

Docker Tutorial

CathayLife DevOps Team

Outline

- Requirements
- Docker
- Container
- Tips
- More About Container
- Appendix
- Reference

Requirements

1. 準備可執行 Docker 的環境

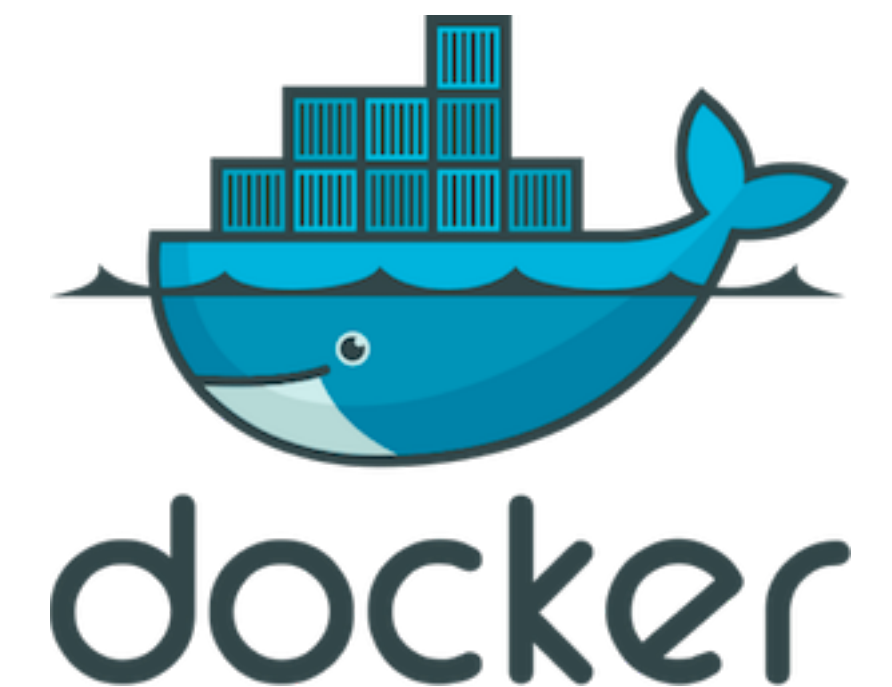
- GCP Cloud Shell
 - 免費使用，使用 Google 帳號登入
 - 配有 VS Code 與 Docker
 - 每週可用 50 小時，閒置一小時自動關閉，5 GB Disk
 - <https://shell.cloud.google.com/>
- 本機
 - Mac/Windows: Docker Desktop
 - Linux: Docker Engines, docker-compose

2. `git clone https://github.com/cathaylife-devops/docker-tutorial.git`

Docker

Docker

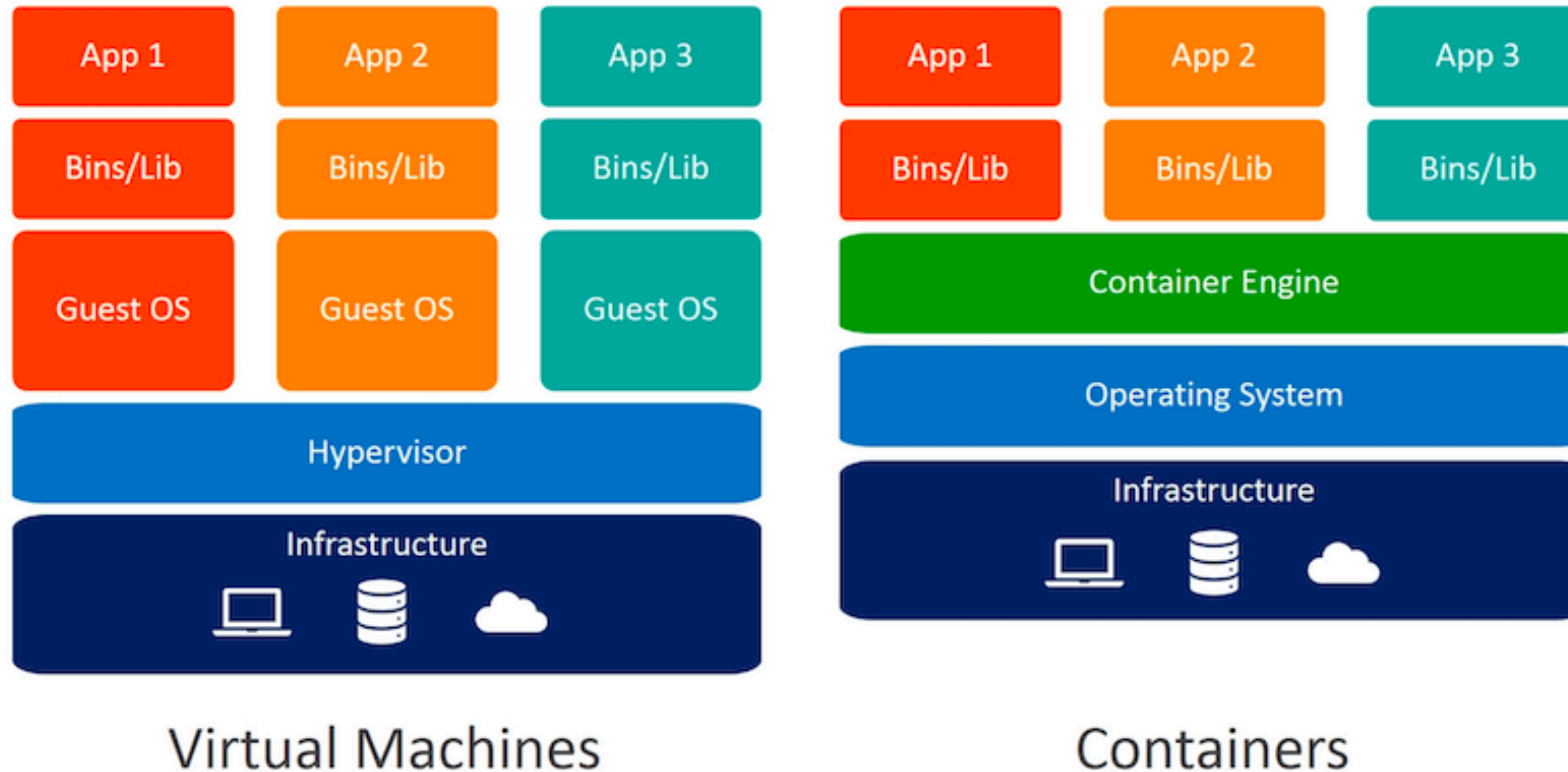
- [Docker](#)
 - [YouTube - Docker in 100 seconds](#)
 - Container 是 **OS-level Virtualization**
 - Docker 實作了 Container (容器) 的概念
 - Container 跟 VM 相比啟動更快、可攜性更高、硬體使用量更低
 - 快速但不完全正確的理解方式，Container 就是的**更快**、**更小**、**更便宜**的 VM



