



()



小而精的 Docker 项目，30 分钟快速入门 Docker 容器

[\(/gitchat/author/5c692e83ff00d04d11dd97ba\)](#)**奔跑的小米 (/gitchat/a...**

从事大数据及数据挖掘工作，就职于互联网企业，精通大数据生态圈，参与编写《Spark内核机制解析及性能调优》、《Scala语言基础与开发实战》热门书籍，即将出版《学好Pytorch成为数据科学家》深度学习书籍，51CTO金牌讲师。微信公众号：三角兽

[查看本场Chat](#)[\(/gitchat/activity/5d97fb048d2e4f018cfdd00a\)](#)

前言

1. Docker 安装

2. 验证 Docker 安装是否成功

3. 获取 Docker 镜像

- 搜索镜像

- 获取镜像

- 查看本地镜像

4. 创建自己的镜像

- 使用容器的快照制作镜像

- 使用 Dockerfile 创建镜像

5. 保存/载入/删除镜像

6. Docker 容器的创建、启动、和停止

- 创建容器

- 启动容器

- 终止容器

7. 进入守护状态运行的容器

8. 导入/到处/删除容器

- 导出容器为快照文件

- 从文件导入为 Docker 镜像

- 删除镜像

9. 创建 Docker 私有仓库

10. Docker 容器绑定外部 IP 和端口

11. 容器互联

12. 一个完整的小例子



前言

为什么要使用 Docker?

- Docker 容器的启动在秒级
- Docker 对系统资源利用率高，一台主机上可以同时运行数千个 Docker 容器。
- Docker 基本不消耗系统资源，使得运行在 Docker 里面的应用的性能很高。

相比于传统的虚拟化技术，Docker 有哪些优势?

- 更快速的支付和部署：开发者可以使用一个标准的镜像来构建一套开发容器，开发完成之后，运维人员可以直接使用这个容器来部署代码。
- 更高效的虚拟化：Docker 容器的运行不需要额外的 Hypervisor 支持，它是内核级的虚拟化，因此可以实现更高的性能和效。
- 更轻松的迁移和扩展：Docker 容器几乎可以在任意的平台上运行，包括物理机、虚拟机、公有云、私有云、个人电脑、服务器等。
- 更简单的管理：使用 Docker，只需要小小的修改，就可以替代以往大量的更新工作。所有的修改都以增量的方式被分发和更新，从而实现自动化并且高效的管理。

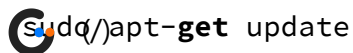
接下来，我们开始 Docker 的学习之旅。

1. Docker 安装

由于 APT 官方库里的 Docker 版本可能比较旧，所以先卸载可能存在的旧版本：

```
sudo apt-get remove docker docker-engine docker-ce docker.io
```

更新 APT 包索引：



安装以下包以使apt可以通过HTTPS使用存储库 (repository)

```
sudo apt-get install -y apt-transport-https ca-certificates curl software-properties-common
```

添加 Docker 官方的 GPG 密钥:

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

使用下面的命令来设置 stable 存储库:

```
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu stable $(lsb_release -cs)"
```

再更新一下 APT 包索引:

```
sudo apt-get update
```

安装最新版本的 Docker CE:

```
sudo apt-get install -y docker-ce
```

2. 验证 Docker 安装是否成功

查看 Docker 服务是否启动:



```
systemctl status docker
```

```
root@ubuntu:/home/hadoop/workspace/compose/composeapp# systemctl st
docker.service - Docker Application Container Engine
Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor
Active: active (running) since Thu 2019-10-03 12:25:14 PDT; 2 days
Docs: https://docs.docker.com
Main PID: 1436 (dockerd)
Tasks: 27
CGroup: /system.slice/docker.service
├─ 1436 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/cor
└─ 42652 /usr/bin/docker-proxy -proto tcp -host-ip 0.0.0.0 -host-pc
```



若未启动, 则启动 Docker 服务:

```
systemctl start docker
```

运行经典的 Hello world:

```
docker run hello-world
```

```
root@ubuntu:/home/hadoop/workspace/compose/composeapp# docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
1b930d010525: Pull complete
Digest: sha256:b8ba256769a0ac28dd126d584e0a2011cd2877f3f76e093a7ae560f2a5301c00
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.
```

3. 获取 Docker 镜像

镜像是一个模板, 运行镜像便可得到 Docker 容器, 可以从镜像服务器上获取做好的镜像, 使用 `docker pull` 命令来获取需要的镜像。Docker Hub 上已经有很多构建好的镜像, 我们可以直接使用, 在制作镜像之前, 有必要搜索一下看是否已经有人共享了相关的镜像, 减少制作成本。通过 `docker search` 实现搜索, `docker pull` 拉取镜像。

