Docker镜像文件（images）的存储结构

如果你是刚开始接触 [Docker](http://docker.io/)的话，按照官方文档里的例子，很容易入门并且进行一些简单的工作。但是，如果再深入一些，比如要创建自己的Dockerfile的话，你就可能会有点感到困惑了。在众多初学者比较容易迷惑的问题中，这是一个非常普遍的问题：

*我的Docker镜像（images）保存在哪里呢？****- 每个人***

这个问题也着实让我挠头。更糟糕的是，作为一个新手，你的最终目标是将自己笨手笨脚创建的Docker镜像发布到官方的Docker镜像索引上。

*快来看看我创建的Docker镜像吧，它非常棒 thoward/i\_have\_no\_idea\_what\_im\_doing****.***

但这不是我想要做的。

还有更糟糕的事情，在这之前很长的一段时间里，你都没法删除你发布的镜像，你那些让人羞愧尴尬的学习过程将永久性的贴在那里。

幸运的是，现在删除一个自己发布的镜像文件已经变得非常简单了。

所以，在这里我做一下小小的澄清。如果你满足下面的条件的话，那么请放心，你的镜像文件是不可能被发布为公开镜像的：

* 你还没有在官方的public index注册账号**。**
* 你没有通过 docker login 命令登录。
* 你没有运行 docker push 命令将自己的镜像发布到public index上。

**术语（用语、关键概念）**

Docker所使用的术语，也是其致使用户产生困惑的原因。比如Docker可能会使用overlap，或者含义模糊，或者被错误使用的用语，也可能使用有悖于人们常识认识的用语。

所以，在这里我们先对一些术语做一下解释。

**镜像（Image） vs Dockerfile**

这组概念很少会让人产生疑惑，但是这两者的区别非常重要。Docker在镜像（image）中运行你的代码，而不是Dockerfile。Dockerfile是通过docker build命令来编译镜像的配置文件。

如果你去浏览一下public index的话，你将会看到那里罗列了很多镜像文件。但是，也许有些奇怪的是，你将看不到任何编译出它们的dockerfile。镜像文件就是从Dockerfile编译而得到的不透明资产（*opaque asset* ）。

当你用docker push命令发布自己的镜像的时候，你发布的不是任何你的代码，而是由你的代码编译出来的镜像文件。

**注册表（Registry） vs 索引（Index）**

*注册表（镜像注册表，registry）和索引（镜像索引，index）这两个概念，以及它们的区别也很特殊***。**

一个索引（index）用来管理用户账号，权限，搜索，打标签（tagging），以及其它可以方便的通过Web界面来完成的工作。

一个注册表（registry）则是真正的用来保存、提供镜像文件的概念。它通过索引（index）来完成用户认证步骤。

当你执行docker search命令的时候，Docker会到index里执行查找工作，而不是registry。从实际上来说，Docker会在索引（index）所知道的多个注册表（registry）中进行查找。

而当你执行docker push 或者 docker pull操作的时候，index会判断你是否有权限访问或者修改该镜像文件，如果通过index的认证操作的话，registry则进行实际的镜像存储工作或者将镜像文件发送给请求端。同样，index会知道你想访问的镜像的存储位置（哪个registry上），然后将访问请求转向到相应的registry。

另外，当你在本地环境下，执行类似docker images工作的时候，你所打交道的既不是一个index，也不是一个registry，而是两者都会有一点。

**镜像仓库（Repository ）**

Docker 使用类似于Github或者其他SCM工具的仓库概念。当然，它们也不完全等价。  
下面这三个问题值得我们先好好思考一下：

* 仓库和注册表的区别是什么？
* 仓库和镜像的区别是什么？
* 仓库和 an index username的区别是什么？

实际上，这是一个问题，因为镜像仓库是所有这些概念的合集，而不是单独的任何一个。 此外，当你执行docker images的话你将得到类似如下的输出：

*$ docker images  
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE  
ubuntu              12****.****04               8dbd9e392a96        8 months ago        131****.****3 MB (virtual 131.3 MB)  
ubuntu              latest              8dbd9e392a96        8 months ago        131****.****3 MB (virtual 131.3 MB)  
ubuntu              precise             8dbd9e392a96        8 months ago        131****.****3 MB (virtual 131.3 MB)  
ubuntu              12.10               b750fe79269d        8 months ago        24****.****65 kB (virtual 179.7 MB)  
ubuntu              quantal             b750fe79269d        8 months ago        24****.****65 kB (virtual 179.7 MB)*

上面镜像列表的东西看起来像一个仓库列表？嗯？实际上，那些GUID才是Docker镜像，但你并不和他们进行交互**。**

让我们来仔细看看这个问题。

当你执行docker build或者docker commit的时候，你可以给它指定一个镜像名称。这个名称一般来说都是类似username/image\_name这样的格式，但这不是必须的。实际上你可以给镜像名称指定任何字符，甚至是那些已经公开的众所周知镜像的名称。

但是，等你进行docker push的时候，index将会检查镜像的名字，查看是否有和其匹配的仓库（respsitory）。如果找到了匹配的仓库，将会继续检查你是否有权限操作这个仓库。如果你有权限的话，那么一个新版本的镜像将成功的存储（发布）到这个仓库中。从中我们可以看出，registry会保持一个仓库列表，每个仓库都有自己的名字。而每个仓库都保存的都是通过GUID来标识的镜像文件列表。

这里又出现了标签的概念。你可以对任何一个镜像进行打标签操作，并且在同一个仓库中，为同一个镜像保存不同的版本，不同的版本通过GUID来进行区分。我们可以通过username/image\_name:tag这种方式来通过标签访问不同版本的镜像。

*$ docker images  
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE  
ubuntu              12****.****04               8dbd9e392a96        8 months ago        131****.****3 MB (virtual 131.3 MB)  
ubuntu              latest              8dbd9e392a96        8 months ago        131****.****3 MB (virtual 131.3 MB)  
ubuntu              precise             8dbd9e392a96        8 months ago        131****.****3 MB (virtual 131.3 MB)  
ubuntu              12.10               b750fe79269d        8 months ago        24****.****65 kB (virtual 179.7 MB)  
ubuntu              quantal             b750fe79269d        8 months ago        24****.****65 kB (virtual 179.7 MB)*

