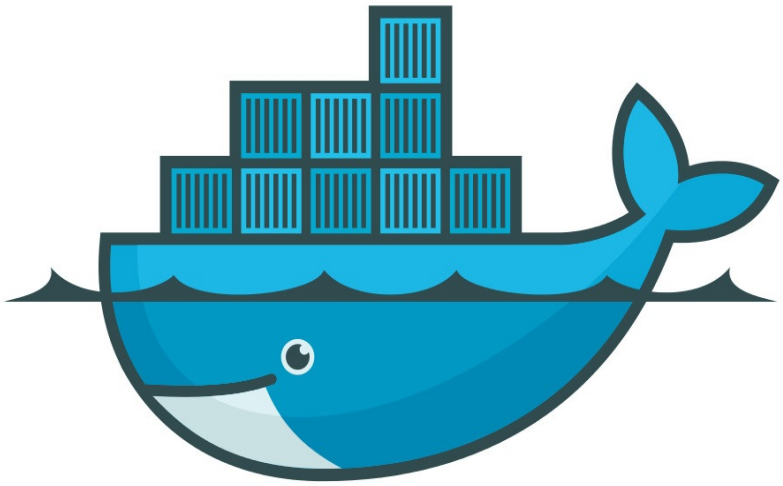


docker

--從入門到實踐

Table of Contents

Docker —— 從入門到實踐

v0.2.9

Docker 是個偉大的專案，它徹底釋放了虛擬化的，讓應用程式的分派、部署和管理都變得前所未有的有效率和輕鬆！

本書既適用於具備基礎 Linux 知識的 Docker 初學者，也可供希望理解原理和實作的進階使用者參考。同時，書中給出的實踐案例，可供在進行實際部署時借鑒。

本書源於 WaitFish 的《Docker 學習手冊 v1.0》內容。後來，yeasy 根據最新 Docker 版本對內容進行了修訂和重寫，並增加內容；經協商將所有內容開源，採用網路合作的方式進行維護。

前六章爲基礎內容，供使用者理解 Docker 的基本概念和操作；7 ~ 9 章介紹一些進階操作；第 10 章給出典型的應用場景和實踐案例；11 ~ 13 章介紹關於 Docker 實作的相關技術。

最新版本線上閱讀：正體版、簡體版 或 DockerPool。

另外，歡迎加入 Docker.Taipei 和 Meetup ，分享 Docker 資源，交流 Docker 技術。

本書原始碼在 Github 上維護，歡迎參與：
https://github.com/philipz/docker_practice。

感謝所有的 貢獻者。

主要版本歷史

- 0.3: 2014-10-TODO
  - 完成倉庫章節；
  - 重寫安全章節；
  - 修正底層實作章節的架構、命名空間、控制組、檔案系統、容器格式等內容；
  - 新增對常見倉庫和鏡像的介紹；
  - 新增 Dockerfile 的介紹；
  - 重新校訂中英文混排格式。

- 0.2: 2014-09-18
  - 對照官方文檔重寫介紹、基本概念、安裝、鏡像、容器、倉庫、資料管理、網路等章節；
  - 新增底層實作章節；
  - 新增命令查詢和資源連結章節；
  - 其它修正。
- 0.1: 2014-09-05
  - 新增基本內容;
  - 修正錯別字和表達不通順的地方。

貢獻力量

如果想做出貢獻的話，你可以：

- 幫忙校對，挑錯別字、語病等等
- 提出修改建議
- 提出術語翻譯建議

翻譯建議

如果你願意一起校對的話，請仔細閱讀：

- 使用 markdown 進行翻譯，文件名必須使用英文，因爲中文的話 gitbook 編譯的時候會出問題
- 引號請使用「」和『』
- fork 過去之後新建一個分支進行翻譯，完成後 pull request 這個分支，沒問題的話我會合併到 master 分支中
- 有其他任何問題都歡迎發 issue，我看到了會盡快回覆

謝謝！

關於術語

翻譯術語的時候請參考這個流程：

- 盡量保證與台灣習慣術語和已翻譯的內容一致
- 盡量先搜尋，一般來說程式語言的大部分術語是一樣的，可以參考這個網站

- 如果以上兩條都沒有找到合適的結果，請自己決定一個合適的翻譯或者直接使用英文原文，後期校對的時候會進行統一
- 校稿時，若有發現沒有被翻譯成台灣術語的大陸術語，可以將它新增到 translation.json 中
- 可以主動提交替換過的文本給我，或是僅提交新增過的 translation.json 也可，我會再進行文本的替換
- 請務必確定提交的翻譯對照組不會造成字串循環替代（ex: 因爲「類」->「類別」，造成下次再執行自動翻譯時「類別」又變成「類別別」）

對翻譯有任何意見都歡迎發 issue，我看到了會盡快回覆

參加步驟

參考 Swift 說明，欲翻譯章節就直接在 github 上發 Issue 中註明或直接發Pull Request 修改。m( )m 有些朋友可能不太清楚如何幫忙翻譯，我這裡寫一個簡單的流程，大家可以參考一下：

1. 首先 fork 我的專案
2. 把 fork 過去的專案也就是你的專案 clone 到你的本地
3. 在命令行執行 `git branch develop` 來建立一個新分支
4. 執行 `git checkout develop` 來切換到新分支
5. 執行 `git remote add upstream https://github.com/philipz/docker_practice` 把我的庫新增爲遠端庫
6. 執行 `git remote update` 更新
7. 執行 `git fetch upstream master` 拉取我的庫的更新到本地
8. 執行 `git rebase upstream/master` 將我的更新合並到你的分支

這是一個初始化流程，只需要做一遍就行，之後請一直在 develop 分支進行修改。

如果修改過程中我的庫有了更新，請重復 6、7、8 步。

修改之後，首先 push 到你的庫，然後登錄 GitHub，在你的 repo 的首頁可以看到一個 `pull request` 按鈕，點擊它，填寫一些說明資訊，然後提交即可。

原出處及參考資料

1. Docker —— 从入门到实践
2. 《The Swift Programming Language》正體中文版

簡介

本章將帶領你進入 Docker 的世界。

什麼是 Docker？

用它會帶來什麼樣的好處？

好吧，讓我們帶著問題開始這神奇之旅。

什麼是 **Docker**

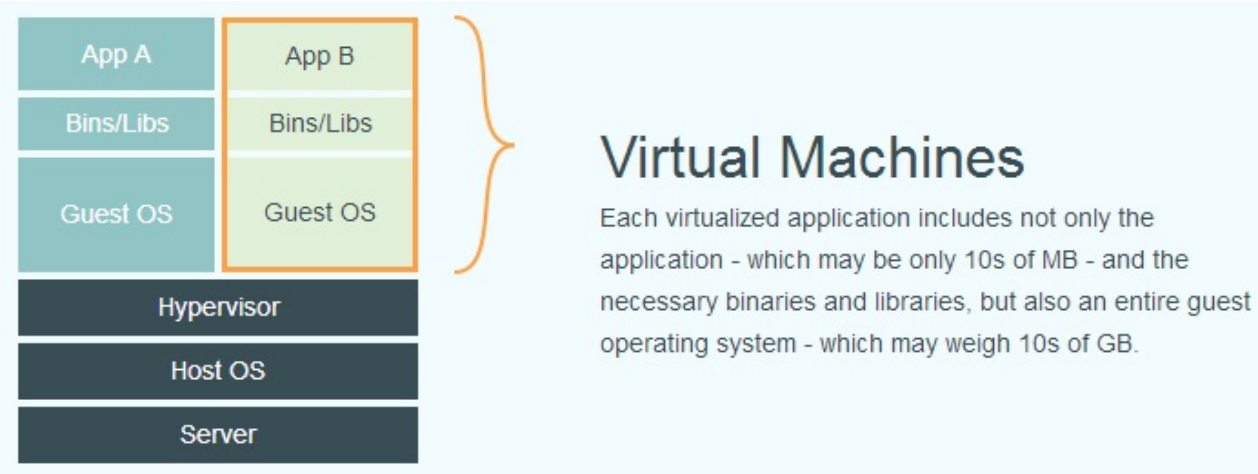
Docker 是一個開源專案，誕生於 2013 年初，最初是 dotCloud 公司內部的一個業餘專案。它基於 Google 公司推出的 Go 語言實作。 專案後來加入了 Linux 基金會，遵從了 Apache 2.0 協議，原始碼在 GitHub 上進行維護。

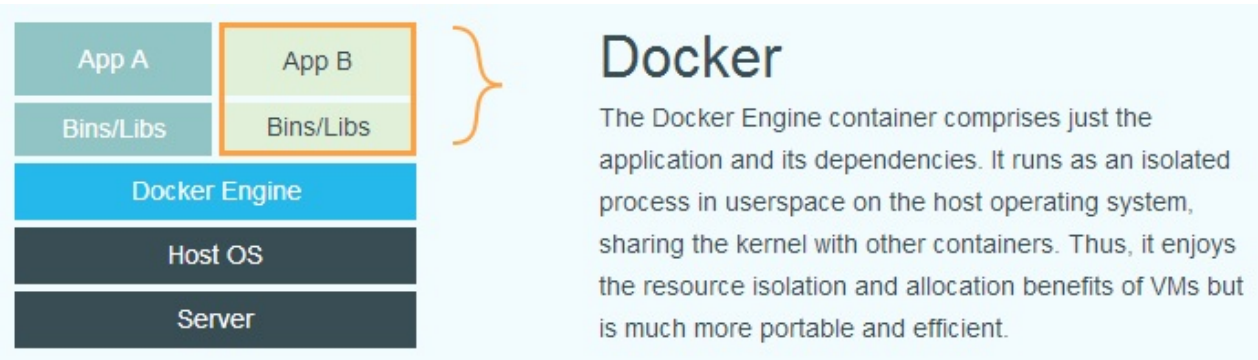
Docker 自開源後受到廣泛的關注和討論，以至於 dotCloud 公司後來都改名爲 Docker Inc。Redhat 已經在其 RHEL6.5 中集中支援 Docker；Google 也在其 PaaS 產品中廣泛應用。

Docker 專案的目標是實作輕量級的作業系統虛擬化解決方案。 Docker 的基礎是 Linux 容器（LXC）等技術。

在 LXC 的基礎上 Docker 進行了進一步的封裝，讓使用者不需要去關心容器的管理，使得操作更爲簡便。使用者操作 Docker 的容器就像操作一個快速輕量級的虛擬機一樣簡單。

下面的圖片比較了 Docker 和傳統虛擬化方式的不同之處，可見容器是在作業系統層面上實作虛擬化，直接使用本地主機的作業系統，而傳統方式則是在硬體層面實作。





爲什麼要使用 **Docker**？

作爲一種新興的虛擬化方式，Docker 跟傳統的虛擬化方式相比具有眾多的優勢。

首先，Docker 容器的啓動可以在秒級實作，這相比傳統的虛擬機方式要快得多。其次，Docker 對系統資源的使用率很高，一台主機上可以同時執行數千個 Docker 容器。

容器除了執行其中應用外，基本不消耗額外的系統資源，使得應用的效能很高，同時系統資源消耗更少。傳統虛擬機方式執行 10 個不同的應用就要啓動 10 個虛擬機，而 Docker 只需要啓動 10 個隔離的應用即可。

具體說來，Docker 在以下幾個方面具有較大的優勢。

更快速的交付和部署

對開發和維運（DevOps）人員來說，最希望的就是一次建立或設定，可以在任意地方正常執行。

開發者可以使用一個標準的映像檔來建立一套開發容器，開發完成之後，維運人員可以直接使用這個容器來部署程式碼。 Docker 可以快速建立容器，快速迭代應用程式，並讓整個過程全程可見，使團隊中的其他成員更容易理解應用程式是如何建立和工作的。 Docker 容器很輕很快！容器的啓動時間是秒級的，大量地節約開發、測試、部署的時間。

更有效率的虛擬化

Docker 容器的執行不需要額外的虛擬化支援，它是核心層級的虛擬化，因此可以實作更高的效能和效率。

更輕鬆的遷移和擴展

Docker 容器幾乎可以在任意的平台上執行，包括實體機器、虛擬機、公有雲、私有雲、個人電腦、伺服器等。 這種兼容性可以讓使用者把一個應用程式從一個平台直接遷移到另外一個。

更簡單的管理

使用 Docker，只需要小小的修改，就可以替代以往大量的更新工作。所有的修改都以增量的方式被分發和更新，從而實作自動化並且有效率的管理。

對比傳統虛擬機總結

特性	容器	虛擬機
啓動	秒級	分鐘級
硬碟容量	一般爲 MB	一般爲 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

基本概念

Docker 包括三個基本概念

- 映像檔（Image）
- 容器（Container）
- 倉庫（Repository）

理解了這三個概念，就理解了 Docker 的整個生命週期。

Docker 映像檔

Docker 映像檔就是一個唯讀的模板。

例如：一個映像檔可以包含一個完整的 ubuntu 作業系統環境，裡面僅安裝了 Apache 或使用者需要的其它應用程式。

映像檔可以用來建立 Docker 容器。

Docker 提供了一個很簡單的機制來建立映像檔或者更新現有的映像檔，使用者甚至可以直接從其他人那裡下載一個已經做好的映像檔來直接使用。

Docker 容器

Docker 利用容器來執行應用。

容器是從映像檔建立的執行實例。它可以被啓動、開始、停止、刪除。每個容器都是相互隔離的、保證安全的平台。

可以把容器看做是一個簡易版的 Linux 環境（包括root使用者權限、程式空間、使用者空間和網路空間等）和在其中執行的應用程式。

*註：映像檔是唯讀的，容器在啓動的時候建立一層可寫層作爲最上層。

Docker 倉庫

倉庫是集中存放映像檔檔案的場所。有時候會把倉庫和倉庫註冊伺服器（Registry）混爲一談，並不嚴格區分。實際上，倉庫註冊伺服器上往往存放著多個倉庫，每個倉庫中又包含了多個映像檔，每個映像檔有不同的標籤（tag）。

倉庫分爲公開倉庫（Public）和私有倉庫（Private）兩種形式。

最大的公開倉庫是 Docker Hub，存放了數量龐大的映像檔供使用者下載。 大陸的公開倉庫包括 Docker Pool 等，可以提供大陸使用者更穩定快速的存取。

當然，使用者也可以在本地網路內建立一個私有倉庫。

當使用者建立了自己的映像檔之後就可以使用 `push` 命令將它上傳到公有或者私有倉庫，這樣下次在另外一台機器上使用這個映像檔時候，只需要從倉庫上 `pull` 下來就可以了。

*註：Docker 倉庫的概念跟 Git 類似，註冊伺服器可以理解爲 GitHub 這樣的託管服務。

安裝

官方網站上有各種環境下的 安裝指南，這裡主要介紹下 Ubuntu 和 CentOS 系列的安裝。

Ubuntu 系列安裝 Docker

透過系統內建套件安裝

Ubuntu 14.04 版本套件庫中已經內建了 Docker 套件，可以直接安裝。

```
$ sudo apt-get update
$ sudo apt-get install -y docker.io
$ sudo ln -sf /usr/bin/docker.io /usr/local/bin/docker
$ sudo sed -i '$acomplete -F _docker docker' /etc/bash_completio
n.d/docker
```

如果使用作業系統內建套件安裝 Docker，目前安裝的版本是比較舊的 0.9.1。 要安裝更新的版本，可以透過更新 Docker 套件庫的方式進行安裝。

透過Docker 套件庫安裝最新版本

要安裝最新的 Docker 版本，首先需要安裝 apt-transport-https 支援，之後透過新增套件庫來安裝。

```
$ sudo apt-get install apt-transport-https
$ sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --r
ecv-keys 36A1D7869245C8950F966E92D8576A8BA88D21E9
$ sudo bash -c "echo deb https://get.docker.io/ubuntu docker mai
n > /etc/apt/sources.list.d/docker.list"
$ sudo apt-get update
$ sudo apt-get install -y lxc-docker
```

快速安裝法(建議使用)

```
$ curl -sSL https://get.docker.com/ubuntu/ | sudo sh
```

14.04 之前版本

如果是較舊版本的 Ubuntu 系統，需要先更新核心。

```
$ sudo apt-get update
$ sudo apt-get install linux-image-generic-lts-raring linux-head
ers-generic-lts-raring
$ sudo reboot
```

然後重複上面的步驟即可。

安裝之後啓動 Docker 服務。

```
$ sudo service docker start
```

CentOS 系列安裝 Docker

Docker 支援 CentOS6 及以後的版本。

CentOS6

對於 CentOS6，可以使用 EPEL 套件庫安裝 Docker，命令以下

```
$ sudo yum install http://mirrors.yun-idc.com/epel/6/i386/epel-release-6-8.noarch.rpm
$ sudo yum install docker-io
```

CentOS7

CentOS7 系統 `CentOS-Extras` 庫中已內建 Docker，可以直接安裝：

```
$ sudo yum install docker
```

安裝之後啓動 Docker 服務，並讓它隨系統啓動自動載入。

```
$ sudo service docker start
$ sudo chkconfig docker on
```

Docker 映像檔

在之前的介紹中，我們知道映像檔是 Docker 的三大組件之一。

Docker 在執行容器前需要本地端存在對應的映像檔，如果映像檔不存在本地端，Docker 會從映像檔倉庫下載（預設是 Docker Hub 公共註冊伺服器中的倉庫）。

本章將介紹更多關於映像檔的內容，包括：

- 從倉庫取得映像檔；
- 管理本地主機上的映像檔；
- 介紹映像檔實作的基本原理

取得映像檔

可以使用 `docker pull` 命令從倉庫取得所需要的映像檔。

下面的例子將從 Docker Hub 倉庫下載一個 Ubuntu 12.04 作業系統的映像檔。

```
$ sudo docker pull ubuntu:12.04
Pulling repository ubuntu
ab8e2728644c: Pulling dependent layers
511136ea3c5a: Download complete
5f0ffaa9455e: Download complete
a300658979be: Download complete
904483ae0c30: Download complete
ffdaafd1ca50: Download complete
d047ae21eeaf: Download complete
```

下載過程中，會輸出取得映像檔的每一層訊息。

該命令實際上相當於 `$ sudo docker pull registry.hub.docker.com/ubuntu:12.04` 命令，即從註冊服務器 `registry.hub.docker.com` 中的 `ubuntu` 倉庫來下載標記爲 `12.04` 的映像檔。

有時候官方倉庫註冊服務器下載較慢，可以從其他倉庫下載。 從其它倉庫下載時需要指定完整的倉庫伺服器位址。例如

```
$ sudo docker pull dl.dockerpool.com:5000/ubuntu:12.04
Pulling dl.dockerpool.com:5000/ubuntu
ab8e2728644c: Pulling dependent layers
511136ea3c5a: Download complete
5f0ffaa9455e: Download complete
a300658979be: Download complete
904483ae0c30: Download complete
ffdaafd1ca50: Download complete
d047ae21eeaf: Download complete
```

完成後，即可隨時使用該映像檔了，例如建立一個容器，讓其中執行 bash。

```
$ sudo docker run -t -i ubuntu:12.04 /bin/bash
root@fe7fc4bd8fc9:/#
```

