日志

docker image是什么，存储在什么位置。

已有 13486 次阅读2015-12-29 17:30 | 个人分类:docker&k8s | docker, image、docker, 镜像、docker镜像存储

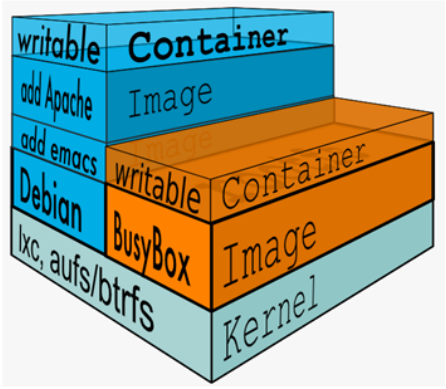
本文回答两个问题：

1. docker image是什么。
2. docker image存储在哪里，以什么形式存储。

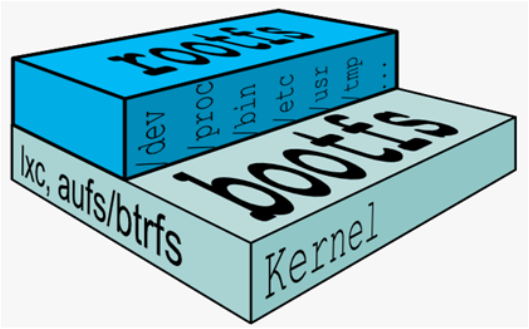
1. docker image是什么

image 里面是一层层文件系统，叫做 Union FS（联合文件系统）。联合文件系统，可以将几层目录挂载到一起，形成一个虚拟文件系统。虚拟文件系统的目录结构就像普通 linux 的目录结构一样，docker 通过这些文件再加上宿主机的内核提供了一个 linux 的虚拟环境。每一层文件系统我们叫做一层 layer，联合文件系统可以对每一层文件系统设置三种权限，只读（readonly）、读写（readwrite）和写出（whiteout-able），但是 docker 镜像中每一层文件系统都是只读的。

构建镜像的时候，从一个最基本的操作系统开始，每个构建的操作都相当于做一层的修改，增加了一层文件系统。一层层往上叠加，上层的修改会覆盖底层该位置的可见性，这也很容易理解，就像上层把底层遮住了一样。当你使用的时候，你只会看到一个完全的整体，你不知道里面有几层，也不清楚每一层所做的修改是什么。结构类似这样：

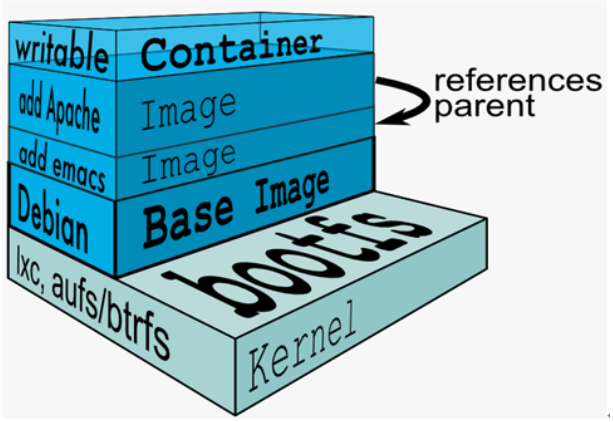


从基本的看起，一个典型的 Linux 文件系统由 bootfs 和 rootfs 两部分组成，bootfs(boot file system) 主要包含 bootloader 和 kernel，bootloader 主要用于引导加载 kernel，当 kernel 被加载到内存中后 bootfs 会被 umount 掉。rootfs (root file system) 包含的就是典型 Linux 系统中的/dev，/proc，/bin，/etc 等标准目录和文件。下图就是 docker image 中最基础的两层结构，不同的 linux 发行版（如 ubuntu 和 CentOS ）在 rootfs 这一层会有所区别，体现发行版本的差异性。