Docker

- Fun Fact
 - Container 的概念在 1979 年就在 Unix 上實作 (chroot)
 - Docker 的開發商 Docker, Inc (原名 dotCloud, Inc) 原本是雲端服務供應商，代管使用者建立的 Web、DB 等。隨著數量的增加，以 Hypervisor 的 VM 為單位切割服務不利管理與資源分配，因此重新包裝了當時流行的 Container 技術 LXC 成 Docker 於內部使用
 - Docker 在 2013/3/13 由 dotCloud, Inc 開源發佈初版，同年 10/29 docCloud, Inc 改名為 Docker, Inc

VM vs Container

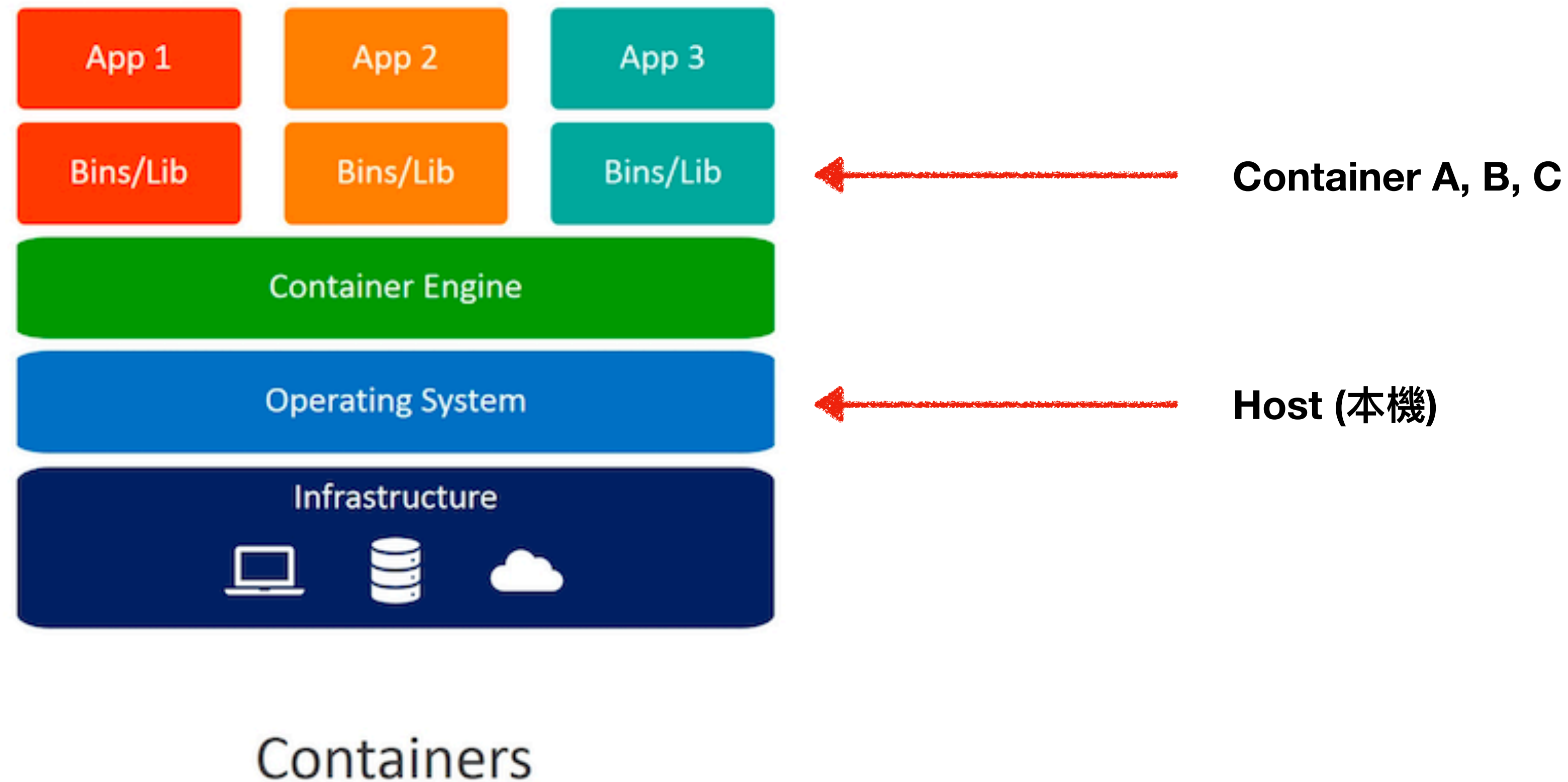


VM: Hardware-Level Virtualization

Container: OS-Level Virtualization

Image Source: [Weaveworks](#)