列出本機映像檔

使用 `docker images` 顯示本機已有的映像檔。

```
$ sudo docker images
REPOSITORY       TAG      IMAGE ID      CREATED      VIRTUAL SIZ
E
ubuntu           12.04    74fe38d11401  4 weeks ago  209.6 MB
ubuntu           precise  74fe38d11401  4 weeks ago  209.6 MB
ubuntu           14.04    99ec81b80c55  4 weeks ago  266 MB
ubuntu           latest   99ec81b80c55  4 weeks ago  266 MB
ubuntu           trusty   99ec81b80c55  4 weeks ago  266 MB
...
```

在列出訊息中，可以看到幾段文字訊息

- 來自於哪個倉庫，比如 ubuntu
- 映像檔的標記，比如 14.04
- 它的 `ID` 號（唯一）
- 建立時間
- 映像檔大小

其中映像檔的 `ID` 唯一標識了映像檔，注意到 `ubuntu:14.04` 和 `ubuntu:trusty` 具有相同的映像檔 `ID`，說明它們實際上是同一映像檔。

`TAG` 用來標記來自同一個倉庫的不同映像檔。例如 `ubuntu` 倉庫中有多個映像檔，通過 `TAG` 來區分發行版本，例如 `10.04`、`12.04`、`12.10`、`13.04`、`14.04` 等。例如下面的命令指定使用映像檔 `ubuntu:14.04` 來啓動一個容器。

```
$ sudo docker run -t -i ubuntu:14.04 /bin/bash
```

如果沒有指定 `TAG`，預設使用 `latest`

建立映像檔

建立映像檔有很多方法，使用者可以從 Docker Hub 取得已有映像檔並更新，也可以在本機建立一個。

修改已有映像檔

先使用下載的映像檔啓動容器。

```
$ sudo docker run -t -i training/sinatra /bin/bash
root@0b2616b0e5a8:/#
```

注意：記住容器的 ID，稍後還會用到。

在容器中加入 json 的 gem 套件。

```
root@0b2616b0e5a8:/# gem install json
```

當結束後，我們使用 exit 來退出，現在我們的容器已經被改變了，使用 `docker commit` 命令來提交更新後的副本。

```
$ sudo docker commit -m "Added json gem" -a "Docker Newbee" 0b26
16b0e5a8 ouruser/sinatra:v2
4f177bd27a9ff0f6dc2a830403925b5360bfe0b93d476f7fc3231110e7f71b1c
```

其中，`-m` 指定提交的說明信息，跟我們使用的版本控制工具一樣；`-a` 可以指定更新的使用者信息；之後是用來建立映像檔的容器的 ID；最後指定新映像檔的名稱和 tag 。建立成功後會印出新映像檔的 ID。

使用 `docker images` 查看新建立的映像檔。

```
$ sudo docker images
REPOSITORY          TAG     IMAGE ID       CREATED       VIRTUAL
 SIZE
training/sinatra    latest  5bc342fa0b91   10 hours ago  446.7 M
B
ouruser/sinatra     v2      3c59e02ddd1a   10 hours ago  446.7 M
B
ouruser/sinatra     latest  5db5f8471261   10 hours ago  446.7 M
B
```

之後，可以使用新的映像檔來啓動容器

```
$ sudo docker run -t -i ouruser/sinatra:v2 /bin/bash
root@78e82f680994:/#
```

利用 **Dockerfile** 建立映像檔

使用 `docker commit` 擴展一個映像檔比較簡單，但是不方便在一個團隊中分享。我們可以使用 `docker build` 來建立一個新的映像檔。爲此，首先需要建立一個 Dockerfile，裡面包含一些用來建立映像檔的指令。

新建一個目錄和一個 Dockerfile

```
$ mkdir sinatra
$ cd sinatra
$ touch Dockerfile
```

Dockerfile 中每一條指令都會建立一層映像檔，例如：

```
# This is a comment
FROM ubuntu:14.04
MAINTAINER Docker Newbee <newbee@docker.com>
RUN apt-get -qq update
RUN apt-get -qqy install ruby ruby-dev
RUN gem install sinatra
```

Dockerfile 基本的語法是

- 使用 `#` 來註釋
- `FROM` 指令告訴 Docker 使用哪個映像檔作爲基底
- 接著是維護者的信息
- `RUN` 開頭的指令會在建立中執行，比如安裝一個套件，在這裏使用 apt-get 來安裝了一些套件

完成 Dockerfile 後可以使用 `docker build` 建立映像檔。

```
$ sudo docker build -t="ouruser/sinatra:v2" .
Uploading context  2.56 kB
Uploading context
Step 0 : FROM ubuntu:14.04
 ---> 99ec81b80c55
Step 1 : MAINTAINER Kate Smith <ksmith@example.com>
 ---> Running in 7c5664a8a0c1
 ---> 2fa8ca4e2a13
Removing intermediate container 7c5664a8a0c1
Step 2 : RUN apt-get -qq update
 ---> Running in b07cc3fb4256
 ---> 50d21070ec0c
Removing intermediate container b07cc3fb4256
Step 3 : RUN apt-get -qqy install ruby ruby-dev
 ---> Running in a5b038dd127e
Selecting previously unselected package libasan0:amd64.
(Reading database ... 11518 files and directories currently inst
alled.)
Preparing to unpack .../libasan0_4.8.2-19ubuntu1_amd64.deb ...
Setting up ruby (1:1.9.3.4) ...
Setting up ruby1.9.1 (1.9.3.484-2ubuntu1) ...
Processing triggers for libc-bin (2.19-0ubuntu6) ...
 ---> 2acb20f17878
Removing intermediate container a5b038dd127e
Step 4 : RUN gem install sinatra
 ---> Running in 5e9d0065c1f7
. . .
Successfully installed rack-protection-1.5.3
Successfully installed sinatra-1.4.5
4 gems installed
 ---> 324104cde6ad
Removing intermediate container 5e9d0065c1f7
Successfully built 324104cde6ad
```

其中 `-t` 標記添加 tag，指定新的映像檔的使用者信息。“.” 是 Dockerfile 所在的路徑（當前目錄），也可以換成具體的 Dockerfile 的路徑。

可以看到 build 指令後執行的操作。它要做的第一件事情就是上傳這個 Dockerfile 內容，因爲所有的操作都要依據 Dockerfile 來進行。 然後，Dockfile 中的指令被一條一條的執行。每一步都建立了一個新的容器，在容器中執行指令並提交修改（就跟之前介紹過的 `docker commit` 一樣）。當所有的指令都執行完畢之後，返回了最終的映像檔 id。所有的中間步驟所產生的容器都會被刪除和清理。

*注意一個映像檔不能超過 127 層

此外，還可以利用 `ADD` 命令複製本地檔案到映像檔；用 `EXPOSE` 命令向外部開放埠號；用 `CMD` 命令描述容器啓動後執行的程序等。例如

```
# put my local web site in myApp folder to /var/www
ADD myApp /var/www
# expose httpd port
EXPOSE 80
# the command to run
CMD ["/usr/sbin/apachectl", "-D", "FOREGROUND"]
```

現在可以利用新建立的映像檔啓動一個容器。

```
$ sudo docker run -t -i ouruser/sinatra:v2 /bin/bash
root@8196968dac35:/#
```

還可以用 `docker tag` 命令修改映像檔的標籤。

```
$ sudo docker tag 5db5f8471261 ouruser/sinatra:devel
$ sudo docker images ouruser/sinatra
REPOSITORY          TAG     IMAGE ID      CREATED        VIRTUAL
 SIZE
ouruser/sinatra     latest  5db5f8471261  11 hours ago   446.7 M
B
ouruser/sinatra     devel   5db5f8471261  11 hours ago   446.7 M
B
ouruser/sinatra     v2      5db5f8471261  11 hours ago   446.7 M
B
```

*註：更多用法，請參考 Dockerfile 章節。

從本機匯入

要從本機匯入一個映像檔，可以使用 OpenVZ（容器虛擬化的先鋒技術）的模板來建立： OpenVZ 的模板下載位址爲 http://openvz.org/Download/templates/precreated。

比如，先下載一個 ubuntu-14.04 的映像檔，之後使用以下命令匯入：

```
sudo cat ubuntu-14.04-x86_64-minimal.tar.gz  |docker import - ub
untu:14.04
```

然後查看新匯入的映像檔。

```
docker images
REPOSITORY          TAG                 IMAGE ID            CREA
TED             VIRTUAL SIZE
ubuntu              14.04               05ac7c0b9383        17 s
econds ago      215.5 MB
```

上傳映像檔

使用者可以通過 `docker push` 命令，把自己建立的映像檔上傳到倉庫中來共享。例如，使用者在 Docker Hub 上完成註冊後，可以推送自己的映像檔到倉庫中。

```
$ sudo docker push ouruser/sinatra

The push refers to a repository [ouruser/sinatra] (len: 1)
Sending image list
Pushing repository ouruser/sinatra (3 tags)
```

儲存和載入映像檔

儲存映像檔

如果要建立映像檔到本地檔案，可以使用 `docker save` 命令。

```
$ sudo docker images
REPOSITORY          TAG                 IMAGE ID            CREA
TED             VIRTUAL SIZE
ubuntu              14.04               c4ff7513909d        5 we
eks ago         225.4 MB
...
$sudo docker save -o ubuntu_14.04.tar ubuntu:14.04
```

載入映像檔

可以使用 `docker load` 從建立的本地檔案中再匯入到本地映像檔庫，例如

```
$ sudo docker load --input ubuntu_14.04.tar
```

或

```
$ sudo docker load < ubuntu_14.04.tar
```

這將匯入映像檔以及其相關的元資料信息（包括標籤等）。

移除本地端映像檔

如果要移除本地端的映像檔，可以使用 `docker rmi` 命令。注意 `docker rm` 命令是移除容器。

```
$ sudo docker rmi training/sinatra
Untagged: training/sinatra:latest
Deleted: 5bc342fa0b91cabf65246837015197eecfa24b2213ed6a51a8974ae250fedd8d
Deleted: ed0fffdcdae5eb2c3a55549857a8be7fc8bc4241fb19ad714364cbfd7a56b22f
Deleted: 5c58979d73ae448df5af1d8142436d81116187a7633082650549c52c3a2418f0
```

*注意：在刪除映像檔之前要先用 `docker rm` 刪掉依賴於這個映像檔的所有容器。

映像檔的實作原理

Docker 映像檔是怎麼實作增量的修改和維護的？ 每個映像檔都由很多層次構成，Docker 使用 Union FS 將這些不同的層結合到一個映像檔中去。

通常 Union FS 有兩個用途, 一方面可以實作不借助 LVM、RAID 將多個 disk 掛到同一個目錄下,另一個更常用的就是將一個唯讀的分支和一個可寫的分支聯合在一起，Live CD 正是基於此方法可以允許在映像檔不變的基礎上允許使用者在其上進行一些寫操作。 Docker 在 AUFS 上建立的容器也是利用了類似的原理。

Docker 容器

容器是 Docker 又一核心概念。

簡單的說，容器是獨立執行的一個或一組應用，以及它們的執行態環境。換句話說，虛擬機可以理解爲模擬執行的一整套作業系統（提供了執行態環境和其他系統環境）和跑在上面的應用。

本章將具體介紹如何來管理一個容器，包括建立、啓動和停止等。

啓動容器

啓動容器有兩種方式，一種是將映像檔新建一個容器並啓動，另外一個是將終止狀態（stopped）的容器重新啓動。

因爲 Docker 的容器實在太輕量級了，使用者可以隨時刪除和新建立容器。

新建並啓動

所需要的命令主要爲 `docker run`。

例如，下面的命令輸出一個 “Hello World”，之後終止容器。

```
$ sudo docker run ubuntu:14.04 /bin/echo 'Hello world'
Hello world
```

這跟在本地直接執行 `/bin/echo 'hello world'` 相同， 幾乎感覺不出任何區別。

下面的命令則啓動一個 bash 終端，允許使用者進行互動。

```
$ sudo docker run -t -i ubuntu:14.04 /bin/bash
root@af8bae53bdd3:/#
```

其中， `-t` 選項讓Docker分配一個虛擬終端（pseudo-tty）並綁定到容器的標準輸入上， `-i` 則讓容器的標準輸入保持打開。

在互動模式下，使用者可以透過所建立的終端來輸入命令，例如

```
root@af8bae53bdd3:/# pwd
/
root@af8bae53bdd3:/# ls
bin boot dev etc home lib lib64 media mnt opt proc root run sbin
 srv sys tmp usr var
```

當利用 `docker run` 來建立容器時，Docker 在後臺執行的標準操作包括：

- 檢查本地是否存在指定的映像檔，不存在就從公有倉庫下載
- 利用映像檔建立並啓動一個容器
- 分配一個檔案系統，並在唯讀的映像檔層外面掛載一層可讀寫層
- 從宿主主機設定的網路橋界面中橋接一個虛擬埠到容器中去
- 從位址池中設定一個 ip 位址給容器
- 執行使用者指定的應用程式
- 執行完畢後容器被終止

啓動已終止容器

可以利用 `docker start` 命令，直接將一個已經終止的容器啓動執行。

容器的核心爲所執行的應用程式，所需要的資源都是應用程式執行所必需的。除此之外，並沒有其它的資源。可以在虛擬終端中利用 `ps` 或 `top` 來查看程式訊息。

```
root@ba267838cc1b:/# ps
  PID TTY          TIME CMD
    1 ?        00:00:00 bash
   11 ?        00:00:00 ps
```

可見，容器中僅執行了指定的 bash 應用。這種特點使得 Docker 對資源的使用率極高，是貨眞價實的輕量級虛擬化。

守護態執行

更多的時候，需要讓 Docker 容器在後臺以守護態（Daemonized）形式執行。此時，可以透過新增 `-d` 參數來實作。

例以下面的命令會在後臺執行容器。

```
$ sudo docker run -d ubuntu:14.04 /bin/sh -c "while true; do ech
o hello world; sleep 1; done"
1e5535038e285177d5214659a068137486f96ee5c2e85a4ac52dc83f2ebe4147
```

容器啓動後會返回一個唯一的 id，也可以透過 `docker ps` 命令來查看容器訊息。

```
$ sudo docker ps
CONTAINER ID  IMAGE         COMMAND               CREATED
 STATUS       PORTS NAMES
1e5535038e28  ubuntu:14.04  /bin/sh -c 'while tr  2 minutes ago
 Up 1 minute        insane_babbage
```

要取得容器的輸出訊息，可以透過 `docker logs` 命令。

```
$ sudo docker logs insane_babbage
hello world
hello world
hello world
. . .
```

終止容器

可以使用 `docker stop` 來終止一個執行中的容器。

此外，當Docker容器中指定的應用終結時，容器也自動終止。 例如對於上一章節中只啓動了一個終端機的容器，使用者透過 `exit` 命令或 `Ctrl+d` 來退出終端時，所建立的容器立刻終止。

終止狀態的容器可以用 `docker ps -a` 命令看到。例如

```
sudo docker ps -a
CONTAINER ID        IMAGE                     COMMAND
    CREATED             STATUS                                PORTS
            NAMES
ba267838cc1b        ubuntu:14.04              "/bin/bash"
    30 minutes ago      Exited (0) About a minute ago
            trusting_newton
98e5efa7d997        training/webapp:latest    "python app.py"
    About an hour ago   Exited (0) 34 minutes ago
            backstabbing_pike
```

處於終止狀態的容器，可以透過 `docker start` 命令來重新啓動。

此外， `docker restart` 命令會將一個執行中的容器終止，然後再重新啓動它。

進入容器

在使用 `-d` 參數時，容器啓動後會進入後臺。 某些時候需要進入容器進行操作，有很多種方法，包括使用 `docker attach` 命令或 `nsenter` 工具等。

exec 命令

`docker exec` 是Docker内建的命令。下面示範如何使用該命令。

```
$ sudo docker run -idt ubuntu
243c32535da7d142fb0e6df616a3c3ada0b8ab417937c853a9e1c251f499f550
$ sudo docker ps
CONTAINER ID        IMAGE               COMMAND             CREA
TED             STATUS              PORTS               NAMES
243c32535da7        ubuntu:latest       "/bin/bash"         18 s
econds ago      Up 17 seconds                           nostalgi
c_hypatia
$sudo docker exec -ti nostalgic_hypatia bash
root@243c32535da7:/#
```

attach 命令

`docker attach` 亦是Docker内建的命令。下面示例如何使用該命令。

```
$ sudo docker run -idt ubuntu
243c32535da7d142fb0e6df616a3c3ada0b8ab417937c853a9e1c251f499f550
$ sudo docker ps
CONTAINER ID        IMAGE               COMMAND             CREA
TED             STATUS              PORTS               NAMES
243c32535da7        ubuntu:latest       "/bin/bash"         18 s
econds ago      Up 17 seconds                           nostalgi
c_hypatia
$sudo docker attach nostalgic_hypatia
root@243c32535da7:/#
```

但是使用 `attach` 命令有時候並不方便。當多個窗口同時 attach 到同一個容器的時候，所有窗口都會同步顯示。當某個窗口因命令阻塞時,其他窗口也無法執行操作了。

nsenter 命令

安装

`nsenter` 工具已含括在 util-linux 2.23 後的版本內。 如果系統中 util-linux 包沒有該命令，可以按照下面的方法從原始碼安裝。

```
$ cd /tmp; curl https://www.kernel.org/pub/linux/utils/util-linu
x/v2.24/util-linux-2.24.tar.gz | tar -zxf-; cd util-linux-2.24;
$ ./configure --without-ncurses
$ make nsenter && sudo cp nsenter /usr/local/bin
```

使用

`nsenter` 可以存取另一個程式的命名空間。nsenter 要正常工作需要有 root 權限。 很不幸，Ubuntu 14.4 仍然使用的是 util-linux 2.20。安裝最新版本的 util-linux（2.24）版，請按照以下步驟：

```
$ wget https://www.kernel.org/pub/linux/utils/util-linux/v2.24/u
til-linux-2.24.tar.gz; tar xzvf util-linux-2.24.tar.gz
$ cd util-linux-2.24
$ ./configure --without-ncurses && make nsenter
$ sudo cp nsenter /usr/local/bin
```

爲了連接到容器，你還需要找到容器的第一個程式的 PID，可以透過下面的命令取得。

```
PID=$(docker inspect --format "{{ .State.Pid }}" <container>)
```

透過這個 PID，就可以連接到這個容器：

```
$ nsenter --target $PID --mount --uts --ipc --net --pid
```

下面給出一個完整的例子。

```
$ sudo docker run -idt ubuntu
243c32535da7d142fb0e6df616a3c3ada0b8ab417937c853a9e1c251f499f550
$ sudo docker ps
CONTAINER ID        IMAGE               COMMAND             CREA
TED             STATUS              PORTS               NAMES
243c32535da7        ubuntu:latest       "/bin/bash"         18 s
econds ago      Up 17 seconds                           nostalgi
c_hypatia
$ PID=$(docker-pid 243c32535da7)
10981
$ sudo nsenter --target 10981 --mount --uts --ipc --net --pid
root@243c32535da7:/#
```