我们再来重新看一下docker images的输出结果，应该会有新的认识吧。我们有5个不同版本的ubuntu的镜像，这些镜像通过标签来区分。镜像仓库通过ubuntu这个名字来管理所有这些镜像。也许你会认为ubuntu是一个镜像的名字，但实际上这是一个仓库的名字，它告诉我们它从哪里取得，或者在发布镜像的时候将保存到何处。

此外，仓库名称还有它特殊的命名模式（schema），索引（index）会将仓库名字的第一部分解析为用户名，并且将定位仓库的位置。

那么问题就出现了，假设我们有一个Docker镜像 thoward/scooby\_snacks。

这个镜像的“正式的名字”应该是thoward/scooby\_snacks，尽管实际上我们潜意识里会认为它的名字就是scooby\_snacks （这有点类似Github等服务里的仓库的概念）

实际上，当Docker的文档里说道仓库的时候，有时候指的是包含用户名在内的所有东西，而有时则只是去掉用户名后的部分。

这是因为一些仓库（比如ubuntu）等没有用户名。对用户名做特殊处理非常重要，因为index会用它来做认证操作，所以仓库名字里的用户名部分有它自己独特的意思。

**Docker主机上的本地存储**

到这里我们已经对Docker复杂的远程存储体系做出了说明，以及这里面容易让人混淆的几个概念。但是，docker images命令显示的内容都是本机上存在的资源。

那么本地资源都保存在什么位置呢？我们首先来看看 /var/lib/docker/**这个文件夹下的内容。**

打开这个文件夹下的 repositories 文件，你将会看到类似下面这样的JSON 文件：

*$ sudo cat /var/lib/docker/repositories | python -mjson.tool  
{  
“Repositories”: {  
“ubuntu”: {  
“12****.****04″: “8dbd9e392a964056420e5d58ca5cc376ef18e2de93b5cc90e868a1bbc8318c1c”,  
“12****.****10″: “b750fe79269d2ec9a3c593ef05b4332b1d1a02a62b4accb2c21d589ff2f5f2dc”,  
“latest”: “8dbd9e392a964056420e5d58ca5cc376ef18e2de93b5cc90e868a1bbc8318c1c”,  
“precise”: “8dbd9e392a964056420e5d58ca5cc376ef18e2de93b5cc90e868a1bbc8318c1c”,  
“quantal”: “b750fe79269d2ec9a3c593ef05b4332b1d1a02a62b4accb2c21d589ff2f5f2dc”  
}  
}  
}*

看看，是不是正好和docker images的内容一致呢。

*REPOSITORY TAG IMAGE ID CREATED SIZE  
ubuntu              12****.****04               8dbd9e392a96        8 months ago        131****.****3 MB (virtual 131****.****3 MB)  
ubuntu              latest              8dbd9e392a96        8 months ago        131****.****3 MB (virtual 131****.****3 MB)  
ubuntu              precise             8dbd9e392a96        8 months ago        131****.****3 MB (virtual 131****.****3 MB)  
ubuntu              12.10               b750fe79269d        8 months ago        24****.****65 kB (virtual 179.7 MB)  
ubuntu              quantal             b750fe79269d        8 months ago        24****.****65 kB (virtual 179.7 MB)*

接着，我们再来看看 /var/lib/docker/graph/这个文件夹：

*$ sudo ls -al /var/lib/docker/graph  
total 24  
drwx—— 6 root root 4096 Nov 22 06:52****.*** *drwx—— 5 root root 4096 Dec 13 04:25 ..  
drwxr-xr-x 3 root root 4096 Dec 13 04:26 27cf784147099545  
drwxr-xr-x 3 root root 4096 Nov 22 06:52 8dbd9e392a964056420e5d58ca5cc376ef18e2de93b5cc90e868a1bbc8318c1c  
drwxr-xr-x 3 root root 4096 Nov 22 06:52 b750fe79269d2ec9a3c593ef05b4332b1d1a02a62b4accb2c21d589ff2f5f2dc  
drwx—— 3 root root 4096 Nov 22 06:52 \_tmp*

这些输出结果可能有点晦涩（Not terribly friendly），但是从中我们可以看出Docker是使用repositories JSON文件来记述镜像信息的，此JSON文件包含了仓库名、标签、以及标签对应的镜像ID。  
我们有两个来自ubuntu仓库的镜像，这其中标签为12.04，precise和latest都指向的是同一个镜像，其ID为 8dbd9e392a964056420e5d58ca5cc376ef18e2de93b5cc90e868a1bbc8318c1c (这个ID长度为64位，但是我们可以使用其12位的简短模式，比如8dbd9e392a96)**。**

那么这些以镜像ID命名的文件夹下面又保存了什么东西呢？

*$ sudo ls -al /var/lib/docker/graph/8dbd9e392a964056420e5d58ca5cc376ef18e2de93b5cc90e868a1bbc8318c1c  
total 20  
drwxr-xr-x  3 root root 4096 Nov 22 06:52****.*** *drwx——  6 root root 4096 Nov 22 06:52 ..  
-rw——-  1 root root  437 Nov 22 06:51 json  
drwxr-xr-x 22 root root 4096 Apr 11  2013 layer  
-rw——-  1 root root    9 Nov 22 06:52 layersize*

这个文件夹下的内容如下：

* json -保存着关于这个镜像的元数据
* layersize – 一个整数，表示layer的大小。
* layer/ – 子文件夹，保存着rootfs该容器的镜像

*$ sudo cat /var/lib/docker/graph/8dbd9e392a964056420e5d58ca5cc376ef18e2de93b5cc90e868a1bbc8318c1c/json | python -mjson.tool  
{  
“comment”: “Imported from -”,  
“container\_config”: {  
“AttachStderr”: false,  
“AttachStdin”: false,  
“AttachStdout”: false,  
“Cmd”: null,  
“Env”: null,  
“Hostname”: “”,  
“Image”: “”,  
“Memory”: 0,  
“MemorySwap”: 0,  
“OpenStdin”: false,  
“PortSpecs”: null,  
“StdinOnce”: false,  
“Tty”: false,  
“User”: “”  
},  
“created”: “2013-04-11T14:13:15****.****57812-07:00″,  
“docker\_version”: “0.1****.****4″,  
“id”: “8dbd9e392a964056420e5d58ca5cc376ef18e2de93b5cc90e868a1bbc8318c1c”  
}*

