

# Linux Container Note and Review

## ▼ Docker

```
#查詢官方映像檔案  
docker search keyword  
eg. docker search ubuntu  
  
#查看已有的映像檔  
docker images  
  
#取得映像檔  
docker pull image_name  
  
#刪除映像檔  
docker rmi image_name
```

### Docker (一) : 介紹與安裝

Docker與VM相似，但是執行的最底單位不是OS，所有的Docker...



<https://lufor129.medium.com/docker-介紹與安裝-5b9183409ce3>

### Docker (二) : 基本操作Image&Container

簡單講解一下Docker的概念，最重要的是兩個地方: Image, Container。如果熟悉物件導向，可以把Image想像成Class而Container就是Image產生的Object。關係如下圖。



<https://lufor129.medium.com/docker-二-基本操作image-container-1bddaee3fa2>

### Docker (三) : 基本操作Volume & Net

Docker內空間的資料能不能保存下來？當我關閉一個Container後下次重開一個新的Container能不能夠承繼之前的檔案？其實這些都是可以的。接下來介紹一下Docker Volume，這是docker資料持久層。



<https://lufor129.medium.com/docker-三-基本操作volume-net-5f323965486>

## ▼ From docker to podman



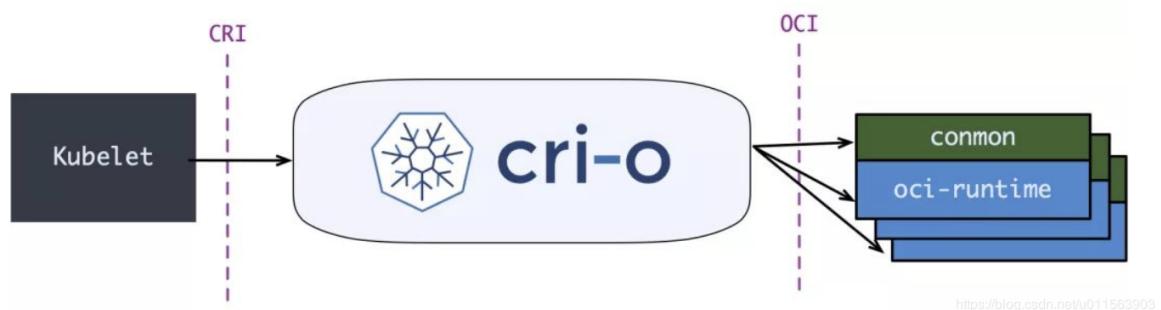
<https://blog.csdn.net/u011563903>

1. Kubelet 透過CRI 介面（gRPC）呼叫dockershim，請求建立一個容器。CRI 即容器執行時間介面（Container Runtime Interface），在這一步驟中，Kubelet 可以視為一個簡單的CRI Client，而dockershim 就是接收請求的Server。目前dockershim 的程式碼其實是內嵌在Kubelet 中的，所以接收呼叫的湊巧就是Kubelet 進程；
2. dockershim 收到請求後，轉換成Docker Daemon 能聽懂的請求，發到Docker Daemon 上請求建立一個容器。
3. 因此Docker Daemon 仍然不能幫我們創建容器，而是要請求containerd 創建一個容器
4. containerd 收到請求後，並不會自己直接去操作容器，而是創建一個叫做containerd-shim 的進程，讓containerd-shim 去操作容器。這是因為容器程序需要一個父進程來做諸如收集狀態，維持stdin 等fd 開啟等工作。而假如這個父進程就是containerd，那每次containerd 掛掉或升級，整個宿主機上所有的容器都得退出了。而引入了containerd-shim 就規避了這個問題（containerd 和shim 並不是父子進程關係）

5. 我們知道創建容器需要做一些設定namespaces 和cgroups，掛載root filesystem 等等操作，而這些事該怎麼做已經有了公開的規範了，那就是OCI (Open Container Initiative，開放容器標準)。它的一個參考實作叫做runC。於是，containerd-shim 在這一步驟需要呼叫runC 這個命令列工具，來啟動容器

6. runC 啟動完容器後本身會直接退出，containerd-shim 則會成為容器進程的父進程，負責收集容器進程的狀態，上報給containerd，並在容器中pid 為1 的進程退出後接管容器中的子進程進行清理，確保不會出現殞屍行程。

runc 希望提供一個“ OCI 套件”，它只是一個根檔案系統和一個config.json 檔案。



<https://blog.csdn.net/u011563903>

本質上是想要去掉docker，把中間的組合在一起

#### ▼ Podman

1. docker 得要使用 root 的身份去管理，所以所有的容器都是 root 的權限！對於某些人來說，實在不太喜歡。podman 可以在一般帳號身份來管理容器！而且也能夠管理由 docker 所建立的映像檔與容器
2. 基本指令與docker大致上相同

```
#啟動container
podman run
#執行container內的指令
podman exec
#ps 或者是加上 -a 的參數，都可以看到運作中或者是全部的 podman container
podman ps
```

