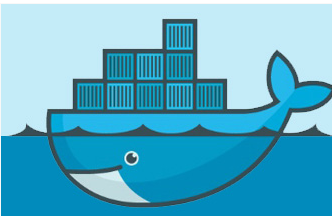
**Docker**

## 一、基础知识

**1 什么是Docker?**

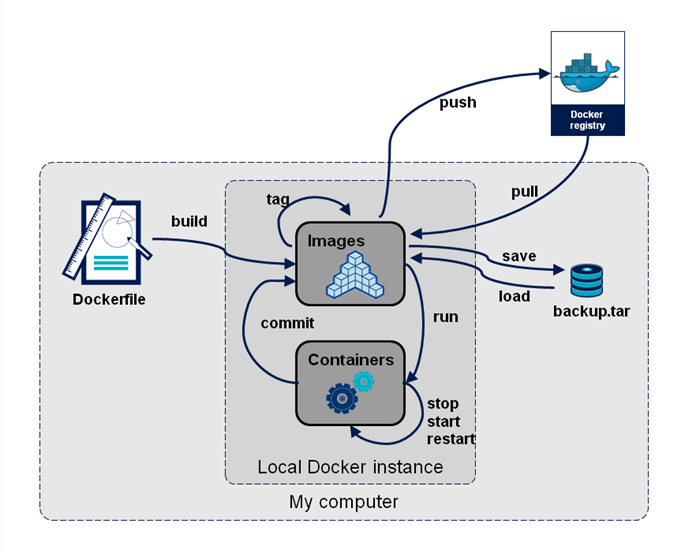
Docker 是一个开源的应用容器引擎，基于 Go 语言 并遵从Apache2.0协议开源。**可以让开发者打包他们的应用以及依赖包到一个轻量级、可移植的容器中，然后发布到任何流行的 Linux 机器上，也可以实现虚拟化。**容器是完全使用沙箱机制，相互之间不会有任何接口（类似 iPhone 的 app）,更重要的是容器性能开销极低。Docker 是一个开源的应用容器引擎，让开发者可以打包他们的应用以及依赖包到一个可移植的容器中，然后发布到任何流行的 Linux 机器上，也可以实现虚拟化。容器是完全使用沙箱机制，相互之间不会有任何接口。



Docker是基于Go语言实现的云开源项目，诞生于2013年初，最初发起者是dotCloud公司。Docker自开源后就受到广泛的关注和讨论，目前已有多个相关项目，逐渐形成了围绕Docker的生态体系。dotCloud公司后来也改名为Docker Inc，专注于Docker相关技术和产品的开发。

现在主流的Linux操作系统都已经支持Docker。例如，Redhat RHEL 6.5/ CentOS 6.5往上的操作系统、Ubuntu 14.04操作系统，都已经默认带有Docker软件包。Google公司宣称在其PaaS平台及服务产品中广泛应用了Docker。微软公司宣布和 Docker公司合作，以加强其云平台Azure对Docker的支持。公有云提供商亚马逊也推出了AWS EC2 Container，提供对Docker的支持。

Docker的英文本意是“搬运工”，在程序员的世界里，Docker搬运的是集装箱（Container），集装箱里装的是任意类型的App，开发者通过Docker可以将App变成一种标准化的、可移植的、自管理的组件，可以在任何主流系统中开发、调试和运行。最重要的是，它不依赖于任何语言、框架或系统。不久前Docker 1.0的发布，意味着Docker自身已经转变为一个分发应用的开放平台。如今的Docker已经备受青睐，云服务提供商，包括微软、 IBM 、 Rackspace 、 Google 以及其他主要的 Linux 提供商如 Canonical 和 Red Hat ，都已经开始支持 Docker 。



**2 Docker的应用场景**

（1）Web 应用的自动化打包和发布。

（2）自动化测试和持续集成、发布。

（3）在服务型环境中部署和调整数据库或其他的后台应用。

（4）从头编译或者扩展现有的OpenShift或Cloud Foundry平台来搭建自己的PaaS环境。

**3 Docker 的优点**

**（1）简化程序**

Docker 让开发者可以打包他们的应用以及依赖包到一个可移植的容器中，然后发布到任何流行的 Linux 机器上，便可以实现虚拟化。Docker改变了虚拟化的方式，使开发者可以直接将自己的成果放入Docker中进行管理。方便快捷已经是 Docker的最大优势，过去需要用数天乃至数周的 任务，在Docker容器的处理下，只需要数秒就能完成。

**（2）避免选择恐惧症**

如果你有选择恐惧症，还是资深患者。Docker 帮你 打包你的纠结！比如 Docker 镜像；Docker 镜像中包含了运行环境和配置，所以 Docker 可以简化部署多种应用实例工作。比如 Web 应用、后台应用、数据库应用、大数据应用比如 Hadoop 集群、消息队列等等都可以打包成一个镜像部署。

**（3）节省开支**

一方面，云计算时代到来，使开发者不必为了追求效果而配置高额的硬件，Docker 改变了高性能必然高价格的思维定势。Docker 与云的结合，让云空间得到更充分的利用。不仅解决了硬件管理的问题，也改变了虚拟化的方式。

