

系統CPU & Memory監控 Flask & Docker 實作

郭益華

目錄

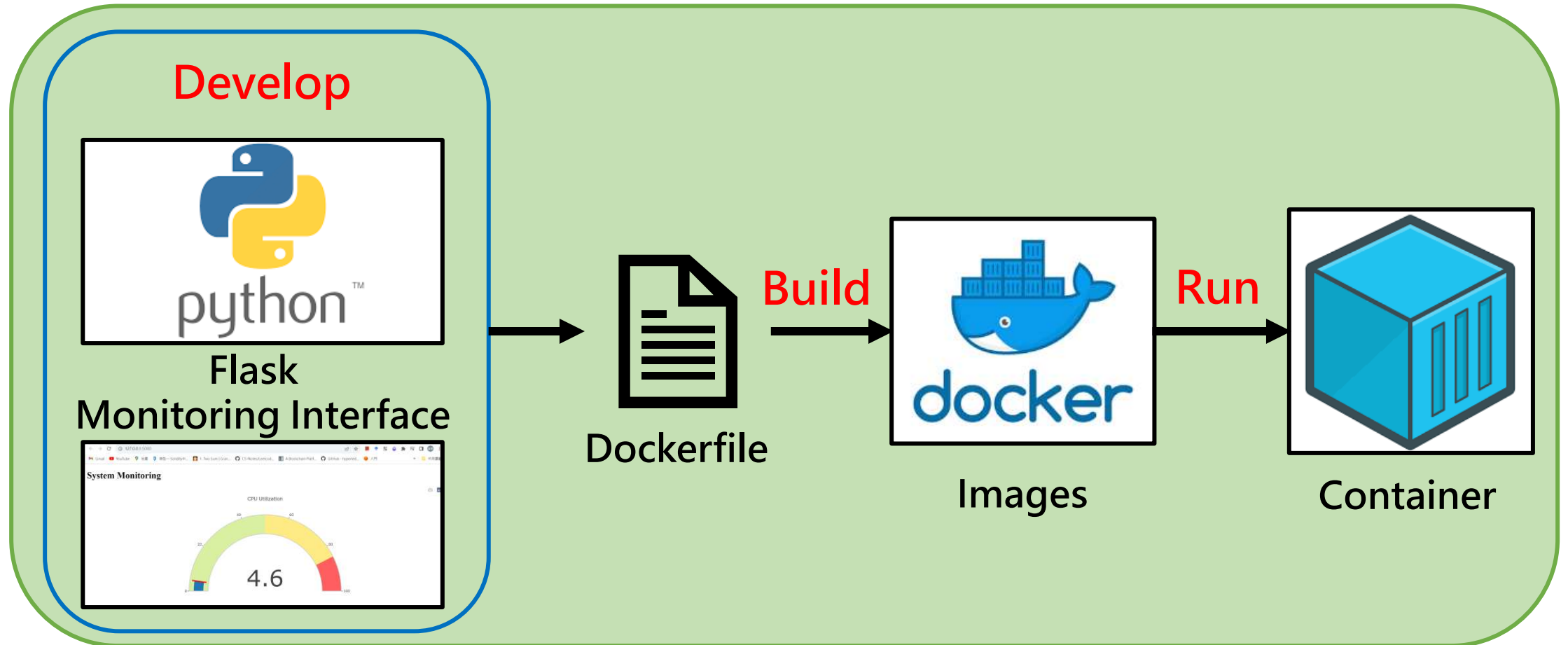
1. [簡介說明](#)
2. [使用Python-Flask 框架撰寫 系統 CPU & Memory 監控程式](#)
3. [撰寫Dockerfile](#)
4. [部署系統 CPU & Memory 監控程式至Docker image](#)
5. [啟動Docker container運作系統 CPU & Memory 監控程式](#)

1. 簡介說明

實作說明

- 使用Python Flask框架開發系統CPU & Memory監控應用程式
- 設定port 5000 於 Local端運作並執行Python應用程式
- 使用Docker將應用程式容器化
- 撰寫Dockerfile
- 使用Dockerfile建立容器
- 使用Docker image執行Docker容器