*$ sudo cat /var/lib/docker/graph/8dbd9e392a964056420e5d58ca5cc376ef18e2de93b5cc90e868a1bbc8318c1c/layersize  
131301903*

*$ sudo ls -al /var/lib/docker/graph/8dbd9e392a964056420e5d58ca5cc376ef18e2de93b5cc90e868a1bbc8318c1c/layer  
total 88  
drwxr-xr-x 22 root root 4096 Apr 11  2013****.*** *drwxr-xr-x  3 root root 4096 Nov 22 06:52 ..  
drwxr-xr-x  2 root root 4096 Apr 11  2013 bin  
drwxr-xr-x  2 root root 4096 Apr 19  2012 boot  
drwxr-xr-x  4 root root 4096 Nov 22 06:51 dev  
drwxr-xr-x 41 root root 4096 Nov 22 06:51 etc  
drwxr-xr-x  2 root root 4096 Apr 19  2012 home  
drwxr-xr-x 11 root root 4096 Nov 22 06:51 lib  
drwxr-xr-x  2 root root 4096 Nov 22 06:51 lib64  
drwxr-xr-x  2 root root 4096 Apr 11  2013 media  
drwxr-xr-x  2 root root 4096 Apr 19  2012 mnt  
drwxr-xr-x  2 root root 4096 Apr 11  2013 opt  
drwxr-xr-x  2 root root 4096 Apr 19  2012 proc  
drwx——  2 root root 4096 Nov 22 06:51 root  
drwxr-xr-x  4 root root 4096 Nov 22 06:51 run  
drwxr-xr-x  2 root root 4096 Nov 22 06:51 sbin  
drwxr-xr-x  2 root root 4096 Mar  5  2012 selinux  
drwxr-xr-x  2 root root 4096 Apr 11  2013 srv  
drwxr-xr-x  2 root root 4096 Apr 14  2012 sys  
drwxrwxrwt  2 root root 4096 Apr 11  2013 tmp  
drwxr-xr-x 10 root root 4096 Nov 22 06:51 usr  
drwxr-xr-x 11 root root 4096 Nov 22 06:51 var*

看到这里你应该有些顿悟了吧，这就是我们为什么即使我们不和远程索引或者注册表交互也能通过仓库名来使用一个镜像的原理了。因为一旦你讲镜像从远程下载到本地，Docker可以通过仓库名称来使用它们。当你创建自己的Dockerfile的时候也不例外。

**创建自己的Dockerfiles**

首先我们来看一个最简单的例子。创建如下的Dockerfile：

*FROM ubuntu*

这个配置文件除了引入ubuntu这个镜像作为基础层（base layer）之外，几乎什么都没干。不过这样也足以保证他能正常工作了。

接着，我们来运行 docker build -t scooby\_snacks 命令**。这个命令将会在我们指定的位置（例子里是当前文件夹，即”.”）寻找Dockerfile文件，并基于此进行镜像编译。这里的** scooby\_snacks 是仓库的名字。

*$ docker build -t scooby\_snacks****.*** *Uploading context 64184320 bytes  
Step 1 : FROM ubuntu  
—> 8dbd9e392a96  
Successfully built 8dbd9e392a96*

怎么样，标准输出立显示了出现了*“Uploading context”的字眼，这是把此镜像发布到官网上的意思么？*

让我们来检查一下就知道了：

*$ docker search scooby\_snacks  
NAME      DESCRIPTION   STARS     OFFICIAL   TRUSTED*

看来上面我们创建的镜像还没有被公开发布。

那上面那句话是什么意思呢？

我也不知道，但是我们可以忽略它。也许它是将镜像文件保存到本机的意思吧。

*$ sudo cat /var/lib/docker/repositories | python -mjson.tool  
{  
“Repositories”: {  
“scooby\_snacks”: {  
“latest”: “8dbd9e392a964056420e5d58ca5cc376ef18e2de93b5cc90e868a1bbc8318c1c”  
},  
“ubuntu”: {  
“12****.****04″: “8dbd9e392a964056420e5d58ca5cc376ef18e2de93b5cc90e868a1bbc8318c1c”,  
“12****.****10″: “b750fe79269d2ec9a3c593ef05b4332b1d1a02a62b4accb2c21d589ff2f5f2dc”,  
“latest”: “8dbd9e392a964056420e5d58ca5cc376ef18e2de93b5cc90e868a1bbc8318c1c”,  
“precise”: “8dbd9e392a964056420e5d58ca5cc376ef18e2de93b5cc90e868a1bbc8318c1c”,  
“quantal”: “b750fe79269d2ec9a3c593ef05b4332b1d1a02a62b4accb2c21d589ff2f5f2dc”  
}  
}  
}*

从上面的结果来看，Docker只是将它“uploaded”到了/var/lib/docker**。**

同样，我们也可以通过docker images命令来确认这一结果：

*$ docker images  
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE  
ubuntu              12****.****04               8dbd9e392a96        8 months ago        131****.****3 MB (virtual 131.3 MB)  
ubuntu              latest              8dbd9e392a96        8 months ago        131****.****3 MB (virtual 131.3 MB)  
ubuntu              precise             8dbd9e392a96        8 months ago        131****.****3 MB (virtual 131.3 MB)  
scooby\_snacks       latest              8dbd9e392a96        8 months ago        131****.****3 MB (virtual 131.3 MB)  
ubuntu              12.10               b750fe79269d        8 months ago        24****.****65 kB (virtual 179.7 MB)  
ubuntu              quantal             b750fe79269d        8 months ago        24****.****65 kB (virtual 179.7 MB)*

这就是我们构建镜像文件的结果。Docker很聪明，由于我们是基于ubuntu:latest这个镜像来创建的scooby\_snacks，而且没有做任何定制修改，所以它的镜像ID也是8dbd9e392a96。