Host and Container



在使用 Container 的情境時 Host 指的是安裝 Container(Docker) Engine 的機器
Host 與 **Container** 可以視為是**完全不同的機器**，在操作時需注意是要在 Host 還是 Container 中

Docker

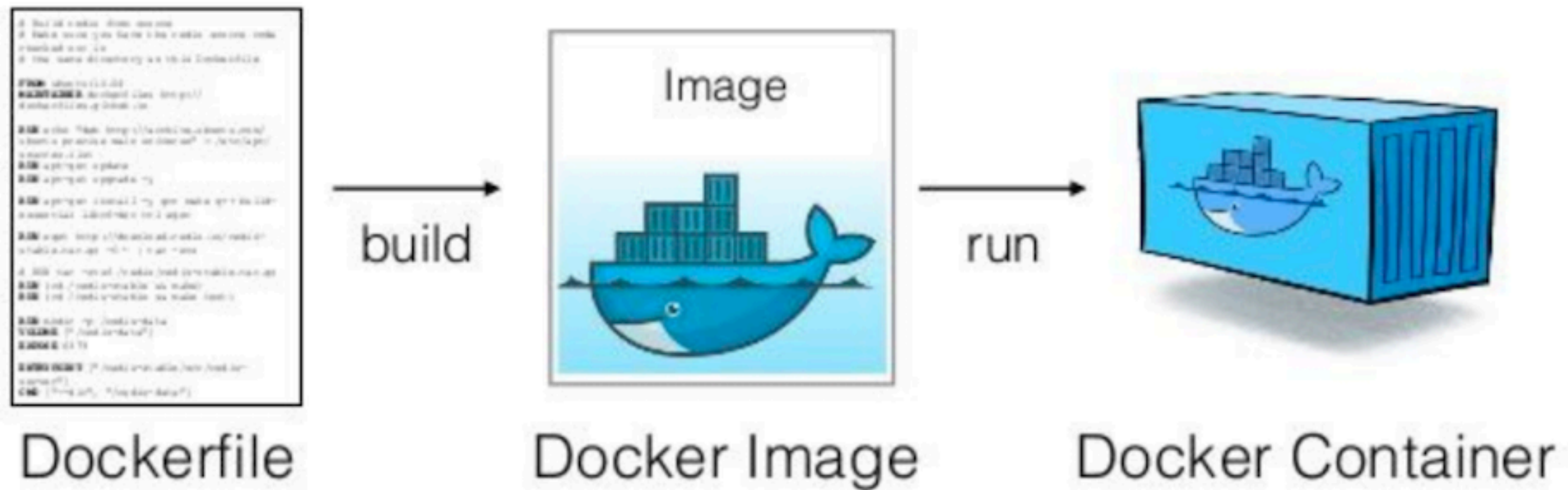


Image Source: [Build a Docker Image just like how you would configure a VM](#)

Hands-on

- Github Repo [docker-tutorial](#)
 - docker-101/0-run-container

Docker Command

- `docker run [image name]:[tag name]`
 - 指定一個 Image 啟動 Container，不指定 tag 時自動使用 latest tag
- `docker images`
 - 列出本機已下載的 Image
 - 欄位說明
 - REPOSITORY - Image 名稱
 - TAG - Image 的版本，同個名稱可以有多个 tag
 - IMAGE ID - Image 的 Unique ID，值為 Image 的 JSON config SHA256 Hash
 - CREATED - Image 打包的時間
 - SIZE - Image 的大小

Docker Command

- `docker ps -a`
 - 列出本機所有的 Container，沒有 `-a` 時只會列出狀態為 `running` 的 Container
 - 欄位說明
 - CONTAINER ID - Container 的 Unique ID
 - IMAGE - Container 使用的 Image
 - COMMAND - Container 啟動時執行的 Command
 - CREATED - Container 已運行時間
 - STATUS - Container 狀態，常見有下面三個
 - `restarting` - Restarting (times) xxx minutes ago
 - `running` - Up xxx minutes
 - `exited` - Exited (exit code) xxx minutes ago
 - PORTS - Container 與 Host 的 port proxy
 - NAMES - 建立 Container 未指定名稱時，預設為兩個隨機單字以底線組合作為名稱

Container Registry / Repository

- Container Registry (Repository) 可以上傳與下載 Container Image
- SaaS
 - [Docker Hub](#)
 - Docker 預設的 Container Registry，由 Docker, Inc 維運
 - Docker Community、開源組織、軟體公司也會發布各種的 Image 供大家下載，如 [Python](#)、[PostgreSQL](#)、[SonarQube](#)
 - GitLab Registry - 建立一個 Repository 後可以上傳
 - Github Packages - 建立一個 Repository 後可以上傳
- Self-hosted Container Registry
 - [Nexus](#) - 除 Container Registry 外，也可管理 Maven, Pypi, npm 等多種 Package Repository
 - [Harbor](#) - 專業的 Container Registry，支援更多 Image 權限控管



