

【上課筆記 】【hiskio】 Docker 完全實作！結合 3 大語言掌握容器實務應用

上課講義

Docker 完全實作 - Dropbox Paper

volumes:

- type: volume
source: mydata
target: /data
volume:
nocopy: true
- type: bind
source: ./static
target: /opt/app/static

nocopy : 存在檔案不覆蓋

Docker 完全實作



讓我們先複習一下 Docker 的安裝

Windows 上的 Docker 環境建置

<https://docs.docker.com/desktop/windows/install/>

- 什麼是WSL？
- 適用於 Linux 的 Windows 子系統可讓開發人員執行 GNU/Linux 環境 (包括大部分的命令列工具、公用程式和應用程式)，直接在 Windows 上執行，不需進行修改，不會造成傳統虛擬機器或 dualboot 設定的額外負荷。
- <https://docs.microsoft.com/zh-tw/windows/wsl/about>

https://www.youtube.com/watch?v=IlkxbE_We1I&

https://youtu.be/IlkxbE_We1I

Mac 上的 Docker 環境建置

<https://docs.docker.com/desktop/mac/install/>

- 如果您的筆電是 Apple M1 (Silicon) CPU 的話，請參考此處的安裝方式：
- <https://docs.docker.com/desktop/mac/apple-silicon/>



Apple M1的限制

- 非所有映像檔都可用於 ARM64 架構。可以使用 `--platform linux/amd64` 模擬運行。但 mysql 不適用於 ARM64，不過可以使用 mariadb 解決此問題。
 - 模擬 Intel 的容器可能會有不穩定的狀況出現，除了速度比較慢之外，也會使用更多的記憶體。
 - 無法從容器內部 ping 到 Internet。要測試網絡，我們建議使用 curl 或 wget。
 - 當 TCP half-close 時，數據可能偶爾會丟失。
-

Linux 上的 Docker 環境建置

<https://docs.docker.com/engine/install/ubuntu/>

實務上需要注意的事



- 在大型組織（超過 250 名員工或超過 1000 萬美元的收入）中使用 Docker Desktop 需要付費訂閱 Docker。雖然此條款的生效日期是 2021 年 8 月 31 日，但對於需要付費訂閱的用戶，寬限期到 2022 年 1 月 31 日。

CPU/Memory資源

Preferences

General

Resources

ADVANCED

FILE SHARING

PROXIES

NETWORK

Docker Engine

Experimental Features

Kubernetes

Resources

Advanced

CPU: 2

Memory: 6.00 GB

Swap: 512 MB

Disk image size: 59.6 GB (5.3 GB used)

Cancel

Apply & Restart

掛載目錄

Preferences

General

Resources

ADVANCED

FILE SHARING

PROXIES

NETWORK

Docker Engine

Experimental Features

Kubernetes

Resources

File sharing

These directories (and their subdirectories) can be bind mounted into Docker containers. You can check the [documentation](#) for more details.

/Users

/Volumes

/private

/tmp

/var/folders

/path/to/exported/directory

⊖

⊖

⊖

⊖

⊖

⊕

Cancel

Apply & Restart

Buildkit

Preferences

General

Resources

Docker Engine

Experimental Features

Kubernetes

Docker Engine

v20.10.8

Configure the Docker daemon by typing a json Docker daemon [configuration](#) file.

This can prevent Docker from starting, reset your daemon settings if it hangs.

```
{  "builder": {    "gc": {      "enabled": true,      "defaultKeepStorage": "20GB"    }  },  "features": {    "buildkit": true  },  "experimental": false}
```

Cancel

Apply & Restart

回復初始設定

docker

Upgrade

macchiang

Troubleshoot

Restart Docker Desktop

All containers and settings will be preserved.

Restart

Support

Get help with Docker Desktop.

Get support

Reset Kubernetes cluster

All stacks and Kubernetes resources will be deleted.

Reset Kubernetes cluster

Clean / Purge data

This will solve problems with disk corruption, Docker Engine not booting...

Clean / Purge data

Reset to factory defaults

All settings and data will be removed.

Reset to factory defaults

Uninstall

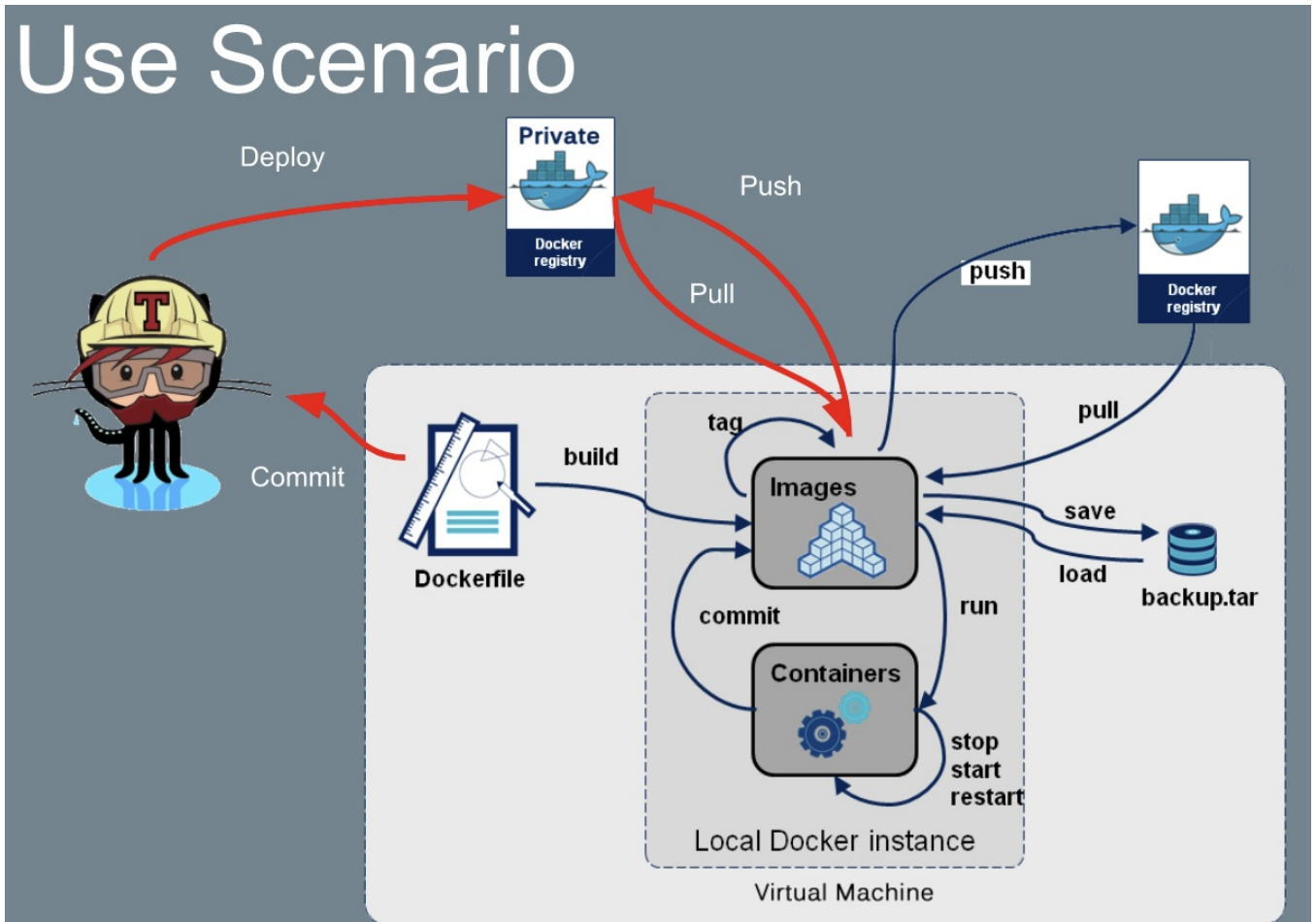
We're sorry to see you go. This completely uninstalls Docker Desktop.