下面，我们来试着对基础层做一些修改，然后编译出一个新的层（layer）来。

我们通过修改 Dockerfile 文件增加如下内容来实现：

*FROM ubuntu  
RUN touch scooby\_snacks.txt*

编辑完之后，我们再次运行“docker build -t scooby\_snacks **.**”来构建这个镜像。

*$ docker build -t scooby\_snacks****.*** *Uploading context 64184320 bytes  
Step 1 : FROM ubuntu  
—> 8dbd9e392a96  
Step 2 : RUN touch scooby\_snacks.txt  
—> Running in 86664242766c  
—> 91acef3a5936  
Successfully built 91acef3a5936*

新的构建完成之后，会在/var/lib/docker/graph下面一个新的文件夹

*$ sudo ls -al /var/lib/docker/graph  
total 28  
drwx—— 7 root root 4096 Dec 13 06:27****.*** *drwx—— 5 root root 4096 Dec 13 06:27 ..  
drwxr-xr-x 3 root root 4096 Dec 13 04:26 27cf784147099545  
drwxr-xr-x 3 root root 4096 Nov 22 06:52 8dbd9e392a964056420e5d58ca5cc376ef18e2de93b5cc90e868a1bbc8318c1c  
drwxr-xr-x 3 root root 4096 Dec 13 06:27 91acef3a5936769f763729529e736681e5079dc6ddf6ab0e61c327a93d163df9  
drwxr-xr-x 3 root root 4096 Nov 22 06:52 b750fe79269d2ec9a3c593ef05b4332b1d1a02a62b4accb2c21d589ff2f5f2dc  
drwx—— 3 root root 4096 Dec 13 06:27 \_tmp*

Docker为这个镜像分配了一个新的 ID：

*91acef3a5936769f763729529e736681e5079dc6ddf6ab0e61c327a93d163df9*

这个新的镜像已经被更新到了/var/lib/docker/repositories 并且可以通过docker images**命令来确认。**

*$ sudo cat /var/lib/docker/repositories | python -mjson.tool  
{  
“Repositories”: {  
“scooby\_snacks”: {  
“latest”: “91acef3a5936769f763729529e736681e5079dc6ddf6ab0e61c327a93d163df9″  
},  
“ubuntu”: {  
“12****.****04″: “8dbd9e392a964056420e5d58ca5cc376ef18e2de93b5cc90e868a1bbc8318c1c”,  
“12****.****10″: “b750fe79269d2ec9a3c593ef05b4332b1d1a02a62b4accb2c21d589ff2f5f2dc”,  
“latest”: “8dbd9e392a964056420e5d58ca5cc376ef18e2de93b5cc90e868a1bbc8318c1c”,  
“precise”: “8dbd9e392a964056420e5d58ca5cc376ef18e2de93b5cc90e868a1bbc8318c1c”,  
“quantal”: “b750fe79269d2ec9a3c593ef05b4332b1d1a02a62b4accb2c21d589ff2f5f2dc”  
}  
}  
}  
$ docker images  
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE  
scooby\_snacks       latest              91acef3a5936        5 minutes ago       12****.****29 kB (virtual 131.3 MB)  
ubuntu              12.04               8dbd9e392a96        8 months ago        131****.****3 MB (virtual 131.3 MB)  
ubuntu              latest              8dbd9e392a96        8 months ago        131****.****3 MB (virtual 131.3 MB)  
ubuntu              precise             8dbd9e392a96        8 months ago        131****.****3 MB (virtual 131.3 MB)  
ubuntu              12.10               b750fe79269d        8 months ago        24****.****65 kB (virtual 179.7 MB)  
ubuntu              quantal             b750fe79269d        8 months ago        24****.****65 kB (virtual 179.7 MB)*

让我们来看看文件/var/lib/docker/graph/91acef3a5936769f763729529e736681e5079dc6ddf6ab0e61c327a93d163df9 的内容：

*$ sudo cat /var/lib/docker/graph/91acef3a5936769f763729529e736681e5079dc6ddf6ab0e61c327a93d163df9/json | python -mjson.tool  
{  
“Size”: 0,  
“architecture”: “x86\_64″,  
“config”: {  
“AttachStderr”: false,  
“AttachStdin”: false,  
“AttachStdout”: false,  
“Cmd”: null,  
“CpuShares”: 0,  
“Dns”: null,  
“Domainname”: “”,  
“Entrypoint”: [],  
“Env”: [  
"HOME=/",  
"PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"  
],  
“ExposedPorts”: {},  
“Hostname”: “86664242766c”,  
“Image”: “8dbd9e392a964056420e5d58ca5cc376ef18e2de93b5cc90e868a1bbc8318c1c”,  
“Memory”: 0,  
“MemorySwap”: 0,  
“NetworkDisabled”: false,  
“OpenStdin”: false,  
“PortSpecs”: null,  
“StdinOnce”: false,  
“Tty”: false,  
“User”: “”,  
“Volumes”: {},  
“VolumesFrom”: “”,  
“WorkingDir”: “”  
},  
“container”: “86664242766c5548f8118716e873835c171811176a710e425c1fcf1fa367b505″,  
“container\_config”: {  
“AttachStderr”: false,  
“AttachStdin”: false,  
“AttachStdout”: false,  
“Cmd”: [  
"/bin/sh",  
"-c",  
"touch scooby\_snacks.txt"  
],  
“CpuShares”: 0,  
“Dns”: null,  
“Domainname”: “”,  
“Entrypoint”: [],  
“Env”: [  
"HOME=/",  
"PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"  
],  
“ExposedPorts”: {},  
“Hostname”: “86664242766c”,  
“Image”: “8dbd9e392a964056420e5d58ca5cc376ef18e2de93b5cc90e868a1bbc8318c1c”,  
“Memory”: 0,  
“MemorySwap”: 0,  
“NetworkDisabled”: false,  
“OpenStdin”: false,  
“PortSpecs”: null,  
“StdinOnce”: false,  
“Tty”: false,  
“User”: “”,  
“Volumes”: {},  
“VolumesFrom”: “”,  
“WorkingDir”: “”  
},  
“created”: “2013-12-13T06:27:03****.****234029255Z”,  
“docker\_version”: “0.6.7″,  
“id”: “91acef3a5936769f763729529e736681e5079dc6ddf6ab0e61c327a93d163df9″,  
“parent”: “8dbd9e392a964056420e5d58ca5cc376ef18e2de93b5cc90e868a1bbc8318c1c”  
}*