#### 專題 - 使用 podman 進行 container 管理

docker 是容器管理的最早企劃之一，所以很多的管理大多以 docker 為依據來進行的。但是，  
docker 必須要使用 root 的身份去啟動與管理，對於某些特別注意資安的朋友來說，感覺就是怕怕的。  
因此，Red Hat 推出了 podman 搭配 runc 來進行輕量級虛擬化，也就是容器的管理與運行！

[https://dic.vbird.tw/network\\_project/zunit12.php](https://dic.vbird.tw/network_project/zunit12.php)

3. `save` will fetch an image

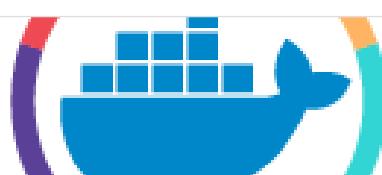
`export` will fetch the whole container

#### ▼ Container registry

##### Docker run reference

Configure containers at runtime

 <https://docs.docker.com/engine/reference/run/>



1. docker search docker.io/library/alpine:latest

NAME	DESCRIPTION	STARS	OFFICIAL	AUTOMATED
d3fk/s3cmd	A simple s3cmd S3 client installed on the Al... foggycam w/ alpine:latest including python3,...	15		
ryanburketllc/foggycam-alpine	Image based on arm32v6/alpine:latest with ph...	0		
intrepidde/rpi-phpmyadmin	official alpine:latest base-image with gosu,...	0		
3x3cut0r/alpine	Docker image built on alpine:latest includin...	1		
freme/docker_alpine_doxxygen	IKEV2 vpn based on alpine:latest and strongs...	0	[OK]	
playniumu/ikev2-vpn	Raspberry pi build of eggdrop based on arm32...	1	[OK]	
acrelle/rpi-eggdrop	Raspberry pi build of eggdrop based on arm32...	0		
opillion/node	based on: alpine:latest	0		
sisnet/alpine-curl	Builds a docker image from alpine:latest add...	0		[OK]
idefix/docker-owncloud-client	headless nextcloud-client on Alpine:latest	4		[OK]
artemklevtsov/r-alpine	R docker images based on the alpine:latest	3		[OK]
ywatase/ansible	ansible on alpine:latest	0		[OK]
fixiu/jdk-alpine	alpine:latest oracle jdk 8u211	1		
mkoppelan/etcctltool	Image based on alpine:latest to run etcdtool	0		
bmauter/alpine-samba-client	alpine:latest + samba-client	0		
robonuka/sslh	Alpine:latest + SSLH v1.2.0	0		
softf/alpine	alpine:latest with mirror aliyun.com	0		
gymnae/alpine-base	A pico base image suited for my needs based ...	0		[OK]
mheindl/alpine-aws	Just a simple Image based on alpine:latest c...	1		[OK]
gymnae/webserverbase	Web server image with alpine:latest origin -...	0		[OK]
paulgoio/toolkit	alpine:latest image with curl, bash, openssh...	0		
robonuka/socks	Alpine:latest + Dante 1.4.2 Socks5 server	0		
biosniper/jackett	Docker build of Jackett using alpine:latest	0		[OK]
neverlock/alpine-wrk	wrk build on alpine:latest	0		[OK]
lazaruslongone/alpine-openrc-base	A simple container that uses alpine:latest a...	0		[OK]

## 2. docker pull alpine:latest

```
zhigang ~ docker pull alpine:latest
latest: Pulling from library/alpine
96526aa774ef: Pull complete
Digest: sha256:eecce025e432126ce23f223450a0326fbebe39cdf496a85d8c016293fc851978
Status: Downloaded newer image for alpine:latest
docker.io/library/alpine:latest

What's Next?
View a summary of image vulnerabilities and recommendations → docker scout quickview alpine:latest
```

## 3. docker images

注意:pull下來是一個image，不能用docker ps -a(這是察看container)

```
zhigang ~ docker images
REPOSITORY          TAG      IMAGE ID   CREATED        SIZE
httpd              2.4      75a48b16cd56  2 days ago   168MB
alpine              latest   8ca4688f4f35  3 weeks ago   7.34MB
docker/welcome-to-docker  latest   912b66cf46e   4 months ago  13.4MB
postgres            latest   ceccf204404e  6 months ago  379MB
```