Uninstall



工欲善其事，必先利其器：Docker 技術快覽

Docker 常用指令



執行容器

```
# 跑一個新的apache容器
docker run httpd:alpine
# 跑一個背景執行的apache容器，並取名為apache
docker run -d --name apache httpd:alpine
# 跑一個redis容器，啟動虛擬終端機，進入互動模式，在容器中執行sh
docker run -it --rm redis:alpine sh
```

- it : i指的就是互動模式，而 t就是指terminal的標示
- rm :指的是當容器結束後，將容器從系統移除

列出容器

```
# 列出所有執行中容器
docker ps
# 列出所有容器
docker ps -a
```

停止容器

```
# 停止apache 容器
docker stop apache
```

重新啟動容器

```
# 重新啟動apache
docker restart apache
```

Kill容器

```
#強制停止容器
docker kill apache
```

移除容器

```
#移除apache
docker rm apache
#檢查所有容器
docker ps -a
```

```
# 移除所有容器
docker rm -f $(docker ps -a -q)
```

複製檔案

```
# 建立一個hello.html
echo "hello" > hello.html
# 建立apache
docker run -d --name apache httpd:alpine
# 複製檔案至apache 容器中
docker cp hello.html apache:/usr/local/apache2/htdocs/
# 複製apache 容器中的檔案至本機
docker cp apache:/usr/local/apache2/htdocs/index.html ./index.html
```

在容器內執行程式

```
# 在容器中執行shell
docker exec -it apache sh
```

掛載目錄

```
# 把hello.html掛成index.html
docker run -d --name apache2 -v $PWD/hello.html:/usr/local/apache2/htdocs/index.html httpd:alpine
# copy出來驗證內容是否一樣
docker cp apache2:/usr/local/apache2/htdocs/index.html ./index.html
diff index.html hello.html
```

建立外部連線

```
# 把「本機的80埠」對應到「容器中的80埠」
docker run -d -v $PWD/hello.html:/usr/local/apache2/htdocs/index.html -p 80:80 httpd:alpine
```

<http://127.0.0.1:80>



- 如何從容器中，連到本機上的服務呢？
- 可以在容器中使用 `host.docker.internal` 來連接本機上的服務喔！

查看容器狀態

```
# 查看 CPU、記憶體與網路用量
docker stats
```

查看容器內的行程

```
# 查看容器內部的行程
docker top apache
docker top apache2
```

查看容器Logs

```
# 查看log
docker logs apache
# 持續查看
docker logs -f apache2
```

建立映像檔

```
# 於當前目錄，按Dockerfile.test中的指令，建立test:v1的映像檔
docker build . -f Dockerfile.test -t test:v1
```

拉取與推送映像檔

```
# 從docker hub下載 node的image
docker pull node:alpine
# 將下來的image，再給於新的標籤
docker tag node:alpine macchiang/node:alpine
# 推送新的標籤至私人倉庫
docker push macchiang/node:alpine
```

清除映像檔與容器

```
# 停掉全部container
docker stop $(docker ps -a -q)
# 刪除所有沒有使用的container與image
docker system prune
# 刪除所有映像檔
docker rmi $(docker images -a -q)
# 清除volume
docker volume rm $(docker volume ls)
# 刪除所有沒有使用的container、image及所有volume
docker system prune -a -f --volumes
```

Dockerfile 起手式

基本組成

```
# Base Image - Official Alpine
FROM alpine

LABEL "maintainer"="macchiang@gmail.com"

# Upgrade existing packages in the base image
RUN apk --no-cache upgrade
```

```
# Install apache from packages with out caching install files
RUN apk add --no-cache apache2

# Open port for httpd access
EXPOSE 80

# Run httpd in foreground
CMD ["-D","FOREGROUND"]

# Start httpd when container runs
ENTRYPOINT ["/usr/sbin/httpd"]
```



- **CMD vs. ENTRYPOINT**
 - CMD配置的是預設命令，是可以被複寫掉的。而ENTRYPOINT是一定會執行的命令。
 - 兩者共用的話，完整的命令會是ENTRYPOINT + CMD

建立docker image

```
docker build . -t apache:v1
```

複寫CMD

```
#直接複寫CMD，改為顯示版號
docker run --rm apache:v1 -v
```

ENV環境變數

```
FROM alpine
ENV NODE_ENV="development"
```

ARG傳入參數

```
FROM alpine
ARG NODE_ARG
ENV NODE_ENV="${NODE_ARG:-development}"
RUN echo "ARG=${NODE_ARG}, ENV=${NODE_ENV}"
CMD echo "ARG=${NODE_ARG}, ENV=${NODE_ENV}"
# 傳入NODE_ARG
DOCKER_BUILDKIT=0 docker build --build-arg NODE_ARG=staging .
# 傳入NODE_ARG
DOCKER_BUILDKIT=0 docker build .
```