*$ sudo cat /var/lib/docker/graph/91acef3a5936769f763729529e736681e5079dc6ddf6ab0e61c327a93d163df9/layersize  
12288*

*$ sudo ls -al /var/lib/docker/graph/91acef3a5936769f763729529e736681e5079dc6ddf6ab0e61c327a93d163df9/layer  
total 16  
drwxr-xr-x 4 root root 4096 Dec 13 06:27****.*** *drwxr-xr-x 3 root root 4096 Dec 13 06:27 ..  
-rw-r–r– 1 root root    0 Dec 13 06:27 scooby\_snacks.txt  
-r–r–r– 1 root root    0 Dec 13 06:27****.****wh..wh.aufs  
drwx—— 2 root root 4096 Dec 13 06:27****.****wh..wh.orph  
drwx—— 2 root root 4096 Dec 13 06:27****.****wh..wh.plnk*

我们对Dockerfile小小的修改，却给编译后的结果带来了莫大的影响。另外需要注意的是Docker的镜像文件里只会记录和parent image的差分信息，这也是理解layer概念的一个关键。

**运行新的镜像文件**

下面我们就可以使用我们的新镜像了。这里我们使用bash这个交互控制台来操作。

*$ docker run -i -t scooby\_snacks /bin/bash  
root@1f8602a7d589:/# ls -al  
total 12308  
drwxr-xr-x  30 root root     4096 Dec 13 06:43****.*** *drwxr-xr-x  30 root root     4096 Dec 13 06:43 ..  
-rw——-   1 root root      208 Dec 13 06:43 .dockerenv  
-rwxr-xr-x   1 root root 12516574 Nov 22 02:34 .dockerinit  
drwxr-xr-x   2 root root     4096 Apr 11  2013 bin  
drwxr-xr-x   2 root root     4096 Apr 19  2012 boot  
drwxr-xr-x   6 root root     4096 Nov 22 06:52 dev  
drwxr-xr-x  41 root root     4096 Nov 22 06:52 etc  
drwxr-xr-x   2 root root     4096 Apr 19  2012 home  
drwxr-xr-x  11 root root     4096 Nov 22 06:51 lib  
drwxr-xr-x   2 root root     4096 Nov 22 06:51 lib64  
drwxr-xr-x   2 root root     4096 Apr 11  2013 media  
drwxr-xr-x   2 root root     4096 Apr 19  2012 mnt  
drwxr-xr-x   2 root root     4096 Apr 11  2013 opt  
dr-xr-xr-x 102 root root        0 Dec 13 06:43 proc  
drwx——   2 root root     4096 Nov 22 06:51 root  
drwxr-xr-x   4 root root     4096 Nov 22 06:51 run  
drwxr-xr-x   2 root root     4096 Nov 22 06:51 sbin  
-rw-r–r–   1 root root        0 Dec 13 06:27 scooby\_snacks.txt  
drwxr-xr-x   2 root root     4096 Mar  5  2012 selinux  
drwxr-xr-x   2 root root     4096 Apr 11  2013 srv  
dr-xr-xr-x  13 root root        0 Dec 13 06:43 sys  
drwxrwxrwt   2 root root     4096 Apr 11  2013 tmp  
drwxr-xr-x  10 root root     4096 Nov 22 06:51 usr  
drwxr-xr-x  11 root root     4096 Nov 22 06:51 var  
root@1f8602a7d589:/#*

从上面的结果可以看出，我们在Dockerfile里添加的如下一行：

*RUN touch scooby\_snacks.txt*

在镜像里成功的创建了scooby\_snacks.txt这个文件。

**发布镜像**

到目前为止，我们的所有操作都是基于本地的，没有和外界进行任何交互。这非常重要，因为我们可以精心的准备一个完美的Dockerfile，直到我们完成它为止。现在，我觉得我已经完成了本地镜像文件的制作，已经可以发布它了。

在进行发布之前，你需要确保在官方网站创建了自己的账号（[make an account](https://index.docker.io/account/signup/)），然后通过docker login来登录进去。

*$ docker login  
Username: thoward  
Password:  
Email: thoward37@gmail.com  
Login Succeeded*

我们通过docker push scooby\_snacks命令来发布上面创建的镜像文件。

*$ docker push scooby\_snacks  
2013/12/13 06:49:36 Impossible to push a “root” repository****.****Please rename your repository in <user>/<repo> (ex: thoward/scooby\_snacks)*

从上面的消息可以看出，Docker不允许我们直接将镜像发布到根仓库（root repository）下，我们必须提供类似<user>/<repo>这样的格式。

我们需要重新构建我们的镜像，并指定用户名属性，命令为**docker build -t thoward/scooby\_snacks .（注意不要丢掉最后的点，代表当前文件夹）**

*$ docker build -t thoward/scooby\_snacks****.*** *Uploading context 64184320 bytes  
Step 1 : FROM ubuntu  
—> 8dbd9e392a96  
Step 2 : RUN touch scooby\_snacks.txt  
—> Using cache  
—> 91acef3a5936  
Successfully built 91acef3a5936*

Nice!上面的输出里有”Using cache”字眼，这说明Docker很聪明，知道我们在这次镜像构建中，并未做任何修改（除了名字），所以它不会真的重新再构建一遍的。

然后我们就可以重新发布镜像了。当然，我们需要指定正确的仓库名称：

*$ docker push thoward/scooby\_snacks  
The push refers to a repository [thoward/scooby\_snacks] (len: 1)  
Sending image list  
Pushing repository thoward/scooby\_snacks (1 tags)  
Pushing 8dbd9e392a964056420e5d58ca5cc376ef18e2de93b5cc90e868a1bbc8318c1c  
Image 8dbd9e392a964056420e5d58ca5cc376ef18e2de93b5cc90e868a1bbc8318c1c already pushed, skipping  
Pushing tags for rev [8dbd9e392a964056420e5d58ca5cc376ef18e2de93b5cc90e868a1bbc8318c1c] on {https://registry-1.docker****.****io/v1/repositories/thoward/scooby\_snacks/tags/latest}  
Pushing 91acef3a5936769f763729529e736681e5079dc6ddf6ab0e61c327a93d163df9*