**1、Docker 容器的启动可以在秒级实现，这相比传统的虚拟机方式要快得多**

**2、Docker 对系统资源的利用率很高，一台主机上可以同时运行数千个 Docker 容器。**

**3、更快速的交付和部署、更轻松的迁移和扩展**

**4 Docker 组件与元素**

Docker有三个组件和三个基本元素，读者可以快速浏览下面这个视频来了解这些组建和元素，以及它们的关系。

**（1）三个组件分别是：**

Docker Client 是用户界面，它支持用户与Docker Daemon之间通信。

Docker Daemon运行于主机上，处理服务请求。

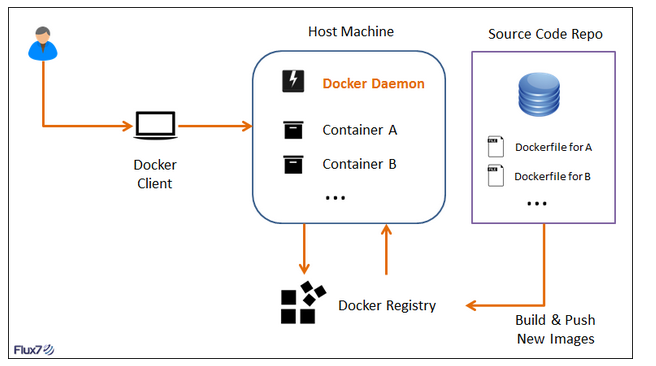
Docker Index是中央registry，支持拥有公有与私有访问权限的Docker容器镜像的备份。

**（2）三个基本要素分别是：**

Docker Containers负责应用程序的运行，包括操作系统、用户添加的文件以及元数据。

Docker Images是一个只读模板，用来运行Docker容器。

DockerFile是文件指令集，用来说明如何自动创建Docker镜像。



## **5 为什么是 Docker 而不是纯 LXC**

## 几年前当我还在 Adcloud 的时候，开始使用 LXC 来建立所有的微型服务。鉴于我在台式机上使用 Archlinux，效果还不错。但是我们的生产环境是 Ubuntu 的操作系统，并且会用 chef 脚本来创建容器，而我仍然把这些容器当做机器来维护。这些容器有自己的 DNS 域名、 IP 地址、 SSH 服务，和所有文件的磁盘绑定。所以每当我开始一个项目时，都得先通过 SSH 登录到机器然后再开始工作。并且我一点也不喜欢 Chef ，所以离开 Adcloud 后我就放弃 Chef ，开始使用 Babushka 。虽然后者能让我以最简单的方式来部署一个单机环境，但是创建新容器十分缓慢，容器起来以后占据了我 SSD 上的大量磁盘空间，并且我还需要手动去配置多个服务。所以当我第一次启动 Docker 容器的时候，它的速度之快真是让人惊讶，更棒的是分层的文件系统可以减少磁盘的使用

## **6 容器不等于机器**

## 有了 Docker 容器以后，你就不再需要考虑机器的概念了。它们只是在不同的内核命名空间中的进程，你希望它们可以保持精简和干净。它们也需要能够运行在不同的环境中，所以我不再在容器中使用 SSH 。那我又是如何来创建我的项目呢？首先我创建了一系列容器，在上面安装了我项目中需要安装但不应该安装在本地的工具。

