

参考: <http://dockone.io/article/101>

## Docker 入门教程（一）介绍

Docker 是一个新的容器化的技术，它轻巧，且易移植，号称“build once, configure once and run anywhere（译者注：这个就不翻译了，翻译出来味道就没了）”。本文是 Flux7 的 Docker 系列教程的第一部分。请和这份教程一起学习和理解 Docker 有什么优势以及如何更好地使用它。

让我们一起来学习 Docker。

本文主要涉及 Docker 的基础知识：Docker 的特征、理念以及如何安装使用 Docker。

## Docker 特征

Docker 有不少有趣的功能，通过本教程相信你会更好地理解它们。Docker 的特性主要包括以下几点：

- 速度飞快以及优雅的隔离框架
- 物美价廉
- CPU/内存的低消耗
- 快速开/关机
- 跨云计算基础构架

## Docker 组件与元素

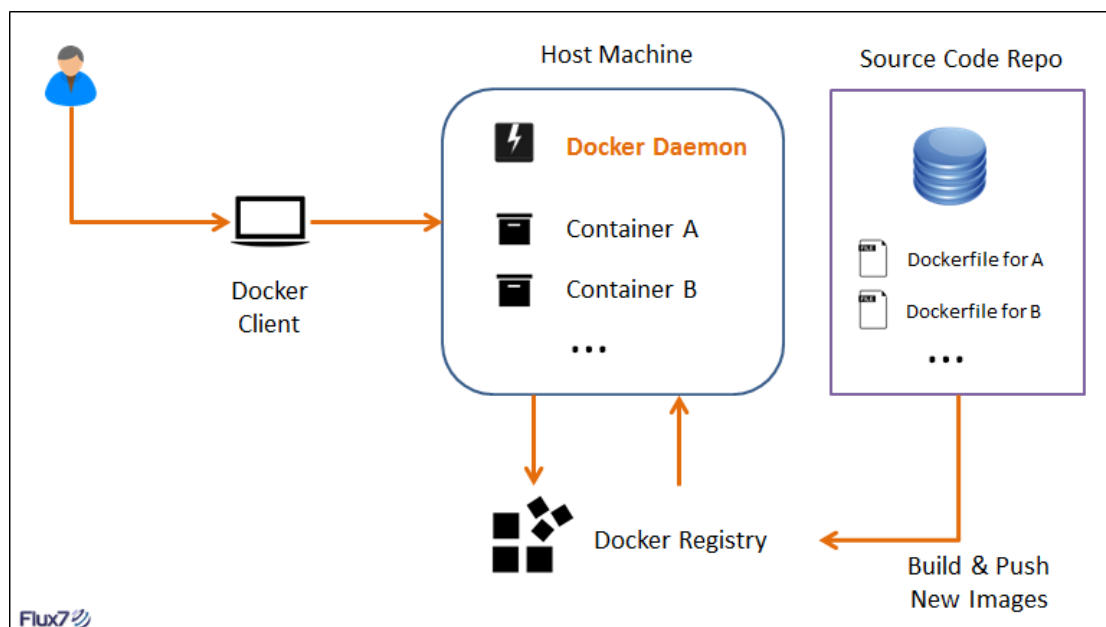
Docker 有三个组件和三个基本元素，读者可以快速浏览下面[这个视频](#)来了解这些组建和元素，以及它们的关系。三个组件分别是：

- **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。

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。



在讨论 Docker 组件和基本要素如何交互之前，让我们来谈谈 Docker 的支柱。Docker 使用以下操作系统的功能来提高容器技术效率：

- **Namespaces** 充当隔离的第一级。确保一个容器中运行一个进程而且不能看到或影响容器外的其它进程。
- **Control Groups** 是 LXC 的重要组成部分，具有资源核算与限制的关键功能。

- **UnionFS**（文件系统）作为容器的构建块。为了支持 Docker 的轻量级以及速度快的特性，它创建了用户层。

## 如何把它们放在一起

运行任何应用程序，都需要有两个基本步骤：

1. 构建一个镜像。
2. 运行容器。

这些步骤都是从 **Docker Client** 的命令开始的。**Docker Client** 使用的是 Docker 二进制文件。在基础层面上，**Docker Client** 会告诉 **Docker Daemon** 需要创建的镜像以及需要在容器内运行的命令。当 Daemon 接收到创建镜像的信号后，会进行如下操作：

### 第 1 步：构建镜像

如前所述，**Docker Image** 是一个构建容器的只读模板，它包含了容器启动所需的所有信息，包括运行程序和配置数据。

每个镜像都源于一个基本的镜像，然后根据 *Dockerfile* 中的指令创建模板。对于每个指令，在镜像上创建一个新的层面。

一旦镜像创建完成，就可以将它们推送到中央 registry: **Docker Index**，以供他人使用。然而，**Docker Index** 为镜像提供了两个级别的访问权限：公有访问和私有访问。你可以将镜像存储在私有仓库，Docker 官网有私有仓库的套餐可以供你选择。总之，公有仓库是可搜索和可重复使用的，而私有仓库只能给那些拥有访问权限的成员使用。**Docker Client** 可用于 **Docker Index** 内的镜像搜索。

### 第 2 步：运行容器

运行容器源于我们在第一步中创建的镜像。当容器被启动后，一个读写层会被添加到镜像的顶层。当分配到合适的网络和 IP 地址后，需要的应用程序就可以在容器中运行了。

如果你还是不太理解，先别急，在接下来的内容中我们将会和你分享很多的实战案例。

目前为止，我们已经介绍了 Docker 的基本概念，接下来，让我们一起安装 Docker！

## 安装 Docker：快速指南

下面让我们来看看如何在 Ubuntu 12.04 LTS 上安装 Docker（译者注：在 CentOS 6.5 安装可以参考[这里](#)）：

1. 检查 APT 系统的 HTTPS 兼容性。如果 `usr/lib/apt/methods/https` 文件不存在，请安装 `apt-transport-https` 程序包。
2. 在本地添加 Docker Repository 密钥。Repository key: `hkp://keyserver.ubuntu.com:80`  
`--recv-keys 36A1D7869245C8950F966E92D8576A8BA88D21E9`
3. 添加 Docker Repository 到 APT 源列表。
4. 安装 lxc-Docker 程序包。 `sudo apt-get update sudo apt-get install lxc-docker`
5. 验证所安装的内容。 `sudo docker run -i -t ubuntu /bin/bash`

原文链接：[Part 1: Introduction](#)（翻译：田浩浩 审校：李颖杰）

## Docker 入门教程（二）命令

本文是系列入门教程的第二篇，介绍了 Docker 的基本命令以及命令的用法和功能。

在 Docker 系列教程的第一篇文章中，我们了解了 Docker 的基础知识，知道了它是如何工作以及如何安装的。在这篇文章中，我们将学习 15 个 Docker 命令，并通过实践来学习它是如何工作的。

首先，让我们通过下面的命令来检查 Docker 的安装是否正确：

```
docker info
```

结果如下：

```
[root@master ~]# docker info
Containers: 0
Images: 0
Storage Driver: devicemapper
  Pool Name: docker-8:2-413022-pool
  Pool Blocksize: 65.54 kB
  Backing Filesystem: extfs
  Data file: /dev/loop0
  Metadata file: /dev/loop1
  Data Space Used: 305.7 MB
  Data Space Total: 107.4 GB
  Data Space Available: 4.817 GB
  Metadata Space Used: 729.1 kB
  Metadata Space Total: 2.147 GB
  Metadata Space Available: 2.147 GB
  Udev Sync Supported: true
  Deferred Removal Enabled: false
  Data loop file: /var/lib/docker/devicemapper/devicemapper/data
  Metadata loop file: /var/lib/docker/devicemapper/devicemapper/metadata
  Library Version: 1.02.117-RHEL6 (2016-08-15)
Execution Driver: native-0.2
Logging Driver: json-file
Kernel Version: 3.10.5-3.el6.x86_64
Operating System: <unknown>
CPUs: 1
Total Memory: 476.4 MiB
Name: master
ID: N6GG:SPIM:K5VH:BRCS:DXAB:P2UP:ZDET:CGBX:GWRB:BYJZ:7XOE:PZPP
WARNING: No swap limit support
[root@master ~]#
```

如果没有找到这条命令，则表示 Docker 安装错误。如果安装正确，则会输出类似上面的内容。

到这一步 Docker 里还没有镜像或是容器。所以，让我们通过使用命令预先构建的镜像来创建一个：

```
sudo docker pull busybox
```

输出如下：

```
[root@master ~]# docker pull busybox
latest: Pulling from busybox
a61cd723bcf2: Pull complete
9967c5ad88de: Pull complete
Digest: sha256:98a0bd48d22ff96ca23bfda2fe1cf72034ea803bd79e64a5a5f274aca0f9c51c
Status: Downloaded newer image for busybox:latest
[root@master ~]#
```

**BusyBox** 是一个最小的 Linux 系统，它提供了该系统的主要功能，不包含一些与 GNU 相关的功能和选项。

下一步我们将运行一个“Hello World”的例子，我们暂且叫它“Hello Docker”吧。

```
docker run busybox /bin/echo Hello Docker
```

现在，让我们以后台进程的方式运行 **heker**：

```
sample_job=$(docker run -d busybox /bin/sh -c "while true; do echo Docker; sleep 1; done")
```

**sample\_job** 命令会隔一秒打印一次 Docker，使用 **docker logs** 可以查看输出的结果。如果没有给这个 job 起名字，那这个 job 就会被分配一个 id，以后使用命令例如 **docker logs** 查看日志就会变得比较麻烦。

运行 **docker logs** 命令来查看 **job** 的当前状态：

```
docker logs $sample_job
```

所有 Docker 命令可以用以下命令查看：

```
docker help
```

名为 `sample_job` 的容器，可以使用以下命令来停止：

```
docker stop $sample_job
```

使用以下命令可以重新启动该容器：

```
docker restart $sample_job
```

如果要完全移除容器，需要先将该容器停止，然后才能移除。像这样：

```
docker stop $sample_job
```

```
docker rm $sample_job
```

将容器的状态保存为镜像，使用以下命令：

```
docker commit $sample_job job1
```

注意，镜像名称只能取字符[a-z]和数字[0-9]。

现在，你就可以使用以下命令查看所有镜像的列表：

```
docker images
```

输出结果如下：

```
[root@master ~]# docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	VIRTUAL SIZE
busybox	latest	9967c5ad88de	6 weeks ago	1.093 MB

```
[root@master ~]#
```

在我们之前的 [Docker 教程](#)中，我们学习过镜像是存储在 Docker registry。在 registry 中的镜像可以使用以下命令查找到：

```
docker search (image-name)
```

这条命令好像是到网上去寻找镜像

查看镜像的历史版本可以执行以下命令：

```
docker history (image_name)
```

输出如下：

```
[root@master ~]# docker history 9967c5ad88de
```

IMAGE	CREATED	CREATED BY	SIZE	COMMENT
9967c5ad88de	6 weeks ago	/bin/sh -c #(nop) CMD ["sh"]		0 B
a61cd723bcf2	6 weeks ago	/bin/sh -c #(nop) ADD file:ced3aa7577c8f97040	1.093 MB	

[root@master ~]#

最后，使用以下命令将镜像推送到 registry:

```
docker push (image_name)
```

非常重要的一点是，你必须要知道存储库不是根存储库，它应该使用此格式

```
(user)/(repo_name)。
```

这都是一些非常基本的 Docker 命令。在我们 Docker 教程系列的第六章，我们将讨论如何使用 Docker 运行 Python 的 Web 应用程序，以及一些进阶的 Docker 命令。

原文链接: [Part 2: The 15 Commands](#) (翻译: 田浩浩 审校: 李颖杰)



## Docker 入门教程（三）Dockerfile

【编者的话】DockerOne 组织翻译了 Flux7 的 Docker 入门教程，本文是系列入门教程的第三篇，介绍了 Dockerfile 的语法，DockerOne 目前在代码高亮部分还有些 Bug，我们会尽快修复，目前在代码部分有会些字符会被转义。

在 Docker 系列教程的上一篇文章中，我们介绍了 15 个 Docker 命令，你应该对 Docker 有个大致的了解了。那 15 个命令在手动创建镜像时会用到，它们涵盖了镜像的创建、提交、搜索、pull 和 push 的功能。

现在问题来了，既然 Docker 能自动创建镜像，那为什么要选择耗时而又乏味的方式来创建镜像呢？

Docker 为我们提供了 Dockerfile 来解决自动化的问题。在这篇文章中，我们将讨论什么是 Dockerfile，它能够做到的事情以及 Dockerfile 的一些基本语法。

### 易于自动化的命令

Dockerfile 包含创建镜像所需要的全部指令。基于在 Dockerfile 中的指令，我们可以使用 `Docker build` 命令来创建镜像。通过减少镜像和容器的创建过程来简化部署。

Dockerfile 支持支持的语法命令如下：

#### INSTRUCTION argument

指令不区分大小写。但是，命名约定为全部大写。

所有 Dockerfile 都必须以 `FROM` 命令开始。`FROM` 命令会指定镜像基于哪个基础镜像创建，

接下来的命令也会基于这个基础镜像（译者注：*CentOS* 和 *Ubuntu* 有些命令可是不一样的）。`FROM` 命令可以多次使用，表示会创建多个镜像。具体语法如下：

#### FROM <image name>

例如：

## FROM ubuntu

上面的指定告诉我们，新的镜像将基于 Ubuntu 的镜像来构建。

继 **FROM** 命令，Dockerfile 还提供了一些其它的命令以实现自动化。在文本文件或 Dockerfile 文件中这些命令的顺序就是它们被执行的顺序。

让我们了解一下这些有趣的 Dockerfile 命令吧。

**1. MAINTAINER:** 设置该镜像的作者。语法如下：

## MAINTAINER <author name>

**2. RUN:** 在 shell 或者 exec 的环境下执行的命令。**RUN** 指令会在新创建的镜像上添加新的层面，接下来提交的结果用在 Dockerfile 的下一条指令中。语法如下：

## RUN command

**3. ADD:** 复制文件指令。它有两个参数<source>和<destination>。destination 是容器内的路径。source 可以是 URL 或者是启动配置上下文中的一个文件。语法如下：

## ADD src destination

**4. CMD:** 提供了容器默认的执行命令。Dockerfile 只允许使用一次 CMD 指令。使用多个 CMD 会抵消之前所有的指令，只有最后一个指令生效。CMD 有三种形式：

```
CMD ["executable","param1","param2"]
```

```
CMD ["param1","param2"]
```

```
CMD command param1 param2
```

**5. EXPOSE:** 指定容器在运行时监听的端口。语法如下：

```
EXPOSE <port>;
```

6. ENTRYPOINT: 配置给容器一个可执行的命令, 这意味着在每次使用镜像创建容器时一个特定的应用程序可以被设置为默认程序。同时也意味着该镜像每次被调用时仅能运行指定的应用。类似于 CMD, Docker 只允许一个 ENTRYPOINT, 多个 ENTRYPOINT 会抵消之前所有的指令, 只执行最后的 ENTRYPOINT 指令。语法如下:

```
ENTRYPOINT ["executable", "param1", "param2"]
```

```
ENTRYPOINT command param1 param2
```

7. WORKDIR: 指定 RUN、CMD 与 ENTRYPOINT 命令的工作目录。语法如下:

```
WORKDIR /path/to/workdir
```

8. ENV: 设置环境变量。它们使用键值对, 增加运行程序的灵活性。语法如下:

```
ENV <key> <value>
```

9. USER: 镜像正在运行时设置一个 UID。语法如下:

```
USER <uid>
```

10. VOLUME: 授权访问从容器内到主机上的目录。语法如下:

```
VOLUME ["/data"]
```

## Dockerfile 最佳实践

与使用的其他任何应用程序一样，总会有可以遵循的最佳实践。你可以阅读更多有关 [Dockerfile](#) 的最佳实践。

以下是我们列出的基本的 Dockerfile 最佳实践：

- 保持常见的指令像 `MAINTAINER` 以及从上至下更新 Dockerfile 命令；
- 当构建镜像时使用可理解的标签，以便更好地管理镜像；
- 避免在 Dockerfile 中映射公有端口；
- `CMD` 与 `ENTRYPOINT` 命令请使用数组语法。

在接下来的文章中，我们将讨论 [Docker Registry](#) 及其工作流程。

原文链接：[Part 3: Automation is the word using Dockerfile](#)（翻译：田浩浩 审校：李颖杰）

## Docker 入门教程（四）Docker Registry

【编者的话】DockerOne 组织翻译了 Flux7 的 Docker 入门教程，本文是系列入门教程的第四篇，介绍了 Docker Registry，它是 Docker 中的重要组件。本文通过情景演绎的方式对其进行了介绍，图文并茂，强烈推荐读者阅读。

在 [Docker 系列教程的上一篇文章](#) 中，我们讨论了 Dockerfile 的重要性并提供了一系列 Dockerfile 的命令，使镜像的自动构建更加容易。在这篇文章中，我们将介绍 Docker 的一个重要组件：Docker Registry。它是所有仓库（包括共有和私有）以及工作流的中央 Registry。在深入 Docker Registry 之前，让我们先去看看一些常见的术语和与仓库相关的概念。

1. *Repositories*（仓库）可以被标记为喜欢或者像书签一样标记起来
2. 用户可以在仓库下评论。
3. 私有仓库和共有仓库类似，不同之处在于前者不会在搜索结果中显示，也没有访问它的权限。只有用户设置为合作者才能访问私有仓库。
4. 成功推送之后配置 [webhooks](#)。

Docker Registry 有三个角色，分别是 *index*、*registry* 和 *registry client*。

### 角色 1 -- Index

**index** 负责并维护有关用户帐户、镜像的校验以及公共命名空间的信息。它使用以下组件维护这些信息：

- *Web UI*
- 元数据存储
- 认证服务
- 符号化

这也分解了较长的 *URL*，以方便使用和验证用户存储库。

## 角色 2 --Registry

*registry* 是镜像和图表的仓库。然而，它没有一个本地数据库，也不提供用户的身份认证，由 *S3*、云文件和本地文件系统提供数据库支持。此外，通过 *Index Auth service* 的 *Token* 方式进行身份认证。*Registries* 可以有不同的类型。现在让我们来分析其中的几种类型：

1. *Sponsor Registry*: 第三方的 *registry*，供客户和 *Docker* 社区使用。
2. *Mirror Registry*: 第三方的 *registry*，只让客户使用。
3. *Vendor Registry*: 由发布 *Docker* 镜像的供应商提供的 *registry*。
4. *Private Registry*: 通过设有防火墙和额外的安全层的私有实体提供的 *registry*。

## 角色 3 --Registry Client

*Docker* 充当 *registry* 客户端来负责维护推送和拉取的任务，以及客户端的授权。

## Docker Registry 工作流程详解

现在，让我们讨论五种情景模式，以便更好地理解 *Docker Registry*。

**情景 A：** 用户要获取并下载镜像。所涉及的步骤如下：

1. 用户发送请求到 *index* 来下载镜像。
2. *index* 发出响应，返回三个相关信息：
  - 该镜像所处的 *registry*
  - 该镜像包括所有层的校验
  - 以授权为目的的 *Token* > 注意：当请求 *header* 里有 *X-Docker-Token* 时才会返回 *Token*。而私人仓库需要基本的身份验证，对于公有仓库这一点不是强制性的。
3. 用户通过响应后返回的 *Token* 和 *registry* 沟通，*registry* 全权负责镜像，它用来存储基本的镜像和继承的层。
4. *registry* 现在要与 *index* 证实该 *token* 是被授权的。
5. *index* 会发送“true” 或者 “false”给 *registry*，由此判定是否允许用户下载所需要的镜像。



**情景 B:** 用户想要将镜像推送到 *registry* 中。其中涉及的步骤如下:

1. 用户发送附带证书的请求到 *index* 要求分配库名。
2. 在认证成功,命名空间可用之后,库名也被分配。*index* 发出响应返回临时的 *token*。
3. 镜像连带 *token*, 一起被推送到 *registry* 中。
4. *registry* 与 *index* 证实 *token* 被授权, 然后在 *index* 验证之后开始读取推送流。
5. 该 *index* 由 *Docker* 校验的镜像更新。



**情景 C:** 用户想要从 *index* 或 *registry* 中删除镜像:

1. *index* 接收来自 *Docker* 一个删除库的信号。
2. 如果 *index* 对库验证成功, 它将删除该库, 并返回一个临时的 *token*。
3. *registry* 现在接收到带有该 *token* 的删除信号。
4. *registry* 与 *index* 核实该 *token*, 然后删除库以及所有与其相关的信息。



5. *Docker* 现在通知有关删除的 *index*，然后 *index* 移除库的所有记录。



**情景 D：** 用户希望在没有 *index* 的独立模式中使用 *registry*。

使用没有 *index* 的 *registry*，这完全由 *Docker* 控制，它最适合于在私有网络中存储镜像。

*registry* 运行在一个特殊的模式里，此模式限制了 *registry* 与 *Docker index* 的通信。所有有关安全性和身份验证的信息需要用户自己注意。

**情景 E：** 用户想要在有 *index* 的独立模式中使用 *registry*。

在这种情况下，一个自定义的 *index* 会被创建在私有网络里来存储和访问镜像的问题。然而，通知 *Docker* 有关定制的 *index* 是耗时的。*Docker* 提供一个有趣的概念 *chaining registries*，从而，实现负载均衡和为具体请求而指定的 *registry* 分配。在接下来的 *Docker* 教程系列中，我们将讨论如何在上述每个情景中使用 *Docker Registry API*，以及深入了解 *Docker Security*。

原文链接：[Part 4: Registry & Workflows](#)（翻译：田浩浩 审校：李颖杰）

## *Docker* 入门教程（五）*Docker* 安全

【编者的话】*DockOne* 组织翻译了 *Flux7* 的 *Docker* 入门教程，本文是系列入门教程的第五篇，介绍了 *Docker* 的安全问题，依然是老话重谈，入门者可以通过阅读本文快速了

解。

我们必须高度重视开源软件的安全问题，当开发者在使用 *Docker* 时，从本地构建应用程序到生产环境部署是没有任何差异的（译者注：作者的言外之意是更应该重视 *Docker* 的安全问题）。当 *Docker* 被越来越多的平台使用的时候，我们需要严格保证 *Docker* 作为一个项目或者平台的安全性。

因此，我们决定在 *Docker* 系列教程的第五篇来讨论 *Docker* 安全性的相关问题以及为什么它们会影响到 *Docker* 的整体安全性。由于 *Docker* 是 *LXC* 的延伸，它也很容易使用 *LXC* 的安全特性。

在本系列的第一篇文章中，我们知道 `docker run` 命令可以用来运行容器。那运行这个命令后，*Docker* 做了哪些具体的工作呢？具体如下：

1. `docker run` 命令初始化。
2. *Docker* 运行 `lxc-start` 来执行 `run` 命令。
3. `lxc-start` 在容器中创建了一组 *namespace* 和 *Control Groups*。

对于那些不知道 *namespace* 和 *control groups* 的概念的读者，我在这里先给他们解释一下：*namespace* 是隔离的第一级，容器是相互隔离的，一个容器是看不到其它容器内部运行的进程情况（译者注：*namespace* 系列教程可以阅读 *DockerOne* 上的[系列教程](#)）。每个容器都分配了单独的网络栈，因此一个容器不可能访问另一容器的 *sockets*。为了支持容器之间的 *IP* 通信，您必须指定容器的公网 *IP* 端口。

*Control Groups* 是非常重要的组件，具有以下功能：

- 负责资源核算和限制。
- 提供 *CPU*、内存、*I/O* 和网络相关的指标。
- 避免某种 *DoS* 攻击。
- 支持多租户平台。

## *Docker Daemon* 的攻击面

*Docker Daemon* 以 *root* 权限运行，这意味着有一些问题需要格外小心。

下面介绍一些需要注意的地方：

- 当 *Docker* 允许与访客容器目录共享而不限其访问权限时，*Docker Daemon* 的控制权应该只给授权用户。
- *REST API* 支持 *Unix sockets*，从而防止了 *cross-site-scripting* 攻击。
- *REST API* 的 *HTTP* 接口应该在可信网络或者 *VPN* 下使用。
- 在服务器上单独运行 *Docker* 时，需要与其它服务隔离。

一些关键的 *Docker* 安全特性包括：

1. 容器以非特权用户运行。

2. *Apparmor*、*SELinux*、*GRSEC* 解决方案，可用于额外的安全层。
3. 可以使用其它容器系统的安全功能。

## Docker.io API

用于管理与授权和安全相关的几个进程，*Docker* 提供 *REST API*。以下表格列出了关于此 *API* 用于维护相关安全功能的一些命令。



*Docker* 系列教程的下一篇文章中我们将继续探讨前面第二篇文章中所讨论的 *Docker* 命令的进阶。

原文链接: [Part 5: Docker Security](#) (翻译: 田浩浩 审校: 李颖杰)

## Docker 入门教程（六）另外的 15 个 Docker 命令

【编者的话】*DockerOne* 组织翻译了 *Flux7* 的 *Docker* 入门教程，本文是系列入门教程的第六篇，继续介绍 *Docker* 命令。之前的第二篇文章中我们就介绍了一些基本的 *Docker* 命令，本文过后，你将会接触到所有的 *Docker* 常用命令。努力学习吧。

在之前的文章中，我们介绍了 15 个 *Docker* 命令，并分享了它们的实践经验。在这篇文

章中，我们将学习另外的 15 个 *Docker* 命令。它们分别是：

### *daemon:*

*Docker daemon* 是一个用于管理容器的后台进程。一般情况下，守护进程是一个长期运行的用来处理请求的进程服务。`-d` 参数用于运行后台进程。

### *build:*

如之前所讨论的，可以使用 *Dockerfile* 来构建镜像。简单的构建命令如下：