*Pushing tags for rev [91acef3a5936769f763729529e736681e5079dc6ddf6ab0e61c327a93d163df9] on {https://registry-1.docker****.****io/v1/repositories/thoward/scooby\_snacks/tags/latest}*

这样，我们就完成了镜像的发布工作。同时，我们也可以用docker search命令来查看这个镜像。

*$ docker search scooby\_snacks  
NAME                    DESCRIPTION   STARS     OFFICIAL   TRUSTED  
thoward/scooby\_snacks                 0*

镜像文件已经成功发布了，下面我们需要将刚才没有发布成功的镜像删除。

*$ docker images  
REPOSITORY              TAG                 IMAGE ID            CREATED             SIZE  
scooby\_snacks           latest              91acef3a5936        30 minutes ago      12****.****29 kB (virtual 131.3 MB)  
thoward/scooby\_snacks   latest              91acef3a5936        30 minutes ago      12****.****29 kB (virtual 131.3 MB)  
ubuntu                  12.04               8dbd9e392a96        8 months ago        131****.****3 MB (virtual 131.3 MB)  
ubuntu                  latest              8dbd9e392a96        8 months ago        131****.****3 MB (virtual 131.3 MB)  
ubuntu                  precise             8dbd9e392a96        8 months ago        131****.****3 MB (virtual 131.3 MB)  
ubuntu                  12.10               b750fe79269d        8 months ago        24****.****65 kB (virtual 179.7 MB)  
ubuntu                  quantal             b750fe79269d        8 months ago        24****.****65 kB (virtual 179.7 MB)*

*$ docker rmi scooby\_snacks  
Untagged: 91acef3a5936769f763729529e736681e5079dc6ddf6ab0e61c327a93d163df9*

*$ docker images  
REPOSITORY              TAG                 IMAGE ID            CREATED             SIZE  
thoward/scooby\_snacks   latest              91acef3a5936        29 minutes ago      12****.****29 kB (virtual 131.3 MB)  
ubuntu                  12.04               8dbd9e392a96        8 months ago        131****.****3 MB (virtual 131.3 MB)  
ubuntu                  latest              8dbd9e392a96        8 months ago        131****.****3 MB (virtual 131.3 MB)  
ubuntu                  precise             8dbd9e392a96        8 months ago        131****.****3 MB (virtual 131.3 MB)  
ubuntu                  12.10               b750fe79269d        8 months ago        24****.****65 kB (virtual 179.7 MB)  
ubuntu                  quantal             b750fe79269d        8 months ago        24****.****65 kB (virtual 179.7 MB)*

上面的docker rmi命令就是删除镜像的命令。

不过，老实说我们刚才所发布的镜像对其他人来说没有任何意义，我也不想被别人当做菜鸟看待，所以我要删掉它。

*$ docker rmi thoward/scooby\_snacks  
Untagged: 91acef3a5936769f763729529e736681e5079dc6ddf6ab0e61c327a93d163df9  
Deleted: 91acef3a5936769f763729529e736681e5079dc6ddf6ab0e61c327a93d163df9*

*$ docker images  
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE  
ubuntu              12****.****04               8dbd9e392a96        8 months ago        131****.****3 MB (virtual 131.3 MB)  
ubuntu              latest              8dbd9e392a96        8 months ago        131****.****3 MB (virtual 131.3 MB)  
ubuntu              precise             8dbd9e392a96        8 months ago        131****.****3 MB (virtual 131.3 MB)  
ubuntu              12.10               b750fe79269d        8 months ago        24****.****65 kB (virtual 179.7 MB)  
ubuntu              quantal             b750fe79269d        8 months ago        24****.****65 kB (virtual 179.7 MB)*

注意上面docker rmi命令的输出，和之前的命令相比，这次多了一条Deleted: 91acef3a….的内容。这是因为Docker知道现在的系统里没有再对这个ID的镜像的引用了，所以可以删除了，而不只是仅仅删除标签（untagging）。

但是，docker rmi命令并不会Docker Index上删除这个镜像。我们可以测试下：

*$ docker search scooby\_snacks  
NAME                    DESCRIPTION   STARS     OFFICIAL   TRUSTED  
thoward/scooby\_snacks                 0*

不错，它还在那里。我们可以通过docker pull命令来下载它并继续使用。

*$ docker pull thoward/scooby\_snacks  
Pulling repository thoward/scooby\_snacks  
91acef3a5936: Download complete  
8dbd9e392a96: Download complete*