Docker Desktop

- Windows 或 Mac 在安裝的時候通常都會使用 Docker Desktop
 - Docker Desktop 安裝時會設定 Linux 的 VM，實際是將 Docker 安裝在 Linux 上
 - Windows
 - with WSL2，可以直接安裝在 Windows 的 WSL 中的 Linux 系統上
 - with WSL，安裝在以 Windows 內建的 hyper-v VM 建立的 LinuxKit VM
 - 更多詳細說明可參考[在 WSL 2 上開始使用 Docker 遠端容器](#)
 - Mac
 - 安裝在 LinuxKit VM 上
 - 安裝 Docker Desktop 同時安裝了 Docker Engine, Docker CLI client, Docker Compose 等工具
- 部分設定需要透過 Docker Desktop 設定
 - Volume、Network
 - VM Resource: CPU, Memory
- Docker Desktop 在 2022 年 1 月開始，只免費授權給中小企業、私人使用或教育使用，或是非營利開源專案使用，企業規模人數超過了250人，或是營收超過1千萬美元的大型企業，得購買付費授權

Hands-on

- Github Repo [docker-tutorial](#)
 - docker-101/1-more-container

Docker Fundamentals

- Container 的工作就是把他負責的 Process 執行完畢
 - Process 執行完畢 Container 狀態會變為 Exited
- Volume
 - Container 關掉後東西是會不見的，如果有資料要保存必須存入於掛載的 Volume 中
- docker run 常用 options
 - -- name 給定 Container 的名字
 - -v 掛載 Volume
 - -it 以互動模式執行虛擬終端機
 - --rm 執行完畢後刪除 Container
 - -d 以背景執行方式 ([Daemon](#)) 啟動 Container

Dockerfile

- Dockerfile 是 Image 的 DNA，決定 Image 長怎樣
- Dockerfile 基礎架構，更多請參考 [Docker Doc](#)
 - FROM - 基底
 - COPY - 從 docker build 的環境中複製檔案至 Image 中
 - RUN - 在基底上再多做一些事情
 - CMD - Container 啟動時會執行的 Command
- Dockerfile 參考範例
 - <https://github.com/ufoym/deepo/blob/master/docker/Dockerfile.all-py36-cpu>

Dockerfile



Dockerfile

```
FROM ubuntu:18.04
```

基底是 ubuntu 18.04

```
WORKDIR /app
```

將 Image 的初始目錄設定為 /app，目錄不存在則自動新建一個

```
COPY ./hello.txt /app
```

複製執行 docker run 當下路徑的 hello.txt 至 /app 目錄中

```
RUN apt-get update && \
    apt-get -y install sl
```

在 Image 中執行一些指令
此處為更新 Package 清單，並安裝 [sl package](#)

```
CMD cat /app/hello.txt
```

Container 啟動時執行 cat /app/hello.txt 指令

Hands-on

- Github Repo [docker-tutorial](#)
 - docker-101/2-build-image

Recap

- Docker 的三個關鍵概念
 - **Dockerfile** - 定義 Image 長怎樣
 - (Docker/Container) **Image** - 由 Dockerfile build 成，可發佈至 Container Registry 供其他機器下載
 - **Container** - 由 Image run 成，實際運行的實體
- Docker command
 - build - 將 Dockerfile 打包成 Image
 - run - 將 Image 執行成 Container
 - images - 列出本機的 Image
 - ps -a - 列出本機所有的 Container
 - logs - 印出 Container 的 Log
 - cp - Host 與 Container 之間的檔案交換
 - exec - 在執行中的 Container 執行指令

好的 Container 只做一件事，並把他做好

Container

Container - Orchestration

orchestration noun



or·ches·tra·tion | \,ôr-kə-'strā-shən \

Definition of *orchestration*

- 1 : the arrangement of a musical composition for performance by an [orchestra](#)
also : [orchestral](#) treatment of a musical composition
- 2 : harmonious organization
// develop a world community through *orchestration* of cultural diversities
— L. K. Frank



Image Source: [Power Music School](#)

Container - Orchestration

英文

↔

中文（繁體）

orchestration
ˌɔrkəˈstrəʃən

×

編排
Biānpái

「orchestration」的翻譯

名詞

管弦樂編曲
orchestration

管弦樂作曲法
orchestration

在 Google 翻譯中開啟 • 意見回饋





協作
電腦

協作是對電腦系統和軟體的自動化組態、協調和管理。
[維基百科](#)

意見回饋

Container - Orchestration

- Container Orchestration
 - Container 的**管理與調度**
 - 當一個服務需要多個 Process 時，例如 Application 的 Process 與 DB 的 Process，最佳的做法是一個 Container 負責一個 Process



Image Source: [Power Music School](#)