## **7 Docker本领**

## 使用Linux container，使得"VM"的创建和销毁在秒级就能完成。由于只是做了网络和进程的隔离，"VM"的运行几乎没有overhead。

## 使用AUFS，可以以递进的方式创建"VM" —— 一个"VM"叠在另一个"VM"上，就像使用git增量开发一样。

## 软件的运行环境（image）和软件本身（container）分离，和数据也分离。

**8 Docker与虚拟机比较**

**作为一种轻量级的虚拟化方式Docker在运行应用上跟传统的虚拟机方式相比有显著优势：**

@Docker容器很快，启动和停止可以在秒级实现，这相比传统的虚拟机方式要快得多。

@Docker容器对系统资源需求很少，一台主机上可以同时运行数千个Docker容器。

@Docker通过类似Git的操作来方便用户获取、分发和更新应用镜像，指令简明，学习成本较低。

@Docker通过Dockerfile配置文件来支持灵活的自动化创建和部署机制，提高工作效率。



## 

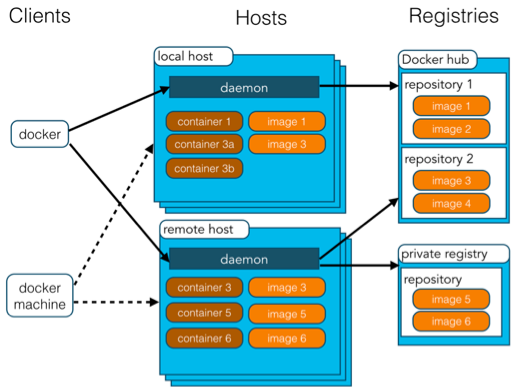
## 二、Docker 架构

**1 Docker 使用客户端-服务器 (C/S) 架构模式**，使用远程API来管理和创建Docker容器。

Docker 容器通过 Docker 镜像来创建。容器与镜像的关系类似于面向对象编程中的对象与类。

**2 容器与镜像**

|  |  |
| --- | --- |
| **Docker** | **面向对象** |
| 容器 | 对象 |
| 镜像 | 类 |



|  |  |
| --- | --- |
| Docker 镜像(Images) | Docker 镜像是用于创建 Docker 容器的模板。 |
| Docker 容器(Container) | 容器是独立运行的一个或一组应用。 |
| Docker 客户端(Client) | Docker 客户端通过命令行或者其他工具使用 Docker API ([https://docs.docker.com/reference/api/docker\_remote\_api](https://docs.docker.com/reference/api/docker_remote_api" \t "http://www.runoob.com/docker/_blank)) 与 Docker 的守护进程通信。 |
| Docker 主机(Host) | 一个物理或者虚拟的机器用于执行 Docker 守护进程和容器。 |
| Docker 仓库(Registry) | Docker 仓库用来保存镜像，可以理解为代码控制中的代码仓库。  Docker Hub([https://hub.docker.com](https://hub.docker.com" \t "http://www.runoob.com/docker/_blank)) 提供了庞大的镜像集合供使用。 |
| Docker Machine | Docker Machine是一个简化Docker安装的命令行工具，通过一个简单的命令行即可在相应的平台上安装Docker，比如VirtualBox、 Digital Ocean、Microsoft Azure。 |

## 三、CentOS Docker 安装

**1 Docker支持以下的CentOS版本**

CentOS 7 (64-bit)

CentOS 6.5 (64-bit) 或更高的版本

**2 前提条件**

目前，CentOS 仅发行版本中的内核支持 Docker。

Docker 运行在 CentOS 7 上，要求系统为64位、系统内核版本为 3.10 以上。

Docker 运行在 CentOS-6.5 或更高的版本的 CentOS 上，要求系统为64位、系统内核版本为 2.6.32-431 或者更高版本。

**3 使用 yum 安装（CentOS 7下）**

1. **Docker 要求 CentOS 系统的内核版本高于 3.10**

查看本页面的前提条件来验证你的CentOS 版本是否支持 Docker 。

通过 uname -r 命令查看你当前的内核版本

[root@runoob ~]# uname -r 3.10.0-327.el7.x86\_64

**（2）安装 Docker**

从 2017 年 3 月开始 docker 在原来的基础上分为两个分支版本: Docker CE 和 Docker EE。

Docker CE 即社区免费版，Docker EE 即企业版，强调安全，但需付费使用。

本文介绍 Docker CE 的安装使用。

**（3）移除旧的版本**

$ sudo yum remove docker \

docker-client \

docker-client-latest \

docker-common \

docker-latest \

docker-latest-logrotate \

docker-logrotate \

docker-selinux \

docker-engine-selinux \

docker-engine

**（4）安装一些必要的系统工具**

sudo yum install -y yum-utils device-mapper-persistent-data lvm2

**（5）添加软件源信息**

sudo yum-config-manager --add-repo http://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo

**（6）更新 yum 缓存**

sudo yum makecache fast

**（7）安装 Docker-ce**

sudo yum -y install docker-ce

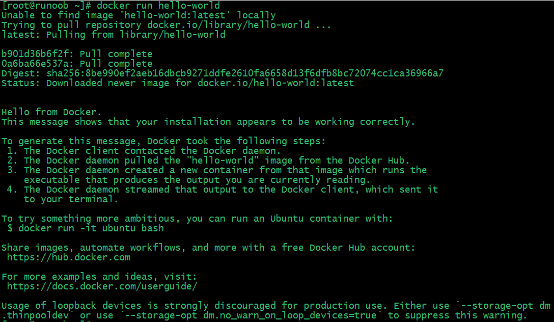
**（8）启动 Docker 后台服务**

sudo systemctl start docker

**（9）测试运行 hello-world**

[root@runoob ~]# docker run hello-world

由于本地没有hello-world这个镜像，所以会下载一个hello-world的镜像，并在容器内运行。



**4 使用脚本安装 Docker**

（1）使用 sudo 或 root 权限登录 Centos。

（2）确保 yum 包更新到最新。

$ sudo yum update

（3）执行 Docker 安装脚本。

$ curl -fsSL https://get.docker.com -o get-docker.sh

$ sudo sh get-docker.sh

执行这个脚本会添加 docker.repo 源并安装 Docker。

（4）启动 Docker 进程。

sudo systemctl start docker

（5）验证 docker 是否安装成功并在容器中执行一个测试的镜像。

$ sudo docker run hello-world

docker ps

到此，Docker 在 CentOS 系统的安装完成。

**5 镜像加速**

鉴于国内网络问题，后续拉取 Docker 镜像十分缓慢，我们可以需要配置加速器来解决，我使用的是网易的镜像地址：http://hub-mirror.c.163.com。新版的 Docker 使用 /etc/docker/daemon.json（Linux） 或者 %programdata%\docker\config\daemon.json（Windows） 来配置 Daemon。请在该配置文件中加入（没有该文件的话，请先建一个）：

{

"registry-mirrors": ["http://hub-mirror.c.163.com"]

