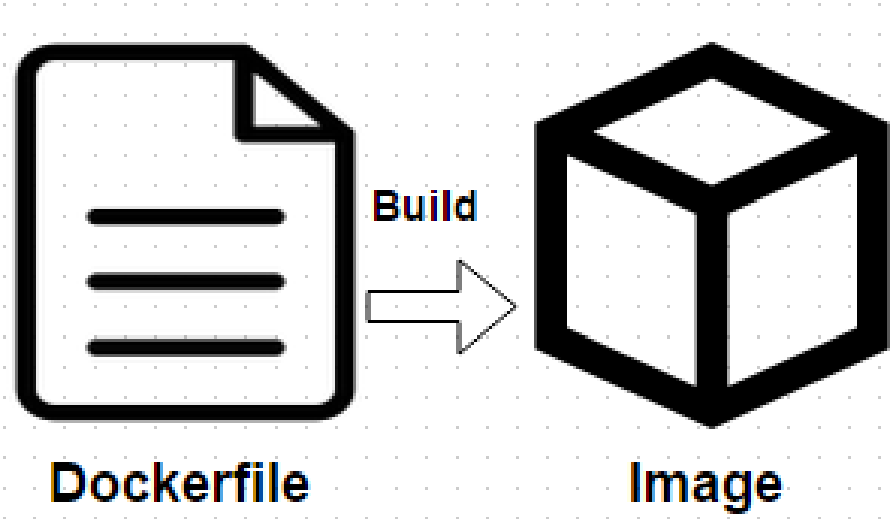
認識 Dockerfile

Dockerfile 就是建置 Docker Image 的腳本



撰寫 Dockerfile

在 **Windows** 系統的 **Power shell** 視窗，執行以下命令

```
$ ssh bigred@<CTN.ALP.Docker IP>

$ mkdir -p ~/wulin/myring; cd ~/wulin/myring
```

開始撰寫 **Dockerfile**

```
$ echo 'FROM alpine
RUN echo "top.secret" > /password.txt
RUN rm /password.txt' > Dockerfile
```