传统的 Linux 加载 bootfs 时会先将 rootfs 设为 read-only，然后在系统自检之后将 rootfs 从 read-only 改为 read-write，然后我们就可以在 rootfs 上进行读写操作了。但 Docker 在 bootfs 自检完毕之后并不会把 rootfs 的 read-only 改为 read-write，而是利用 union mount（UnionFS 的一种挂载机制）将 image 中的其他的 layer 加载到之前的 read-only 的 rootfs 层之上，每一层 layer 都是 rootfs 的结构，并且是read-only 的。所以，我们是无法修改一个已有镜像里面的 lay

的！只有当我们创建一个容器，也就是将 Docker 镜像进行实例化，系统会分配一层空的 read-write 的 rootfs，用于保存我们做的修改。一层 layer 所保存的修改是增量式的，就像 git 一样。



综上，image其实就是一个文件系统，它与宿主机的内核一起为程序提供一个虚拟的linux环境。在启动docker container时，依据image，docker会为container构建出一个虚拟的linux环境。

2. docker image存储在哪里，以什么形式存储。

当运行sudo docker pull ubuntu命令，从Docker HUB上下载一个ubuntu image后，这个image存储在宿主机的什么位置，以什么形式存储的呢。下面我们一步步来揭开这个答案。

Docker目前支持五种镜像层次的存储driver：aufs、device mapper、btrfs、vfs、overlay。下面以aufs做介绍，这个也是我们系统上用的。

下载一个ubuntu image，修改tag为xuzw，删除其他所有镜像。

```
root@ubuntu:~# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
ubuntu              xuzw                ca4d7b1b9a51        Less than a second ago   187.9 MB
root@ubuntu:~#
```

在安装docker时，默认的安装位置是/var/lib/docker。让我们看看这个目录下面都有什么。

```
root@ubuntu:/# cd /var/lib/docker/
root@ubuntu:/var/lib/docker# dir
aufs  containers  graph  init  linkgraph.db  repositories-aufs  tmp  trust  volumes
root@ubuntu:/var/lib/docker# ls -l
total 40
drwxr-xr-x 5 root root 4096 Oct 17 02:47 aufs
drwx------ 3 root root 4096 Oct 18 03:02 containers
drwx------ 7 root root 4096 Oct 17 21:30 graph
drwx------ 2 root root 4096 Oct 17 02:47 init
-rw-r--r-- 1 root root 5120 Oct 29 02:41 linkgraph.db
-rw------- 1 root root  103 Oct 29 03:43 repositories-aufs
drwx------ 2 root root 4096 Oct 18 03:02 tmp
drwx------ 2 root root 4096 Oct 17 02:47 trust
drwx------ 2 root root 4096 Oct 17 02:47 volumes
```

repositories-aufs：记录了镜像名称以及对应的Id的json文件，

```
root@ubuntu:/var/lib/docker#
root@ubuntu:/var/lib/docker# cat repositories-aufs | python -mjson.tool
{
    "Repositories": {
        "ubuntu": {
            "xuzw": "ca4d7b1b9a51f72ff4da652d96943f657b4898889924ac3dae5df958dba0dc4a"
        }
    }
}
```

