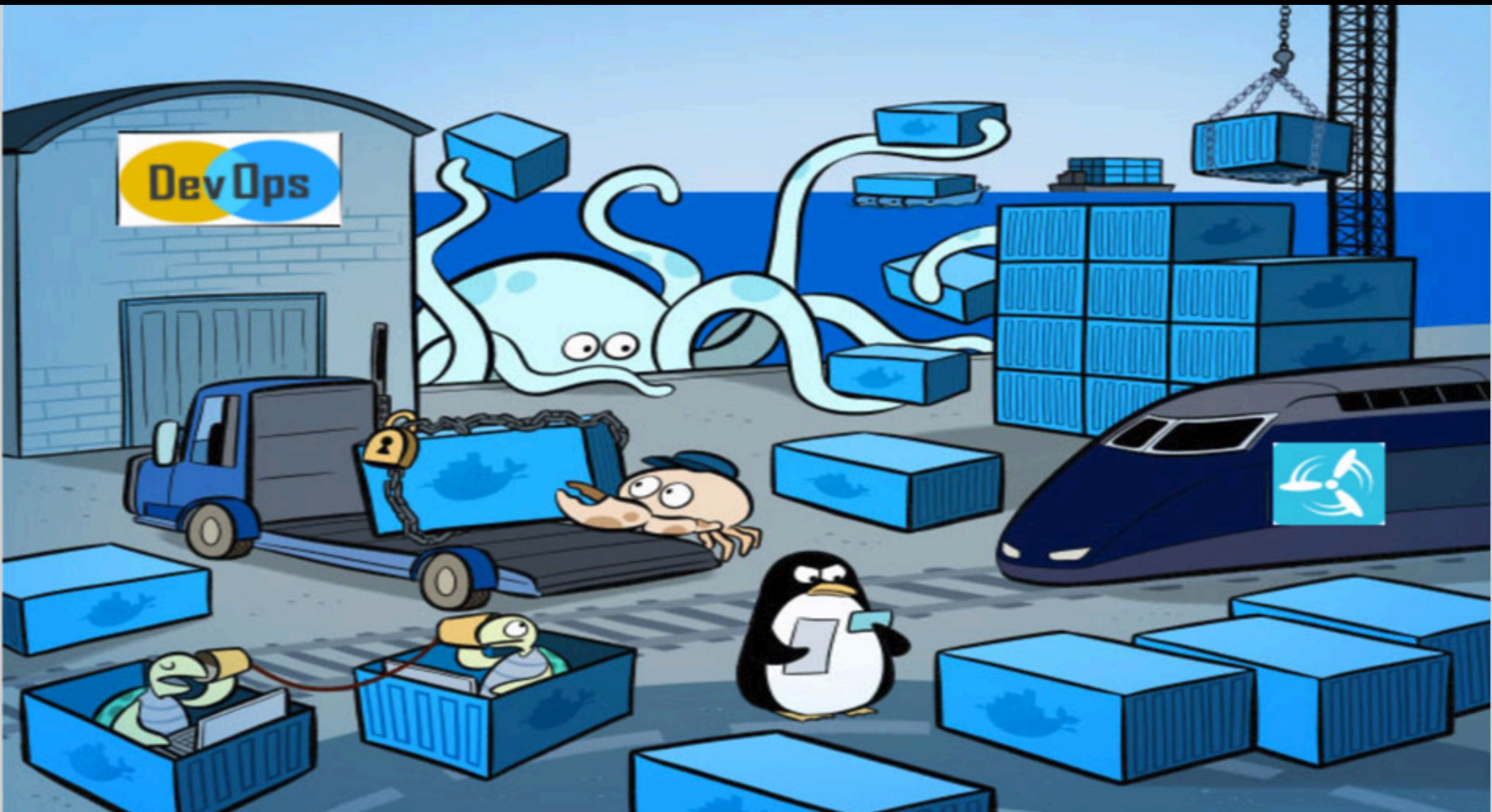


BUILD, SHIP, AND RUN ANY APP, ANYWHERE.