COPY與ADD 複製文件

- 功能：將檔案複製到image中
- 差異：
 - COPY只能複製本機端的檔案或目錄
 - ADD可增加url的檔案到docker image (建議改用RUN curl/wget)
 - ADD可將本機端認可的壓縮格式tar解開後複製進去(要小心)
- 最佳實踐：盡量使用COPY而不是ADD

```
#複製package.json到/app下
COPY package.json /app/
#複製所有file開頭的檔案到/app下，並改變檔案所屬用戶與群組
COPY --chown=user:group file* /app/
```



- 若有些檔案不需要加入到image中，可以建立 .dockerignore 來忽略喔！

WORKDIR工作目錄

```
# hi.txt會在/目錄下
RUN echo "say hi" > hi.txt
WORKDIR /app
# world.txt會在/app的目錄下
RUN echo "hello" > world.txt
```

USER指定當前用戶

```
#新增group與user
RUN groupadd -r redis && useradd -r -g redis redis
#指定當前的user為redis
USER redis
#以redis的身份執行redis-server
RUN [ "redis-server" ]
```

HEALTHCHECK 健康檢查

```
FROM nginx
RUN apt-get update && apt-get install -y curl && rm -rf /var/lib/apt/lists/*
HEALTHCHECK --interval=5s --timeout=3s \
CMD curl -fs http://localhost/ || exit 1
```

打造最小的Docker映像檔

多階段建置Multi-stage Build

```
# Stage 1 建置環境
FROM alpine AS build
RUN echo "hello" > mytest

# Stage 2 執行環境
FROM alpine
COPY --from=build /mytest .
RUN cat /mytest
```

Distroless

<https://github.com/GoogleContainerTools/distroless>

<https://github.com/GoogleContainerTools/distroless/tree/main/examples/go>

main.go

```
package main

import "fmt"

func main() {
    fmt.Println("Hello, world!")
}
```

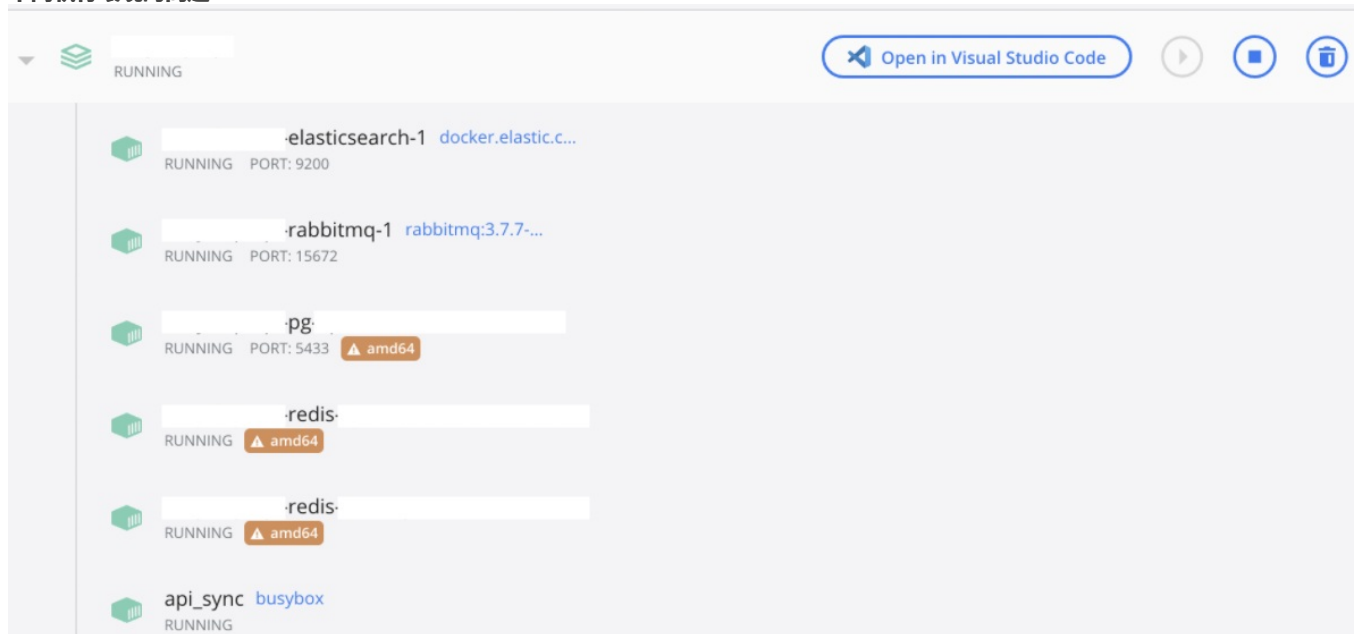
Dockerfile

```
FROM golang:1.12 as build-env
WORKDIR /go/src/app
COPY . /go/src/app
RUN go get -d -v ./...
RUN go build -o /go/bin/app

FROM gcr.io/distroless/base
COPY --from=build-env /go/bin/app /
CMD ["/app"]
```

建立多重系統架構的映像檔

不同執行環境的問題



使用Buildx建構多重系統架構的映像檔

<https://docs.docker.com/desktop/multi-arch/>

```
#確認目前的builder為何
docker buildx ls
#建立新的builder並切換過去
docker buildx create --name multiarch --use
```



若在linux環境中建立builder的話，需要額外跑qemu-user-static容器，並將driver設為docker-container

```
docker run --rm --privileged multiarch/qemu-user-static --reset -p yes
docker buildx create --name multiarch --driver docker-container --use
#檢查目前的builder
docker buildx inspect --bootstrap
```

```
#建立支援amd64與arm64的image，並且推送至registry
docker buildx build . --platform linux/amd64,linux/arm64 --push -t hello:v1
```

