**Docker**

是一個用於管理容器化應用程式的工具，它提供了一個平台，讓開發人員可以將應用程式及其相依的環境打包成輕量、可移植的容器，並在不同的環境中運行這些容器。

Server端

1. 編寫Dockerfile，加入需要的應用程式和服務
2. 利用此Dockerfile 進行Build Docker Image
3. 將此Docker Image傳給要部署的Client端或主機上
4. 可透過Repository上傳下載Image

Client 端

1. 找到/下載該Image 並執行 run以建立Container
2. 開啟/暫停Container 以啟動容器內相關服務，如Web 應用程式、資料庫
3. 若對 Web 應用程式進行了擴充或調整，需要重新封裝 Docker Image並再次傳給用戶端
4. 若使用Docker 架設網站，可將 Web 應用程式與 IIS 等相關服務一起build到 Image中。客戶端只需要運行這個 Docker 容器，便可擁有一個包含網站和相關服務的完整環境。

Build範例

在當前目錄中找到名為 Dockerfile 的文件並使用它來構建 Docker Image。 **<image\_name>** 映像指定的名稱。句點**.**表示當前目錄

docker build -t <image\_name> .

**Docker-Compose**

把 Docker 的啟動方法、環境變數、容器間的關係寫成成配置文件，並利用此文件簡單的啟動所有容器

version: "3.5": 選定 docker-compose 的版本，每個版本提供的 API 方法有所差異。

services: 此欄位底下會有所有的容器，以下分別有server與db兩個 容器。

build: 說明此容器要使用特定 Dockerfile 來 build，context為檔案目錄，dockerfile為 Dockerfile 的名字。

working\_dir: 指定 docker 啟動時所在的目錄，如果目錄不存在會自動創建一個。

volumes: 將本機檔案掛載至 docker 內部，本機檔案更新時 docker 內部的掛載檔案也會更新。

ports: 將本機的 port 做 mapping 與 docker 內部的 poart。

depends\_on: 說明 a 容器與 b 容器有相關，會等到 b 容器啟動完畢後，再啟動 a 容器。

entrypoint: 指定 docker 啟動時的預設指令。

restart: 當容器不正常關閉時，會重新啟動容器。

image: 如果不使用 Dockerfile 來建立容器，你可以直接使用 docker image 來啟動容器。

environment: 指定容器內的環境變數。

由於entrypoint中指定了go run main.go，所以我們也必須創建一個main.go檔，才會讓 docker 啟動時有檔案執行

應用範例：

---將 docker 內部需要 build 的容器都 build

$ docker-compose build

---build 完畢後啟動

$ docker-compose up