}

**6 删除 Docker CE**

执行以下命令来删除 Docker CE：

$ sudo yum remove docker-ce

$ sudo rm -rf /var/lib/docker

## 三、Docker实例讲解

**1 Docker Hello world**

**（1）Docker 允许你在容器内运行应用程序**， 使用 docker run 命令来在容器内运行一个应用程序。

输出Hello world

runoob@runoob:~$ docker run ubuntu:15.10 /bin/echo "Hello world"

Hello world



**（2）各个参数解析：**

docker: Docker 的二进制执行文件。

run:与前面的 docker 组合来运行一个容器。

ubuntu:15.10指定要运行的镜像，Docker首先从本地主机上查找镜像是否存在，如果不存在，Docker 就会从镜像仓库 Docker Hub 下载公共镜像。

/bin/echo "Hello world": 在启动的容器里执行的命令

以上命令完整的意思可以解释为：Docker 以 ubuntu15.10 镜像创建一个新容器，然后在容器里执行 bin/echo "Hello world"，然后输出结果。

**（3）运行交互式的容器**

我们通过docker的两个参数 -i -t，让docker运行的容器实现"对话"的能力

runoob@runoob:~$ docker run -i -t ubuntu:15.10 /bin/bash

root@dc0050c79503:/#

**（4）各个参数解析：**

-t:在新容器内指定一个伪终端或终端。

-i:允许你对容器内的标准输入 (STDIN) 进行交互。

此时我们已进入一个 ubuntu15.10系统的容器

我们尝试在容器中运行命令 cat /proc/version和ls分别查看当前系统的版本信息和当前目录下的文件列表

IMG_256

我们可以通过运行exit命令或者使用CTRL+D来退出容器。

（**5）启动容器（后台模式）**

使用以下命令创建一个以进程方式运行的容器

runoob@runoob:~$ docker run -d ubuntu:15.10 /bin/sh -c "while true; do echo hello world; sleep 1; done"

2b1b7a428627c51ab8810d541d759f072b4fc75487eed05812646b8534a2fe63

在输出中，我们没有看到期望的"hello world"，而是一串长字符

2b1b7a428627c51ab8810d541d759f072b4fc75487eed05812646b8534a2fe63

这个长字符串叫做容器ID，对每个容器来说都是唯一的，我们可以通过容器ID来查看对应的容器发生了什么。

首先，我们需要确认容器有在运行，可以通过 docker ps 来查看

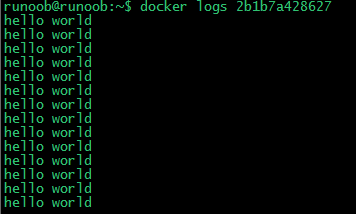
runoob@runoob:~$ docker ps

CONTAINER ID:容器ID

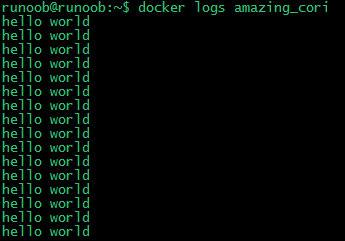
NAMES:自动分配的容器名称

在容器内使用docker logs命令，查看容器内的标准输出

runoob@runoob:~$ docker logs 2b1b7a428627



runoob@runoob:~$ docker logs amazing\_cori



**（6）停止容器**

我们使用 docker stop 命令来停止容器:

IMG_256

通过docker ps查看，容器已经停止工作:

runoob@runoob:~$ docker ps

也可以用下面的命令来停止:

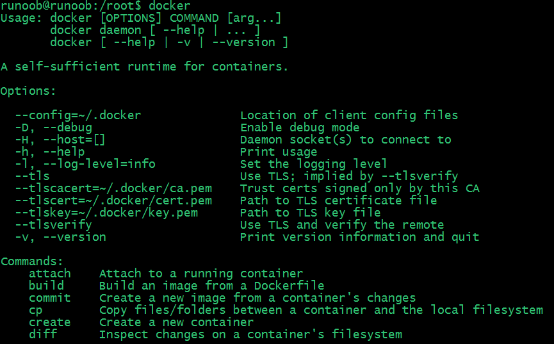
runoob@runoob:~$ docker stop amazing\_cori

**2 Docker 容器使用**

**（1）Docker 客户端**

docker 客户端非常简单 ,我们可以直接输入 docker 命令来查看到 Docker 客户端的所有命令选项。

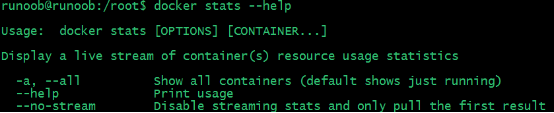
runoob@runoob:~# docker



可以通过命令 docker command --help 更深入的了解指定的 Docker 命令使用方法。

例如我们要查看 docker stats 指令的具体使用方法：

runoob@runoob:~# docker stats --help



**（2）运行一个web应用**

前面我们运行的容器并没有一些什么特别的用处。

接下来让我们尝试使用 docker 构建一个 web 应用程序。

我们将在docker容器中运行一个 Python Flask 应用来运行一个web应用。

runoob@runoob:~# docker pull training/webapp # 载入镜像

runoob@runoob:~# docker run -d -P training/webapp python app.py

IMG_256

参数说明:

-d:让容器在后台运行。

-P:将容器内部使用的网络端口映射到我们使用的主机上。

**（3）查看 WEB 应用容器**

使用 docker ps 来查看我们正在运行的容器：

runoob@runoob:~# docker ps

CONTAINER ID IMAGE COMMAND ... PORTS

d3d5e39ed9d3 training/webapp "python app.py" ... 0.0.0.0:32768->5000/tcp

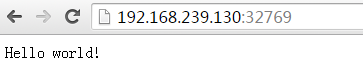
这里多了端口信息。

PORTS

0.0.0.0:32769->5000/tcp

Docker 开放了 5000 端口（默认 Python Flask 端口）映射到主机端口 32769 上。

这时我们可以通过浏览器访问WEB应用



我们也可以通过 -p 参数来设置不一样的端口：

runoob@runoob:~$ docker run -d -p 5000:5000 training/webapp python app.py

docker ps查看正在运行的容器

runoob@runoob:~# docker ps

CONTAINER ID IMAGE PORTS NAMES

bf08b7f2cd89 training/webapp ... 0.0.0.0:5000->5000/tcp wizardly\_chandrasekhar

d3d5e39ed9d3 training/webapp ... 0.0.0.0:32768->5000/tcp xenodochial\_hoov

容器内部的 5000 端口映射到我们本地主机的 5000 端口上。

**（4）网络端口的快捷方式**

通过 docker ps 命令可以查看到容器的端口映射，docker 还提供了另一个快捷方式 docker port，使用 docker port 可以查看指定 （ID 或者名字）容器的某个确定端口映射到宿主机的端口号。

上面我们创建的 web 应用容器 ID 为 bf08b7f2cd89 名字为 wizardly\_chandrasekhar。

我可以使用 docker port bf08b7f2cd89 或 docker port wizardly\_chandrasekhar 来查看容器端口的映射情况。

runoob@runoob:~$ docker port bf08b7f2cd89

5000/tcp -> 0.0.0.0:5000

runoob@runoob:~$ docker port wizardly\_chandrasekhar

5000/tcp -> 0.0.0.0:5000

**（5）查看 WEB 应用程序日志**

docker logs [ID或者名字] 可以查看容器内部的标准输出。

runoob@runoob:~$ docker logs -f bf08b7f2cd89

\* Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)

192.168.239.1 - - [09/May/2016 16:30:37] "GET / HTTP/1.1" 200 -

192.168.239.1 - - [09/May/2016 16:30:37] "GET /favicon.ico HTTP/1.1" 404 -

-f: 让 docker logs 像使用 tail -f 一样来输出容器内部的标准输出。

从上面，我们可以看到应用程序使用的是 5000 端口并且能够查看到应用程序的访问日志。

**（6）查看WEB应用程序容器的进程**

我们还可以使用 docker top 来查看容器内部运行的进程

runoob@runoob:~$ docker top wizardly\_chandrasekhar

UID PID PPID ... TIME CMD

root 23245 23228 ... 00:00:00 python app.py

**（7）检查 WEB 应用程序**

使用 docker inspect 来查看 Docker 的底层信息。它会返回一个 JSON 文件记录着 Docker 容器的配置和状态信息。

runoob@runoob:~$ docker inspect wizardly\_chandrasekhar

[

{

"Id": "bf08b7f2cd897b5964943134aa6d373e355c286db9b9885b1f60b6e8f82b2b85",

"Created": "2018-09-17T01:41:26.174228707Z",

"Path": "python",

"Args": [

"app.py"

],

"State": {

"Status": "running",

"Running": true,

"Paused": false,

"Restarting": false,

"OOMKilled": false,

"Dead": false,

"Pid": 23245,

"ExitCode": 0,

"Error": "",

"StartedAt": "2018-09-17T01:41:26.494185806Z",

"FinishedAt": "0001-01-01T00:00:00Z"

},

......

**（8）停止 WEB 应用容器**

runoob@runoob:~$ docker stop wizardly\_chandrasekhar

wizardly\_chandrasekhar

**（9）重启WEB应用容器**

已经停止的容器，我们可以使用命令 docker start 来启动。

runoob@runoob:~$ docker start wizardly\_chandrasekhar

wizardly\_chandrasekhar

docker ps -l 查询最后一次创建的容器：

# docker ps -l

CONTAINER ID IMAGE PORTS NAMES

bf08b7f2cd89 training/webapp ... 0.0.0.0:5000->5000/tcp wizardly\_chandrasekhar

正在运行的容器，我们可以使用 docker restart 命令来重启

**（10）移除WEB应用容器**

我们可以使用 docker rm 命令来删除不需要的容器

runoob@runoob:~$ docker rm wizardly\_chandrasekhar

wizardly\_chandrasekhar

删除容器时，容器必须是停止状态，否则会报如下错误

runoob@runoob:~$ docker rm wizardly\_chandrasekhar

Error response from daemon: You cannot remove a running container bf08b7f2cd897b5964943134aa6d373e355c286db9b9885b1f60b6e8f82b2b85. Stop the container before attempting removal or force remove