```
docker build [options] PATH | URL
```

还有一些 *Docker* 提供的额外选项，如：

`--rm=true` 表示构建成功后，移除所有中间容器

`--no-cache=false` 表示在构建过程中不使用缓存

下面是一张使用 `Docker build` 命令的截图。



### *attach:*

*Docker* 允许使用 `attach` 命令与运行中的容器交互，并且可以随时观察容器内进程的运行状况。退出容器可以通过两种方式来完成：

- `Ctrl+C` 直接退出
- `Ctrl-\` 退出并显示堆栈信息 (*stack trace*)

`attach` 命令的语法是：

```
docker attach container
```

下面是一张显示执行 `attach` 命令的截图。



### *diff:*

*Docker* 提供了一个非常强大的命令 `diff`，它可以列出容器内发生变化的文件和目录。这

些变化包括添加 (*A-add*)、删除 (*D-delete*)、修改 (*C-change*)。该命令便于 *Debug*，并支持快速的共享环境。

语法是：

```
docker diff container
```

截图显示 `diff` 的执行。



### *events:*

打印指定时间内的容器的实时系统事件。

### *import:*

*Docker* 可以导入远程文件、本地文件和目录。使用 *HTTP* 的 *URL* 从远程位置导入，而本地文件或目录的导入需要使用 `-` 参数。从远程位置导入的语法是：

```
docker import http://example.com/example.tar
```

截图表示本地文件：



### *export:*

类似于 `import`，`export` 命令用于将容器的系统文件打包成 `tar` 文件。

下图描述了其执行过程：



### *cp:*

这个命令是从容器内复制文件到指定的路径上。语法如下：

```
docker cp container:path hostpath.
```

截图展示了 `cp` 命令的执行。



### *login:*

此命令用来登录到 *Docker registry* 服务器，语法如下：

```
docker login [options] [server]
```

如要登录自己主机的 *registry* 请使用：

```
docker login localhost:8080
```



### *inspect:*

`Docker inspect` 命令可以收集有关容器和镜像的底层信息。这些信息包括：

- 容器实例的 *IP* 地址

- 端口绑定列表
- 特定端口映射的搜索
- 收集配置的详细信息

该命令的语法是：

```
docker inspect container/image
```



### **kill:**

发送 **SIGKILL** 信号来停止容器的进程。语法是：

```
docker kill [options] container
```



### **rmi:**

该命令可以移除一个或者多个镜像，语法如下：

```
docker rmi image
```

镜像可以有多个标签链接到它。在删除镜像时，你应该确保删除所有相关的标签以避免错误。下图显示了该命令的示例。



### **wait:**

阻塞对指定容器的其它调用方法，直到容器停止后退出阻塞。





**load:**

该命令从 `tar` 文件中载入镜像或仓库到 `STDIN`。

截图显示载入 `app_box.tar` 到 `STDIN`:



**save:**

类似于 `load`，该命令保存镜像为 `tar` 文件并发送到 `STDOUT`。语法如下：

```
docker save image
```

简单截图示例如下：



*Docker* 系列教程的下一篇文章我们将探讨 *Docker APIs*。

原文链接：[Part 6: The Next 15 Commands](#)（翻译：[田浩浩](#) 审校：[李颖杰](#)）

## Docker 入门教程（七）Docker API

【编者的话】*DockerOne* 组织翻译了 *Flux7* 的 *Docker* 入门教程，本文是系列入门教程的第七篇，重点介绍了 *Docker Registry API* 和 *Docker Hub API*。

纵观我们的 *Docker* 系列教程，我们已经讨论了很多重要的 *Docker* 组件与命令。在本文中，我们将继续深入学习 *Docker*：剖析 *Docker APIs*。

*Docker* 提供了很多的 *API* 以便用户使用。这些 *API* 包含四个方面：

- *Docker Registry API*
- *Docker Hub API*
- *Docker OAuth API*
- *Docker Remote API*

具体到这篇文章，我们将讨论 *Docker Registry API* 以及 *Docker Hub API*。

## *Docker Registry API*

*Docker Registry API* 是 *Docker Registry* 的 *REST API*，它简化了镜像和仓库的存储。

该 *API* 不能访问用户帐户或者获得授权。你可以阅读 *Docker* 系列教程的第四章，以了解更多有关 *Registry* 的类型（译者注：*Docker* 中有几种不同的 *Registry*）。

### *Extract image layer:*

取出镜像层：

```
GET /v1/images/(image_id)/layer
```



## *Insert image layer:*

插入镜像层:

```
PUT /v1/images/(image_id)/layer
```

## *Retrieve an image:*

检索镜像:

```
GET /v1/images/(image_id)/json
```

## *Retrieve roots of an image:*

检索根镜像:

```
GET /v1/images/(image_id)/ancestry
```

## *Obtain all tags or specific tag of a repository:*

获取库里所有的标签或者指定标签:

```
GET /v1/repositories/(namespace)/(repository)/tags
```

或者

```
GET /v1/repositories/(namespace)/(repository)/tags/(tag*)
```



## *Delete a tag:*

删除标签:

```
DELETE /v1/repositories/(namespace)/(repository)/tags/(tag*)
```



## Status check of registry:

registry 状态检查:

```
GET /v1/_ping
```



## Docker Hub API

Docker Hub API 是 Docker Hub 的一个简单的 REST API。再提醒一下, 请参考 [Docker 系列教程的第四篇文章](#) 了解 Docker Hub。Docker Hub 通过管理校验 (checksums) 以及公共命名空间 (public namespaces) 来控制着用户帐户和授权。该 API 还支持有关用户仓库和 library 仓库的操作。

首先, 让我们来看看特殊的 library 仓库 (需要管理员权限) 的命令:

1. 创建一个新的仓库。使用以下命令可以创建新的 library 仓库:

```
PUT /v1/repositories/(repo_name)/
```

其中, `repo_name` 是新的仓库名称。

2. 删除已经存在的仓库。命令如下:

```
DELETE /v1/repositories/(repo_name)/
```

其中, `repo_name` 是要删除的仓库名称。

3. 更新仓库镜像。命令如下:

```
PUT /v1/repositories/(repo_name)/images
```

4. 从仓库中获取镜像。命令如下：

```
GET /v1/repositories/(repo_name)/images
```

5. 授权。使用 *Token* 获取仓库授权，如下：

```
PUT /v1/repositories/(repo_name)/auth
```

接下来，让我们来看看用户仓库的命令。*library* 仓库与用户仓库命令之间的主要区别是命名空间的使用。

1. 创建用户仓库。命令如下：

```
PUT /v1/repositories/(namespace)/(repo_name)/
```



2. 删除用户仓库，命令如下：

```
DELETE /v1/repositories/(namespace)/(repo_name)/
```



3. 更新用户仓库镜像，命令如下：

```
PUT /v1/repositories/(namespace)/(repo_name)/images
```



4. 从仓库中下载镜像。如下：

```
GET /v1/repositories/(namespace)/(repo_name)/images
```



5.验证用户登录，如下：

```
GET /v1/users
```



6.添加新用户，命令如下：

```
POST /v1/users
```

7.更新用户信息，如下：

```
PUT /v1/users/(username)/
```

现在，我们已经走过了 *Docker API* 之旅的第一站，第二站是有关 *Docker OAuth* 以及

*Remote API* 的内容，我们将在 [Docker](#) 系列教程的下一篇见。

原文链接：[Ultimate Guide for Docker APIs](#)（翻译：田浩浩 审校：李颖杰）

## Docker 入门教程（八）Docker Remote API

【编者的话】*DockerOne* 组织翻译了 *Flux7* 的 *Docker* 入门教程，本文是系列入门教程的第八篇，重点介绍了 *Docker Remote API*。

在 [Docker](#) 系列教程的上一篇文章中，我们学习了 *Docker Hub* 以及 *Docker Registry API*。在本文中，让我们来看看 *Docker Remote API*。

## Docker Remote API

*Docker Remote API* 是一个取代远程命令行界面 (*rcli*) 的 *REST API*。本文中,我们将使用命令行工具 *cURL* 来处理 *url* 相关操作。*cURL* 可以发送请求、获取以及发送数据、检索信息。

**容器列表** 获取所有容器的清单:

```
GET /containers/json
```



**创建新容器**。命令如下:

```
POST /containers/create
```



**监控容器**。使用容器 *id* 获取该容器底层信息:

```
GET /containers/(id)/json
```



**进程列表**。获取容器内进程的清单:

```
GET /containers/(id)/top
```



**容器日志。**获取容器的标准输出和错误日志：

```
GET /containers/(id)/logs
```



**导出容器。**导出容器内容：

```
GET /containers/(id)/export
```



**启动容器。**如下：

```
POST /containers/(id)/start
```



**停止容器。**命令如下：

```
POST /containers/(id)/stop
```



**重启容器，**如下：

```
POST /containers/(id)/restart
```



**终止容器：**

```
POST /containers/(id)/kill
```





现在，我们已经带你走过了 *Docker API* 的第二站，*Docker* 系列教程的下一篇文章会介绍有关镜像的 *Docker Remote API* 命令。我们所有的 *Docker* 系列教程你都可以[在这里](#)找到。

原文链接：[Docker Remote API](#)（翻译：[田浩浩](#) 审校：李颖杰）

## Docker 入门教程（九）10 个镜像相关的 API

【编者的话】*DockerOne* 组织翻译了 *Flux7* 的 *Docker* 入门教程，本文是系列入门教程的第九篇，重点介绍了镜像相关的 *Docker Remote API*。

在 *Docker* 系列教程的上一篇文章中，我们讨论了 *Docker Remote API*，并具体学习了有关容器的命令。在这篇文章中，我们将讨论有关镜像的命令。

### 创建镜像

镜像可以通过以下两种方式来创建：

- 从 *Registry* 中提取
- 导入镜像

```
POST /images/create
```

截图示例：



## 利用容器创建镜像

`POST /commit`

截图示例：



获取镜像清单：

`GET /images/json`

截图示例：



## *Insert a File*

导入指定的路径文件：

`POST /images/(name)/insert`

截图示例：



删除镜像:

```
DELETE /images/(name)
```

截图示例:



推送镜像到 *Registry*

```
POST /images/(name)/push
```

截图示例:



*Tag* 镜像

```
POST /images/(name)/tag
```

截图示例:



搜索镜像:

```
GET /images/search
```

截图示例:



## 查看镜像历史

GET /images/(name)/history

截图示例：



## 构建镜像

POST /build

截图示例：



原文链接：[10 Docker Remote API Commands for Images](#)（翻译：田浩浩 审校：李颖杰）

参考网址: <http://www.centoscn.com/image-text/install/2014/1128/4202.html>

**Docker** 是一个开源的应用容器引擎, 可以轻松的为任何应用创建一个轻量级的、可移植的、自给自足的容器。利用 Linux 的 LXC、AUFS、Go 语言、cgroup 实现了资源的独立, 可以很轻松的实现文件、资源、网络等隔离, 其最终的目标是实现类似 PaaS 平台的应用隔离。

**Docker** 值得关注的特性:

文件系统隔离: 每个进程容器运行在一个完全独立的根文件系统里。

资源隔离: 系统资源, 像 CPU 和内存等可以分配到不同的容器中, 使用 cgroup。

网络隔离: 每个进程容器运行在自己的网络空间, 虚拟接口和 IP 地址。

日志记录: **Docker** 将会收集和记录每个进程容器的标准流 (stdout/stderr/stdin), 用于实时检索或批量检索。

变更管理: 容器文件系统的变更可以提交到新的映像中, 并可重复使用以创建更多的容器。无需使用模板或手动配置。

交互式 shell: **Docker** 可以分配一个虚拟终端并关联到任何容器的标准输入上, 例如运行一个一次交互 shell。

**Docker** 通常用于如下场景:

web 应用的自动化打包和发布;

自动化测试和持续集成、发布;

在服务型环境中部署和调整数据库或其他的后台应用;

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。

本文介绍如何在 RedHat/CentOS 环境下安装 Docker。官方文档要求 Linux kernel 至少 3.8 以上, 且 docker 只能运行在 64 位的系统中。由于 RHEL6 和 CentOS6 的内核版本为 2.6, 因此必须要先升级内核。

下面以 CentOS6.5 (64 位) 为例, 介绍下 docker 安装步骤和使用方法:

## 一、升级内核 (带 aufs 模块)

1、yum 安装带 aufs 模块的 3.10 内核（或到这里下载 kernel 手动安装：

<http://down.51cto.com/data/1903250>）

```
cd /etc/yum.repos.d
wget http://www.hop5.in/yum/el6/hop5.repo
yum install kernel-ml-aufs kernel-ml-aufs-devel
```

2、修改 grub 的主配置文件/etc/grub.conf，设置 default=0，表示第一个 title 下的内容为默认启动的 kernel（一般新安装的内核在第一个位置）。

```
[root@localhost ~]# cat /etc/grub.conf
# grub.conf generated by anaconda
#
# Note that you do not have to rerun grub after making changes to this file
# NOTICE: You have a /boot partition. This means that
#         all kernel and initrd paths are relative to /boot/, eg.
#         root (hd0,0)
#         kernel /vmlinuz-version ro root=/dev/vda3
#         initrd /initrd-[generic-]version.img
#boot=/dev/vda
default=0
timeout=5
splashimage=(hd0,0)/grub/splash.xpm.gz
hiddenmenu
title CentOS (3.10.5-3.el6.x86_64)
    root (hd0,0)
    kernel /vmlinuz-3.10.5-3.el6.x86_64 ro root=UUID=5fee524c-3edb-4625-b5b7-c7906
    initrd /initramfs-3.10.5-3.el6.x86_64.img
title CentOS (2.6.32-431.el6.x86_64)
    root (hd0,0)
    kernel /vmlinuz-2.6.32-431.el6.x86_64 ro root=UUID=5fee524c-3edb-4625-b5b7-c7906
    initrd /initramfs-2.6.32-431.el6.x86_64.img
[root@localhost ~]#
```

3、重启系统，这时候你的内核就成功升级了。

```
[root@localhost ~]# uname -r
3.10.5-3.el6.x86_64
```

查看内核是否支持 aufs：

```
[root@localhost ~]# grep aufs /proc/filesystems
nodev      aufs
```

## 二、安装 docker

1、首先关闭 selinux：

```
setenforce 0
```

```
sed -i '/^SELINUX=/c\SELINUX=disabled' /etc/selinux/config
```

2、在 Fedora EPEL 源中已经提供了 docker-io 包，下载安装 epel:

3、yum 安装 docker-io:

```
rpm -ivh http://mirrors.sohu.com/fedora-epel/6/x86_64/epel-release-6-8.noarch.rpm
```

```
sed -i 's/^mirrorlist=https/mirrorlist=http/' /etc/yum.repos.d/epel.repo
```

```
yum -y install docker-io
```

```
Dependencies Resolved
```

Package	Arch	Version	Repository	Size
Installing:				
docker-io	x86_64	1.3.1-2.el6	epel	4.3 M
Installing for dependencies:				
libcgroup	x86_64	0.40.rc1-15.el6_6	updates	129 k
lua-alt-getopt	noarch	0.7.0-1.el6	epel	6.9 k
lua-filesystem	x86_64	1.4.2-1.el6	epel	24 k
lua-lxc	x86_64	1.0.6-1.el6	epel	15 k
lxc	x86_64	1.0.6-1.el6	epel	120 k
lxc-libs	x86_64	1.0.6-1.el6	epel	248 k
xz	x86_64	4.999.9-0.5.beta.20091007git.el6	base	137 k
Updating for dependencies:				
xz-libs	x86_64	4.999.9-0.5.beta.20091007git.el6	base	89 k

```
Transaction Summary
```

Install	8 Package(s)
Upgrade	1 Package(s)

4、启动 docker:

```
service docker start
```

```
[root@master ~]# service docker start
```

```
Starting cgconfig service: Error: cannot mount cpuset to /cgroup/cpuset:  
Device or resource busy
```

```
/sbin/cgconfigparser; error loading /etc/cgconfig.conf: Cgroup mounting  
failed
```

```
Failed to parse /etc/cgconfig.conf or /etc/cgconfig.d [FAILED]
```

```
Starting docker: [ OK ]
```

出现这个错误的解决办法:

<http://stackoverflow.com/questions/25183063/docker-on-rhel-6-cgroup-m>

[ounting-failing](#)

### 【办法】

I have the same issue.

```
[root@localhost ~]# service docker restart
Stopping docker: [ OK ]
Starting cgconfig service: Error: cannot mount cpuset to /cgroup/cpus
/sbin/cgconfigparser; error loading /etc/cgconfig.conf: Cgroup mounti
Failed to parse /etc/cgconfig.conf or /etc/cgconfig.d [FAILED]
Starting docker: [ OK ]
```

(1) check cgconfig status

```
# /etc/init.d/cgconfig status
```

if it stopped, restart it

```
# /etc/init.d/cgconfig restart
```

check cgconfig is running

```
[root@localhost ~]# /etc/init.d/cgconfig status
Running
```

(2) check cgconfig is on

```
# chkconfig --list cgconfig
```

```
cgconfig 0:off 1:off 2:off 3:off 4:off 5:off 6:off
```

if cgconfig is off, turn it on

```
[root@localhost ~]# chkconfig --list cgconfig
cgconfig      0:off  1:off  2:on   3:on   4:on   5:on   6:off
```

(3) if still does not work, may be some cgroups modules is missing. In the kernel .config file, make menuconfig, add those modules into kernel and recompile and reboot

after that, it should be OK



```
[root@localhost ~]# service docker restart
Stopping docker: [ OK ]
Starting docker: [ OK ]
```

shareimprove this answer

```
[root@localhost ~]# service docker start http://qicheng0211.blog.51cto.com/
Starting cgconfig service: [ 确定 ]
Starting docker: [ 确定 ]
[root@localhost ~]#
```

5、查看 docker 版本:

```
[root@localhost ~]# docker version
Client version: 1.3.1
Client API version: 1.15
Go version (client): go1.3.3
Git commit (client): c78088f/1.3.1
OS/Arch (client): linux/amd64
Server version: 1.3.1
Server API version: 1.15
Go version (server): go1.3.3
Git commit (server): c78088f/1.3.1
[root@localhost ~]#
```

```
[root@master ~]# docker version
```

```
Client version: 1.7.1
Client API version: 1.19
Go version (client): go1.4.2
Git commit (client): 786b29d/1.7.1
OS/Arch (client): linux/amd64
Server version: 1.7.1
Server API version: 1.19
Go version (server): go1.4.2
Git commit (server): 786b29d/1.7.1
OS/Arch (server):
```

查看 docker 日志:

```
cat /var/log/docker
```

```
[root@master Desktop]# cat /var/log/docker
\nFri Nov 4 09:30:29 PDT 2016\n
time="2016-11-04T09:30:30.615758467-07:00" level=warning msg="You are
running linux kernel version 2.6.32-642.el6.x86_64, which might be
unstable running docker. Please upgrade your kernel to 3.10.0."
```

### 三、docker 命令的使用

1、直接输入 docker 命令来查看所有的 **Options** 和 **Commands**。

查看某一个 command 的详细使用方法: **docker COMMAND --help**

```
[root@localhost ~]# docker pull --help

Usage: docker pull [OPTIONS] NAME[:TAG]

Pull an image or a repository from the registry

-a, --all-tags=false    Download all tagged images in the repository
[root@localhost ~]#
```

2、搜索可用的 docker 镜像: **docker search NAME**

```
[root@localhost ~]# docker search centos
```

NAME	DESCRIPTION	STARS	OFFICIAL	AUTOMATED
centos	The official build of CentOS.	624	[OK]	
tianon/centos	CentOS 5 and 6, created using rinse instea...	28		
ansible/centos7-ansible	Ansible on Centos7	16		[OK]
saltstack/centos-6-minimal		8		[OK]
tutum/centos	Centos image with SSH access. For the root...	8		[OK]
blalor/centos	Bare-bones base CentOS 6.5 image	7		[OK]
ariya/centos6-teamcity-server	TeamCity Server 8.1 on CentOS 6	6		[OK]
steef/ graphite-centos	CentOS 6.x with Graphite and Carbon via ng...	6		[OK]
dockerfiles/centos-lamp		6		[OK]
jdeathe/centos-ssh-apache-php	CentOS-6 6.5 x86_64 / Apache / PHP / PHP m...	5		[OK]
gluster/gluster	GlusterFS 3.5 - CentOS 6.5 Docker repo	5		[OK]
berngp/docker-zabbix	Runs Zabbix Server and Zabbix Web UI on a ...	5		[OK]
tutum/centos-6.4	DEPRECATED. Use tutum/centos:6.4 instead. ...	5		[OK]
cern/centos-wlcg-wn	CentOS 6 image with pre-installed softwa...	4		
ariya/centos6-teamcity-agent	Build agent for TeamCity 8.1	4		[OK]
ingensi/oracle-jdk	Official Oracle JDK installed on centos.	4		[OK]
openshift/centos-mongodb		4		[OK]

3、下载镜像: **docker pull NAME[:TAG]**

比如获取最新的 centos 镜像: **docker pull centos:latest**

注意：这里要写用 `docker search` 搜索到的完整的镜像名。

4、查看安装的镜像：`docker images [NAME]`

```
[root@master ~]# docker images busybox
```

REPOSITORY	TAG	IMAGE ID	CREATED	VIRTUAL SIZE
busybox	latest	9967c5ad88de	6 weeks ago	1.093 MB

```
[root@localhost ~]# docker images centos
REPOSITORY    TAG       IMAGE ID       CREATED        VIRTUAL SIZE
centos        centos7   ae0c2d0bdc10  3 weeks ago   224 MB
centos        latest    ae0c2d0bdc10  3 weeks ago   224 MB
[root@localhost ~]#
```

5、在 docker 容器中运行命令：`docker run IMAGE [COMMAND] [ARG...]`

`docker run` 命令有两个参数，一个是镜像名，一个是要在镜像中运行的命令。

注意：`IMAGE=REPOSITORY[:TAG]`，如果 `IMAGE` 参数不指定镜像的 `TAG`，默认 `TAG` 为 `latest`。

```
[root@master ~]# docker run busybox echo Hello World
```

```
Hello World
```

在刚刚下载的镜像中输出“hello word”：`docker run centos echo 'hello world!'`

```
[root@localhost ~]# docker run centos echo 'hello world!'
hello world!
[root@localhost ~]#
```

6、列出容器：`docker ps -a`

查看最近生成的容器：`docker ps -l`