Flow

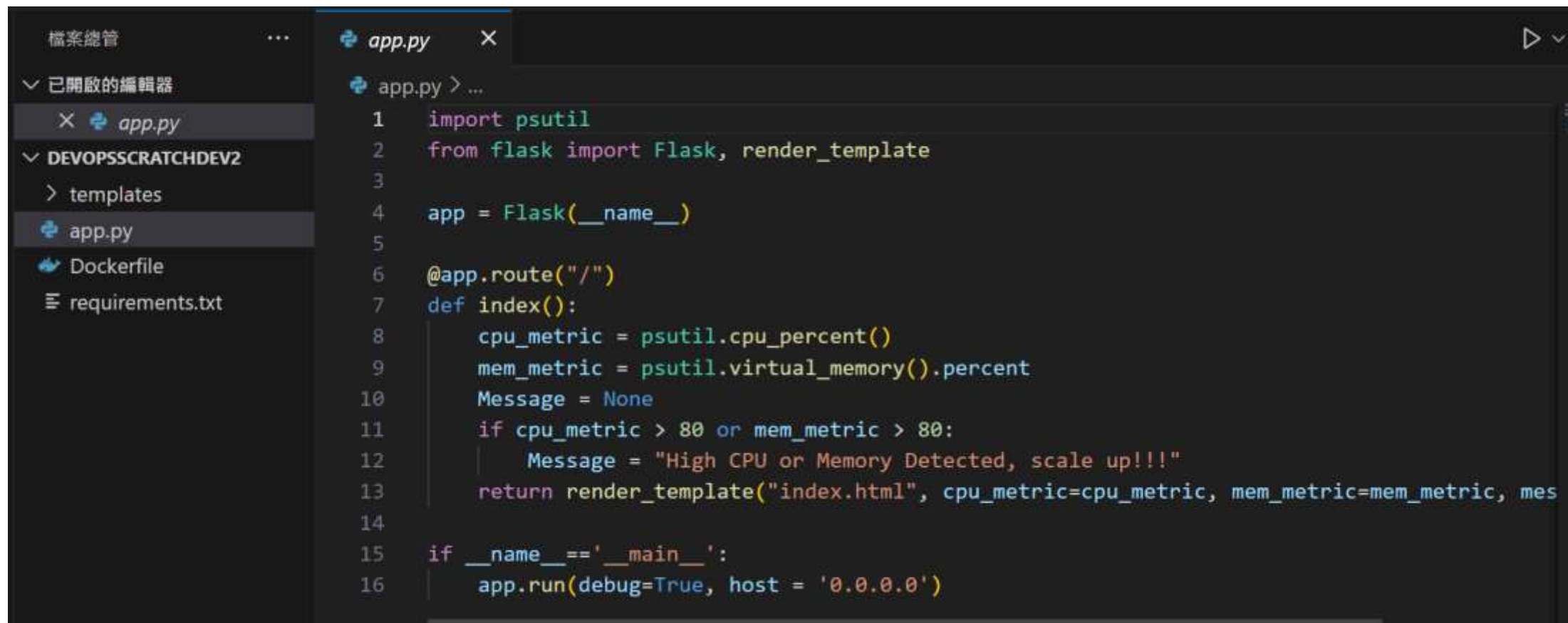


前置準備

- 套件安裝 `pip install -r requirement.txt`
 - Flask==2.2.3
 - MarkupSafe==2.1.2
 - Werkzeug==2.2.3
 - itsdangerous==2.1.2
 - psutil==5.8.0
 - plotly==5.5.0