## 4. docker image inspect alpine

```
zhigang ~ docker image inspect alpine
[{"Id": "sha256:8ca4688f4f356596b5ae539337c9941abc78eda10021d35cbc52659c74d9b443",
 "RepoTags": [
     "alpine:latest"
 ],
 "RepoDigests": [
     "alpine@sha256:eece025e432126ce23f223450a0326fbebe39cdf496a85d8c016293fc851978"
 ],
 "Parent": "",
 "Comment": "",
 "Created": "2023-09-28T21:19:27.801479409Z",
 "Container": "6fd75b3ad2cde1fb91e80db8f58701b0d6710f1b88c383492c28612918d81549",
 "ContainerConfig": {
     "Hostname": "6fd75b3ad2cd",
     "Domainname": "",
     "User": "",
     "AttachStdin": false,
     "AttachStdout": false,
     "AttachStderr": false,
     "Tty": false,
     "OpenStdin": false,
     "StdinOnce": false,
     "Env": [
         "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
     ],
     "Cmd": [
         "/bin/sh",
         "-c",
         "#(nop) ",
         "CMD [\"/bin/sh\"]"
     ],
     "Image": "sha256:6913aeba049e6e1fb16d610436f78c039eec4a06bce4bfaffa43ae89fcf67942",
     "Volumes": null,
     "WorkingDir": "",
     "Entrypoint": null,
     "OnBuild": null
 }
```

## ▼ Dockerfile

```
FROM centos:7
MAINTAINER Neil

RUN yum install -y wget
RUN cd /
ADD jdk-12.0.2_linux-x64_bin.tar.gz /
RUN wget http://apache.stu.edu.tw/tomcat/tomcat-7/v7.0.94/bin/apache-tomcat-7.0.94.tar.gz
RUN tar zxvf apache-tomcat-7.0.94.tar.gz

ENV JAVA_HOME=/jdk-12.0.2
ENV PATH=$PATH:/jdk-12.0.2/bin

CMD ["/apache-tomcat-7.0.94/bin/catalina.sh", "run"]

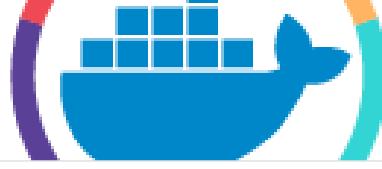
#FROM： 使用到的 Docker Image 名稱，今天使用 CentOS
#MAINTAINER： 用來說明，撰寫和維護這個 Dockerfile 的人是誰，也可以給 E-mail 的資訊
#RUN： RUN 指令後面放 Linux 指令，用來執行安裝和設定這個 Image 需要的東西
#ADD： 把 Local 的檔案(要與Dockerfile同目錄)複製到 Image 裡，如果是 tar.gz 檔複製進去 Image 時會順便自動解壓縮。Dockerfile 另外還有一個複製檔案的指令
#ENV： 用來設定環境變數
#CMD： 在指行 docker run 的指令時會直接呼叫開啟 Tomcat Service
#ARG： 在build時候是可以從外部以--build-arg帶入的變數，讓build的動作可結合外部的指令給定一些建構時候所需的參數
#EXPOSE： The EXPOSE instruction informs Docker that the container listens on the specified network ports at runtime. You can specify
```

## Docker 範例 - HackMD

# Docker 範例 在要離開 docker container 的 terminal 時有一個坑。就是如果輸入 exit 指令時，container 會被關閉，如下圖：

 <https://hackmd.io/@karta134033/BJJS6RmfH>

## Dockerfile reference

Dockerfile reference Docker can build images automatically by reading the instructions from a Dockerfile. A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image. This page describes the commands you can use in a Dockerfile. Format Here is the  <https://docs.docker.com/engine/reference/builder/>

## ▼ Docker-compose

- 實務上一個服務，一定由眾多 service 共同運作若只使用 `docker run`，則勢必寫 Bash 來管理 4 個 service，還必須考慮：

- 4 個 service 必須在同一個虛擬 network 下
- 4 個 service 的啟動順序... 等問題

Docker 為此提出 Docker Compose 概念，在 `docker-compose.yml` 檔描述各 service 間的參數與關係

eg.

```
version: "3"

services:
  netcore:
    image: microsoft/dotnet
    container_name: MyNETCore
    volumes:
      - ${NETCORE_HOST_DIR}:/code/
    tty: true
    networks:
      - netcore-dev
    depends_on:
      - postgres

  postgres:
    image: postgres
    container_name: MyPostgres
    volumes:
      - ${POSTGRES_HOST_DIR}/data:/var/lib/postgresql/data
    expose:
      - "5432"
    ports:
      - "${POSTGRES_PORT}:5432"
    environment:
      - POSTGRES_DB=${POSTGRES_DB}
      - POSTGRES_USER=${POSTGRES_USER}
      - POSTGRES_PASSWORD=${POSTGRES_PASSWORD}
    networks:
      - netcore-dev

networks:
  netcore-dev:
```