更簡單的，建議大家下載 .bashrc_docker，並將內容放到 .bashrc 中。

```
$ wget -P ~ https://github.com/yeasy/docker_practice/raw/master/
_local/.bashrc_docker;
$ echo "[ -f ~/.bashrc_docker ] && . ~/.bashrc_docker" >> ~/.bas
hrc; source ~/.bashrc
```

這個檔案中定義了很多方便使用 Docker 的命令，例如 `docker-pid` 可以取得某個容器的 PID；而 `docker-enter` 可以進入容器或直接在容器內執行命令。

```
$ echo $(docker-pid <container>)
$ docker-enter <container> ls
```

匯出和匯入容器

匯出容器

如果要匯出本地某個容器，可以使用 `docker export` 命令。

```
$ sudo docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREA
TED             STATUS                      PORTS               NA
MES
7691a814370e        ubuntu:14.04        "/bin/bash"         36 h
ours ago        Exited (0) 21 hours ago                         te
st
$ sudo docker export 7691a814370e > ubuntu.tar
```

這樣將匯出容器快照到本地檔案。

匯入容器快照

可以使用 `docker import` 從容器快照檔案中再匯入爲映像檔，例如

```
$ cat ubuntu.tar | sudo docker import - test/ubuntu:v1.0
$ sudo docker images
REPOSITORY          TAG                 IMAGE ID            CREA
TED             VIRTUAL SIZE
test/ubuntu         v1.0                9d37a6082e97        Abou
t a minute ago   171.3 MB
```

此外，也可以透過指定 URL 或者某個目錄來匯入，例如

```
$sudo docker import http://example.com/exampleimage.tgz example/
imagerepo
```

*註：使用者既可以使用 `docker load` 來匯入映像檔儲存檔案到本地映像檔庫，也可以使用 `docker import` 來匯入一個容器快照到本地映像檔庫。這兩者的區別在於容器快照檔案將丟棄所有的歷史記錄和原始資料訊息（即僅保存容器當時的快照狀態），而映像檔儲存檔案將保存完整記錄，檔案體積也跟著變大。此外，從容器快照檔案匯入時可以重新指定標籤等原始資料訊息。

刪除容器

可以使用 `docker rm` 來刪除一個處於終止狀態的容器。 例如

```
$sudo docker rm  trusting_newton
trusting_newton
```

如果要刪除一個執行中的容器，可以新增 `-f` 參數。Docker 會發送 `SIGKILL` 信號給容器。

倉庫

倉庫（Repository）是集中存放映像檔的地方。

一個容易混淆的概念是註冊伺服器（Registry）。實際上註冊伺服器是管理倉庫的具體伺服器，每個伺服器上可以有多個倉庫，而每個倉庫下面有多個映像檔。從這方面來說，倉庫可以被認爲是一個具體的專案或目錄。例如對於倉庫位址 `dl.dockerpool.com/ubuntu` 來說， `dl.dockerpool.com` 是註冊伺服器位址， `ubuntu` 是倉庫名。

大部分時候，並不需要嚴格區分這兩者的概念。

Docker Hub

目前 Docker 官方維護了一個公共倉庫 Docker Hub，其中已經包括了超過 15,000 的映像檔。大部分需求，都可以透過在 Docker Hub 中直接下載映像檔來實作。

登錄

可以透過執行 `docker login` 命令來輸入使用者名稱、密碼和電子信箱來完成註冊和登錄。 註冊成功後，本地使用者目錄的 `.dockercfg` 中將保存使用者的認證訊息。

基本操作

使用者無需登錄即可透過 `docker search` 命令來查詢官方倉庫中的映像檔，並利用 `docker pull` 命令來將它下載到本地。

例如以 centos 爲關鍵字進行搜索：

```
$ sudo docker search centos
NAME                                              DESCRIPTION
                                 STARS     OFFICIAL   AUTOMATED
centos                                            The official bui
ld of CentOS.                    465       [OK]
tianon/centos                                     CentOS 5 and 6,
created using rinse instea...    28
blalor/centos                                     Bare-bones base
CentOS 6.5 image                 6                      [OK]
saltstack/centos-6-minimal
                                 6                      [OK]
tutum/centos-6.4                                  DEPRECATED. Use
tutum/centos:6.4 instead. ...    5                      [OK]
...
```

可以看到顯示了很多包含關鍵字的映像檔，其中包括映像檔名字、描述、星級（表示該映像檔的受歡迎程度）、是否官方建立、是否自動建立。 官方的映像檔說明是官方專案組建立和維護的，automated 資源允許使用者驗證映像檔的來源和內容。

根據是否是官方提供，可將映像檔資源分爲兩類。一種是類似 centos 這樣的基礎映像檔，被稱爲基礎或根映像檔。這些基礎映像檔是由 Docker 公司建立、驗證、支援、提供。這樣的映像檔往往使用單個單詞作爲名字。還有一種類型，比如 `tianon/centos` 映像檔，它是由 Docker 的使用者建立並維護的，往往帶有使用者名稱前綴。可以透過前綴 `user_name/` 來指定使用某個使用者提供的映像檔，比如 tianon 使用者。

另外，在查詢的時候透過 `-s N` 參數可以指定僅顯示評價爲 `N` 星以上的映像檔。

下載官方 centos 映像檔到本地。

```
$ sudo docker pull centos
Pulling repository centos
0b443ba03958: Download complete
539c0211cd76: Download complete
511136ea3c5a: Download complete
7064731afe90: Download complete
```

使用者也可以在登錄後透過 `docker push` 命令來將映像檔推送到 Docker Hub。

自動建立

自動建立（Automated Builds）功能對於需要經常升級映像檔內程式來說，十分方便。有時候，使用者建立了映像檔，安裝了某個軟體，如果軟體發布新版本則需要手動更新映像檔。。

而自動建立允許使用者透過 Docker Hub 指定跟蹤一個目標網站（目前支援 GitHub 或 BitBucket）上的專案，一旦專案發生新的提交，則自動執行建立。

要設定自動建立，包括以下的步驟：

- 建立並登陸 Docker Hub，以及目標網站；
- 在目標網站中連接帳戶到 Docker Hub；
- 在 Docker Hub 中 設定一個自動建立；
- 選取一個目標網站中的專案（需要含 Dockerfile）和分支；
- 指定 Dockerfile 的位置，並提交建立。

之後，可以 在Docker Hub 的 自動建立頁面 中跟蹤每次建立的狀態。

私有倉庫

有時候使用 Docker Hub 這樣的公共倉庫可能不方便，使用者可以建立一個本地倉庫供私人使用。

本節介紹如何使用本地倉庫。

`docker-registry` 是官方提供的工具，可以用於建立私有的映像檔倉庫。

安裝執行 **docker-registry**

容器執行

在安裝了 Docker 後，可以透過取得官方 registry 映像檔來執行。

```
$ sudo docker run -d -p 5000:5000 registry
```

這將使用官方的 registry 映像檔來啓動本地的私有倉庫。 使用者可以透過指定參數來設定私有倉庫位置，例如設定映像檔儲存到 Amazon S3 服務。

```
$ sudo docker run \
         -e SETTINGS_FLAVOR=s3 \
         -e AWS_BUCKET=acme-docker \
         -e STORAGE_PATH=/registry \
         -e AWS_KEY=AKIAHSHB43HS3J92MXZ \
         -e AWS_SECRET=xdDowwlK7TJajV1Y7EoOZrmuPEJlHYcNP2k4j49T
\
         -e SEARCH_BACKEND=sqlalchemy \
         -p 5000:5000 \
         registry
`
```

此外，還可以指定本地路徑（如 `/home/user/registry-conf` ）下的設定檔案。

```
$ sudo docker run -d -p 5000:5000 -v /home/user/registry-conf:/r
egistry-conf -e DOCKER_REGISTRY_CONFIG=/registry-conf/config.yml
 registry
```

預設情況下，倉庫會被建立在容器的 `/tmp/registry` 下。可以透過 `-v` 參數來將映像檔檔案存放在本地的指定路徑。 例以下面的例子將上傳的映像檔放到 `/opt/data/registry` 目錄。

```
$ sudo docker run -d -p 5000:5000 -v /opt/data/registry:/tmp/reg
istry registry
```

本地安裝

對於 Ubuntu 或 CentOS 等發行版，可以直接透過套件庫安裝。

- Ubuntu

```
$ sudo apt-get install -y build-essential python-dev libeven
t-dev python-pip liblzma-dev swig
$ sudo pip install docker-registry
```

- CentOS

```
$ sudo yum install -y python-devel libevent-devel python-pip
 gcc xz-devel
$ sudo python-pip install docker-registry
```

也可以從 docker-registry 專案下載原始碼進行安裝。

```
$ sudo apt-get install build-essential python-dev libevent-dev p
ython-pip libssl-dev liblzma-dev libffi-dev
$ git clone https://github.com/docker/docker-registry.git
$ cd docker-registry
$ sudo python setup.py install
```

然後修改設定檔案，主要修改 dev 模板段的 `storage_path` 到本地的儲存倉庫的路徑。

```
$ cp config/config_sample.yml config/config.yml
```

之後啓動 Web 服務。

```
$ sudo gunicorn -c contrib/gunicorn.py docker_registry.wsgi:appl
ication
```

或者

```
$ sudo gunicorn --access-logfile - --error-logfile - -k gevent -
b 0.0.0.0:5000 -w 4 --max-requests 100 docker_registry.wsgi:appl
ication
```

此時使用連結本地的 5000 連接埠，看到輸出 docker-registry 的版本訊息說明執行成功。

*註：`config/config_sample.yml` 檔案是範例設定檔案。

在私有倉庫上傳、下載、搜索映像檔

建立好私有倉庫之後，就可以使用 `docker tag` 來標記一個映像檔，然後推送它到倉庫，別的機器上就可以下載下來了。例如私有倉庫位址爲 `192.168.7.26:5000`。

先在本機查看已有的映像檔。

```
$ sudo docker images
REPOSITORY                          TAG                 IMAGE ID
          CREATED             VIRTUAL SIZE
ubuntu                              latest              ba5877dc9b
ec        6 weeks ago         192.7 MB
ubuntu                              14.04               ba5877dc9b
ec        6 weeks ago         192.7 MB
```

使用 `docker tag` 將 `ba58` 這個映像檔標記爲 `192.168.7.26:5000/test`（格式爲 `docker tag IMAGE[:TAG] [REGISTRYHOST/][USERNAME/]NAME[:TAG]`）。

```
$ sudo docker tag ba58 192.168.7.26:5000/test
root ~ # docker images
REPOSITORY                          TAG                 IMAGE ID
          CREATED             VIRTUAL SIZE
ubuntu                              14.04               ba5877dc9b
ec        6 weeks ago         192.7 MB
ubuntu                              latest              ba5877dc9b
ec        6 weeks ago         192.7 MB
192.168.7.26:5000/test              latest              ba5877dc9b
ec        6 weeks ago         192.7 MB
```

使用 `docker push` 上傳標記的映像檔。

```
$ sudo docker push 192.168.7.26:5000/test
The push refers to a repository [192.168.7.26:5000/test] (len: 1
)
Sending image list
Pushing repository 192.168.7.26:5000/test (1 tags)
Image 511136ea3c5a already pushed, skipping
Image 9bad880da3d2 already pushed, skipping
Image 25f11f5fb0cb already pushed, skipping
Image ebc34468f71d already pushed, skipping
Image 2318d26665ef already pushed, skipping
Image ba5877dc9bec already pushed, skipping
Pushing tag for rev [ba5877dc9bec] on {http://192.168.7.26:5000/
v1/repositories/test/tags/latest}
```

用 curl 查看倉庫中的映像檔。

```
$ curl http://192.168.7.26:5000/v1/search
{"num_results": 7, "query": "", "results": [{"description": "",
"name": "library/miaxis_j2ee"}, {"description": "", "name": "lib
rary/tomcat"}, {"description": "", "name": "library/ubuntu"}, {"
description": "", "name": "library/ubuntu_office"}, {"descriptio
n": "", "name": "library/desktop_ubu"}, {"description": "", "nam
e": "dockerfile/ubuntu"}, {"description": "", "name": "library/t
est"}]}
```

這裡可以看到 `{"description": "", "name": "library/test"}` ，表明映像檔已經被成功上傳了。

現在可以到另外一臺機器去下載這個映像檔。

```
$ sudo docker pull 192.168.7.26:5000/test
Pulling repository 192.168.7.26:5000/test
ba5877dc9bec: Download complete
511136ea3c5a: Download complete
9bad880da3d2: Download complete
25f11f5fb0cb: Download complete
ebc34468f71d: Download complete
2318d26665ef: Download complete
$ sudo docker images
REPOSITORY                          TAG                 IMAGE ID
          CREATED             VIRTUAL SIZE
192.168.7.26:5000/test              latest              ba5877dc9
bec       6 weeks ago         192.7 MB
```

可以使用 這個腳本 批次上傳本地的映像檔到註冊伺服器中，預設爲本地註冊伺服器 `127.0.0.1:5000` 。例如：

```
$ wget https://github.com/yeasy/docker_practice/raw/master/_local/push_images.sh; sudo chmod a+x push_images.sh
$ ./push_images.sh ubuntu:latest centos:centos7
The registry server is 127.0.0.1
Uploading ubuntu:latest...
The push refers to a repository [127.0.0.1:5000/ubuntu] (len: 1)
Sending image list
Pushing repository 127.0.0.1:5000/ubuntu (1 tags)
Image 511136ea3c5a already pushed, skipping
Image bfb8b5a2ad34 already pushed, skipping
Image c1f3bdbd8355 already pushed, skipping
Image 897578f527ae already pushed, skipping
Image 9387bcc9826e already pushed, skipping
Image 809ed259f845 already pushed, skipping
Image 96864a7d2df3 already pushed, skipping
Pushing tag for rev [96864a7d2df3] on {http://127.0.0.1:5000/v1/repositories/ubuntu/tags/latest}
Untagged: 127.0.0.1:5000/ubuntu:latest
Done
Uploading centos:centos7...
The push refers to a repository [127.0.0.1:5000/centos] (len: 1)
Sending image list
Pushing repository 127.0.0.1:5000/centos (1 tags)
Image 511136ea3c5a already pushed, skipping
34e94e67e63a: Image successfully pushed
70214e5d0a90: Image successfully pushed
Pushing tag for rev [70214e5d0a90] on {http://127.0.0.1:5000/v1/repositories/centos/tags/centos7}
Untagged: 127.0.0.1:5000/centos:centos7
Done
```

倉庫設定檔案

Docker 的 Registry 利用設定檔案提供了一些倉庫的模組（flavor），使用者可以直接使用它們來進行開發或生產部署。

模組

在 `config_sample.yml` 檔案中，可以看到一些現成的模組段：

- `common`：基礎設定
- `local`：儲存資料到本地檔案系統
- `s3`：儲存資料到 AWS S3 中
- `dev`：使用 `local` 模組的基本設定
- `test`：單元測試使用
- `prod`：生產環境設定（基本上跟s3設定類似）
- `gcs`：儲存資料到 Google 的雲端
- `swift`：儲存資料到 OpenStack Swift 服務
- `glance`：儲存資料到 OpenStack Glance 服務，本地檔案系統爲後備
- `glance-swift`：儲存資料到 OpenStack Glance 服務，Swift 爲後備
- `elliptics`：儲存資料到 Elliptics key/value 儲存

使用者也可以新增自定義的模版段。

預設情況下使用的模組是 `dev`，要使用某個模組作爲預設值，可以新增 `SETTINGS_FLAVOR` 到環境變數中，例如

```
export SETTINGS_FLAVOR=dev
```

另外，設定檔案中支援從環境變數中載入值，語法格式爲 `_env:VARIABLENAME[:DEFAULT]`。

範例設定

```
common:
    loglevel: info
    search_backend: "_env:SEARCH_BACKEND:"
    sqlalchemy_index_database:
        "_env:SQLALCHEMY_INDEX_DATABASE:sqlite:////tmp/docker-re
gistry.db"

prod:
    loglevel: warn
    storage: s3
    s3_access_key: _env:AWS_S3_ACCESS_KEY
    s3_secret_key: _env:AWS_S3_SECRET_KEY
    s3_bucket: _env:AWS_S3_BUCKET
    boto_bucket: _env:AWS_S3_BUCKET
    storage_path: /srv/docker
    smtp_host: localhost
    from_addr: docker@myself.com
    to_addr: my@myself.com

dev:
    loglevel: debug
    storage: local
    storage_path: /home/myself/docker

test:
    storage: local
    storage_path: /tmp/tmpdockertmp
```

選項

Docker 資料管理

這一章介紹如何在 Docker 內部以及容器之間管理資料，在容器中管理資料主要有兩種方式：

- 資料卷（Data volumes）
- 資料卷容器（Data volume containers）

資料卷

資料卷是一個可供一個或多個容器使用的特殊目錄，它繞過 UFS，可以提供很多有用的特性：

- 資料卷可以在容器之間共享和重用
- 對資料卷的修改會立馬生效
- 對資料卷的更新，不會影響映像檔
- 卷會一直存在，直到沒有容器使用

*資料卷的使用，類似於 Linux 下對目錄或檔案進行 mount。

建立一個資料卷

在用 `docker run` 命令的時候，使用 `-v` 標記來建立一個資料卷並掛載到容器裡。在一次 run 中多次使用可以掛載多個資料卷。

下面建立一個 web 容器，並載入一個資料卷到容器的 `/webapp` 目錄。

```
$ sudo docker run -d -P --name web -v /webapp training/webapp py
thon app.py
```

*注意：也可以在 Dockerfile 中使用 `VOLUME` 來新增一個或者多個新的卷到由該映像檔建立的任意容器。

掛載一個主機目錄作爲資料卷

使用 `-v` 標記也可以指定掛載一個本地主機的目錄到容器中去。

```
$ sudo docker run -d -P --name web -v /src/webapp:/opt/webapp tr
aining/webapp python app.py
```

上面的命令載入主機的 `/src/webapp` 目錄到容器的 `/opt/webapp` 目錄。這個功能在進行測試的時候十分方便，比如使用者可以放置一些程式到本地目錄中，來查看容器是否正常工作。本地目錄的路徑必須是絕對路徑，如果目錄不存在

Docker 會自動爲你建立它。

*注意：Dockerfile 中不支援這種用法，這是因爲 Dockerfile 是爲了移植和分享用的。然而，不同作業系統的路徑格式不一樣，所以目前還不能支援。

Docker 掛載資料卷的預設權限是讀寫，使用者也可以透過 `:ro` 指定爲唯讀。

```
$ sudo docker run -d -P --name web -v /src/webapp:/opt/webapp:ro
training/webapp python app.py
```

加了 `:ro` 之後，就掛載爲唯讀了。

掛載一個本地主機檔案作爲資料卷

`-v` 標記也可以從主機掛載單個檔案到容器中

```
$ sudo docker run --rm -it -v ~/.bash_history:/.bash_history ubu
ntu /bin/bash
```