FROM : Docker 命令
alpine : 從官網下載的光碟片

執行linux命令

RUN在container執行

建立與測試 Docker image

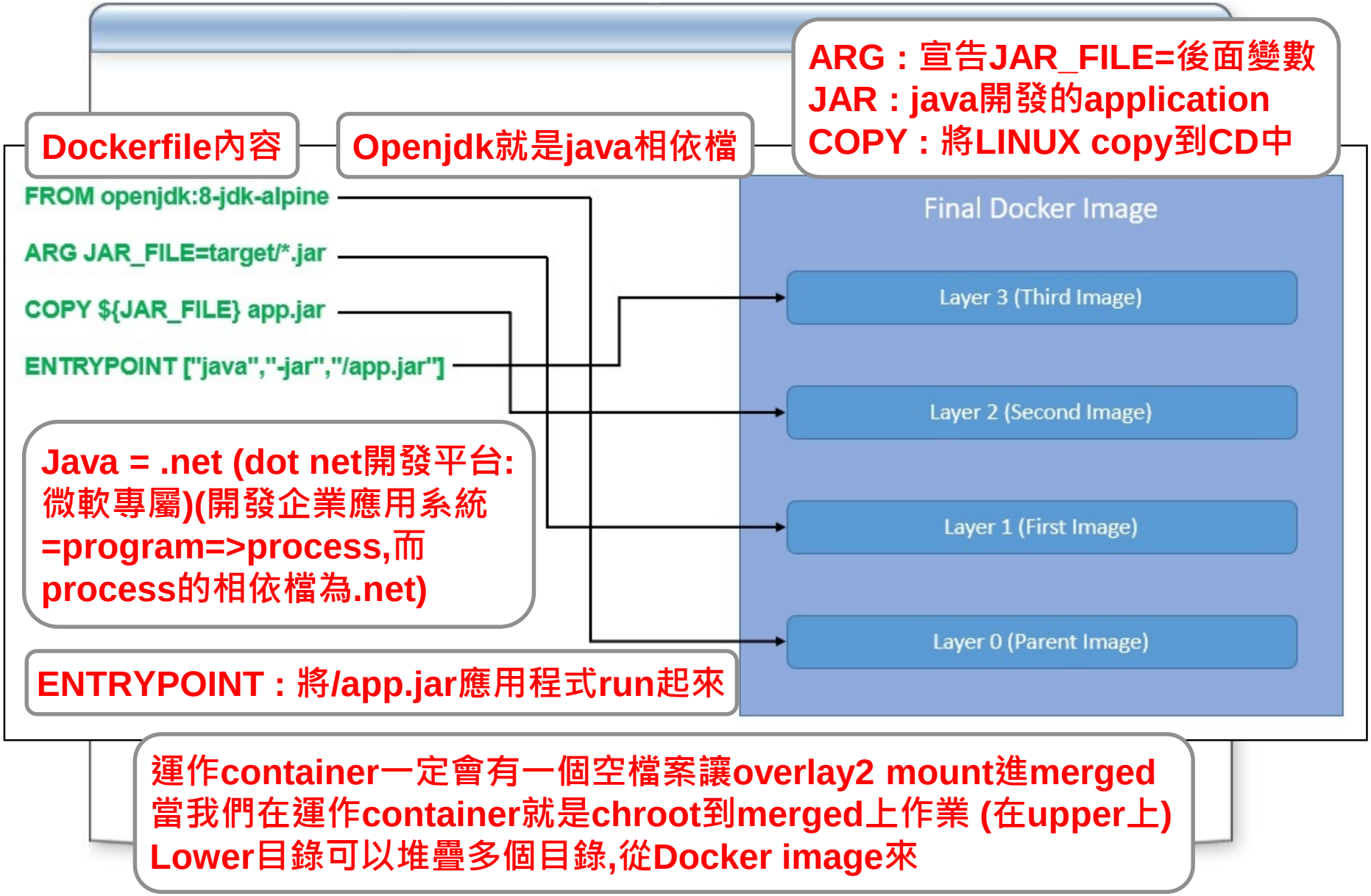
```
$ docker build --no-cache -t myring  .

$ docker images
REPOSITORY   TAG       IMAGE ID         CREATED              SIZE
myring       latest    97f72354b5a6     9 seconds ago        5.61MB
alpine       latest    6dbb9cc54074     3 weeks ago          5.61MB

$ docker run --rm myring  ls /password.txt
ls: /password.txt: No such file or directory
```

no cache :下載完刪除
‘.’ :在當前目錄找Dockerfile

Docker image 目錄結構



研究 Docker image

```
$ docker save myring > myring.tar

$ mkdir imglayers; tar -xf myring.tar -C imglayers/

$ tree imglayers/
```

Docker image 內容

```
imglayers/
├── 579fb4fe2cc21a56bb5518ce0e55a8150ed388221deefb09657948ec5d3bfd79
│   ├── VERSION
│   ├── json
│   └── layer.tar
├── 824341e8ce2bb06b616b9116547f7fff6df15d72b9a9465e20b97e055e2747ec
│   ├── VERSION
│   ├── json
│   └── layer.tar
├── 97f72354b5a6c180f16ebf83b43d4ba876c0c70ae99bde2a06a1fefaaafb8cf4.json
├── cb6b5a6bbd1d00cdb311f08b7c4e4ed6320eac032177c6722072d1371b0c63f0
│   ├── VERSION
│   ├── json
│   └── layer.tar
├── manifest.json
└── repositories
```

Tree 所看到的是Docker image結構

還有一種結構叫做OCI image和Docker image結構不同

Docker image 清單檔

```
$ cat imglayers/manifest.json | jq
```

Json quary


```
[
  {
    "Config":
"97f72354b5a6c180f16ebf83b43d4ba876c0c70ae99bde2a06a1fefaaafb8cf4.json",
    "RepoTags": [
      "myring:latest"
    ],
    "Layers": [

"579fb4fe2cc21a56bb5518ce0e55a8150ed388221deefb09657948ec5d3bfd79/layer.tar
",

"824341e8ce2bb06b616b9116547f7fff6df15d72b9a9465e20b97e055e2747ec/layer.tar
",

"cb6b5a6bbd1d00cdb311f08b7c4e4ed6320eac032177c6722072d1371b0c63f0/layer.ta
r"
    ]
  }
]
```