2.使用Python-Flask 框架撰寫系統 CPU & Memory 監控程式

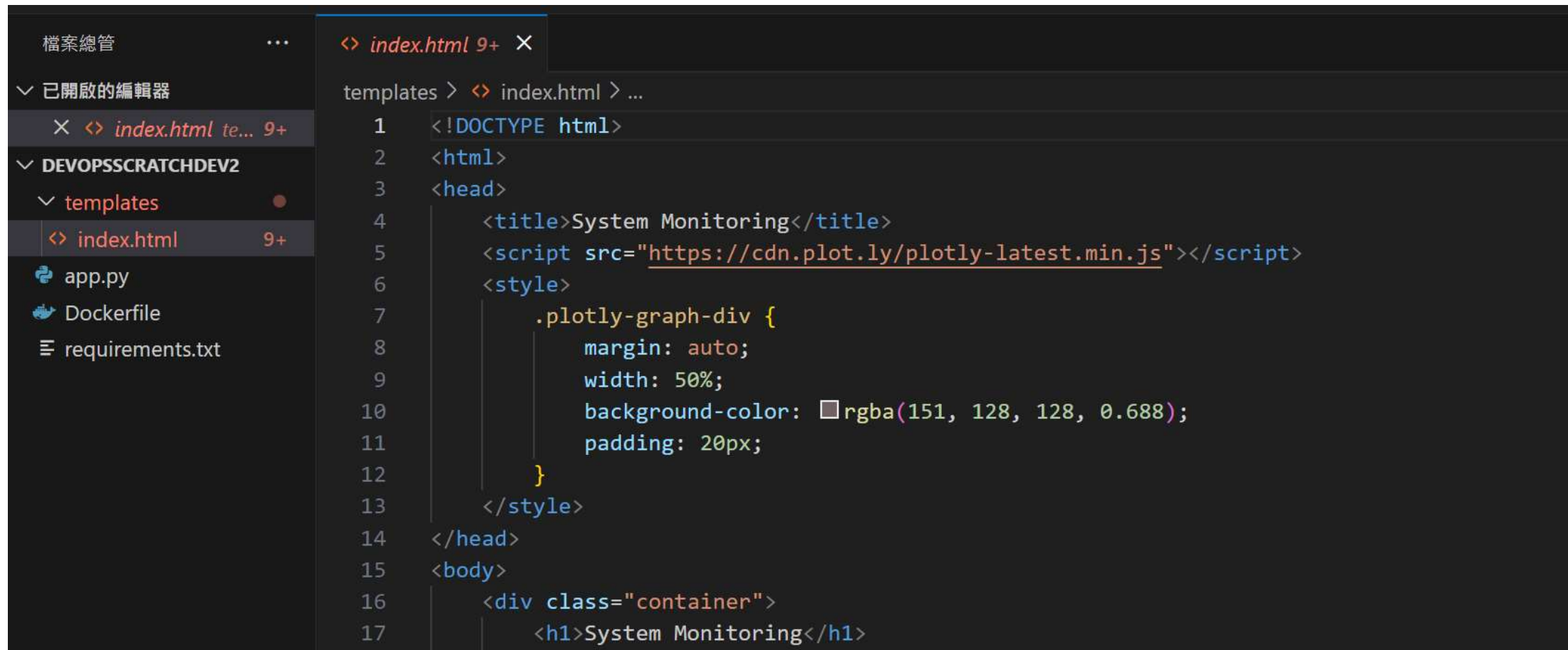
建立app.py並撰寫 cpu & memory監控



```
檔案總管  ...  app.py x
已開啟的編輯器
  x app.py
DEVOPSSCRATCHDEV2
  > templates
  app.py
  Dockerfile
  requirements.txt

1  import psutil
2  from flask import Flask, render_template
3
4  app = Flask(__name__)
5
6  @app.route("/")
7  def index():
8      cpu_metric = psutil.cpu_percent()
9      mem_metric = psutil.virtual_memory().percent
10     Message = None
11     if cpu_metric > 80 or mem_metric > 80:
12         Message = "High CPU or Memory Detected, scale up!!!"
13     return render_template("index.html", cpu_metric=cpu_metric, mem_metric=mem_metric, mes
14
15 if __name__ == '__main__':
16     app.run(debug=True, host = '0.0.0.0')
```


使用 plotly 套件 撰寫介面呈現畫面



```
<> index.html 9+ X
templates > <> index.html > ...
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>System Monitoring</title>
5   <script src="https://cdn.plot.ly/plotly-latest.min.js"></script>
6   <style>
7     .plotly-graph-div {
8       margin: auto;
9       width: 50%;
10      background-color: rgba(151, 128, 128, 0.688);
11      padding: 20px;
12    }
13  </style>
14 </head>
15 <body>
16   <div class="container">
17     <h1>System Monitoring</h1>
```

Local端應用程式測試

```
③ PS C:\Users\jerry\Desktop\mastercourse\dataEngineer\DevOpsScratch\DevOpsScratchDEV2> python app.py
* Serving Flask app 'app' (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: on
* Running on all addresses (0.0.0.0)
  WARNING: This is a development server. Do not use it in a production deployment.
```

點選此連結

```
* Debug mode: on
* Running on all addresses (0.0.0.0)
  WARNING: This is a development server. Do not use it in a production deployment.
* Running on http://127.0.0.1:5000
* Running on http://192.168.91.86:5000 (Press CTRL+C to quit)
* Restarting with watchdog (windowsapi)
* Debugger is active!
* Debugger PIN: 485-589-407
```