這樣就可以記錄在容器輸入過的命令了。

*注意：如果直接掛載一個檔案，很多檔案編輯工具，包括 `vi` 或者 `sed --in-place` ，可能會造成檔案 inode 的改變，從 Docker 1.1 .0起，這會導致報錯誤訊息。所以最簡單的辦法就直接掛載檔案的父目錄。

資料卷容器

如果你有一些持續更新的資料需要在容器之間共享，最好建立資料卷容器。

資料卷容器，其實就是一個正常的容器，專門用來提供資料卷供其它容器掛載的。

首先，建立一個命名的資料卷容器 dbdata：

```
$ sudo docker run -d -v /dbdata --name dbdata training/postgres
echo Data-only container for postgres
```

然後，在其他容器中使用 `--volumes-from` 來掛載 dbdata 容器中的資料卷。

```
$ sudo docker run -d --volumes-from dbdata --name db1 training/p
ostgres
$ sudo docker run -d --volumes-from dbdata --name db2 training/p
ostgres
```

還可以使用多個 `--volumes-from` 參數來從多個容器掛載多個資料卷。 也可以從其他已經掛載了容器卷的容器來掛載資料卷。

```
$ sudo docker run -d --name db3 --volumes-from db1 training/post
gres
```

*注意：使用 `--volumes-from` 參數所掛載資料卷的容器自己並不需要保持在執行狀態。

如果刪除了掛載的容器（包括 dbdata、db1 和 db2），資料卷並不會被自動刪除。 如果要刪除一個資料卷，必須在刪除最後一個還掛載著它的容器時使用 `docker rm -v` 命令來指定同時刪除關聯的容器。 這可以讓使用者在容器之間升級和移動資料卷。具體的操作將在下一節中進行講解。

利用資料卷容器來備份、恢復、遷移資料卷

可以利用資料卷對其中的資料進行進行備份、恢復和遷移。

備份

首先使用 `--volumes-from` 標記來建立一個載入 dbdata 容器卷的容器，並從本地主機掛載當前到容器的 /backup 目錄。命令以下：

```
$ sudo docker run --volumes-from dbdata -v $(pwd):/backup ubuntu
 tar cvf /backup/backup.tar /dbdata
```

容器啓動後，使用了 `tar` 命令來將 dbdata 卷備份爲本地的 `/backup/backup.tar` 。

恢復

如果要恢復資料到一個容器，首先建立一個帶有資料卷的容器 dbdata2。

```
$ sudo docker run -v /dbdata --name dbdata2 ubuntu /bin/bash
```

然後建立另一個容器，掛載 dbdata2 的容器，並使用 `untar` 解壓備份檔案到掛載的容器卷中。

```
$ sudo docker run --volumes-from dbdata2 -v $(pwd):/backup busyb
ox tar xvf
/backup/backup.tar
```

Docker 中的網路功能介紹

Docker 允許透過外部存取容器或容器互聯的方式來提供網路服務。

外部存取容器

容器中可以執行一些網路應用，要讓外部也可以存取這些應用，可以通過 `-P` 或 `-p` 參數來指定連接埠映射。

當使用 -P 參數時，Docker 會隨機映射一個 `49000~49900` 的連接埠到內部容器開放的網路連接埠。

使用 `docker ps` 可以看到，本地主機的 49155 被映射到了容器的 5000 連接埠。此時連結本機的 49155 連接埠即可連結容器內 web 應用提供的界面。

```
$ sudo docker run -d -P training/webapp python app.py
$ sudo docker ps -l
CONTAINER ID  IMAGE                     COMMAND       CREATED
   STATUS        PORTS                     NAMES
bc533791f3f5  training/webapp:latest  python app.py 5 seconds ag
o  Up 2 seconds  0.0.0.0:49155->5000/tcp  nostalgic_morse
```

同樣的，可以透過 `docker logs` 命令來查看應用的訊息。

```
$ sudo docker logs -f nostalgic_morse
* Running on http://0.0.0.0:5000/
10.0.2.2 - - [23/May/2014 20:16:31] "GET / HTTP/1.1" 200 -
10.0.2.2 - - [23/May/2014 20:16:31] "GET /favicon.ico HTTP/1.1"
404 -
```

-p（小寫的）則可以指定要映射的連接埠，並且在一個指定連接埠上只可以綁定一個容器。支援的格式有 `ip:hostPort:containerPort | ip::containerPort | hostPort:containerPort` 。

映射所有遠端位址

使用 `hostPort:containerPort` 格式本地的 5000 連接埠映射到容器的 5000 連接埠，可以執行

```
$ sudo docker run -d -p 5000:5000 training/webapp python app.py
```

此時預設會綁定本地所有遠端上的所有位址。

映射到指定位址的指定連接埠

可以使用 `ip:hostPort:containerPort` 格式指定映射使用一個特定位址，比如 localhost 位址 127.0.0.1

```
$ sudo docker run -d -p 127.0.0.1:5000:5000 training/webapp pyth
on app.py
```

映射到指定位址的任意連接埠

使用 `ip::containerPort` 綁定 localhost 的任意連接埠到容器的 5000 連接埠，本地主機會自動分配一個連接埠。

```
$ sudo docker run -d -p 127.0.0.1::5000 training/webapp python a
pp.py
```

還可以使用 udp 標記來指定 udp 連接埠

```
$ sudo docker run -d -p 127.0.0.1:5000:5000/udp training/webapp
python app.py
```

查看映射連接埠配置

使用 `docker port` 來查看當前映射的連接埠配置，也可以查看到綁定的位址

```
$ docker port nostalgic_morse 5000
127.0.0.1:49155.
```

注意：

- 容器有自己的內部網路和 ip 位址（使用 `docker inspect` 可以獲取所有的變數，Docker 還可以有一個可變的網路設定。）
- -p 標記可以多次使用來綁定多個連接埠

例如

```
$ sudo docker run -d -p 5000:5000  -p 3000:80 training/webapp py
thon app.py
```

容器互聯

容器的連接（linking）系統是除了連接埠映射外，另一種跟容器中應用互動的方式。

該系統會在來源端容器和接收端容器之間創建一個隧道，接收端容器可以看到來源端容器指定的信息。

自定義容器命名

連接系統依據容器的名稱來執行。因此，首先需要自定義一個好記的容器命名。

雖然當創建容器的時候，系統會預設分配一個名字。自定義命名容器有2個好處：

- 自定義的命名，比較好記，比如一個web應用容器我們可以給它起名叫web
- 當要連接其他容器時候，可以作爲一個有用的參考點，比如連接web容器到db容器

使用 `--name` 標記可以爲容器自定義命名。

```
$ sudo docker run -d -P --name web training/webapp python app.py
```

使用 `docker ps` 來驗證設定的命名。

```
$ sudo docker ps -l
CONTAINER ID  IMAGE                     COMMAND        CREATED
  STATUS       PORTS                        NAMES
aed84ee21bde  training/webapp:latest python app.py  12 hours ago
  Up 2 seconds 0.0.0.0:49154->5000/tcp  web
```

也可以使用 `docker inspect` 來查看容器的名字

```
$ sudo docker inspect -f "{{ .Name }}" aed84ee21bde
/web
```

注意：容器的名稱是唯一的。如果已經命名了一個叫 web 的容器，當你要再次使用 web 這個名稱的時候，需要先用 `docker rm` 來刪除之前建立的同名容器。

在執行 `docker run` 的時候如果新增 `--rm` 標記，則容器在終止後會立刻刪除。注意，`--rm` 和 `-d` 參數不能同時使用。

容器互聯

使用 `--link` 參數可以讓容器之間安全的進行互動。

下面先建立一個新的資料庫容器。

```
$ sudo docker run -d --name db training/postgres
```

刪除之前建立的 web 容器

```
$ docker rm -f web
```

然後建立一個新的 web 容器，並將它連接到 db 容器

```
$ sudo docker run -d -P --name web --link db:db training/webapp
python app.py
```

此時，db 容器和 web 容器建立互聯關系。

`--link` 參數的格式爲 `--link name:alias`，其中 `name` 是要連接的容器名稱，`alias` 是這個連接的別名。

使用 `docker ps` 來查看容器的連接

```
$ docker ps
CONTAINER ID  IMAGE                          COMMAND                CR
EATED             STATUS              PORTS                      NA
MES
349169744e49  training/postgres:latest  su postgres -c '/usr  Ab
out a minute ago  Up About a minute  5432/tcp                   db
, web/db
aed84ee21bde  training/webapp:latest    python app.py          16
 hours ago        Up 2 minutes       0.0.0.0:49154->5000/tcp  we
b
```

可以看到自定義命名的容器，db 和 web，db 容器的 names 列有 db 也有 web/db。這表示 web 容器連接到 db 容器，web 容器將被允許存取 db 容器的訊息。

Docker 在兩個互聯的容器之間創建了一個安全隧道，而且不用映射它們的連接埠到宿主主機上。在啓動 db 容器的時候並沒有使用 `-p` 和 `-P` 標記，從而避免了暴露資料庫連接埠到外部網路上。

Docker 透過 2 種方式爲容器公開連接訊息：

- 環境變數
- 更新 `/etc/hosts` 檔案

使用 `env` 命令來查看 web 容器的環境變數

```
$ sudo docker run --rm --name web2 --link db:db training/webapp
env
. . .
DB_NAME=/web2/db
DB_PORT=tcp://172.17.0.5:5432
DB_PORT_5000_TCP=tcp://172.17.0.5:5432
DB_PORT_5000_TCP_PROTO=tcp
DB_PORT_5000_TCP_PORT=5432
DB_PORT_5000_TCP_ADDR=172.17.0.5
. . .
```

其中 DB_ 開頭的環境變數是供 web 容器連接 db 容器使用，前綴採用大寫的連接別名。

除了環境變量，Docker 還新增 host 訊息到父容器的 `/etc/hosts` 的檔案。下面是父容器 web 的 hosts 檔案

```
$ sudo docker run -t -i --rm --link db:db training/webapp /bin/b
ash
root@aed84ee21bde:/opt/webapp# cat /etc/hosts
172.17.0.7  aed84ee21bde
. . .
172.17.0.5  db
```

這裡有 2 個 hosts，第一個是 web 容器，web 容器用 id 作爲他的主機名，第二個是 db 容器的 ip 和主機名。 可以在 web 容器中安裝 ping 命令來測試跟db容器的連通。

```
root@aed84ee21bde:/opt/webapp# apt-get install -yqq inetutils-pi
ng
root@aed84ee21bde:/opt/webapp# ping db
PING db (172.17.0.5): 48 data bytes
56 bytes from 172.17.0.5: icmp_seq=0 ttl=64 time=0.267 ms
56 bytes from 172.17.0.5: icmp_seq=1 ttl=64 time=0.250 ms
56 bytes from 172.17.0.5: icmp_seq=2 ttl=64 time=0.256 ms
```

用 ping 來測試db容器，它會解析成 `172.17.0.5` 。*注意：官方的 ubuntu 映像檔預設沒有安裝 ping，需要自行安裝。

使用者可以連接多個子容器到父容器，比如可以連接多個 web 到 db 容器上。

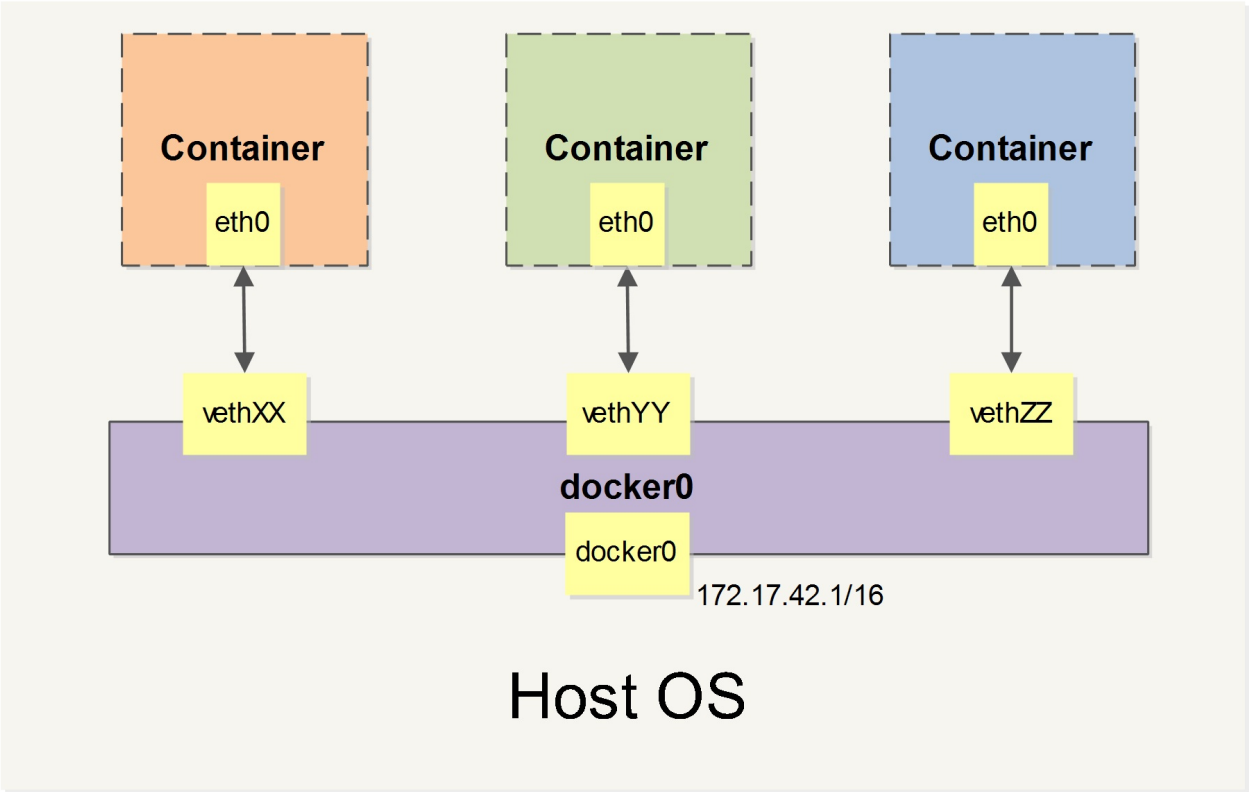
進階網路設定

本章將介紹 Docker 的一些進階網路設定和選項。

當 Docker 啓動時，會自動在主機上建立一個 `docker0` 虛擬橋接器，實際上是 Linux 的一個 bridge，可以理解爲一個軟體交換機。它會在掛載到它的網卡之間進行轉發。

同時，Docker 隨機分配一個本地未占用的私有網段（在 RFC1918 中定義）中的一個位址給 `docker0` 界面。比如典型的 `172.17.42.1`，網路遮罩爲 `255.255.0.0`。此後啓動的容器內的網卡也會自動分配一個同一網段（`172.17.0.0/16`）的網址。

當建立一個 Docker 容器的時候，同時會建立了一對 `veth pair` 界面（當資料包發送到一個界面時，另外一個界面也可以收到相同的資料包）。這對界面一端在容器內，即 `eth0`；另一端在本地並被掛載到 `docker0` 橋接器，名稱以 `veth` 開頭（例如 `vethAQI2QT`）。透過這種方式，主機可以跟容器通信，容器之間也可以相互通信。Docker 就建立了在主機和所有容器之間一個虛擬共享網路。



接下來的部分將介紹在一些場景中，Docker 所有的網路自訂設定。以及透過 Linux 命令來調整、補充、甚至替換 Docker 預設的網路設定。

快速設定指南

下面是一個跟 Docker 網路相關的命令列表。

其中有些命令選項只有在 Docker 服務啓動的時候才能設定，而且不能馬上生效。

- `-b BRIDGE or --bridge=BRIDGE` --指定容器掛載的橋接器
- `--bip=CIDR` --定制 docker0 的遮罩
- `-H SOCKET... or --host=SOCKET...` --Docker 服務端接收命令的通道
- `--icc=true|false` --是否支援容器之間進行通信
- `--ip-forward=true|false` --請看下文容器之間的通信
- `--iptables=true|false` --禁止 Docker 新增 iptables 規則
- `--mtu=BYTES` --容器網路中的 MTU

下面2個命令選項既可以在啓動服務時指定，也可以 Docker 容器啓動（ `docker run` ）時候指定。在 Docker 服務啓動的時候指定則會成爲預設值，後面執行 `docker run` 時可以覆蓋設定的預設值。

- `--dns=IP_ADDRESS...` --使用指定的DNS伺服器
- `--dns-search=DOMAIN...` --指定DNS搜索域

最後這些選項只有在 `docker run` 執行時使用，因爲它是針對容器的特性內容。

- `-h HOSTNAME or --hostname=HOSTNAME` --設定容器主機名
- `--link=CONTAINER_NAME:ALIAS` --新增到另一個容器的連接
- `--net=bridge|none|container:NAME_or_ID|host` --設定容器的橋接模式
- `-p SPEC or --publish=SPEC` --映射容器連接埠到宿主主機
- `-P or --publish-all=true|false` --映射容器所有連接埠到宿主主機

設定 DNS

Docker 沒有爲每個容器專門定制映像檔，那麼怎麼自定義設定容器的主機名和 DNS 設定呢？ 秘訣就是它利用虛擬檔案來掛載到來容器的 3 個相關設定檔案。

在容器中使用 mount 命令可以看到掛載訊息：

```
$ mount
...
/dev/disk/by-uuid/1fec...ebdf on /etc/hostname type ext4 ...
/dev/disk/by-uuid/1fec...ebdf on /etc/hosts type ext4 ...
tmpfs on /etc/resolv.conf type tmpfs ...
...
```

這種機制可以讓宿主主機 DNS 訊息發生更新後，所有 Docker 容器的 dns 設定透過 `/etc/resolv.conf` 檔案立刻得到更新。

如果使用者想要手動指定容器的設定，可以利用下面的選項。

`-h HOSTNAME or --hostname=HOSTNAME` 設定容器的主機名，它會被寫到容器內的 `/etc/hostname` 和 `/etc/hosts` 。但它在容器外部看不到，既不會在 `docker ps` 中顯示，也不會在其他的容器的 `/etc/hosts` 看到。

`--link=CONTAINER_NAME:ALIAS` 選項會在建立容器的時候，新增一個其他容器的主機名到 `/etc/hosts` 檔案中，讓新容器的程式可以使用主機名 ALIAS 就可以連接它。

`--dns=IP_ADDRESS` 新增 DNS 伺服器到容器的 `/etc/resolv.conf` 中，讓容器用這個伺服器來解析所有不在 `/etc/hosts` 中的主機名。

`--dns-search=DOMAIN` 設定容器的搜索域，當設定搜索域爲 `.example.com` 時，在搜索一個名爲 host 的主機時，DNS 不僅搜索host，還會搜索 `host.example.com` 。 注意：如果沒有上述最後 2 個選項，Docker 會預設用主機上的 `/etc/resolv.conf` 來設定容器。

容器存取控制

容器的存取控制，主要透過 Linux 上的 `iptables` 防火墻來進行管理和實作。`iptables` 是 Linux 上預設的防火墻軟件，在大部分發行版中都內建。

