

第一部分 Docker 入门

第一节 引言

1 虚拟化技术

整个虚拟化技术的发展有超过半个世纪的历史，当我看到这段历史的时候，不禁感叹老美在科技方面有多强悍。虚拟化一般分为硬件级虚拟化（hardware-level-virtualization）和操作系统级虚拟化（os-level-virtualization）。硬件级虚拟化是运行在硬件之上的虚拟化技术，它的管理软件也就是我们通常说的 hypervisor 或者 virtual machine monitor，它需要模拟的就是一个完整的操作系统，也就是我们通常所说的基于 Hyper-V 的虚拟化技术，VMWare, Xen, VirtualBox，亚马逊 AWS 和阿里云都是用的这种技术。操作系统级虚拟化是运行在操作系统之上的，它模拟的是运行在操作系统上的多个不同进程，并将其封装在一个密闭的容器里面，也称为容器化技术。Docker 正是容器虚拟化中目前最流行的一种实现。

1.1 硬件级虚拟化的历史

- 19 世纪 60 年代：美国出现了第一个虚拟化系统，它是由 IBM 开发的 CP-40 Mainframes 系统，虽然这个系统只是在实验室使用，但却为后来的 CP-67 系统奠定了基础。在那个时代，虚拟化系统主要由通用，贝尔实验室和 IBM 主导研发。
- 1987 年：一个非常牛逼的公司 Insignia Solutions 演示了一个称为 SoftPC 的软件模拟器，这个模拟器允许用户在 Unix Workstations 上运行 DOS 应用。在此之前这是不可能办到的，当时一个可以运行 MS DOS 的个人电脑需要 1,500 美金，而通过 SoftPC 模拟之后，可降低到 500 美金。可以看出，当时的需求就是在大型工作站上运行微软的 DOS。到了 1989 年的时候，Insignia Solutions 发布了 Mac 版的 SoftPC，使苹果用户不仅能运行 DOS，还能运行 Windows 操作系统。

- 1997 年：随着 SoftPC 的一炮而红，其他虚拟化公司如雨后春笋般的出现了。在 1997 年，苹果开发了 Virtual PC，后来又卖给了 Connectix。
- 1998 年：真正的王者 VMWare 出现了，他们在 1999 年开始销售 VMWare workstation，也就是我们很多人使用过得桌面版的虚拟机。
- 2001 年：VMWare 又发行了 ESX 和 GSX，也就是我们现在经常使用的 ESX-i 的前身。
- 2003 年：之前所说的 Connectix 被微软收购，后续推出了 Microsoft Virtual PC，再之后就没什么音讯了。同年 VMWare 也被 EMC 收购，成为 EMC 迄今最成功的一笔收购。就在这一年，一个开源的虚拟化项目 Xen 启动了，并在 2007 年被 Citrix 收购。

备注：看了这个历史，不禁内心发出感叹。Insignia Solutions 的衰败，Connectix 的没落，以及 VMWare 的半路杀出，都说明了商业和科技的竞争是不间断的，就像是一场长跑，一开始领先的，往往并不是最后的胜利者，你不进步，就肯定会被超越。

1.2 操作系统级虚拟化的历史

- 1982 年：你一定会很惊讶，第一个操作系统级的虚拟化技术是什么。答案就是 chroot，直到现在我们依然在使用的一个系统调用。这个系统调用会改变运行进程的工作目录，并且只能在这个目录里面工作。这种操作其实就是一种文件系统层的隔离。
- 2000 年：FreeBSD jail，真正意义上的第一个功能完整的操作系统级虚拟化技术。所以，真正的容器化技术出现到现在已经过去了 16 年，并不是几年的时间。
- 2005 年：OpenVZ，这是 linux 平台上的容器化技术实现，同时也是 LXC，即 docker 最初使用的容器技术核心实现。
- 2008 年：LXC 发布，这是 docker 最初使用的具体内核功能实现。
- 2013 年：Docker 发布，可以看出，docker 本身是使用了 LXC，同时封装了其他的一些功能。Docker 的成功，与其说是技术的创新，还不如说是一次组合式的创新。

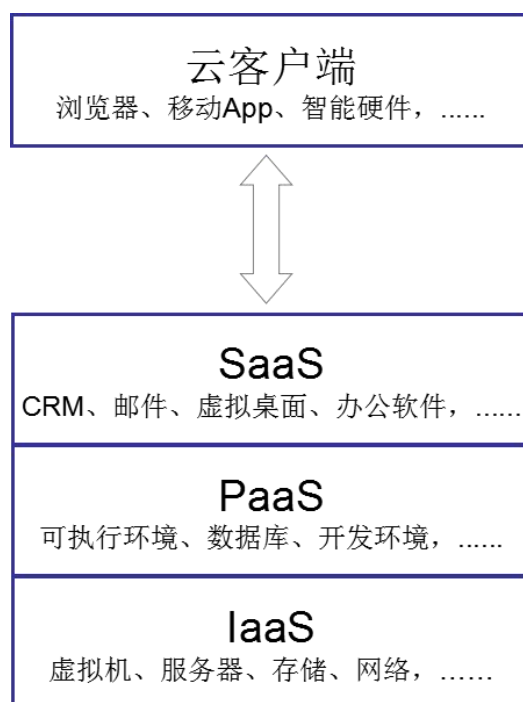
备注：曾经听一位老大说过，iPhone 你要说有多创新，真的说不上。手机很早就有了，电脑很早就有了，触摸屏很早就有了，但是苹果将所有这些有机的组合到了一起，再提供极致的用户体验，就产生跨时代的产品。同样 Docker 所使用的技术也都不是新技术，它将这一系列技术有机的组合到一起，并提供极致的用户体验，就产生

2 基于服务的云计算模式

我们知道传统的服务器或者电脑主机，基本都是一锤子买卖，商家卖给你之后基本就很难再从消费者身上获得其他收入。随着云的概念出现，越来越多的厂商都意识到卖硬件是不可能获得长期利益的，只有服务才是可持续的赢利点。因此，在 2010 年左右，出现了以大批提供云服务的公司。总体来说基本都可以归为下面几大类的一种或多种：

1. 基础设施即服务（Infrastructure as a service），通常指的是在云端为用户提供基础设施，如：虚拟机、服务器、存储、负载均衡、网络等等。亚马逊的 AWS 就是这个领域的佼佼者，国内则以阿里云为首。
2. 平台即服务（Platform as a service），通常指的是在云端为用户提供可执行环境、数据库、网站服务器、开发工具等等。国外的 OpenShift, Red Hat, Cloudera, Cloud Foundry, Google App Engine 都是这个领域的，当然还有一个非常有名的公司，那就是 dotCloud，后续我会再介绍一下这个公司。
3. 软件即服务（Software as a service），通常指的是在云端为用户提供软件，如 CRM 系统，邮件系统，在线协作，在线办公等等。比如微软就把自己的 Office 搬到了云端，国内的有道、麦客、Tower 都是属于这个领域的。

一般认为这三种模式，是最基本的云服务模式，其分层结构如下图：



3 Docker 介绍

Docker 主要解决什么问题？

Docker 对外宣称的是 Build, Ship and Run, Docker 要解决的核心问题就是快速的干这三种事情。它通过将运行环境和应用程序打包到一起，来解决部署的环境依赖问题，真正做到垮平台的分发和使用。而这一点和 DevOps 不谋而合，通过 Docker 可以大大提升开发、测试和运维的效率，在这个移动互联网的时代，工程师都是很贵的，如果一个工具能节省人力，提升效率，必定会火起来。

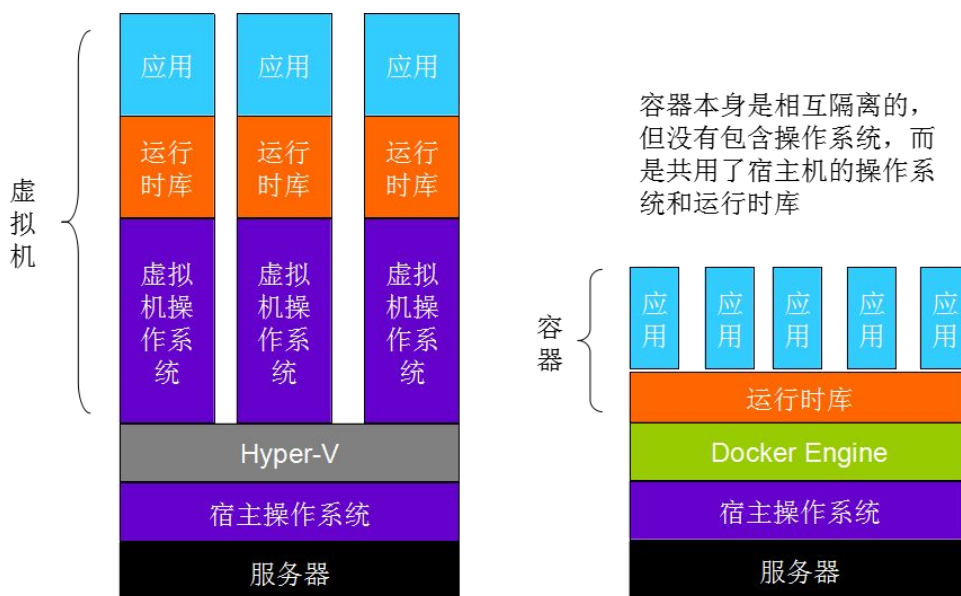
Docker 的历史

之前提到一家公司叫 dotCloud，这家公司是一家法国的公司，最初也是提供 PaaS 服务的，他们提供了对多种语言的运行环境支持，如：java, python, ruby, node.js 等。但是，可能叫生不逢时吧，在 PaaS 领域太多的巨头和大企业了，有一天 Solomon Hykes（Docker 之父）就召集了公司的哥儿几个来商量了一下，最后得出的结论是，如果要和那些大厂商硬干肯定是不行的，那么干脆就把他们做的项目 Docker 开源了。即使赚不到钱，至少也在开源社区得到一个好名声。因此，在 2013 年 3 月，Docker 正式以开源软件形式发布了。正是由于这次开源，让容器领域焕发了第二春，截止 2015 年 11 月，Docker 在 github 上收到了 25,600

个赞，超过 6800 次克隆，以及超过 1100 名的贡献者，成为 20 个最具影响力的 github 开源项目。可以说，Docker 是继 Linux 之后，最让人感到兴奋的系统层面的开源项目。据不完全统计，包括 Docker 公司、Red Hat, IBM, Google Cisco、亚马逊以及国内的华为等等，在为它贡献代码。在美国，几乎所有的云计算厂商都在拥抱 Docker 这个生态圈。

Docker 是什么呢？

Docker 其实是容器化技术的其中一种实现，根据我们之前的介绍，容器化技术并不是最近才出现的，那为什么 Docker 会如此的火爆呢？我觉得还是这个时代造就的，因为我们处在一个云计算发展异常迅猛的时代，而云计算又是所有移动互联网、IT、以及未来消费者行业的基础。从云计算服务的三层架构，我们可以看出传统的 IaaS 层，虚拟机是最基础组成部分，而虚拟机都是基于 Hyper-V 架构的，也就是说每一个虚拟机都会运行一个完整的操作系统，一个操作系统至少需要占用 5G 左右的磁盘空间，但是操作系统对于我们来说完全无用的，我们关心的是虚拟主机所能提供的服务。因此，我们迫切需要轻量级的主机，那就是 Docker 容器。我们可以看一下，Docker 的基本结构：



从上图，我们可以看到，容器由于省去了操作系统，整个层级更简化，可以在单台服务器上运行更多的应用，而这正是 IaaS 所需要的，可能 5G 左右的空间对你来说不是什么大事，但是如果你需要对外提供成千上万的主机，那就是不得

不考虑的问题，而这正是容器虚拟化要解决的问题。每到这里，我都喜欢举一个例子：波音公司造飞机肯定不会考虑在水上航行的问题，造船厂也绝对不会考虑要在天上飞的问题，汽车制造公司也不会考虑要在水上跑的问题，那么对于广大的移动互联网公司和云计算公司，也可以只关注最顶层的应用，而不需要去考虑操作系统的问题。

4 预备知识

这一节，主要介绍使用 Docker 需要了解的一些预备知识，主要包括 Linux 和公有云的使用，如果你对这两块已经比较熟悉，可以直接跳过进入下一节。为什么我们要介绍这两部分呢？因为 Docker 主要是运行在 Linux 下的，而且它的基础镜像也是基于 Linux，如果你了解 Linux 将会遇到很多障碍；而公有云的话是日后的一个趋势，也是 Docker 为什么这么火热的促成因素，所以我们也会做简单的介绍。

4.1 Linux 快速入门

Linux 是开源软件的鼻祖，Docker 目前基本上都依赖于 linux 的内核，这一节主要快速简单的介绍一下 linux。除去 Docker 的原因，我们还有以下原因要使用 linux：

- 本身开源免费，试想如果你是一个云主机提供商，如果每个主机都要付费，那得要给多少钱啊；
- 支持众多的开源软件，这也是很大一笔开支，并且可以灵活的自由配置；
- 基本上 90%以上的互联网公司都是用 linux 作为后端服务器，如果你想学习开发，那么 Linux 就是一项基本功；
- 云主机大多数都是基于 Linux 系统。

很多刚开始学习的朋友一听见 Linux 就会害怕，其实根本就没什么可怕的。所有的现代操作系统基本都是类似的，Linux 也有图形化界面，也支持鼠标键盘操作。所不同的是，Linux 下更倾向于用命令行来进行操作，这样有更多的灵活性，你可以认为就跟 windows 下的 cmd 类似。只要你愿意学习，顶多就 1 个月

就可以快速入手。

4.1.1 选取什么发行版本

Linux 包含了很多的发行版本, 包括 ubuntu, centos, redhat, fedora 等等, 但是他们都是基于 linux kernel, 各个发行版本都会做相应的包装、优化和简化, 但是基本上内核版本不会有太大的差异。根据我的经验, 我推荐使用 ubuntu 或者 centos。Ubuntu 的优点是:

- 内核更新及时
- 软件安装和更新方便
- GUI 简单实用 CentOS 就是 Red Hat Enterprise 的开源版本, 也是不错选择, 考虑到 Ubuntu 对 Docker 的完美支持, 我一般推荐使用 Ubuntu。Ubuntu 可以在 Docker 的存储部分使用 aufs, 而 CentOS 只能使用 DeviceMapper, 前者的性能要稍好一些。

4.1.2 使用图形界面还是命令行界面

Linux 的发行版本基本都提供了带图形界面和不带图形界面的版本, 也可以在安装后进行调整。很多人认为 Linux 系统就不适合带 IDE 界面的开发环境, 其实是错误的。我们可以在 Linux 下运行 Eclipse 等工具进行开发, 而且跟其他平台没有任何差异。所以如果你要进行开发, 可以选择图形界面。如果你是刚初学 Linux, 也可以使用图形界面, 来进行过度。

但是又回到我之前的飞机-轮船-汽车理论, 凡是没有必要的东西都应该去掉, 所以如果你是用 Linux 来运行后台程序, 最好就选择服务器版, 只运行命令行程序, 多使命令行也是一种学习和锻炼。

4.1.3 英文还是中文

一般来说使用 linux 的用户都是相对专业的用户, 我建议一律使用英文操作系统。英文的系统可以强迫你来学习英语, 同时系统上不会出现乱码。目前很多一手的技术资料基本都是英文的, 英文好是非常大的优势。

4.1.4 安装 Ubuntu 14.04

1. 下载镜像 <http://www.ubuntu.org.cn/download/server>
2. 安装

4.1.5 Linux 常用工具

如果你完全没有 Linux 基础，需要至少熟悉以下工具或者命令：

- ssh - 远程连接命令
- vim - 一个 linux 下命令行编辑文件的工具；
- ls - 列举文件及文件夹
- cp - 拷贝文件
- rm - 删除文件
- sudo - 以 root 用户执行命令
- cat - 查看文件
- pwd - 查看当前路径
- mkdir - 创建文件夹
- find - 查找文件
- grep - 搜索文件内容
- which - 查看命令在什么位置
- tar - 打包和压缩命令
- apt-get - Ubuntu 包管理工具

4.1.6 启用 root 用户

root 用户是 linux 的最高权限用户，相当于 windows 的超级管理员。我们可以通过下面的方式来启用 root 用户：

```
root@ghostcloud:~# sudo passwd
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
```

根据提示输入当前用户的密码，然后再输入 root 的密码。sudo 是以管理员

身份运行命令。然后通过 `su` 命令切换到 `root` 用户。

4.1.7 使用 vim

`vim` 是 `ubuntu` 默认的文本编辑器，学习使用 `linux` 第一步就是学会使用 `vi`。有的时候 `vim` 可能是没有安装的，我们需要手动来进行安装：

```
root@ghostcloud:~# sudo apt-get update && apt-get install vim
Hit http://mirrors.163.com precise Release.gpg
Hit http://mirrors.163.com precise Release
Hit http://apt.ghostcloud.cn ubuntu-precise Release.gpg
```

安装成功之后，我们就可以使用 `vim` 了。`vim` 是 `vi` 的升级版，有了很多优化。常用的命令有：

```
i - 从当前位置开始插入数据
a - 在当前位置后面插入数据
esc - 退出编辑模式
: - 在vim中执行一条指令，比如wq就是保存加退出
/ - 搜索文字
上下左右键 - 移动光标，vi里面不能用方向键，但是vim里面是可以使用的
```

虽然还有很多命令，但是用上面的基本就能操作了。

4.1.8 配置网络

ubuntu 的网络配置是放在 `/etc/network/interfaces` 下的，我们通过 `vim` 来进行查看和修改：

```
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).
# The loopback network interface
auto lo
iface lo inet loopback
# The primary network interface
auto eth0
iface eth0 inet static
address 192.168.1.10
netmask 255.255.0.0
gateway 192.168.0.1
dns-nameservers 61.139.2.69 218.6.200.139
```

修改完毕后，我们需要重启网络，一个比较好的方式是禁用再启用网络：

```
root@ghostcloud:~# ifdown -a && ifup -a
```

4.1.9 启用 SSH Server

SSH 是 Secure Shell 的缩写，是 linux 的标准远程连接工具，通过这个工具我们可以以命令行的方式远程连接到 linux 主机之上。首先我们需要检查在主机上是否安装了 ssh server：

```
root@ghostcloud:~# dpkg -l | grep openssh-server
ii openssh-server 1:6.6p1-2ubuntu2 amd64 Secure shell (SSH) server, for secure access
from remote machines
```

如果没有安装就不会有下面的输出。接下来，我们需要配置 ssh

```
# vim /etc/ssh/sshd_config
#允许Root 登录
PermitRootLogin yes
#允许通过密码进行验证登录
PasswordAuthentication yes
```

保存退出后，执行

```
root@ghostcloud:~# restart ssh
```

然后验证能否本地登录

```
root@ghostcloud:~# ssh root@localhost
The authenticity of host 'localhost (:::1)' can't be established.
```

```
ECDSA key fingerprint is 3a:8c:00:76:4d:4d:62:a7:c7:18:a0:00:e6:d0:17:c7.
Are you sure you want to continue connecting (yes/no)?
```

根据提示输入用户密码，如果可以登录说明安装成功，最后执行 `exit`，退出 `ssh` 连接。

4.1.10 通过客户端远程连接 linux 主机

SSH 是 Secure Shell 的缩写，是 linux 的标准远程连接工具，通过这个工具我们可以以命令行的方式远程连接到 linux 主机之上。首先我们需要检查在主机上是否安装了 `ssh server`

```
root@ghostcloud:~# dpkg -l | grep openssh-server
ii openssh-server 1:6.6p1-2ubuntu2 amd64 secure shell (SSH) server, for secure access
from remote machines
```

如果没有安装就不会有下面的输出。接下来，我们需要配置 `ssh`

```
# vim /etc/ssh/sshd_config
#允许Root 登录
PermitRootLogin yes
#允许通过密码进行验证登录
PasswordAuthentication yes
```

保存退出后，执行

```
root@ghostcloud:~# restart ssh
```

然后验证能否本地登录

```
root@ghostcloud:~# ssh root@localhost
The authenticity of host 'localhost (:::1)' can't be established.
ECDSA key fingerprint is 3a:8c:00:76:4d:4d:62:a7:c7:18:a0:00:e6:d0:17:c7.
Are you sure you want to continue connecting (yes/no)?
```

根据提示输入用户密码，如果可以登录说明安装成功，最后执行 `exit`，退出 `ssh` 连接。

4.1.11 免密码登录 linux 主机

免密码登录的原理是在你需要登录的远程主机上，存放当前机器的公钥。

1. 在当前机器生成公钥和私钥 `ssh-keygen`
2. 根据提示生成以后，会在 `~/.ssh/` 目录下生成相关的文件。这里的 `~` 指

的是用户的目录，比如，在 linux 下 abc 用户的目录为/home/abc，
root 用户的目录为/root，在 mac 下是在/Users/<用户名>

3. 将公钥 id_rsa.pub 拷贝到目标机器上 `scp ~/.ssh/id_rsa.pub root@192.168.1.10:~/` 这行命令将当前用户的公钥拷贝到远程机器的 root 用户目录下
4. `ssh root@192.168.1.10`
5. `ssh-keygen` #在远端产生密钥
6. `cat id_rsa.pub >> ~/.ssh/authorized_keys` #加入信任列表
7. `rm id_rsa.pub` #删除公钥
8. `exit` #退出远程机器 这时已经返回到当前机器，再执行 `ssh root@192.168.1.10`就不再需要输入密码了。