檢視 docker image 設定檔

```
$ cat imglayers/97f72354b5a6c180f16ebf83b43d4ba876c0c70ae99bde2a06a1fefaaafb8cf4.json | jq
{
  .......
  "history": [
    {
      "created": "2021-04-14T19:19:39.267885491Z",
      "created_by": "/bin/sh -c #(nop) ADD file:8ec69d882e7f29f0652d537557160e638168550f738d0d49f90a7ef96bf31787 in / "
    },
    {
      "created": "2021-04-14T19:19:39.643236135Z",
      "created_by": "/bin/sh -c #(nop)  CMD [\"/bin/sh\"]",
      "empty_layer": true
    },
    {
      "created": "2021-05-11T15:50:10.495312977Z",
      "created_by": "/bin/sh -c echo \"top.secret\" > /password.txt"
    },
    {
      "created": "2021-05-11T15:50:11.067783811Z",
      "created_by": "/bin/sh -c rm /password.txt"
    }
  ],
.........
```

.json : Dockerfile 所有執行命令，就算有刪除還是會有紀錄

檢視 Docker image Layer 0 內容

```
$ mkdir rootfs; tar -xf
imglayers/579fb4fe2cc21a56bb5518ce0e55a8150ed388221deefb0
9657948ec5d3bfd79/layer.tar -C rootfs

$ tree -L 1 rootfs
rootfs
├── bin
├── dev
├── etc
├── home
├── lib
├── media
├── mnt
├── opt
├── proc
├── root
├── run
├── sbin
├── srv
├── sys
├── tmp
├── usr
└── var
```

檢視 Docker image Layer 1 & 2 內容

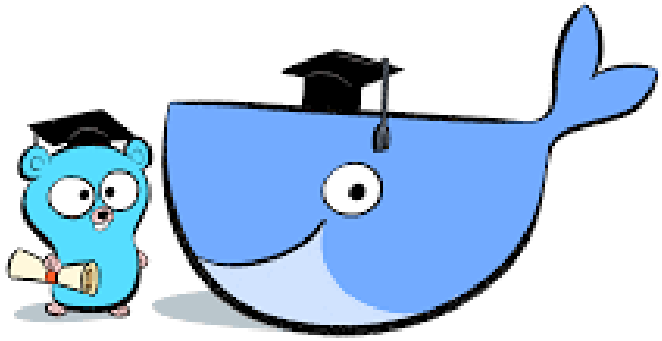
```
$ tar -xf
imglayers/afa90734739a2cd23aa24fb402fade696fe721765433bb0a5e99
465025bb9440/layer.tar -C rootfs

$ ls -al rootfs/password.txt
-rw-r--r-- 1 bigred bigred 11 May 11 23:50 rootfs/password.txt

$ tar -xf
imglayers/819e4e099fed990c3c2ccdfbcf869ce054921d089f0d9888fbf7c8
32e5978e30/layer.tar -C rootfs

$ find rootfs -name '*.txt'
rootfs/password.txt
rootfs/.wh.password.txt
```

Golang Application Image



開發 Golang 網站

```
$ cd ~/wulin; mkdir mygo; cd mygo

$ echo 'package main
import (
        "fmt"
        "log"
        "net/http"
)
func handler(w http.ResponseWriter, r *http.Request) {
        fmt.Fprintln(w, "Hello, 世界")
}
func main() {
        http.HandleFunc("/", handler)
        log.Fatal(http.ListenAndServe(":8888", nil))
}' > main.go

$ go mod init mygo

$ CGO_ENABLED=0 GOOS=linux GOARCH=amd64 go build -o
main
```

fmt : format 顯示格式
log : 記錄所有命令
net/http : 架網站系統,要符合http 協定

main : 程式從此開始執行

go mod = go module (相當於.net)

因為把相依檔包含在內,所以main就是自走砲

自製原生 Docker Image

```
bigred@ALP:~/wulin/mygo$ CGO_ENABLED=0 GOOS=linux GOARCH=amd64 go build -o main
bigred@ALP:~/wulin/mygo$ dir
total 5.9M
drwxr-sr-x 2 bigred bigred 4.0K May 12 14:09 .
drwxr-sr-x 4 bigred bigred 4.0K May 12 13:53 ..
-rw-r--r-- 1 bigred bigred   21 May 12 14:05 go.mod
-rwxr-xr-x 1 bigred bigred 5.9M May 12 14:09 main
```


```
$ dir
total 5.9M
drwxr-sr-x 2 bigred bigred 4.0K Apr  9 00:12 .
drwxr-sr-x 3 bigred bigred 4.0K Apr  9 00:09 ..
-rw-r--r-- 1 bigred bigred   21 Apr  9 00:09 go.mod
-rwxr-xr-x 1 bigred bigred 5.9M Apr  9 00:12 main
-rw-r--r-- 1 bigred bigred  234 Apr  9 00:09 main.go
```

```
$ echo 'FROM scratch
ADD main /
CMD ["/main"] ' > Dockerfile
```