容器存取外部網路

容器要想存取外部網路，需要本地系統的轉發支援。在Linux 系統中，檢查轉發是否打開。

```
$sysctl net.ipv4.ip_forward
net.ipv4.ip_forward = 1
```

如果爲 0，說明沒有開啓轉發，則需要手動打開。

```
$sysctl -w net.ipv4.ip_forward=1
```

如果在啓動 Docker 服務的時候設定 `--ip-forward=true`, Docker 就會自動設定系統的 `ip_forward` 參數爲 1。

容器之間存取

容器之間相互存取，需要兩方面的支援。

- 容器的網路拓撲是否已經互聯。預設情況下，所有容器都會被連接到 `docker0` 橋接器上。
- 本地系統的防火墻軟件 -- `iptables` 是否允許透過。

存取所有連接埠

當啓動 Docker 服務時候，預設會新增一條轉發策略到 iptables 的 FORWARD 鏈上。策略爲透過（ `ACCEPT` ）還是禁止（ `DROP` ）取決於設定 `--icc=true` （預設值）還是 `--icc=false` 。當然，如果手動指定 `--iptables=false` 則不會新增 `iptables` 規則。

可見，預設情況下，不同容器之間是允許網路互通的。如果爲了安全考慮，可以在 `/etc/default/docker` 檔案中設定 `DOCKER_OPTS=--icc=false` 來禁止它。

存取指定連接埠

在透過 `-icc=false` 關閉網路存取後，還可以透過 `--link=CONTAINER_NAME:ALIAS` 選項來存取容器的開放連接埠。

例如，在啓動 Docker 服務時，可以同時使用 `icc=false --iptables=true` 參數來關閉允許相互的網路存取，並讓 Docker 可以修改系統中的 `iptables` 規則。

此時，系統中的 `iptables` 規則可能是類似

```
$ sudo iptables -nL
...
Chain FORWARD (policy ACCEPT)
target     prot opt source               destination
DROP       all  --  0.0.0.0/0            0.0.0.0/0
...
```

之後，啓動容器（`docker run`）時使用 `--link=CONTAINER_NAME:ALIAS` 選項。Docker 會在 `iptable` 中爲兩個容器分別新增一條 `ACCEPT` 規則，允許相互存取開放的連接埠（取決於 Dockerfile 中的 EXPOSE 行）。

當新增了 `--link=CONTAINER_NAME:ALIAS` 選項後，新增了 `iptables` 規則。

```
$ sudo iptables -nL
...
Chain FORWARD (policy ACCEPT)
target     prot opt source               destination
ACCEPT     tcp  --  172.17.0.2           172.17.0.3           tc
p spt:80
ACCEPT     tcp  --  172.17.0.3           172.17.0.2           tc
p dpt:80
DROP       all  --  0.0.0.0/0            0.0.0.0/0
```

注意：`--link=CONTAINER_NAME:ALIAS` 中的 `CONTAINER_NAME` 目前必須是 Docker 分配的名字，或使用 `--name` 參數指定的名字。主機名則不會被識別。

映射容器連接埠到宿主主機的實作

預設情況下，容器可以主動存取到外部網路的連接，但是外部網路無法存取到容器。

容器存取外部實作

容器所有到外部網路的連接，源位址都會被NAT成本地系統的IP位址。這是使用 `iptables` 的源位址僞裝操作實作的。

查看主機的 NAT 規則。

```
$ sudo iptables -t nat -nL
...
Chain POSTROUTING (policy ACCEPT)
target     prot opt source               destination
MASQUERADE  all  --  172.17.0.0/16       !172.17.0.0/16
...
```

其中，上述規則將所有源位址在 `172.17.0.0/16` 網段，目標位址爲其他網段（外部網路）的流量動態僞裝爲從系統網卡發出。MASQUERADE 跟傳統 SNAT 的好處是它能動態從網卡取得位址。

外部存取容器實作

容器允許外部存取，可以在 `docker run` 時候透過 `-p` 或 `-P` 參數來啓用。

不管用那種辦法，其實也是在本地的 `iptable` 的 nat 表中新增相應的規則。

使用 `-P` 時：

```
$ iptables -t nat -nL
...
Chain DOCKER (2 references)
target     prot opt source               destination
DNAT       tcp  --  0.0.0.0/0            0.0.0.0/0           tc
p dpt:49153 to:172.17.0.2:80
```

使用 `-p 80:80` 時：

```
$ iptables -t nat -nL
Chain DOCKER (2 references)
target     prot opt source               destination
DNAT       tcp  --  0.0.0.0/0            0.0.0.0/0           tc
p dpt:80 to:172.17.0.2:80
```

注意：

- 這裡的規則映射了 0.0.0.0，意味著將接受主機來自所有界面的流量。使用者可以透過 `-p IP:host_port:container_port` 或 `-p IP::port` 來指定允許存取容器的主機上的 IP、界面等，以制定更嚴格的規則。
- 如果希望永久綁定到某個固定的 IP 位址，可以在 Docker 設定檔案 `/etc/default/docker` 中指定 `DOCKER_OPTS="--ip=IP_ADDRESS"`，之後重啓 Docker 服務即可生效。

設定 **docker0** 橋接器

Docker 服務預設會建立一個 `docker0` 橋接器（其上有一個 `docker0` 內部界面），它在核心層連通了其他的物理或虛擬網卡，這就將所有容器和本地主機都放到同一個物理網路。

Docker 預設指定了 `docker0` 界面 的 IP 位址和子網遮罩，讓主機和容器之間可以透過橋接器相互通信，它還給出了 MTU（界面允許接收的最大傳輸單元），通常是 1500 Bytes，或宿主主機網路路由上支援的預設值。這些值都可以在服務啓動的時候進行設定。

- `--bip=CIDR` -- IP 位址加遮罩格式，例如 192.168.1.5/24
- `--mtu=BYTES` -- 覆蓋預設的 Docker mtu 設定

也可以在設定檔案中設定 DOCKER_OPTS，然後重啓服務。 由於目前 Docker 橋接器是 Linux 橋接器，使用者可以使用 `brctl show` 來查看橋接器和連接埠連接訊息。

```
$ sudo brctl show
bridge name     bridge id               STP enabled     interfac
es
docker0         8000.3a1d7362b4ee       no              veth65f9
                                                        vethdda6
```

*註： `brctl` 命令在 Debian、Ubuntu 中可以使用 `sudo apt-get install bridge-utils` 來安裝。

每次建立一個新容器的時候，Docker 從可用的位址段中選擇一個未使用的 IP 位址分配給容器的 eth0 連接埠。使用本地主機上 `docker0` 界面的 IP 作爲所有容器的預設網關。

```
$ sudo docker run -i -t --rm base /bin/bash
$ ip addr show eth0
24: eth0: <BROADCAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast stat
e UP group default qlen 1000
    link/ether 32:6f:e0:35:57:91 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.3/16 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::306f:e0ff:fe35:5791/64 scope link
       valid_lft forever preferred_lft forever
$ ip route
default via 172.17.42.1 dev eth0
172.17.0.0/16 dev eth0  proto kernel  scope link  src 172.17.0.3
$ exit
```

自定義橋接器

除了預設的 `docker0` 橋接器，使用者也可以指定橋接器來連接各個容器。

在啓動 Docker 服務的時候，使用 `-b BRIDGE` 或 `--bridge=BRIDGE` 來指定使用的橋接器。

如果服務已經執行，那需要先停止服務，並刪除舊的橋接器。

```
$ sudo service docker stop
$ sudo ip link set dev docker0 down
$ sudo brctl delbr docker0
```

然後建立一個橋接器 `bridge0`。

```
$ sudo brctl addbr bridge0
$ sudo ip addr add 192.168.5.1/24 dev bridge0
$ sudo ip link set dev bridge0 up
```

查看確認橋接器建立並啓動。

```
$ ip addr show bridge0
4: bridge0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state UP g
roup default
    link/ether 66:38:d0:0d:76:18 brd ff:ff:ff:ff:ff:ff
    inet 192.168.5.1/24 scope global bridge0
       valid_lft forever preferred_lft forever
```

設定 Docker 服務，預設橋接到建立的橋接器上。

```
$ echo 'DOCKER_OPTS="-b=bridge0"' >> /etc/default/docker
$ sudo service docker start
```

啓動 Docker 服務。 新建一個容器，可以看到它已經橋接到了 `bridge0` 上。

可以繼續用 `brctl show` 命令查看橋接的訊息。另外，在容器中可以使用 `ip addr` 和 `ip route` 命令來查看 IP 位址設定和路由訊息。

工具和示例

在介紹自定義網路拓撲之前，你可能會對一些外部工具和例子感興趣：

pipework

Jérôme Petazzoni 編寫了一個叫 pipework 的 shell 腳本，可以幫助使用者在比較復雜的場景中完成容器的連接。

playground

Brandon Rhodes 建立了一個提供完整的 Docker 容器網路拓撲管理的 Python庫，包括路由、NAT 防火墻；以及一些提供 HTTP, SMTP, POP, IMAP, Telnet, SSH, FTP 的伺服器。

編輯網路設定檔案

Docker 1.2.0 開始支援在執行中的容器裡編輯 `/etc/hosts`，`/etc/hostname` 和 `/etc/resolve.conf` 檔案。

但是這些修改是臨時的，只在執行的容器中保留，容器終止或重啓後並不會被保存下來。也不會被 `docker commit` 提交。

示例：建立一個點到點連接

預設情況下，Docker 會將所有容器連接到由 `docker0` 提供的虛擬子網中。

使用者有時候需要兩個容器之間可以直連通信，而不用透過主機橋接器進行橋接。

解決辦法很簡單：建立一對 `peer` 界面，分別放到兩個容器中，設定成點到點鏈路類型即可。

首先啓動 2 個容器：

```
$ sudo docker run -i -t --rm --net=none base /bin/bash
root@1f1f4c1f931a:/#
$ sudo docker run -i -t --rm --net=none base /bin/bash
root@12e343489d2f:/#
```

找到程式號，然後建立網路命名空間的跟蹤檔案。

```
$ sudo docker inspect -f '{{.State.Pid}}' 1f1f4c1f931a
2989
$ sudo docker inspect -f '{{.State.Pid}}' 12e343489d2f
3004
$ sudo mkdir -p /var/run/netns
$ sudo ln -s /proc/2989/ns/net /var/run/netns/2989
$ sudo ln -s /proc/3004/ns/net /var/run/netns/3004
```

建立一對 `peer` 界面，然後設定路由

```
$ sudo ip link add A type veth peer name B

$ sudo ip link set A netns 2989
$ sudo ip netns exec 2989 ip addr add 10.1.1.1/32 dev A
$ sudo ip netns exec 2989 ip link set A up
$ sudo ip netns exec 2989 ip route add 10.1.1.2/32 dev A

$ sudo ip link set B netns 3004
$ sudo ip netns exec 3004 ip addr add 10.1.1.2/32 dev B
$ sudo ip netns exec 3004 ip link set B up
$ sudo ip netns exec 3004 ip route add 10.1.1.1/32 dev B
```

現在這 2 個容器就可以相互 ping 通，並成功建立連接。點到點鏈路不需要子網和子網遮罩。

此外，也可以不指定 `--net=none` 來建立點到點鏈路。這樣容器還可以透過原先的網路來通信。

利用類似的辦法，可以建立一個只跟主機通信的容器。但是一般情況下，更推薦使用 `--icc=false` 來關閉容器之間的通信。

實戰案例

介紹一些典型的應用情境和案例。

使用 Supervisor 來管理程式

Docker 容器在啓動的時候開啓單個程式，比如，一個 ssh 或者 apache 的 daemon 服務。但我們經常需要在一個機器上開啓多個服務，這可以有很多方法，最簡單的就是把多個啓動命令方到一個啓動腳本裡面，啓動的時候直接啓動這個腳本，另外就是安裝程式管理工具。

本小節將使用程式管理工具 supervisor 來管理容器中的多個程式。使用 Supervisor 可以更好的控制、管理、重啓我們希望執行的程式。在這裡我們演示一下如何同時使用 ssh 和 apache 服務。

設定

首先建立一個 Dockerfile，內容和各部分的解釋以下。

```
FROM ubuntu:13.04
MAINTAINER examples@docker.com
RUN echo "deb http://archive.ubuntu.com/ubuntu precise main univ
erse" > /etc/apt/sources.list
RUN apt-get update
RUN apt-get upgrade -y
```

安裝 supervisor

安裝 ssh、apache 和 supervisor。

```
RUN apt-get install -y openssh-server apache2 supervisor
RUN mkdir -p /var/run/sshd
RUN mkdir -p /var/log/supervisor
```

這裡安裝 3 個軟件，還建立了 2 個 ssh 和 supervisor 服務正常執行所需要的目錄。

```
COPY supervisord.conf /etc/supervisor/conf.d/supervisord.conf
```

新增 supervisord 的設定檔案，並複製設定檔案到對應目錄下面。

```
EXPOSE 22 80
CMD ["/usr/bin/supervisord"]
```

這裡我們映射了 22 和 80 連接埠，使用 supervisord 的可執行路徑啓動服務。

supervisor設定檔案內容

```
[supervisord]
nodaemon=true
[program:sshd]
command=/usr/sbin/sshd -D

[program:apache2]
command=/bin/bash -c "source /etc/apache2/envvars && exec /usr/s
bin/apache2 -DFOREGROUND"
```

設定檔案包含目錄和程式，第一段 supervsord 設定軟件本身，使用 nodaemon 參數來執行。第二段包含要控制的 2 個服務。每一段包含一個服務的目錄和啓動這個服務的命令。

使用方法

建立映像檔。

```
$ sudo docker build -t test/supervisord .
```

啓動 supervisor 容器。

```
$ sudo docker run -p 22 -p 80 -t -i test/supervisords
2013-11-25 18:53:22,312 CRIT Supervisor running as root (no user
 in config file)
2013-11-25 18:53:22,312 WARN Included extra file "/etc/superviso
r/conf.d/supervisord.conf" during parsing
2013-11-25 18:53:22,342 INFO supervisord started with pid 1
2013-11-25 18:53:23,346 INFO spawned: 'sshd' with pid 6
2013-11-25 18:53:23,349 INFO spawned: 'apache2' with pid 7
```

使用 `docker run` 來啓動我們建立的容器。使用多個 `-p` 來映射多個連接埠，這樣我們就能同時存取 ssh 和 apache 服務了。

可以使用這個方法建立一個只有 ssh 服務的基礎映像檔，之後建立映像檔可以使用這個映像檔爲基礎來建立

建立 tomcat/weblogic 集群

安裝 tomcat 映像檔

準備好需要的 jdk、tomcat 等軟件放到 home 目錄下面，啓動一個容器

```
docker run -t -i -v /home:/opt/data  --name mk_tomcat ubuntu /bi
n/bash
```

這條命令掛載本地 home 目錄到容器的 /opt/data 目錄，容器內目錄若不存在，則會自動建立。接下來就是 tomcat 的基本設定，jdk 環境變量設定好之後，將 tomcat 程式放到 /opt/apache-tomcat 下面 編輯 /etc/supervisor/conf.d/supervisor.conf 檔案，新增 tomcat 項

```
[supervisord]
nodaemon=true

[program:tomcat]
command=/opt/apache-tomcat/bin/startup.sh

[program:sshd]
command=/usr/sbin/sshd -D
docker commit  ac6474aeb31d  tomcat
```

新建 tomcat 檔案夾，新建 Dockerfile。

```
FROM mk_tomcat
EXPOSE  22 8080
CMD ["/usr/bin/supervisord"]
```

根據 Dockerfile 建立映像檔。

```
docker build tomcat tomcat
```

安裝 weblogic 映像檔

步驟和 tomcat 基本一致，這裡貼一下設定檔案

```
supervisor.conf
[supervisord]
nodaemon=true


[program:weblogic]
command=/opt/Middleware/user_projects/domains/base_domain/bin/st
artWebLogic.sh

[program:sshd]
command=/usr/sbin/sshd -D
dockerfile
FROM weblogic
EXPOSE  22 7001
CMD ["/usr/bin/supervisord"]
```

tomcat/weblogic 映像檔的使用

儲存的使用

在啓動的時候，使用 `-v` 參數

```
-v, --volume=[]            Bind mount a volume (e.g. from the ho
st: -v /host:/container, from docker: -v /container)
```

將本地磁碟映射到容器內部，它在主機和容器之間是實時變化的，所以我們更新程式、上傳代碼只需要更新物理主機的目錄就可以了

tomcat 和 weblogic 集群的實作

tomcat 只要開啓多個容器即可

```
docker run -d -v -p 204:22 -p 7003:8080 -v /home/data:/opt/data
--name tm1 tomcat /usr/bin/supervisord
docker run -d -v -p 205:22 -p 7004:8080 -v /home/data:/opt/data
--name tm2 tomcat /usr/bin/supervisord
docker run -d -v -p 206:22 -p 7005:8080 -v /home/data:/opt/data
--name tm3 tomcat /usr/bin/supervisord
```

這裡說一下 weblogic 的設定，大家知道 weblogic 有一個域的概念。如果要使用常規的 administrator +node 的方式部署，就需要在 supervisord 中分別寫出 administartor server 和 node server 的啓動腳本，這樣做的優點是：

- 可以使用 weblogic 的集群，同步等概念
- 部署一個集群應用程式，只需要安裝一次應用到集群上即可

缺點是：

- Docker 設定復雜了
- 沒辦法自動擴展集群的計算容量，如需新增節點，需要在 administrator 上先建立節點，然後再設定新的容器 supervisor 啓動腳本，然後再啓動容器 另外種方法是將所有的程式都安裝在 adminiserver 上面，需要擴展的時候，啓動多個節點即可，它的優點和缺點和上一種方法恰恰相反。（建議使用這種方式來部署開發和測試環境）

  ```
  docker run -d -v -p 204:22 -p 7001:7001 -v /home/data:/opt/d
  ata --name node1 weblogic /usr/bin/supervisord
  docker run -d -v -p 205:22 -p 7002:7001 -v /home/data:/opt/d
  ata --name node2 weblogic /usr/bin/supervisord
  docker run -d -v -p 206:22 -p 7003:7001 -v /home/data:/opt/d
  ata --name node3 weblogic /usr/bin/supervisord
  ```

這樣在前端使用 nginx 來做負載均衡就可以完成設定了

多臺物理主機之間的容器互聯（暴露容器到眞實網路中）

Docker 預設的橋接網卡是 docker0。它只會在本機橋接所有的容器網卡，舉例來說容器的虛擬網卡在主機上看一般叫做 veth* 而 Docker 只是把所有這些網卡橋接在一起，以下：

```
[root@opnvz ~]# brctl show
bridge name     bridge id               STP enabled     interfac
es
docker0         8000.56847afe9799       no              veth0889
                                                        veth3c7b
                                                        veth4061
```

