基本命令

2015年11月24日 20:25

镜像部分

docker pull ubuntu:14.04
获得 ubuntu 标签为 14.04的镜像，如果省略14.04则下载 latest 标签的镜像，ubuntu默认为最新的长期支持版本。相当于：docker pull registry.hub.docker.com/ubuntu:14.04 。

docker images
显示本地已有镜像。镜像id表示了唯一的镜像，有些的镜像可能存在多个 TAG ，但是id一致表示实际是一个镜像。

docker commit -m "提交说明" -a "提交用户" 容器id 用户名/仓库名:TAG
提交镜像，只是提交到本地，并没有上传到 docker hub 。

sudo cat ubuntu-14.04-x86_64-minimal.tar.gz |docker import - ubuntu:14.04
从本地导入镜像。源可以是 openvz 模板，地址：http://openvz.org/Download/templates/precreated

docker push ouruser/sinatra
上传镜像到 docker hub

docker save -o ubuntu14.04.tar ubuntu:14.04
导出镜像到文件

docker load --input ubuntu14.04.tar 或 docker load < ubuntu14.04.tar
导入镜像，这将导入镜像及相关元数据（包括标签等信息）。

docker rmi aaaaa/bbbb
删除镜像，注意 docker rm 是删除容器。无法删除使用中的镜像。

docker rmi $(docker images -q -f "dangling=true")
移除未打标签的中间镜像。

容器

docker run ubuntu:14.04 /bin/echo 'hello word'

使用 ubuntu:14.04 镜像启动一个新的容器并执行 /bin/echo 'hello word' 命令。 默认容器停止后还会继续保留，可以通过 start 命令重新启动，可以通过 --rm 参数实现容器停止自动删除的功能。--restart=always选项用来保证Docker守护进程在容器出错或者重启后自动启动容器。

docker run -t -i ubuntu:14.04 /bin/bash
启动容器并执行 /bin/bash 命令。 -t 是分配伪终端并绑定到容器的标准输入上面。-i 则是保持标准输入打开状态。

docker start 容器id
启动已停止的容器，将继续执行创建时指定的命令。通过增加 -i 参数可以进入交互模式。

docker stop 容器id
停止容器。并不会丢失对容器所做的修改！！通过 start 重新启动容器后可以发现之前的修改都保存着！

docker restart 容器id
重启容器。同样不会丢失所做的修改！

docker run -d 镜像名称 命令
后台启动容器，就是不要将标准输出在当前终端下。注意：容器是否会长期运行是和指定的命令有关，而和 -d 参数没有关系。

docker log 容器id
查询容器日志，就是标准输出记录。

docker attach 容器id
附加到指定容器，个人理解就是将容器标准输出连接到当前终端，将当前中断的标准输入连接到容器。注意，多个附加操作是显示一样的内容。实际操作容器时建议通过新命令 docker exec 大体。退出，一定不要用ctrl+c，那样就是让docker容器停止了。要用如下快捷键：先按，ctrl+p 再按，ctrl+q 。使用CTRL-c或CTRL-d退出容器，将向容器发送SIGKILL，导致容器停止。使用CTRL-\退出容器时，将会输出docker客户端的堆栈信息。
docker run -t -i → can be detached with ^P^Q and reattached with docker attach
docker run -i → cannot be detached with ^P^Q; will disrupt stdin
docker run → cannot be detached with ^P^Q; can SIGKILL client; can reattach with docker attach

docker exec 容器id 命令
在指定的容器内执行命令。执行/bin/bash命令需要通过 -ti 选项打开伪终端及标准输入。已经关闭的容器无法执行命令。

nsenter
老的在指定容器执行命令的方案。

docker export 容器id > ubuntu.tar
导出容器快照到文件。

cat ubuntu.tar | sudo docker import -test/ubuntu:v1.0
导入容器快照为镜像。容器快照不保存历史纪录及元数据，体积小；load 导入的镜像储存包含历史纪录及元数据(标签等)，体积比较大。

docker import http://www.aaa.com/bbb.tgz
通过网络导入容器快照。

docker rm 容器id
删除容器。 - v 参数表示同时删除数据卷，否则即使容器被删除数据卷也会永久保留。

docker rm -v $(docker ps -a -q)
删除所有未运行的容器，-v 参数使得会同时删除数据卷。

docker inspect 容器id
审计容器，显示容器的详细信息。

数据管理

sudo docker run -ti -v /user_data ubuntu /bin/bash
启动一个容器，并且挂载了一个 /user_data 的数据卷。数据卷绕过了 UFS ，对数据卷的更新不会影响镜像，默认会一直存在，不会随着容器的删除而删除。实际实现是见一个目录挂载到了容器指定的目录，这里是 /user_data 。未指定数据卷源时源是在主机的 /var/lib/docker/volumes 目录下，docker inspect 容器id 输出的容器详细信息里面有数据卷的详细信息。

```
"Volumes": {
    "/udata":
"/var/lib/docker/volumes/1328a35ef12eb9cfa97210f567a13d70ccf157856c2d79f922a858ef4da1fad8/_data"
},
"VolumesRW": {
    "/udata": true
},
```