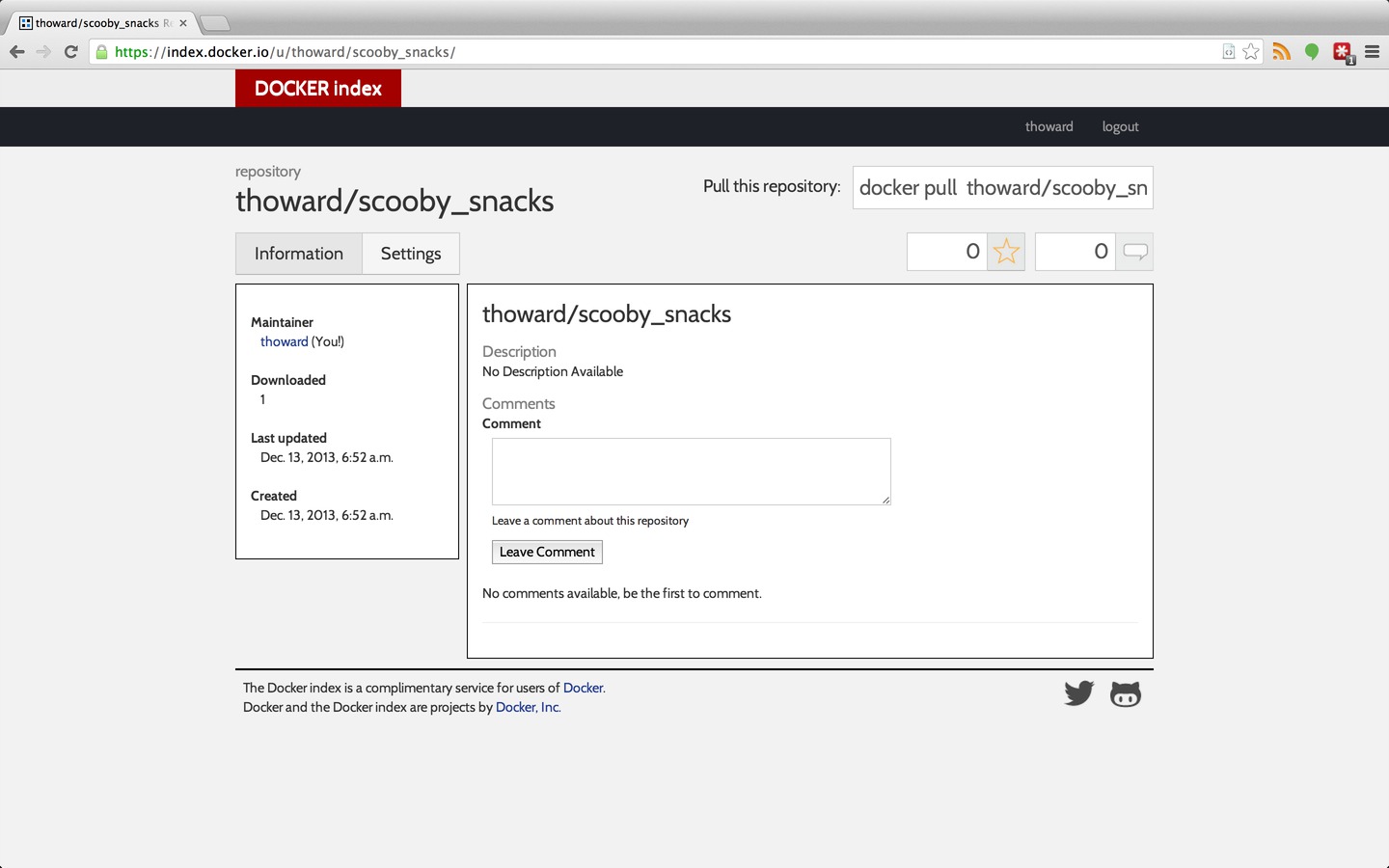
*$ docker run -i -t thoward/scooby\_snacks /bin/bash  
root@90f6546bf3b7:/# ls -al  
total 12308  
drwxr-xr-x  30 root root     4096 Dec 13 07:03****.*** *drwxr-xr-x  30 root root     4096 Dec 13 07:03 ..  
-rw——-   1 root root      208 Dec 13 07:03 .dockerenv  
-rwxr-xr-x   1 root root 12516574 Nov 22 02:34 .dockerinit  
drwxr-xr-x   2 root root     4096 Apr 11  2013 bin  
drwxr-xr-x   2 root root     4096 Apr 19  2012 boot  
drwxr-xr-x   6 root root     4096 Nov 22 06:52 dev  
drwxr-xr-x  41 root root     4096 Nov 22 06:52 etc  
drwxr-xr-x   2 root root     4096 Apr 19  2012 home  
drwxr-xr-x  11 root root     4096 Nov 22 06:51 lib  
drwxr-xr-x   2 root root     4096 Nov 22 06:51 lib64  
drwxr-xr-x   2 root root     4096 Apr 11  2013 media  
drwxr-xr-x   2 root root     4096 Apr 19  2012 mnt  
drwxr-xr-x   2 root root     4096 Apr 11  2013 opt  
dr-xr-xr-x 105 root root        0 Dec 13 07:03 proc  
drwx——   2 root root     4096 Nov 22 06:51 root  
drwxr-xr-x   4 root root     4096 Nov 22 06:51 run  
drwxr-xr-x   2 root root     4096 Nov 22 06:51 sbin  
-rw-r–r–   1 root root        0 Dec 13 06:27 scooby\_snacks.txt  
drwxr-xr-x   2 root root     4096 Mar  5  2012 selinux  
drwxr-xr-x   2 root root     4096 Apr 11  2013 srv  
dr-xr-xr-x  13 root root        0 Dec 13 07:03 sys  
drwxrwxrwt   2 root root     4096 Apr 11  2013 tmp  
drwxr-xr-x  10 root root     4096 Nov 22 06:51 usr  
drwxr-xr-x  11 root root     4096 Nov 22 06:51 var  
root@90f6546bf3b7:/#*