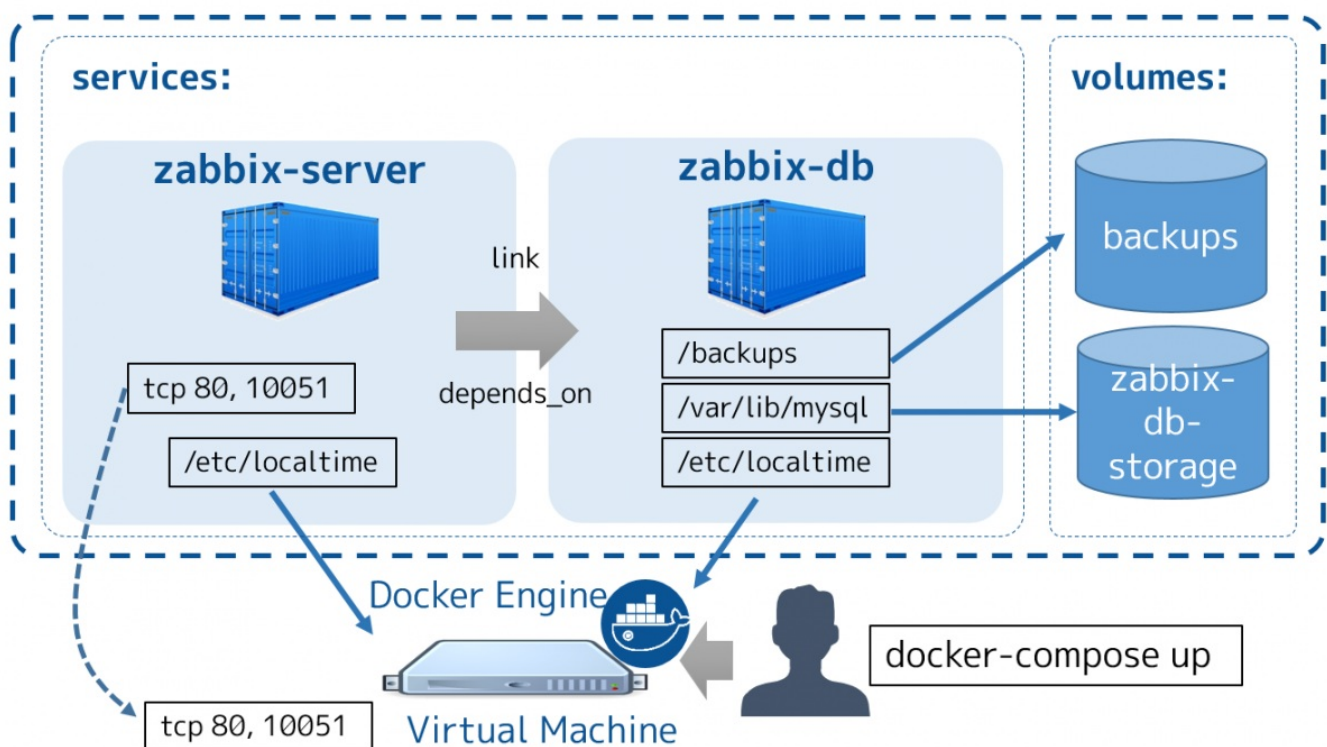


一次控制多個容器：Docker Compose

Docker Compose 的概念

- 定義和運行多個容器應用。
- 使用 YAML 文件來配置應用服務
- 使用單一命令從配置中創建和啟動所有服務
- 適用於所有環境：生產、Staging、開發、測試以及 CI 工作流

docker-compose.yml (v2 format)



Docker Compose 文件格式

基本格式

預設文件名稱為 `docker-compose.yml`

```
version: "3"
services:
  webapp:
    #指定image
    image: httpd:alpine
    ports:
      - "80:80"
    volumes:
      - "/data"
```

build 建立映像檔

```
version: '3'
services:
  webapp:
    #從Dockerfile-alternate建立映像檔
    build:
      context: .
```



```
dockerfile: Dockerfile-alternate
```

```
args:
```

```
buildno: 1
```

<https://docs.microsoft.com/zh-tw/visualstudio/docker/tutorials/use-docker-compose>

<https://docs.docker.com/compose/compose-file/compose-file-v3/>

Docker Compose 命令參考

<https://docs.docker.com/compose/reference/>

執行docker compose

```
#前景執行
```

```
docker-compose up
```

```
#背景執行
```

```
docker-compose up -d
```

停止/啟動docker compose

```
docker-compose stop
```

移除docker compose

```
docker-compose down
```

```
#也同步移除volume
```

```
docker-compose down -v
```

列出containers

```
docker-compose ps
```

監看logs

```
docker-compose logs
```

```
#follow logs
```

```
docker-compose logs -f
```

執行程式

```
docker-compose exec webapp sh
```

顯示執行程序

```
docker-compose top
```

Docker Compose 環境變數

environment

```
environment:
```

```
  RACK_ENV: development
```

```
  SHOW: 'true'
```

```
  SESSION_SECRET:
```

```
environment:
```

```
  - RACK_ENV=development
```

```
  - SHOW=true
```

```
  - SESSION_SECRET
```

env_file

```
.env
```

```
VAR=hello
```

```
MYENV=development
```

```
env file: .env
```

```
env file:
```

```
  - ./common.env
```

```
  - ./apps/web.env
```

```
  - /opt/runtime_opts.env
```

Docker Compose 健康檢查

```
healthcheck:
```

```
  test: ["CMD", "curl", "-f", "http://localhost"]
```

```
  interval: 1m30s
```

```
  timeout: 10s
```

```
  retries: 3
```

```
  start_period: 40s
```

Docker Compose 起動順序

```
version: "3"
```

```
services:
```

```
  web:
```

```
    build: .
```

```
ports:
  - "80:8000"
depends_on:
  - "db"
command: [". /wait-for-it.sh", "db:5432", "--", "python", "app.py"]
db:
  image: postgres
```



小工具

- [wait-for-it](#)
- [wait-for](#)