graph：保存的是下载镜像的元数据，包括json和layersize，其中json文件记录了相应的image id、依赖关系、创建时间和配置信息等。layersize为对应层的大小。进入graph文件会发现下面包含着多个文件夹，其中有一个是以我们下载的镜像命名的文件夹ca4……，进入这个文件夹，可以看到json和layersize文件。

```
root@ubuntu:/var/lib/docker/graph# ls -l
total 20
drwx------ 2 root root 4096 Oct 17 21:30 2332d8973c9393d58c03693bb4d8ec8bd853bafda3b897d48b391a1d0ba9ffb0
drwx------ 2 root root 4096 Oct 17 21:30 a467a7c6794fd7ebd5bd0e2dcb83a656ac8302e549c4a2cc29c524aea5c5623b
drwx------ 2 root root 4096 Oct 17 21:30 ca4d7b1b9a51f72ff4da652d96943f657b4898889924ac3dae5df958dba0dc4a
drwx------ 2 root root 4096 Oct 17 21:30 ea358092da773eff1664fd484edeffb0011f26b4f1dd344ad11b73db57c91d8ae
drwx------ 2 root root 4096 Oct 17 21:30 _tmp
root@ubuntu:/var/lib/docker/graph# cd ca4d7b1b9a51f72ff4da652d96943f657b4898889924ac3dae5df958dba0dc4a/
root@ubuntu:/var/lib/docker/graph/ca4d7b1b9a51f72ff4da652d96943f657b4898889924ac3dae5df958dba0dc4a# ls -l
total 8
-rw------- 1 root root 1347 Oct 17 21:30 json
-rw------- 1 root root    1 Oct 17 21:30 layersize
root@ubuntu:/var/lib/docker/graph/ca4d7b1b9a51f72ff4da652d96943f657b4898889924ac3dae5df958dba0dc4a#
```

json文件内容部分截图如下，layersize显示为0。怎么会为0呢，不用着急，这一层为0，不表示镜像大小就是0。Graph存储镜像时，是分层存储的，graph目录下多出的文件夹其实都对应一个layer。这些layer都与我们的镜像名命名的layer有关联，关系就记录在json文件中。

```
{
    "Size": 0,
    "architecture": "amd64",
    "config": {
        "AttachStderr": false,
        "AttachStdin": false,
        "AttachStdout": false,
        "Cmd": [
            "/bin/bash"
        ],
        "CpuShares": 0,
        "Cpuset": "",
        "Domainname": "",
        "Entrypoint": null,
        "Env": [],
        "ExposedPorts": null,
        "Hostname": "a52c17016130",
        "Image": "a467a7c6794fd7ebd5bd0e2dcb83a656ac8302e549c4a2cc29c524aea5c5623b",
        "Labels": {},
        "MacAddress": "",
        "Memory": 0,
        "MemorySwap": 0,
        "NetworkDisabled": false,
        "OnBuild": null,
        "OpenStdin": false,
        "PortSpecs": null,
        "StdinOnce": false,
        "Tty": false,
        "User": "",
        "Volumes": null,
        "WorkingDir": ""
    },
    "container": "84382e3b0cae0e65ad88f32aee022f08bd2b88ba703b3b52b2f5af6bafdedc62",
    "container_config": {
```


从这个json文件中，可以看到起父镜像或者上一层镜像就是a46……，graph目录下也存储着这一层的信息。再往下看，可以看到层次关系是：ca4--> a46--> ea3- -> 233。

在graph这个目录里并没有找到我们想找到的镜像内容存放地。graph目录下只是一些镜像相关的信息数据。由上一节我们已经知道，image应该包含一个类似linux的文件系统才对。

containers：这个下面记录的是容器相关的信息，每运行一个容器，就在这个目录下面生成一个容器Id对应的子目录，如下图。

```
root@ubuntu:/var/lib/docker# docker ps -al
CONTAINER ID        IMAGE                                          COMMAND               CREATED             STATUS
0bcb2115ab3f        10.62.55.202:5000/library/ubuntu:latest   "/bin/echo dhujjkdfk   11 days ago         Exited (0) 1
root@ubuntu:/var/lib/docker# cd containers/
root@ubuntu:/var/lib/docker/containers# ls -l
total 4
drwx------ 2 root root 4096 Oct 18 03:02 0bcb2115ab3ff7caa2094086c9d91defaf0b67038683a6688e1a046417aa1622
root@ubuntu:/var/lib/docker/containers# cd 0bcb2115ab3ff7caa2094086c9d91defaf0b67038683a6688e1a046417aa1622/
root@ubuntu:/var/lib/docker/containers/0bcb2115ab3ff7caa2094086c9d91defaf0b67038683a6688e1a046417aa1622# ls -al
total 36
drwx------ 2 root root 4096 Oct 18 03:02 .
drwx------ 3 root root 4096 Oct 18 03:02 ..
-rw------- 1 root root   84 Oct 18 03:02 0bcb2115ab3ff7caa2094086c9d91defaf0b67038683a6688e1a046417aa1622-json.log
-rw-r--r-- 1 root root 1901 Oct 18 03:02 config.json
-rw-r--r-- 1 root root  499 Oct 18 03:02 hostconfig.json
-rw-r--r-- 1 root root   13 Oct 18 03:02 hostname
-rw-r--r-- 1 root root  174 Oct 18 03:02 hosts
-rw-r--r-- 1 root root  189 Oct 18 03:02 resolv.conf
-rw-r--r-- 1 root root   71 Oct 18 03:02 resolv.conf.hash
```