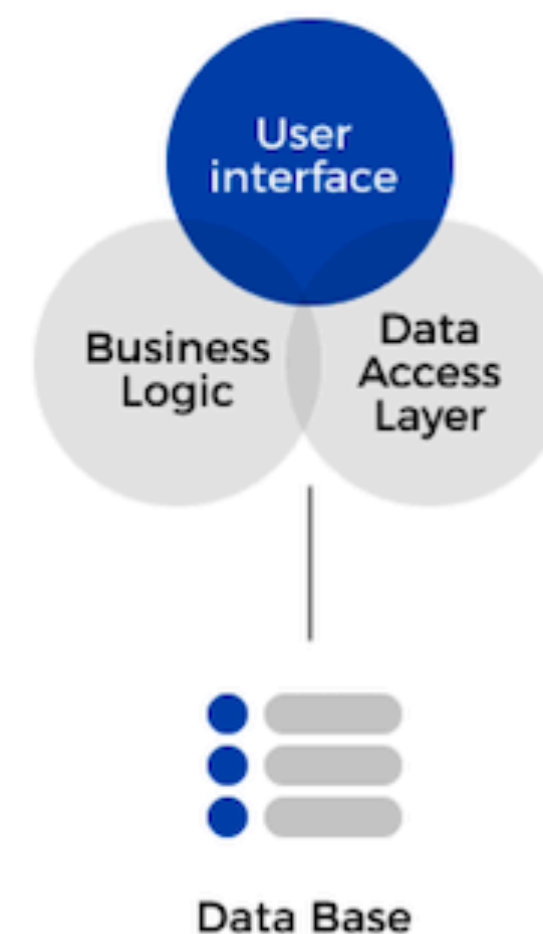
Container - Orchestration

- Container Orchestration 工具
 - Singel Computer
 - [docker compose](#) - Docker, Inc 開發，安裝 Docker Desktop 時自動安裝
 - Computer Cluster
 - [Kubernetes](#) - Google 設計與開源，捐贈給 [CNCF](#)，為目前主流容器管理平台
 - [Apache Mesos](#) - Cluster 管理工具，支援 Container 管理，一度被移入 EOF 專案清單中，雖然經社群投票後救回，但已非主流工具
 - [Docker Swarm](#) - Docker, Inc 開發，管理與調度 Container，已停止更新，改為 Docker 的附屬功能 Swarm mode

Container - Microservice

- Microservice (微服務)
 - 一種軟體架構風格
 - 專注於單一責任與功能的小型功能區塊(Small Building Blocks) 為基礎
 - 利用模組化的方式組合出複雜的大型應用程式
 - 功能區塊使用與語言無關的 API 相互通訊
 - 可用不同技術 Implement
 - Container (K8s)
 - [Spring Cloud](#)

**MONOLITHIC
ARCHITECTURE**



**MICROSERVICE
ARCHITECTURE**

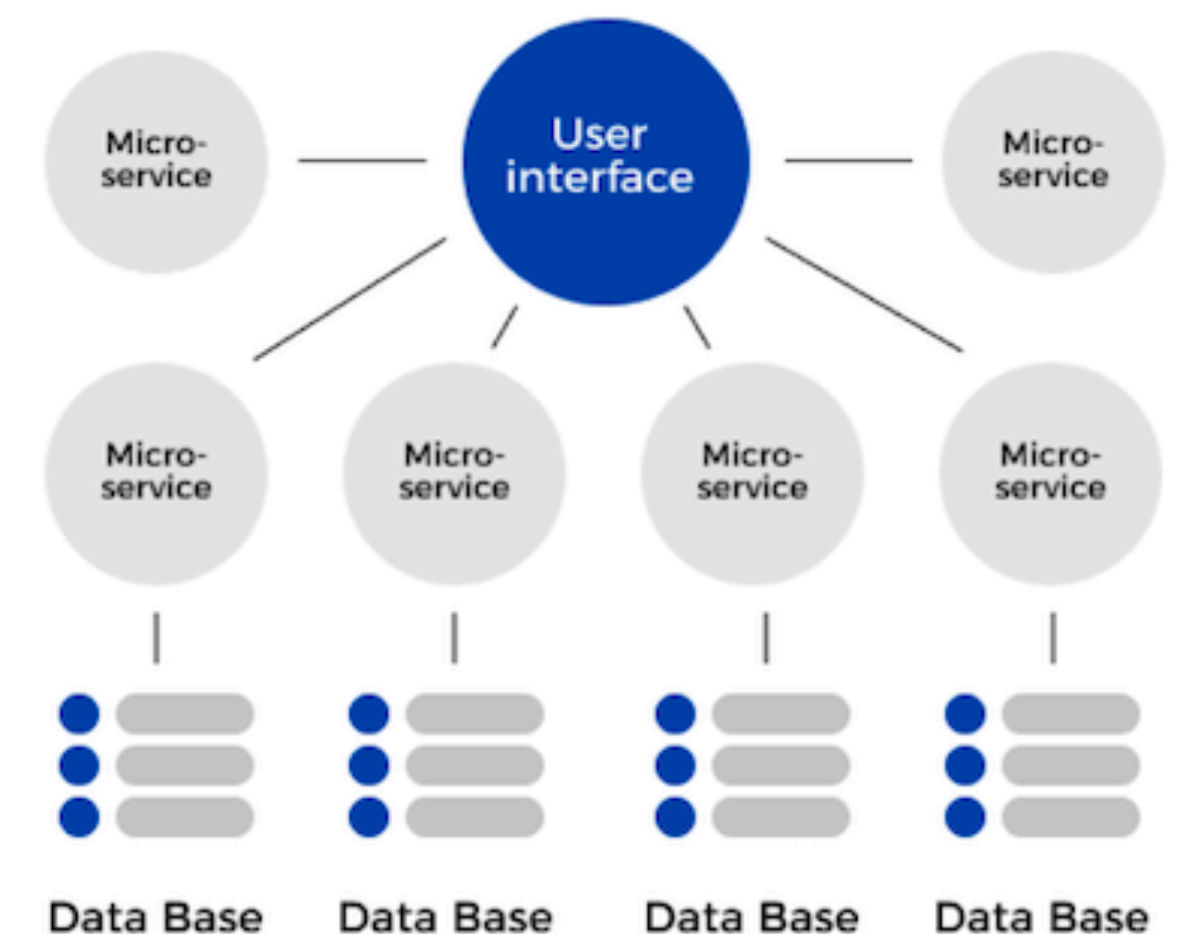


Image Source: [divante](#)