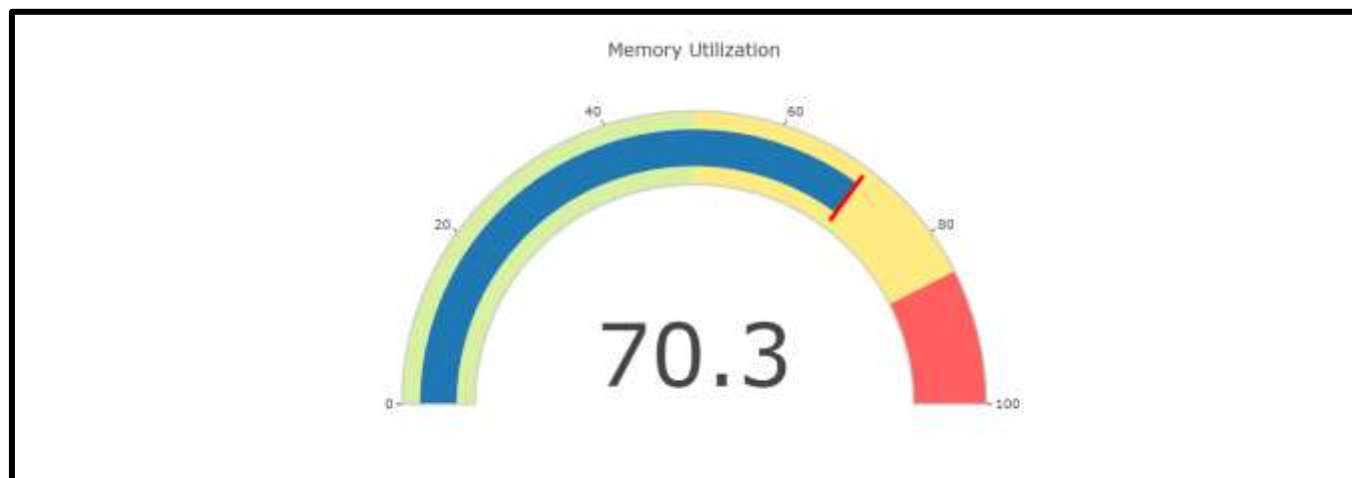
成功畫面

CPU



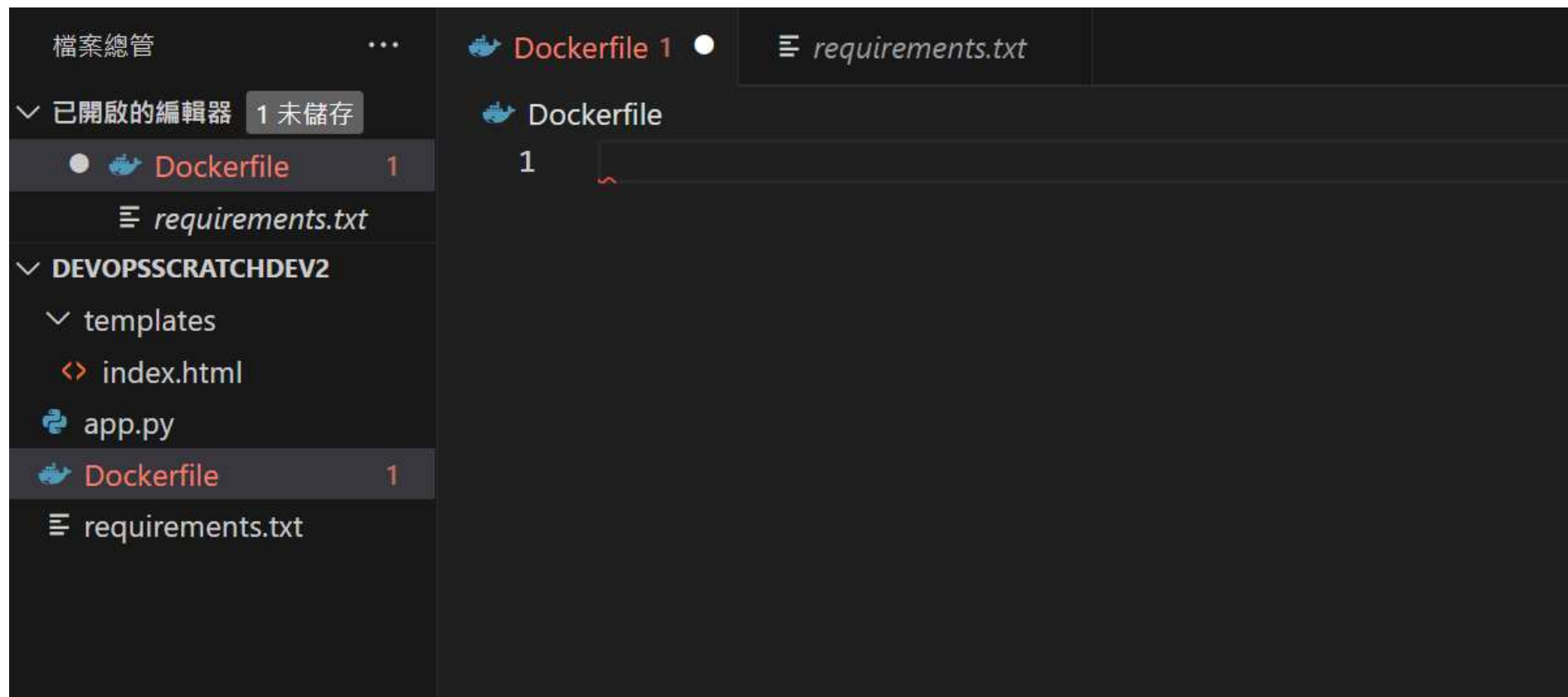
可看到自己電腦當前的CPU及Memory使用量

Memory



3. 撰寫Dockerfile

建立名為 Dockerfile 的檔案



撰寫Dockerfile

```
Dockerfile > ...
1  # 使用Docker官方的 python 基礎版 images
2  FROM python:3.9-buster
3
4  # 在容器中設定工作目錄
5  WORKDIR /app
6
7  # 複製requirements.txt
8  COPY requirements.txt .
9
10 # 安裝requirements.txt中的套件
11 RUN pip3 install --no-cache-dir -r requirements.txt
12
13 COPY . .
14
15 # 設定HOST為 0.0.0.0
16 ENV FLASK_RUN_HOST=0.0.0.0
17
18 # 設定port為5000
19 EXPOSE 5000
20
21 # 執行 flask run 命令
22 CMD ["flask", "run"]
```

FROM:

指定base image

WORKDIR:

指定docker執行起來時候的預設目錄位置

COPY:

從原本的開發資料夾將檔案複製到image中

RUN:

Dockerfile建立image內部再跑的指令

ENV:

設定環境

EXPOSE:

指定所有發布的port

CMD:

指定Instance啟動後所要執行的指令

4. 部署系統 CPU & Memory 監控程式至Docker image

將應用程式部署到Docker建立image

指令: Docker build -t <imageName>

```
● PS C:\Users\jerry\Desktop\mastercourse\dataEngineer\DevOpsScratch\DevOpsScratchDEV2> docker build -t my-flask -app .  
[+] Building 164.9s (10/10) FINISHED  