可以看到 /user_data 数据卷实际是主机的/var/lib/docker/volumes/1328a35ef12eb9cfa97210f567a13d70ccf157856c2d79f922a858ef4da1fad8/_data目录，对这个目录的修改容器的/user_data 会实时的反应出来。
不指定数据卷源时会自动创建一个新的目录作为数据卷的源，默认每个容器的数据卷不相关。

docker run -ti -v /user_data:~/user_data:ro ubuntu /bin/bash
使用主机的 ~/user_data 目录作为源为容器创建一个路径是 /user_data 的只读数据卷，ro 表示只读。

也可以挂载单个文件作为数据卷，但是vi 等文本编辑工具会造成文件 inode 改变，会使得docker报告错误。

docker rm -v 容器id

删除容器并删除数据卷。资料显示目前 docker 默认删除容器时不会删除容器卷，同时不存在删除无主地数据卷的功能，是个大坑...

docker run -d -v /dbdata --name dbdata training/postgres echo Data-only container for postgres

创建一个名叫 dbdata 的容器，它有一个数据卷 /dbdata ，它的启动命令是 echo ... 表示启动就输出一句话就结束容器（数据卷容器不需要启动即可使用）。

docker run -d --volumes-from dbdata -ti --name db1 ubuntu bash

启动一个新容器，它挂载数据卷容器 dbdata 。也就是它将完全以和 dbdata 容器 相同的容器路径挂载相同的主机目录。也就是本容器 /dbdata 和 容器 dbdata 下的 /dbdata 都是挂载的相同的主机目录。可以通过多个 --volumes-from 参数来从多个数据卷容器挂载数据，一个数据卷容器可以同时被多个容器挂载。

同样删除容器及数据卷容器默认不会实际删除数据，必须在删除最后一个容器时通过 docker rm -v 增加 -v 参数来实际删除数据。

docker run --volumes-from dbdata -v $(pwd):/backup ubuntu tar cvf /backup/backup.tar /dbdata

备份数据卷容器的数据。命令实际执行的操作：容器启动后，使用了 tar 命令来将 dbdata 卷备份为容器中 /backup/backup.tar 文件，也就是主机当前目录下的名为 backup.tar 的文件。

docker run -v /dbdata --name dbdata2 ubuntu /bin/bash

创建一个空数据卷容器

docker run --volumes-from dbdata2 -v $(pwd):/backup busybox tar xvf /backup/backup.tar

将当前目录的backup.tar 内的数据恢复到 数据卷容器 dbdata2 。命令实际执行的操作：创建一个容器，挂载 dbdata2 容器卷中的数据卷，并使用 untar 解压备份文件到挂载的容器卷中。

docker run --volumes-from dbdata2 busybox /bin/ls /dbdata

检查恢复是否成功。

网络功能

docker run -d -P training/webapp python app.py

启动一个容器， 使用 -P 参数时，docker 将会随机的映射一个 49000-49900 的端口到容器开放的网络端口。

$ **sudo docker ps -l**
]CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
532192b4328c training/webapp "python app.py" About a minute ago Up About a minute

```
0.0.0.0:32768->5000/tcp   suspicious_lovelace
```
可以参看映射的端口号。

```
$ sudo docker port 5321
5000/tcp -> 0.0.0.0:32768
```
可以查看指定容器的端口映射。

docker run -d -p 5000:5000 training/webapp python app.py

小写的 p 映射制定的端口。可选的格式是：ip:hostPort:containerPort | ip::containerPort | hostPort:containerPort 。

docker run -d -p 5000:5000/udp training/webapp python app.py

增加 /udp 标记表示映射 udp 端口。

```
$ sudo iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination
DOCKER     all  --  anywhere             anywhere
ACCEPT     all  --  anywhere             anywhere             ctstate RELATED,ESTABLISHED
ACCEPT     all  --  anywhere             anywhere
ACCEPT     all  --  anywhere             anywhere

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination

Chain DOCKER (1 references)
target     prot opt source               destination
ACCEPT     tcp  --  anywhere             172.17.0.24          tcp dpt:5000
```
可以看到端口映射是通过 iptables 实现的。

docker run -d --name db -e AAA=A1 -e bbb=b1 training/postgres
docker run -d -P --name web --link db:db training/webapp python app.py