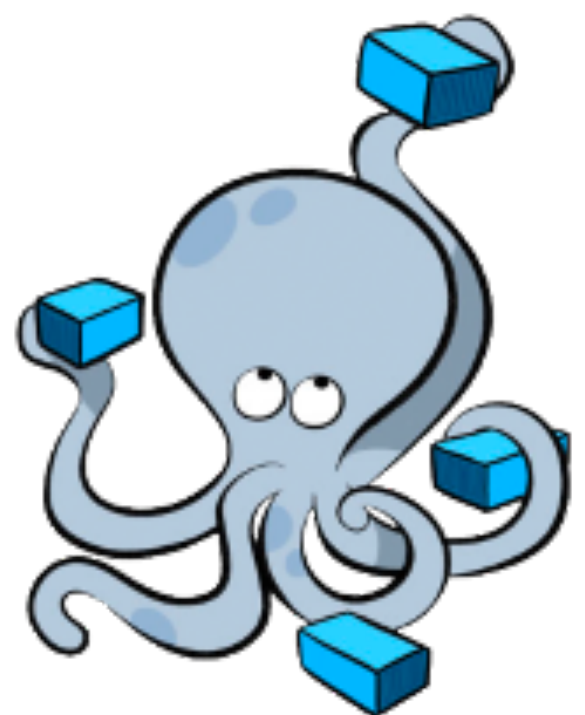
Container - docker-compose

- [docker-compose](#)
 - 可以透過 compose file (YAML) 定義一個或多個 Container，並透過 docker-compose up 啟動

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

```
docker-compose.yml

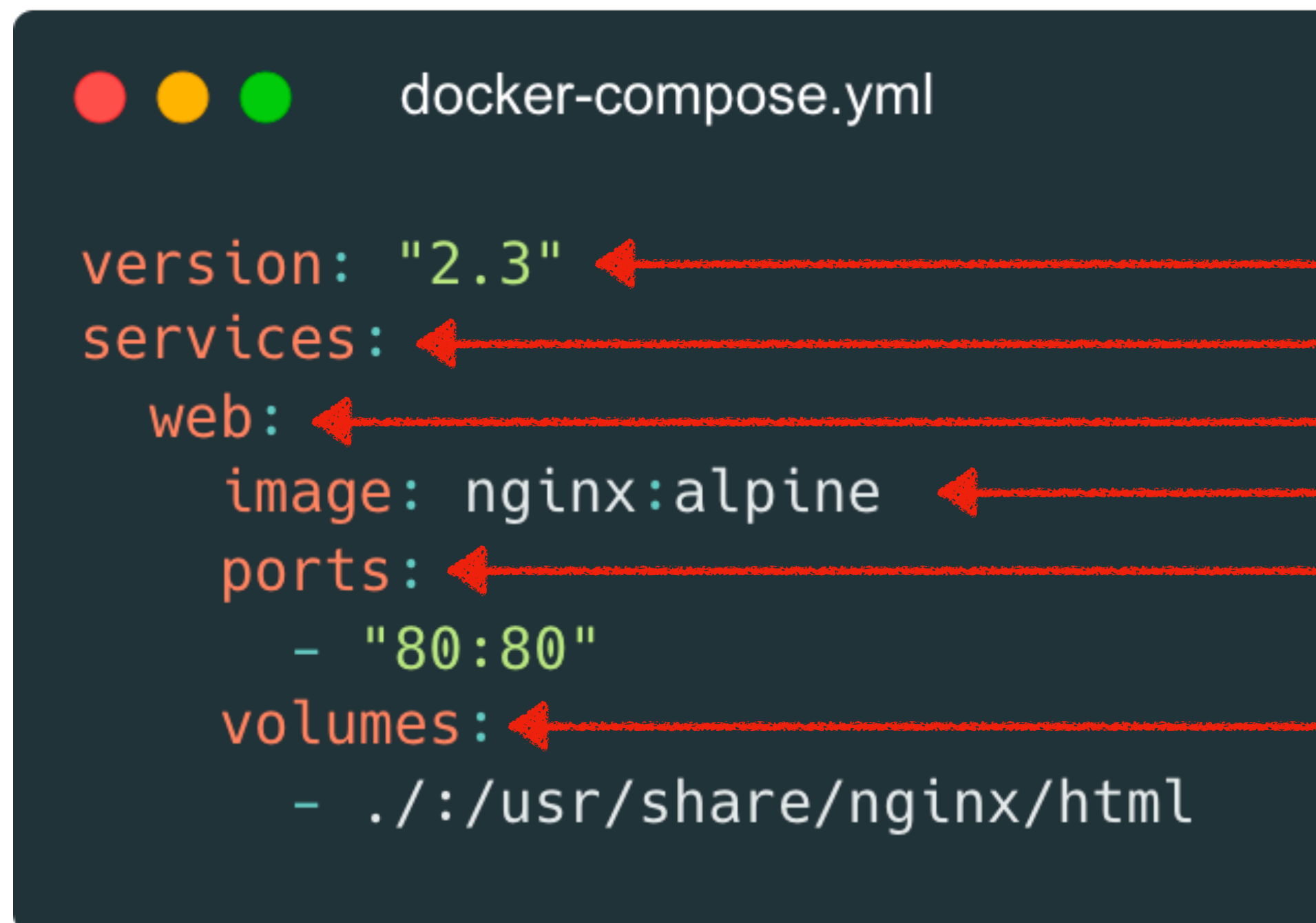
version: "2.3"
services:
  web:
    image: nginx:alpine
    ports:
      - "80:80"
    volumes:
      - ./:/usr/share/nginx/html
```