init：保存的是docker init相关的信息。

tmp：是一个空目录，具体起什么作用还不清楚。

volumes：与docker的数据券相关，在此不进行扩展。

现在就只剩下aufs这个目录没有看了，进入aufs这个目录。

```
root@ubuntu:~# cd /var/lib/docker/
root@ubuntu:/var/lib/docker# cd aufs/
root@ubuntu:/var/lib/docker/aufs# ls -l
total 12
drwxr-xr-x 8 root root 4096 Oct 18 03:02 diff
drwxr-xr-x 2 root root 4096 Oct 18 03:02 layers
drwxr-xr-x 8 root root 4096 Oct 18 03:02 mnt
root@ubuntu:/var/lib/docker/aufs# cd diff/
root@ubuntu:/var/lib/docker/aufs/diff# ls -l
total 24
drwxr-xr-x  4 root root 4096 Oct 18 03:02 0bcb2115ab3ff7caa2094086c9d91defaf0b67038683a6688e1a046417aa1622
drwxr-xr-x  6 root root 4096 Oct 18 03:02 0bcb2115ab3ff7caa2094086c9d91defaf0b67038683a6688e1a046417aa1622-init
drwxr-xr-x 21 root root 4096 Oct 17 21:29 2332d8973c9393d58c03693bb4d8ec8bd853bafda3b897d48b391a1d0ba9ffb0
drwxr-xr-x  3 root root 4096 Oct 17 21:30 a467a7c6794fd7ebd5bd0e2dcb83a656ac8302e549c4a2cc29c524aea5c5623b
drwxr-xr-x  2 root root 4096 Oct 17 21:30 ca4d7b1b9a51f72ff4da652d96943f657b4898889924ac3dae5df958dba0dc4a
drwxr-xr-x  6 root root 4096 Oct 17 21:30 ea358092da773eff1664fd484edeffb0011f26b4f1dd34ad11b73db57c91d8ae
root@ubuntu:/var/lib/docker/aufs/diff# cd ../layers/
root@ubuntu:/var/lib/docker/aufs/layers# ls -l
total 20
-rw-r--r-- 1 root root 330 Oct 18 03:02 0bcb2115ab3ff7caa2094086c9d91defaf0b67038683a6688e1a046417aa1622
-rw-r--r-- 1 root root 260 Oct 18 03:02 0bcb2115ab3ff7caa2094086c9d91defaf0b67038683a6688e1a046417aa1622-init
-rw-r--r-- 1 root root   0 Oct 17 21:23 2332d8973c9393d58c03693bb4d8ec8bd853bafda3b897d48b391a1d0ba9ffb0
-rw-r--r-- 1 root root 130 Oct 17 21:30 a467a7c6794fd7ebd5bd0e2dcb83a656ac8302e549c4a2cc29c524aea5c5623b
-rw-r--r-- 1 root root 195 Oct 17 21:30 ca4d7b1b9a51f72ff4da652d96943f657b4898889924ac3dae5df958dba0dc4a
-rw-r--r-- 1 root root  65 Oct 17 21:30 ea358092da773eff1664fd484edeffb0011f26b4f1dd34ad11b73db57c91d8ae
root@ubuntu:/var/lib/docker/aufs/layers# cd ../mnt/
root@ubuntu:/var/lib/docker/aufs/mnt# ls -l
total 24
drwxr-xr-x 2 root root 4096 Oct 18 03:02 0bcb2115ab3ff7caa2094086c9d91defaf0b67038683a6688e1a046417aa1622
drwxr-xr-x 2 root root 4096 Oct 18 03:02 0bcb2115ab3ff7caa2094086c9d91defaf0b67038683a6688e1a046417aa1622-init
drwxr-xr-x 2 root root 4096 Oct 17 21:23 2332d8973c9393d58c03693bb4d8ec8bd853bafda3b897d48b391a1d0ba9ffb0
drwxr-xr-x 2 root root 4096 Oct 17 21:30 a467a7c6794fd7ebd5bd0e2dcb83a656ac8302e549c4a2cc29c524aea5c5623b
drwxr-xr-x 2 root root 4096 Oct 17 21:30 ca4d7b1b9a51f72ff4da652d96943f657b4898889924ac3dae5df958dba0dc4a
drwxr-xr-x 2 root root 4096 Oct 17 21:30 ea358092da773eff1664fd484edeffb0011f26b4f1dd34ad11b73db57c91d8ae
```