搜索镜像



docker search ubuntu

```
root@ubuntu:/home/hadoop/workspace/compose/composeapp# docker search ubuntu
NAME                DESCRIPTION                STARS     OFFICIAL    AUTOMATED
ubuntu              Ubuntu is a Debian-based Linux operating sys... 10017     [OK]
dorowu/ubuntu-desktop-lxde-vnc  Docker image to provide HTML5 VNC interface ... 349       [OK]
rastaseep/ubuntu-ssh  Dockerized SSH service, built on top of offi... 231       [OK]
consol/ubuntu-xfce-vnc  Ubuntu container with "headless" VNC session... 187       [OK]
ubuntu-upstart        Upstart is an event-based replacement for th... 99        [OK]
ansible/ubuntu14.04-ansible  Ubuntu 14.04 LTS with ansible 98         [OK]
neurodebian           NeuroDebian provides neuroscience research s... 59        [OK]
landinternet/ubuntu-16-nginx-php-phpmysql-5  ubuntu-16-nginx-php-phpmysql-5 50         [OK]
ubuntu-debootstrap    debootstrap --variant=minbase --components=m... 40        [OK]
nuagebec/ubuntu       Simple always updated Ubuntu docker images w... 24         [OK]
```

获取镜像

docker pull ubuntu

```
root@ubuntu:/home/hadoop/workspace/compose/composeapp# docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
5667fdb72017: Downloading [=====] 24.71MB/26.68MB
d83811f270d5: Download complete
ee671aafb583: Download complete
7fc152dfb3a6: Download complete
```

查看本地镜像

docker images

```
root@ubuntu:/home/hadoop/workspace/compose/composeapp# docker images
REPOSITORY          TAG             IMAGE ID        CREATED         SIZE
composeapp          latest         0d049ec3df56   2 hours ago    891MB
redis               latest         01a52b3b5cd1   10 days ago    98.2MB
ubuntu              latest         2ca708c1c9cc   2 weeks ago    64.2MB
python              2.7            b707c2244b7d   3 weeks ago    886MB
ubuntu              18.04          a2a15febcdcf3   7 weeks ago    64.2MB
hello-world         latest         fce289e99eb9   9 months ago    1.84kB
```

如果 Docker Hub 中没有自己中意的镜像，怎么办呢？当然就是自己制作镜像了，接下来介绍制作镜像的两种方式。

4. 创建自己的镜像

常见的创建镜像的方式有两种。

使用容器的快照制作镜像



```
# 下载镜像
docker pull training/sinatra
# 启动镜像, 进入bash命令行
sudo docker run -i -t training/sinatra /bin/bash
```

在容器中添加两个应用 gem 和 json:

```
gem install json
```

添加完成后, 使用 exit 退出容器, 我们在容器中添加了新的应用, 容器被改变。使用 docker commit 提交更改后的副本。

```
docker commit -m 'add json gem' -a 'Docker Container' 7b789b19757d my/
```



提交后, 再次使用 docker images 查看本地镜像, 发现多了一个名字为 my/sinatra:v2 的镜像。

```
[root@so4714-instance-qepkuo6yyype5 ~]# docker commit -m 'add json gem' -a 'Docker Container' 7b789b19757d my/sinatra:v2
sha256:6fc131c6ae56493a1edfad93bbe356c730ff63c5486131660f6c682490ace16b
[root@so4714-instance-qepkuo6yyype5 ~]# docker images
REPOSITORY          TAG          IMAGE ID          CREATED          SIZE
my/sinatra          v2          6fc131c6ae56     6 seconds ago   453 MB
mysql               latest       5fac85ee2c68     2 days ago     408 MB
cluster             latest       d8e86fa4fdab     4 weeks ago    4.18 GB
centos              latest       196e0ce0c9fb     4 weeks ago    197 MB
nkxujun/mysql_eml   latest       dd923dce8161     3 months ago   406 MB
nkxujun/ubuntu_eml latest       ad6c47604a44     6 months ago   265 MB
training/sinatra    latest       49d952a36c58     3 years ago    447 MB
[root@so4714-instance-qepkuo6yyype5 ~]#
```

- -m 参数指定提交的说明, 和 Git 中的 git commit -m 的参数是一样的;
- -a 指定更新的用户信息, 后面是用来创建镜像的容器 ID, 最后是创建镜像的仓库名和 tag 信息。创建成功后, 命令会返回这个镜像的 ID 信息。

接下来, 就可以使用我们自己创建的镜像来启动容器了。

```
docker run -i -t my/sinatra:v2
```

使用 Dockerfile 创建镜像

使用 `docker commit` 虽然很容易扩展镜像, 但不便于团队分享, 我们可以使用 `docker build` 来创建一个新的镜像, 为此首先要创建一个 Dockerfile 文件, 这个文件中包含如何创建镜像的指令, 下面是一个简单的例子:



```
FROM centos
MAINTAINER REGAN 626692024@qq.com
RUN yum -qqy install python
```