Container - docker-compose

- Compose file 有版本之分 ([Docker Compose Docs](#))，支援的 Docker Engine 版本不同，部分參數設定方式也有所不同，使用時須多加注意

```
docker run -p "80:80" -v $PWD:/usr/share/nginx/html nginx:alpine
```



The screenshot shows a terminal window with a dark background. At the top, there are three colored circles (red, yellow, green) followed by the filename 'docker-compose.yml'. The file content is as follows:

```
version: "2.3"
services:
  web:
    image: nginx:alpine
    ports:
      - "80:80"
    volumes:
      - ./:/usr/share/nginx/html
```

Red arrows point from the Chinese annotations on the right to the corresponding lines in the file:

- version: "2.3" → Compose File 版本
- services: → Container List
- web: → 在此 Compose 群組中的名稱
- image: nginx:alpine → Container 使用的 Image
- ports: → Port 對映
- volumes: → Volume 對映

Compose File 版本

Container List

在此 Compose 群組中的名稱

Container 使用的 Image

Port 對映

Volume 對映

Container - docker-compose

- 仍是使用 docker 的指令管理 Container，docker 可以增加的設定都可以在 compose file 中
- 使用場景
 - docker run 的各種參數長到難以閱讀時
 - Container 需要重複啟用時，但又不想把 docker run 存成 shell 時
 - 服務需要多個 Container 互相串聯時，如 DB、Application、Proxy
 - 除了建完就刪的測試用 Container，一律建議都使用 docker-compose 管理

Hands-on

- docker compose up (-d)
- Github Repo [docker-tutorial](#)
 - docker-102/0-db
 - docker-102/1-nginx
 - docker-102/2-web-app
 - docker-102/4-nexus

Recap

- Docker-compose 可以使用 YAML 格式的 Compose File 清楚定義一個或多個 Container
- docker-compose
 - up - 啟動 Compose 中所有 Container
 - down - 停止並移除 Compose 中所有 Container
 - restart - 重啟 Compose 中所有 Container，但不會重新讀取 docker-compose.yaml
 - logs - 檢視 Compose 中所有 Container 的 Log

Tips

Tips

- 讓 Container 持續執行，方便 Debug 或當作虛擬環境使用
 - `docker run -d [image] tail -f /dev/null`
 - `tail` - 列出檔案的最後 10 (default) 行
 - `-f` - 跟隨模式
 - `/dev/null` - 為 Linux 中一個特殊的檔案，檔案內容永遠只有 EOF
 - 合併後的效果就是跟隨一個永遠為空的檔案，成為一個永久執行但消耗極少資源的程序，確保 Container 永久處於執行狀態
- ENTRYPOINT
 - 除 CMD 外 Image 中可以額外定義 ENTRYPOINT，Container 啟動時會先執行 ENTRYPOINT 中的指令，之後再執行 CMD 的指令

Hands-on

- Github Repo [docker-tutorial](#)
 - docker-102/5-entrypoint

Scenarios

- Container 具有以下特性
 - 獨立的環境
 - Image 可以包含各種預先安裝好的 Package, 程式碼
 - 在網路設定好的狀況下，Container 之間與 Host 可以互相溝通
- 常用情境
 - 程式碼編譯
 - 乾淨的開發、執行環境
 - Proxy Server
 - 各種開源網路服務，如：Nexus, SonarQube, Jenkins...

Compile / Build with Container

- 負責編譯程式碼、打包 Image 時是否有以下痛點
 - 專案的語言種類與版本排列組合呈現爆炸性增長
 - 打包環境髒亂，Package Manager 共用相同 Cache 目錄、版本切換複雜
 - 重新下載相依套件進行打包很花時間
- 打包環境很難準備，直接用 Container 吧！

Hands-on

- Github Repo [docker-tutorial](#)
 - docker-102/3-multi-stage-build

More Scenarios

- [SonarQube](#) Server
 - [Docker Compose File](#)
- SonarQube 掃描 - [sonarsource/sonar-scanner-cli](#)
 - `docker run --rm -e SONAR_HOST_URL=[SONAR HOST] -v $PWD:/app sonarsource/sonar-scanner-cli -Dsonar.projectBaseDir=/app`
- [Jenkins](#)
 - [Install with Docker](#)
- Python [Jupyter Notebook](#) 開發環境
 - [Jupyter Docker Hub](#)
 - `docker run -d -p 8888:8888 jupyter/datascience-notebook:python-3.9.7`

More Scenarios


- Github Repo [docker-tutorial](#)
 - Proxy Server
 - docker-103/0-traefik
 - 3-Tier Application
 - docker-103/1-three-tier-application
 - Selenium end-to-end testing
 - docker-103/2-selenium-end-to-end-testing


More About Container

Cloud Native

- Cloud Native 雲原生
 - 最初是針對服務定義
 - 服務容器化
 - 面向微服務架構
 - 服務支援以容器的方式被調度
 - 現在多指的是一種文化
 - 利用雲讓服務的開發與維運更加快速與敏捷的行為或方法
- [Cloud Native Computing Foundation \(CNCF\)](#)
 - CNCF 致力於推廣 Cloud Native
 - 組織成員包括 Google, Amazon, Microsoft, Alibaba, Intel, IBM 等
- [Cloud Native Landscape](#)
 - [路線圖](#)
 - [工具清單](#)

What is Cloud Native?