mnt是aufs的挂载目录，diff是实际数据来源，也就是我们image实际存储的地方，包括只读层和可读写层，所有这些层最终都一起挂载到mmt所在的目录。layers下为每层依赖有关的描述文件。

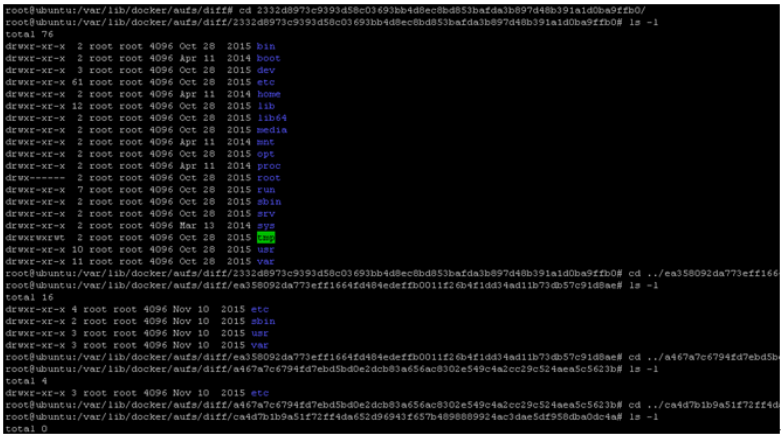
在diff、mnt、layers下面有6个文件或子目录，但是从graph目录下看我们的image应该是4层，为什么会多出来2个呢。

细观察多出来的来个文件或者子目录，会发现其名称和容器Id一致，且有一个包含init。这是为什么呢?

其实，在容器启动之前，mnt和layers都是空目录，diff下面也只有graph目录下我们看到的镜像层对应的4个目录。

在Docker利用image启动一个容器时，会在aufs下新建容器id对应的文件和子目录，同时在镜像的可读层执行新建一个可读写的layer。至于id-init文件或者子目录记录的都是与容器内环境相关的信息，与镜像无关。

既然我们现在已经知道了镜像实际是存储在diff目录下的，那么我们就看看diff目录下各个子目录中的内容。依照镜像的层次关系ca4--> a46--> ea3- -> 233查看。



最终我们看到linux一样的文件系统。已表示image确实是存储在diff目录下的。综上，docker image最终是存储在在/var/lib/docker/aufs/diff中的，同时在graph中有有关进行的记录。在容器启动时，diff下的可读层image和新增的可读写层"容器Id"都将挂载到mmt目录下以容器id命名的子目录下。

参考资料：http://blog.csdn.net/jcjc918/article/details/46500031