Scratch：空白光碟片
ADD：把已有的檔案copy到指定目錄
CMD：如果沒給命令就跑內定的

```
$ docker build -t bigboss/goweb  .
```

```
$ docker images
REPOSITORY    TAG       IMAGE ID       CREATED          SIZE
goweb         latest    7f9652539fc0   14 minutes ago   6.13MB
```

建立 g1 container

```
$ docker run --rm --name g1 -d -p 8888:8888 goweb

$ curl http://localhost:8888
Hello, 世界

$ docker stop g1
```

前面是Alpine,後面是container

Multi-stage builds Docker Image

```
$ dktag golang | grep -E "^1.1.-*alpine$"
.....
1.15-alpine
1.16-alpine

$ echo 'FROM golang:1.16-alpine AS build
WORKDIR /src/
COPY main.go /src/
RUN go mod init mygo && CGO_ENABLED=0 go build -o /bin/demo

FROM scratch
COPY --from=build  /bin/demo  /bin/demo
ENTRYPOINT ["/bin/demo"] ' > Dockerfile

$ docker build -t goweb .
$ docker images
REPOSITORY    TAG          IMAGE ID       CREATED          SIZE
goweb         latest       a356ba6caa9e   7 seconds ago    7.41MB
golang        1.14-alpine  285e050bfca6   2 days ago       370MB
```

go與alpine 結合的版本

對原始碼做翻譯,讓後面的空光碟做使用,用完就刪除

ENTRYPOINT : 後面不能打命令,只能執行光碟的命令

做image就是為了container

再次建立 g1 container

```
$ docker run --rm --name g1 -p 8080:8888 -d goweb

$ curl http://localhost:8080
Hello, 世界

$ docker stop g1
```

光碟片做出來用Dockerfile
CMD：內定執行命令
ENTRYPOINT：只執行光碟的命令

Docker Image 內定的執行命令

檢視原廠 Image 內定執行命令

```
$  docker run –rm –it alpine
/ # ps
PID   USER     TIME  COMMAND
    1 root      0:00 /bin/sh
    7 root      0:00 ps
/ # exit

$ docker run --rm -it busybox
/ # exit
```

問題 ： 上述二個命令中，不需指定執行命令，一樣可以啟動貨櫃主機，請問內定執行命令為何 ？

使用 docker history 命令得知 Docker Image 內定執行命令

```
$ docker history alpine
IMAGE          CREATED       CREATED BY                                      SIZE
3fd9065eaf02   4 months ago  /bin/sh -c #(nop) CMD ["/bin/sh"]               0B
<missing>      4 months ago  /bin/sh -c #(nop) ADD file:093f0…               4.15MB

$ docker history busybox
IMAGE          CREATED        CREATED BY                                      SIZE
8ac48589692a   6 weeks ago    /bin/sh -c #(nop) CMD ["sh"]                    0B
<missing>      6 weeks ago    /bin/sh -c #(nop) ADD file:c94ab8f8614 …        1.15MB
```

撰寫 alpine.base Dockerfile

```
$ cd ~/wulin; mkdir base

$ echo 'FROM alpine:3.13.4
RUN apk update && apk upgrade && apk add --no-cache nano sudo wget curl \
    tree elinks bash shadow procps util-linux coreutils binutils findutils grep && \
    wget https://busybox.net/downloads/binaries/1.28.1-defconfig-
multiarch/busybox-x86_64 && \
    chmod +x busybox-x86_64 && mv busybox-x86_64 bin/busybox1.28 && \
    mkdir -p /opt/www && echo "let me go" > /opt/www/index.html

CMD ["/bin/bash"] ' > base/Dockerfile
```