DOCKER TUTORIAL



OUTLINE

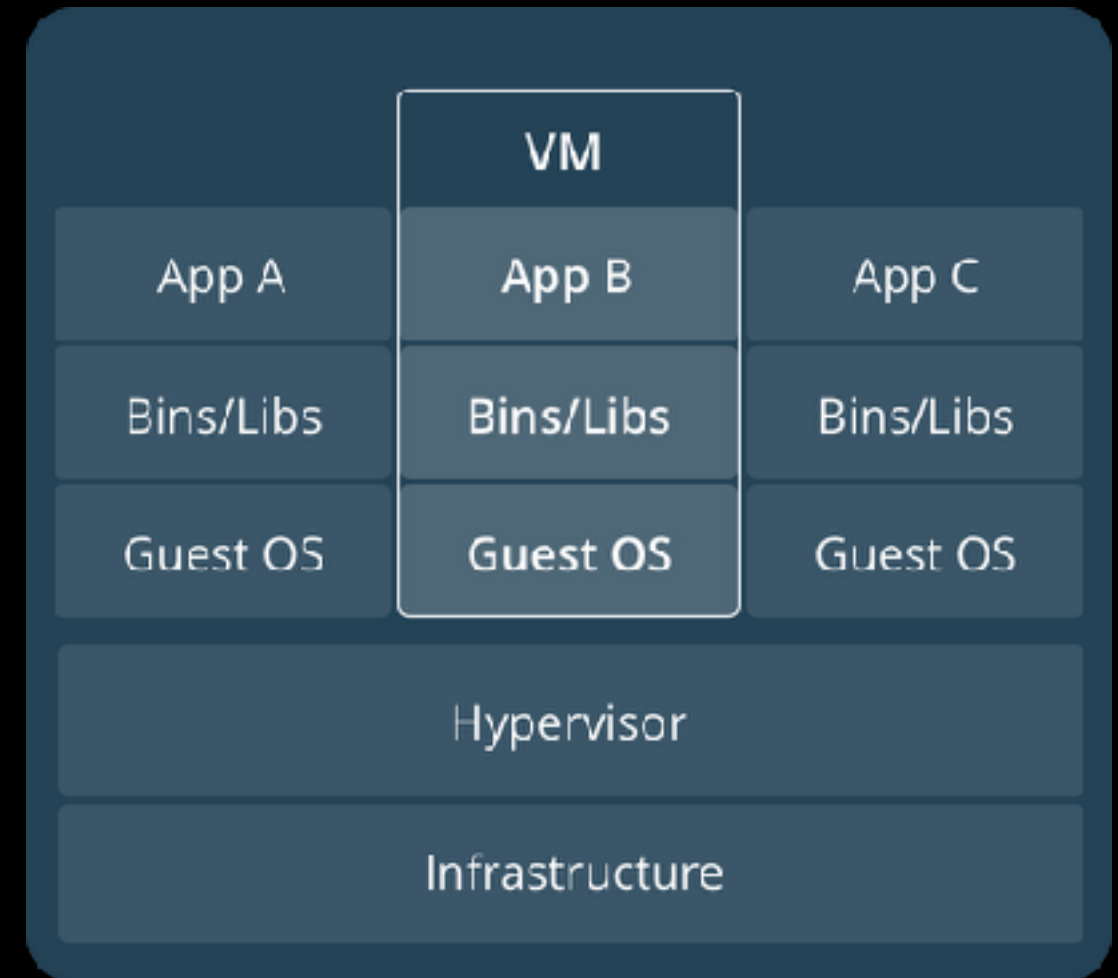
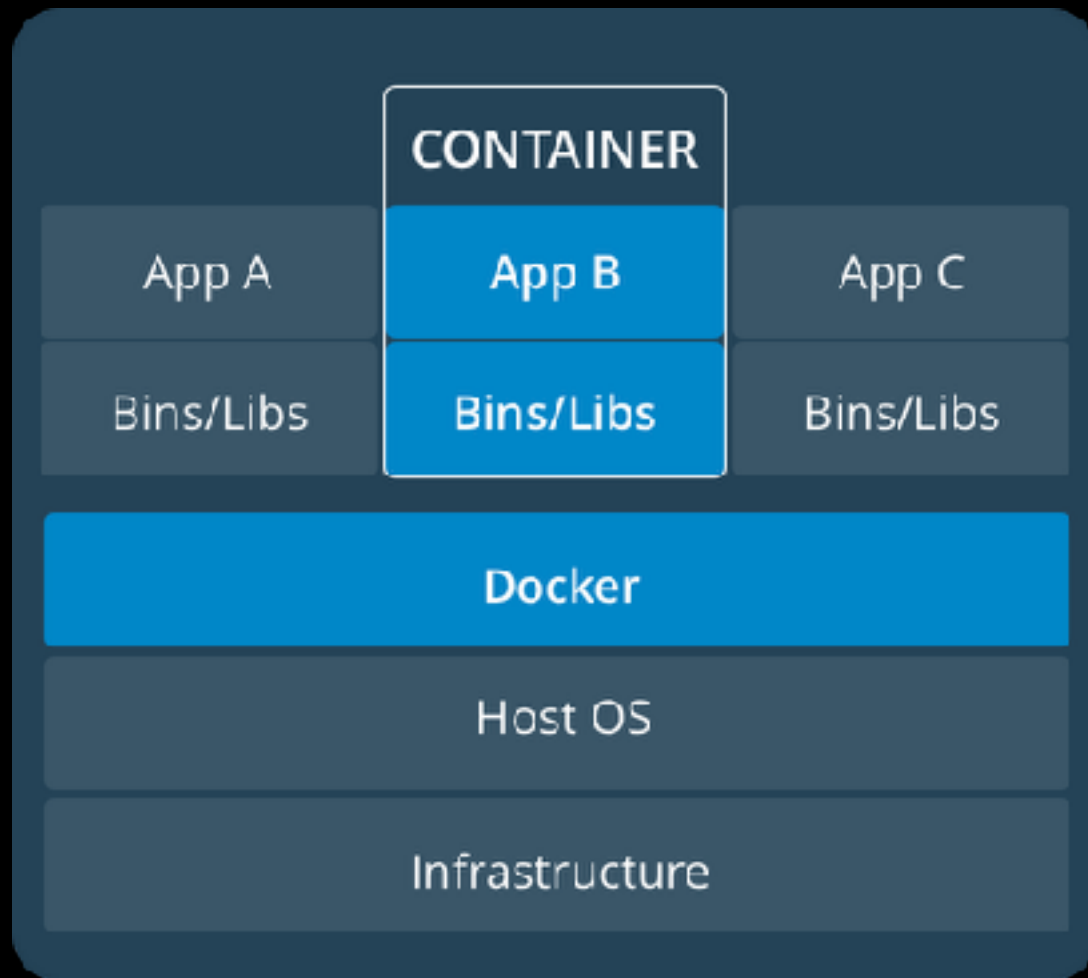
- 一、What is Docker
- 二、Docker v.s. VM
- 三、Docker Objects
- 四、Docker Architecture
- 五、Why Use Docker
- 六、常用Commands
- 七、Dockerfile設計方式
- 八、Azure Container Registry
- 九、Reference