**3 Docker 镜像使用**

当运行容器时，使用的镜像如果在本地中不存在，docker 就会自动从 docker 镜像仓库中下载，默认是从 Docker Hub 公共镜像源下载。

**（1）列出镜像列表**

我们可以使用 docker images 来列出本地主机上的镜像。

runoob@runoob:~$ docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

ubuntu 14.04 90d5884b1ee0 5 days ago 188 MB

php 5.6 f40e9e0f10c8 9 days ago 444.8 MB

nginx latest 6f8d099c3adc 12 days ago 182.7 MB

mysql 5.6 f2e8d6c772c0 3 weeks ago 324.6 MB

httpd latest 02ef73cf1bc0 3 weeks ago 194.4 MB

ubuntu 15.10 4e3b13c8a266 4 weeks ago 136.3 MB

hello-world latest 690ed74de00f 6 months ago 960 B

training/webapp latest 6fae60ef3446 11 months ago 348.8 MB

各个选项说明:

REPOSITORY：表示镜像的仓库源

TAG：镜像的标签

IMAGE ID：镜像ID

CREATED：镜像创建时间

SIZE：镜像大小

同一仓库源可以有多个 TAG，代表这个仓库源的不同个版本，如ubuntu仓库源里，有15.10、14.04等多个不同的版本，我们使用 REPOSITORY:TAG 来定义不同的镜像。

所以，我们如果要使用版本为15.10的ubuntu系统镜像来运行容器时，命令如下：

runoob@runoob:~$ docker run -t -i ubuntu:15.10 /bin/bash

root@d77ccb2e5cca:/#

如果要使用版本为14.04的ubuntu系统镜像来运行容器时，命令如下：

runoob@runoob:~$ docker run -t -i ubuntu:14.04 /bin/bash

root@39e968165990:/#

如果你不指定一个镜像的版本标签，例如你只使用 ubuntu，docker 将默认使用 ubuntu:latest 镜像。

**（2）获取一个新的镜像**

当我们在本地主机上使用一个不存在的镜像时 Docker 就会自动下载这个镜像。如果我们想预先下载这个镜像，我们可以使用 docker pull 命令来下载它。

Crunoob@runoob:~$ docker pull ubuntu:13.10

13.10: Pulling from library/ubuntu

6599cadaf950: Pull complete

23eda618d451: Pull complete

f0be3084efe9: Pull complete

52de432f084b: Pull complete

a3ed95caeb02: Pull complete

Digest: sha256:15b79a6654811c8d992ebacdfbd5152fcf3d165e374e264076aa435214a947a3

Status: Downloaded newer image for ubuntu:13.10

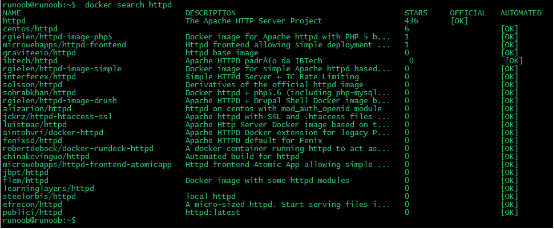
下载完成后，我们可以直接使用这个镜像来运行容器。

**（3）查找镜像**

我们可以从 Docker Hub 网站来搜索镜像，Docker Hub 网址为： https://hub.docker.com/

我们也可以使用 docker search 命令来搜索镜像。比如我们需要一个httpd的镜像来作为我们的web服务。我们可以通过 docker search 命令搜索 httpd 来寻找适合我们的镜像。

runoob@runoob:~$ docker search httpd



NAME:镜像仓库源的名称

DESCRIPTION:镜像的描述

OFFICIAL:是否docker官方发布

**（4）拖取镜像**

我们决定使用上图中的httpd 官方版本的镜像，使用命令 docker pull 来下载镜像。

runoob@runoob:~$ docker pull httpd

Using default tag: latest

latest: Pulling from library/httpd

8b87079b7a06: Pulling fs layer

a3ed95caeb02: Download complete

0d62ec9c6a76: Download complete

a329d50397b9: Download complete

ea7c1f032b5c: Waiting

be44112b72c7: Waiting

下载完成后，我们就可以使用这个镜像了。

runoob@runoob:~$ docker run httpd

**（5）创建镜像**

当我们从docker镜像仓库中下载的镜像不能满足我们的需求时，我们可以通过以下两种方式对镜像进行更改。

1）从已经创建的容器中更新镜像，并且提交这个镜像

2）使用 Dockerfile 指令来创建一个新的镜像

**（6）更新镜像**

1）更新镜像之前，我们需要使用镜像来创建一个容器。

runoob@runoob:~$ docker run -t -i ubuntu:15.10 /bin/bash

root@e218edb10161:/#

在运行的容器内使用 apt-get update 命令进行更新。

在完成操作之后，输入 exit命令来退出这个容器。

2）此时ID为e218edb10161的容器，是按我们的需求更改的容器。我们可以通过命令 docker commit来提交容器副本。

runoob@runoob:~$ docker commit -m="has update" -a="runoob" e218edb10161 runoob/ubuntu:v2