看，我们又重新使用这个镜像。

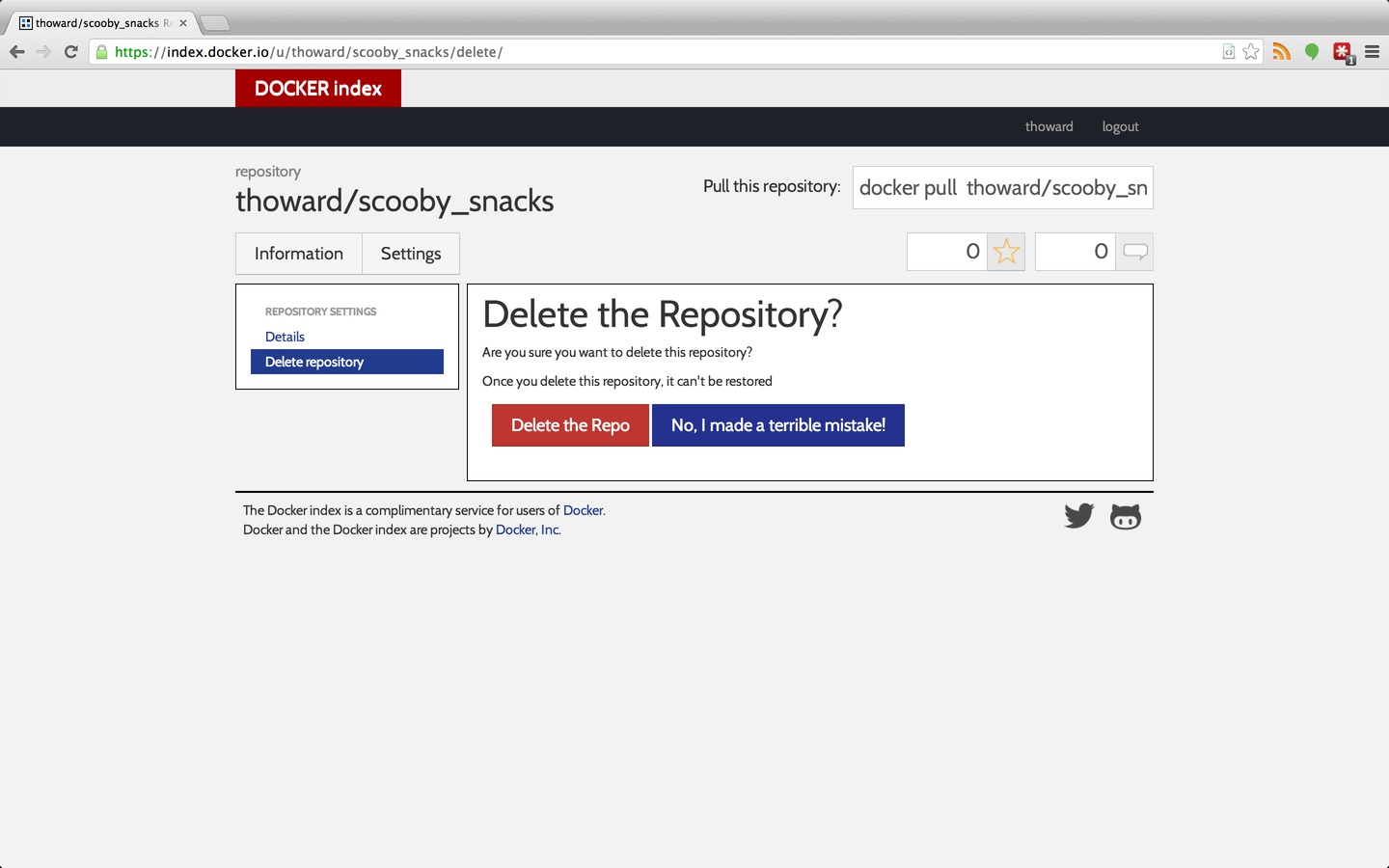
**删除已发布仓库**

不过有点不幸的是，命令行工具没有提供从public index/registry上删除镜像的功能，我们必须从官网的Web页面上来删除它。

首先，登录进Docker的index网站（[login via the web](https://index.docker.io/account/login/) ），然后访问要删除的镜像仓库网址 [https://index.docker**.**io/u/thoward/scooby\_snacks/](https://index.docker.io/u/thoward/scooby_snacks/) 。



依次选择’Settings’、 ‘Delete Repository’ 页，然后点击 ‘Delete Repo’ 按钮。

  
在Web上删除镜像后，我们再回到控制台下，通过docker search scooby\_snacks 命令来确认一下这个镜像确实已经被删除了。

*$ docker search scooby\_snacks  
NAME      DESCRIPTION   STARS     OFFICIAL   TRUSTED*

由于我们在本地重新pull过这个镜像后没有对它进行删除操作，所以我们还会在命令docker images中看到它，因为我们保存着它的本地拷贝。

*$ docker images  
REPOSITORY              TAG                 IMAGE ID            CREATED             SIZE  
thoward/scooby\_snacks   latest              91acef3a5936        46 minutes ago      12****.****29 kB (virtual 131.3 MB)  
ubuntu                  12.04               8dbd9e392a96        8 months ago        131****.****3 MB (virtual 131.3 MB)  
ubuntu                  latest              8dbd9e392a96        8 months ago        131****.****3 MB (virtual 131.3 MB)  
ubuntu                  precise             8dbd9e392a96        8 months ago        131****.****3 MB (virtual 131.3 MB)  
ubuntu                  12.10               b750fe79269d        8 months ago        24****.****65 kB (virtual 179.7 MB)  
ubuntu                  quantal             b750fe79269d        8 months ago        24****.****65 kB (virtual 179.7 MB)*

如果想彻底删除这个镜像，我们需要再次执行docker rmi命令。

*$ docker rmi thoward/scooby\_snacks  
Untagged: 91acef3a5936769f763729529e736681e5079dc6ddf6ab0e61c327a93d163df9  
Deleted: 91acef3a5936769f763729529e736681e5079dc6ddf6ab0e61c327a93d163df9*

辛辛苦苦制作的镜像，就这么被删除了：-（。不过不用可惜，我们可以轻松的通过Dockerfile再次创建同样的镜像出来。

**安全性相关问题**

在我们做诸如上述工作的时候，不得不考虑安全上的问题。

即使你的Docker镜像被从Docker索引服务器上删除了，它也可能还在某人的机器里存在着，而且对此你将无能为力。

而且，正如我们看到的那样，位于本地的镜像并不是真正的“透明的二进制”文件。在Dockerfile文件里设置的所有镜像的内容都保存在JSON文件里，Dockerfile里命令执行所产生的物件（artifacts ）也都被保存在层中，并作为文件系统供我们使用。如果你不小心将密码或者key等敏感信息公开了，那么你将无法挽回这个错误，人们可以像在开源代码里搜索东西一样，很轻松的就发现这些信息。

对自己发布的东西要万分小心，如果一旦你的发布里包含了不该被他人看到的敏感信息，那么首先要尽快下线该镜像，并且更新所有使用了此敏感信息的所有相关系统。  
**总结**

尽管Docker所使用的术语有时候会给用户带来一定的困惑，但是一旦你熟悉了上面的基本流程的话，在构建自己的镜像的时候将会游刃有余，并且清楚的知道自己在什么时候，以何种方式来和他人分享你的成果。  
  
**译者注：**  
**1. 一些术语对照表：**  
**layer ： 层**  
**build： 构建、编译、创建，本文根据语境使用了这三种翻译。**  
**image：镜像**  
**index：索引或镜像索引**  
**respository：仓库或镜像仓库**  
**registry：注册表或者镜像注册表**  
2. 参考资料：  
官方术语index/registry：<http://docs.docker.io/en/latest/api/registry_index_spec/>