What is Docker ?

跨平台容器管理工具

二、DOCKER VS VM

- 容器與VM的概念很像，但有點不一樣



二、DOCKER VS VM

比較	容器	虛擬機
啟動	秒級	分鐘級
硬碟容量	一般為 MB	一般為 GB
效能	接近原生	比較慢
系統支援量	單機支援上千個容器	一般幾十個

三、DOCKER OBJECTS

- Image
- Container
- Registries

三、DOCKER OBJECTS

- Image
 - Container是透過Image所建立起來的
 - Image是一種唯讀的模版
 - Image會以 os lib 為底建立起來，並可以讓使用者自行加入所要執行的程式、框架和軟體等
 - Image都可以客製化包裝成另一種Image

三、DOCKER OBJECTS

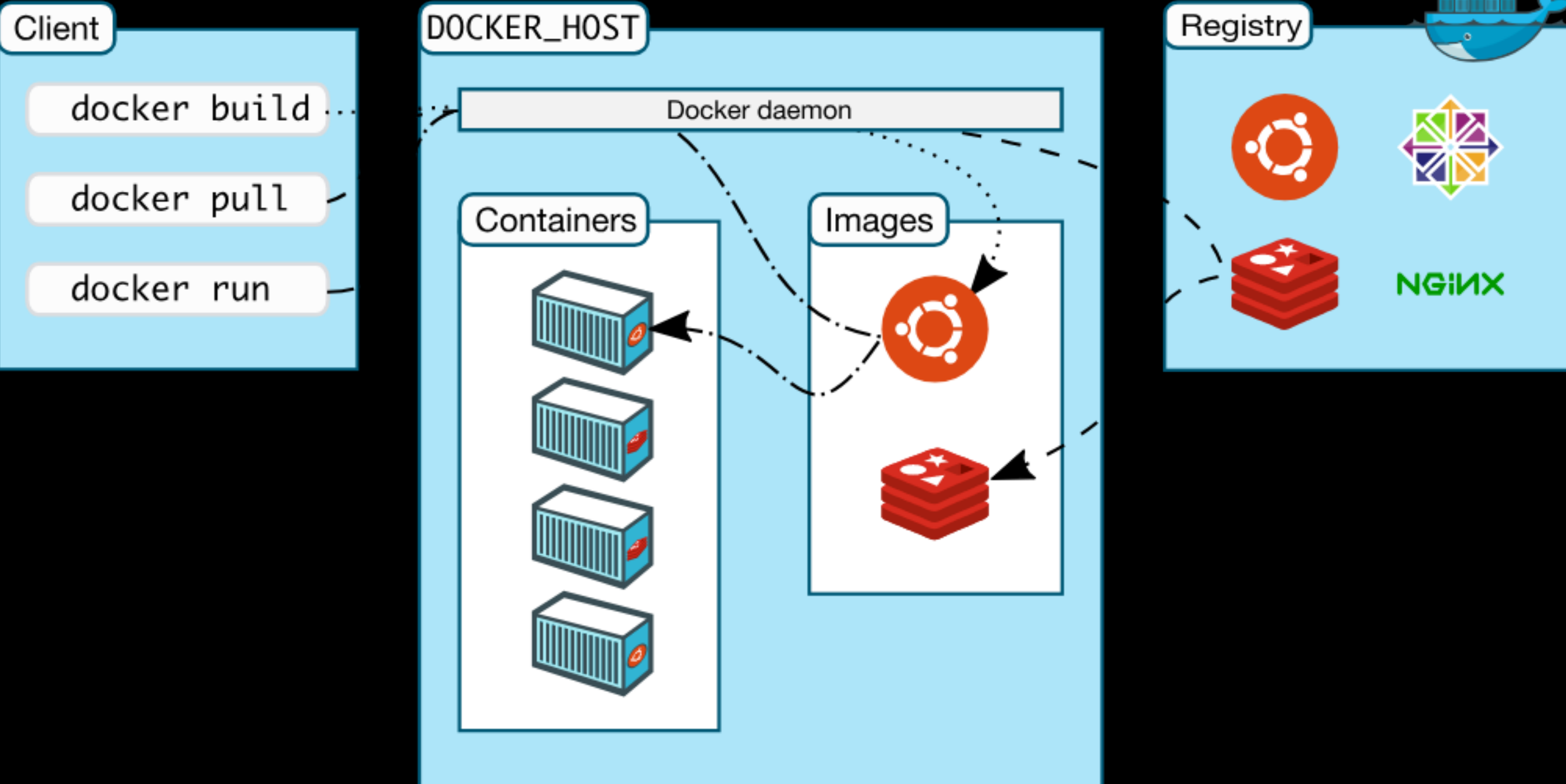
- Container
 - 透過Image所建立起來的執行個體
 - 一個Image可以建立多個Container，每個Container都是獨立的，並擁有屬於自己的設定