深入淺出 Dockerfile 與 Docker Compose

兩個初學者最容易卡關的概念

👉 <https://old-oomusou.goodjack.tw/docker/dockerfile-dockercompose/>



## 2. dockerfile VS. docker-compose

- `Dockerfile` 是用來描述 image，也就是如何產生客製化的 image，通常用來安裝 package，將檔案複製進 image 用
- `docker-compose.yml` 是用來描述 container，也就是管理一個以上的 container，彼此串連，把同一組架構寫在一起，通常用來設定 container 參數，設定 container 的網路，設定 container 啟動順序或 service 的環境變數 ... 等

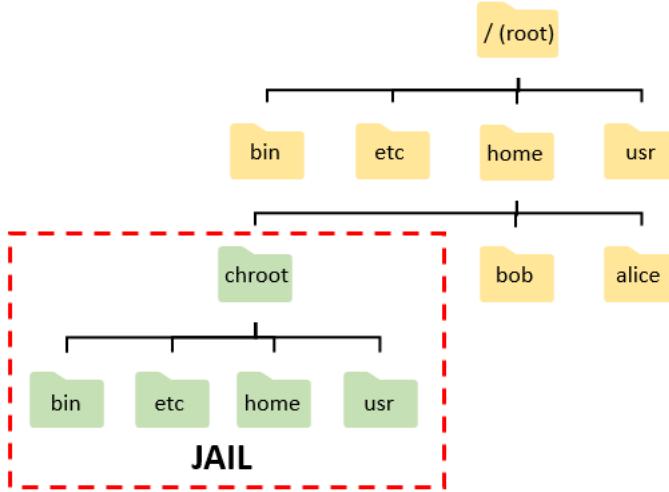
`services` 提供哪些的服務

`volumes` 絶對路徑

`ports` 在哪個port提供服務

## ▼ Before docker

- `chroot` (change root)



## 2. `schroot` (secure chroot)

is an enhancement and management tool for the `chroot`

## 3. `fakeroot`

simulates the effect of running a command with superuser (root) privileges without actually requiring root access

### ▼ container == isolated linux process

#### 1. `namespace`

Namespace 其中一個應用是「隔離」同一個作業系統上的不同行程。把一個行程放在某一個種類的 namespace 中，它就會只看得到該 namespace 下看得到的資源。儘管處在該 namespace 中的行程可能以為自己可以存取整個根目錄、以為自己是它 root，但在 namespace 以外的行程看來只是一個普通權限的行程。而這也是容器 (container) 的基礎。

系統程式設計 - Namespaces - HackMD  
# 系統程式設計 - Namespaces [TOC] ## References \*Michael Kerrisk\* 是 TLPI 的作者，也是 `man-pages` 的維護者～所以可能聽  
<https://hackmd.io/@0xff07/r1wCFz0ut>

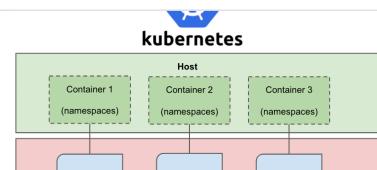


#### 2. `cgroup` (Control Groups)

cgroups 主要提供有底下四個功能：

- **Resource Limiting:** Group 可以設定 Memory 的使用上限，其中也包含 File System 的 Cache
- **Prioritization:** 不同的 Group 可以擁有不同的 CPU 跟 Disk I/O 使用優先順序
- **Accounting:** 計算 Group 內的資源使用狀況，例如可以拿來當作計費的依據
- **Control:**凍結或是重啟一整個 Group 的 Process

第一千零一篇的 cgroups 介紹  
最近自己打算開始把手上的系統從 cgroup v1 轉換到 v2 在設定的過程中，發現自己對於其一知半解，因此在查詢資料的過程中，順便將筆記整理成文章，讓跟我一樣有興趣的人可以快速理解其底層概念，了解 Rootless Container 跟他的淵源，最後提到目前跟整個...  
<https://medium.com/starbugs/第一千零一篇的-cgroups-介紹-a1c5005be88c>



#### 3. `pod`

Pod 是在 k8s 最基本的組成單位(也是最小的可佈署單位)，實際在 k8s 上運行的很多 resource object 都是以 pod 型式存在，它封裝了許多不同的資源，也因此每個 pod 都有以下特性：