建立一个名为 db 的容器，然后创建一个名为 web 的容器，并连接到 db 容器。注意：即使不执行连接操作默认所有容器也都在同一个虚拟局域网里面，是可以互相通信的，只不过不知道 docker为容器分配的ip地址而已。可以通过 docker 服务的 --icc=false 参数禁止默认容器之间的互联，同时 --iptables=true 允许 docker 修改 iptables 时，link 操作会自动添加 iptables 规则允许相互访问开放的端口。

```
$docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS
NAMES
3113cef6d6b0        training/webapp     "python app.py"     2 minutes ago       Up About a minute
0.0.0.0:32770->5000/tcp   web
b2fbf68028c3        training/postgres   "su postgres -c '/us   4 minutes ago       Up 4 minutes        5432/tcp
db
```
文档上面说 ps 命令在NAMES字段会显示连接关系，但是实测没有...

root@vps11:~# **docker exec -ti web bash**
root@5a48dc56a38e:/opt/webapp# env
HOSTNAME=5a48dc56a38e
DB_NAME=/web/db
DB_PORT_5432_TCP_ADDR=172.17.0.5
DB_PORT=tcp://172.17.0.5:5432
DB_PORT_5432_TCP=tcp://172.17.0.5:5432
LS_COLORS=
DB_ENV_bbb=b1
DB_ENV_AAA=A1
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
PWD=/opt/webapp
DB_PORT_5432_TCP_PORT=5432
SHLVL=1
HOME=/root
DB_PORT_5432_TCP_PROTO=tcp
LESSOPEN=| /usr/bin/lesspipe %s
DB_ENV_PG_VERSION=9.3
LESSCLOSE=/usr/bin/lesspipe %s %s
_=/usr/bin/env
root@5a48dc56a38e:/opt/webapp# cat /etc/hosts
172.17.0.6 5a48dc56a38e
127.0.0.1 localhost
::1 localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
172.17.0.5 db 2f115673d7cd

可以看到 docker 通过环境变量及hosts文件为web容器公开了连接信息(并没有为 db 容器公开 web 的信息)，DB_的前缀是连接别名。同时 link 还会为web容器公开了db容器 -e 设置的环境变量(环境变量前面增加大写的被连接容器名字，小写的环境变量也会同样小写转发过去。)

inspect web 能看到 link 了 db，但是看不到 db 的环境变量，只能看到 db -e 设置的自身的环境变量。

Dockerfile

```
# This dockerfile uses the ubuntu image
# VERSION 2 - EDITION 1
# Author: docker_user
# Command format: Instruction [arguments / command] ..

# 基础镜像
FROM ubuntu

# Maintainer: 维护者信息
MAINTAINER docker_user docker_user@email.com

# Commands to update the image
```

```
RUN echo "deb http://archive.ubuntu.com/ubuntu/ raring main universe" >> /etc/apt/sources.list
RUN apt-get update && apt-get install -y nginx
RUN echo "\ndaemon off;" >> /etc/nginx/nginx.conf

# 启动容器时运行的命令，如果指定多个，只执行最后一个。如果 run 时指定了命令会覆盖当前设置。
CMD /usr/sbin/nginx
```

FROM

格式为 `FROM <image>`或`FROM <image>:<tag>`。

第一条指令必须为 `FROM` 指令。并且，如果在同一个Dockerfile中创建多个镜像时，可以使用多个 `FROM` 指令（每个镜像一次）。

MAINTAINER

格式为 `MAINTAINER <name>`，指定维护者信息。

RUN

格式为 `RUN <command>` 或 `RUN ["executable", "param1", "param2"]`。

前者将在 shell 终端中运行命令，即 `/bin/sh -c`；后者则使用 `exec` 执行。指定使用其它终端可以通过第二种方式实现，例如 `RUN ["/bin/bash", "-c", "echo hello"]`。

每条 `RUN` 指令将在当前镜像基础上执行指定命令，并提交为新的镜像。当命令较长时可以使用`\` 来换行。

CMD

支持三种格式

- `CMD ["executable","param1","param2"]` 使用 `exec` 执行，推荐方式；
- `CMD command param1 param2` 在 `/bin/sh` 中执行，提供给需要交互的应用；
- `CMD ["param1","param2"]` 提供给 `ENTRYPOINT` 的默认参数；