在容器中看到的位址一般是像下面這樣的位址：

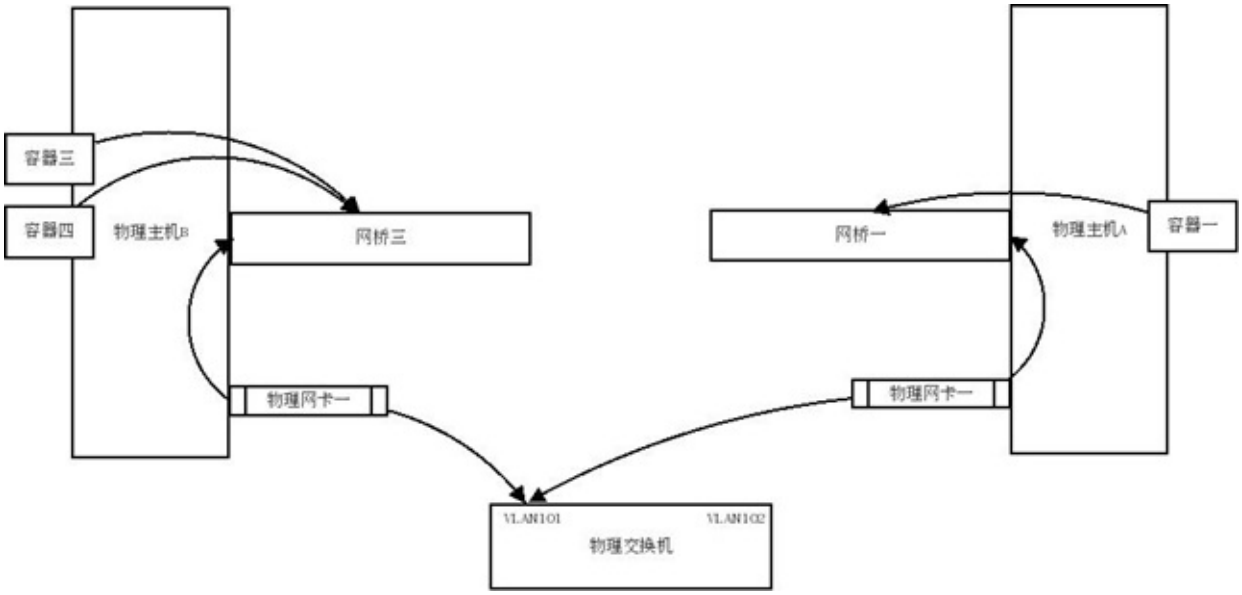
```
root@ac6474aeb31d:~# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNO
WN group default
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
11: eth0: <BROADCAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast stat
e UP group default qlen 1000
    link/ether 4a:7d:68:da:09:cf brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.3/16 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::487d:68ff:feda:9cf/64 scope link
       valid_lft forever preferred_lft forever
```

這樣就可以把這個網路看成是一個私有的網路，透過 nat 連接外網，如果要讓外網連接到容器中，就需要做連接埠映射，即 -p 參數。

如果在企業內部應用，或者做多個物理主機的集群，可能需要將多個物理主機的容器組到一個物理網路中來，那麼就需要將這個橋接器橋接到我們指定的網卡上。

拓撲圖

主機 A 和主機 B 的網卡一都連著物理交換機的同一個 vlan 101,這樣橋接器一和橋接器三就相當於在同一個物理網路中了，而容器一、容器三、容器四也在同一物理網路中了，他們之間可以相互通信，而且可以跟同一 vlan 中的其他物理機器互聯。



ubuntu 示例

下面以 ubuntu 爲例建立多個主機的容器聯網: 建立自己的橋接器,編輯 /etc/network/interface 檔案

```
auto br0
iface br0 inet static
address 192.168.7.31
netmask 255.255.240.0
gateway 192.168.7.254
bridge_ports em1
bridge_stp off
dns-nameservers 8.8.8.8 192.168.6.1
```

將 Docker 的預設橋接器綁定到這個新建的 br0 上面，這樣就將這臺機器上容器綁定到 em1 這個網卡所對應的物理網路上了。

ubuntu 修改 /etc/default/docker 檔案，新增最後一行內容

```
# Docker Upstart and SysVinit configuration file
# Customize location of Docker binary (especially for developmen
t testing).
#DOCKER="/usr/local/bin/docker"
# Use DOCKER_OPTS to modify the daemon startup options.
#DOCKER_OPTS="--dns 8.8.8.8 --dns 8.8.4.4"

# If you need Docker to use an HTTP proxy, it can also be specif
ied here.
#export http_proxy="http://127.0.0.1:3128/"

# This is also a handy place to tweak where Docker's temporary f
iles go.
#export TMPDIR="/mnt/bigdrive/docker-tmp"

DOCKER_OPTS="-b=br0"
```

在啓動 Docker 的時候 使用 -b 參數 將容器綁定到物理網路上。重啓 Docker 服務後，再進入容器可以看到它已經綁定到你的物理網路上了。

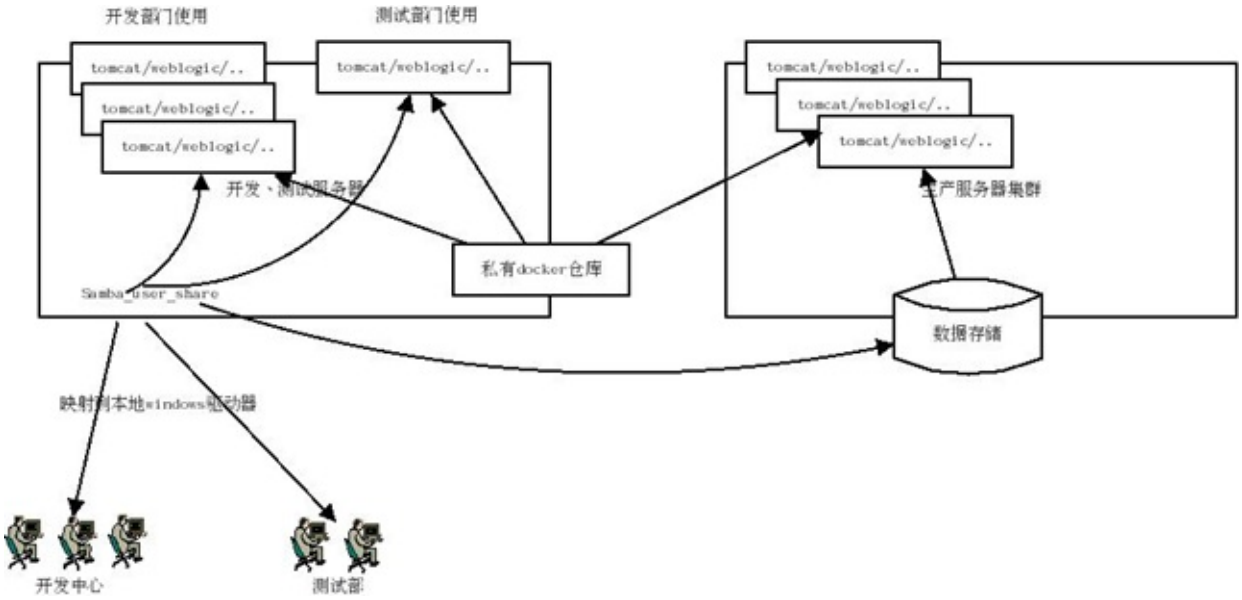
```
root@ubuntudocker:~# docker ps
CONTAINER ID        IMAGE               COMMAND             CREA
TED             STATUS              PORTS
 NAMES
58b043aa05eb        desk_hz:v1          "/startup.sh"       5 da
ys ago          Up 2 seconds        5900/tcp, 6080/tcp, 22/tcp
 yanlx
root@ubuntudocker:~# brctl show
bridge name     bridge id               STP enabled     interfac
es
br0             8000.7e6e617c8d53       no              em1
                                            vethe6e5
```

這樣就直接把容器暴露到物理網路上了，多臺物理主機的容器也可以相網路了。需要注意的是，這樣就需要自己來保證容器的網路安全了。

標準化開發測試和生產環境

對於大部分企業來說，搭建 PaaS 既沒有那個精力，也沒那個必要，用 Docker 做個人的 sandbox 用處又小了點。

可以用 Docker 來標準化開發、測試、生產環境。



Docker 占用資源小，在一臺 E5 128 G 記憶體的伺服器上部署 100 個容器都綽綽有余，可以單獨抽一個容器或者直接在宿主物理主機上部署 samba，利用 samba 的 home 分享方案將每個使用者的 home 目錄映射到開發中心和測試部門的 Windows 機器上。

針對某個專案組，由架構師搭建好一個標準的容器環境供專案組和測試部門使用，每個開發工程師可以擁有自己單獨的容器，透過 `docker run -v` 將使用者的 home 目錄映射到容器中。需要提交測試時，只需要將代碼移交給測試部門，然後分配一個容器使用 `-v` 載入測試部門的 home 目錄啓動即可。這樣，在公司內部的開發、測試基本就統一了，不會出現開發部門提交的代碼，測試部門部署不了的問題。

測試部門發布測試透過的報告後，架構師再一次檢測容器環境，就可以直接交由部署工程師將代碼和容器分別部署到生產環境中了。這種方式的部署橫向效能的擴展性也極好。

安全

評估 Docker 的安全性時，主要考慮三個方面:

- 由核心的命名空間和控制組機制提供的容器內在安全
- Docker程式（特別是服務端）本身的抗攻擊性
- 核心安全性的加強機制對容器安全性的影響

核心命名空間

Docker 容器和 LXC 容器很相似，所提供的安全特性也差不多。當用 `docker run` 啓動一個容器時，在後臺 Docker 爲容器建立了一個獨立的命名空間和控制組集合。

命名空間提供了最基礎也是最直接的隔離，在容器中執行的程式不會被執行在主機上的程式和其它容器發現和作用。

每個容器都有自己獨有的網路堆疊，意味著它們不能存取其他容器的 sockets 或界面。不過，如果主機系統上做了相應的設定，容器可以像跟主機互動一樣的和其他容器互動。當指定公共連接埠或使用 links 來連接 2 個容器時，容器就可以相互通信了（可以根據設定來限制通信的策略）。

從網路架構的角度來看，所有的容器透過本地主機的橋接器界面相互通信，就像物理機器透過物理交換機通信一樣。

那麼，核心中實作命名空間和私有網路的代碼是否足夠成熟？

核心命名空間從 2.6.15 版本（2008 年 7 月發布）之後被引入，數年間，這些機制的可靠性在諸多大型生產系統中被實踐驗證。

實際上，命名空間的想法和設計提出的時間要更早，最初是爲了在核心中引入一種機制來實作 OpenVZ 的特性。 而 OpenVZ 專案早在 2005 年就發布了，其設計和實作都已經十分成熟。

控制組

控制組是 Linux 容器機制的另外一個關鍵組件，負責實作資源的統計和限制。

它提供了很多有用的特性；以及確保各個容器可以公平地分享主機的記憶體、CPU、磁碟 IO 等資源；當然，更重要的是，控制組確保了當容器內的資源使用產生負載時不會連累主機系統。

盡管控制組不負責隔離容器之間相互存取、處理資料和程式，它在防止分散式阻斷服務（DDOS）攻擊方面是必不可少的。尤其是在多使用者的平台（比如公有或私有的 PaaS）上，控制組十分重要。例如，當某些應用程式表現異常的時候，可以保證一致地正常執行和效能。

控制組機制始於 2006 年，核心從 2.6.24 版本開始被引入。

Docker服務端的防護

執行一個容器或應用程式的核心是透過 Docker 服務端。Docker 服務的執行目前需要 root 權限，因此其安全性十分關鍵。

首先，確保只有可信的使用者才可以存取 Docker 服務。Docker 允許使用者在主機和容器間共享檔案夾，同時不需要限制容器的存取權限，這就容易讓容器突破資源限制。例如，惡意使用者啓動容器的時候將主機的根目錄 `/` 映射到容器的 `/host` 目錄中，那麼容器理論上就可以對主機的檔案系統進行任意修改了。這聽起來很瘋狂？但是事實上幾乎所有虛擬化系統都允許類似的資源共享，而沒法禁止使用者共享主機根檔案系統到虛擬機系統。

這將會造成很嚴重的安全後果。因此，當提供容器建立服務時（例如透過一個 web 伺服器），要更加注意進行參數的安全檢查，防止惡意的使用者用特定參數來建立一些破壞性的容器

爲了加強對服務端的保護，Docker 的 REST API（客户端用來跟服務端通信）在 0.5.2 之後使用本地的 Unix socket機制替代了原先綁定在 127.0.0.1 上的 TCP socket，因爲後者容易遭受跨站腳本攻擊。現在使用者使用 Unix 權限檢查來加強 socket的存取安全。

使用者仍可以利用 HTTP 提供 REST API 存取。建議使用安全機制，確保只有可信的網路或 VPN，或憑證保護機制（例如受保護的 stunnel 和 SSL 認證）下的存取可以進行。此外，還可以使用 HTTPS 和憑證來加強保護。

最近改進的 Linux 命名空間機制將可以實作使用非 root 使用者來執行全功能的容器。這將從根本上解決了容器和主機之間共享檔案系統而引起的安全問題。

終極目標是改進 2 個重要的安全特性：

- 將容器的 root 使用者映射到本地主機上的非 root 使用者，減輕容器和主機之間因權限提升而引起的安全問題；
- 允許 Docker 服務端在非 root 權限下執行，利用安全可靠的子行程來代理執行需要特權權限的操作。這些子行程將只允許在限定範圍內進行操作，例如僅僅負責虛擬網路設定或檔案系統管理、設定操作等。

最後，建議採用專用的伺服器來執行 Docker 和相關的管理服務（例如管理服務比如 ssh 監控和程式監控、管理工具 nrpe、collectd 等）。其它的業務服務都放到容器中去執行。

核心能力機制

能力機制（Capability）是 Linux 核心一個強大的特性，可以提供細緻的權限存取控制。 Linux 核心自 2.2 版本起就支援能力機制，它將權限劃分爲更加細緻的操作能力，既可以作用在程式上，也可以作用在檔案上。

例如，一個 Web 服務程式只需要綁定一個低於 1024 的連接埠的權限，並不需要 root 權限。那麼它只需要被授權 `net_bind_service` 能力即可。此外，還有很多其他的類似能力來避免程式取得 root 權限。

預設情況下，Docker 啓動的容器被嚴格限制只允許使用核心的一部分能力。

使用能力機制對加強 Docker 容器的安全有很多好處。通常，在伺服器上會執行一堆需要特權權限的程式，包括有 ssh、cron、syslogd、硬體管理工具模組（例如負載模組）、網路設定工具等等。容器跟這些程式是不同的，因爲幾乎所有的特權程式都由容器以外的支援系統來進行管理。

- ssh 存取被主機上ssh服務來管理；
- cron 通常應該作爲使用者程式執行，權限交給使用它服務的應用來處理；
- 日誌系統可由 Docker 或第三方服務管理；
- 硬體管理無關緊要，容器中也就無需執行 udevd 以及類似服務；
- 網路管理也都在主機上設定，除非特殊需求，容器不需要對網路進行設定。

從上面的例子可以看出，大部分情況下，容器並不需要“眞正的” root 權限，容器只需要少數的能力即可。爲了加強安全，容器可以禁用一些沒必要的權限。

- 完全禁止任何 mount 操作；
- 禁止直接存取本地主機的socket；
- 禁止存取一些檔案系統的操作，比如建立新的設備、修改檔案屬性等；
- 禁止模組載入。

這樣，就算攻擊者在容器中取得了 root 權限，也不能獲得本地主機的較高權限，能進行的破壞也有限。

預設情況下，Docker採用 白名單 機制，禁用 必需功能 之外的其它權限。 當然，使用者也可以根據自身需求來爲 Docker 容器啓用額外的權限。

其它安全特性

除了能力機制之外，還可以利用一些現有的安全機制來增強使用 Docker 的安全性，例如 TOMOYO, AppArmor, SELinux, GRSEC 等。

Docker 當前預設只啓用了能力機制。使用者可以採用多種方案來加強 Docker 主機的安全，例如：

- 在核心中啓用 GRSEC 和 PAX，這將增加很多編譯和執行時的安全檢查；透過位址隨機化避免惡意探測等。並且，啓用該特性不需要 Docker 進行任何設定。
- 使用一些有增強安全特性的容器模板，比如帶 AppArmor 的模板和 Redhat 帶 SELinux 策略的模板。這些模板提供了額外的安全特性。
- 使用者可以自定義存取控制機制來定制安全策略。

跟其它新增到 Docker 容器的第三方工具一樣（比如網路拓撲和檔案系統共享），有很多類似的機制，在不改變 Docker 核心情況下就可以強化現有的容器。

總結

總體來看，Docker 容器還是十分安全的，特別是在容器內不使用 root 權限來執行程式的話。

另外，使用者可以使用現有工具，比如 AppArmor, SELinux, GRSEC 來增強安全性；甚至自己在核心中實作更複雜的安全機制。

Dockerfile

使用 Dockerfile 讓使用者可以建立自定義的映像檔。

基本結構

Dockerfile 由一行行命令語句組成，並且支援以 `#` 開頭的註解行。

一般而言，Dockerfile 分爲四部分：基底映像檔資訊、維護者資訊、映像檔操作指令和容器啓動時執行指令。

例如

```
# This dockerfile uses the ubuntu image
# VERSION 2 - EDITION 1
# Author: docker_user
# Command format: Instruction [arguments / command] ..

# 基本映像檔，必須是第一個指令
FROM ubuntu

# 維護者： docker_user <docker_user at email.com> (@docker_user)
MAINTAINER docker_user docker_user@email.com

# 更新映像檔的指令
RUN echo "deb http://archive.ubuntu.com/ubuntu/ raring main univ
erse" >> /etc/apt/sources.list
RUN apt-get update && apt-get install -y nginx
RUN echo "\ndaemon off;" >> /etc/nginx/nginx.conf

# 建立新容器時要執行的指令
CMD /usr/sbin/nginx
```

其中，一開始必須指明作爲基底的映像檔名稱，接下來說明維護者資訊（建議）。

接著則是映像檔操作指令，例如 `RUN` 指令，`RUN` 指令將對映像檔執行相對應的命令。每運行一條 `RUN` 指令，映像檔就會新增一層。

最後是 `CMD` 指令，指定執行容器時的操作命令。

下面來看一個更複雜的例子

```
# Nginx
```

```
#
# VERSION               0.0.1

FROM      ubuntu
MAINTAINER Victor Vieux <victor@docker.com>

RUN apt-get update && apt-get install -y inotify-tools nginx apa
che2 openssh-server

# Firefox over VNC
#
# VERSION               0.3

FROM ubuntu

# Install vnc, xvfb in order to create a 'fake' display and fire
fox
RUN apt-get update && apt-get install -y x11vnc xvfb firefox
RUN mkdir /.vnc
# Setup a password
RUN x11vnc -storepasswd 1234 ~/.vnc/passwd
# Autostart firefox (might not be the best way, but it does the
trick)
RUN bash -c 'echo "firefox" >> /.bashrc'

EXPOSE 5900
CMD    ["x11vnc", "-forever", "-usepw", "-create"]

# Multiple images example
#
# VERSION               0.1

FROM ubuntu
RUN echo foo > bar
# Will output something like ===> 907ad6c2736f

FROM ubuntu
RUN echo moo > oink
# Will output something like ===> 695d7793cbe4
```

```
# You'll now have two images, 907ad6c2736f with /bar, and 695d77
93cbe4 with
# /oink.
```