Docker Compose 磁碟掛載

```
Long syntax
version: "3.9"
services:
  web:
    image: nginx:alpine
    volumes:
      - type: volume
        source: mydata
        target: /data
        volume:
          nocopy: true
      - type: bind
        source: ./static
        target: /opt/app/static
  db:
    image: postgres:latest
    volumes:
      - "/var/run/postgres/postgres.sock:/var/run/postgres/postgres.sock"
      - "dbdata:/var/lib/postgresql/data"
volumes:
  mydata:
  dbdata:
```

```
Short syntax
volumes:
  # Just specify a path and let the Engine create a volume
  - /var/lib/mysql

  # Specify an absolute path mapping
  - /opt/data:/var/lib/mysql

  # Path on the host, relative to the Compose file
  - ./cache:/tmp/cache

  # User-relative path
  - ~/configs:/etc/configs/:ro

  # Named volume
  - datavolume:/var/lib/mysql
```

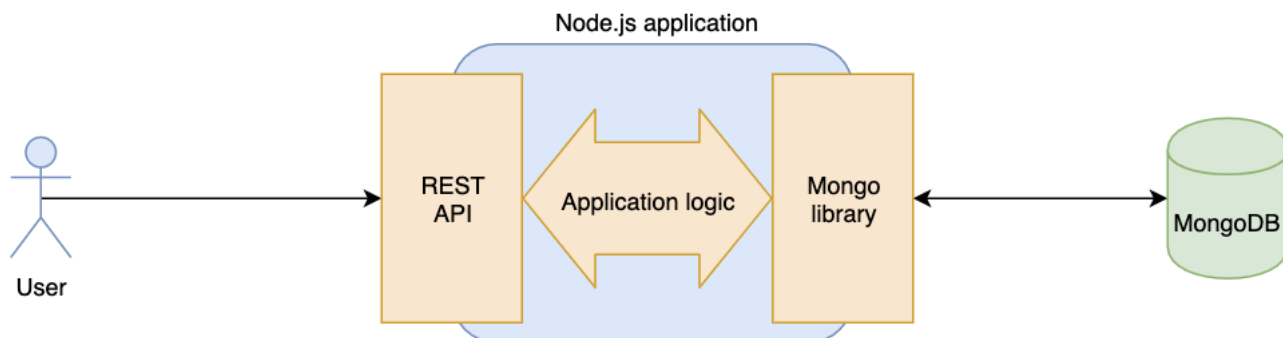


如何導入Docker

Docker導入流程

1. 確認應用程式使用的程式語言與版本
2. 參考相同程式語言的Dockerfile寫法
3. 列出應用程式相依的套件(包含系統、語言、字型等等)
4. 列出應用程式的設定(環境變數、設定檔)
5. 撰寫Dockerfile並測試
6. 檢查是否有檔案不該加入映像檔中(.dockerignore)
7. 列出應用程式相依的服務(Database、Redis)

實作：Node.js + MongoDB打造API服務



<https://github.com/macchiang/node-express-mongo-example>

透過 Dockerfile 部署 Node.js 環境

```
# Base image
FROM node:16-alpine

RUN apk upgrade --no-cache

# Create app directory
RUN mkdir -p /app
WORKDIR /app

# Install app dependencies
COPY package.json package-lock.json ./
RUN npm install

# copy app source
COPY . /app
```

```
# service port
EXPOSE 3000

CMD [ "node", "index.js" ]
```

Docker Compose YAML

```
version: "3.9"
services:
  express:
    build:
      context: ./
      dockerfile: Dockerfile
    image: express:latest
    ports:
      - 3000:3000
    depends_on:
      - mongo
  mongo:
    image: mongo:5
    ports:
      - 27017:27017
volumes:
  mongo_data:
```



作業-health check的妙用

- 請在Node.js + MongoDB的 Docker Compose YAML中加入health check，以確保當MongoDB啟動後，才啟動express



Docker X Python實作：快速打造

LINE Bot

透過 Dockerfile 建立 Python 環境

確認python的image與版本

https://hub.docker.com/_/python

了解Dockerfile

```
FROM python:3-slim-bullseye

WORKDIR /usr/src/app

COPY requirements.txt ./
RUN pip install --no-cache-dir -r requirements.txt

COPY . .

CMD [ "python", "./your-daemon-or-script.py" ]
```

Flask + Redis 打造計數器

<https://github.com/macchiang/hit-counter>

docker-compose.yml

```
version: '2'
services:
  myapp:
    build: .
    ports:
      - "80:5000"
  redis-lb:
    image: "redis:alpine"
```

構建 Django Web server

<https://github.com/macchiang/django-todolist>

<https://developer.mozilla.org/zh-TW/docs/Learn/Server-side/Django/Introduction>

先了解專案的安裝方式

Dockerfile

```
FROM python:3-slim-bullseye

WORKDIR /app

COPY requirements.txt ./
RUN pip install --no-cache-dir -r requirements.txt

COPY todo/* .
RUN python manage.py migrate
CMD [ "python", "manage.py", "runserver" ]
```

docker-compose.yml

```
version: '3.9'
services:
  myweb:
    build: .
    ports:
      - "8000:8000"
```

部署 LINE Bot 應用

<https://github.com/macchiang/line-bot-tutorial>

<https://github.com/line/line-bot-sdk-python/tree/master/examples/simple-server-echo>

在你開始之前

確保您具有以下內容：

- 擁有 Line 帳號

- 擁有 [Heroku](#) 帳戶（您可以免費創建一個）

建立LINE Bot 頻道

- [Messaging API 介紹](#)
- [開始使用 Messaging API](#)
- 取得 **Channel access token** 與 **Channel secret**

TOP > Hiskio > docker > Basic settings

Terms of use URL

optional

Edit

App types

Bot

Permissions ⓘ

PROFILE

Channel secret ⓘ

ba77855b76baa10262e884cab346e389 ⓘ

Issue

TOP > Hiskio > docker > Messaging API

Webhook URL ⓘ

Edit

LINE Official Account features

Edit the message text and other settings for these features in the LINE Official Account Manager

Allow bot to join group chats ⓘ

Disabled

Edit ⓘ

Auto-reply messages ⓘ

Enabled

Edit ⓘ

Greeting messages ⓘ

Enabled

Edit ⓘ

Channel access token

Channel access token (long-lived) ⓘ

ro1nKAYEFq2VkjRbXmIGcb0KuZL3J/casZpA9PKAfr8i7dLA5cJlwzAdGd1IqQcu2hW8HvXuJhcz81fApbjOHhS757exlp2R.Jj8smVHyZRGWIZRVfxZF9C34mPU3ysWUypBLJ9mWVKgxZN1zamWM1wdB04t89/1O/w1cDnyilFU= ⓘ