4.1.12 安装软件

Ubuntu 软件安装用的是和 debian 一样的系统——apt。apt 就是一个软件仓库，你只需要指定仓库地址，然后就可以进行搜索和安装。

- 添加源：默认的 ubuntu 源是指向国外的，位于/etc/apt/sources.list，我们可以在网上搜索国内的源，比如：网易源，阿里源等

```
deb http://mirrors.163.com/ubuntu/ trusty main restricted universe multiverse
deb http://mirrors.163.com/ubuntu/ trusty-security main restricted universe multiverse
deb http://mirrors.163.com/ubuntu/ trusty-updates main restricted universe multiverse
deb http://mirrors.163.com/ubuntu/ trusty-proposed main restricted universe multiverse
deb http://mirrors.163.com/ubuntu/ trusty-backports main restricted universe multiverse
deb-src http://mirrors.163.com/ubuntu/ trusty main restricted universe multiverse
deb-src http://mirrors.163.com/ubuntu/ trusty-security main restricted universe multiverse
deb-src http://mirrors.163.com/ubuntu/ trusty-updates main restricted universe multiverse
```

- 搜索软件

```
root@ghostcloud:/etc/apt# apt-cache search apache2 | grep ^apache2
apache2 - Apache HTTP Server
```

```
apache2-bin - Apache HTTP Server (binary files and modules)
apache2-data - Apache HTTP Server (common files)
apache2-dbg - Apache debugging symbols
apache2-dev - Apache HTTP Server (development headers)
apache2-doc - Apache HTTP Server (on-site documentation)
apache2-mpm-event - transitional event MPM package for apache2
apache2-mpm-prefork - transitional prefork MPM package for apache2
apache2-mpm-worker - transitional worker MPM package for apache2
apache2-utils - Apache HTTP Server (utility programs for web servers)
apache2.2-bin - Transitional package for apache2-bin
apache2-mpm-itk - transitional itk MPM package for apache2
apache2-suexec - transitional package for apache2-suexec-pristine
apache2-suexec-custom - Apache HTTP Server configurable suexec program for mod_suexec
apache2-suexec-pristine - Apache HTTP Server standard suexec program for mod_suexec
```

● 安装软件

```
root@ghostcloud:/etc/apt# apt-get install apache2
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
apache2-bin apache2-data libapr1 libaprutil1 libaprutil1-dbd-sqlite3
libaprutil1-ldap ssl-cert
Suggested packages:
apache2-doc apache2-suexec-pristine apache2-suexec-custom apache2-utils
openssl-blacklist
The following NEW packages will be installed:
apache2 apache2-bin apache2-data libapr1 libaprutil1 libaprutil1-dbd-sqlite3
libaprutil1-ldap ssl-cert
0 upgraded, 8 newly installed, 0 to remove and 72 not upgraded.
Need to get 1285 kB of archives.
After this operation, 5348 kB of additional disk space will be used.
Do you want to continue? [Y/n] y
```

● 卸载软件

```
root@ghostcloud:/etc/apt# apt-get purge apache2
```

● 查找本地安装的软件

```
root@ghostcloud:/etc/apt# dpkg -l
```

4.2 公有云主机快速入门

公有云主机，其实就是将主机托管到公网上，用户不需要去关心如何管理主

机，把这些事情交给专业的厂商。目前能提供公有云主机的厂商有很多，在国外 Amazon AWS, Microsoft Azure, Google App Engine，在国内有阿里云，UCloud，青云等，接下来我们就以国内最常用的阿里云 ECS 为例来讲一下，如何使用公有云主机。阿里云是国内目前最大的公有云提供商，它的产品线也比较广，不过最出名的还是 ECS 云主机。

4.2.1 购买云主机

1. 首先打开主页 <https://www.aliyun.com/>，注册用户之后就进入官方页面，选择产品->云服务器 ECS。



2. 服务器选型

云服务器 ECS 推荐购买

关联产品: 负载均衡 其他产品

包年包月 按量付费 购买云盘 购买须知

① 若 ECS 用于网站 Web 访问, 请及时备案。若 ECS 用于 SLB, 请前往 SLB 新购页面购买带宽, ECS 仅需保留少量带宽以便您管理。

购买清单 0 台

当前配置

地域: 深圳 (可用区随机分配)
规格: 1 核 1GB (简约型 t1)
镜像: -
存储: -
网络: 带宽 1Mbps (经典网络)
购买量: 1 个月 X 1 台
免费开通云盾服务
配置费用: **¥68.00**
立即购买 加入清单
实际扣费以账单为准 购买和计费说明 >>

地域

地域: 深圳 北京 青岛 杭州 香港 美国 (硅谷) 上海
亚太 (新加坡)
不同地域之间的产品内网不互通; 订购后不支持更换地域, 请谨慎选择 教我选择>> 查看我的产品地域

可用区: 随机分配 查看实例分布详情>> ②

安全组

安全组名称: 选择安全组
安全组类似防火墙功能, 用于设置网络访问控制, 您也可以到管理控制台 创建新安全组>> 教我选择>>

实例系列

实例系列: 系列 I ②
系列 I 采用 Intel Xeon CPU, DDR3 的内存。

I/O 优化

I/O 优化: ☐ I/O 优化实例 ②

实例规格

实例规格: 1 核 1GB (简约型 t1, ecs.t1.small)
请选择实例规格

由于页面比较长, 我在这里分两部分进行截取。其中,

- **地域:** 你购买的主机所在的区域, 每个区域内的主机都拥有一个公网 IP 和一个私网 IP。在同一个区域内, 私网 IP 是可以直接访问的, 一般是 10.*开头的 IP 地址。公网 IP 是可以在任何地方进行访问的, 同时还以作为将网站的域名指向该 IP。阿里云的双网结构, 给用户带来了很大的方便, 其中内网之间的速度是非常快的, 可以充分利用这一特性, 配合阿里云的其他相关产品使用, 例如: 对象存储。

- **可用区:** 可以理解为一个地域里面的不同接入点。
- **安全组:** 类似防火墙功能, 用于设置网络访问控制, 初期可以忽略。
- **实例系列:** 主机类型
- **IO 优化:** 可以加速的增值服务, 不过价格很贵, 不推荐使用。
- **实例规格:** 具体的主机配置

The screenshot displays the Alibaba Cloud ECS console configuration interface. It is divided into five main sections:
1. **网络 (Network)**: Options for '公网带宽' (Public Bandwidth) including '按固定带宽' (Fixed Bandwidth) and '按使用流量' (Pay-as-you-go). A bandwidth of 1 Mbps is selected.
2. **镜像 (Image)**: Options for '镜像类型' (Image Type) including '公共镜像' (Public Image), '自定义镜像' (Custom Image), '共享镜像' (Shared Image), and '镜像市场' (Image Market). '公共镜像' is selected.
3. **存储 (Storage)**: Options for '系统盘' (System Disk) including '普通云盘' (General Disk) and 'SSD云盘' (SSD Disk). '普通云盘' is selected with a size of 40 GB.
4. **密码 (Password)**: Options for '设置密码' (Set Password) including '立即设置' (Set Now) and '创建后设置' (Set After Creation). '立即设置' is selected.
5. **购买配置 (Purchase Configuration)**: Options for '购买时长' (Purchase Duration) including 1 year, 2 years, and 3 years. '1年' is selected. The quantity is set to 1.
On the right, the '当前配置' (Current Configuration) summary shows: 地域: 深圳 (可用区随机分配), 规格: 1核1GB (简约型 t1), 镜像: -, 存储: -, 网络: 带宽1Mbps (经典网络), 购买量: 1个月 X 1台. The price is ¥68.00. Buttons for '立即购买' (Buy Now) and '加入清单' (Add to Cart) are visible.

这一部分的配置有：

- 公网带宽：可以按固定带宽收费，也可以按流量收费。如果你的访问量比较大，最好选择固定带宽收费。
- 带宽：公网带宽数值，价格很贵，慎重选择。
- 镜像类型：包括公共镜像、自定义镜像、共享镜像和镜像市场。在镜像市场中，可以购买一些第三方服务商的镜像，直接使用。
- 系统盘：可以选择普通盘或者 SSD 盘。
- 密码：可以在创建的时候设置登录的 root 密码。
- 购买时长和数量：计费用。

服务器选型注意事项：

- 操作系统：一般来说，建议选择 linux，具有更好的灵活性，毕竟绝大多数云主机都是 linux 系统。
- CPU/内存：需要根据你的业务来合理配置，如果你是计算密集型（如：编解码），需要选择更好的 CPU 和内存；
- 磁盘大小：一般系统盘的 40G 对于普通应用足够了，如果是 IO 密集型，

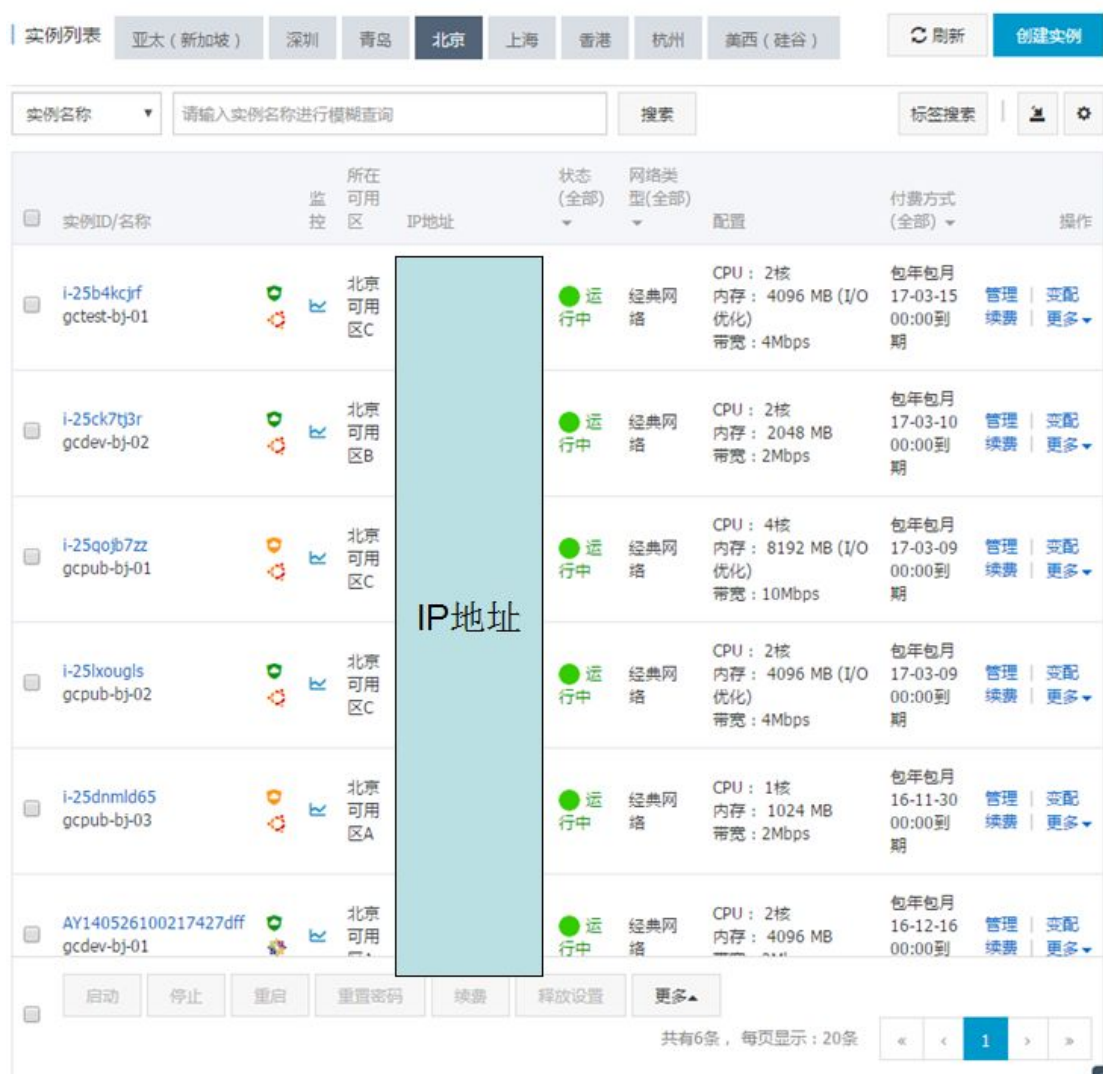
可以选择 SSD，如果需要存放大量文件或数据库数据，则需要外接硬盘。

- 网络带宽：网络是主机里面最贵的资源，必须合理的规划网络，只有确实需要大带宽时才购买，如果不对公网提供访问尽量使用内部带宽，既省钱又快速。

这里我选择 Ubuntu 14.04 作为我们的主机镜像，点击购买、支付后就可以在控制台中看到相关的主机信息。

4.2.1 连接到云主机

首先，点击位于顶部的“管理控制台”，我们可以看到现在你的主机列表：



实例ID/名称	所在可用区	IP地址	状态 (全部)	网络类型 (全部)	配置	付费方式 (全部)	操作
i-25b4kcjrf gctest-bj-01	北京可用区C		运行中	经典网络	CPU: 2核 内存: 4096 MB (I/O 优化) 带宽: 4Mbps	包年包月 17-03-15 00:00到期	管理 续费 更多
i-25ck7tj3r gcdev-bj-02	北京可用区B		运行中	经典网络	CPU: 2核 内存: 2048 MB 带宽: 2Mbps	包年包月 17-03-10 00:00到期	管理 续费 更多
i-25qojb7zz gcpub-bj-01	北京可用区C		运行中	经典网络	CPU: 4核 内存: 8192 MB (I/O 优化) 带宽: 10Mbps	包年包月 17-03-09 00:00到期	管理 续费 更多
i-25lxougls gcpub-bj-02	北京可用区C		运行中	经典网络	CPU: 2核 内存: 4096 MB (I/O 优化) 带宽: 4Mbps	包年包月 17-03-09 00:00到期	管理 续费 更多
i-25dnmld65 gcpub-bj-03	北京可用区A		运行中	经典网络	CPU: 1核 内存: 1024 MB 带宽: 2Mbps	包年包月 16-11-30 00:00到期	管理 续费 更多
AY140526100217427dff gcdev-bj-01	北京可用区A		运行中	经典网络	CPU: 2核 内存: 4096 MB 带宽: 4Mbps	包年包月 16-12-16 00:00到期	管理 续费 更多

刚购买的机器状态可能是启动中，上图所有主机都处于运行状态，在 IP 地

址一列我们可以看到公网 IP 和私网 IP，接下来我们将通过 SSH 进行远程连接。

```
welcome to Ubuntu 14.04.2 LTS (GNU/Linux 3.13.0-65-generic x86_64)

* Documentation:  https://help.ubuntu.com/

welcome to aliyun Elastic Compute Service!

Last login: Mon Mar 14 11:20:14 2016 from 182.148.56.118
root@iz25b4kcjrfz:~#
```

- 修改主机名:

```
# hostname ghostcloud
```

- 重新登录

```
welcome to Ubuntu 14.04.2 LTS (GNU/Linux 3.13.0-65-generic x86_64)

* Documentation:  https://help.ubuntu.com/

welcome to aliyun Elastic Compute Service!

Last login: Mon Mar 14 11:23:14 2016 from 182.148.56.118
root@ghostcloud:~#
```

- 获取 IP 地址

```
root@ghostcloud:~# ifconfig
eth0      Link encap:Ethernet  Hwaddr 00:16:3e:1c:01:3e
          inet addr:10.46.183.255 Bcast:10.46.183.255 Mask:255.255.248.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:701 errors:0 dropped:0 overruns:0 frame:0
          TX packets:43 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:51948 (51.9 KB)  TX bytes:6159 (6.1 KB)

eth1      Link encap:Ethernet  Hwaddr 00:16:3e:1c:0c:a7
          inet addr:101.201.31.255 Bcast:101.201.31.255 Mask:255.255.252.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:20394 errors:0 dropped:0 overruns:0 frame:0
          TX packets:558 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1500968 (1.5 MB)  TX bytes:80097 (80.0 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:648 errors:0 dropped:0 overruns:0 frame:0
          TX packets:648 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:51924 (51.9 KB)  TX bytes:51924 (51.9 KB)

root@ghostcloud:~#
```

- 修改密码（如果有需要的恶化，可以直接在主机内修改密码）

```
root@ghostcloud:~# passwd
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
root@ghostcloud:~# █
```

至此，就已经成功购买并登录到了云主机。

5 小结

在本节中，我们首先从虚拟化技术的历史开始，对 Docker 的历史也做了简单介绍，然后介绍了云计算的服务模式，最后对学习使用 Docker 需要用到的 Linux 和公有云主机的预备知识，做了介绍。

§ 实验

在阿里云上注册用户，并购买一台最便宜的主机，之后通过 SSH 连接到云主机上，并学习第 4 小节中的 Linux 常用命令。

第一节 Docker 的安装

Docker 公司维护了多个和 Docker 相关的项目，每一个项目的侧重点都不一样，我们这里所说的 Docker 安装指的是 Docker Engine 的安装，即 Docker 最核心的容器处理部分。Docker 原生态是只能运行在 Linux 系统下，目前几乎所有的 Linux 发行版本都可以运行 Docker，包括 Ubuntu，CentOS，RHEL，SuSE 等等。

1 在 linux 系统上安装 Docker

1.1 Ubuntu 下安装 Docker

Docker 官方支持以下的 Ubuntu 版本：

- Ubuntu Wily 15.10
- Ubuntu Trusty 14.04 (LTS)
- Ubuntu Precise 12.04 (LTS)

请注意，Ubuntu Utopic 14.10 和 15.04 并不是官方支持的版本，同时由于 12.04 太过陈旧，我建议是用 14.04 或者 15.10 版本。

1.1.1 前置条件

Docker 需要使用 Linux 中内核的 CGroups 和 Namespace 功能，因此必须使用包含这两项功能的 Linux 内核，所以内核必须是高于 3.10 的 64 位系统，可以通过 `uname -r` 查看当前的内核版本。

```
root@ghostcloud:~# uname -r
3.13.0-65-generic
```

1.1.2 更新 apt 源

Apt 是 Ubuntu 默认的包管理系统，apt 在安装的时候会根据你的 apt 配置文件去搜索安装源。一个系统可以包含多个不同的安装源，安装的时候 apt 会逐个进行搜索，Docker 官方的 apt 仓库只包含 Docker-engine 的安装源，对于其依赖

的包并不在内。所以在设置 Docker 源之前，我们需要针对国内的环境设置一下 apt 源，由于国内外网速的差距和 GFW 的原因，如果不设置安装时间会很长，同时可能安装失败。

1. 以 root 用户登录系统

2. 更新国内 ubuntu 源：在 /etc/apt/sources.list.d/ 目录下，新建一个 ghostcloud.list 文件（可以是任一文件名），然后加入一行：

```
deb http://mirrors.163.com/ubuntu/ trusty main
```

3. 将新加的文件对应的仓库信息加到 apt 系统中，并安装 https 的 CA 证书

```
root@ghostcloud:~# apt-get update
root@ghostcloud:~# apt-get install apt-transport-https ca-certificates
```

4. 添加 GPG key，这是访问 docker 源的公钥。

```
root@ghostcloud:~# apt-key adv --keyserver
hkp://p80.pool.sks-keyserver.net:80--recv-keys58118E89F3A912897C070ADB76221572C52609D
```

5. 打开/etc/apt/sources.list.d/ghostcloud.list，根据你的系统，加入 docker 安装源

➤ Ubuntu Trusty 14.04(LTS)

```
deb https://apt.dockerproject.org/repo ubuntu-trusty main
```

➤ Ubuntu wily 15.10

```
deb https://apt.dockerproject.org/repo ubuntu-wily main
```

6. 再次更新源

```
root@ghostcloud:~# apt-get update
```

7. 删除老的 lxc

```
root@ghostcloud:~# apt-purge lxc-docker
```

8. 检查 apt 是否从 docker 官方源安装

```
root@ghostcloud:~# apt-cache policy docker-engine
docker-engine:
  Installed: 1.10.3-0~trusty
  Candidate: 1.10.3-0~trusty
  Version table:
 *** 1.10.3-0~trusty 0
      500 https://apt.dockerproject.org/repo/ ubuntu-trusty/main amd64 Packages
      100 /var/lib/dpkg/status
 1.10.2-0~trusty 0
      500 https://apt.dockerproject.org/repo/ ubuntu-trusty/main amd64 Packages
 1.10.1-0~trusty 0
      500 https://apt.dockerproject.org/repo/ ubuntu-trusty/main amd64 Packages
 1.10.0-0~trusty 0
```

```

500 https://apt.dockerproject.org/repo/ ubuntu-trusty/main amd64 Packages
1.9.1-0~trusty 0
500 https://apt.dockerproject.org/repo/ ubuntu-trusty/main amd64 Packages
1.9.0-0~trusty 0
500 https://apt.dockerproject.org/repo/ ubuntu-trusty/main amd64 Packages
500 http://apt.ghostcloud.cn/repo/ ubuntu-trusty/main amd64 Packages
1.8.3-0~trusty 0
500 https://apt.dockerproject.org/repo/ ubuntu-trusty/main amd64 Packages
1.8.2-0~trusty 0
500 https://apt.dockerproject.org/repo/ ubuntu-trusty/main amd64 Packages
1.8.1-0~trusty 0
500 https://apt.dockerproject.org/repo/ ubuntu-trusty/main amd64 Packages
1.8.0-0~trusty 0
500 https://apt.dockerproject.org/repo/ ubuntu-trusty/main amd64 Packages
1.7.1-0~trusty 0
500 https://apt.dockerproject.org/repo/ ubuntu-trusty/main amd64 Packages
1.7.0-0~trusty 0
500 https://apt.dockerproject.org/repo/ ubuntu-trusty/main amd64 Packages
1.6.2-0~trusty 0
500 https://apt.dockerproject.org/repo/ ubuntu-trusty/main amd64 Packages
1.6.1-0~trusty 0
500 https://apt.dockerproject.org/repo/ ubuntu-trusty/main amd64 Packages
1.6.0-0~trusty 0
500 https://apt.dockerproject.org/repo/ ubuntu-trusty/main amd64 Packages
1.5.0-0~trusty 0
500 https://apt.dockerproject.org/repo/ ubuntu-trusty/main amd64 Packages

```

1.1.3 不同版本 Ubuntu 14.04 的特殊处理

对于 Ubuntu14.04，官方推荐安装 linux-image-extra 内核包，这个包允许用户使用 aufs，aufs 可以在某种程度上提升容器的 IO 性能。通过下面的语句进行安装：

```

root@ghostcloud:~# apt-get install linux-image-extra-$(uname -r)
Reading package lists... Done
Building dependency tree
Reading state information... Done
linux-image-extra-3.19.0-25-generic is already the newest version.
linux-image-extra-3.19.0-25-generic set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 126 not upgraded.