三、DOCKER OBJECTS

- Registries
 - 存放Image的遠端儲存庫
 - Docker Hub
 - Azure Container Registry Service

四、DOCKER ARCHITECTURE

`docker run -i -t ubuntu`



五、WHY USE DOCKER

- 簡化系統的建立、搬移和擴充
- 統一系統環境，不用擔心系統執行環境的差異性
 - 系統的開發、PM和正式環境，可能是不同的OS或不同的OS版本
- 雲端和地端都可以通用
 - Azure Function只能跑在雲端嗎？
- DevOps的好幫手
 - CI/CD

六、常用COMMANDS

- docker pull (從Docker Hub下載Image)
- docker image (Image相關指令)
- docker ps (查詢Container)
- docker run,stop & rm (執行、停止和刪除Container)
- docker exec (進入Container)
- docker volume (資料卷冊)

六、常用COMMANDS

- 下載Image
 - `docker pull [ImageName]:[tag]`
 - `docker pull ubuntu:18.10`
- 查詢Images
 - `docker images`
- 刪除Images
 - `docker rmi [ImageName]:[tag]`
 - `docker rmi ubuntu:18.10`
- 刪除本機所有Image
 - `docker image prune`

六、常用COMMANDS

- 查詢Container
 - `docker ps [option]`
 - 常用option
 - `-a`：查詢所有的Container (包含已掛掉的)

六、常用COMMANDS

- 執行Container
 - `docker run [option] [ImageName]:[tag]`
 - **如果本機沒有此Image，docker daemon會先執行docker pull，下載完image後，再執行docker run

六、常用COMMANDS

- 執行Container
 - 常用的docker run Option
 - --name : 指定Container名稱
 - --publish , -p : 開啟Container對外的Port (HostPort:ContainerPort)
 - --restart={always,no} : 是否重新啟動Container，預設值為no
 - always : 當Container停掉後，會自動重新啟動Container
 - no : 當Container停掉後，不進行任何動作
 - --rm : 當Container停掉時，自動刪除
 - -d, --detach : 讓Container在背景執行
 - -it, --interactive : 讓Container 可以在CommandLine界面與User互動
 - -v, --volume : 讓Container掛載Volume

六、常用COMMANDS

- 執行Container
 - `docker run --rm -it --name MyUbuntuInterface ubuntu:18.10`
 - `docker run --name MyTempUbuntuBack -d ubuntu:18.10`
 - `docker run --name mynginx -p 88:80 -d --restart=always nginx:alpine`

六、常用COMMANDS

- 停止Container
 - docker stop {ContainerID, ContainerID前三碼, Container名稱}
 - `docker stop mynginx`
- 刪除Container
 - docker rm {ContainerID, ContainerID前三碼, Container名稱}
 - `docker rm mynginx`

六、常用COMMANDS

- 進入Container
 - docker exec [option] {ContainerID, ContainerID前三碼, Container名稱} [Command]

六、常用COMMANDS

- 進入Container
 - 常用的docker exec Option
 - -d, --detach
 - -it
 - -i
 - `docker exec -it mynginx /bin/sh`

六、常用COMMANDS

- `docker volume`
 - 當Container刪除後，存在Container的資料還會存在嗎
 - Volume Driver 類型
 - local (本機檔案系統)
 - nfs (網路檔案系統)
 - btrfs (B-tree檔案系統)

六、常用COMMANDS

- `docker volume`
 - `create` : 建立volume
 - `ls` : 輸出目前本機的volume列表
 - `inspect` : 查詢volume詳細資訊
 - `rm` : 刪除指定volume
 - `prune` : 刪除全部volume

六、常用COMMANDS

- create：建立volume
 - docker volume create [option] [volumeName]
 - Option參數
 - -d, --driver：volume driver類型，預設值為local
 - -o, --opt：volume driver 的設定
 - docker volume create -d local myVolume
 - docker volume ls