Reissue

- 關閉預設罐頭回覆訊息

Auto-reply messages ⓘ

Disabled

建立Docker Image

```
FROM python:3-slim-bullseye
WORKDIR /app
COPY requirements.txt ./
RUN pip install --no-cache-dir -r requirements.txt
COPY app.py .
CMD [ "python", "app.py" ]
docker build . -t line-bot:latest
```

安裝Heroku CLI

<https://devcenter.heroku.com/articles/heroku-cli#download-and-install>

建立Heroku App

```
#登入heroku
heroku login
#登入heroku registry
heroku container:login
#建立app
heroku create
#Creating app... done, ● morning-journey-43865
# https://morning-journey-43865.herokuapp.com/ | https://git.heroku.com/morning-journey-43865.git
```

```
#推送docker image至heroku registry
docker tag line-bot:latest registry.heroku.com/ morning-journey-43865 /web
docker push registry.heroku.com/ morning-journey-43865 /web

#設定環境變數
heroku config:set CHANNEL_ACCESS_TOKEN=xxxxxxxxx
heroku config:set CHANNEL_SECRET=xxxxxxxxx

#deploy docker image
heroku container:release web
設定Webhook
```

Webhook settings

Webhook URL ?

https://morning-journey-43865.herokuapp.com/callback

- ✓ Don't leave this empty
- ✓ Enter a valid HTTPS URL
- ✓ Enter no more than 500 characters

Update

Cancel

Webhook settings

Webhook URL ?

https://morning-journey-43865.herokuapp.com/callback

Verify

Edit

Use webhook ?



LINE Official Account features

Edit the message text and other settings for these features in the LINE Official Account Manager

Success

OK

Webhook settings

Webhook URL ?

https://morning-journey-43865.herokuapp.com/callback

Verify

Edit

Use webhook ?



<https://devcenter.heroku.com/articles/container-registry-and-runtime>

加入好友與測試



docker

Admin

Messaging API

Basic settings

Messaging API

LIFF

Security

Statistics

Roles

Messaging API settings

Bot information

Bot basic ID @053zupit

QR code



Scan this QR code with LINE to add your LINE Official Account as a friend. You can share the code with others.



Mac Chiang, 您好 🐼 !

感謝您成為docker的好友！

若不想接收提醒，可以點選面右上方的選單圖示，然後關閉「提醒」喔。

上午 8:22

已讀

上午 8:22

Hello



感謝您的訊息！

很抱歉，本帳號無法個別回覆用戶的訊息。

敬請期待我們下次發送的内容喔 🙏

上午 8:22

已讀

上午 8:23

Okay



感謝您的訊息！

很抱歉，本帳號無法個別回覆用戶的訊息。

敬請期待我們下次發送的内容喔 🙏

Okay

上午 8:23

已讀

上午 8:23

Thanks



感謝您的訊息！

很抱歉，本帳號無法個別回覆用戶的訊息。

敬請期待我們下次發送的内容喔 🙏

Thanks

上午 8:23



Debug

> heroku logs --tail

```
2021-11-09T15:17:06.932541+00:00 app[api]: Release v1 created by user macchiang@gmail.com
2021-11-09T15:17:06.932541+00:00 app[api]: Initial release by user macchiang@gmail.com
2021-11-09T15:17:07.532399+00:00 app[api]: Release v2 created by user macchiang@gmail.com
2021-11-09T15:17:07.532399+00:00 app[api]: Enable Logplex by user macchiang@gmail.com
2021-11-09T15:22:12.485507+00:00 app[api]: Deployed web (1be43e5135d6) by user macchiang@gmail.com
2021-11-09T15:22:12.485507+00:00 app[api]: Release v3 created by user macchiang@gmail.com
2021-11-09T15:22:12.503987+00:00 app[api]: Scaled to web@1:Free by user macchiang@gmail.com
2021-11-09T15:22:15.916499+00:00 heroku[web.1]: Starting process with command `python app.py`
2021-11-09T15:22:18.365293+00:00 app[web.1]: Traceback (most recent call last):
2021-11-09T15:22:18.365308+00:00 app[web.1]: File "/app/app.py", line 15, in <module>
2021-11-09T15:22:18.365410+00:00 app[web.1]: line bot api = LineBotApi(os.getenv('CHANNEL ACCESS TOKEN'))
2021-11-09T15:22:18.365411+00:00 app[web.1]: File "/usr/local/lib/python3.10/site-packages/linebot/api.py", line 63, in __init__
2021-11-09T15:22:18.365516+00:00 app[web.1]: 'Authorization': 'Bearer ' + channel access token,
2021-11-09T15:22:18.374292+00:00 app[web.1]: TypeError: can only concatenate str (not "NoneType") to str
2021-11-09T15:22:18.660220+00:00 heroku[web.1]: Process exited with status 1
2021-11-09T15:22:19.074969+00:00 heroku[web.1]: State changed from starting to crashed
2021-11-09T15:22:19.218365+00:00 heroku[web.1]: State changed from crashed to starting
2021-11-09T15:22:22.107500+00:00 heroku[web.1]: Starting process with command `python app.py`
2021-11-09T15:22:23.219784+00:00 app[web.1]: Traceback (most recent call last):
2021-11-09T15:22:23.219806+00:00 app[web.1]: File "/app/app.py", line 15, in <module>
2021-11-09T15:22:23.219930+00:00 app[web.1]: line bot api = LineBotApi(os.getenv('CHANNEL ACCESS TOKEN'))
2021-11-09T15:22:23.219934+00:00 app[web.1]: File "/usr/local/lib/python3.10/site-packages/linebot/api.py", line 63, in __init__
2021-11-09T15:22:23.219959+00:00 app[web.1]: 'Authorization': 'Bearer ' + channel access token,
2021-11-09T15:22:23.219961+00:00 app[web.1]: TypeError: can only concatenate str (not "NoneType") to str
2021-11-09T15:22:23.417290+00:00 heroku[web.1]: Process exited with status 1
2021-11-09T15:22:23.492672+00:00 heroku[web.1]: State changed from starting to crashed
2021-11-09T15:22:38.443725+00:00 heroku[router]: at=error code=H10 desc="App crashed" method=GET path="/" host=morning-journey-43865.herokuapp.com request_id=b91dba06-950e-4887-840e-60fc5986866f fwd="36.227.157.168" dyno= connect= service= status=503 bytes= protocol=https
2021-11-09T15:22:39.424038+00:00 heroku[router]: at=error code=H10 desc="App crashed" method=GET path="/favicon.ico" host=morning-journey-43865.herokuapp.com request_id=becc5a52-35a7-4557-b751-02d382a2b67e fwd="36.227.157.168" dyno= connect= service= status=503 bytes= protocol=https
```