```

1.1.4 正式安装

同样以 root 用户登录系统，

1. 安装

```
root@ghostcloud:~# apt-get install docker-engine
```

2. 启动 docker daemon

```
root@ghostcloud:~# service docker start
```

3. 测试安装是否成功

```
root@ghostcloud:~# docker version

Client:

Version:      1.10.3

API version:  1.22

Go version:   go1.5.3

Git commit:   20f81dd

Built:        Thu Mar 10 15:54:52 2016

OS/Arch:      linux/amd64


Server:

Version:      1.10.3

API version:  1.22

Go version:   go1.5.3

Git commit:   20f81dd

Built:        Thu Mar 10 15:54:52 2016

OS/Arch:      linux/amd64
```

4. 运行第一个容器

```
root@ghostcloud:~# docker run hello-world

Unable to find image 'hello-world:latest' locally

latest: Pulling from library/hello-world

03f4658f8b78: Pull complete

a3ed95caeb02: Pull complete
```

```
Digest: sha256:8be990ef2aeb16dbcb9271ddfe2610fa6658d13f6dfb8bc72074cc1ca36966a7

Status: Downloaded newer image for hello-world:latest

Hello from Docker.

This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:

1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
3. The Docker daemon created a new container from that image which runs the
   executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it
   to your terminal.

To try something more ambitious, you can run an Ubuntu container with:

$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker Hub account:

https://hub.docker.com

For more examples and ideas, visit:

https://docs.docker.com/userguide/
```

如上图，我们启动了一个容器，然后打印出了一些语句，最后容器退出。

1.2 CentOS 下安装

Docker 可以运行在 CentOS 7.X 上，本小节讲述如何在 CentOS 上安装 Docker。

1.2.1 前置条件

同 Ubuntu 一样，依然需要内核版本高于 3.10 的 64 位系统，通过 `uname -r` 可以检查：

```
root@ghostcloud:~# uname -r
3.10.0-229.el7.x86_64
```

1.2.2 更新 yum

Yum 是 CentOS 的包管理工具，类似于 apt，我们还是以 root 用户登录系统，然后更新 yum 源：

```
root@ghostcloud:~# yum update
```

1.2.3 添加仓库

在 `/etc/yum.repos.d/ghostcloud.repo` 中加入

```
[dockerrepo]
name=Docker Repository
baseurl=https://yum.dockerproject.org/repo/main/centos/$releasever/
enabled=1
gpgcheck=1
gpgkey=https://yum.dockerproject.org/gpg
```

1.2.4 正式安装

1. 安装

```
root@ghostcloud:~# yum install docker-engine=
```

2. 启动 docker daemon

```
root@ghostcloud:~# service docker start
```

3. 测试安装是否成功

```
root@ghostcloud:~# docker version

Client:
  Version:      1.10.3
  API version:  1.22
  Go version:   go1.5.3
```

```
Git commit: 20f81dd

Built:      Thu Mar 10 15:54:52 2016

OS/Arch:    Linux/amd64


Server:


Version:     1.10.3

API version: 1.22

Go version:  go1.5.3

Git commit:  20f81dd

Built:      Thu Mar 10 15:54:52 2016

OS/Arch:    Linux/amd64
```

4. 运行第一个容器

```
root@ghostcloud:~# docker run hello-world

Unable to find image 'hello-world:latest' locally

latest: Pulling from library/hello-world

03f4658f8b78: Pull complete

a3ed95caeb02: Pull complete

Digest: sha256:8be990ef2aeb16dbcb9271ddfe2610fa6658d13f6dfb8bc72074cc1ca36966a7

Status: Downloaded newer image for hello-world:latest


Hello from Docker.

This message shows that your installation appears to be working correctly.


To generate this message, Docker took the following steps:

1. The Docker client contacted the Docker daemon.

2. The Docker daemon pulled the "hello-world" image from the Docker Hub.

3. The Docker daemon created a new container from that image which runs the
   executable that produces the output you are currently reading.

4. The Docker daemon streamed that output to the Docker client, which sent it
   to your terminal.
```

```
To try something more ambitious, you can run an Ubuntu container with:  
  
$ docker run -it ubuntu bash  
  
Share images, automate workflows, and more with a free Docker Hub account:  
  
https://hub.docker.com  
  
For more examples and ideas, visit:  
  
https://docs.docker.com/userguide/
```

如上图，我们启动了以个容器，然后打印出了一些语句，最后容器退出。

1.3 通过 ghostcloud 进行安装

上面的安装步骤其实是非常繁琐的，需要自己去更新源，而且由于国外网络的原因，有时候可能会失败。因此，国内出现了一些 PaaS 服务商将 docker 源在国内做了一个镜像，同时提供单个命令进行安装。目前 www.ghostcloud.cn 是安装速度最快的平台，这里我们选用它来进行安装：

1.3.1 注册 ghostcloud 账号

登录 www.ghostcloud.cn，进行用户注册，每个新用户账户上都有 50 元进行免费体验，点击进入“控制台”：



1.3.2 接入新主机



1.3.3 选择接入区域

1. 选择接入主机的类型

2. 配置主机基本信息

3. 等待主机接入

4. 完成确认

配置主机基本信息

请为您的新主机指定接入区域/接入服务器/描述等信息。

接入区域

北京大区

接入位置

北京服务器1

主机描述

第一台 docker 主机

上一步

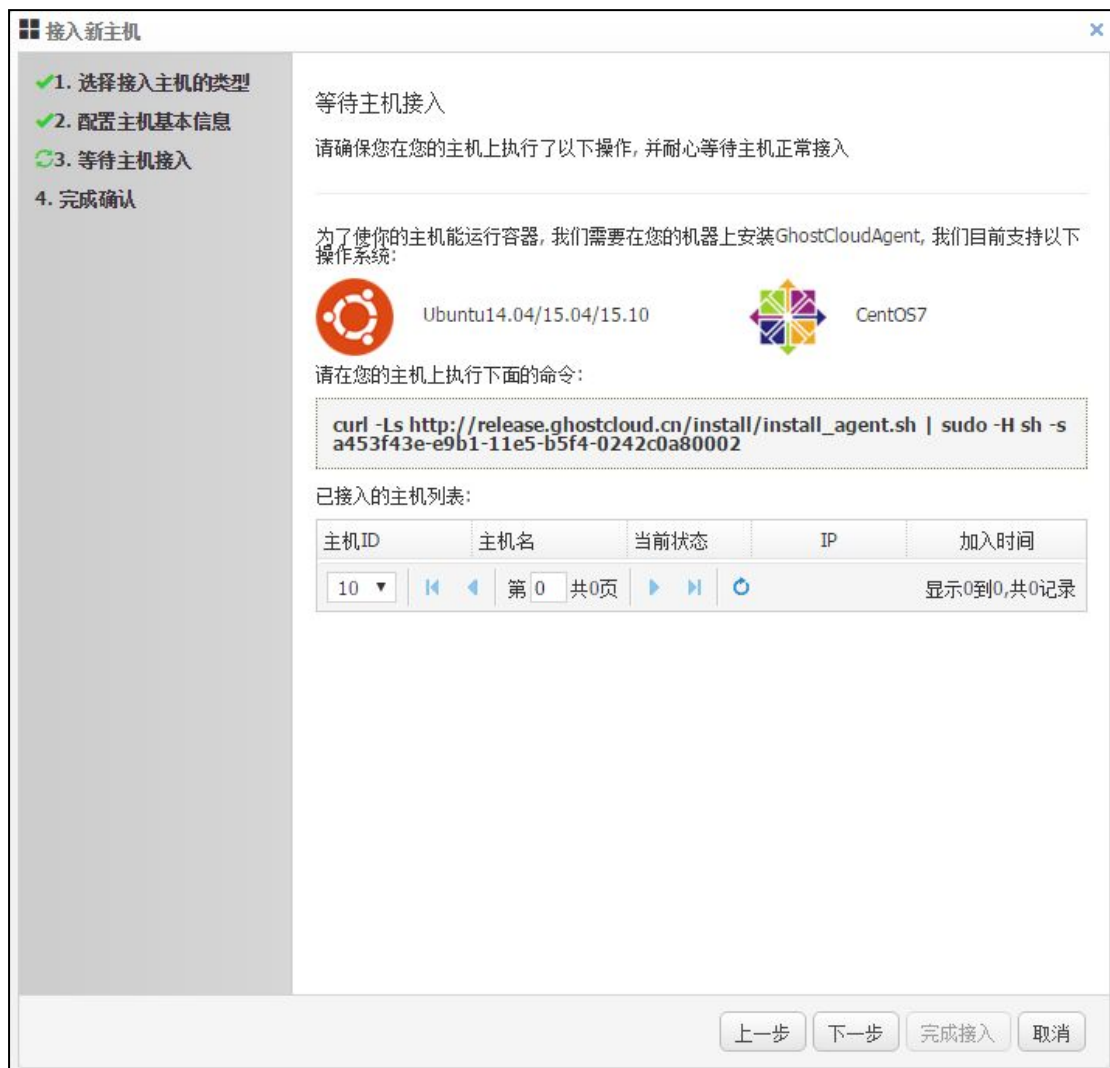
下一步

完成接入

取消

注意：ghostcloud 是一个混合云管理平台，在选择接入类型的时候，需要根据你的实际环境选择。如果你的机器是在内网中，使用“私有云”进行接入，如果你的机器是在公网中，使用“公有云”进行接入。

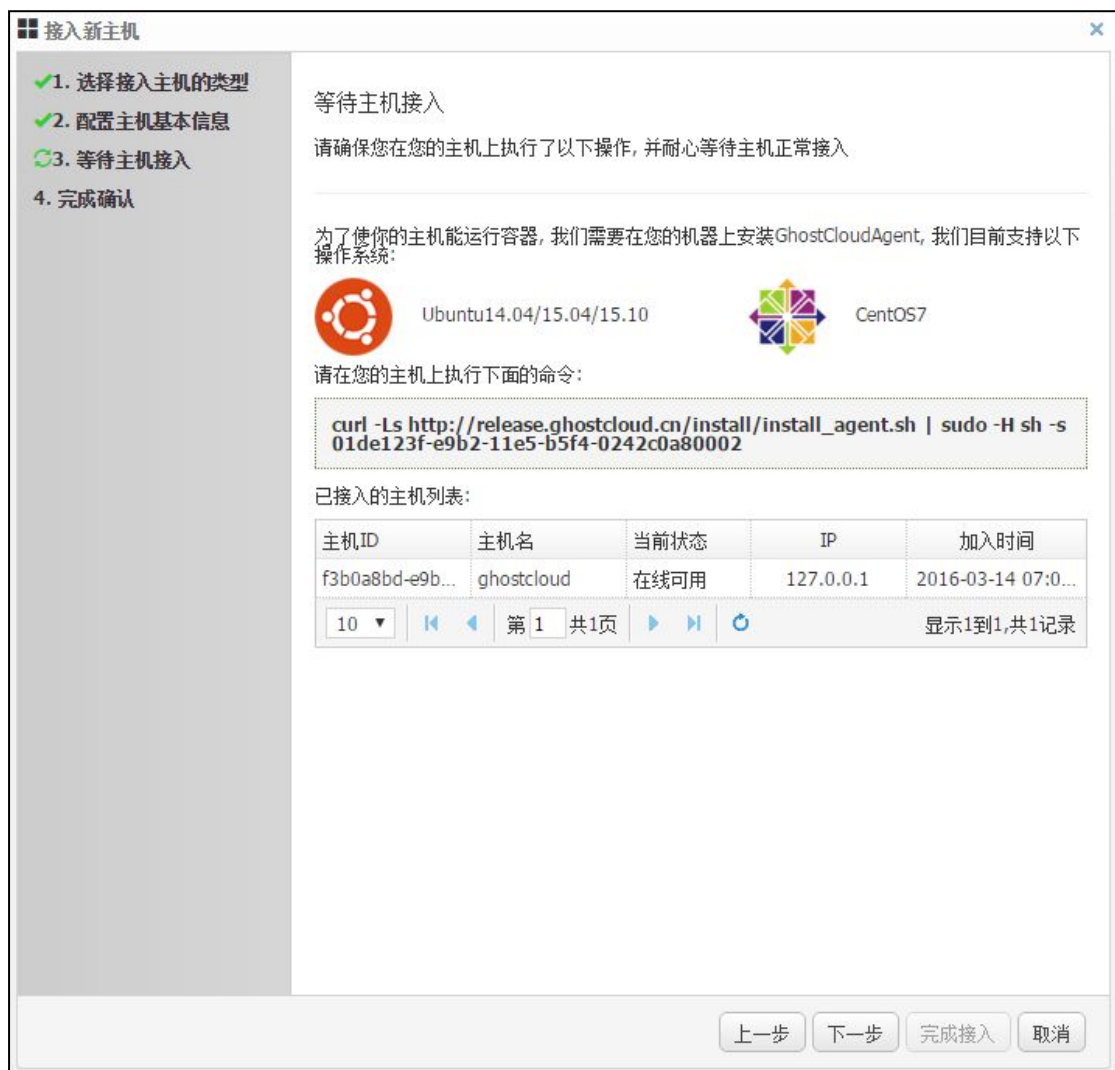
1.3.4 获取安装脚本



复制添加主机的命令，然后在你的主机上执行。

```
root@ghostcloud:~# curl -Ls http://release.ghostcloud.cn/install/install_agent.sh | sudo -H sh -s 01de123f-e9b2-11e5-b5f4-0242c0a80002d
```

安装成功后，ghostcloud 界面会有提示，同时接入到了统一管理平台



1.3.5 验证 docker 安装是否成功

```
root@ghostcloud:~# docker version

Client:
 Version:      1.9.0
 API version:  1.21
 Go version:   go1.4.2
 Git commit:   76d6bc9
 Built:        Tue Nov  3 17:43:42 UTC 2015
 OS/Arch:     Linux/amd64

Server:
 Version:      1.9.0
 API version:  1.21
 Go version:   go1.4.2
 Git commit:   76d6bc9
 Built:        Tue Nov  3 17:
```

1.3.6 运行第一个容器

```
root@ghostcloud:~# docker version
Client:
  Version:      1.9.0
  API version:  1.21
  Go version:   go1.4.2
  Git commit:   76d6bc9
  Built:        Tue Nov  3 17:43:42 UTC 2015
  OS/Arch:      Linux/amd64

Server:
  Version:      1.9.0
  API version:  1.21
  Go version:   go1.4.2
  Git commit:   76d6bc9
  Built:        Tue Nov  3 17:43:42 UTC 2015
  OS/Arch:      Linux/amd64

root@ghostcloud:~# docker run hello-world
Unable to find image 'hello-world:latest' locally
Pulling repository docker.io/library/hello-world
975b84d108f1: Pulling image (latest) from docker.io/library/hello-world, mirror:
http://mirror.975b84d108f1: Pulling image (latest) from docker.io/library/hello-world,
endpoint: https://regi975b84d108f1: Download complete
3f12c794407e: Download complete
Status: Downloaded newer image for hello-world:latest
docker.io/library/hello-world: this image was pulled from a legacy registry. Important:
This registry version will not be supported in future versions of docker.

Hello from Docker.

This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker Hub account:
```



```
https://hub.docker.com
```

For more examples and ideas, visit:

```
https://docs.docker.com/userguide/
```

通过 ghostcloud 安装一般只需要 20 秒，同时该公司提供了仓库的 mirror，可以大大提高 pull 镜像的速度。如果你不进行集群管理，你可以只使用命令行即可，如果你需要进行集群管理，可以研究一下他们的管理平台，在这儿就不做累述。

1.4 通过官方的安装脚本安装

官方还有一个通用安装脚本：

```
# curl -fsSL https://get.docker.com/ | sh
```

这行命令就是下载一个脚本，然后执行。然后，由于网络原因，中途很可能会失败，一切取决于你的人品，如下所示：

```
root@ghostcloud:~# curl -fsSL https://get.docker.com/ | sh
apparmor is enabled in the kernel and apparmor utils were already installed
+ sh -c apt-key adv --keyserver hkp://p80.pool.sks-keyservers.net:80 --recv-keys
58118E89F3A912897C070ADB76221572C52609D
Executing: gpg --ignore-time-conflict --no-options --no-default-keyring --homedir
/tmp/tmp.cr8hxY45Ve --no-auto-check-trustdb --trust-model always --keyring
/etc/apt/trusted.gpg --primary-keyring /etc/apt/trusted.gpg --keyserver
hkp://p80.pool.sks-keyservers.net:80 --recv-keys 58118E89F3A912897C070ADB76221572C52609D
gpg: requesting key 2C52609D from hkp server p80.pool.sks-keyservers.net
?: p80.pool.sks-keyservers.net: Host not found
gpgkeys: HTTP fetch error 7: couldn't connect: Success
gpg: no valid OpenPGP data found.
gpg: Total number processed: 0
```

上面就是一个安装失败的例子。

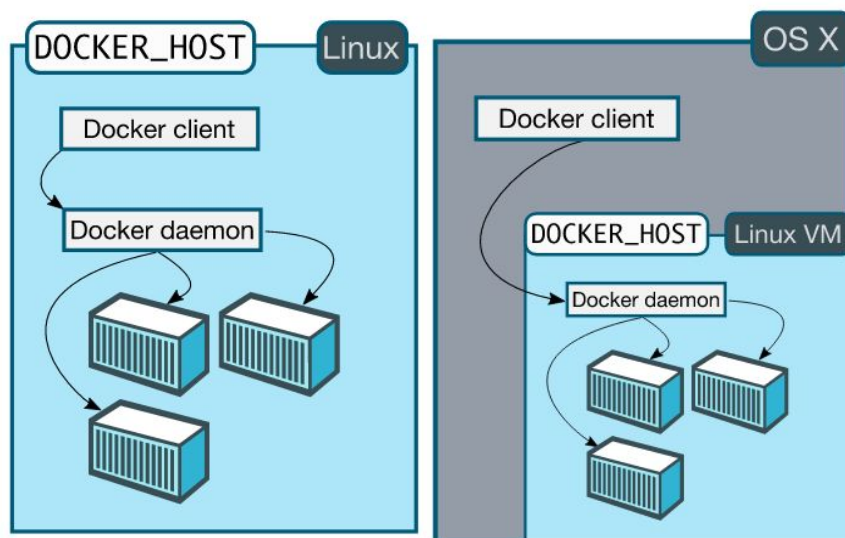
2 在非 linux 系统下安装 docker

由于 Docker 都需要 linux 内核的支持，如果要在非 linux 环境下安装 docker 基本上都需要通过虚拟机的形式来进行，其原理就是在非 linux 系统上安装一个客户端，然后连接到虚拟机里面去操作，所以这种方式其实并不是原生态的安装。Docker-machine 就是专门为了解决这种问题而生的，目前 Mac OS X 和 Windows

操作系统下，都是可以安装 Docker-machine 的，而且 docker 将其封装的非常易用，基本上你感知不到和原生态的差别，不过这只能运用在测试和开发环境中。

尽管 Docker-machine 提供了跨平台的非 linux 系统支持，但是其他操作系统也是可以支持 docker 的，目前 windows server 2016 已经提供了对 docker 的全套支持，有兴趣的朋友可以在网上搜索相关的视频教程。

下面这幅图是原生态的 docker 和使用 docker-machine 的结构示意图：



如果你需要使用 docker-machine，可以在 <https://www.docker.com/toolbox> 进行下载安装。

§ 实验

1. 在你的内网中，安装一个能运行 docker 的操作系统，然后安装 docker。
2. 在公有云中，运行一个能够运行 docker 的操作系统镜像，然后安装 do
3. 在 windows 环境下，通过 docker-machine，安装 docker。

第二节 使用 Docker

曾经有个笑话是，有一个程序员某天决定练书法，他拿起毛笔，精神抖擞的书写了两个大字——Hello World。因此，我们如何使用 docker 也是从 Hello World 开始。

1 回到 hello-world

在第一节中，我们执行了一个命令：

```
root@ghostcloud:~# docker run hello-world
```

Hello from Docker.