六、常用COMMANDS

- inspect：查詢volume明細資料
 - docker volume inspect [option]
[volumeName]
 - docker volume inspect myVolume

六、常用COMMANDS

- rm : 刪除Container
 - docker volume rm [volumeName]
 - docker volume rm myVolume
- 刪除本機全部的Volume
 - docker volume prune

六、常用COMMANDS

- Container掛載volume
 - `docker run [option] --name [ContainerName] -v [volumeName:ContainerFolderName]`
 - `docker run -d --name=volumetest -v myvolume:/appVolume nginx:alpine`

七、DOCKERFILE設計方式

- 1. Docker Build Commands
- 2. Dockerfile 設計格式
- 3. Dockerfile 常用參數
- 4. Dockerfile MultiStep 設計格式
- 5. 如何使用Dockerfile ARG

七、DOCKERFILE設計方式

- 1. Docker Build Commands

- `docker build [option] PATH | URL | .`

- Option參數

- `-f` : dockerfile檔案路徑 (如果當前目錄已有Dockerfile檔案，則可以不用輸入此參數)
- `--no-cache` : 建立image時，不使用之前的cache
- `-t` : 指定image名稱 ([ImageName]:[tag])
- `--build-arg [arg option]` : 動態傳入參數至Image
- `--rm` : 當Image成功建立後，將暫時性的Container刪除 (預設為true)

七、DOCKERFILE設計方式

- 1. Docker Build Commands
 - Hands-On : 0-DockerBuildCommand

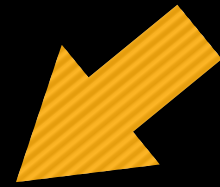
```
docker build --no-cache --rm -t dockerng:latest .
```

```
docker run --rm -p 855:80 dockerng:latest
```

七、DOCKERFILE設計方式

- 2. Dockerfile設計格式

載入Image檔



```
FROM nginx:alpine as BaseImage
```

```
RUN rm -rf /usr/share/nginx/html/*
```

```
COPY default.conf /etc/nginx/conf.d/default.conf
```

```
COPY /dist/DockerNG /usr/share/nginx/html
```

```
EXPOSE 80
```



建立Image的暫時性Container，並執行Dockerfile參數，執行完成後，再將此Container封裝成新Image後，再自動砍掉此Container

七、DOCKERFILE設計方式

- 3. Dockerfile常用參數

- RUN [command]：執行Image os lib的指令 (或已安裝好的套件的指令)
- ENTRYPOINT [command]：在背景執行Image os lib 的指令 (或已安裝好的套件的指令)
- EXPOSE {PORT}：開啟Container對外的port
- COPY [host dir] [Image dir]：複製本機的檔案或資料夾到Image的資料夾
- ENV [parameter]：Image的環境變數
- ARG [parameter]：接收 --build-arg 所傳入的參數
- WORKDIR [path]：執行docker參數的資料夾位置

七、DOCKERFILE設計方式

- Hands on : 1-BuildDotnetCoreSDKImage (建立擁有 DotnetCore sdk 的Image)

#載入ubuntu:18.04 image

FROM ubuntu:18.04 as baseImage

#執行安裝dotnet-core-sdk指令

RUN apt-get update

RUN apt-get install wget -y && apt-get install gpg -y

RUN wget -qO- https://packages.microsoft.com/keys/microsoft.asc | gpg --dearmor > microsoft.asc.gpg

RUN mv microsoft.asc.gpg /etc/apt/trusted.gpg.d/

RUN wget -q https://packages.microsoft.com/config/ubuntu/18.04/prod.list

RUN mv prod.list /etc/apt/sources.list.d/microsoft-prod.list

RUN chown root:root /etc/apt/trusted.gpg.d/microsoft.asc.gpg

RUN chown root:root /etc/apt/sources.list.d/microsoft-prod.list

RUN apt-get install apt-transport-https

RUN apt-get update

RUN apt-get install dotnet-sdk-2.1 -y

七、DOCKERFILE設計方式

- Hands on : 1-BuildDotnetCoreSDKImage (建立擁有DotnetCore sdk 的Image)
- `docker build --rm --no-cache -t dotnet-core-base-image:latest .`