```
[root@master ~]# docker ps -l
```

CONTAINER ID	IMAGE	COMMAND	CREATED
STATUS	PORTS	NAMES	
f75f388773d2	busybox	"echo Hello World"	About a minute ago
Exited (0)	About a minute ago		cocky_brattain

由返回结果可见还包括了最近一次运行的命令

查看正在运行的容器: `docker ps`

```
[root@master ~]# docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED
STATUS	PORTS	NAMES	

7、显示容器的标准输出: `docker logs CONTAINERID`

无需拷贝完整的 id, 一般写最开始的三至四个字符即可。

```
[root@master ~]# docker ps -l
```

CONTAINER ID	IMAGE	COMMAND	CREATED
STATUS	PORTS	NAMES	
f75f388773d2	busybox	"echo Hello World"	About a minute ago
Exited (0)	About a minute ago		cocky_brattain

```
[root@master ~]# docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED
VIRTUAL SIZE			
busybox	latest	9967c5ad88de	6 weeks ago
1.093 MB			

```
[root@master ~]# docker logs f75
```

Hello World #这是最近一次命令执行的返回结果

```
[root@localhost ~]# docker run centos echo 'hello world!'
hello world!
[root@localhost ~]# docker ps -l
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS        NAMES
6f1183f7e2db   centos:centos7 "echo 'hello world!'"    11 seconds ago Exited (0) 10 seconds ago evil_turing
[root@localhost ~]# docker logs 6f11
hello world!
[root@localhost ~]#
```

8、在容器中安装新程序, 比如安装 `ifconfig` 命令 (centos7 默认没有 `ifconfig`):

`docker run centos yum install net-tools -y` #与平常的命令无异

如果 `yum` 不指定 `-y` 参数的话, `yum` 命令会进入交互模式, 需要用户输入命令来进行确认, 在 `docker` 环境中是无法响应这种交互的。但使用 `docker run` 的 `-i -t` 参数就会响应这种交互, 用户可以输入命令了, 比如: `docker run -i -t centos yum install net-tools`

9、保存对容器的修改并生成新的镜像：`docker commit CONTAINERID`

`[REPOSITORY[:TAG]]`

`REPOSITORY` 参数可以是新的镜像名字，也可以是旧的镜像名；如果和旧的镜像名和 `TAG` 都相同，会覆盖掉旧的镜像。

```
[root@master ~]# docker commit f75f388773d2 busybox:new
4a435d816e2bea471b1d6ad982037d9738797623a75ab381bf17a466cb387da3
[root@master ~]# docker images
```

	REPOSITORY	TAG	IMAGE ID	CREATED
VIRTUAL SIZE				
	busybox	new	4a435d816e2b	5 seconds
ago	1.093 MB			
	busybox	latest	9967c5ad88de	6 weeks ago
1.093 MB				

```
[root@localhost ~]# docker ps -l
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS              NAMES
3d473cd937f4       centos:centos7     "yum install net-  2 minutes ago      Exited (0)          About a minute ago elegant_davinci

[root@localhost ~]# docker commit 3d47 centos:ifconfig 保存对容器的修改, 生成新的镜像
c347f239c87b911c2bb663e80c72ffc5f9d199c7870a37f88772e789d52d98ed

[root@localhost ~]# docker images centos
REPOSITORY          TAG                IMAGE ID            CREATED             VIRTUAL SIZE
centos               ifconfig          c347f239c87b       5 seconds ago      297.1 MB
centos               centos7           ae0c2d0bdc10       3 weeks ago        224 MB
centos               latest            ae0c2d0bdc10       3 weeks ago        224 MB

[root@localhost ~]# docker run centos:ifconfig ifconfig 在新的镜像里执行ifconfig命令
eth0: flags=3<UP,BROADCAST> mtu 1500
    inet 172.17.0.49 netmask 255.255.0.0 broadcast 0.0.0.0
    inet6 fe80::42:acff:fe11:31 prefixlen 64 scopeid 0x20<link>
    ether 02:42:ac:11:00:31 txqueuelen 0 (Ethernet)
    RX packets 1 bytes 90 (90.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 1 bytes 90 (90.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 0 (Local Loopback)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

10、停止正在运行的容器: **docker stop CONTAINERID**

默认等待 10 秒钟再杀死指定容器。可以使用 -t 参数来设置等待时间。

```
[root@localhost ~]# docker run centos sleep 100 &
[1] 13897
[root@localhost ~]# docker ps -l
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS              PC
841ebe032f1b       centos:centos7     "sleep 100"        3 seconds ago      Up 2 seconds
[root@localhost ~]# docker stop 841ebe032f1b
841ebe032f1b
[1]+  Exit 255                  docker run centos sleep 100
```

11、查看容器或镜像的详细信息: **docker inspect CONTAINERID|IMAGE**

参数可以是容器的 ID 或者是镜像名 (NAME:TAG)。

```
[root@master ~]# docker inspect busybox:new
```

```
[
```

```
{
  "Id":
    "4a435d816e2bea471b1d6ad982037d9738797623a75ab381bf17a466cb387da3",
  "Parent":
    "9967c5ad88de8c101809f7f22d4774b6791fe46ac3033d57abf7ebb1dd8e36ee",
  "Comment": "",
  "Created": "2016-11-19T14:09:10.919471902Z",
  "Container":
    "f75f388773d29d6df28e352a2892380da632020b35cb496b74c9ffc3b8bb0c17",
  "ContainerConfig": {
    "Hostname": "f75f388773d2",
    "Domainname": "",
    "User": "",
    "AttachStdin": false,
    "AttachStdout": true,
    "AttachStderr": true,
    "PortSpecs": null,
    "ExposedPorts": null,
    "Tty": false,
    "OpenStdin": false,
    "StdinOnce": false,
    "Env": [
      "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
    ],
    "Cmd": [
      "echo",
      "Hello",
      "World"
    ],
    "Image": "busybox",
    "Volumes": null,
    "VolumeDriver": "",
    "WorkingDir": ""
  }
}
```

```
"Entrypoint": null,
"NetworkDisabled": false,
"MacAddress": "",
"OnBuild": null,
"Labels": {}
    },
"DockerVersion": "1.7.1",
"Author": "",
"Config": {
  "Hostname": "",
  "Domainname": "",
  "User": "",
  "AttachStdin": false,
  "AttachStdout": false,
  "AttachStderr": false,
  "PortSpecs": null,
  "ExposedPorts": null,
  "Tty": false,
  "OpenStdin": false,
  "StdinOnce": false,
  "Env": [
    "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
  ],
  "Cmd": [
    "echo",
    "Hello",
    "World"
  ],
  "Image": "",
  "Volumes": null,
  "VolumeDriver": "",
  "WorkingDir": "",
  "Entrypoint": null,
```



```

    "NetworkDisabled": false,
    "MacAddress": "",
    "OnBuild": null,
    "Labels": {}
  },
  "Architecture": "amd64",
  "Os": "linux",
  "Size": 0,
  "VirtualSize": 1093484
}
]

[root@master ~]#

```

```

[root@localhost ~]# docker run centos sleep 100 &
[1] 14092
[root@localhost ~]# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
09cddf7d80f2       centos:centos7     "sleep 100"        5 seconds ago
[root@localhost ~]# docker inspect 09cd
[
  {
    "Args": [
      "100"
    ],
    "Config": {
      "AttachStderr": true,
      "AttachStdin": false,
      "AttachStdout": true,
      "Cmd": [
        "sleep",
        "100"
      ],
      "CpuShares": 0,
      "Cpuset": "",
      "Domainname": "",
      "Entrypoint": null,
      "Env": null,
      "ExposedPorts": null,
      "Hostname": "09cd",
      "Image": "centos:centos7",
      "Labels": null,
      "Memory": 0,
      "NetworkDisabled": false,
      "OnBuild": null,
      "OpenStdin": false,
      "StdinOnce": false,
      "Tty": false,
      "User": "",
      "Volumes": null,
      "WorkingDir": ""
    },
    "HostConfig": {
      "Binds": null,
      "ContainerIDFile": null,
      "Cpuset": "",
      "CpuShares": 0,
      "DeviceRequests": null,
      "Devices": null,
      "Dns": null,
      "DnsOptions": null,
      "DnsSearch": null,
      "ExtraHosts": null,
      "IpcMode": "private",
      "Isolation": "default",
      "Links": null,
      "LogConfig": {
        "Config": {},
        "Driver": "json-file"
      },
      "Memory": 0,
      "MemoryReservation": 0,
      "MemorySwap": 0,
      "Mounts": null,
      "NetworkMode": "bridge",
      "OomKillDisable": false,
      "PidMode": "host",
      "PortMaps": null,
      "Privileged": false,
      "ReadonlyRootfs": false,
      "RestartPolicy": {
        "Name": "no",
        "MaximumRetryCount": 0
      },
      "Runtime": "runc",
      "SecurityOptions": null,
      "StorageOptions": null,
      "Tmpfs": null,
      "Ulimits": null,
      "UsernsMode": "default",
      "VolumeDriver": null,
      "VolumesFrom": null,
      "WorkingContainer": null,
      "XtreamDevice": null
    },
    "State": {
      "Status": "running",
      "Running": true,
      "Paused": false,
      "Restarting": false,
      "Dead": false,
      "OomKilled": false,
      "Pid": 14092,
      "ExitCode": 0,
      "Error": "",
      "StartedAt": "2017-07-11T11:11:11.111Z",
      "FinishedAt": "2017-07-11T11:11:11.111Z"
    }
  }
]

```

12、删除容器: `docker rm CONTAINERID`

查看所有容器 ID: `docker ps -a -q`

删除所有的容器: `docker rm $(docker ps -a -q)`

13、删除镜像: `docker rmi IMAGE`

```
[root@master ~]# docker rmi busybox:new
```

Untagged: busybox:new

Deleted:

4a435d816e2bea471b1d6ad982037d9738797623a75ab381bf17a466cb387da3

```
[root@localhost ~]# docker images centos
REPOSITORY          TAG                 IMAGE ID            CREATED
centos               ifconfig           fa7b6246e1f5       31 minutes ago
centos               latest             ae0c2d0bdc10       3 weeks ago
centos               centos7            ae0c2d0bdc10       3 weeks ago
[root@localhost ~]# docker rmi centos:ifconfig
Untagged: centos:ifconfig
Deleted: fa7b6246e1f5e6a3bd32c267fea707e99e46ff850b0b0bcab351d1e91b0b258d
Deleted: 565b78b7002ee7d9f52a0007b8fba85a15abf91310c2dcbd1a6ea46000b644ef
[root@localhost ~]# docker images centos
REPOSITORY          TAG                 IMAGE ID            CREATED
centos               latest             ae0c2d0bdc10       3 weeks ago
centos               centos7            ae0c2d0bdc10       3 weeks ago
[root@localhost ~]#
```

14. 查看 docker 的信息，包括 Containers 和 Images 数目、kernel 版本等。

```
[root@localhost ~]# docker info
Containers: 5
Images: 17
Storage Driver: aufs
 Root Dir: /var/lib/docker/aufs
  Dirs: 27
Execution Driver: native-0.2
Kernel Version: 3.10.5-3.el6.x86_64
Operating System: <unknown>
WARNING: No swap limit support
```

## 四、创建容器并登入的操作

1、创建一个新容器并登入：`docker run -i -t IMAGE /bin/bash`

使用 image 创建 container 并进入交互模式, login shell 是/bin/bash, 现在可以自由的对容器进行操作了。最后使用 `exit` 退出容器。

注意：如果 IMAGE 参数不指定 TAG，默认 TAG 为 latest。

```
[root@da09cd751694 tmpfiles]# echo hello Docker >>test.txt
```

```
[root@da09cd751694 tmpfiles]# more test.txt
```

This is the first file...

