国内有 daocloud。

Docker实例教程

<http://www.cnblogs.com/luwenbin/p/5590655.html>

Docker简明教程

<http://kb.cnblogs.com/page/536115/>

**docker常用命令详解**

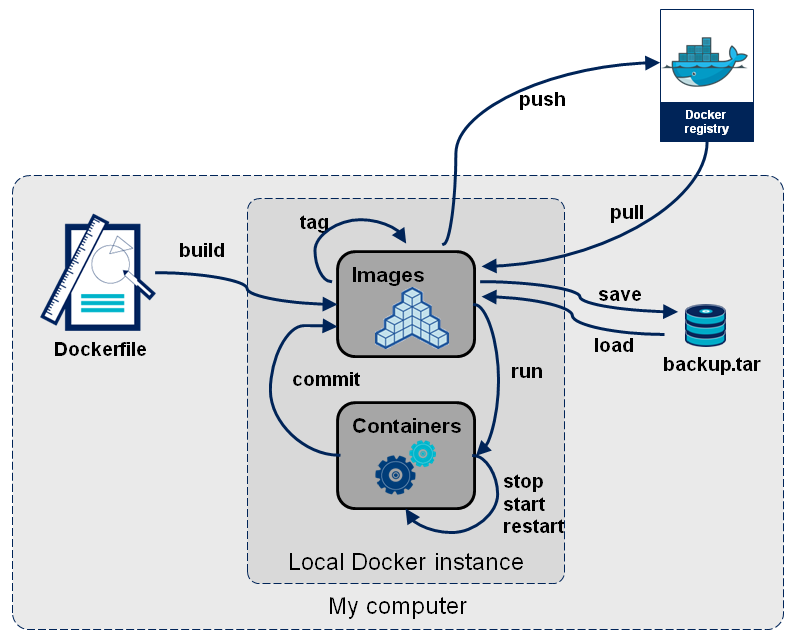
<http://blog.csdn.net/permike/article/details/51879578>

本文只记录Docker命令在大部分情境下的使用，如果想了解每一个选项的细节，请参考官方文档，这里只作为自己以后的备忘记录下来。

根据自己的理解，总的来说分为以下几种：

* 容器生命周期管理 — docker [run|start|stop|restart|kill|rm|pause|unpause]
* 容器操作运维 — docker [ps|inspect|top|attach|events|logs|wait|export|port]
* 容器rootfs命令 — docker [commit|cp|diff]
* 镜像仓库 — docker [login|pull|push|search]
* 本地镜像管理 — docker [images|rmi|tag|build|history|save|import]
* 其他命令 — docker [info|version]

看一个变迁图



**1. 列出机器上的镜像（images）**

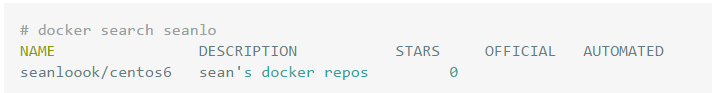


其中我们可以根据REPOSITORY来判断这个镜像是来自哪个服务器，如果没有 / 则表示官方镜像，类似于username/repos\_name表示Github的个人公共库，类似于regsistory.example.com:5000/repos\_name则表示的是私服。

IMAGE ID列其实是缩写，要显示完整则带上 --no-trunc选项

**2. 在docker index中搜索image（search）**

Usage: docker search TERM



搜索的范围是官方镜像和所有个人公共镜像。NAME列的 / 后面是仓库的名字。

**3. 从docker registry server 中下拉image或repository（pull）**

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

# docker pull centos

上面的命令需要注意，在docker v1.2版本以前，会下载官方镜像的centos仓库里的所有镜像，而从v.13开始官方文档里的说明变了：will pull the centos:latest image, its intermediate layers and any aliases of the same id，也就是只会下载tag为latest的镜像（以及同一images id的其他tag）。