指令

指令的一般格式爲 `INSTRUCTION arguments` ，指令包括 `FROM` 、 `MAINTAINER` 、 `RUN` 等。

FROM

格式爲 `FROM <image>` 或 `FROM <image>:<tag>` 。

第一條指令必須爲 `FROM` 指令。並且，如果在同一個Dockerfile中建立多個映像檔時，可以使用多個 `FROM` 指令（每個映像檔一次）。

MAINTAINER

格式爲 `MAINTAINER <name>` ，指定維護者訊息。

RUN

格式爲 `RUN <command>` 或 `RUN ["executable", "param1", "param2"]` 。

前者將在 shell 終端中運行命令，即 `/bin/sh -c` ；後者則使用 `exec` 執行。指定使用其它終端可以透過第二種方式實作，例如 `RUN ["/bin/bash", "-c", "echo hello"]` 。

每條 `RUN` 指令將在當前映像檔基底上執行指定命令，並產生新的映像檔。當命令較長時可以使用 `\` 來換行。

CMD

支援三種格式

- `CMD ["executable","param1","param2"]` 使用 `exec` 執行，推薦使用；
- `CMD command param1 param2` 在 `/bin/sh` 中執行，使用在給需要互動的指令；
- `CMD ["param1","param2"]` 提供給 `ENTRYPOINT` 的預設參數；

指定啓動容器時執行的命令，每個 Dockerfile 只能有一條 `CMD` 命令。如果指定了多條命令，只有最後一條會被執行。

如果使用者啓動容器時候指定了運行的命令，則會覆蓋掉 `CMD` 指定的命令。

EXPOSE

格式爲 `EXPOSE <port> [<port>...]`。

設定 Docker 伺服器容器對外的埠號，供外界使用。在啓動容器時需要透過 -P，Docker 會自動分配一個埠號轉發到指定的埠號。

ENV

格式爲 `ENV <key> <value>`。指定一個環境變數，會被後續 `RUN` 指令使用，並在容器運行時保持。

例如

```
ENV PG_MAJOR 9.3
ENV PG_VERSION 9.3.4
RUN curl -SL http://example.com/postgres-$PG_VERSION.tar.xz | ta
r -xJC /usr/src/postgress && …
ENV PATH /usr/local/postgres-$PG_MAJOR/bin:$PATH
```

ADD

格式爲 `ADD <src> <dest>`。

該命令將複製指定的 `<src>` 到容器中的 `<dest>`。其中 `<src>` 可以是 Dockerfile 所在目錄的相對路徑；也可以是一個 URL；還可以是一個 tar 檔案（其複製後會自動解壓縮）。

COPY

格式爲 `COPY <src> <dest>`。

複製本地端的 `<src>`（爲 Dockerfile 所在目錄的相對路徑）到容器中的 `<dest>`。

當使用本地目錄爲根目錄時，推薦使用 `COPY`。

ENTRYPOINT

兩種格式：

- `ENTRYPOINT ["executable", "param1", "param2"]`
- `ENTRYPOINT command param1 param2`（shell中執行）。

指定容器啓動後執行的命令，並且不會被 `docker run` 提供的參數覆蓋。

每個 Dockerfile 中只能有一個 `ENTRYPOINT`，當指定多個時，只有最後一個會生效。

VOLUME

格式爲 `VOLUME ["/data"]`。

建立一個可以從本地端或其他容器掛載的掛載點，一般用來存放資料庫和需要保存的資料等。

USER

格式爲 `USER daemon`。

指定運行容器時的使用者名稱或 UID，後續的 `RUN` 也會使用指定使用者。

當服務不需要管理員權限時，可以透過該命令指定運行使用者。並且可以在之前建立所需要的使用者，例如：`RUN groupadd -r postgres && useradd -r -g postgres postgres`。要臨時取得管理員權限可以使用 `gosu`，而不推薦 `sudo`。

WORKDIR

格式爲 `WORKDIR /path/to/workdir`。

爲後續的 `RUN` 、 `CMD` 、 `ENTRYPOINT` 指令指定工作目錄。

可以使用多個 `WORKDIR` 指令，後續命令如果參數是相對路徑，則會基於之前命令指定的路徑。例如

```
WORKDIR /a
WORKDIR b
WORKDIR c
RUN pwd
```

則最終路徑爲 `/a/b/c` 。

ONBUILD

格式爲 `ONBUILD [INSTRUCTION]` 。

指定當建立的映像檔作爲其它新建立映像檔的基底映像檔時，所執行的操作指令。

例如，Dockerfile 使用以下的內容建立了映像檔 `image-A` 。

```
[...]
ONBUILD ADD . /app/src
ONBUILD RUN /usr/local/bin/python-build --dir /app/src
[...]
```

如果基於 image-A 建立新的映像檔時，新的 Dockerfile 中使用 `FROM image-A` 指定基底映像檔時，會自動執行 `ONBUILD` 指令內容，等於在後面新增了兩條指令。

```
FROM image-A

#Automatically run the following
ADD . /app/src
RUN /usr/local/bin/python-build --dir /app/src
```

使用 `ONBUILD` 指令的映像檔，推薦在標籤中註明，例如 `ruby:1.9-onbuild` 。

建立映像檔

編輯完成 Dockerfile 之後，可以透過 `docker build` 命令建立映像檔。

基本的格式爲 `docekr build [選項] 路徑` ，該命令將讀取指定路徑下（包括子目錄）的 Dockerfile，並將該路徑下所有內容發送給 Docker 伺服端，由伺服端來建立映像檔。因此一般會建議放置 Dockerfile 的目錄爲空目錄。也可以透過 `.dockerignore` 檔案（每一行新增一條排除模式：exclusion patterns）來讓 Docker 忽略路徑下的目錄和檔案。

要指定映像檔的標籤資訊，可以透過 `-t` 選項，例如

```
$ sudo docker build -t myrepo/myapp /tmp/test1/
```

從映像檔產生 Dockerfile

CenturyLinkLabs 釋出 dockerfile-from-image 工具，以逆向工程建立出 Dockerfile。類似 `docker history` 指令，透過映像檔每一層的 metadata 來重建出那 Dockerfile，即便沒有提供任何資訊。

使用方法

首先 `docker pull centurylink/dockerfile-from-image` 這已包好 Ruby script 的映像檔，接下來，執行下面命令，就可得到反推所產生的 Dockerfile.txt：

```
docker run -v /var/run/docker.sock:/var/run/docker.sock \
  centurylink/dockerfile-from-image <IMAGE_TAG_OR_ID> > Dockerfi
le.txt
```

那 `<IMAGE_TAG_OR_ID>` 參數可以任何包含 tag 的映像檔名稱。

範例

以下是個示範，如何將官方 Ruby 的映像檔來產生出 Dockerfile。

```
$ docker pull ruby
Pulling repository ruby

$ docker run -v /run/docker.sock:/run/docker.sock centurylink/do
ckerfile-from-image
Usage: dockerfile-from-image.rb [options] <image_id>
    -f, --full-tree                  Generate Dockerfile for all
 parent layers
    -h, --help                       Show this message

$ docker run -v /run/docker.sock:/run/docker.sock centurylink/do
ckerfile-from-image ruby
FROM buildpack-deps:latest
RUN useradd -g users user
RUN apt-get update && apt-get install -y bison procps
RUN apt-get update && apt-get install -y ruby
ADD dir:03090a5fdc5feb8b4f1d6a69214c37b5f6d653f5185cddb6bf7fd71e
6ded561c in /usr/src/ruby
WORKDIR /usr/src/ruby
RUN chown -R user:users .
USER user
RUN autoconf && ./configure --disable-install-doc
RUN make -j"$(nproc)"
RUN make check
USER root
RUN apt-get purge -y ruby
RUN make install
RUN echo 'gem: --no-rdoc --no-ri' >> /.gemrc
RUN gem install bundler
ONBUILD ADD . /usr/src/app
ONBUILD WORKDIR /usr/src/app
ONBUILD RUN [ ! -e Gemfile ] || bundle install --system
```

底層實作

Docker 底層的核心技術包括 Linux 上的命名空間（Namespaces）、控制組（Control groups）、Union 檔案系統（Union file systems）和容器格式（Container format）。

我們知道，傳統的虛擬機透過在宿主主機中執行 hypervisor 來模擬一整套完整的硬體環境提供給虛擬機的作業系統。虛擬機系統看到的環境是可限制的，也是彼此隔離的。 這種直接的做法實作了對資源最完整的封裝，但很多時候往往意味著系統資源的浪費。 例如，以宿主機和虛擬機系統都爲 Linux 系統爲例，虛擬機中執行的應用其實可以利用宿主機系統中的執行環境。

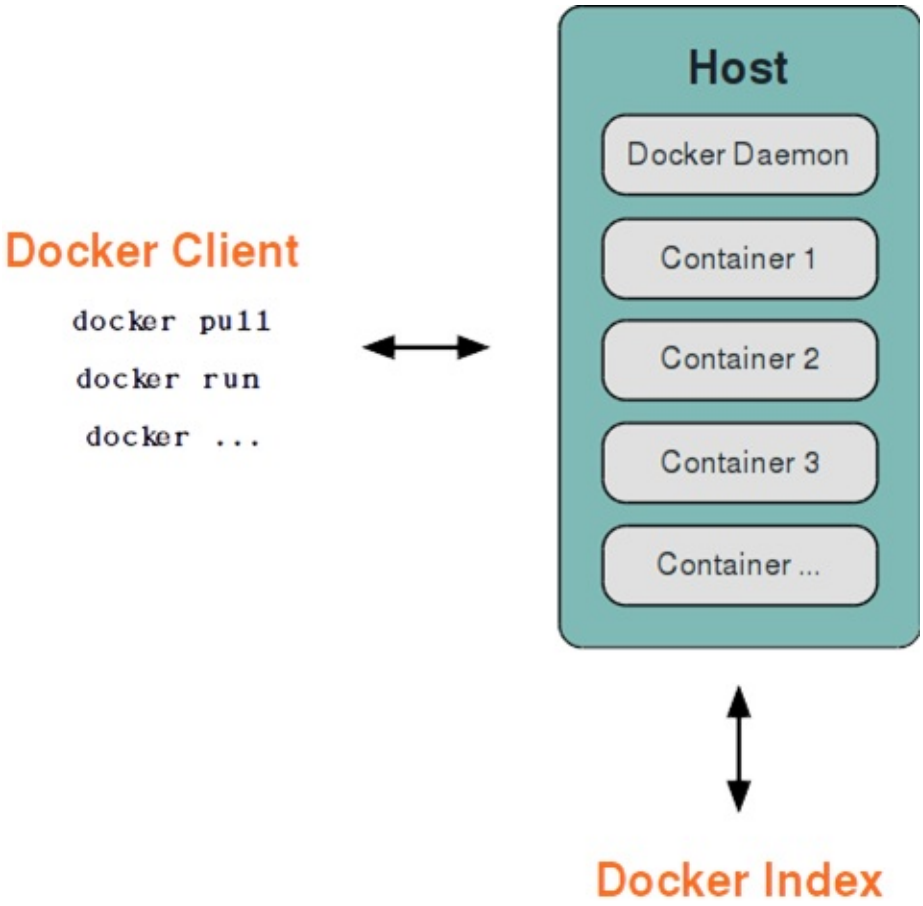
我們知道，在作業系統中，包括核心、檔案系統、網路、PID、UID、IPC、記憶體、硬盤、CPU 等等，所有的資源都是應用程式直接共享的。 要想實作虛擬化，除了要實作對記憶體、CPU、網路 IO、硬盤 IO、儲存空間等的限制外，還要實作檔案系統、網路、PID、UID、IPC 等等的相互隔離。

前者相對容易實作一些，後者則需要宿主機系統的深入支援。

隨著 Linux 系統對於命名空間功能的完善實作，程式員已經可以實作上面的所有需求，讓某些程式在彼此隔離的命名空間中執行。大家雖然都共用一個核心和某些執行時環境（例如一些系統命令和系統函式庫），但是彼此卻看不到，都以爲系統中只有自己的存在。這種機制就是容器（Container），利用命名空間來做權限的隔離控制，利用 cgroups 來做資源分配。

基本架構

Docker 採用了 C/S 架構，包括客户端和服務端。 Docker daemon 作爲服務端接受來自客户的請求，並處理這些請求（建立、執行、分發容器）。 客户端和服務端既可以執行在一個機器上，也可透過 socket 或者 RESTful API 來進行通信。



Docker daemon 一般在宿主主機後臺執行，等待接收來自客户端的消息。 Docker 客户端則爲使用者提供一系列可執行命令，使用者用這些命令實作跟 Docker daemon 互動。

命名空間

命名空間是 Linux 核心一個強大的特性。每個容器都有自己單獨的命名空間，執行在其中的應用都像是在獨立的作業系統中執行一樣。命名空間保證了容器之間彼此互不影響。

pid 命名空間

不同使用者的程式就是透過 pid 命名空間隔離開的，且不同命名空間中可以有相同 pid。所有的 LXC 程式在 Docker 中的父程式爲 Docker 程式，每個 LXC 程式具有不同的命名空間。同時由於允許巢狀結構，因此可以很方便的實作巢狀結構的 Docker 容器。

net 命名空間

有了 pid 命名空間, 每個命名空間中的 pid 能夠相互隔離，但是網路連接埠還是共享 host 的連接埠。網路隔離是透過 net 命名空間實作的， 每個 net 命名空間有獨立的網路設備, IP 位址, 路由表, /proc/net 目錄。這樣每個容器的網路就能隔離開來。Docker 預設採用 veth 的方式，將容器中的虛擬網卡同 host 上的一個 Docker 橋接器 docker0 連接在一起。

ipc 命名空間

容器中程式互動還是採用了 Linux 常見的程式間互動方法 (interprocess communication - IPC), 包括信號量、消息隊列和共享記憶體等。然而同 VM 不同的是，容器的程式間互動實際上還是 host 上具有相同 pid 命名空間中的程式間互動，因此需要在 IPC 資源申請時加入命名空間訊息，每個 IPC 資源有一個唯一的 32 位 id。

mnt 命名空間

類似 chroot，將一個程式放到一個特定的目錄執行。mnt 命名空間允許不同命名空間的程式看到的檔案結構不同，這樣每個命名空間 中的程式所看到的檔案目錄就被隔離開了。同 chroot 不同，每個命名空間中的容器在 /proc/mounts 的訊息只包含

所在命名空間的 mount point。

uts 命名空間

UTS("UNIX Time-sharing System") 命名空間允許每個容器擁有獨立的 hostname 和 domain name, 使其在網路上可以被視作一個獨立的節點而非主機上的一個程式。

user 命名空間

每個容器可以有不同的使用者和組 id, 也就是說可以在容器內用容器內部的使用者執行程式而非主機上的使用者。

*註：關於 Linux 上的命名空間，這篇文章 介紹的很好，另外 Michael Crosby - Creating containers - Part 1 也非常推薦。。

控制組

控制組（cgroups）是 Linux 核心的一個特性，主要用來對共享資源進行隔離、限制、統計等。只有能控制分配到容器的資源，才能避免當多個容器同時執行時的對系統資源的競爭。

控制組技術最早是由 Google 的程式員 2006 年起提出，Linux 核心自 2.6.24 開始支援。

控制組可以提供對容器的記憶體、CPU、磁碟 IO 等資源的限制和統計管理。

Union 檔案系統

Union檔案系統（UnionFS）是一種分層、輕量級並且高效能的檔案系統，它支援對檔案系統的修改作爲一次提交來一層層的疊加，同時可以將不同目錄掛載到同一個虛擬檔案系統下 (unite several directories into a single virtual filesystem)。

Union 檔案系統是 Docker 映像檔的基礎。映像檔可以透過分層來進行繼承，基於基礎映像檔（沒有父映像檔），可以制作各種具體的應用映像檔。

另外，不同 Docker 容器就可以共享一些基礎的檔案系統層，同時再加上自己獨有的改動層，大大提高了儲存的效率。

Docker 中使用的 AUFS（AnotherUnionFS）就是一種 Union FS。 AUFS 支援爲每一個成員目錄（類似 Git 的分支）設定唯讀（readonly）、讀寫（readwrite）和寫出（whiteout-able）權限, 同時 AUFS 裡有一個類似分層的概念, 對唯讀權限的分支可以邏輯上進行增量地修改 (不影響唯讀部分的)。

Docker 目前支援的 Union 檔案系統種類包括 AUFS, btrfs, vfs 和 DeviceMapper。

容器格式

最初，Docker 採用了 LXC 中的容器格式。自 1.20 版本開始，Docker 也開始支援新的 libcontainer 格式，並作爲預設選項。

對更多容器格式的支援，還在進一步的發展中。

Docker 網路實作

Docker 的網路實作其實就是利用了 Linux 上的網路命名空間和虛擬網路設備（特別是 veth pair）。建議先熟悉了解這兩部分的基本概念再閱讀本章。

基本原理

首先，要實作網路通信界面，機器需要至少一個網路界面（物理界面或虛擬界面）來收發資料包；此外，如果不同子網之間要進行通信，需要路由機制。

Docker 中的網路界面預設都是虛擬的界面。虛擬界面的優勢之一是轉發效率較高。 Linux 透過在核心中進行資料複製來實作虛擬界面之間的資料轉發，發送界面的發送緩存中的資料包被直接複製到接收界面的接收緩存中。對於本地系統和容器內系統看來就像是一個正常的乙太網卡，只是它不需要眞正同外部網路設備通信，速度要快很多。

Docker 容器網路就利用了這項技術。它在本地主機和容器內分別建立一個虛擬界面，並讓它們彼此連通（這樣的一對界面叫做 `veth pair` ）。

建立網路參數

Docker 建立一個容器的時候，會執行以下操作：

- 建立一對虛擬界面，分別放到本地主機和新容器中；
- 本地主機一端橋接到預設的 docker0 或指定橋接器上，並具有一個唯一的名字，如 veth65f9；
- 容器一端放到新容器中，並修改名字作爲 eth0，這個界面只在容器的命名空間可見；
- 從橋接器可用位址段中取得一個未使用位址分配給容器的 eth0，並設定預設路由到橋接網卡 veth65f9。

完成這些之後，容器就可以使用 eth0 虛擬網卡來連接其他容器和其他網路。

可以在 `docker run` 的時候透過 `--net` 參數來指定容器的網路設定，有4個可選值：

- `--net=bridge` 這個是預設值，連接到預設的橋接器。

- `--net=host` 告訴 Docker 不要將容器網路放到隔離的命名空間中，即不要容器化容器內的網路。此時容器使用本地主機的網路，它擁有完全的本地主機界面存取權限。容器程式可以跟主機其它 root 程式一樣可以打開低範圍的連接埠，可以存取本地網路服務比如 D-bus，還可以讓容器做一些影響整個主機系統的事情，比如重啓主機。因此使用這個選項的時候要非常小心。如果進一步的使用 `--privileged=true`，容器會被允許直接設定主機的網路堆棧。
- `--net=container:NAME_or_ID` 讓 Docker 將新建容器的程式放到一個已存在容器的網路堆疊中，新容器程式有自己的檔案系統、程式列表和資源限制，但會和已存在的容器共享 IP 位址和連接埠等網路資源，兩者程式可以直接透過 `lo` 迴路界面通信。
- `--net=none` 讓 Docker 將新容器放到隔離的網路堆疊中，但是不進行網路設定。之後，使用者可以自己進行設定。