- 包含一到多個 container
- 同一個 pod 的 container 都共享相同的檔案系統 & volume ... 等資源

- container 共享相同的 network namespace(container 之間可以透過 `localhost` + `port number` 互相通訊), 且有獨一無二的 IP address
- container 之間也可以透過進程間通信, 例如: SystemV or POSIX shared memory
- container 共享 pod 中的 volume resource
- pod 中的 container 總是被同時調度 & 有共同的運行環境

## Pods

Pods are the smallest deployable units of computing that you can create and manage in Kubernetes. A Pod (as in a pod of whales or pea pod) is a group of one or more containers, with shared storage and network resources, and a specification for how to run the containers. A Pod's contents are

<https://kubernetes.io/docs/concepts/workloads/pods/>

# kubernetes

## [Kubernetes] Pod 的設計 & 相關運作機制

本篇文章的主題圍繞在 Kubernetes 中的 Pod resource object, 其中包含 Pod 的設計 & 相關運作機制, 並對這些概念做一些概略的介紹

<https://godeon.github.io/blog/Kubernetes/k8s-Pod-Overview/>

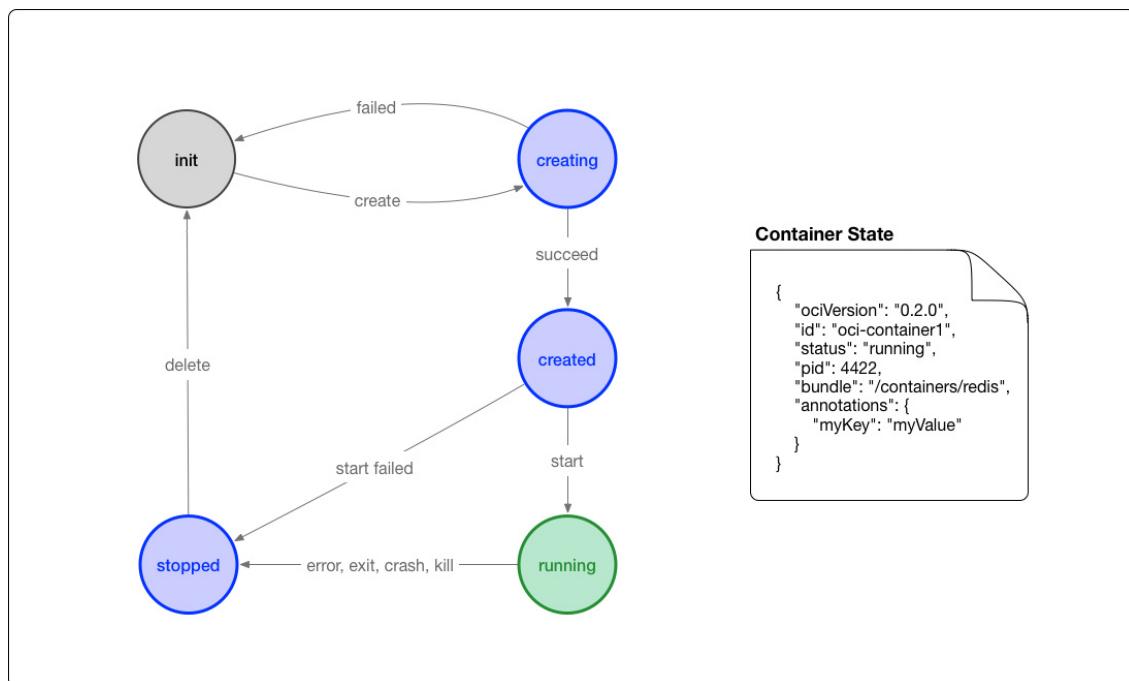


## ▼ Container specs

1. OCI定義了容器運行時標準, runC是Docker按照開放容器格式標準（OCF, Open Container Format）制定的一種具體實現

### 2. OCI runtime spec

OCI 對容器 runtime 的標準主要是指定容器的運行狀態, 和 runtime 需要提供的命令



- **creating** : 使用 `create` 命令創建容器, 這個過程稱為創建中
- **created** : 容器創建出來, 但是還沒有運行, 表示鏡像和配置沒有錯誤, 容器能夠運行在當前平臺
- **running** : 容器的運行狀態, 裡面的進程處於 up 狀態, 正在執行使用者設定的任務
- **stopped** : 容器運行完成, 或者運行出錯, 或者 `stop` 命令之後, 容器處於暫停狀態。這個狀態, 容器還有很多資訊保存在平臺中, 並沒有完全被刪除