=> [internal] load build definition from Dockerfile 0.0s  
=> => transferring dockerfile: 431B 0.0s  
=> [internal] load .dockerignore 0.0s  
=> => transferring context: 2B 0.0s  
=> [internal] load metadata for docker.io/library/python:3.9-buster 2.2s  
=> [1/5] FROM docker.io/library/python:3.9-buster@sha256:bae5b428ebf32d01a902718b0a58874cbf33d7a4b6 126.5s  
=> => resolve docker.io/library/python:3.9-buster@sha256:bae5b428ebf32d01a902718b0a58874cbf33d7a4b6a6 0.0s  
=> => sha256:b1e7e053c9f6f57c6d95002167a6d57aed6aacf04dd2f8e681cb4f74a7ca4381 51.87MB / 51.87MB 58.7s  
=> => sha256:bae5b428ebf32d01a902718b0a58874cbf33d7a4b6a65b7cd7b21d48b0d2e2f1 988B / 988B 0.0s  
=> => sha256:191a38be55f72871359ebb522ab9621b872e67cac9b4b1834ccf1d2bce3a749d 2.01kB / 2.01kB 0.0s  
=> => sha256:3b1c264c0ad4598c25048a6dbd3030086cc5c74000e11d04ac27944cb116aabb 17.58MB / 17.58MB 14.9s  
=> => sha256:2646d4308aee325505bf575b2ecbafb8ef5088d6727585cb6e8215a1534b2472 7.51kB / 7.51kB 0.0s  
=> => sha256:ac8bb7e1a32398e26c129ce64e2ddc3e7ec6c34d93424b247f16049f5a91cff4 50.45MB / 50.45MB 56.9s
```