Article • 11/11/2021 • 21 minutes to read •  +9

Is this page helpful? 

Stop what you're doing and text several of your colleagues. Ask them to define the term "Cloud Native". There's a good chance you'll get several different answers.

From: Microsoft Docs



CLOUD NATIVE
COMPUTING FOUNDATION

Best Practice

- [RedHat Cloud Native Container Design White Paper](#)
- [The best Docker base image for your Python application](#)
- [Docker Docs - Image Building Best Practices](#)

Future of Container

- Docker != Container
- [Open Container Initiative \(OCI\)](#)
 - 定義 Runtime 以及 Image 兩個標準規範 Container
- [Container Runtime Interface \(CRI\)](#)
 - 由 K8s 提出的介面標準，只要 Container Runtime 有實作 CRI 介面就可以被 K8s 管理
- Docker Alternatives
 - [Buildah](#)、[Podman](#)、[Kaniko](#)
- K8s v1.20 之後開始棄用 Docker !?
 - [Don't Panic: Kubernetes and Docker](#)
 - [\[FB\] 對開發人員與維運人員的影響](#)

Appendix - YAML

- [YAML](#) (YAML Ain't a Markup Language)
 - 表達數據的一種資料格式，在 Cloud Native 的領域中被廣泛運用
 - 副檔名為 .yaml 或 .yml
 - 使用縮排定義結構化資料，可以增加註解

```
data.yaml

stores:
- name: seven-eleven
  city: Taipei # 所在城市
  products: # 販賣產品
    - coke
    - chips
    - coffee
```

==

```
data.json

{
  "stores": [
    {
      "name": "seven-eleven",
      "city": "Taipei",
      "products": [
        "coke",
        "chips",
        "coffee"
      ]
    }
  ]
}
```

Appendix - Unix and Linux

- [Unix Vs Linux: What Is Difference Between UNIX And Linux](#)
- [Difference between Linux distributions](#)

Appendix - Container OS/ARCH

雖然 Container 標榜可以完全重現環境，只要有 Container Runtime 就可以執行 Image，但根本性的 CPU 架構問題 Container 是無法解決的。如果 Container 或其中的程式在 Compile 時與之後執行的 Host CPU 架構不同，可能會造成無法執行出現 Core Dumped 等錯誤。

在 Docker Hub 的 Image Tag 頁籤可以看到各 Tag 有標註 OS/ARCH，OS 指的 Container 的 OS，ARCH 則是 CPU 架構。例如 Nginx 提供了多種不同 ARCH 的 Image，而 Nexus 只提供 amd64 架構的 Image。

- M1 常見問題
 - [\[Docker\] Mac M1 – no matching manifest for linux/arm64/v8 in the manifest list entries](#)
 - [Mac M1系列-解決docker安裝mysql error:no matching manifest for linux/arm64/v8 in the manifest list entries](#)
 - [M1 使用本地 docker push 到 cloud run 出現錯誤](#)

Appendix - ERR_UNSAFE_PORT

- 問題
 - 以 6666 port 開啟服務，使用 Chrome 瀏覽時出現 ERR_UNSAFE_PORT 錯誤訊息
- 原因
 - 基於安全理由 Chrome 或其他瀏覽器會直接阻擋特定 port 的服務。Chromium 的原始碼有列出會被阻擋的清單，在使用時應特別注意。
 - 6665~6669 是 IRC protocol 預設使用的 Port。IRC 有許多安全漏洞，為了避免 Chrome 變成跳板去影響其他服務，所以會主動進行阻擋。因此這些阻擋是保護 Service 本身，而不是保護 Chrome 的使用者。
- Reference
 - [Chrome错误代码:ERR_UNSAFE_PORT](#)
 - [Which ports are considered unsafe by Chrome?](#)
 - [Why does Chrome consider some ports unsafe?](#)

Appendix - 127.0.0.1 vs localhost vs 0.0.0.0

- localhost vs 0.0.0.0 vs 127.0.0.1
- 127.0.0.1
 - This is a "fake" network adapter that can only communicate within the same host.
 - A process that is listening on 127.0.0.1 for connections will **only receive local connections** on that socket.
- localhost
 - **Hostname** for the **127.0.0.1** IP address.
- 0.0.0.0
 - When a server is told to listen on 0.0.0.0 that means "**listen on every available network interface**".
- [Stack Overflow - What is the difference between 0.0.0.0, 127.0.0.1 and localhost?](#)

Reference

- [Container的歷史](#)
- [Docker 传奇之 dotCloud](#)
- [初探Docker - Docker 跟 LXC 以及一般Hypervisor有何差別？](#)
- [最完整的Docker聖經 - Docker原理圖解及全環境安裝](#)
- [The Magic Behind the Scenes of Docker Desktop](#)
- [Explaining Docker Image IDs](#)
- [Lifecycle of Docker Container](#)
- [Understanding Docker Container Exit Codes](#)
- [邱牛 iThome 鐵人賽 K8s 系列文](#)
- [Docker Images Without Docker — A Practical Guide](#)