用原廠一定要鎖定版本代號

procps:ps全功能

Busybox1.28 有httpd

mv busybox-x86_64 bin/busybox1.28 沒把原廠刪掉

寫Dockerfile linux命令,只用一個RUN就好

建立與測試 alpine.base image

建立 alpine.base image

```
$ docker build --no-cache  -t alpine.base base/
```

執行 alpine.base image 內定命令

```
$ docker run --rm -it alpine.base
bash-5.0# /bin/busybox1.28 | head -n 1
BusyBox v1.28.1 (2018-02-15 14:34:02 CET) multi-call binary.

bash-5.0# /bin/busybox | head -n 1
BusyBox v1.30.1 (2019-06-12 17:51:55 UTC) multi-call binary.

bash-5.0# busybox httpd -h /opt/www
httpd: applet not found

bash-5.0# busybox1.28 httpd -h /opt/www
bash-5.0# curl http://localhost
let me go

bash-5.0# exit
exit
```

測試 alpine.base image

```
$ docker run --rm --name b1 alpine.base busybox1.28 httpd -h /opt/www
$ docker exec -it b1 bash
Error: No such container: b1

$ docker run --rm --name b1 -d -p 80:80 alpine.base busybox1.28 httpd -f -h /opt/www
9a61cf33c20ca7d1ac463ab4a2c55676aac4f2c808d3d204e909edfc11d0c3ff

$ curl http://localhost
let me go

$ docker stop b1
```

自製 Alpine OpenSSH Server 的 Docker Image

撰寫 alpine.plus Dockerfile

```
$ cd ~/wulin; mkdir plus
$ echo $'
FROM alpine.base
RUN apk update && \
    apk add --no-cache openssh-server tzdata && \
    # 設定時區
    cp /usr/share/zoneinfo/Asia/Taipei /etc/localtime && \
    ssh-keygen -t rsa -P "" -f /etc/ssh/ssh_host_rsa_key && \
    echo -e \'Welcome to Alpine 3.13.4\\n\' > /etc/motd && \
    # 建立管理者帳號 bigred
    adduser -s /bin/bash -h /home/bigred -G wheel -D bigred && echo \'%wheel
ALL=(ALL) NOPASSWD: ALL\' >> /etc/sudoers && \
    echo -e "bigred\\nbigred\\n" | passwd bigred &>/dev/null && [ "$?" == "0" ]
&& echo "bigred ok"

EXPOSE 22

ENTRYPOINT ["/usr/sbin/sshd"]
CMD ["-D"] ' > plus/Dockerfile
```

time zone : container時區一定要同步

openssh 國際標準一定是22 port

sshd 內定背景執行,要道前景要打-D

[重要] ENTRYPOINT 所指定的執行命令, 在建立 Container 時強制執行此命令並且不可被其它命令取代

建立與使用 alpine.plus image

```
$ docker build --no-cache  -t alpine.plus plus/

$ docker run --rm --name s1 -h s1 alpine.plus sh
Extra argument sh.
```

[重要] 因使用 entrypoint 宣告內定命令, 便無法自行指定執行命令

建立 s1 container

```
$ docker run  --rm --name s1 -h s1 -d -p 22100:22 alpine.plus

登入 s1 貨櫃主機
$ ssh bigred@localhost -p 22100
bigred@192.168.122.47's password: bigred

$ ps aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.3  0.0   4328  2944 pts/0    Ss+  09:11   0:00 /usr/sbin/sshd -D
root         9  0.1  0.0   4356  3500 ?        Ss   09:12   0:00 sshd: bigred [priv]
bigred      11  0.0  0.0   4356  2340 ?        R    09:12   0:00 sshd: bigred@pts/1
........

$ sudo kill -9 1
sudo: setrlimit(RLIMIT_CORE): Operation not permitted

$ exit

$ docker stop s1
```