This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:

- 1. The Docker client contacted the Docker daemon.*
- 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.*
- 3. The Docker daemon created a new container from that image which runs the executable that produces the output you are currently reading.*
- 4. The Docker daemon streamed that output to the Docker client, which sent it to your terminal.*

To try something more ambitious, you can run an Ubuntu container with:

```
$ docker run -it ubuntu bash
```

Share images, automate workflows, and more with a free Docker Hub account:

<https://hub.docker.com>

For more examples and ideas, visit:

<https://docs.docker.com/userguide/>

这个命令包括了三部分：

- docker - docker 的客户端程序
- run - 子命令，用于运行容器
- hello-world - 镜像的名称

这条命令首先会在本地查找是否有 hello-world 这个镜像，如果没有，会自动从 dockerhub（docker 主仓库）去拉取，之后会启动一个容器并把 image 装载进

容器中运行。所有 `docker` 的命令格式都是下面的格式：

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

其中 `OPTIONS` 是运行的参数, `COMMAND` 是真正的子命令, `arg` 是该条子命令对应的参数集合。

下面我们通过 `docker images` 来看一下系统中都有哪些镜像：

```
root@ghostcloud:~# docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	VIRTUAL SIZE
centos	latest	eb98cdc284d4	9 days ago	196.6 MB
ubuntu	latest	56063ad57855	10 days ago	188 MB
hello-world	latest	975b84d108f1	5 months ago	960 B

在该系统存在三个镜像，分别是 `centos`, `ubuntu`, `hello-world`。每一列的具体意思是什么呢？

- `REPOSITORY` - 这是镜像的名称，但是官方取得名字感觉有点奇怪。
- `TAG` - 这是镜像的版本，类似于 `git` 的 `tag`，默认如果你 `run` 的时候不带版本就会拉取最新的版本 `latest`
- `IMAGE ID` - 这是镜像的截短后的 ID
- `CREATED` - 镜像是在什么时候创建的，这个时间是指的仓库里面的时间
- `VIRTUAL SIZE` - 这是这个镜像所占空间的大小

2 容器和镜像

2.1 什么是容器？

根据我们之前的介绍，容器其实就是运行在操作系统上的一个进程，但是加入了对资源的隔离和限制。以前我们在运行一个进程的时候，如果里面出现死循环，CPU 就会一下被占用完；如果出现内存泄露或者大内存分配，就可能会把系统的内存吃完。因此，linux 里面出现 `CGroup` 技术来限定一个进程的资源使用。在一个操作系统之上，用户 ID，机器名等资源是全局的，运行的进程间都是访问的同一份资源，为了达到隔离的目的，linux 又出现了 `Namespace` 技术来划分不同的命名空间。

所以，容器就是拥有受限资源和单独命名空间的进程。

注: CGroup 和 Namespace 是 docker 的最核心组成部分, 有兴趣的朋友可以自行在网上搜索详细的教程, 在这儿我就不再深入。

2.2 什么是镜像?

除了 CGroup 和 Namespace 技术, 进程运行的时候它的文件系统也是全局的, 那么如何来做限定呢? 在很早之前, chroot 可以用来处理, 不过它只能做简单的隔离, 而且存在安全隐患。因此, Docker 就设计了一个 Layered FS, 它把文件系统分成多个层, 使多个容器间可以使用公共的部分。而镜像就是由 Layered FS 组成的, 并且它是只读的。当容器运行时, 会在镜像之上再加上一层可读可写层。

所以, 镜像就是容器的文件系统。如果说容器并没有什么创新, 那么 Layered FS 绝对是一个了不起的创新组合。

2.3 容器和镜像的相互转化

我们在使用 docker 的时候, 首先需要拉取镜像, 然后再通过一个容器来运行这个镜像。通过镜像是可以生成容器的。同时, 当容器运行到一个时间的时候, 我们也可以通过 docker commit, 将容器的可读可写层转化为镜像。因此, 镜像和容器是相生相伴的, 缺一不可的。

3 Docker 入门操作

本小节将通过一个实验来更进一步了解如何使用 Docker。

1. 首先我们查看一下 docker 的基本信息

```
root@ghostcloud:~# docker info
Containers: 6
Images: 15
Server Version: 1.9.0
Storage Driver: aufs
Root Dir: /var/lib/docker/aufs
Backing Filesystem: extfs
Dirs: 31
Dirperm1 Supported: true
Execution Driver: native-0.2
Logging Driver: json-file
```

```

Kernel Version: 3.19.0-25-generic
Operating System: Ubuntu 14.04.3 LTS
CPUs: 2
Total Memory: 3.844 GiB
Name: ghostcloud
ID: I37F:V3WA:R5OW:J7CU:3YPU:MLYI:AKVN:6EAG:CKLE:H6NS:MQCQ:7WCW
WARNING: No swap limit support
root@ghostcloud:~#

```

`docker info` 是整个 `docker daemon` 运行状况的缩影，包括了容器个数，镜像个数，`daemon` 版本，使用的存储驱动等等信息。如果安装成功，该命令都会执行成功。如果提示

```
Cannot connect to the Docker daemon. Is the docker daemon running on this host?
```

很可能是 `Daemon` 没有启动，或者安装没有成功。

2. 下载一个真正以上的基础镜像

```

root@ghostcloud:~# docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu

073de23ee32b: Downloading 15.28 MB/65.69 MB
073de23ee32b: Pull complete
9d89fd8f8a3e: Pull complete
0b427fcc4cbb: Pull complete
8ed581e3fa7a: Pull complete
Digest: sha256:4e85ebe01d056b43955250bbac22bdb8734271122e3c78d21e55ee235fc6802d
Status: Downloaded newer image for ubuntu:latest

```

`docker pull` 会默认从 `docker hub` 拉取镜像，如果本地已经有了就直接结束，否则的话会根据文件系统分层逐个下载。在上面的下载过程中，就下载了 4 个文件系统分层。

3. 运行一个含 shell 的容器

之前我们运行的容器都是运行完成后就退出了，那么可不可以运行一个可交互的容器呢，就像一个虚拟机一样。答案是可以的。

```

root@ghostcloud:~# docker run -i -t ubuntu /bin/bash
root@ee87f6127814:/# ls
bin dev home lib64 mnt proc run srv tmp var
boot etc lib media opt root sbin sys usr
root@ee87f6127814:/# pwd
/
root@ee87f6127814:/# touch 1.txt
root@ee87f6127814:/# ls

```

```
1.txt  boot  etc   lib   media  opt   root /sbin  sys  usr
bin    dev   home  lib64 mnt    proc  run   srv   tmp  var
```

看见了吗，就是这么简单，一键就拥有了一个 `ubuntu` 系统，这就是容器的魅力。从现在开始，我们将带你正式进入容器的世界。首先来剖析一下运行的参数：

- `-i` 表示启动一个可交互的容器
- `-t` 表示使用 `pseudo-TTY`，关联到容器的 `stdin` 和 `stdout`
- `ubuntu` - 表示我们要运行的镜像
- `/bin/bash` - 表示我们启动容器的时候，要运行的命令

4. 开启另一个终端，查看运行的容器

我们保持 ubuntu 的 shell，再开启一个终端查看容器运行状态：

```
root@ghostcloud:~# docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED        STATUS        PORTS        NAMES
ee87f6127814   ubuntu    "/bin/bash"            6 minutes ago Up 6 minutes          pensive_booth
root@ghostcloud:~#
```

5. 接下来我们运行一个长时间运行的程序

```

root@ghostcloud:~# docker run -d ubuntu /bin/sh -c "while true; do echo Hello world; sleep 1; d
one"
be9c863ca0e39548e44a062aaead3b7894f135fbc05e3109d055efeabd3bccba
root@ghostcloud:~#
root@ghostcloud:~# docker logs be9c
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
root@ghostcloud:~# docker ps
CONTAINER ID        COMMAND               CREATED            STATUS
PORTS              IMAGE                NAMES
be9c863ca0e3       ubuntu               "/bin/sh -c 'while tr
nd                    trusting_franklin    24 seconds ago    Up 23 seco
ee87f6127814       ubuntu               "/bin/bash"        9 minutes ago     Up 9 minut
es                   pensive_booth
root@ghostcloud:~#

```

这条事例中，我们让容器去不停循环打印 hello world，剖析：

- `-d` 让容器在后台运行，不关联到当前终端的 shell 上
- `docker logs <container_id>` 查看容器的日志，其实就是查看容器的标准输出日志。
- 为了避免产生过多的日志，我们可以通过 `docker kill <container_id>` 来将其杀掉。

6. 查看容器

我们可以通过 `docker ps` 来查看容器，默认只会列出运行中的容器，如果要查看所有的容器需要加 `-a` 参数：

```
root@ghostcloud:~# docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS
ee87f6127814   ubuntu   "/bin/bash"             13 minutes ago Up 13 minutes
root@ghostcloud:~# docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS
be9c863ca0e3   ubuntu   "/bin/sh -c 'while tr"   4 minutes ago   Exited (13
7) About a minute ago
ee87f6127814   ubuntu   "/bin/bash"             13 minutes ago Up 13 minu
tes
db2bf300f148   hello-world "/hello"                About an hour ago Exited (0)
About an hour ago
7316d8dc8377   hello-world "/hello"                2 hours ago      Exited (0)
274edbe026b3   centos     "/bin/bash"             3 hours ago      Exited (0)
e314f0a8ec31   56063ad57855 "/bin/bash"             3 hours ago      Exited (0)
e5ca6bd8b515   hello-world "/hello"                3 hours ago      Exited (0)
5ed09e80015b   56063ad57855 "/bin/bash"             2 days ago       Exited (10
0) 6 hours ago
root@ghostcloud:~#
```

7. 容器的一系列操作，包括启动、停止、重启、强行停止和删除

```
root@ghostcloud:~# docker start be9c863ca0e3
be9c863ca0e3
root@ghostcloud:~# docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS
be9c863ca0e3   ubuntu   "/bin/sh -c 'while tr"   7 minutes ago   Up 3 second
ds
ee87f6127814   ubuntu   "/bin/bash"             17 minutes ago Up 17 minu
tes
root@ghostcloud:~# docker stop be9c863ca0e3

be9c863ca0e3
root@ghostcloud:~#
root@ghostcloud:~#
root@ghostcloud:~# docker restart be9c863ca0e3
be9c863ca0e3
root@ghostcloud:~# docker kill be9c863ca0e3
No command 'doker' found, did you mean:
Command 'docker' from package 'docker' (universe)
doker: command not found
root@ghostcloud:~# docker kill be9c863ca0e3
be9c863ca0e3
root@ghostcloud:~# docker rm be9c863ca0e3
be9c863ca0e3
```

8. 容器导出服务

如果容器只是能运行程序，没有网络无疑就是一个密闭的盒子，没有生气。

Docker 同 VM 一样，也可以配置相应的网络。容器通常支持五种网络模式：

- None - 在该模式下容器没有对外网络，本机只有一个回路地址。
- Container - 与另一个容器共享网络。
- Host - 与主机共享网络

- Bridge - Docker 默认的网络模式，在这种模式下，Docker 容器与外部的通信都是通过 iptable 来实现的。

- Overlay - Docker 目前原生的跨主机多子网模型，主要是通过 vxlan 技术来实现的

我们先使用默认的桥接模式来进行服务导出：

1) 在容器中启动 nc 监听在 4444 端口（nc 是一个简单的监听器，可以接受用户的输入）

```
root@ghostcloud:~# JOB=$(docker run -d -p 4444 ubuntu /bin/nc -l 4444)
```

2) 查看容器导出的端口

```
root@ghostcloud:~# docker port $JOB
4444/tcp -> 0.0.0.0:32768
```

3) 向主机发送 hello world

```
root@ghostcloud:~# echo hello world | nc 127.0.0.1 32768
```

4) 确认主机是否收到消息

```
root@ghostcloud:~# docker logs $JOB
hello world
```

剖析：

- nc - 一个简单的网络程序，监听在指定端口，并打印收到的数据
- -p - 表示容器内部得端口，由于没有指定主机端口，主机端口是随机分配的，默认是从 32768 开始

- 外部程序端口 - 通过 docker port 可以查看容器的端口映射关系

9. 提交容器到镜像

之前我们提到，容器是可以转化为镜像的，容器转化为镜像的过程是将可读写层变成只读层，具体操作如下：

```
root@ghostcloud:~# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
PORTS              NAMES
ee87f6127814       ubuntu             "/bin/bash"        About an hour ago   Up About an
hour               pensive_booth
root@ghostcloud:~# docker commit ee87 myubuntu
b1616061a25fa6c00b67e2f5fada19c1a23e885f6ab330d6d0070a030481ffcc
root@ghostcloud:~# docker images
REPOSITORY          TAG               IMAGE ID            CREATED             VIRTUAL SIZE
myubuntu            latest           b1616061a25f       3 seconds ago      188 MB
```

这里再强调一下，容器运行中并不会修改镜像的内容，所以如果需要对容器

做持久化，就必须通过 `commit` 来进行，否则容器退出后数据就会丢失。

4 容器的基础命令

之前我们已经接触了一些 `docker` 的命令，主要包括：

- `docker pull` - 拉取镜像
- `docker run` - 运行容器
- `docker images` - 列出所有镜像
- `docker ps` - 查看容器
- `docker stop/start/restart/kill` - 容器的启停
- `docker logs` - 查看容器的日志

接下来，我们要对 `docker` 命令的运行机制进行讲解。

4.1 docker client

Docker-engine 在主机上是 C/S 架构，其中 C 指的是 `docker client`，S 指的是 `docker daemon`，但是这二者使用的是同一个程序：

```
root@ghostcloud:~# which docker
/usr/bin/docker
```

我们可以通过 `docker --help` 来查看 `docker` 的所有命令及选项：

```
root@ghostcloud:~# docker --help
Usage: docker [OPTIONS] COMMAND [arg...]

    docker daemon [ --help | ... ]
    docker [ --help | -v | --version ]

A self-sufficient runtime for containers.

Options:

--config=~/.docker           Location of client config files
-D, --debug=false            Enable debug mode
--disable-legacy-registry=false Do not contact legacy registries
-H, --host=[]                Daemon socket(s) to connect to
-h, --help=false             Print usage
-l, --log-level=info          Set the logging level
--tls=false                  Use TLS; implied by --tlsverify
```

```
--tlscacert=~/.docker/ca.pem      Trust certs signed only by this CA
--tlscert=~/.docker/cert.pem      Path to TLS certificate file
--tlskey=~/.docker/key.pem        Path to TLS key file
--tlsverify=false                 Use TLS and verify the remote
-v, --version=false               Print version information and quit
```

Commands:

```
attach  Attach to a running container
build   Build an image from a Dockerfile
commit  Create a new image from a container's changes
cp      Copy files/folders between a container and the local filesystem
create  Create a new container
diff    Inspect changes on a container's filesystem
events  Get real time events from the server
exec    Run a command in a running container
export  Export a container's filesystem as a tar archive
history Show the history of an image
images  List images
import  Import the contents from a tarball to create a filesystem image
info    Display system-wide information
inspect Return low-level information on a container or image
kill    Kill a running container
load    Load an image from a tar archive or STDIN
login   Register or log in to a Docker registry
logout  Log out from a Docker registry
logs    Fetch the logs of a container
network Manage Docker networks
pause   Pause all processes within a container
port    List port mappings or a specific mapping for the CONTAINER
ps      List containers
pull    Pull an image or a repository from a registry
push    Push an image or a repository to a registry
rename  Rename a container
restart Restart a container
rm      Remove one or more containers
rmi     Remove one or more images
run     Run a command in a new container
save    Save an image(s) to a tar archive
search  Search the Docker Hub for images
start   Start one or more stopped containers
stats   Display a live stream of container(s) resource usage statistics
stop    Stop a running container
tag     Tag an image into a repository
top     Display the running processes of a container
```

```

unpause    Unpause all processes within a container
version    Show the Docker version information
volume     Manage Docker volumes
wait       Block until a container stops, then print its exit code

```

Run 'docker COMMAND --help' for more information on a command.

我们先看一下头几行：

```

Usage: docker [OPTIONS] COMMAND [arg...]
       docker daemon [ --help | ... ]
       docker [ --help | -v | --version ]

```

- 第一行是 docker client 的用法, 我们还可以通过 docker COMMAND --help 来查看每个命令的详细用法
- 第二行是 docker daemon 的用法, 我们可以通过 docker daemon --help 来查看

```
root@ghostcloud:~# docker daemon --help
```

```
Usage: docker daemon [OPTIONS]
```

Enable daemon mode

```

--api-cors-header=          Set CORS headers in the remote API
-b, --bridge=               Attach containers to a network bridge
--bip=                      Specify network bridge IP
--cluster-advertise=        Address or interface name to advertise
--cluster-store=            Set the cluster store
--cluster-store-opt=map[]   Set cluster store options
-D, --debug=false           Enable debug mode
--default-gateway=          Container default gateway IPv4 address
--default-gateway-v6=       Container default gateway IPv6 address
--default-ulimit=[]         Set default ulimits for containers
--disable-legacy-registry=false Do not contact legacy registries
--dns=[]                    DNS server to use
--dns-opt=[]                DNS options to use
--dns-search=[]             DNS search domains to use
-e, --exec-driver=native    Exec driver to use
--exec-opt=[]               Set exec driver options
--exec-root=/var/run/docker Root of the Docker execdriver
--fixed-cidr=               IPv4 subnet for fixed IPs
--fixed-cidr-v6=            IPv6 subnet for fixed IPs
-G, --group=docker          Group for the unix socket
-g, --graph=/var/lib/docker Root of the Docker runtime
-H, --host=[]               Daemon socket(s) to connect to

```

```

--help=false          Print usage
--icc=true            Enable inter-container communication
--insecure-registry=[] Enable insecure registry communication
--ip=0.0.0.0          Default IP when binding container ports
--ip-forward=true     Enable net.ipv4.ip_forward
--ip-masq=true        Enable IP masquerading
--iptables=true       Enable addition of iptables rules
--ipv6=false          Enable IPv6 networking
-l, --log-level=info   Set the logging level
--label=[]            Set key=value labels to the daemon
--log-driver=json-file Default driver for container logs
--log-opt=map[]       Set log driver options
--mtu=0               Set the containers network MTU
-p, --pidfile=/var/run/docker.pid Path to use for daemon PID file
--registry-mirror=[] Preferred Docker registry mirror
-s, --storage-driver= Storage driver to use
--selinux-enabled=false Enable selinux support
--storage-opt=[]      Set storage driver options
--tls=false           Use TLS; implied by --tlsverify
--tlscacert=~/.docker/ca.pem Trust certs signed only by this CA
--tlscert=~/.docker/cert.pem Path to TLS certificate file
--tlskey=~/.docker/key.pem Path to TLS key file
--tlsverify=false     Use TLS and verify the remote

```

这里面包含了很多 docker 可以配置的选项，我们可以停止服务，直接用程序方式带参数运行，也可以修改服务的配置文件/etc/default/docker 来加入默认配置。如果要查看 client 和 daemon 的信息，我们可以使用 docker version 来查看：

```

root@ghostcloud:~# docker version

Client:
 Version:      1.9.0
 API version:  1.21
 Go version:   go1.4.2
 Git commit:   76d6bc9
 Built:        Tue Nov  3 17:43:42 UTC 2015
 OS/Arch:      Linux/amd64

Server:
 Version:      1.9.0
 API version:  1.21
 Go version:   go1.4.2
 Git commit:   76d6bc9
 Built:        Tue Nov  3 17:43:42 UTC 2015
 OS/Arch:      Linux/amd64

```

4.2 通过容器来运行 web 应用

容器可以运行任何的 linux 程序，接下来我们将开启互联网之旅，实验一下容器运行 web 应用，本节我们将使用一个国内的仓库，而不是 docker 的主仓库。

4.2.1 使用国内仓库

由于国外的 dockerhub 访问速度非常缓慢，经常会 pull 不下来镜像，所以我们在这一节将使用由 ghostcloud 提供的 docker 仓库。如果你是通过 ghostcloud 来安装的 docker，默认就可以使用 hub.ghostcloud.cn 的仓库，否则需要修改一下本机的配置：将