sha256:70bf1840fd7c0d2d8ef0a42a817eb29f854c1af8f7c59fc03ac7bdee9545aff8

3）各个参数说明：

-m:提交的描述信息

-a:指定镜像作者

e218edb10161：容器ID

runoob/ubuntu:v2:指定要创建的目标镜像名

4）我们可以使用 docker images 命令来查看我们的新镜像 runoob/ubuntu:v2：

runoob@runoob:~$ docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

runoob/ubuntu v2 70bf1840fd7c 15 seconds ago 158.5 MB

ubuntu 14.04 90d5884b1ee0 5 days ago 188 MB

php 5.6 f40e9e0f10c8 9 days ago 444.8 MB

nginx latest 6f8d099c3adc 12 days ago 182.7 MB

mysql 5.6 f2e8d6c772c0 3 weeks ago 324.6 MB

httpd latest 02ef73cf1bc0 3 weeks ago 194.4 MB

ubuntu 15.10 4e3b13c8a266 4 weeks ago 136.3 MB

hello-world latest 690ed74de00f 6 months ago 960 B

training/webapp latest 6fae60ef3446 12 months ago 348.8 MB

使用我们的新镜像 runoob/ubuntu 来启动一个容器

runoob@runoob:~$ docker run -t -i runoob/ubuntu:v2 /bin/bash

root@1a9fbdeb5da3:/#

**（7）构建镜像**

1）我们使用命令 docker build ， 从零开始来创建一个新的镜像。为此，我们需要创建一个 Dockerfile 文件，其中包含一组指令来告诉 Docker 如何构建我们的镜像。

runoob@runoob:~$ cat Dockerfile

FROM centos:6.7

MAINTAINER Fisher "fisher@sudops.com"

RUN /bin/echo 'root:123456' |chpasswd

RUN useradd runoob

RUN /bin/echo 'runoob:123456' |chpasswd

RUN /bin/echo -e "LANG=\"en\_US.UTF-8\"" >/etc/default/local

EXPOSE 22

EXPOSE 80

CMD /usr/sbin/sshd -D

2）每一个指令都会在镜像上创建一个新的层，每一个指令的前缀都必须是大写的。

第一条FROM，指定使用哪个镜像源

RUN 指令告诉docker 在镜像内执行命令，安装了什么。。。

然后，我们使用 Dockerfile 文件，通过 docker build 命令来构建一个镜像。

runoob@runoob:~$ docker build -t runoob/centos:6.7 .

Sending build context to Docker daemon 17.92 kB

Step 1 : FROM centos:6.7

---&gt; d95b5ca17cc3

Step 2 : MAINTAINER Fisher "fisher@sudops.com"

---&gt; Using cache

---&gt; 0c92299c6f03

Step 3 : RUN /bin/echo 'root:123456' |chpasswd

---&gt; Using cache

---&gt; 0397ce2fbd0a

Step 4 : RUN useradd runoob

......

3）参数说明：

-t ：指定要创建的目标镜像名

. ：Dockerfile 文件所在目录，可以指定Dockerfile 的绝对路径

4）使用docker images 查看创建的镜像已经在列表中存在,镜像ID为860c279d2fec

runoob@runoob:~$ docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

runoob/centos 6.7 860c279d2fec About a minute ago 190.6 MB

runoob/ubuntu v2 70bf1840fd7c 17 hours ago 158.5 MB

ubuntu 14.04 90d5884b1ee0 6 days ago 188 MB

php 5.6 f40e9e0f10c8 10 days ago 444.8 MB

nginx latest 6f8d099c3adc 12 days ago 182.7 MB

mysql 5.6 f2e8d6c772c0 3 weeks ago 324.6 MB

httpd latest 02ef73cf1bc0 3 weeks ago 194.4 MB

ubuntu 15.10 4e3b13c8a266 5 weeks ago 136.3 MB

hello-world latest 690ed74de00f 6 months ago 960 B

centos 6.7 d95b5ca17cc3 6 months ago 190.6 MB

training/webapp latest 6fae60ef3446 12 months ago 348.8 MB

5）我们可以使用新的镜像来创建容器

runoob@runoob:~$ docker run -t -i runoob/centos:6.7 /bin/bash

[root@41c28d18b5fb /]# id runoob

uid=500(runoob) gid=500(runoob) groups=500(runoob)