Docker X PHP實作：架設一個

WordPress 網站

透過 Dockerfile 部署 PHP 環境

先從**Php官方Dockerfile**學起

<https://github.com/macchiang/php-hello-world>

Dockerfile

```
FROM php:7.3-apache
```

```
COPY index.php /var/www/html/
```

建立Docker image與Container

```
docker build . -t hello:latest
```

```
docker run -it --rm -p 8080:80 hello:latest
```

<https://www.astralweb.com.tw/what-is-differences-between-fastcgi-php-fpm/>

將 MySQL 加入部署行列

<https://github.com/macchiang/php-docker-simple>

docker-compose.yml

```
# Use root/example user/password credentials
```

```
version: '3.1'
```

```
services:
```

```
  php:
```

```
    build:
```

```
      context: .
```

```
      dockerfile: Dockerfile
```

```
    ports:
```

```
      - 80:80
```

```
    volumes:
```

```
      - ./src:/var/www/html/
```

```
  db:
```

```
    image: mysql
```

```
    command: --default-authentication-plugin=mysql_native_password
```

```
    restart: always
```

```
    environment:
```

```
      MYSQL_ROOT_PASSWORD: example
```

```
    volumes:
```

```
      - mysql-data:/var/lib/mysql
```

```
  adminer:
```

```
    image: adminer
```

```
    restart: always
```

```
    ports:
```

```
      - 8080:8080
```

```
volumes:
```

```
  mysql-data:
```

建立WordPress 網站

再由**WordPress官方Dockerfile**學起

```
#  
# NOTE: THIS DOCKERFILE IS GENERATED VIA "apply-templates.sh"  
#  
# PLEASE DO NOT EDIT IT DIRECTLY.  
#  
FROM php:7.3-apache  
  
# persistent dependencies  
RUN set -eux; \  
    apt-get update; \  
    apt-get install -y --no-install-recommends \  
# Ghostscript is required for rendering PDF previews  
    ghostscript \  
; \  

```

```

rm -rf /var/lib/apt/lists/*

# install the PHP extensions we need (https://make.wordpress.org/hosting/handbook/handbook/server-environment/#php-extensions)
RUN set -ex; \
    \
    savedAptMark="$(apt-mark showmanual)"; \
    \
    apt-get update; \
    apt-get install -y --no-install-recommends \
        libfreetype6-dev \
        libjpeg-dev \
        libmagickwand-dev \
        libpng-dev \
        libwebp-dev \
        libzip-dev \
    ; \
    \
    docker-php-ext-configure gd \
        --with-freetype-dir=/usr \
        --with-jpeg-dir=/usr \
        --with-png-dir=/usr \
        --with-webp-dir=/usr \
    ; \
    docker-php-ext-install -j "$(nproc)" \
        bcmath \
        exif \
        gd \
        mysqli \
        zip \
    ; \
    # https://pecl.php.net/package/imagick
    pecl install imagick-3.5.0; \
    docker-php-ext-enable imagick; \
    rm -r /tmp/pear; \
    \
    # reset apt-mark's "manual" list so that "purge --auto-remove" will remove all build dependencies
    apt-mark auto '.*' > /dev/null; \
    apt-mark manual $savedAptMark; \
    ldd "$(php -r 'echo ini_get("extension_dir");')/*.so \
        | awk '/=>/ { print $3 }' \
        | sort -u \
        | xargs -r dpkg-query -S \
        | cut -d: -f1 \
        | sort -u \
        | xargs -rt apt-mark manual; \
    \
    apt-get purge -y --auto-remove -o APT::AutoRemove::RecommendsImportant=false; \
    rm -rf /var/lib/apt/lists/*

# set recommended PHP.ini settings
# see https://secure.php.net/manual/en/opcache.installation.php
RUN set -eux; \
    docker-php-ext-enable opcache; \
    { \
        echo 'opcache.memory_consumption=128'; \
        echo 'opcache.interned_strings_buffer=8'; \
        echo 'opcache.max_accelerated_files=4000'; \
        echo 'opcache.revalidate_freq=2'; \
        echo 'opcache.fast_shutdown=1'; \
    } > /usr/local/etc/php/conf.d/opcache-recommended.ini
    # https://wordpress.org/support/article/editing-wp-config-php/#configure-error-logging
RUN { \
    # https://www.php.net/manual/en/errorfunc.constants.php
    # https://github.com/docker-library/wordpress/issues/420#issuecomment-517839670
    echo 'error_reporting = E_ERROR | E_WARNING | E_PARSE | E_CORE_ERROR | E_CORE_WARNING | E_COMPILE_ERROR |'
E_COMPILE_WARNING | E_RECOVERABLE_ERROR'; \
    echo 'display_errors = Off'; \
    echo 'display_startup_errors = Off'; \
    echo 'log_errors = On'; \
    echo 'error_log = /dev/stderr'; \
    echo 'log_errors_max_len = 1024'; \
    echo 'ignore_repeated_errors = On'; \
    echo 'ignore_repeated_source = Off'; \
    echo 'html_errors = Off'; \
    } > /usr/local/etc/php/conf.d/error-logging.ini

RUN set -eux; \
    a2enmod rewrite expires; \
    \
    # https://httpd.apache.org/docs/2.4/mod/mod_remoteip.html
    a2enmod remoteip; \