```
--insecure-registry=hub.ghostcloud.cn
```

添加到在/etc/default/docker 的 DOCKER_OPTS 中。

4.2.2 拉取 apache-php 镜像

```
root@ghostcloud:~# docker pull hub.ghostcloud.cn/apache-php
Using default tag: latest
latest: Pulling from apache-php
b3efe11ed0e2: Pull complete
dacf881c67ea: Pull complete
1d5466c852f4: Pull complete
a053e7ea2233: Pull complete
95893f055a98: Pull complete
5ab76f7b9b5a: Pull complete
5f426bf84ca4: Pull complete
7f9885e7d7c3: Pull complete
ad7ca9b768a8: Pull complete
131a92c5d64b: Pull complete
Digest: sha256:3ffadad4c12bdd37ada051ce5316dcdfd31979d93df25252147f277f5ff876c8
Status: Downloaded newer image for hub.ghostcloud.cn/apache-php:latest
```

镜像的参数是一个域名加上镜像名。如果不带域名，默认是从 dockerhub 去拉取，否则从指定的网站去拉取。

4.2.3 运行镜像

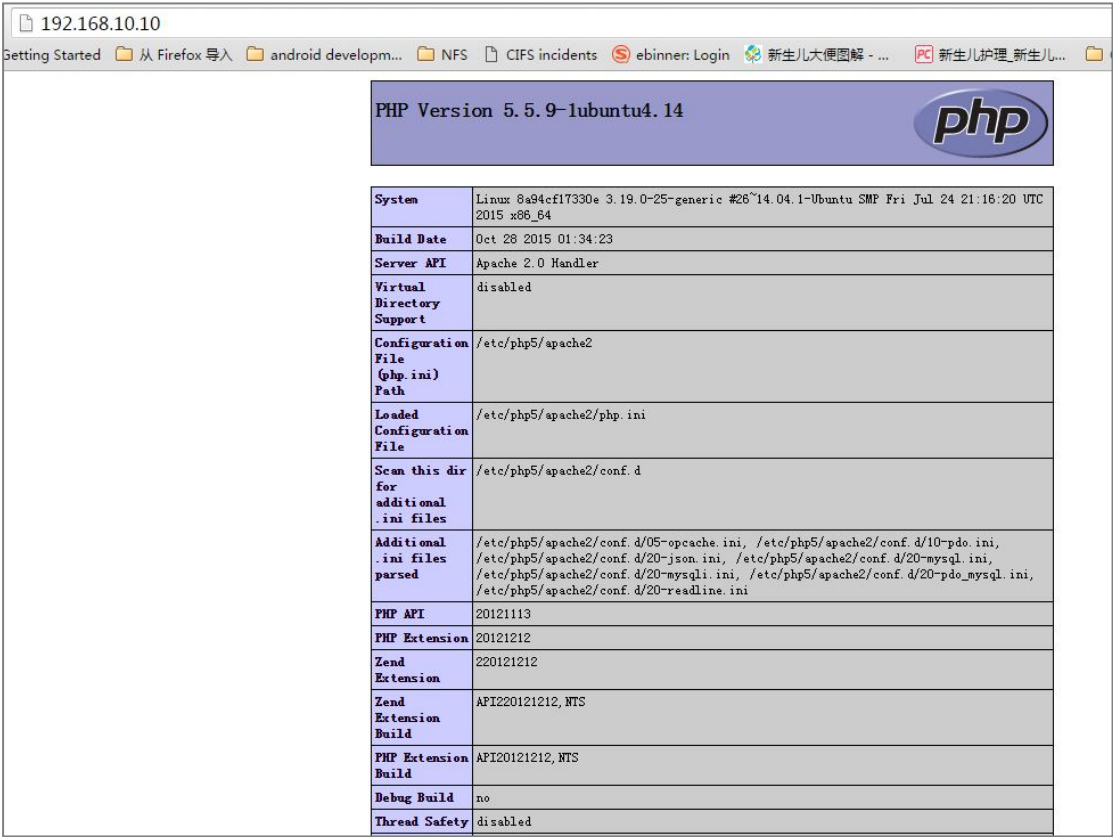
由于是 web 容器，我们将使用 -p 参数指定内外端口映射关系，同时用 -d, 让

其在后台运行。

```
root@ghostcloud:~# docker run -d -p 80:80 hub.ghostcloud.cn/apache-php

0a899413049f4f68963c5dba5c05be231a90269547eeca0bb73ad1c61b1ee512
```

4.2.4 网页访问



如图我们已经访问到了容器默认页面。

4.2.5 修改页面内容

下面我们将进入容器去修改默认页面：

1. 通过 exec 进入容器

```
root@ghostcloud:~# docker exec -it 0a8 /bin/bash
root@0a899413049f:/# ls
bin boot dev etc home lib lib64 media mnt opt proc root run sbin srv sys
tmp usr var
root@0a899413049f:/#
```

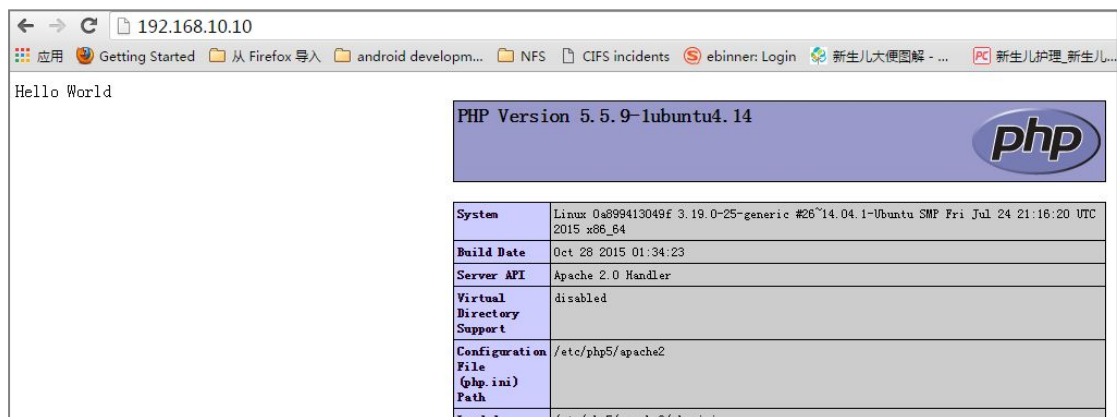
2. 进入网页目录/var/www/html，并修改 index.php

```
Hello World
```

```
<?php
phpinfo();
?>
```

在第一行加入 Hello World。

3. 通过网页再次访问



我们可以看到，左上角多了一个“Hello World”

4.2.6 持久化容器

根据我们之前的介绍，容器在退出后并不会更改镜像。所以，如果你希望保存容器中的数据，就需要通过 `commit` 来保存成镜像：

```
root@ghostcloud:~# docker commit 0a89 my-apache-php
99a35bc09c482164bfd05f20e9477303531f4c10f7310f0da361f92330ac2b03
root@ghostcloud:~# docker images my-apache-php
```

REPOSITORY	TAG	IMAGE ID	CREATED	VIRTUAL SIZE
my-apache-php	Latest	99a35bc09c48	28 seconds ago	250.3 MB

4.2.7 小节

本例中，我们首先添加了一个国内的仓库，然后下载了一个 `apache-php` 镜像，后来又进入容器做了修改，最后我们将新的容器转化为了镜像。

4.3 镜像操作

从本节开始，我们将系统性的介绍 docker 一些基础概念。镜像是容器的基础，当你运行 `docker run` 的时候，你会指定一个镜像，这个镜像一方面承载了容器的文件系统，同时也包含了容器启动时需要执行的入口文件等信息。首先来熟悉一下镜像的基础命令：

4.3.1 查看本机镜像

```
root@ghostcloud:~# docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	VIRTUAL SIZE
my-apache-php	latest	99a35bc09c48	26 minutes ago	250.3 MB
dccabffa-e9e6-11e5-b6c8-0242c0a80002	latest	c524c2760b6f	20 hours ago	204.7 MB
b3e82393-e9e6-11e5-b6c8-0242c0a80002	latest	f338840d2e17	20 hours ago	204.7 MB
myubuntu	latest	b1616061a25f	21 hours ago	188 MB
hub.ghostcloud.cn/apache-php	latest	131a92c5d64b	5 days ago	250.3 MB
fedora	latest	9bdb5101e5fc	10 days ago	204.7 MB
centos	latest	eb98cdc284d4	10 days ago	196.6 MB
<none>	<none>	56063ad57855	11 days ago	188 MB
ubuntu	latest	8ed581e3fa7a	11 days ago	188 MB
hello-world	latest	975b84d108f1	5 months ago	960 B
training/webapp	latest	02a8815912ca	10 months ago	348.8 MB
<none>	<none>	36bf8ea12ad2	19 months ago	773.3 MB

```
root@ghostcloud:~#
```

这是我本机的镜像列表，之前我们已经讲过 REPOSITORY, TAG,等的含义，这里就不再累述。

4.3.2 获取镜像的 3 种方式

1. 通过 pull 来拉取

这种方式会接收一个镜像名作为参数，但是镜像名是有特定含义的，其格式为：[域名]/[用户名]/镜像名[:版本号]。

- 当用户指定了域名时，会从该域名进行下载，否则从 `dockerhub` 下载；
- 用户名是隶属于该域名下的子目录，在使用私有仓库时有用；
- 镜像名是必填项；
- 版本号默认是 `latest`，也可以指定。

在我的镜像列表中，就有来自 `hub.ghostcloud.cn` 的镜像，也有属于 `training` 用户的镜像，`<none>` 镜像中间镜像，现在我们可以不用去管。

```
root@ghostcloud:~# docker pull hub.ghostcloud.cn/mysql
```

Using default tag: latest

latest: Pulling from mysql

044ffdf80f70: Downloading [=====] 14.46

```

MB/51.37 MB
 9d6be9c04d1c: Download complete
 555dd9d744e8: Download complete
 5d9945c35afd: Download complete
 b801734f5057: Download complete
 4aa7ca2c1a59: Download complete
 01da3cd98fb7: Download complete
 c6283fb42105: Download complete
 838b735ebc0b: Download complete
 e9c6b2cc4b90: Downloading [=====>] 14.3
MB/63.96 MB
 0a983fabb1e9: Download complete
 ac1f583509c9: Download complete
 f2784f200768: Download complete
 4b93b18502fe: Download complete
 7ca99f1ad085: Download complete
 7ce9d75f98f0: Download complete

```

2. 通过容器 commit 得到，之前我们其实就已经通过 docker commit 生成过镜像了。

3. 通过 Dockerfile 生成，Dockerfile 就像一个 shell 脚本一样，只是略微加入了一些规则，有兴趣的可以在网上查看具体的写法。

4.3.3 查找 dockerhub 镜像

查找默认是从 dockerhub 进行搜索：

```

root@ghostcloud:~# docker search
docker: "search" requires 1 argument.
See 'docker search --help'.

Usage:  docker search [OPTIONS] TERM

Search the Docker Hub for images

```

我们来搜索一下 dockerhub 中的所有 mysql 镜像：

```

root@ghostcloud:~# docker search
docker: "search" requires 1 argument.
see 'docker search --help'.

Usage: docker search [OPTIONS] TERM

Search the Docker Hub for images
root@ghostcloud:~# docker search mysql

```

NAME	DESCRIPTION	STARS	OFFICIAL	AUTOMATED
mysql	MySQL is a widely used, open-source relational database management system.	1933	[OK]	
mariadb	MariaDB is a community-developed fork of MySQL, focused on performance, stability and security.	494	[OK]	
mysql/mysql-server	Optimized MySQL Server Docker images. Created by the MySQL team.	114		[OK]
centurylink/mysql	Image containing mysql. Optimized to be lightweight.	38		[OK]
sameersbn/mysql	MySQL Docker image.	31		[OK]
google/mysql	MySQL server for Google Compute Engine.	14		[OK]
appcontainers/mysql	Centos/Debian/Ubuntu Based Customizable MySQL Docker image.	7		[OK]
marvambass/mysql	MySQL Server based on Ubuntu 14.04.	5		[OK]
drupaldocker/mysql	MySQL for Drupal.	2		[OK]
alterway/mysql	Docker MySQL.	2		[OK]
yfix/mysql	Yfix docker built mysql.	2		[OK]
azukiapp/mysql	Docker image to run MySQL by Azuki - http://azukiapp.com.	2		[OK]
bahmni/mysql	Mysql container for bahmni. Contains the MySQL server and client.	1		[OK]
frodenas/mysql	A Docker Image for MySQL.	1		[OK]
phpmentors/mysql	MySQL server image.	1		[OK]
ivories/mysql	mysql.	1		[OK]
wint/mysql	MySQL (MariaDB).	0		[OK]
tozd/mysql	MySQL (MariaDB fork) Docker image.	0		[OK]
cloudposse/mysql	Improved 'mysql' service with support for various configurations.	0		[OK]
nanobox/mysql	MySQL service for nanobox.io.	0		[OK]
projectomakase/mysql	Docker image for MySQL.	0		[OK]
dockerizedrupal/mysql	docker-mysql.	0		[OK]
wenzizone/mysql	mysql.	0		[OK]
xlight/mysql	mysql with my.cnf.	0		[OK]
lancehudson/docker-mysql	MySQL is a widely used, open-source relational database management system.	0		[OK]

Dockerhub 由于在国外，有时候搜索会失败，可以多次尝试。

4.3.4 查找其他仓库镜像

如果不是其他仓库的镜像，目前只能通过 restful 接口去查找，比如我们要搜索 `hub.ghostcloud.cn` 的镜像，可以用以下命令：

```

root@ghostcloud:~# curl http://hub.ghostcloud.cn/v2/_catalog

{"repositories":
  [
    "alpine",
    "apache-php",
    "apache_tomcat",
    "apachetomcat",
    "busybox",
    "centos",
    "chenpeng/12300000",
    "chenpeng/chenpengtestimage",
    "chenpeng/java",
    "chenpeng/jave",
    "chenpeng/mmmmmmmmm",
    "chenpeng/testimage",
    "debian",
    "discuz",
    "discuz-zh",
    "django",
    "drown_scan",
    "ecshop",
    "fedora",

```

```
"gcauth",
"gcbuild",
"gchelp/ghostcloud",
"gchomepage",
"gcmongodb",
"gcnginx",
"gcplatform",
"gcrelease",
"gcreping",
"gcserver",
"gctestlink",
"gitlab-ce",
"golang",
"hadoop-docker",
"httpd",
"httpd-image-php5",
"java",
"lamp",
"library/alpine",
"library/buildpack-deps",
"library/busybox",
"library/cassandra",
"library/centos",
"library/debian",
"library/django",
"library/docker",
"library/elasticsearch",
"library/fedora",
"library/ghost",
"library/golang",
"library/haproxy",
"library/hello-world",
"library/httpd",
"library/iojs",
"library/java",
"library/jenkins",
"library/jetty",
"library/jruby",
"library/kibana",
"library/logstash",
"library/mariadb",
"library/maven",
"library/memcached",
"library/mongo",
```

```

    "library/mysql",
    "library/neo4j",
    "library/nginx",
    "library/node",
    "library/oraclelinux",
    "library/owncloud",
    "library/php",
    "library/postgres",
    "library/python",
    "library/rabbitmq",
    "library/rails",
    "library/redis",
    "library/registry",
    "library/rethinkdb",
    "library/ruby",
    "library/swarm",
    "library/tomcat",
    "library/ubuntu",
    "library/ubuntu-debootstrap",
    "library/wordpress",
    "liufm/ping",
    "liufm/ttwwe",
    "liuj/test",
    "lnmp",
    "maxiaoping/test1image",
    "mongo",
    "mongodb",
    "mysql",
    "nodejs",
    "php",
    "php5.6-apache",
    "php5.6-cli",
    "php5.6-fpm",
    "php5.6-zts",
    "phpmyadmin",
    "postgres",
    "python"
]
}

```

如果你要查看 ubuntu 的版本，可以用这个链接：

```

root@ghostcloud:~# curl http://hub.ghostcloud.cn/v2/ubuntu/tags/list
{"name": "ubuntu", "tags": ["14.04"]}

```

4.3.5 push 镜像

之前我们都是从 hub 上拉取镜像，这一小节，我们将讲一下如何向 hub 上 push 镜像。在 push 之前我们需要先 tag 镜像：

```
root@ghostcloud:~# docker tag
docker: "tag" requires 2 arguments.
see 'docker tag --help'.

Usage:  docker tag [OPTIONS] IMAGE[:TAG] [REGISTRYHOST/][USERNAME/]NAME[:TAG]

Tag an image into a repository
root@ghostcloud:~#
```

这是 tag 的语法，它可以把一个镜像的 push 目的地进行标记，为 push 做准备。下面我将本地的 myubuntu 标记为 hub.ghostcloud.cn/yandong/myubuntu，之后再调用 push 到远端的私有仓库：

```
root@ghostcloud:~# docker tag myubuntu hub.ghostcloud.cn/yandong/myubuntu
root@ghostcloud:~# docker images
REPOSITORY          TAG          IMAGE ID          CREATED          VIRTUAL SIZE
my-apache-php        latest       99a35bc09c48      About an hour ago 250.3 MB
dccabffa-e9e6-11e5-b6c8-0242c0a80002 latest       c524c2760b6f      21 hours ago    204.7 MB
b3e82393-e9e6-11e5-b6c8-0242c0a80002 latest       f338840d2e17      21 hours ago    204.7 MB
hub.ghostcloud.cn/yandong/myubuntu latest       b1616061a25f      22 hours ago    188 MB
myubuntu             latest       b1616061a25f      22 hours ago    188 MB
hub.ghostcloud.cn/apache-php latest       131a92c5d64b      5 days ago      250.3 MB
fedora               latest       9bdb5101e5fc      10 days ago     204.7 MB
centos               latest       eb98cdc284d4      10 days ago     196.6 MB
ubuntu               latest       8ed581e3fa7a      11 days ago     188 MB
<none>               <none>       56063ad57855      11 days ago     188 MB
hub.ghostcloud.cn/mysql latest       7ce9d75f98f0      12 days ago     361.3 MB
hello-world          latest       975b84d108f1      5 months ago    960 B
training/webapp       latest       02a8815912ca      10 months ago   348.8 MB
<none>               <none>       36bf8ea12ad2      19 months ago   773.3 MB
root@ghostcloud:~# docker push hub.ghostcloud.cn/yandong/myubuntu
The push refers to a repository [hub.ghostcloud.cn/yandong/myubuntu] (len: 1)
b1616061a25f: Pushed
8ed581e3fa7a: Pushed
0b427fcc4cbb: Pushed
9d89fd8f8a3e: Pushed
073de23ee32b: Pushed
latest: digest: sha256:bba40fd8d96e4dbd8e7f00dc3bda2bf7ca731d3769ea4a9262ebf1378220bb8e size: 8001
root@ghostcloud:~#
```

4.3.6 根据 Dockerfile 编译镜像

如果对 linux 开源软件熟悉的话，一定知道 makefile，Dockerfile 和 makefile 类似，就是一个编译脚本，用于生成镜像。

1. 创建一个文件夹，并新建一个 Dockerfile

```
root@ghostcloud:~/myimage# cat Dockerfile

# This is a comment

FROM ubuntu:latest

MAINTAINER Shev Yan <yandong_8212@163.com>

CMD echo 'hello my image from Dockerfile.'
```

2. Build

```
root@ghostcloud:~/myimage# docker build .
Sending build context to Docker daemon 2.048 kB
Step 1 : FROM ubuntu:latest
```

```

---> 8ed581e3fa7a
Step 2 : MAINTAINER Shev Yan <yandong_8212@163.com>
---> Running in 9b9d552f8184
---> 727c430beada
Removing intermediate container 9b9d552f8184
Step 3 : CMD echo 'hello my image from Dockerfile.'
---> Running in 343f2abeb557
---> 30331f9ff01a
Removing intermediate container 343f2abeb557
Successfully built 30331f9ff01a

```

3. 查看并运行新生成的镜像

```

root@ghostcloud:~/myimage# docker images myimage

```

REPOSITORY	TAG	IMAGE ID	CREATED	VIRTUAL SIZE
myimage	latest	30331f9ff01a	4 minutes ago	188 MB

```

root@ghostcloud:~/myimage# docker run --rm myimage
hello my image from Dockerfile.