網路設定細節

使用者使用 `--net=none` 後，可以自行設定網路，讓容器達到跟平常一樣具有存取網路的權限。透過這個過程，可以了解 Docker 設定網路的細節。

首先，啓動一個 `/bin/bash` 容器，指定 `--net=none` 參數。

```
$ sudo docker run -i -t --rm --net=none base /bin/bash
root@63f36fc01b5f:/#
```

在本地主機查找容器的程式 id，並爲它建立網路命名空間。

```
$ sudo docker inspect -f '{{.State.Pid}}' 63f36fc01b5f
2778
$ pid=2778
$ sudo mkdir -p /var/run/netns
$ sudo ln -s /proc/$pid/ns/net /var/run/netns/$pid
```

檢查橋接網卡的 IP 和子網遮罩訊息。

```
$ ip addr show docker0
21: docker0: ...
inet 172.17.42.1/16 scope global docker0
...
```

建立一對 “veth pair” 界面 A 和 B，綁定 A 到橋接器 `docker0`，並啓用它

```
$ sudo ip link add A type veth peer name B
$ sudo brctl addif docker0 A
$ sudo ip link set A up
```

將B放到容器的網路命名空間，命名爲 eth0，啓動它並設定一個可用 IP（橋接網段）和預設網關。

```
$ sudo ip link set B netns $pid
$ sudo ip netns exec $pid ip link set dev B name eth0
$ sudo ip netns exec $pid ip link set eth0 up
$ sudo ip netns exec $pid ip addr add 172.17.42.99/16 dev eth0
$ sudo ip netns exec $pid ip route add default via 172.17.42.1
```

以上，就是 Docker 設定網路的具體過程。

當容器結束後，Docker 會清空容器，容器內的 eth0 會隨網路命名空間一起被清除，A 界面也被自動從 `docker0` 卸載。

此外，使用者可以使用 `ip netns exec` 命令來在指定網路命名空間中進行設定，從而設定容器內的網路。

Docker命令查詢

基本語法

```
docker [OPTIONS] COMMAND [arg...]
```

一般來說，Docker 命令可以用來管理 daemon，或者透過 CLI 命令管理映像檔和容器。可以透過 `man docker` 來查看這些命令。

選項

```
-D=true|false
    使用 debug 模式。預設爲 false。

-H, --host=[unix:///var/run/docker.sock]: tcp://[host:port]來綁定或者 unix://[/path/to/socket] 來使用。
    在 daemon 模式下綁定的 socket，透過一個或多個 tcp://host:port, unix:///path/to/socket, fd://* or fd://socketfd 來指定。

--api-enable-cors=true|false
    在遠端 API 中啓用 CORS 頭。預設爲 false。

-b=""
    將容器掛載到一個已存在的橋接器上。指定爲 'none' 時則禁用容器的網路。

--bip=""
    讓動態建立的 docker0 採用指定的 CIDR 位址; 與 -b 選項互斥。

-d=true|false
    使用 daemon 模式。預設爲 false。

--dns=""
    讓 Docker 使用指定的 DNS 伺服器。

-g=""
```

```
    指定 Docker 執行時的 root 路徑。預設爲 /var/lib/docker。

--icc=true|false
    啓用容器間通信。預設爲 true。

--ip=""
    綁定連接埠時候的預設 IP 位址。預設爲 0.0.0.0。

--iptables=true|false
    禁止 Docker 新增 iptables 規則。預設爲 true。

--mtu=VALUE
    指定容器網路的 mtu。預設爲 1500。

-p=""
    指定 daemon 的 PID 檔案路徑。預設爲 /var/run/docker.pid。

-s=""
    強制 Docker 執行時使用指定的儲存驅動。

-v=true|false
    輸出版本資訊並退出。預設值爲 false。

--selinux-enabled=true|false
    啓用 SELinux 支援。預設值爲 false。SELinux 目前不支援 BTRFS 儲存驅
動。
```

命令

Docker 的命令可以採用 `docker-CMD` 或者 `docker CMD` 的方式執行。兩者一致。

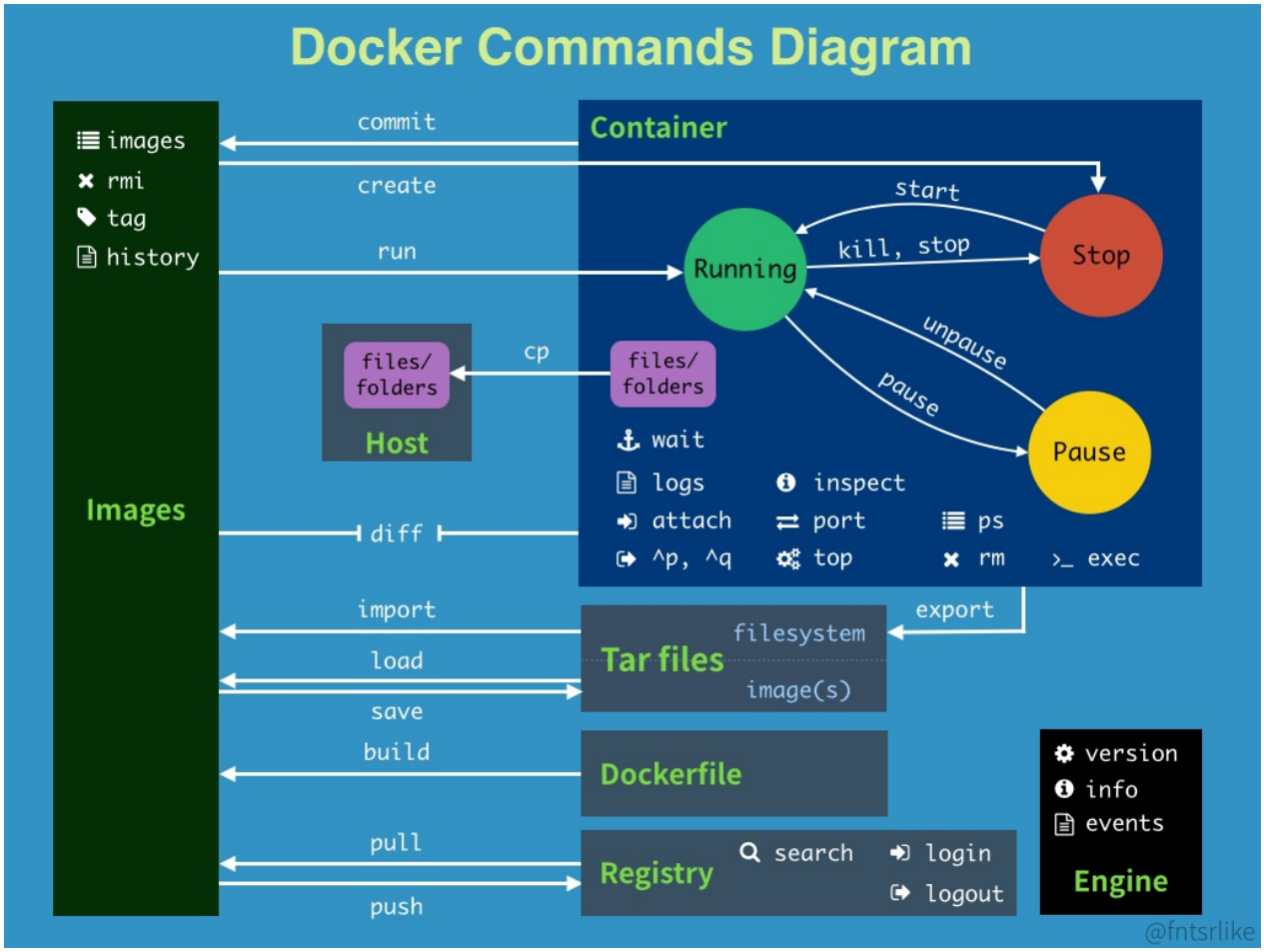
```
docker-attach(1)
    依附到一個正在執行的容器中。

docker-build(1)
    從一個 Dockerfile 建立一個映像檔
```

```
docker-commit(1)
    從一個容器的修改中建立一個新的映像檔

docker-cp(1)
    從容器中複製檔案到宿主系統中

docker-diff(1)
    檢查一個容器檔案系統的修改

docker-events(1)
    從服務端取得實時的事件

docker-export(1)
    匯出容器內容爲一個 tar 包

docker-history(1)
    顯示一個映像檔的歷史

docker-images(1)
    列出存在的映像檔

docker-import(1)
    匯入一個檔案（典型爲 tar 包）路徑或目錄來建立一個映像檔

docker-info(1)
    顯示一些相關的系統資訊

docker-inspect(1)
    顯示一個容器的底層具體資訊。

docker-kill(1)
    關閉一個執行中的容器 (包括程式和所有資源)

docker-load(1)
    從一個 tar 包中載入一個映像檔

docker-login(1)
    註冊或登錄到一個 Docker 的倉庫伺服器

docker-logout(1)
```

```
    從 Docker 的倉庫伺服器登出

docker-logs(1)
    取得容器的 log 資訊

docker-pause(1)
    暫停一個容器中的所有程式

docker-port(1)
    查找一個 nat 到一個私有網口的公共口

docker-ps(1)
    列出容器

docker-pull(1)
    從一個Docker的倉庫伺服器下拉一個映像檔或倉庫

docker-push(1)
    將一個映像檔或者倉庫推送到一個 Docker 的註冊伺服器

docker-restart(1)
    重新啓動一個執行中的容器

docker-rm(1)
    刪除指定的數個容器

docker-rmi(1)
    刪除指定的數個映像檔

docker-run(1)
    建立一個新容器，並在其中執行指定命令

docker-save(1)
    保存一個映像檔爲 tar 包檔案

docker-search(1)
    在 Docker index 中搜索一個映像檔

docker-start(1)
    啓動一個容器
```

```
docker-stop(1)
    終止一個執行中的容器

docker-tag(1)
    爲一個映像檔打標籤

docker-top(1)
    查看一個容器中的正在執行的程式資訊

docker-unpause(1)
    將一個容器內所有的程式從暫停狀態中恢復

docker-version(1)
    輸出 Docker 的版本資訊

docker-wait(1)
    阻塞直到一個容器終止，然後輸出它的退出符
```

一張圖總結 Docker 的命令


Docker Commands Diagram
images
rmi
tag
history
Images
commit
create
run
diff
import
load
save
build
pull
push
Container
start
kill, stop
Stop
Running
unpause
pause
Pause
files/
folders
cp
files/
folders
Host
wait
logs
inspect
attach
port
ps
^p, ^q
top
rm
exec
export
Tar files
filesystem
image(s)
Dockerfile
Registry
search
login
logout
version
info
events
Engine
@fntsrlike

常見倉庫介紹

本章將介紹常見的一些倉庫和映像檔的功能，使用方法和建立它們的 Dockerfile 等。包括 Ubuntu、CentOS、MySQL、MongoDB、Redis、Nginx、Wordpress、Node.js 等。

Ubuntu

基本訊息

Ubuntu 是流行的 Linux 發行版，其內建軟件版本往往較新一些。 該倉庫提供了 Ubuntu 從 12.04 ~ 14.10 各個版本的映像檔。

使用方法

預設會啓動一個最小化的 Ubuntu 環境。

```
$ sudo docker run --name some-ubuntu -i -t ubuntu
root@523c70904d54:/#
```

Dockerfile

- 12.04 版本
- 14.04 版本
- 14.10 版本

CentOS

基本訊息

CentOS 是流行的 Linux 發行版，其軟件套件大多跟 RedHat 系列保持一致。 該倉庫提供了 CentOS 從 5 ~ 7 各個版本的映像檔。

使用方法

預設會啓動一個最小化的 CentOS 環境。

```
$ sudo docker run --name some-centos -i -t centos bash
bash-4.2#
```

Dockerfile

- CentOS 5 版本
- CentOS 6 版本
- CentOS 7 版本

MySQL

基本訊息

MySQL 是開源的關聯資料庫實作。 該倉庫提供了 MySQL 各個版本的映像檔，包括 5.6 系列、5.7 系列等。

使用方法

預設會在 `3306` 連接埠啓動資料庫。

```
$ sudo docker run --name some-mysql -e MYSQL_ROOT_PASSWORD=mysec
retpassword -d mysql
```

之後就可以使用其它應用來連接到該容器。

```
$ sudo docker run --name some-app --link some-mysql:mysql -d app
lication-that-uses-mysql
```

或者透過 `mysql` 。

```
$ sudo docker run -it --link some-mysql:mysql --rm mysql sh -c '
exec mysql -h"$MYSQL_PORT_3306_TCP_ADDR" -P"$MYSQL_PORT_3306_TCP
_PORT" -uroot -p"$MYSQL_ENV_MYSQL_ROOT_PASSWORD"'
```

Dockerfile

- 5.6 版本
- 5.7 版本

MongoDB

基本訊息

MongoDB 是開源的 NoSQL 資料庫實作。 該倉庫提供了 MongoDB 2.2 ~ 2.7 各個版本的映像檔。

使用方法

預設會在 `27017` 連接埠啓動資料庫。

```
$ sudo docker run --name some-mongo -d mongo
```

使用其他應用連接到容器，可以用

```
$ sudo docker run --name some-app --link some-mongo:mongo -d app
lication-that-uses-mongo
```

或者透過 `mongo`

```
$ sudo docker run -it --link some-mongo:mongo --rm mongo sh -c '
exec mongo "$MONGO_PORT_27017_TCP_ADDR:$MONGO_PORT_27017_TCP_POR
T/test"'
```

Dockerfile

- 2.2 版本
- 2.4 版本
- 2.6 版本
- 2.7 版本

Redis

基本訊息

Redis 是開源的記憶體 Key-Value 資料庫實作。

使用方法

預設會在 `6379` 連接埠啓動資料庫。

```
$ sudo docker run --name some-redis -d redis
```

另外還可以啓用 持久儲存。

```
$ sudo docker run --name some-redis -d redis redis-server --appe
ndonly yes
```

預設資料儲存位置在 `VOLUME/data`。可以使用 `--volumes-from some-volume-container` 或 `-v /docker/host/dir:/data` 將資料存放到本地。

使用其他應用連接到容器，可以用

```
$ sudo docker run --name some-app --link some-redis:redis -d app
lication-that-uses-redis
```

或者透過 `redis-cli`

```
$ sudo docker run -it --link some-redis:redis --rm redis sh -c '
exec redis-cli -h "$REDIS_PORT_6379_TCP_ADDR" -p "$REDIS_PORT_63
79_TCP_PORT"'
```

Dockerfile

- 2.6 版本

- 最新 2.8 版本

Nginx

基本訊息

Nginx 是開源的、有效率的 Web 伺服器實作，支援 HTTP、HTTPS、SMTP、POP3、IMAP 等協議。 該倉庫提供了 Nginx 1.0 ~ 1.7 各個版本的映像檔。

使用方法

下面的命令將作爲一個靜態頁面伺服器啓動。

```
$ sudo docker run --name some-nginx -v /some/content:/usr/share/
nginx/html:ro -d nginx
```

使用者也可以不使用這種映射方式，透過利用 Dockerfile 來直接將靜態頁面內容放到映像檔中，內容爲

```
FROM nginx
COPY static-html-directory /usr/share/nginx/html
```

之後建立新的映像檔，並啓動一個容器。

```
$ sudo docker build -t some-content-nginx .
$ sudo docker run --name some-nginx -d some-content-nginx
```

開放連接埠，並映射到本地的 `8080` 連接埠。

```
sudo docker run --name some-nginx -d -p 8080:80 some-content-ngi
nx
```

Nginx的預設設定檔案路徑爲 `/etc/nginx/nginx.conf` ，可以透過映射它來使用本地的設定檔案，例如

```
docker run --name some-nginx -v /some/nginx.conf:/etc/nginx/ngin
x.conf:ro -d nginx
```

使用設定檔案時，爲了在容器中正常執行，需要保持 `daemon off;` 。

Dockerfile

- 1 ~ 1.7 版本

WordPress

基本訊息

WordPress 是開源的 Blog 和內容管理系統框架，它基於 PHP 和 MySQL。該倉庫提供了 WordPress 4.0 版本的映像檔。

使用方法

啓動容器需要 MySQL 的支援，預設連接埠爲 `80`。

```
$ sudo docker run --name some-wordpress --link some-mysql:mysql
-d wordpress
```

啓動 WordPress 容器時可以指定的一些環境參數包括

- `-e WORDPRESS_DB_USER=...` 預設爲 “root”
- `-e WORDPRESS_DB_PASSWORD=...` 預設爲連接 mysql 容器的環境變量 `MYSQL_ROOT_PASSWORD` 的值
- `-e WORDPRESS_DB_NAME=...` 預設爲 “wordpress”
- `-e WORDPRESS_AUTH_KEY=...`, `-e WORDPRESS_SECURE_AUTH_KEY=...`, `-e WORDPRESS_LOGGED_IN_KEY=...`, `-e WORDPRESS_NONCE_KEY=...`, `-e WORDPRESS_AUTH_SALT=...`, `-e WORDPRESS_SECURE_AUTH_SALT=...`, `-e WORDPRESS_LOGGED_IN_SALT=...`, `-e WORDPRESS_NONCE_SALT=...` 預設爲隨機 SHA1 串

Dockerfile

- 4.0 版本

Node.js

基本訊息

Node.js是基於 JavaScript 的可擴展服務端和網路軟件開發平台。 該倉庫提供了 Node.js 0.8 ~ 0.11 各個版本的映像檔。

使用方法

在專案中建立一個 Dockerfile。

```
FROM node:0.10-onbuild
# replace this with your application's default port
EXPOSE 8888
```

然後建立映像檔，並啓動容器

```
$ sudo docker build -t my-nodejs-app
$ sudo docker run -it --rm --name my-running-app my-nodejs-app
```

也可以直接執行一個簡單容器。

```
$ sudo docker run -it --rm --name my-running-script -v "$(pwd)":
/usr/src/myapp -w /usr/src/myapp node:0.10 node your-daemon-or-s
cript.js
```

Dockerfile

- 0.8 版本
- 0.10 版本
- 0.11 版本

資源鏈接

- Docker 主站台: https://www.docker.com
- Docker 註冊中心 API: http://docs.docker.com/reference/api/registry_api/
- Docker Hub API: http://docs.docker.com/reference/api/docker-io_api/
- Docker 遠端應用 API: http://docs.docker.com/reference/api/docker_remote_api/
- Dockerfile 參考: https://docs.docker.com/reference/builder/
- Dockerfile 最佳實踐: https://docs.docker.com/articles/dockerfile_best-practices/