Docker Image 備份與還原

Docker Image 備份

```
$ cd ~/wk/wulin

$ docker history alpine.plus
IMAGE               CREATED             CREATED BY                                      SIZE                COMMENT
eb14bc497f7c        5 hours ago         /bin/sh -c #(nop)  CMD ["-D"]                   0B
1e05936fd376        5 hours ago         /bin/sh -c #(nop)  ENTRYPOINT ["/usr/sbin/ss…   0B
c5c25dc4a60c        5 hours ago         /bin/sh -c #(nop)  EXPOSE 22                    0B
6a42f8c6183f        5 hours ago         /bin/sh -c apk update &&     apk add --no-ca…   4MB
37dfd3d3318e        6 hours ago         /bin/sh -c #(nop)  CMD ["/bin/bash"]            0B
86c8cd04a262        6 hours ago         /bin/sh -c apk update && apk upgrade && apk …   36.4MB
f70734b6a266        2 weeks ago         /bin/sh -c #(nop)  CMD ["/bin/sh"]              0B
<missing>           2 weeks ago         /bin/sh -c #(nop) ADD file:b91adb67b670d3a6f…   5.61MB

備份 Image
$ docker save alpine.plus > alpine.plus.tar
```

Docker Image 還原

還原 Image

```
$ docker rmi alpine.plus

$ docker load < alpine.plus.tar

$ docker history alpine.plus
IMAGE               CREATED             CREATED BY                                      SIZE
COMMENT
eb14bc497f7c        5 hours ago         /bin/sh -c #(nop)  CMD ["-D"]                   0B
1e05936fd376        5 hours ago         /bin/sh -c #(nop)  ENTRYPOINT ["/usr/sbin/ss…   0B
c5c25dc4a60c        5 hours ago         /bin/sh -c #(nop)  EXPOSE 22                    0B
6a42f8c6183f        5 hours ago         /bin/sh -c apk update &&     apk add --no-ca…   4MB
37dfd3d3318e        6 hours ago         /bin/sh -c #(nop)  CMD ["/bin/bash"]            0B
86c8cd04a262        6 hours ago         /bin/sh -c apk update && apk upgrade && apk …
36.4MB
f70734b6a266        2 weeks ago         /bin/sh -c #(nop)  CMD ["/bin/sh"]              0B
<missing>           2 weeks ago         /bin/sh -c #(nop) ADD file:b91adb67b670d3a6f…
5.61MB
```

【重點】 還原的 image 的目錄架構, 與原先 Image目錄架構一樣

Docker Container 備份與還原

Docker Container 備份與還原

```
$ docker run --name s2 -h s2 -d alpine.plus
6e52bd5aee18c0aed8dbb17920037be0b9109158329b401c56b4f64417c2d784
```

[重要] 使用上述命令建立 s2 Container, 這個 Container 有啟動 openssh server, 所以這個 Container 有 openssh 的執行狀態資訊檔, 這時使用 docker export 命令匯出的 Tar 檔中就會有殘留 openssh 的執行暫存檔, 以至後續做出的 image 無法啟動 openssh server, 所以必須先關閉 s2 container, 才可備份此 container

```
$ docker stop s2
$ docker export s2 > s2.tar
```

把contianer備份(消磁)

```
$ docker rm s2 && docker rmi alpine.plus
$ cat s2.tar | docker import - alpine.plus &>/dev/null

$ docker history alpine.plus
IMAGE          CREATED         CREATED BY   SIZE      COMMENT
00a487305c5b   12 seconds ago               44.3MB    Imported from -
```

使用重製 alpine.plus image

```
$ docker run --name s2 -h s2 -d alpine.plus
docker: Error response from daemon: No command specified.
See 'docker run --help'.
```

[重要] 重製後的 alpine.plus image 必須指定 執行命令

```
$ docker run --name s2 -h s2 -d alpine.plus /usr/sbin/sshd -D

$ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
d4f93f15556c        alpine.plus         "/usr/sbin/sshd -D"   5 seconds ago       Up
3 seconds                               s2

$ docker rm -f s2
```

Image Labels 實務應用

Image label 實務應用

```
$ mkdir label; nano label/Dockerfile
FROM alpine.plus
LABEL RUN="docker run --name test -h test -d alpine.plus
/usr/sbin/sshd -D"

$ docker build --no-cache -t myssh label/.
$ docker inspect -f  '{{ .Config.Labels.RUN }}'  myssh
docker run --name test -h test -d alpine.plus /usr/sbin/sshd -D

建立 test Container
$ `docker inspect -f  '{{ .Config.Labels.RUN }}'  myssh`
a4727566c30bdd163b5ff04c71d70a768726c3db1e25af97055180f8b2
61e8f0

[註] 以上命令可解決不知如何測試與執行這 image 的困境, 並可在
docker 及 Podman 這二個系統中執行

$ docker ps -a | grep test

$ docker rm -f test
```