hello Docker

2、启动一个退出的容器: `docker start CONTAINERID`

3、attach 到运行中的容器: `docker attach CONTAINERID`

```
[root@localhost ~]# docker run -i -t centos /bin/bash 创建一个新的容器, 并登入
[root@29e28dcb2faa /]#
[root@29e28dcb2faa /]#
[root@29e28dcb2faa /]# exit 退出容器
exit
[root@localhost ~]# docker ps -l
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
29e28dcb2faa        centos:centos7     "/bin/bash"        21 seconds ago     Exited (
[root@localhost ~]# docker start 29e2 启动这个容器
29e2
[root@localhost ~]# docker ps -l
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
29e28dcb2faa        centos:centos7     "/bin/bash"        35 seconds ago     Up 3 sec
[root@localhost ~]# docker attach 29e2 attach到这个容器

[root@29e28dcb2faa /]#
[root@29e28dcb2faa /]#
```

```
[root@master ~]# docker attach da
```

You cannot attach to a stopped container, start it first

```
[root@master ~]# docker start da
```

da

```
[root@master ~]# docker ps -l
```

CONTAINER ID	IMAGE	COMMAND	CREATED
STATUS	PORTS	NAMES	
da09cd751694	centos	"/bin/bash"	16 minutes ago
6 seconds		cocky_pasteur	Up

```
[root@master ~]# docker attach da
```

^C

```
[root@da09cd751694 /]# ^C
```

## Docker 手册

参考: <http://www.docker.org.cn/book/docker/what-is-docker-16.html>

## Docker 入门教程

- 1 什么是 Docker?
- 2 关于 docker 入门教程
- 3 准备
- 4 搜索可用 docker 镜像
- 5 下载容器镜像
- 6 在 docker 容器中运行 hello world!
- 7 在容器中安装新的程序
- 8 保存对容器的修改
- 9 运行新的镜像
- 10 检查运行中的镜像
- 11 发布自己的镜像

## 什么是 Docker?

**简介:** Docker 是一个开源的引擎, 可以轻松地为任何应用创建一个**轻量级的、可移植的、自给自足的容器**。开发者在笔记本上编译测试通过的容器可以批量地在生产环境中部署, 包括 VMs (虚拟机)、bare metal、OpenStack 集群和其他的基础应用平台。

**Docker 通常用于如下场景:**

- web 应用的自动化打包和发布;
- 自动化测试和持续集成、发布;
- 在服务型环境中部署和调整数据库或其他的后台应用;
- 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。

## 关于 docker 入门教程

docker 入门教程翻译自 docker 官方网站的 Docker getting started 教程, 官方网站: <https://docs.docker.com/linux/started/>

官方网站是一个交互的教程, 在左侧是相应的说明, 右侧是一个交互的终端, 输入预期的目录, 可以跳到下一步, 大家可以参考我们的翻译, 在官网上面运行相应的命令, 以验证效果。

译者按: 之前的交互教程在新版本的 docker 官网上已无法找到, 但核心的概念流程没有变, 仍然可以参考。(2015/9/16)

## 准备

### 准备开始

Docker 系统有两个程序：**docker 服务端**和**docker 客户端**。其中 docker 服务端是一个服务进程，管理着所有的容器。docker 客户端则扮演着 docker 服务端的远程控制器，可以用来控制 docker 的服务端进程。大部分情况下，docker 服务端和客户端运行在一台机器上。

### 目标：

检查 docker 的版本，这样可以用来确认 docker 服务在运行并可通过客户端链接。

### 提示：

可以通过在终端输入 docker 命令来查看所有的参数。  
官网的在线模拟器只提供了有限的命令，无法保证所有的命令可以正确执行。

正确的命令：

**\$docker version**

```
Welcome to the interactive Docker tutorial
you@tutorial:~$ docker version
Docker Emulator version 0.1.3

Emulating:
Client version: 0.5.3
Server version: 0.5.3
Go version: go1.1
you@tutorial:~$
```



Well done! Let's move to

## 搜索可用 docker 镜像

### 搜索可用的 docker 镜像

使用 docker 最简单的方式莫过于从现有的容器镜像开始。Docker 官方网站专门有一个页面来存储所有可用的镜像，网址是：[index.docker.io](https://index.docker.io)。你可以通过浏览这个网页来查找你想要使用的镜像，或者使用命令行的工具来检索。

### 提示：

命令行的格式为：docker search 镜像名字

正确的命令：

**\$docker search tutorial**

```
Welcome to the interactive Docker tutorial
you@tutorial:~$ docker version
Docker Emulator version 0.1.3

Emulating:
Client version: 0.5.3
Server version: 0.5.3
Go version: go1.1
you@tutorial:~$ docker search tutorial
Found 1 results matching your query ("tutorial")
NAME                DESCRIPTION
learn/tutorial      An image for the interactive tutorial
you@tutorial:~$
```



You found it!

## 下载容器镜像

### 学会使用 **docker** 命令来下载镜像

下载镜像的命令非常简单，使用 **docker pull** 命令即可。(译者按：docker 命令和 git 有一些类似的地方)。在 docker 的镜像索引网站上面，镜像都是按照用户名/镜像名的方式来存储的。有一组比较特殊的镜像，比如 **ubuntu** 这类基础镜像，经过官方的验证，值得信任，可以直接用镜像名来检索到。

提示：

执行 **pull** 命令的时候要写完整的名字，比如 **"learn/tutorial"**。

正确的命令：

**\$docker pull learn/tutorial**



```
you@tutorial:~$ docker version
Docker Emulator version 0.1.3
```

```
Emulating:
```

```
Client version: 0.5.3
```

```
Server version: 0.5.3
```

```
Go version: go1.1
```

```
you@tutorial:~$ docker search tutorial
```

```
Found 1 results matching your query ("tutorial")
```

NAME	DESCRIPTION
learn/tutorial	An image for the int

```
you@tutorial:~$ docker pull learn/tutorial
```

```
Pulling repository learn/tutorial
```

```
2013/06/19 19:27:03 HTTP code: 404
```

```
you@tutorial:~$ docker pull learn/tutorial
```

```
Pulling repository learn/tutorial from https://index.docker.io/v1
```

```
Pulling image 8dbd9e392a964056420e5d58ca5cc376ef18e2de93b5cc90e868a1bbc8318c1c (pre
```

```
Pulling image b750fe79269d2ec9a3c593ef05b4332b1d1a02a62b4accb2c21d589ff2f5f2dc (12.
```

```
Pulling image 27cf784147099545 () from tutorial
```

```
you@tutorial:~$
```



Cool. Look at the results.  
Docker has downloaded  
Docker all images (except  
made up of several cumu

## 在 docker 容器中运行 hello world!

### 在 docker 容器中运行 hello world!

**docker 容器可以理解为在沙盒中运行的进程。**这个沙盒包含了该进程运行所必须的资源，包括文件系统、系统类库、shell 环境等等。但这个沙盒默认是不会运行任何程序的。你需要在沙盒中运行一个进程来启动某一个容器。这个进程是该容器的唯一进程，所以当该进程结束的时候，容器也会完全的停止。

提示：

**docker run 命令有两个参数，一个是镜像名，一个是要在镜像中运行的命令。**

**正确的命令：**

**\$docker run learn/tutorial echo "hello word"**

```
you@tutorial:~$ docker run learn/tutorial echo "hello word"
hello word
you@tutorial:~$
```



Great! Hellooooo World!  
You have just started a container, executed a program inside, and the program stopped, so did the container.

## 在容器中安装新的程序

### 在容器中安装新的程序

下一步我们要做的事情是在容器里面安装一个简单的程序(ping)。我们之前下载的 tutorial 镜像是基于 ubuntu 的，所以你可以使用 ubuntu 的 apt-get 命令来安装 ping 程序：apt-get install -y ping。

备注：apt-get 命令执行完毕之后，容器就会停止，但对容器的改动不会丢失。

提示：

在执行 apt-get 命令的时候，要带上-y 参数。如果不指定-y 参数的话，apt-get 命令会进入交互模式，需要用户输入命令来进行确认，但在 docker 环境中是无法响应这种交互的。

正确的命令：

**\$docker run learn/tutorial apt-get install -y ping**

```
you@tutorial:~$ docker run learn/tutorial apt-get install -y ping
Reading package lists...
Building dependency tree...
The following NEW packages will be installed:
  iputils-ping
0 upgraded, 1 newly installed, 0 to remove and 0 not installed.
Need to get 56.1 kB of archives.
After this operation, 143 kB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu/ precise/main amd64 iputils-ping
debconf: delaying package configuration, since apt-utils is not installed
Fetched 56.1 kB in 1s (50.3 kB/s)
Selecting previously unselected package iputils-ping.
(Reading database ... 7545 files and directories currently installed.)
Unpacking iputils-ping (from .../iputils-ping_3%3a20101006-1ubuntu1_amd64.deb) ...
Setting up iputils-ping (3:20101006-1ubuntu1) ...
you@tutorial:~$
```



That worked! You have installed a new program on top of a base image. Your container's filesystem have been kept saved.



## 保存对容器的修改

### 保存对容器的修改

当你为某一个容器做了修改之后（通过在容器中运行某一个命令），可以把对容器的修改保存下来，这样下次可以从保存后的最新状态运行该容器。`docker` 中保存状态的过程称之为 `committing`，它保存的新旧状态之间的区别，**从而产生一个新的版本**。

目标：

首先使用 `docker ps -l` 命令获得安装完 `ping` 命令之后容器的 `id`。然后把这个镜像保存为 `learn/ping`。


提示：

1. 运行 `docker commit`，可以查看该命令的参数列表。
2. 你需要指定要提交保存容器的 `ID`。(译者按：通过 `docker ps -l` 命令获得)
3. 无需拷贝完整的 `id`，通常来讲最开始的三至四个字母即可区分。（译者按：非常类似 `git` 里面的版本号）

正确的命令：

`$docker commit 698 learn/ping`

```
you@tutorial:~$ docker ps -l
ID                IMAGE             COMMAND
6982a9948422      ubuntu:12.04      apt-get
you@tutorial:~$ docker commit 698 learn/ping
effb66b31edb
you@tutorial:~$
```



That worked! Please take  
returned a new ID. This i

执行完 `docker commit` 命令之后，会返回新版本镜像的 `id` 号。

## 运行新的镜像

### 运行新的镜像

ok，到现在为止，你**已经建立了一个完整的、自成体系的 `docker` 环境**，并且安装了 `ping` 命令在里面。它可以在任何支持 `docker` 环境的系统中运行啦！（译者按：是不是很神奇呢？）让我们来体验一下吧！

提示：

一定要使用新的镜像名 `learn/ping` 来运行 `ping` 命令。（译者按：最开始下载的 `learn/tutorial` 镜像中是没有 `ping` 命令的）

正确的命令：

**\$ docker run learn/ping ping www.google.com**

```
you@tutorial:~$ docker run learn/ping ping www.google.com
```

```
PING www.google.com (74.125.239.129) 56(84) bytes of data:
64 bytes from nuq05s02-in-f20.1e100.net: icmp_seq=1 ttl=64 time=0.125 ms
64 bytes from nuq05s02-in-f20.1e100.net: icmp_seq=2 ttl=64 time=0.125 ms
64 bytes from nuq05s02-in-f20.1e100.net: icmp_seq=3 ttl=64 time=0.125 ms
64 bytes from nuq05s02-in-f20.1e100.net: icmp_seq=4 ttl=64 time=0.125 ms
64 bytes from nuq05s02-in-f20.1e100.net: icmp_seq=5 ttl=64 time=0.125 ms
64 bytes from nuq05s02-in-f20.1e100.net: icmp_seq=6 ttl=64 time=0.125 ms
64 bytes from nuq05s02-in-f20.1e100.net: icmp_seq=7 ttl=64 time=0.125 ms
64 bytes from nuq05s02-in-f20.1e100.net: icmp_seq=8 ttl=64 time=0.125 ms
64 bytes from nuq05s02-in-f20.1e100.net: icmp_seq=9 ttl=64 time=0.125 ms
64 bytes from nuq05s02-in-f20.1e100.net: icmp_seq=10 ttl=64 time=0.125 ms
^C
-> This would normally just keep going. However, this emulator does not support Ctrl-C
you@tutorial:~$
```



That worked! Note that you can press Ctrl-C to disconnect. The container is still running. This container will stop automatically.