```

4.3.7 删除镜像

```

root@ghostcloud:~/myimage# docker rmi myimage
Untagged: myimage:latest
Deleted: 30331f9ff01ae83a3eea81782e1f3083d63803866e3cecd5bdca54a92cafa9c6
Deleted: 727c430beada158f26a0a7c19caa864a66e8330354ef0a022b0a8ca08fe6f0ba

```

4.4 docker run

之前我们已经接触到了一些容器相关的操作，在本小节中，我们将详细讲解 docker run，可以说这个命令是 docker 所有命令里面最核心的命令。Docker 的更新速度非常快，我们是以 docker 1.9 作为本书的 docker 版本。

4.4.1 docker run 的语法规则

下面是 docker run 的基本格式：

```
$ docker run [OPTIONS] IMAGE[:TAG|@DIGEST][COMMAND][ARG...]
```

这个命令必须指定 IMAGE，一个镜像默认包含了以下信息：

- 前台运行还是后台运行
- 容器的 id
- 网络设置

- 运行中的 CPU 和内存

但是几乎所有的信息，都可以通过 `docker run [OPTIONS]` 来进行修改。
`OPTIONS` 是一个可选项，如果不带，就会首先使用镜像的参数，然后再使用全局的默认参数。所有的以下介绍都可以通过 `man docker run` 得到。

4.4.2 前后台运行

1. 后台运行(-d)

在启动容器的时候，如果你要将容器放在后台运行，可以使用 `-d=true` 或者 `-d` 参数，`-d` 的英文对应的是 `detached`，其原理就是不跟容器的 `stdin`, `stdout` 进行绑定。有些时候可能你会使用 `service xxx start` 命令来作为容器启动命令，但是这是有问题的，虽然这个命令执行成功了，但是后续就退出了，紧接着容器也会退出。因此，容器的生存周期是直接和你启动容器的命令生命周期一致的，虽然你可以通过该命令 `fork` 出子进程，但是一旦主进程退出，整个容器就结束了。如果需要再容器中运行多个程序，可以使用 `Supervisord`。

当以 `-d` 参数运行后，如果要再次进入容器可以使用 `docker attach <cid>` 的方式重新绑定到当前 `shell` 的终端上。如果要再次进入 `-d` 模式，不能输入 `ctrl+c`，而应该使用 `ctrl+pq`

2. 前台运行

如果不带 `-d` 参数，就是放在前台运行。有些时候我们要 `attach` 到容器里面的 `shell`，我们可以通过 `-i -t` 参数：

```
root@ghostcloud:~# docker run -it ubuntu /bin/bash
root@cb90c945e2b8:/# ls
bin boot dev etc home lib lib64 media mnt opt proc root run sbin srv sys
tmp usr var
```

当我们进入容器的时候，默认的主机名就是容器的短 ID - `cb90c945e2b8`

4.4.3 容器的标识

1. --name 属性

容器有三种方式来进行标识：

名称	举例
长 UUID	"f78375b1c487e03c9438c729345e54db9d20cfa2ac1fc3494b6eb60872e74778"
短 UUID	"f78375b1c487"
Name	"evil_ptolemy"

UUID 是 Docker daemon 产生的，在一台主机上是唯一的，在创建容器的时候可以通过 `--name` 来指定容器的名字，如果不指定会自动分配一个字符串名称。该 name 一个比较大的用途就是可以用于在两个容器之间建立 link 通信。

2. Image[:tag]

之前已经讲过，docker run 的时候必须指定镜像名，这个就是它的格式，例如：

```
$ docker run ubuntu:14.04
```

3. Image[@digest]

从镜像的 V2 版本格式开始，每一个镜像都包含了一个含有镜像内容的签名，你同样也可以通过这种方式来指定镜像。通过 `docker inspect <image_id>` 可以看到 digest。

4.4.4 PID 设置

```
--pid="": Set the PID (Process) Namespace mode for the container, 'host': use the host's PID namespace inside the container
```

PID 用于控制，容器中的进程使用什么 pid，一般来说主机上的进程 ID 是从 1 开始的，通常是 init 进程，而容器中执行的程序的 pid 也是从 1 开始的，这就是 pid namespace 的实现：

```
root@cb90c945e2b8:/# ps -ef
UID          PID    PPID  C STIME TTY          TIME CMD
root           1         0  0 02:00 ?           00:00:00 /bin/bash
root          16         1  0 03:47 ?           00:00:00 ps -ef
```

如果你需要在容器里面共享主机的 pid namespace，那就加上 `--pid=host`，之后的情况会是：

```
root@ghostcloud:~# docker run -it --rm --pid=host ubuntu /bin/bash
root@158c0108a34f:/# ps -ef
UID          PID    PPID  C STIME TTY          TIME CMD
root           1         0  0 Mar12 ?           00:00:03 /sbin/init
```

```

root      2      0  0 Mar12 ?      00:00:00 [kthreadd]
root      3      2  0 Mar12 ?      00:00:08 [ksoftirqd/0]
root      5      2  0 Mar12 ?      00:00:00 [kworker/0:0H]
root      7      2  0 Mar12 ?      00:00:13 [rcu_sched]
...

```

4.4.5 UTS(--uts) 设置

Docker 1.10 之后支持：

```
--uts="": Set the UTS namespace mode for the container, 'host': use the host's UTS namespace inside the container
```

UTS 可以让容器同主机使用相同的hostname和domain,使用的时候要慎重。

4.4.6 IPC(--ipc)设置

```
--ipc="": Set the IPC mode for the container, 'container:<name/id>': reuses another container's IPC namespace 'host': use the host's IPC namespace inside the container
```

这个是进程间通讯的支持，可以和主机共享。

4.4.7 网络设置

```

--dns=[] : Set custom dns servers for the container
--net="bridge" : Connect a container to a network
                'bridge': create a network stack on the default Docker bridge
                'none': nonetworking
                'container:<name/id>': reuse another container's network stack
                'host': use the Docker host network stack
                '<network-name>|<network-id>': connect to a user-defined network
--net-alias=[] : Add network-scoped alias for the container
--add-host="" : Add a line to /etc/hosts (host:IP)
--mac-address="" : Sets the container's Ethernet device's MAC address
--ip="" : Sets the container's Ethernet device's IPv4 address
--ip6="" : Sets the container's Ethernet device's IPv6 address

```

之前我们说过，容器有 5 种网络方式，默认都是用的 bridge，通过主机和容器的端口映射来通信。同时，也可以给容器设置 dns。下面是 docker 支持的各种网络模式：

网络	描述
None	不在容器中使用网络

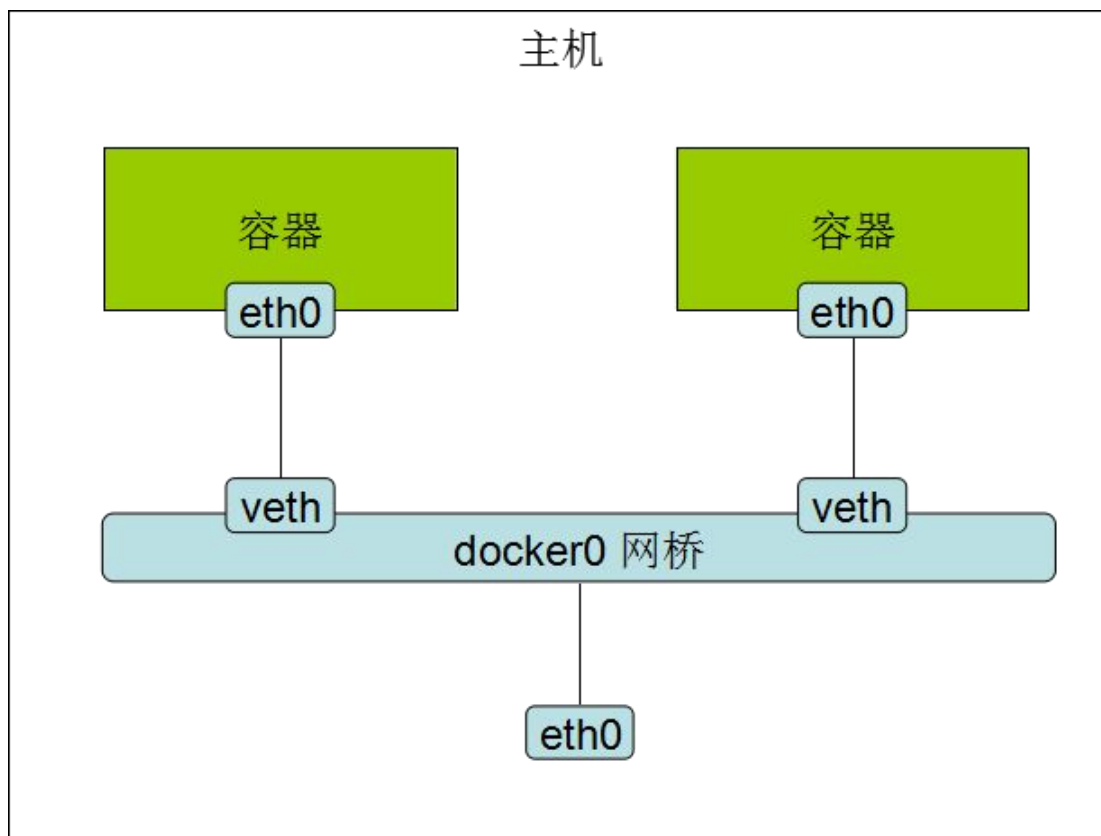
Bridge(默认)	容器通过 veth 连接到主机的桥接上
Host	使用和主机相同的网络
Container:<name id>	使用别的容器的网络
NETWORK	使用用户自定义的网络(通过 <code>docker network create</code> 创建)

- None

容器不能访问外部网络，内部存在回路地址。

- Bridge

桥接是在主机上的，一般叫 `docker0`，每启动一个容器的时候，会为该容器创建一个 veth，veth 一端连接到 `docker0`，一端连接到容器内的 `eth0`，如下图所示：



- Host

就是使用主机的网络，没什么特殊的。相比于桥接，该模式由于没有通过 `iptables` 的转发，性能上要比桥接好一些。但是根据测试，也只有 15%左右的损失，相比于 `nat` 的封闭和安全性，更建议使用桥接。在对网络性能特别关注的时候，

如 HAProxy 等负载均衡器时，可以使用 host 模式。

- Container

该模式将容器的网络栈合并到了一起，如果二者在同一台机器，且不需要独立的网络地址，是比较好的而一种选择。

- 自定义网络

Network 是用户自定义的网络，这一功能主要是为了解决 docker 跨网络通信能力不足的问题和特殊网络需求问题。对于 overlay 和用户定义的 mutlti-host 模式，就应该使用该模式。

- /etc/hosts 管理

通过--add-host 可以在容器中添加 ip 到 host 的解析规则。

4.4.8 重启策略 (--restart)

--restart 参数用来指定当容器退出后的行为，当容器在重启的时候，docker ps 可以看到处于 Up 或 Restarting，也可以在 docker events 中看到相关信息。目前 docker 支持以下几种 restart 策略：

策略	作用
No	没有任何重启操作，也是默认的属性
On-failure	当容器的命令返回非 0 值，即有错时，进行重启
Always	不管容器状态，都重启。同时，容器会在 daemon 启动时，附带自启动
Unless-stopped	好 Always 类似，只不过不会随 daemon 自启动

容器的退出状态就是执行命令的错误码，linux 下一般 0 表示正常，其他表示错误。

4.4.9 Clean up (--rm)

默认情况下容器退出后，并不会被删除，如果通过 docker ps -a，可以看到所有的容器，包括运行中的和停止的。通过--rm 可以在容器退出时自动删除容器，方便清楚残留的文件。

4.4.10 CGroup 控制

之前我们讲了，CGroup 是用来限定容器中，资源使用情况的，接下来我们来讲一下具体可以控制的东西有哪些：

1. 用户态内存的控制
2. 内核态内存控制
3. CPU 的控制
4. 磁盘 IO 的控制
5. 设备读写速率的控制
6. 内存耗尽（OOM）时的行为

对于具体的细节可以查看

<https://docs.docker.com/engine/reference/run/#specifying-custom-cgroups>

4.4.11 特权模式和 Capabilities

默认情况下，容器是运行在非特权模式下的，这样容器内是不能访问任何宿主机的设备的，但是有些时候用户可能会需要访问某些设备，就必须使用 `--privileged` 参数。如果你希望使用某个特定设备可以使用 `--device`，比如你想使用 GPU 时。

```
$ docker run --device=/dev/snd:/dev/snd...
```

默认情况设备是可读可写，并且支持 `mknod`，也可以通过 `rwm` 来修改

```
$ docker run --device=/dev/sda:/dev/xvdc--rm -it ubuntu fdisk /dev/xvdc
Command (m for help):q

$ docker run --device=/dev/sda:/dev/xvdc:r--rm -it ubuntu fdisk /dev/xvdc
You will not be able to write the partition table.

Command (m for help):q

$ docker run --device=/dev/sda:/dev/xvdc:w--rm -it ubuntu fdisk /dev/xvdc crash...
$ docker run --device=/dev/sda:/dev/xvdc:m--rm -it ubuntu fdisk /dev/xvdc
fdisk:unable to open /dev/xvdc:Operation not permitted
```

除此之外，你还可以使用 `--cap-add`, `--cap-drop` 来设置 Capabilities.

4.4.12 日志驱动 (--log-driver)

容器默认的日志是输出到 stdout, stderr 中的，你也可以在 docker daemon 中设置不同的日志输出方式。通过 --log-driver=VALUE 也能为单个容器指定不同的日志格式：

日志驱动	描述
None	不显示日志，docker logs 没有输出
Json-file	默认的日志输出方式，通过 JSON 来保存日志。
Syslog	将日志输出到系统日志中，通常是 /var/log/message
gelf	Graylog Extended Log Format(GELF)日志，日志会写到 GELF 的收集器中，如 Graylog 或者 Logstash
journald	将日志写到 journald
fluentd	写到 fluentd
awslogs	写入到 Amazon CloudWatch
splunk	写入到 splunk 日志收集器中

4.4.13 覆盖 image 的默认参数

当用户编写 Dockerfile 时可能会带一些默认参数，目前只有 FROM, MAINTAINER, RUN 和 ADD 是不能覆盖的，其他都是可以覆盖的，比如：

- CMD（默认命令或属性）
- ENTRYPOINT（默认执行的命令）
- EXPOSE（导出的端口）
- ENV（环境变量）
- VOLUME（卷挂载）
- USER
- WORKDIR

CMD 和 ENTRYPOINT 的区别：CMD 其实更像是一个命令参数，你在 run 的时候指定了命令，然后 CMD 就会作为参数；ENTRYPOINT 是镜像的默认启

动项，所以 CMD 和你所指定的参数都会作为 ENTRYPOINT 的参数。

5 容器的网络

容器如果离开了网络，就是一个密闭的盒子。Docker 初期的网络功能并不是很完善，随着不断的演化，目前 docker 的网络已经大大加强，本小节将介绍 Docker 的网络部分。

5.1 自带的网络

当你安装 docker 成功后，就会创建三种网络，你可以通过 `docker network ls` 进行查看：

```
$ docker network ls
NETWORK ID NAME DRIVER
7fca4eb8c647 bridge bridge
9f904ee27bf5 none null
cf03ee007fb4 host host
```

这三种网络就是系统自带的，创建容器的时候你可以通过 `--net` 进行指定。对于 `bridge` 而言，默认是主机挂接在主机的 `docker0` 上的，在主机上通过 `ifconfig` 可以查看到：

```
root@ghostcloud:~# ifconfig
br-40719032dd42 Link encap:Ethernet HWaddr 02:42:02:e6:b2:db
    inet addr:172.18.0.1 Bcast:0.0.0.0 Mask:255.255.0.0
    inet6 addr: fe80::42:2ff:fee6:b2db/64 Scope:Link
    UP BROADCAST MULTICAST MTU:1500 Metric:1
    RX packets:1 errors:0 dropped:0 overruns:0 frame:0
    TX packets:2 errors:0 dropped:0 overruns:0 carrier:0
    collisions:0 txqueuelen:0
    RX bytes:76 (76.0 B) TX bytes:168 (168.0 B)

docker0 Link encap:Ethernet HWaddr 02:42:8c:ca:18:18
    inet addr:172.17.0.1 Bcast:0.0.0.0 Mask:255.255.0.0
    UP BROADCAST MULTICAST MTU:1500 Metric:1
    RX packets:0 errors:0 dropped:0 overruns:0 frame:0
    TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
    collisions:0 txqueuelen:0
    RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)

eth0 Link encap:Ethernet HWaddr 00:0c:29:0f:5d:4d
```



```

    inet addr:192.168.10.10 Bcast:192.168.255.255 Mask:255.255.0.0
    inet6 addr: fe80::20c:29ff:fe0f:5d4d/64 Scope:Link
    UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
    RX packets:368767 errors:1 dropped:181 overruns:0 frame:0
    TX packets:25936 errors:0 dropped:0 overruns:0 carrier:0
    collisions:0 txqueuelen:1000
    RX bytes:68266498 (68.2 MB) TX bytes:31563732 (31.5 MB)

Lo    Link encap:Local Loopback
    inet addr:127.0.0.1 Mask:255.0.0.0
    inet6 addr: ::1/128 Scope:Host
    UP LOOPBACK RUNNING MTU:65536 Metric:1
    RX packets:0 errors:0 dropped:0 overruns:0 frame:0
    TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
    collisions:0 txqueuelen:0

```

5.2 网络详情

我们可以通过 `docker network inspect <net>` 来查看相信信息：

```

root@ghostcloud:~# docker network inspect bridge
[
  {
    "Name": "bridge",
    "Id": "33e251770bda7a9adfb0ef7b7f468a71f577cc8af0e1a44369a5a295fdc66b0",
    "Scope": "local",
    "Driver": "bridge",
    "IPAM": {
      "Driver": "default",
      "Options": null,
      "Config": [
        {
          "Subnet": "172.17.0.0/16",
          "Gateway": "172.17.0.1"
        }
      ]
    },
    "Containers": {},
    "Options": {
      "com.docker.network.bridge.default_bridge": "true",
      "com.docker.network.bridge.enable_icc": "true",
      "com.docker.network.bridge.enable_ip_masquerade": "true",
      "com.docker.network.bridge.host_binding_ipv4": "0.0.0.0",
      "com.docker.network.bridge.name": "docker0",

```

```

        "com.docker.network.driver.mtu": "1500"
    }
}
]

```

当我启动一个容器的时候，就会在 Containers 下面加入 Subnet 和 Gateway:

```

root@ghostcloud:~# docker network inspect bridge
[
  {
    "Name": "bridge",
    "Id": "33e251770bda7a9adfb0ef7b7f468a71f577cc8af0e1a44369a5a295fdc66b0",
    "Scope": "local",
    "Driver": "bridge",
    "IPAM": {
      "Driver": "default",
      "Options": null,
      "Config": [
        {
          "Subnet": "172.17.0.0/16",
          "Gateway": "172.17.0.1"
        }
      ]
    },
    "Containers": {
      "7849f9e530c7d650646673e64755d933c56051326a9e57586a8a272c415a2b0d": {
        "Name": "elated_dijkstra",
        "EndpointID":
"0730a4057116c296a1c0c19c796d68cb6f949284c8777c42094ee91a04b04f5c",
        "MacAddress": "02:42:ac:11:00:02",
        "IPv4Address": "172.17.0.2/16",
        "IPv6Address": ""
      }
    },
    "Options": {
      "com.docker.network.bridge.default_bridge": "true",
      "com.docker.network.bridge.enable_icc": "true",
      "com.docker.network.bridge.enable_ip_masquerade": "true",
      "com.docker.network.bridge.host_binding_ipv4": "0.0.0.0",
      "com.docker.network.bridge.name": "docker0",
      "com.docker.network.driver.mtu": "1500"
    }
  }
]

```

5.3 用户自定义网络

Docker 允许用户创建自己的网络，主要包括三种，1) 桥接网络；2) Overlay 网络；3) 插件网络。

5.3.1 桥接网络

系统默认的桥接是 docker0，如果你想将多个容器隔离在一个新的桥接网络中，可以使用以下命令：

```
root@ghostcloud:~# docker network create --driver bridge mynet
e4a0a19ebb624f82b396915a6439f4bf4c8d520f50fb29ddc86dfd012106c6d6

root@ghostcloud:~# docker network inspect mynet
[
  {
    "Name": "mynet",
    "Id": "e4a0a19ebb624f82b396915a6439f4bf4c8d520f50fb29ddc86dfd012106c6d6",
    "Scope": "local",
    "Driver": "bridge",
    "IPAM": {
      "Driver": "default",
      "Options": {},
      "Config": [
        {
          "Subnet": "172.19.0.0/16",
          "Gateway": "172.19.0.1/16"
        }
      ]
    },
    "Containers": {},
    "Options": {}
  }
]
```