七、DOCKERFILE設計方式

- Hands on : 2-BuildDotnetCoreMVCwebsiteImage (建立執行DotnetCore MVC網站的Image)

#載入Hands-on 1所作出來的image

FROM dotnet-core-base-image:latest as baseImage

複製dotnet core mvc web site Publish Files到Image

COPY ./publish/..

開啟Container對外的5000 port

EXPOSE 5000

執行dotnet core mvc website

ENTRYPOINT ["dotnet", "DockerMVC.dll", "--server.urls",
"http://*:5000"]

七、DOCKERFILE設計方式

- Hands on : 2-BuildDotnetCoreMVCwebsiteImage (建立執行DotnetCore MVC網站的Image)

```
docker build -t dotnet-core-mvc-website-publish:latest --rm --no-cache .
```

```
docker run --rm -p 5001:5000 dotnet-core-mvc-website-publish:latest
```

七、DOCKERFILE設計方式

- 4. Dockerfile MultiStep 設計格式
 - 在一個Dockerfile中載入多個Image
 - 只會將Step中的最後一個Image的Container重新包裝成新的Image
 - 優點：可以統一整個專案編譯、還原和執行的環境
 - Hands-on : 3-MultiStepBuildImage

#Step 1 建立可以還原和發行Dotnet Core mvc網站專案的Image

FROM dotnet-core-base-image:latest as publishImage

建立app資料夾

RUN mkdir /app

設定app資料夾為WORKDIR

WORKDIR /app

複製整個dotnet core mvc專案至publishImage

COPY ./DockerMVC/..

還原專案套件

RUN dotnet restore

發行專案，並將發行網站用的檔案移至publish資料夾

RUN dotnet publish -c Release -o ./publish

- Hands -on : 3-MultiStepBuildImage

#Step 2 建立可以執行dotnet core mvc 網站的Image

FROM dotnet-core-base-image:latest as execlImage

複製publishImage的 publish 檔案到execlImage

COPY --from=publishImage /app/publish/..

開啟Container對外的5000 port

EXPOSE 5000

執行dotnet core mvc website

ENTRYPOINT ["dotnet", "DockerMVC.dll", "--server.urls", "http://*:5000"]

七、DOCKERFILE設計方式

- 4. Dockerfile MultiStep 設計格式
 - Hands-on : 3-MultiStepBuildImage

```
docker build --rm --no-cache -t dotnet-core-multi-step-publish:latest .
```

```
docker run --rm -p 5011:5000 dotnet-core-multi-step-publish:latest
```

七、DOCKERFILE設計方式

- 4. Dockerfile MultiStep 設計格式
 - Hands-on : 4-BuildImageFromCustomImage

使用官方所提供的DotnetCore SDK Image

```
FROM microsoft/dotnet:2.1.302-sdk-alpine as publishImage
```

```
RUN mkdir /app
```

```
WORKDIR /app
```

```
COPY ./DockerMVC/. .
```

```
RUN dotnet restore
```

```
RUN dotnet publish -c Release -o ./publish
```

```
FROM microsoft/dotnet:2.1.2-aspnetcore-runtime as execlmage
```

```
COPY --from=publishImage /app/publish/. .
```

```
ENTRYPOINT ["dotnet", "DockerMVC.dll"]
```


七、DOCKERFILE設計方式

- 4. Dockerfile MultiStep 設計格式
 - Hands-on : 4-BuildImageFromCustomImage

使用官方所提供的DotnetCore SDK Image

```
docker build --no-cache --rm -t dotnet-core-official-multi-step-publish:latest .
```

```
docker run -p 5500:80 --rm dotnet-core-official-multi-step-publish:latest
```