## 检查运行中的镜像

### 检查运行中的镜像

现在你已经运行了一个 docker 容器，让我们来看下正在运行的容器。

使用 `docker ps` 命令可以查看所有正在运行中的容器列表，使用 `docker inspect` 命令我们可以查看更详细的关于某一个容器的信息。

提示：

可以使用镜像 id 的前面部分，不需要完整的 id。

正确的命令：

**\$ docker inspect efe**

```
you@tutorial:~$ docker ps
```

```
  ID          IMAGE          COMMAND
  efefdc74a1d5 learn/ping:latest ping www.google.com
you@tutorial:~$ docker inspect efe
[2013/07/30 01:52:26 GET /v1.3/containers/efefdc74a1d5]
{
  "ID": "efefdc74a1d5900d7d7a74740e5261c09f5f9",
  "Created": "2013-07-30T00:54:12.417119736Z",
  "Path": "ping",
  "Args": [
    "www.google.com"
  ]
}
```



Success! Have a look at the output of `docker inspect` to see the ip-address, status and other information.

## 发布自己的镜像

发布 docker 镜像

现在我们已经验证了新镜像可以正常工作，下一步我们可以将其发布到官方的索引网站。还记得我们最开始下载的 `learn/tutorial` 镜像吧，我们也可以把我们自己编译的镜像发布到索引页面，一方面可以自己重用，另一方面也可以分享给其他人使用。

目标：

把 `learn/ping` 镜像发布到 `docker` 的 `index` 网站。


提示：

1. `docker images` 命令可以列出所有安装过的镜像。
2. `docker push` 命令可以将某一个镜像发布到官方网站。
3. 你只能将镜像发布到自己的空间下面。这个模拟器登录的是 `learn` 帐号。

预期的命令：

**\$ `docker push learn/ping`**

```
you@tutorial:~$ docker images
ubuntu          latest
learn/tutorial  latest
learn/ping      latest
you@tutorial:~$ docker push learn/ping
The push refers to a repository [learn/ping]
Processing checksums
Sending image list
Pushing repository learn/ping (1 tags)
Pushing 8dbd9e392a964056420e5d58ca5cc376ef18e2de93b5cc90e868a1bbc8318c1c
Image 8dbd9e392a964056420e5d58ca5cc376ef18e2de93b5cc90e868a1bbc8318c1c already pushed
Pushing tags for rev [8dbd9e392a964056420e5d58ca5cc376ef18e2de93b5cc90e868a1bbc8318c1c]
1.
docker.io/v1/repositories/learn/ping/tags/latest}
Pushing a1dbb48ce764c6651f5af98b46ed052a5f751233d731b645a6c57f91a4cb7158
Pushing 11.5 MB/11.5 MB (100%)
Pushing tags for rev [a1dbb48ce764c6651f5af98b46ed052a5f751233d731b645a6c57f91a4cb7158]
1.
docker.io/v1/repositories/learn/ping/tags/latest}
```



All done!. You are now pushing the image to the index. You can now pull the image like pull, happens layer by layer.

## Docker 学习笔记：Docker 基础用法和命令帮助

参考: <http://www.docker.org.cn/dockerppt/106.html>

### 一、Docker 的基础用法

Docker 镜像首页，包括官方镜像和其它公开镜像

因为国情的原因，国内下载 Docker HUB 官方的相关镜像比较慢，可以使用 [docker.cn](http://docker.cn) 镜像，镜像保持和官方一致，关键是速度快，推荐使用。

#### 3.1 Search images

```
$ sudo docker search ubuntu
```

#### 3.2 Pull images

```
$ sudo docker pull ubuntu # 获取 ubuntu 官方镜像
```

```
$ sudo docker images # 查看当前镜像列表
```

#### 3.3 Running an interactive shell

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

- `docker run` - 运行一个容器
- `-t` - 分配一个 ( 伪 ) tty (link is external)
- `-i` - 交互模式 (so we can interact with it)
- `ubuntu:14.04` - 使用 ubuntu 基础镜像 14.04

- **/bin/bash - 运行命令 bash shell**

注: ubuntu 会有多个版本, 通过指定 tag 来启动特定的版本 [image]:[tag]

```
$ sudo docker ps # 查看当前运行的容器,
```

```
ps -a #列出当前系统所有的容器
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
--------------	-------	---------	---------	--------	-------	-------

6c9129e9df10	ubuntu:14.04	/bin/bash	6 minutes ago	Up 6 minutes		
--------------	--------------	-----------	---------------	--------------	--	--

```
cranky_babbage
```

### 3.4 相关快捷键

- 退出 : Ctrl-D or exit
- detach : Ctrl-P + Ctrl-Q
- attach: **docker attach CONTAINER-ID**

## 二、Docker 命令帮助

### 4.1 docker help

#### docker command

```
$ sudo docker # docker 命令帮助
```

```
Commands:
```

```
attach    Attach to a running container    # 当前 shell 下 attach 连接指定运行镜像
```

```
build      Build an image from a Dockerfile    # 通过 Dockerfile 定制镜像
```

```
commit     Create a new image from a container's changes # 提交当前容器为新的镜像
```

```
cp         Copy files/folders from the containers filesystem to the host path
```

```
# 从容器中拷贝指定文件或者目录到宿主机中
```

create	Create a new container	# 创建一个新的容器，同 run，但不启动容器
diff	Inspect changes on a container's filesystem	# 查看 docker 容器变化

```
[root@master testtmp]# docker ps -l
```

CONTAINER ID	IMAGE	COMMAND	CREATED
1f4d00854721	centos:latest	"/bin/bash"	About an hour ago
		Names	Exited
		jovial_meitner	

```
[root@master testtmp]# docker diff 1f4d0085
```

```
C /root
```

```
A /root/.bash_history
```

```
[root@master testtmp]#
```

events	Get real time events from the server	# 从 docker 服务获取容器实时事件
exec	Run a command in an existing container	# 在已存在的容器上运行命令
export	Stream the contents of a container as a tar archive	
		# 导出容器的内容流作为一个 tar 归档文件 [对应 import ]
history	Show the history of an image	# 展示一个镜像形成历史

```
root@master testtmp]# docker history centos
```

IMAGE	CREATED	CREATED BY	SIZE	COMMENT
97cad5e16cb6	2 weeks ago	/bin/sh -c #(nop)		CMD ["/bin/bash"]
05fe84bf6d3f	2 weeks ago	/bin/sh -c #(nop)		LABEL name=CentOS Base Ima
af0819ed1fac	2 weeks ago	/bin/sh -c #(nop)		ADD file:54df3580ac9fb66389
196.5 MB				
3690474eb5b4	11 weeks ago	/bin/sh -c #(nop)		MAINTAINER https://github.

images	List images	# 列出系统当前镜像
import	Create a new filesystem image from the contents of a tarball	
		# 从 tar 包中的内容创建一个新的文件系统映像 [对应 export]
info	Display system-wide information	# 显示系统相关信息

```
[root@master testtmp]# docker info
```

```
Containers: 6
```

```
Images: 6
```

```
Storage Driver: devicemapper
```

```
Pool Name: docker-8:2-413022-pool
```

```
Pool Blocksize: 65.54 kB
```

```
Backing Filesystem: extfs
```

Data file: /dev/loop0  
Metadata file: /dev/loop1

...

inspect	Return low-level information on a container	# 查看容器详细信息
---------	---	------------

```
[root@master testtmp]# docker inspect 1f
Error: No such image or container: 1f
[]
[root@master testtmp]# docker inspect 1f4d
[
{
  "Id": "1f4d00854721b28b87698f85e1201492edae6a93665e6ed3592e0f1f1c320201",
  "Created": "2016-11-19T16:55:54.450886779Z",
  "Path": "/bin/bash",
  "Args": [],
  "State": {
    ...
```

kill	Kill a running container	# kill 指定 docker 容器
load	Load an image from a tar archive	# 从一个 tar 包中加载一个镜像
[对应 save]		
login	Register or Login to the docker registry server	
	# 注册或者登陆一个 docker 源服务器	
logout	Log out from a Docker registry server	# 从当前 Docker registry 退出
logs	Fetch the logs of a container	# 输出当前容器日志信息
port	Lookup the public-facing port which is NAT-ed to PRIVATE_PORT	
	# 查看映射端口对应的容器内部源端口	
pause	Pause all processes within a container	# 暂停容器
ps	List containers	# 列出容器列表

```
[root@master Desktop]# docker attach 1f4
You cannot attach to a stopped container, start it first
[root@master Desktop]# docker start 1f4
1f4
[root@master Desktop]# docker attach 1f4

[root@1f4d00854721 /]# ls
```

pull	Pull an image or a repository from the docker registry server	
	# 从 docker 镜像源服务器拉取指定镜像或者库镜像	
push	Push an image or a repository to the docker registry server	
	# 推送指定镜像或者库镜像至 docker 源服务器	
restart	Restart a running container	# 重启运行的容器
rm	Remove one or more containers	# 移除一个或者多个容器
rmi	Remove one or more images	
	# 移除一个或多个镜像[无容器使用该镜像才可删除, 否则需删除相关容器才可继续或 -f 强制删除]	
run	Run a command in a new container	
	# 创建一个新的容器并运行一个命令	
save	Save an image to a tar archive	# 保存一个镜像为一个 tar 包[对应 load]
search	Search for an image on the Docker Hub	# 在 docker hub 中搜索镜像
start	Start a stopped containers	# 启动容器
stop	Stop a running containers	# 停止容器
tag	Tag an image into a repository	# 给源中镜像打标签
top	Lookup the running processes of a container	# 查看容器中运行的进程信息

[root@master Desktop]# docker top 1f4

Error response from daemon: Container 1f4 is not running

[root@master Desktop]# docker start 1f4

1f4

[root@master Desktop]# docker top 1f4

UID	PID	PPID	C
STIME	TTY	TIME	CMD
root	3246	2222	0
12:04	pts/2	00:00:00	/bin/bash

unpause	Unpause a paused container	# 取消暂停容器
version	Show the docker version information	# 查看 docker 版本

本号



wait      Block until a container stops, then print its exit code

# 截取容器停止时的退出状态值

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

## docker option

Usage of docker:

--api-enable-cors=false      Enable CORS headers in the remote API

# 远程 API 中开启 CORS 头

-b, --bridge=""      Attach containers to a pre-existing network bridge

# 桥接网络

use 'none' to disable container networking

--bip=""      Use this CIDR notation address for the network bridge's IP, not compatible with -b

# 和 -b 选项不兼容，具体没有测试过

-d, --daemon=false      Enable daemon mode

# daemon 模式

-D, --debug=false      Enable debug mode

# debug 模式

--dns=[]      Force docker to use specific DNS servers

# 强制 docker 使用指定 dns 服务器

--dns-search=[]      Force Docker to use specific DNS search domains

# 强制 docker 使用指定 dns 搜索域

-e, --exec-driver="native"      Force the docker runtime to use a specific exec driver

# 强制 docker 运行时使用指定执行驱动器

--fixed-cidr=""      IPv4 subnet for fixed IPs (ex: 10.20.0.0/16)

this subnet must be nested in the bridge subnet (which is defined by -b or --bip)

-G, --group="docker"      Group to assign the unix socket specified by -H when running in daemon mode

use "" (the empty string) to disable setting of a group

p

<code>-g, --graph="/var/lib/docker"</code>	Path to use as the root of the docker runtime
# 容器运行的根目录路径	
<code>-H, --host=[]</code>	The socket(s) to bind to in daemon mode
# daemon 模式下 docker 指定绑定方式[tcp or 本地 socket]	
specified using one or more tcp://host:port, unix:///path/to/socket, fd://* or fd://socketfd.	
<code>--icc=true</code>	Enable inter-container communication
# 跨容器通信	
<code>--insecure-registry=[]</code>	Enable insecure communication with specified registries (no certificate verification for HTTPS and enable HTTP fallback) (e.g., localhost:5000 or 10.20.0.0/16)
<code>--ip="0.0.0.0"</code>	Default IP address to use when binding container ports
# 指定监听地址, 默认所有 ip	
<code>--ip-forward=true</code>	Enable net.ipv4.ip_forward
# 开启转发	
<code>--ip-masq=true</code>	Enable IP masquerading for bridge's IP range
<code>--iptables=true</code>	Enable Docker's addition of iptables rules
# 添加对应 iptables 规则	
<code>--mtu=0</code>	Set the containers network MTU
# 设置网络 mtu	
if no value is provided: default to the default route MTU or 1500 if no default route is available	
<code>-p, --pidfile="/var/run/docker.pid"</code>	Path to use for daemon PID file
# 指定 pid 文件位置	
<code>--registry-mirror=[]</code>	Specify a preferred Docker registry mirror
<code>-s, --storage-driver=""</code>	Force the docker runtime to use a specific storage driver
# 强制 docker 运行时使用指定存储驱动	
<code>--selinux-enabled=false</code>	Enable selinux support
# 开启 selinux 支持	
<code>--storage-opt=[]</code>	Set storage driver options
# 设置存储驱动选项	
<code>--tls=false</code>	Use TLS; implied by tls-verify flags
# 开启 tls	
<code>--tlscacert="/root/.docker/ca.pem"</code>	Trust only remotes providing a certificate signed by the CA given here

<code>--tlscert="/root/.docker/cert.pem"</code>	Path to TLS certificate file
# tls 证书文件位置	
<code>--tlskey="/root/.docker/key.pem"</code>	Path to TLS key file
# tls key 文件位置	
<code>--tlsverify=false</code>	Use TLS and verify the remote (daemon: verify client, client: verify daemon) # 使用 tls 并确认远程控制主机
<code>-v, --version=false</code>	Print version information and quit
# 输出 docker 版本信息	

## 4.2 docker search