之后，你可以通过--net 属性将容器挂接到 mynet 中：

```
root@ghostcloud:~# docker run --net=mynet --rm -it ubuntu

root@ghostcloud:~# docker network inspect mynet
[
  {
    "Name": "mynet",
```

```

    "Id": "e4a0a19ebb624f82b396915a6439f4bf4c8d520f50fb29ddc86dfd012106c6d6",
    "Scope": "local",
    "Driver": "bridge",
    "IPAM": {
      "Driver": "default",
      "Options": {},
      "Config": [
        {
          "Subnet": "172.19.0.0/16",
          "Gateway": "172.19.0.1/16"
        }
      ]
    },
    "Containers": {
      "b8321c436612a350964ad08c0a1f6c31da328c67e399b30be160eefe1a9a0631": {
        "Name": "desperate_hodgkin",
        "EndpointID":
"14d0e3aa846bba610b19eaca31436d4b80e190796a9b2f1f45e7731fa0110575",
        "MacAddress": "02:42:ac:13:00:02",
        "IPv4Address": "172.19.0.2/16",
        "IPv6Address": ""
      }
    },
    "Options": {}
  }
]

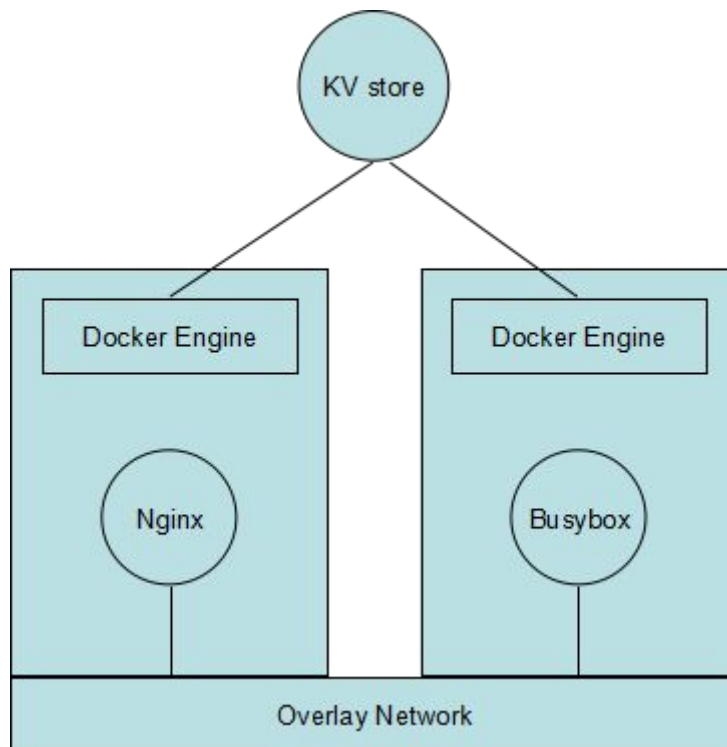
```

在同一个桥接下，就形成了一个私网，相互间是可以通讯的，但是这仅限于同一台主机上，如果你要跨主机通讯，就必须使用 overlay 网络。

5.3.2 overlay 网络

Overlay 是一种虚拟交换技术，主要是解决不同 IP 地址段之间的网络通讯问题。Docker 使用的 overlay 技术是 VXLAN，借助于 libnetwork 来实现的。Overlay 网络需要一个 K-V 的服务来存储相关的主机信息，目前 Docker 支持的 K-V 存储服务有 Consul，Etcd 和 ZooKeeper，其中 Consul 和 Etcd 都是用 Go 来进行开发的，ZooKeeper 是用 Java 开发的，由于 Go 的原生跨平台型，Consul 就成为了默认的发现服务。

对于 Overlay，主机还必须开放 UDP/4789 和 TCP/UDP/7946，分别用作数据通道和控制通道。下面我们就来创建一个 overlay 网络：



圆形的部分都是容器，步骤：

1. 创建一个 docker-machine 用于运行 KV store

```
$ docker-machine create -d virtualbox mh-keystore
```

2. 在#1 中创建的虚拟机中，运行 consul KV store

```
$ docker $(docker-machine config mh-keystore) run -d -p "8500:8500" -h consul progrium/consul
-server -bootstrap

# 查看运行状态

$ docker $(docker-machine config mh-keystore) ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
8c16fceaeeff	progrium/consul	"/bin/start -server -"	2 minutes ago	Up 2 minutes

```

PORTS
53/tcp, 53/udp, 8300-8302/tcp, 8400/tcp, 8301-8302/udp,
0.0.0.0:8500->8500/tcp tender_golick

```

3. 创建两个 VM，其中一个作为 swarm 的 master

```
$ docker-machine create \ -d virtualbox \ --swarm --swarm-master \
--swarm-discovery="consul://$(docker-machine ip mh-keystore):8500" \
--engine-opt="cluster-store=consul://$(docker-machine ip mh-keystore):8500" \
--engine-opt="cluster-advertise=eth1:2376" \ mhs-demo0

$ docker-machine create -d virtualbox \ --swarm \
--swarm-discovery="consul://$(docker-machine ip mh-keystore):8500" \
--engine-opt="cluster-store=consul://$(docker-machine ip mh-keystore):8500" \
--engine-opt="cluster-advertise=eth1:2376" \ mhs-demo1
```

4. 创建一个 overlay 网络

```
$ docker $(docker-machine config mhs-demo0) network create -d overlay my-net
26c7f5d3fff8e3cb725c93c486da2ff9a8efbea19a90859e00376c9ca08f622a
```

```
Administrator@EKEUSER-PC MINGW64 /
```

```
$ docker $(docker-machine config mhs-demo0) network ls
```

NETWORK ID	NAME	DRIVER
26c7f5d3fff8	my-net	overlay
33a92eabdd52	bridge	bridge
32c656ab911d	none	null
75cec8a41354	host	host

```
Administrator@EKEUSER-PC MINGW64 /
```

```
$ docker $(docker-machine config mhs-demo1) network ls
```

NETWORK ID	NAME	DRIVER
26c7f5d3fff8	my-net	overLay
49eb74d5a193	bridge	bridge
a5483ee39c78	none	null
95e713841aba	host	host

5. 在一台主机上创建一个 nginx 容器

```
$ docker $(docker-machine config mhs-demo1) run -itd --name=web --net=my-net nginx
```

6. 在另一台机器上访问 nginx

```
$ docker $(docker-machine config mhs-demo0) run --rm --net=my-net busybox wget -O- http://web
```

```
Connecting to web (10.0.0.2:80)
```

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
<title>Welcome to nginx!</title>
```

```
<style>
```

```
body {
```

```
width: 35em;
```

```
margin: 0 auto;
```

```
font-family: Tahoma, Verdana, Arial, sans-serif;
```

```
}
```

```
</style>
```

```
</head>
```

```
<body>
```

```
<h1>Welcome to nginx!</h1>
```

```
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>
```

```
<p>For online documentation and support please refer to
```

```
<a href="http://nginx.org/">nginx.org</a>.<br/>
```

```
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>

- 100% /*****/ 612 0:00:00 ETA
```

6 容器的数据

之前我们讲过，容器中的文件系统是由分层文件系统提供的，包含只读层——镜像和可读可写层——容器运行时层，这些都是被封装在容器内部的。如果用户需要将主机上的文件系统共享给容器使用，那怎么办呢？目前有两种处理方式：

1. 数据卷——将主机的卷 mount 进入容器
2. 数据容器——将外部容器分享给容器

6.1 数据卷

数据卷提供了一种主机和容器共享数据的方式，有些时候如果我们需要用它来做持久化和数据共享。当做持久化时，通常数据卷都会比较大，你可以将其放在单独的磁盘、卷或者阵列上，这个时候容器只是一个执行环境。当做数据共享时，可以用于开发和测试分布式系统，比如需要用到共享盘，需要处理 fencing 等等。数据卷主要通过 -v 参数来指定。

6.1.1 创建一个数据卷

有时候，你需要一个外部卷来存放持久化数据，而不想把数据包含在容器内部。例如：

```
root@ghostcloud:~# docker run -d -P --name datatest -v /webapp ubuntu
13ffee3a3f50d07fd5a737aaf2efc60ceec28e2b04c5d534cfd84d8b70019c11
```

这条命令创建了一个名为 datatest 的容器，同时为其创建了一个 /webapp 的数据卷，这是数据卷在其内部的位置。那么它在主机上的什么位置呢？

```
root@ghostcloud:~# docker inspect 13ff
```

```

...
"Mounts": [
  {
    "Name":
"ec1c427a6a76be4918d6e8bac3247e2836dc8f424c9e06466fcf1baab6e7ee79",
    "Source":
"/var/lib/docker/volumes/ec1c427a6a76be4918d6e8bac3247e2836dc8f424c9e06466fcf1baab6e7ee79
/_data",
    "Destination": "/webapp",
    "Driver": "local",
    "Mode": "",
    "RW": true,
    "Propagation": ""
  }
],
...

```

它的位置在/var/lib/docker 下。

6.1.2 映射一个外部卷

如果以-v src:des 的方式指定，那么容器则会直接将主机的卷 mount 到内部：

```

root@ghostcloud:~# docker run -it -v /root:/hostroot ubuntu
root@86ea1246ae5d:/# ls /hostroot
Dockerfile  composetest  gcagent  myimage  uninstall_agent.sh

```

上面我将/root 映射到了容器的/hostroot 中，从这儿可以看出这是有很大的风险的，因此在做数据卷映射的时候，一定要特别小心，任何时候都不要将/映射到容器内部。

6.2 使用数据型容器

由于容器本身就可以包含文件系统，那么可不可以把容器的卷分享给另一个容器用呢，答案是可以的。具体的步骤如下：

1. 创建一个包含外部卷的容器，注意是 create，并不是 run。Run 是 create 后再 start，这里我们只需要容器的文件系统，所以只需要 create。

```

root@ghostcloud:~# docker create -v /dbdata --name dbstore ubuntu
d95cdc1139ed1011fe51843f524c377cd7497629e9a4434508f422f15b61a03c

```

2. 在另一个容器中通过--volumes-from 来映射：


```
root@ghostcloud:~# docker run --rm -it --volumes-from dbstore ubuntu
root@4b61bb181471:/# df -h
```

Filesystem	Size	Used	Avail	Use%	Mounted on
none	8.8G	4.2G	4.2G	50%	/
tmpfs	2.0G	0	2.0G	0%	/dev
tmpfs	2.0G	0	2.0G	0%	/sys/fs/cgroup
/dev/disk/by-uuid/27d8b1c5-4bfc-4499-94d6-6e5f5c42e923	8.8G	4.2G	4.2G	50%	/dbdata

6.3 备份、还原和迁移数据卷

下面是一条通过容器型数据卷和数据卷联合使用做备份的例子：

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

剖析：

- `--volumes-from` 表示使用 `dbstore` 这个容器的数据卷
- `-v $(pwd):/backup` 表示将当前路径映射到容器中的 `/backup` 中，用于后续备份
- `ubuntu tar cvf /backup/backup.tar /dbdata` 表示将 `/dbdata` 的内容备份到当前目录。

上面这个例子就是典型的将容器作为一个工具来使用，如果更进一步你可以自己写一个 `dockerfile`，然后产生一个 `image`，将参数上面都指定好，以后就只需启动容器就可备份，备份完成后又自动退出。

那么还原呢？

```
$ docker run --rm --volumes-from dbstore2 -v $(pwd):/backup ubuntu bash -c "cd /dbdata && tar xvf /backup/backup.tar --strip 1"
```

类似的，通过 `tar` 来解压即可。

6.4 总结

我们来总结一下，数据卷有下面几个特点：

- 数据卷在容器创建的时候进行初始化
- 数据卷可以共享，也可以在容器之间重用
- 对于数据卷的读写是直接下发的
- `Commit` 命令不会将改动保存到镜像中

- 即使容器被删除了，数据卷仍然存在，因此这一块需要特别注意，避免产生垃圾数据卷

7 镜像仓库

其实从我们的 `hello-world` 开始，我们就一直在和镜像仓库打交道，而且也许你也经历过拉取镜像失败的情况，这主要归咎于国内网路的一些限制。镜像仓库顾名思义就是存放镜像的，有了它你可以在不同的主机上使用相同的镜像，有了它可以使你的私有云和公有云进行联通。一般来说，我们可能会用到下面几种仓库：

1. **Docker Hub 主仓库**：这是所有的公共仓库，截止 2016 年 1 月，已经有超过 10 亿次的下载量。Docker Hub 有着最全面的镜像，也是 Docker 客户端默认的拉取仓库，但是由于其在国外，拉取和查询速度都非常慢。而且它默认镜像是全公开的，就是你的镜像可以被任何人下载。
2. **内部私有仓库**：用户可以在内部创建私有的仓库。
3. **公共的私有仓库**：即将仓库放在公网上，并自己管理权限。

7.1 仓库相关的 Docker 命令

主要包含 `search`, `pull`, `login` 和 `push`

7.1.1 登录

```
root@ghostcloud:~# docker login --help

Usage:  docker login [OPTIONS] [SERVER]

Register or Log in to a Docker registry.
If no server is specified, the default is defined by the daemon.

-e, --email      Email
--help          Print usage
-p, --password   Password
-u, --username   Username
```

如果不带 SERVER 将会使用 docker hub, 否则可以指定一个第三方 SERVER。
当用户登录成功后, 用户的认证信息是保存在 ~/.docker/config.json 中。

7.1.2 查找

之前其实我们已经使用过了, 我们再来查一下跟 mysql 相关的镜像:

```
root@ghostcloud:~# docker search mysql
```

NAME	DESCRIPTION	STARS	OFFICIAL	AUTOMATED
mysql	MySQL is a widely used, open-source relati...	1948	[OK]	
mysql/mysql-server	Optimized MySQL Server Docker images. Crea...	116		[OK]
centurylink/mysql	Image containing mysql. Optimized to be Li...	39		[OK]
sameersbn/mysql		31		[OK]
google/mysql	MySQL server for Google Compute Engine	14		[OK]
appcontainers/mysql	Centos/Debian/Ubuntu Based Customizable My...	7		[OK]
marvambass/mysql	MySQL Server based on Ubuntu 14.04	5		[OK]
drupaldocker/mysql	MySQL for Drupal	2		[OK]
azukiapp/mysql	Docker image to run MySQL by Azuki - http:...	2		[OK]
yfix/mysql	Yfix docker built mysql	2		[OK]
alterway/mysql	Docker Mysql	2		[OK]
boomtownroi/mysql-dev	A mysql box with consul integration for de...	1		[OK]
ivorries/mysql	mysql	1		[OK]
phpmentors/mysql	MySQL server image	1		[OK]
frodenas/mysql	A Docker Image for MySQL	1		[OK]
bahmni/mysql	Mysql container for bahmni. Contains the ...	1		[OK]
cloudposse/mysql	Improved `mysql` service with support for ...	0		[OK]
dockerizedrupal/mysql	docker-mysql	0		[OK]
akilli/mysql	debian:jessie based mysql image	0		[OK]
projectomakase/mysql	Docker image for MySQL	0		[OK]
nanobox/mysql	MySQL service for nanobox.io	0		[OK]
wint/mysql	MySQL (MariaDB)	0		[OK]
Lancehudson/docker-mysql	MySQL is a widely used, open-source relati...	0		[OK]
wenzizone/mysql	mysql	0		[OK]
tozd/mysql	MySQL (MariaDB fork) Docker image.	0		[OK]

7.1.3 拉取

比如拉取 mysql:

```
root@ghostcloud:~# docker pull mysql
```

7.1.4 提交

提交就是通过 push 来的，在 push 之前一般都需要 login，然提交到指定的目录位置，其格式为：

```
$ docker push [OPTIONS] [server/][user/]imagename[:TAG]
```

最后的那部分是通过 docker tag 来标记的，也就是说，如果你要提交到某个地方，首先是要 docker tag 成指定镜像名，然后再进行 push。

§ 实验

在你的 docker 主机上，根据教程，尝试相关的命令。

第二部分 Docker 三剑客

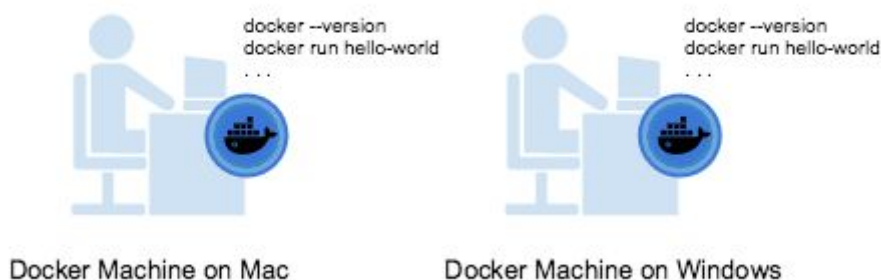
第一部分，我们主要关注在 docker-engine 本身，docker-engine 解决的是如何运行单个容器的问题，随着容器的轻量级，势必会出现大量的容器，容器与容器之间如何管理，如何有机的结合起来形成一个整体架构就是迫切需要解决的问题。除此之外，随着公有云、私有云及混合云的逐步兴起，如何让 docker 快速的、稳定的运行在这些环境，也是摆在我们面前的问题，因此第二部分，主要从这容器的规模化运用方面进行讲解。在规模化运用中，就不得不提到 Docker 三剑客，Docker Machine、Docker Compose 和 Docker Swarm。

第一节 Docker Machine

如果你想在 windows 或者 mac 的机器上运行 docker 怎么办？由于 Docker 需要借助 linux 内核的 CGroup 和 Namespace，要在这两个系统上运行的话，就需要借助虚拟机。那么 Docker Machine 就是帮助你构建拥有 Docker 运行环境的虚拟机，并能够远程的管理虚拟机及其里面的容器。

很多人也许会觉得奇怪，容器不是更轻量级吗，为什么不直接使用容器，还要使用虚拟机。主要有下面几个考虑：

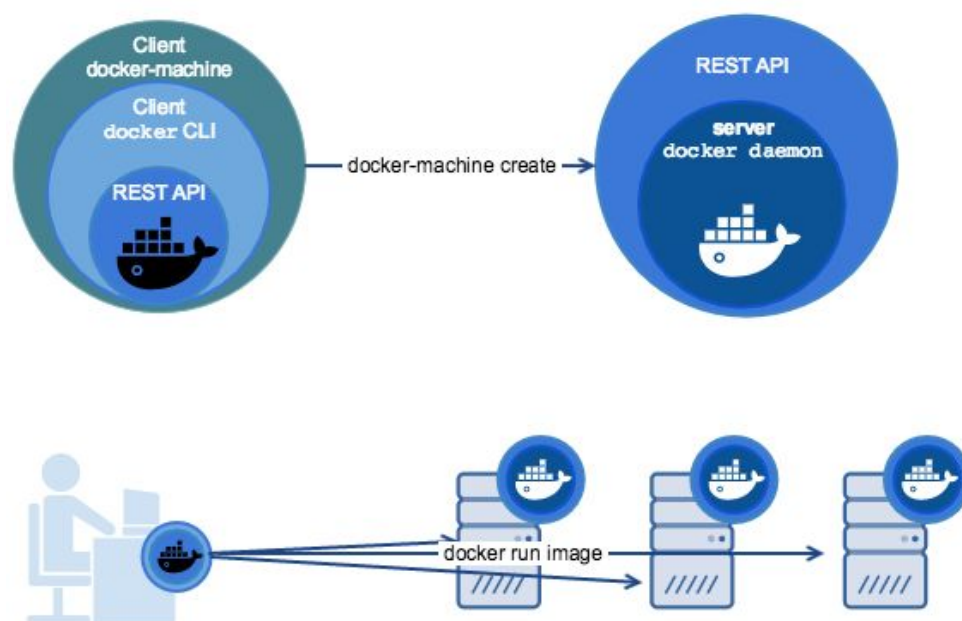
- 跨平台的支持，在非 linux 下环境，仍然需要虚拟机来运行。



- 我希望管理远端的系统，并且希望一个地方总控。比如我希望快速的部署 Docker 在云主机上，快速的扩容虚拟结点，这些都可以借助 Docker Machine。



那么 Docker Machine 和 Docker Engine 有什么区别呢？Docker Engine 是一个运行 Docker 的程序，它可以运行在虚拟机也可以运行在物理机上。Docker Machine 是将 Docker Engine 安装在了虚拟机上，并进行了打包，使其可以快速生成和方便的管理。下图就是 Docker Machine 和 Docker Engine 的示意图：



每一条 Docker Machine 的执行步骤是：

1. Docker Machine 命令发起调用
2. 内部转化成 Docker 命令行调用
3. Docker Client 通过 REST API 向远端发起调用
4. 远端的 Daemon 收到请求，并返回相应数据

1 安装 Docker Machine

Docker Machine 是一个开源项目，地址是：

<https://github.com/docker/machine>

在它的目录里面有一个 driver 目录，

..		
amazonec2	Fix amazonec2 test nitpicks	11 days ago
azure	azure: Add --azure-static-public-ip flag	4 days ago
digitalocean	[DigitalOcean] Allow to use an existing SSH key (closes #1445)	16 days ago
errdriver	FIX #2553 Fusion Driver is supported only on darwin	3 months ago
exoscale	FIX #3056 Is timeout when instance is stopped	a month ago
fakedriver	Logs for Kill	3 months ago
generic	FIX #1858 Add engine port	8 days ago
google	Google driver: add --google-use-internal-ip-only flag	18 days ago
hyperv	Merge pull request #2932 from sipicsg/master	a month ago
none	Another pass at linting	4 months ago
openstack	Revert "Rackspace/Openstack - Add/enhance --net-* create switches."	22 days ago
rackspace	Revert "Rackspace/Openstack - Add/enhance --net-* create switches."	22 days ago
softlayer	FIX #3056 Is timeout when instance is stopped	a month ago
virtualbox	Merge pull request #3124 from StefanScherer/set-nictype1	4 days ago
vmwarefusion	Updated VMWare Fusion driver to attempt to bind mount the share direc...	17 days ago
vmwarevcloudair	FIX #3056 Is timeout when instance is stopped	a month ago
vmwarevsphere	Default to port 443 for all vsphere actions, not just creation	11 days ago

看见了吗，这就是 docker machine 支持的所有的虚拟主机类型，我们不难看出从基本的 virtualbox 到 azure，再到 aws，都在其支持范围内。换句话说，只要你跟据 driver 的要求填入参数，你可以通过一个命令创建云主机，一个命令动态扩展主机。Docker-machine 会给 VM 安装一个操作系统，默认的本地系统格式 boot2docker，远端的系统是 ubuntu12.04。

安装 Docker Machine 的步骤如下：

1. 安装 docker binary。如果在 windows 或者 mac 下需要安装 docker toolbox。
2. 下载 Docker Machine，如果是 mac 下面的话，运行：

```
$ curl -L
https://github.com/docker/machine/releases/download/v0.6.0/docker-machine-`uname
-s`-`uname -m`> /usr/local/bin/docker-machine && \ chmod +x /usr/local/bin/docker-machine
```

如果是 windows 下面的话，需要先下一个 git bash，然后在里面运行：

```
$ if[ ! -d"$HOME/bin"]; then mkdir -p "$HOME/bin"; fi && \ curl -L
```

```
https://github.com/docker/machine/releases/download/v0.6.0/docker-machine-Windows-x86_64.exe > "$HOME/bin/docker-machine.exe"&& \ chmod +x "$HOME/bin/docker-machine.exe"
```

3. 安装完毕之后，检查版本

```
$ docker-machine version
docker-machine version 0.6.0, build 61388e9
```

2 Hello Docker Machine

本机可以作为一个实验，跟着我们一起运行，我们将一步一步的使用 Docker Machine 来创建一个 nginx web 服务。在进行实验之前，首先需要确认本机安装了 VirtualBox，VirtualBox 是一个免费的本地虚拟化运行程序，由 Oracle 开发，类似于 VMWare Workstation。我们先来看一下 Docker Machine 都有哪些参数：

```
$ docker-machine --help
Usage: docker-machine.exe [OPTIONS] COMMAND [arg...]

Create and manage machines running Docker.

Version: 0.6.0, build e27fb87

...
Commands:
  active          Print which machine is active
  config          Print the connection config for machine
  create          Create a machine
  env            Display the commands to set up the environment for the Docker client
  inspect        Inspect information about a machine
  ip             Get the IP address of a machine
  kill           Kill a machine
  ls            List machines
  provision      Re-provision existing machines
  regenerate-certs  Regenerate TLS Certificates for a machine
  restart       Restart a machine
  rm            Remove a machine
  ssh          Log into or run a command on a machine with SSH.
  scp         Copy files between machines
  start       Start a machine
  status      Get the status of a machine
  stop       Stop a machine
  upgrade    Upgrade a machine to the latest version of Docker
  url       Get the URL of a machine
```



```
version          Show the Docker Machine version or a machine docker version
help            Shows a list of commands or help for one command
```

Run 'docker-machine.exe COMMAND --help' for more information on a command.

