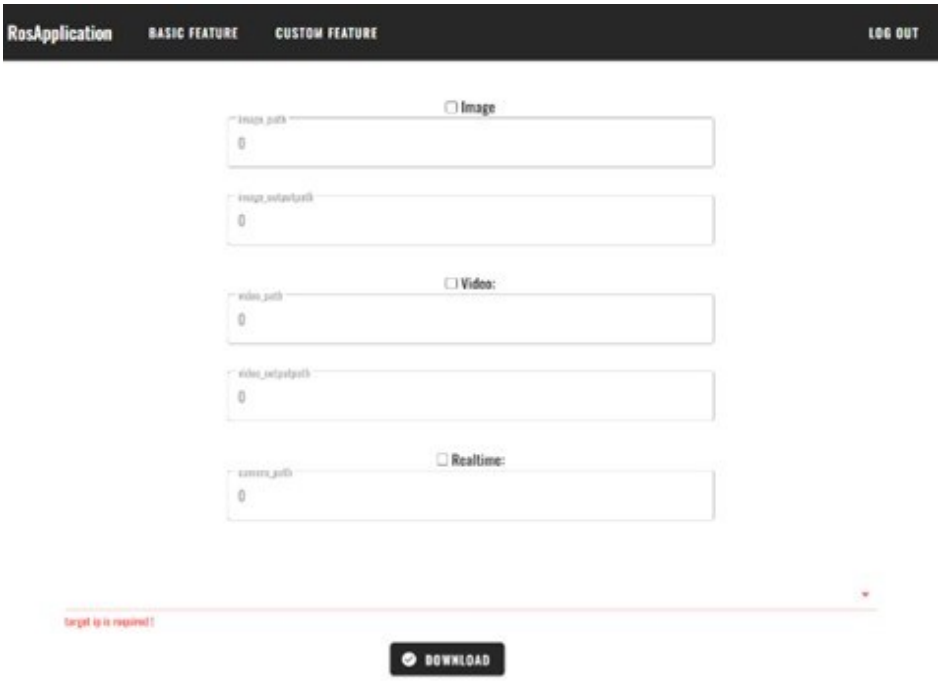


圖 19 download 介面


圖 20 custom 介面

### 5.2.2 設備端


圖 21 即時影像辨識畫面

# 第六章　結論與未來展望

本專題所提及的功能並不僅限於我們所提供的，其可延伸性極高。無論是它組專題或者日後各項可運用於機器人上的功能，都可透過此專題中運用的 Dockerfile 建置成 image 的方式進行快速且簡易的封裝。除去初次安裝於系統上的繁複過程及耗費的時間，日後普及到各設備或環境時所需的時間成本僅剩從 Docker hub 下載的時間。同時亦可建置私有的 image 倉庫去存放個人或是公司所需之 image，可逐漸演變成現在大家所習慣支援 ios 系統的的 APP store 或是 Andriod 系統上的 Play 商店，提供給使用者最快速、方便的操作方式。

# 第七章　參考文獻與網站


[1] https://ithelp.ithome.com.tw/articles/10238498

[2] https://medium.com/swlh/understand-dockerfile-dd11746ed183

[3] https://www.runoob.com/nodejs/nodejs-net-module.html

[4] https://kknews.cc/zh-tw/code/lj4rjye.html

[5] https://www.cnblogs.com/chyingp/p/6072338.html

[6] https://kknews.cc/zh-tw/code/lj4rjye.html

[7] https://ithelp.ithome.com.tw/articles/10208564

[8] https://progressbar.tw/posts/297

[9] https://hackmd.io/@angela1393aa/SyYRBrnGt

[10] https://ithelp.ithome.com.tw/m/articles/10298436

[11] https://blog.csdn.net/fuijiawei/article/details/122736214

[12] https://ithelp.ithome.com.tw/articles/10212268

[13] https://book.vue.tw/CH3/3-1-vue-cli-introduction.html

[14] https://reurl.cc/EXga9v