指定启动容器时执行的命令，每个 Dockerfile 只能有一条 `CMD` 命令。如果指定了多条命令，只有最后一条会被执行。

如果用户启动容器时候指定了运行的命令，则会覆盖掉 `CMD` 指定的命令。

EXPOSE

格式为 `EXPOSE <port> [<port>...]`。

告诉 Docker 服务端容器暴露的端口号，供互联系统使用。在启动容器时需要通过 -P，

Docker 主机会自动分配一个端口转发到指定的端口。

ENV

格式为 `ENV <key> <value>`。 指定一个环境变量，会被后续 `RUN` 指令使用，并在容器运行时保持。

例如

```
ENV PG_MAJOR 9.3
ENV PG_VERSION 9.3.4
RUN curl -SL http://example.com/postgres-$PG_VERSION.tar.xz | tar -xJC
/usr/src/postgress && …
ENV PATH /usr/local/postgres-$PG_MAJOR/bin:$PATH
```

ADD

格式为 `ADD <src> <dest>`。

该命令将复制指定的 `<src>` 到容器中的 `<dest>`。 其中 `<src>` 可以是Dockerfile所在目录的一个相对路径；也可以是一个 URL；还可以是一个 tar 文件（自动解压为目录）。

COPY

格式为 `COPY <src> <dest>`。

复制本地主机的 `<src>`（为 Dockerfile 所在目录的相对路径）到容器中的 `<dest>`。

当使用本地目录为源目录时，推荐使用 `COPY`。

ENTRYPOINT

两种格式：

- `ENTRYPOINT ["executable", "param1", "param2"]`
- `ENTRYPOINT command param1 param2`（shell中执行）。

配置容器启动后执行的命令，并且不可被 `docker run` 提供的参数覆盖。

每个 Dockerfile 中只能有一个 `ENTRYPOINT`，当指定多个时，只有最后一个起效。

VOLUME

格式为 `VOLUME ["/data"]`。

创建一个可以从本地主机或其他容器挂载的挂载点，一般用来存放数据库和需要保持的数据等。相当于 docker -v 参数，为了可移植性考虑，不允许指定主机路径，只能指定容器内的路径。

USER

格式为 `USER daemon`。

指定运行容器时的用户名或 UID，后续的 `RUN` 也会使用指定用户。

当服务不需要管理员权限时，可以通过该命令指定运行用户。并且可以在之前创建所需要的用户，例如：`RUN groupadd -r postgres && useradd -r -g postgres postgres`。要临时获取管理员权限可以使用 `gosu`，而不推荐 `sudo`。

WORKDIR

格式为 `WORKDIR /path/to/workdir`。

为后续的 `RUN`、`CMD`、`ENTRYPOINT` 指令配置工作目录。

可以使用多个 `WORKDIR` 指令，后续命令如果参数是相对路径，则会基于之前命令指定的路径。例如

```
WORKDIR /a
WORKDIR b
WORKDIR c
RUN pwd
```

则最终路径为 `/a/b/c`。

ONBUILD

格式为 `ONBUILD [INSTRUCTION]`。

配置当所创建的镜像作为其它新创建镜像的基础镜像时，所执行的操作指令。

例如，Dockerfile 使用如下的内容创建了镜像 `image-A`。

```
[...]
ONBUILD ADD . /app/src
ONBUILD RUN /usr/local/bin/python-build --dir /app/src
[...]
```

如果基于 image-A 创建新的镜像时，新的Dockerfile中使用 `FROM image-A`指定基础镜像时，会自动执行 `ONBUILD` 指令内容，等价于在后面添加了两条指令。

```
FROM image-A
#Automatically run the following
ADD . /app/src
RUN /usr/local/bin/python-build --dir /app/src
```

使用 `ONBUILD` 指令的镜像，推荐在标签中注明，例如 `ruby:1.9-onbuild`。

来自 <http://dockerpool.com/static/books/docker_practice/dockerfile/instructions.html>

创建镜像

编写完成 Dockerfile 之后，可以通过 docker build 命令来创建镜像。

基本的格式为 docker build [选项] 路径，该命令将读取指定路径下（包括子目录）的 Dockerfile，并将该路径下所有内容发送给 Docker 服务端，由服务端来创建镜像。因此一般建议放置 Dockerfile 的目录为空目录。也可以通过 .dockerignore 文件（每一行添加一条匹配模式）来让 Docker 忽略路径下的目录和文件。

要指定镜像的标签信息，可以通过 -t 选项，例如

$ sudo docker build -t myrepo/myapp /tmp/test1/