我们主要关注 create, start, stop, ssh 等命令，顾名思义就是创建 machine，启动，ssh 进入。下面开始我们正式实验：

1. 创建一个 machine，并查看是否成功

```
$ docker-machine create --driver virtualbox m1
Running pre-create checks...
Creating machine...
(m1) Copying C:\Users\Administrator\.docker\machine\cache\boot2docker.iso to
C:\Users\Administrator\.docker\machine\machines\m1\boot2docker.iso...
(m1) Creating VirtualBox VM...
(m1) Creating SSH key...
(m1) Starting the VM...
(m1) Check network to re-create if needed...
(m1) Waiting for an IP...
Waiting for machine to be running, this may take a few minutes...
Detecting operating system of created instance...
Waiting for SSH to be available...
Detecting the provisioner...
Provisioning with boot2docker...
Copying certs to the local machine directory...
Copying certs to the remote machine...
Setting Docker configuration on the remote daemon...
Checking connection to Docker...
Docker is up and running!

To see how to connect your Docker Client to the Docker Engine running on this virtual machine,
run: C:\Users\Administrator\bin\docker-machine.exe env m1
```

查看

```
$ docker-machine ls
```

NAME	ACTIVE	DRIVER	STATE	URL	SWARM	DOCKER	ERRORS
m1	-	virtualbox	Running	tcp://192.168.99.103:2376		v1.10.3	
mh-keystore	-	virtualbox	Running	tcp://192.168.99.100:2376		v1.10.3	
mhs-demo0	-	virtualbox	Running	tcp://192.168.99.101:2376	mhs-demo0 (master)	v1.10.3	
mhs-demo1	-	virtualbox	Running	tcp://192.168.99.102:2376	mhs-demo0	v1.10.3	

2. 在 machine 上启动一个 nginx 容器

```
$ eval $(docker-machine env m1) #绑定 m1 的 TLS 证书，之后操作 docker-machine 就像操作本地 docker 一样了
```

```
$ docker run -itd -p 80:80 nginx
005cc090710a7c4bc5681db6cc2c2916fbddf7d2de6f602994c0b7b12824821a
```

3. 访问 nginx 服务

```
$ docker-machine ip m1
192.168.99.103

Administrator@EKEUSER-PC MINGW64 /
$ curl 192.168.99.103
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  612    100    612    0     0  38250      0 --:--:-- --:--:-- --:--:-- 38250<!DOCTYPE
html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
  body {
    width: 35em;
    margin: 0 auto;
    font-family: Tahoma, Verdana, Arial, sans-serif;
  }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

在上例中，我们通过 `eval` 命令来进行 `docker` 的操作切换，本质是将相应的证书和环境变量导入到 `docker client` 去。

3 Docker Machine 操作云端主机

之前说过 Docker Machine 可以控制远端的主机，说到远端的主机，那么云主机就是一个标准的远程主机，你打开 docker machine 的开源代码，在 driver 目录中，能够清晰的看见一些内嵌的 driver，比如 Digital Ocean, AWSEC2 等等。

3.1 Digital Ocean

Digital Ocean 是国外除了 AWS 外的第二大公有云厂商，在它的平台商，运行了超过 1000 多万台主机。我们可以通过下面的命令创建一台 Digital Ocean 主机。

```
$ docker-machine create --driver digitalocean --digitalocean-access-token xxxxx  
docker-sandbox
```

其中--digitalocean-access-token 是用户的认证信息。

3.2 AWS EC2

AWS 是全球最大的公有云厂商，同样我们可以通过 docker-machine 来快速生成运行 docker 的主机：

```
$ docker-machine create --driver amazonec2 --amazonec2-access-key AKI*****  
--amazonec2-secret-key 8T93C***** aws-sandbox
```

其中 access-key 和 secret-key 都是用户的认证信息。

3.3 第三方 plugin

我们在系统默认的 plugin 里面是找不到国内的，但是国内的阿里云和 UCloud 已经开发出了插件，不过你需要将 plugin 编译进 docker-machine，具体可以参照：

```
https://github.com/denverdino/docker-machine-driver-alyunecs  
https://github.com/ucloud/docker-machine-ucloud
```

第二节 Docker compose

Docker-engine 解决的是如何运行单个容器的问题，随着容器的轻量级，势必会出现大量的容器，容器与容器之间如何管理，如何联系起来就是迫切需要解决的问题，基于此 Docker Compose 就应运而生了。Compose 是一个开源项目，其地址是 <https://github.com/docker/compose>。首先我们要来看一下 Compose 到底是要解决什么问题，什么他能解决，什么他不能解决。

- Compose 是一个处理容器与容器之间编排的工具
- 初期 compose 只是一个处理单主机的工具，不过最新的发展是它可以和 swarm 一起使用。
- Compose 适合开发和测试环境

我们可以将 compose 组装起来的环境称为 app，通常我们使用 compose 的流程是：

1. 在你的环境中定义好 Dockerfile 或生成 Image，以保证可以重复使用
2. 定义 docker-compose.yml，保证其可以在多个主机上使用
3. 通过 docker-compose up 来启动 app
4. 将 yml 分享到不同环境使用

下面是一个简单的 compose 的配置文件：

```
version: '2'
services:
  web:
    build: .
    ports:
      - "5000:5000"
    volumes:
      - ../code
      - logvolume01:/var/log
    links:
      - redis
  redis:
    image: redis
    volumes:
      logvolume01: {}
```

具体的含义我们先不讲，不过一看上去就跟我们之前的 docker run 命令很类似。确实 docker compose 本身就可以取代 docker 命令，如果你厌倦了很长的命

令，完全可以用 `compose` 来使运行命令更清晰，更有层次。

1 安装 `compose`

`Compose` 安装非常简单，只需要两行语句即可：

```
$ curl -L
https://github.com/docker/compose/releases/download/1.6.2/docker-compose-`uname
-s`-`uname -m`> /usr/local/bin/docker-compose

$ chmod+x/usr/local/bin/docker-compose
```

安装是从 `github` 上去下载，然后将 `docker-compose` 添加可执行权限。安装成功后执行下面的命令，检查是否成功：

```
root@ghostcloud:~# docker-compose --version
docker-compose version 1.6.2, build 4d72027
```

2 hello `compose`

本小节我们我们将通过一个例子来实际学习一下 `compose`，我们将搭建一个简单的 `python` web 应用，这个应用使用 `Flask framework`，应用中会操作 `Redis` 数据库。对于这个部分，如果你要跟着一一起测试，最好是通过 `docker-machine`，然后在外层挂接 `VPN`，否则很可能拉取不了镜像。

2.1 创建工程

1. 我们创建一个文件夹，用于存放工程文件

```
root@ghostcloud:~# mkdir composetest
root@ghostcloud:~# cd composetest/
root@ghostcloud:~/composetest#
```

2. 创建一个文件 `app.py`

```
from flask import Flask
from redis import Redis

app = Flask(__name__)
redis = Redis(host='redis', port=6379)

@app.route('/')
```

```
def hello():
    redis.incr('hits')
    return 'Hello World! I have been seen %s times.' % redis.get('hits')

if __name__ == "__main__":
    app.run(host="0.0.0.0", debug=True)
```

3. 创建一个 requirements.txt

```
flask
redis
```

2.2 创建一个 Docker Image

在这一步，我们将通过 dockerfile 来构建 image

1. 创建一个 Dockerfile

```
FROM python:2.7
ADD . /code
WORKDIR /code
RUN pip install -r requirements.txt
CMD python app.py
```

上面的描述，会执行以下步骤：

- 新的 image 集成自 python2.7
- 将当前目录的文件，添加到镜像的/code
- 设置工作目录为/code
- 安装 python 的依赖库
- 设置镜像的默认命令为 python app.py

2. 编译 image

```
root@ghostcloud:~/composetest# docker build -t web .
Sending build context to Docker daemon 3.072 kB
Step 1 : FROM python:2.7
Pulling repository docker.io/library/python
```

当 build 执行完毕的时候，一个名为 web 的镜像就产生了，同时下载完毕了所有的依赖库。

2.3 定义 compose.yml

创建一个 docker-compose.yml

```
version: '2'
services:
  web:
    build: .
    ports:
      - "5000:5000"
    volumes:
      - ./code
    depends_on:
      - redis
  redis:
    image: redis
```

这个 yml 干了下面几件事:

- 编译 Dockerfile
- 将主机的 5000 端口转发到容器中
- 将当前目录映射到/code 中，便于修改代码
- 同时配置了一个 redis 数据库，作为依赖

2.4 编译并运行 Compose

之前的工作，都是为了下面这一句命令：

```
$ docker-compose up
Creating composetest_redis_1
Creating composetest_web_1
Attaching to composetest_redis_1, composetest_web_1
redis_1 | 1:C 24 Mar 12:18:20.257 # Warning: no config file specified, using the default
config. In order to specify a config file use redis-server /path/to/redis.conf
redis_1 |
redis_1 |
redis_1 | Redis 3.0.7 (00000000/0) 64 bit
redis_1 |
redis_1 | Running in standalone mode
redis_1 | Port: 6379
redis_1 | PID: 1
redis_1 |
```

```

redis_1 | | \_-'-'-'_ _-'-'-' | http://redis.io
redis_1 | \_-'-'-'_ _-'-'-'
redis_1 | | \_-'-'-'_ _-'-'-' |
redis_1 | | \_-'-'-'_ _-'-'-' |
redis_1 | \_-'-'-'_ _-'-'-'
redis_1 | \_-'-'-'_ _-'-'-'
redis_1 | \_-'-'-'_ _-'-'-'
redis_1 | \_-'-'-'_ _-'-'-'
redis_1 | \_-'-'-'_ _-'-'-'

redis_1 | 1:M 24 Mar 12:18:20.259 # WARNING: The TCP backlog setting of 511 cannot be enforced
because /proc/sys/net/core/somaxconn is set to the lower value of 128.

redis_1 | 1:M 24 Mar 12:18:20.259 # Server started, Redis version 3.0.7

redis_1 | 1:M 24 Mar 12:18:20.259 # WARNING overcommit_memory is set to 0! Background save
may fail under low memory condition. To fix this issue add 'vm.overcommit_memory = 1' to
/etc/sysctl.conf and then reboot or run the command 'sysctl vm.overcommit_memory=1' for this
to take effect.

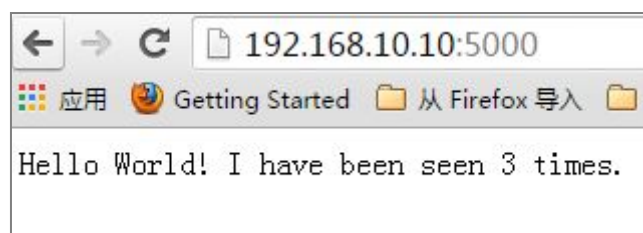
redis_1 | 1:M 24 Mar 12:18:20.259 # WARNING you have Transparent Huge Pages (THP) support
enabled in your kernel. This will create latency and memory usage issues with Redis. To fix
this issue run the command 'echo never > /sys/kernel/mm/transparent_hugepage/enabled' as root,
and add it to your /etc/rc.local in order to retain the setting after a reboot. Redis must
be restarted after THP is disabled.

redis_1 | 1:M 24 Mar 12:18:20.259 * The server is now ready to accept connections on port
6379

web_1 | * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
web_1 | * Restarting with stat
web_1 | * Debugger is active!
web_1 | * Debugger pin code: 105-865-067

```

接下来，你可以通过 `http://<ip>:5000` 来访问页面，每访问一次，页面计数就会增加。



Compose 的其他命令，这里就不做多的介绍了，可以通过下面命令查询：

```
$ docker-compose --help
```


3 Compose file

Compose file 的格式为 YAML，主要用来定义 services, networks 和 volumes。默认的路径是在当前文件夹下的 docker-compose.yml。服务是容器的集合，它里面可以包含一个或多个容器。Services 对应的是 docker run，networks 对应的是 docker network create，volumes 对应的是 docker volume create。下面我们就挨个讲一下 compose 的各个配置参数。

3.1 build

配置的参数是在 build 时启用的，build 后面可以直接跟一个路径，也可以在它的子属性里面设置 context、dockerfile 和 args:

```
build: ./dir

build:
  context: ./dir
  dockerfile: Dockerfile-alternate
  args:
    buildno: 1
```

- context：存放 Dockerfile 的路径
- dockerfile: 默认是 Dockerfile，可以指定为其他文件
- args: 编译时候用到的参数，只支持 key-value 形式

3.2 cap_add, cap_drop

Capabilities 是系统自带的权限控制的功能，你可以通过 man 7 capabilities 来查看所有的 capabilities。

3.3 command

覆盖镜像默认的命令

```
command: bundle execthin -p 3000
```

3.4 cgroup_parent

设置 cgroup 继承自什么

```
cgroup_parent:m-executor-abcd
```

3.5 container_name

设置容器的名字，而不是使用默认名

```
container_name:my-web-container
```

3.6 devices

类似于--device 命令：

```
devices:  
  - "/dev/ttyUSB0:/dev/ttyUSB0"
```

3.7 depends_on

设置服务的依赖，docker-compose up 的时候会自动根据依赖关系启动。

```
version: '2'  
services:  
  web:  
    build: .  
    depends_on:  
      - db  
      - redis  
  redis:  
    image: redis  
  db:  
    image: postgres
```

3.8 dns

设置 DNS，比如：

```
dns: 8.8.8.8  
dns:  
  - 8.8.8.8  
  - 9.9.9.9
```

3.9 dns_search

设置 dns 的 search domains:

```
dns_search: example.com

dns_search:
- dc1.example.com
- dc2.example.com
```

3.10 entrypoint

覆盖默认的 entrypoint:

```
entrypoint: /code/entrypoint.sh
```

同时也接受列表作为参数:

```
entrypoint:
- php
- -d
- zend_extension=/usr/local/lib/php/extensions/no-debug-non-zts-20100525/xdebug.so
- -d
- memory_limit=-1
- vendor/bin/phpunit
```

3.10 env_file

可以通过文件来设置环境变量:

```
env_file: .env

env_file:
- ./common.env
- ./apps/web.env
- /opt/secrets.env
```

3.11 environment

Docker 命令一个比较麻烦的地方就是当环境变量很多时, 命令就会非常长。

通过 YML 设置环境变量就会非常清晰:

```
environment:
  RACK_ENV: development
  SHOW: 'true'
  SESSION_SECRET:
```

```
environment:
  - RACK_ENV=development
  - SHOW=true
  - SESSION_SECRET
```

3.12 expose

容器要导出的端口，如：

```
expose:
  - "3000"
  - "8000"
```

3.13 extends

Extend 用于扩展其他的 services，比如别的 yml 中：

```
extends:
  file: common.yml
  service: webapp
```

3.14 external_links

链接外部容器，这个可以链接其他 compose file 里面的容器，也可以链接非 compose 的容器，如：

```
external_links:
  - redis_1
  - project_db_1:mysql
  - project_db_1:postgresql
```

3.15 extra_hosts

和--add-host 功能相同，如：

```
extra_hosts:
  - "somehost:162.242.195.82"
  - "otherhost:50.31.209.229"
```

3.16 image

指定容器启动时的 image，如：

```
image: redis
image: ubuntu:14.04
image: tutum/influxdb
image: example-registry.com:4000/postgresql
image: a4bc65fd
```

如果镜像不存在，compose 会自动去下载镜像。

3.17 labels

为容器添加特殊的标记，如：

```
Labels:
  com.example.description: "Accounting webapp"
  com.example.department: "Finance"
  com.example.label-with-empty-value: ""

Labels:
  - "com.example.description=Accounting webapp"
  - "com.example.department=Finance"
  - "com.example.label-with-empty-value"
```

3.18 links

链接到其他的 service，如：

```
web:
  links:
    - db
    - db:database
    - redis
```

3.19 logging

配置容器的日志形式，如：

```
Logging:
  driver: syslog
  options:
    syslog-address: "tcp://192.168.0.42:123"
```

支持的日志类型有：

```
driver: "json-file"
driver: "syslog"
driver: "none"
```

3.20 network_mode

和--net 参数类似，所有的 5 种方式为：

```
network_mode: "bridge"
network_mode: "host"
network_mode: "none"
network_mode: "service:[service name]"
network_mode: "container:[container name/id]"
```

3.21 networks

设置容器要加入的网络：

```
networks:
  - some-network
  - other-network
```

3.22 pid

使用主机的 PID，类似于--pid，如：

```
pid: "host"
```

3.23 ports

容器导出的端口配置，如：

```
ports:
  - "3000"
  - "3000-3005"
  - "8000:8000"
  - "9090-9091:8080-8081"
  - "49100:22"
  - "127.0.0.1:8001:8001"
  - "127.0.0.1:5000-5010:5000-5010"
```

3.24 volumes

使用主机的卷，如：

```
volumes:
  # Just specify a path and let the Engine create a volume
  - /var/lib/mysql

  # Specify an absolute path mapping
  - /opt/data:/var/lib/mysql

  # Path on the host, relative to the Compose file
  - ./cache:/tmp/cache

  # User-relative path
  - ~/configs:/etc/configs/:ro

  # Named volume
  - datavolume:/var/lib/mysql
```

3.25 volumes_from

Mount 其他 service 或容器的卷，如：

```
volumes_from:
  - service_name
  - service_name:ro
  - container:container_name
  - container:container_name:rw
```

4 控制容器启动顺序

容器的启动顺序，主要是通过 `depends_on` 来的，但是也会依赖于 `links`, `volumes_from`, `network_mode`:”services:...”。

但是，`compose` 本身只控制启动顺序，并不知道容器什么时候是可用的。主要原因是，目前 `compose` 并没有一个通知的机制，而且每一种应用启动成功的标志并不相同。因此，用户需要自行对这些有状态的信息进行维护。同样，当容器倒下的时候，其他关联的容器也要去考虑重连。基于此，最好的方式是设计成无

状态协议，如果一定要用有状态协议，那么一定要做容错处理，这也是微服务架构的准则。