```
$ sudo docker search --help
```

```
[root@master Desktop]# docker search --help
```

Usage: docker search [OPTIONS] TERM

Search the Docker Hub for images

<code>--automated=false</code>	Only show automated builds
<code>--help=false</code>	Print usage
<code>--no-trunc=false</code>	Don't truncate output
<code>-s, --stars=0</code>	Only displays with at least x stars

```
[root@master Desktop]#
```

```
Usage: docker search TERM

Search the Docker Hub for images # 从 Docker Hub 搜索镜像

--automated=false Only show automated builds

--no-trunc=false Don't truncate output

-s, --stars=0 Only displays with at least xxx stars
```

示例：

```
$ sudo docker search -s 100 ubuntu # 查找 star 数至少为 100 的镜像, 找出只有官方镜像 start 数超过
100, 默认不加 s 选项找出所有相关 ubuntu 镜像 NAME DESCRIPTION STAR
S OFFICIAL AUTOMATED
```

ubuntu      Official Ubuntu base image 425 [OK]

## 4.3 docker info

```
$ sudo docker info
```

```
Containers: 1 # 容器个数 Images: 22 # 镜像个数 Storage Driver: devicemapper # 存储驱动 Pool Name:
docker-8:17-3221225728-pool
```

```
Pool Blocksize: 65.54 kB
```

```
Data file: /data/docker/devicemapper/devicemapper/data
```

```
Metadata file: /data/docker/devicemapper/devicemapper/metadata
```

```
Data Space Used: 1.83 GB
```

```
Data Space Total: 107.4 GB
```

```
Metadata Space Used: 2.191 MB
```

```
Metadata Space Total: 2.147 GB
```

```
Library Version: 1.02.84-RHEL7 (2014-03-26) Execution Driver: native-0.2 # 存储驱动 Kernel Version:
3.10.0-123.el7.x86_64
```

```
Operating System: CentOS Linux 7 (Core)
```

## 4.4 docker pull && docker push

```
$ sudo docker pull --help # pull 拉取镜像 Usage: docker pull [OPTIONS] NAME[:
TAG] Pull an image or a repository from the registry
```

```
-a, --all-tags=false Download all tagged images in the repository $ sudo d
ocker push # push 推送指定镜像 Usage: docker push NAME[:TAG] Push an image or
a repository to the registry
```

示例：

```
$ sudo docker pull ubuntu # 下载官方 ubuntu docker 镜像，默认下载所有 ubuntu 官方
库镜像 $ sudo docker pull ubuntu:14.04 # 下载指定版本 ubuntu 官方镜像
```

```
$ sudo docker push 192.168.0.100:5000/ubuntu # 推送镜像库到私有源[可注册 docker
官方账户，推送到官方自有账户] $ sudo docker push 192.168.0.100:5000/ubuntu:14.04
# 推送指定镜像到私有源
```

## 4.5 docker images

### 列出当前系统镜像

```
$ sudo docker images --help
```

```
Usage: docker images [OPTIONS] [NAME] List images
```

`-a, --all=false` Show all images (by default filter out the intermediate image layers) # `-a` 显示当前系统的所有镜像，包括过渡层镜像，默认 `docker images` 显示最终镜像，不包括过渡层镜像

`-f, --filter=[]` Provide filter values (i.e. 'dangling=true')

`--no-trunc=false` Don't truncate output

`-q, --quiet=false` Only show numeric IDs

`--digests=false` Show digests

### 示例：

```
$ sudo docker images # 显示当前系统镜像，不包括过渡层镜像
```

```
$ sudo docker images -a # 显示当前系统所有镜像，包括过渡层镜像
```

```
$ sudo docker images ubuntu # 显示当前系统 docker ubuntu 库中的所有镜像 REPOSITORY
```

TAG	IMAGE ID	CREATED	VIRTUAL SIZE
ubuntu	12.04	ebe4be4dd427 4 weeks ago	210.6 MB
ubuntu	14.04	e54ca5efa2e9 4 weeks ago	276.5 MB
ubuntu	14.04-ssh	6334d3ac099a 7 weeks ago	383.2 MB

## 4.6 docker rmi

### 删除一个或者多个镜像

```
$ sudo docker rmi --help
```

```
Usage: docker rmi IMAGE [IMAGE...] Remove one or more images
```

`-f, --force=false` Force removal of the image # 强制移除镜像不管是否有容器使用该镜像

`--no-prune=false` Do not delete untagged parents # 不要删除未标记的父镜像

## 4.7 docker run

```
$ sudo docker run --help
```

Usage: docker run [OPTIONS] IMAGE [COMMAND] [ARG...] Run a command in a new container

-a, --attach=[] Attach to stdin, stdout or stderr.

-c, --cpu-shares=0 CPU shares (relative weight) # 设置 cpu 使用权重

--cap-add=[] Add Linux capabilities

--cap-drop=[] Drop Linux capabilities

--cidfile="" Write the container ID to the file # 把容器 id 写入到指定文件

--cpuset="" CPUs in which to allow execution (0-3, 0,1) # cpu 绑定 -d,

--detach=false Detached mode: Run container in the background, print new container id # 后台运行容器

--device=[] Add a host device to the container (e.g. --device=/dev/sdc:/dev/xvdc)

--dns=[] Set custom dns servers # 设置

dns --dns-search=[] Set custom dns search domains # 设置 dns 域搜索

-e, --env=[] Set environment variables # 定义环境变量

--entrypoint="" Overwrite the default entrypoint of the image # ? -

--env-file=[] Read in a line delimited file of ENV variables # 从指定文

件读取变量值

--expose=[] Expose a port from the container without publishing it to your host # 指定对外提供服务端口

-h, --hostname="" Container host name # 设置容器主机名

-i, --interactive=false Keep stdin open even if not attached # 保持标准输出开启即使没有 attached

--link=[] Add link to another container (name:alias) # 添加链接到另外一个容器

--lxc-conf=[] (lxc exec-driver only) Add custom lxc options --lxc-conf="lxc.cgroup.cpuset.cpus = 0,1"

-m, --memory="" Memory limit (format: <number><optional unit>, where unit = b, k, m or g) # 内存限制

--name="" Assign a name to the container # 设置容器名

--net="bridge" Set the Network mode for the container # 设置容器网络模式 'bridge': creates a new network stack for the container on the docker bridge 'none': no networking for this container 'container:<name|id>': reuses another container network stack 'host': use the host net

work stack inside the container. Note: the host mode gives the container full access to local system services such as D-bus and is therefore considered insecure.

**-P, --publish-all=false** Publish all exposed ports to the host interfaces # 自动映射容器对外提供服务的端口 **-p, --publish=[]** Publish a container's port to the host # 指定端口映射 **format:** ip:hostPort:containerPort | ip::containerPort | hostPort:containerPort (use 'docker port' to see the actual mapping) **--privileged=false** Give extended privileges to this container # 提供更多的权限给容器 **--restart=""** Restart policy to apply when a container exits (no, on-failure[:max-retry], always) **--rm=false** Automatically remove the container when it exits (incompatible with -d) # 如果容器退出自动移除和 -d 选项冲突 **--security-opt=[]** Security Options

**--sig-proxy=true** Proxify received signals to the process (even in non-tty mode). SIGCHLD is not proxied.

**-t, --tty=false** Allocate a pseudo-tty # 分配伪终端 **-u, --user=""** Username or UID # 指定运行容器的用户 uid 或者用户名 **-v, --volume=[]** Bind mount a volume (e.g., from the host: -v /host:/container, from docker: -v /container) # 挂载卷 **--volumes-from=[]** Mount volumes from the specified container(s) # 从指定容器挂载卷 **-w, --workdir=""** Working directory inside the container # 指定容器工作目录

示例：

```
$ sudo docker images ubuntu
```

REPOSITORY	TAG	IMAGE ID	CREATED	V
------------	-----	----------	---------	---

ubuntu	14.04	e54ca5efa2e9	4 weeks ago	276.5 MB
--------	-------	--------------	-------------	----------

```
... .. $ sudo docker run -t -i -c 100 -m 512MB -h test1 -d --name="docker_test1" ubuntu /bin/bash # 创建一个 cpu 优先级为 100, 内存限制 512MB, 主机名为 test1, 名为 docker_test1 后台运行 bash 的容器 a424ca613c9f2247cd3ede95adfbaf8d28400cbcb1d5f9b69a7b56f97b2b52e5 $ sudo docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STAT
--------------	-------	---------	---------	------

a424ca613c9f	ubuntu:14.04	/bin/bash	6 seconds ago	Up 5 seconds
--------------	--------------	-----------	---------------	--------------

```
docker_test1 $ sudo docker attach docker_test1
```

```
root@test1:/# pwd /
```

```
root@test1:/# exit exit
```

关于 cpu 优先级:

By default all groups have 1024 shares. A group with 100 shares will

get a ~10% portion of the CPU time -[archlinux cgroups](#)

#### 4.8 docker start|stop|kill... ..

Dockerstart|stop|kill|restart|pause|unpause|rm|commit|inspect|logs

**docker start CONTAINER [CONTAINER...]**# 运行一个或多个停止的容器

**docker stop CONTAINER [CONTAINER...]**# 停掉一个或多个运行的容器  
**-t** 选项可指定超时时间

**docker kill [OPTIONS] CONTAINER [CONTAINER...]**# 默认 kill 发送 SIGKILL 信号  
**-s** 可以指定发送 kill 信号类型

**docker restart [OPTIONS] CONTAINER [CONTAINER...]**# 重启一个或多个运行的容器  
**-t** 选项可指定超时时间

**docker pause CONTAINER**# 暂停一个容器，方便 commit

**docker unpause CONTAINER**# 继续暂停的容器

**docker rm [OPTIONS] CONTAINER [CONTAINER...]**# 移除一个或多个容器

**-f, --force=false** Force removal of running container #强制删除一个正在运行中的容器

**-l, --link=false** Remove the specified link and not the underlying container

**-v, --volumes=false** Remove the volumes associated with the container

**docker commit [OPTIONS] CONTAINER**

**[REPOSITORY[:TAG]]**# 提交指定容器为镜像

**-a, --author=""** Author (e.g., "John Hannibal Smith hannibal@a-team.com")

**-m, --message=""** Commit message



-p, --pause=true Pause container during commit

# 默认 commit 是暂停状态

## `docker inspect CONTAINER|IMAGE`

`[CONTAINER|IMAGE...]`# 查看容器或者镜像的详细信息

## `docker logs CONTAINER`# 输出指定容器日志信息

-f, --follow=false Follow log output

# 类似 tail -f

-t, --timestamps=false Show timestamps

--tail="all" Output the specified number of lines at the end of logs (defaults to all logs)

参考文档 : [Docker Run Reference](#)