```

```

{ \
    echo 'RemoteIPHeader X-Forwarded-For'; \
# these IP ranges are reserved for "private" use and should thus *usually* be safe inside Docker
    echo 'RemoteIPTrustedProxy 10.0.0.0/8'; \
    echo 'RemoteIPTrustedProxy 172.16.0.0/12'; \
    echo 'RemoteIPTrustedProxy 192.168.0.0/16'; \
    echo 'RemoteIPTrustedProxy 169.254.0.0/16'; \
    echo 'RemoteIPTrustedProxy 127.0.0.0/8'; \
} > /etc/apache2/conf-available/remoteip.conf; \
a2enconf remoteip; \

# https://github.com/docker-library/wordpress/issues/383#issuecomment-507886512
# (replace all instances of "%h" with "%a" in LogFormat)
find /etc/apache2 -type f -name '*.conf' -exec sed -ri 's/([[:space:]])*LogFormat([[:space:]]+)"^"*)%h([^"]*)"\1%a\2/g' '{}' +

RUN set -eux; \
    version='5.8.2'; \
    sha1='c3b1b59553eafbf301c83b14c5eeae4cf1c86044'; \
    \
    curl -o wordpress.tar.gz -fL "https://wordpress.org/wordpress-$version.tar.gz"; \
    echo "$sha1 *wordpress.tar.gz" | sha1sum -c -; \
    \
# upstream tarballs include ./wordpress/ so this gives us /usr/src/wordpress
    tar -xzf wordpress.tar.gz -C /usr/src/; \
    rm wordpress.tar.gz; \
    \
# https://wordpress.org/support/article/htaccess/
    [ ! -e /usr/src/wordpress/.htaccess ]; \
    { \
        echo '# BEGIN WordPress'; \
        echo ''; \
        echo 'RewriteEngine On'; \
        echo 'RewriteRule .* - [E=HTTP_AUTHORIZATION:%{HTTP:Authorization}];'; \
        echo 'RewriteBase /'; \
        echo 'RewriteRule ^index\.php$ - [L]'; \
        echo 'RewriteCond %{REQUEST_FILENAME} !-f'; \
        echo 'RewriteCond %{REQUEST_FILENAME} !-d'; \
        echo 'RewriteRule . /index.php [L]'; \
        echo ''; \
        echo '# END WordPress'; \
    } > /usr/src/wordpress/.htaccess; \
    \
    chown -R www-data:www-data /usr/src/wordpress; \
# pre-create wp-content (and single-level children) for folks who want to bind-mount themes, etc so permissions are pre-created properly
instead of root:root
# wp-content/cache: https://github.com/docker-library/wordpress/issues/534#issuecomment-705733507
    mkdir wp-content; \
    for dir in /usr/src/wordpress/wp-content/*/ cache; do \
        dir="$(basename "$dir%/*")"; \
        mkdir "wp-content/$dir"; \
    done; \
    chown -R www-data:www-data wp-content; \
    chmod -R 777 wp-content

```

VOLUME /var/www/html

COPY --chown=www-data:www-data wp-config-docker.php /usr/src/wordpress/
COPY docker-entrypoint.sh /usr/local/bin/

ENTRYPOINT ["docker-entrypoint.sh"]
CMD ["apache2-foreground"]

<https://github.com/macchiang/wordpress-demo>

docker-compose.yaml

```

version: "3.3"
services:
  db:
    image: mysql
    volumes:
      - db_data:/var/lib/mysql
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: somewordpress
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wordpress
  wordpress:
    depends_on:
      - db
    image: wordpress
    ports:
      - "8000:80"
    restart: always

```

```
environment:
  WORDPRESS_DB_HOST: db:3306
  WORDPRESS_DB_USER: wordpress
  WORDPRESS_DB_PASSWORD: wordpress
volumes:
  db_data:
```



Docker X Node.js實作：打造前後端具備的 To-Do List

建立Express + Nginx組合

<https://github.com/macchiang/express-nginx>

Nginx config

```
server {
  listen 80 default_server;
  listen [::]:80 default_server;
  server_tokens off;
  client_max_body_size 15M;
  if ($http_x_forwarded_proto = "http") {
    return 301 https://$host$request_uri;
  }
  location / {
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header Host $http_host;
    proxy_set_header X-NginX-Proxy true;
    proxy_pass http://node:3000;
    proxy_redirect off;
  }
}
```

docker-compose.yml

```
version: "3"
services:
  nginx:
    image: nginx:alpine
    volumes:
      - ./nginx:/etc/nginx/conf.d
    links:
      - node
    ports:
      - 80:80
  node:
    build:
      context: .
    image: express-nginx:latest
```

打造To-Do List

<https://github.com/macchiang/express-mongoDB-ToDoList>

Dockerfile

```
# Base image
FROM node:16-alpine
RUN apk upgrade --no-cache
# Create app directory
RUN mkdir -p /app
WORKDIR /app
# Install app dependencies
COPY package.json package-lock.json ./
RUN npm install
# copy app source
COPY . /app
# service port
EXPOSE 3000
CMD [ "node", "app.js"]
```

docker-compose.yml

```
version: "3"
services:
  nginx:
    image: nginx:alpine
    volumes:
      - ./nginx:/etc/nginx/conf.d
    links:
      - node
    ports:
      - 80:80
  node:
    build:
      context: .
    depends on:
      - mongo
  mongo:
    image: mongo:5
    ports:
      - 27017:27017
    volumes:
      - mongo_data:/data/db
volumes:
  mongo_data:
```



參考資料

- <https://blog.gtwang.org/linux/docker-commands-and-container-management-tutorial/>
- <https://hub.docker.com/r/broadtech/alpine-apache2/dockerfile>
- <https://ithelp.ithome.com.tw/articles/10251549>
- <https://www.tpisoftware.com/tpu/articleDetails/1216>
- https://yeasy.gitbook.io/docker_practice
- <https://blog.techbridge.cc/2018/09/07/docker-compose-tutorial-intro/>
- https://yeasy.gitbook.io/docker_practice/compose/usage
- <https://dev.to/truthseekers/setup-a-basic-local-php-development-environment-in-docker-kod>
- <https://dotblogs.com.tw/exploosion/2018/09/15/194754#2>
- https://dca.gitbooks.io/nodejs-tw-wiki-book/content/book/todo_list/todo_list.html

推薦學習

<https://dev.to/ankit01oss/7-github-projects-to-supercharge-your-docker-practices-2i80>

🕒 修訂版本 #11

★ 由 treeman 建立於 18 🕒 @🕒🕒🕒 2022 18:15:03

✍ 由 treeman 更新於 5 🕒🕒🕒🕒 2023 10:14:58