从上面看到新镜像已经包含我们创建的用户runoob

**（8）设置镜像标签**

我们可以使用 docker tag 命令，为镜像添加一个新的标签。

runoob@runoob:~$ docker tag 860c279d2fec runoob/centos:dev

docker tag 镜像ID，这里是 860c279d2fec ,用户名称、镜像源名(repository name)和新的标签名(tag)。

使用 docker images 命令可以看到，ID为860c279d2fec的镜像多一个标签。

runoob@runoob:~$ docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

runoob/centos 6.7 860c279d2fec 5 hours ago 190.6 MB

runoob/centos dev 860c279d2fec 5 hours ago 190.6 MB

runoob/ubuntu v2 70bf1840fd7c 22 hours ago 158.5 MB

ubuntu 14.04 90d5884b1ee0 6 days ago 188 MB

php 5.6 f40e9e0f10c8 10 days ago 444.8 MB

nginx latest 6f8d099c3adc 13 days ago 182.7 MB

mysql 5.6 f2e8d6c772c0 3 weeks ago 324.6 MB

httpd latest 02ef73cf1bc0 3 weeks ago 194.4 MB

ubuntu 15.10 4e3b13c8a266 5 weeks ago 136.3 MB

hello-world latest 690ed74de00f 6 months ago 960 B

centos 6.7 d95b5ca17cc3 6 months ago 190.6 MB

training/webapp latest 6fae60ef3446 12 months ago 348.8 MB

**4 Docker 容器连接**

前面我们实现了通过网络端口来访问运行在docker容器内的服务。下面我们来实现通过端口连接到一个docker容器

**（1）网络端口映射**

1）我们创建了一个 python 应用的容器。

runoob@runoob:~$ docker run -d -P training/webapp python app.py

fce072cc88cee71b1cdceb57c2821d054a4a59f67da6b416fceb5593f059fc6d

2）另外，我们可以指定容器绑定的网络地址，比如绑定 127.0.0.1。

我们使用 -P 参数创建一个容器，使用 docker ps 来看到端口5000绑定主机端口32768。

runoob@runoob:~$ docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

fce072cc88ce training/webapp "python app.py" 4 minutes ago Up 4 minutes 0.0.0.0:32768->5000/tcp grave\_hopper

3）我们也可以使用 -p 标识来指定容器端口绑定到主机端口。

两种方式的区别是:

-P :是容器内部端口随机映射到主机的高端口。

-p : 是容器内部端口绑定到指定的主机端口。

runoob@runoob:~$ docker run -d -p 5000:5000 training/webapp python app.py

33e4523d30aaf0258915c368e66e03b49535de0ef20317d3f639d40222ba6bc0

runoob@runoob:~$ docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

33e4523d30aa training/webapp "python app.py" About a minute ago Up About a minute 0.0.0.0:5000->5000/tcp berserk\_bartik

fce072cc88ce training/webapp "python app.py" 8 minutes ago Up 8 minutes 0.0.0.0:32768->5000/tcp grave\_hopper

4）另外，我们可以指定容器绑定的网络地址，比如绑定127.0.0.1。

runoob@runoob:~$ docker run -d -p 127.0.0.1:5001:5000 training/webapp python app.py

95c6ceef88ca3e71eaf303c2833fd6701d8d1b2572b5613b5a932dfdfe8a857c

runoob@runoob:~$ docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

95c6ceef88ca training/webapp "python app.py" 6 seconds ago Up 6 seconds 5000/tcp, 127.0.0.1:5001->5000/tcp adoring\_stonebraker

33e4523d30aa training/webapp "python app.py" 3 minutes ago Up 3 minutes 0.0.0.0:5000->5000/tcp berserk\_bartik

fce072cc88ce training/webapp "python app.py" 10 minutes ago Up 10 minutes 0.0.0.0:32768->5000/tcp grave\_hopper

5）这样我们就可以通过访问127.0.0.1:5001来访问容器的5000端口。

上面的例子中，默认都是绑定 tcp 端口，如果要绑定 UDP 端口，可以在端口后面加上 /udp。

runoob@runoob:~$ docker run -d -p 127.0.0.1:5000:5000/udp training/webapp python app.py

6779686f06f6204579c1d655dd8b2b31e8e809b245a97b2d3a8e35abe9dcd22a

runoob@runoob:~$ docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

6779686f06f6 training/webapp "python app.py" 4 seconds ago Up 2 seconds 5000/tcp, 127.0.0.1:5000->5000/udp drunk\_visvesvaraya

95c6ceef88ca training/webapp "python app.py" 2 minutes ago Up 2 minutes 5000/tcp, 127.0.0.1:5001->5000/tcp adoring\_stonebraker

33e4523d30aa training/webapp "python app.py" 5 minutes ago Up 5 minutes 0.0.0.0:5000->5000/tcp berserk\_bartik

fce072cc88ce training/webapp "python app.py" 12 minutes ago Up 12 minutes 0.0.0.0:32768->5000/tcp grave\_hopper

docker port 命令可以让我们快捷地查看端口的绑定情况。

runoob@runoob:~$ docker port adoring\_stonebraker 5000

127.0.0.1:5001

**（2）Docker容器连接**

端口映射并不是唯一把 docker 连接到另一个容器的方法。

docker有一个连接系统允许将多个容器连接在一起，共享连接信息。

docker连接会创建一个父子关系，其中父容器可以看到子容器的信息。

**（3）容器命名**

当我们创建一个容器的时候，docker会自动对它进行命名。另外，我们也可以使用--name标识来命名容器，例如：

runoob@runoob:~$ docker run -d -P --name runoob training/webapp python app.py

43780a6eabaaf14e590b6e849235c75f3012995403f97749775e38436db9a441

我们可以使用 docker ps 命令来查看容器名称。

runoob@runoob:~$ docker ps -l

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

43780a6eabaa training/webapp "python app.py" 3 minutes ago Up 3 minutes 0.0.0.0:32769->5000/tcp runoob

**5 其他安装**

**Nginx**

**Mysql**

**Tomcat**

**Python**

**Redis**

**MongoDB**

## 四、Docker 命令大全

**1 容器生命周期管理**

run

start/stop/restart

kill

rm

pause/unpause

create

exec

**2 容器操作**

ps

inspect

top

attach

events

logs

wait

export

port

**3 容器rootfs命令**

commit

cp

diff

**4 镜像仓库**

login

pull

push

search

**5 本地镜像管理**

images

rmi

tag

build

history

save

import

**6 info|version**

info

version