也可以明确指定具体的镜像：

# docker pull centos:centos6

当然也可以从某个人的公共仓库（包括自己是私人仓库）拉取，形如docker pull username/repository<:tag\_name> ：

# docker pull seanlook/centos:centos6

如果你没有网络，或者从其他私服获取镜像，形如docker pull registry.domain.com:5000/repos:<tag\_name>

# docker pull dl.dockerpool.com:5000/mongo:latest

**4. 推送一个image或repository到registry（push）**

与上面的pull对应，可以推送到Docker Hub的Public、Private以及私服，但不能推送到Top Level Repository。

# docker push seanlook/mongo

# docker push registry.tp-link.net:5000/mongo:2014-10-27

registry.tp-link.NET也可以写成IP，172.29.88.222。

在repository不存在的情况下，命令行下push上去的会为我们创建为私有库，然而通过浏览器创建的默认为公共库。

**5. 从image启动一个container（run）**

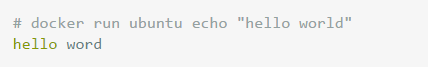
docker run命令首先会从特定的image创之上create一层可写的Container，然后通过start命令来启动它。停止的container可以重新启动并保留原来的修改。run命令启动参数有很多，以下是一些常规使用说明，更多部分请参考 <http://www.cnphp6.com/archives/24899>

当利用 docker run 来创建容器时，Docker 在后台运行的标准操作包括：

* 检查本地是否存在指定的镜像，不存在就从公有仓库下载
* 利用镜像创建并启动一个容器
* 分配一个文件系统，并在只读的镜像层外面挂载一层可读写层
* 从宿主主机配置的网桥接口中桥接一个虚拟接口到容器中去
* 从地址池配置一个 ip 地址给容器
* 执行用户指定的应用程序
* 执行完毕后容器被终止

Usage: docker run [OPTIONS] IMAGE [COMMAND] [ARG...]

**5.1 使用image创建container并执行相应命令，然后停止**



这是最简单的方式，跟在本地直接执行echo 'hello world' 几乎感觉不出任何区别，而实际上它会从本地ubuntu:latest镜像启动到一个容器，并执行打印命令后退出（docker ps -l可查看）。需要注意的是，默认有一个--rm=true参数，即完成操作后停止容器并从文件系统移除。因为Docker的容器实在太轻量级了，很多时候用户都是随时删除和新创建容器。

容器启动后会自动随机生成一个CONTAINER ID，这个ID在后面commit命令后可以变为IMAGE ID

**使用image创建container并进入交互模式, login shell是/bin/bash**



上面的--name参数可以指定启动后的容器名字，如果不指定则docker会帮我们取一个名字。镜像centos:centos6也可以用IMAGE ID (68edf809afe7) 代替），并且会启动一个伪终端，但通过ps或top命令我们却只能看到一两个进程，因为容器的核心是所执行的应用程序，所需要的资源都是应用程序运行所必需的，除此之外，并没有其它的资源，可见Docker对资源的利用率极高。此时使用exit或Ctrl+D退出后，这个容器也就消失了（消失后的容器并没有完全删除？） **经测试，exit或Ctrl+D退出后** **容器会被删除的**。

（那么多个TAG不同而IMAGE ID相同的的镜像究竟会运行以哪一个TAG启动呢

**5.2 运行出一个container放到后台运行**

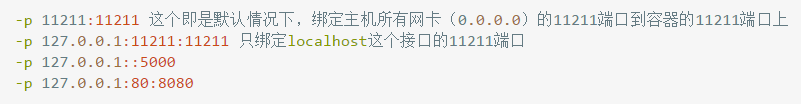
GGP(~O]B@ZNM6TP}4ULG~B0

它将直接把启动的container挂起放在后台运行（这才叫saas），并且会输出一个CONTAINER ID，通过docker ps可以看到这个容器的信息，可在container外面查看它的输出docker logs ae60c4b64205，也可以通过docker attach ae60c4b64205连接到这个正在运行的终端，此时在Ctrl+C退出container就消失了，按ctrl-p ctrl-q可以退出到宿主机，而保持container仍然在运行。