七、DOCKERFILE設計方式

- 5. 如何使用Dockerfile ARG
 - 適合用在建立不同版本的Image
 - 減少Dockerfile的數量

七、DOCKERFILE設計方式

- 5. 如何使用Dockerfile ARG
- `docker build --build-arg [ArgName]=[ArgValue] [option] [ImageName:tag] .`

```
FROM [image:tag]
```

```
Arg ArgName
```

```
ENV ASPNETCORE_ENVIRONMENT=${ArgName}
```

七、DOCKERFILE設計方式

- 5. 如何使用Dockerfile ARG (Hands-on : 5-BuildImageUsingArgs)
- `docker build --build-arg ReleaseType=[ArgValue] [option] [ImageName:tag] .`

```
FROM microsoft/dotnet:2.1.302-sdk-alpine as publishImage
RUN mkdir /app
WORKDIR /app
COPY ./DockerMVC/. .
RUN dotnet restore
RUN dotnet publish -c Release -o ./publish
```

```
FROM microsoft/dotnet:2.1.2-aspnetcore-runtime as execlmage
Arg ReleaseType
ENV ASPNETCORE_ENVIRONMENT=${ReleaseType}
COPY --from=publishImage /app/publish/. .
ENTRYPOINT ["dotnet", "DockerMVC.dll"]
```

七、DOCKERFILE設計方式

- 5. 如何使用Dockerfile ARG (Hands-on : 5-BuildImageUsingArgs)
- 建立開發環境用的Image
 - `docker build --no-cache --build-arg ReleaseType=Development -t dotnet-core-mvc-website-using-arg:dev .`
 - `docker run -p 5002:80 --rm dotnet-core-mvc-website-using-arg:dev`
- 建立PM環境用的Image
 - `docker build --no-cache --build-arg ReleaseType=pm -t dotnet-core-mvc-website-using-arg:pm .`
 - `docker run -p 5003:80 --rm dotnet-core-mvc-website-using-arg:pm`
- 建立正式環境用的Image
 - `docker build --no-cache --build-arg ReleaseType=Production -t dotnet-core-mvc-website-using-arg:prod .`
 - `docker run -p 5004:80 --rm dotnet-core-mvc-website-using-arg:prod`

8、AZURE CONTAINER REGISTRY

- 3-1 Azure cli
- 3-2 Azure Container Registry (ACR)簡介

8、AZURE CONTAINER REGISTRY

- 8-1 Azure cli
- 8-2 Azure Container Registry (ACR)簡介

8、AZURE CONTAINER REGISTRY

- 8-1 Azure cli

- 從終端機登入Azure

- az login

- 查詢自己的azure帳號有哪些訂閱

- az account list -o table

- 切換訂閱

- az account set -s [SubscriptionId]

- 查詢自己的azure帳號目前是使用哪個訂閱

- az account show -o table

8、AZURE CONTAINER REGISTRY

- 8-2 ACR

- Azure 私人Image儲存庫

- az acr login --name [RegistryName]
- az acr list -o table

8、AZURE CONTAINER REGISTRY

- 8-2 ACR

- Azure 私人Image儲存庫
- docker tag dotnet-core-mvc-website-using-arg:dev [ACR-LoginServer]/docker-mvc:[YourDevTag]
- docker tag dotnet-core-mvc-website-using-arg:pm [ACR-LoginServer]/docker-mvc:[YourPMTag]
- docker tag dotnet-core-mvc-website-using-arg:prod [ACR-LoginServer]/docker-mvc:[YourProdTag]

8、AZURE CONTAINER REGISTRY

- 8-2 ACR

- Azure 私人Image儲存庫
- docker push [ACR-LoginServer]/docker-mvc:[YourDevTag]
- docker push [ACR-LoginServer]/docker-mvc:[YourPMTag]
- docker push [ACR-LoginServer]/docker-mvc:[YourProdTag]

REFERENCE

- Docker Official Website
- Docker For Beginners
- Docker 從入門到實踐
- <https://www.pearltrees.com/benkung/docker/id17033125>