建立過程

```
=> => sha256:a2e1e233599c00054fb839db78b4d42e6f12f36b64280aa62d482a3ad0ad7109 191.88MB / 191.88MB 117.4s
=> => extracting sha256:ac8bb7e1a32398e26c129ce64e2ddc3e7ec6c34d93424b247f16049f5a91cff4 2.2s
=> => sha256:0ebfe287e9761b9b7dd1703470ff3473a62fe75238f3de01282165f8725968af 6.15MB / 6.15MB 62.1s
=> => sha256:e4e5f05940354b6c226635bb38623cb858415f00f6f856b25d064a5891f887f8 18.00MB / 18.00MB 73.1s
=> => extracting sha256:3b1c264c0ad4598c25048a6dbd3030086cc5c74000e11d04ac27944cb116aabb 0.5s
=> => extracting sha256:b1e7e053c9f6f57c6d95002167a6d57aed6aacf04dd2f8e681cb4f74a7ca4381 2.6s
=> => sha256:508cf9d1df6776c6d10cd077eb4546b9bf3bb774ed9c6051da6849961ea49fa2 242B / 242B 62.6s
=> => sha256:b0eda9c97fd4a2d42571513f0c205e1ae4276f12b3700716064327bdd7a93000 2.85MB / 2.85MB 65.5s
=> => extracting sha256:a2e1e233599c00054fb839db78b4d42e6f12f36b64280aa62d482a3ad0ad7109 7.4s
=> => extracting sha256:0ebfe287e9761b9b7dd1703470ff3473a62fe75238f3de01282165f8725968af 0.3s
=> => extracting sha256:e4e5f05940354b6c226635bb38623cb858415f00f6f856b25d064a5891f887f8 0.7s
=> => extracting sha256:508cf9d1df6776c6d10cd077eb4546b9bf3bb774ed9c6051da6849961ea49fa2 0.0s
=> => extracting sha256:b0eda9c97fd4a2d42571513f0c205e1ae4276f12b3700716064327bdd7a93000 0.2s
=> [internal] load build context 0.0s
=> => transferring context: 562B 0.0s
=> [2/5] WORKDIR /app 0.5s
=> [3/5] COPY requirements.txt . 0.0s
```

建立docker image完成

```
=> [4/5] RUN pip3 install --no-cache-dir -r requirements.txt 32.7s
=> [5/5] COPY . . 0.0s
=> exporting to image 2.9s
=> => exporting layers 2.9s
=> => writing image sha256:3af02193fbc7ae82c4ef65b25edc9c332c796f6b1f234a3e9f5480c42165a063 0.0s
=> => naming to docker.io/library/my-flask-app 0.0s
```

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them

```
PS C:\Users\jerry\Desktop\mastercourse\dataEngineer\DevOpsScratch\DevOpsScratchDEV2> █
```

查看image


```
PS C:\Users\jerry> docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
my-flask-app        latest             3af02193fbc7       24 seconds ago    1.09GB
sqlc/sqlc           latest             e5c3b6c57c08       10 days ago       98.2MB
postgres            12-alpine          57609cf5d1df       5 weeks ago       230MB
postgres            14-alpine          955d825eef13       5 weeks ago       235MB
PS C:\Users\jerry>
```

5. 啟動Docker container運作系統 CPU & Memory 監控程式

執行容器

指令: `docker run -p 5000:5000 <image ID or Repository Name >`

```
PS C:\Users\jerry\Desktop\mastercourse\dataEngineer\DevOpsScratch\DevOpsScratchDEV2> docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
my-flask-app   latest    3af02193fbc7   6 minutes ago  1.09GB
sqlc/sqlc      latest    e5c3b6c57c08   10 days ago    98.2MB
postgres       12-alpine 57609cf5d1df   5 weeks ago    230MB
postgres       14-alpine 955d825eef13   5 weeks ago    235MB
PS C:\Users\jerry\Desktop\mastercourse\dataEngineer\DevOpsScratch\DevOpsScratchDEV2> docker run -p 5000:5000 3af02193fbc7
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://172.17.0.2:5000
Press CTRL+C to quit
172.17.0.1 - - [17/Sep/2023 11:17:10] "GET / HTTP/1.1" 200 -
```



點選

`-p 5000:5000` = `-p` Local端port : docker前面所設定的port

成功部署到Docker並執行

System Monitoring

CPU Utilization



Memory Utilization



End