**（是先按 ctrl-p 再按 ctrl-q ，这样就可以退出到宿主机）**  
另外，如果-d启动但后面的命令执行完就结束了，如/bin/bash、echo test，则container做完该做的时候依然会终止。而且-d不能与--rm同时使用  
可以通过这种方式来运行memcached、apache等。

**5.3 映射host到container的端口和目录**

映射主机到容器的端口是很有用的，比如在container中运行memcached，端口为11211，运行容器的host可以连接container的 internel\_ip:11211 访问，如果有从其他主机访问memcached需求那就可以通过-p选项，形如-p <host\_port:contain\_port>，存在以下几种写法：



目录映射其实是“绑定挂载”host的路径到container的目录，这对于内外传送文件比较方便，在搭建私服那一节，为了避免私服container停止以后保存的images不被删除，就要把提交的images保存到挂载的主机目录下。使用比较简单，-v <host\_path:container\_path>，绑定多个目录时再加-v。

-v /tmp/docker:/tmp/docker

另外在两个container之间建立联系可用--link，详见高级部分或官方文档。

下面是一个例子：



在主机的/tmp/docker下建立index.html，就可以通过http://localhost:80/或http://host-ip:80访问了。

**6. 将一个container固化为一个新的image（commit）**

当我们在制作自己的镜像的时候，会在container中安装一些工具、修改配置，如果不做commit保存起来，那么container停止以后再启动，这些更改就消失了。

docker commit <container> [repo:tag]

后面的repo:tag可选

只能提交正在运行的container，即通过docker ps可以看见的容器，

**2. 连接到正在运行中的container（attach）**

I{`BXY{$9@TZ(2EJ8[50KEF

要attach上去的容器必须正在运行，可以同时连接上同一个container来共享屏幕（与screen命令的attach类似）。

官方文档中说attach后可以通过CTRL-C来detach，但实际上经过我的测试，如果container当前在运行bash，CTRL-C自然是当前行的输入，没有退出；如果container当前正在前台运行进程，如输出nginx的access.log日志，CTRL-C不仅会导致退出容器，而且还stop了。这不是我们想要的，detach的意思按理应该是脱离容器终端，但容器依然运行。好在attach是可以带上--sig-proxy=false来确保CTRL-D或CTRL-C不会关闭容器。

我自己测试 CTRL-C是不会退出的（只会中断当前执行的程序），输入exit或 CTRL-D 会退出并销毁容器。如果退出，可以用 **ctrl-p ctrl-q**

也可以使用 docker stop停止容器并删掉。

**4. 删除一个或多个container、image（rm、rmi）**

你可能在使用过程中会build或commit许多镜像，无用的镜像需要删除。但删除这些镜像是有一些条件的：

同一个IMAGE ID可能会有多个TAG（可能还在不同的仓库），首先你要根据这些 image names 来删除标签，当删除最后一个tag的时候就会自动删除镜像；

承上，如果要删除的多个IMAGE NAME在同一个REPOSITORY，可以通过docker rmi <image\_id>来同时删除剩下的TAG；若在不同Repo则还是需要手动逐个删除TAG；

还存在由这个镜像启动的container时（即便已经停止），也无法删除镜像；

删除容器

docker rm <container\_id/contaner\_name>

删除所有停止的容器

docker rm $(docker ps -a -q)

删除镜像

docker rmi <image\_id/image\_name ...>

1.docker pull: 从仓库中拖镜像

2.docker run:运行容器，如果当前要运行的容器对应镜像不存在，会自动拉取。例如：docker run hello-world

3.docker stop：停止容器运行

4.docker start：开始容器运行

5.docker commit：提交容器到镜像

6.docker images: 查看当前有的镜像

7.docker ps：查看当前启动的容器

8.docker build : 创建镜像

9.docker load : 加载镜像