其中 FROM 告诉 Docker 使用哪个镜像作为基础, 接着是维护者的信息, RUN 开头的指令会在创建中运行, 例如安装一些软件包, 这里使用 yum 安装 python。注意使用 yum 需要制定参数 -qqy, 不然呢可能会报错的。然后使用 `docker build` 构建镜像:

```
docker build -t='my/python:v1' .
```

注意后面的 `.` 不要弄丢了。它表示在当前目录下寻找 Dockerfile 文件。

```
[root@so4714-instance-qepkuo6yyype5 sinatra]# docker build -t='my/python:v1' .
Sending build context to Docker daemon 2.56 kB
Step 1/3 : FROM centos
--> 196e0ce0c9fb
Step 2/3 : MAINTAINER REGAN 626692024@qq.com
--> Using cache
--> 45b961a78f6b
Step 3/3 : RUN yum -qqy install python
--> Running in 29a75030e53d
Loaded plugins: fastestmirror, ovl
Ignored option -q, -v, -d or -e (probably due to merging: -yq != -y -q)
Determining fastest mirrors
* base: mirrors.sohu.com
* extras: mirrors.sohu.com
* updates: mirrors.sohu.com
Package python-2.7.5-58.el7.x86_64 already installed and latest version
Nothing to do
--> 3ca51ddb3afe
Removing intermediate container 29a75030e53d
Successfully built 3ca51ddb3afe
```

其中 -t 标记来添加 tag, 指定新的镜像的用户信息。 `.` 是 Dockerfile 所在的路径 (当前目录), 也可以替换为一个具体的 Dockerfile 的路径。

build 构建过程它要做的第一件事情就是上传这个 Dockerfile 文件到 Docker Engine, 因为所有的操作都要依据 Dockerfile 来进行。Dockerfile 中的指令被一条一条地执行。每一步都创建了一个新的容器, 在容器中执行指令并提交修改 (就跟之前介绍过的 `docker commit` 一样)。

当所有的指令都执行完毕之后, 返回了最终的镜像 id。所有的中间步骤所产生的容器都被删除和清理了。需要注意的是一个镜像是不能操作超过 127 层的, 否则会报错!

当然 Dockerfile 中还可以输入其他的命令，例如可以使用 ADD 命令复制本地文件到镜像中；用 EXPOSE 命令对外开放端口；用 CMD 命令描述容器启动后运行的程序。



下面是 Dockerfile 中常用的 13 种命令：

命令名称	使用
FROM	格式为 FROM image 或 FROM image:tag, 并且 Dockerfile 中第一条指令必须是 FROM 指令，且在同一个 Dockerfile 中创建多个镜像时，可以使用多个 FROM 指令
MAINTAINER	格式为 MAINTAINER username useremail, 指定维护者信息
RUN	格式为 RUN command 或 RUN ["EXECUTABLE","PARAM1","PARAM2"...], 前者在 shell 终端中运行命令，/bin/sh -c command, 例如：/bin/sh -c "echo hello"; 后者使用 exec 执行，指定其他运行终端使用 RUN["/bin/bash","-c","echo hello"]
CMD	CMD 用于指定容器启动时执行的命令，每个 Dockerfile 只能有一个 CMD 命令，多个 CMD 命令只执行最后一个。若容器启动时指定了运行的命令，则会覆盖掉 CMD 中指定的命令。支持三种格式：CMD ["executable","param1","param2"], 使用 exec 执行，这是推荐的方式。CMD command param1 param2 在 / bin/sh 中执行。CMD ["param1","param2"] 提供给 ENTRYPOINT 的默认参数。
EXPOSE	格式为 EXPOSE port [port2,port3,...], 例如 EXPOSE 80 这条指令告诉 Docker 服务器暴露 80 端口，供容器外部连接使用。在启动容器的使用使用 - P, Docker 会自动分配一个端口和转发指定的端口，使用 - p 可以具体指定使用哪个本地的端口来映射对外开放的端口。
ENV	格式为：ENV key value。用于指定环境变量，这些环境变量，后续可以被 RUN 指令使用，容器运行起来之后，也可以在容器中获取这些环境变量。例如 ENV word hello RUN echo \$word
ADD	该命令将复制指定本地目录中的文件到容器中的 dest 中，src 可以是是一个绝对路径，也可以是一个 URL 或一个 tar 文件，tar 文件会自动解压为目录。格式：ADD src dest
COPY	复制本地主机 src 目录或文件到容器的 desc 目录，desc 不存在时会自动创建。格式为：COPY src desc
ENTRYPOINT	用于配置容器启动后执行的命令，这些命令不能被 docker run 提供的参数覆盖。和 CMD 一样，每个 Dockerfile 中只能有一个 ENTRYPOINT，当有多个时最后一个生效。格式有两种：ENTRYPOINT ["executable","param1","param2"] ENTRYPOINT command param1,param2 会在 shell 中执行。



VOLUME 作用是创建在本地主机或其他容器可以挂载的数据卷，用来存放数据。 格式为 `VOLUME ["/data"]`

USER 指定容器运行时的用户名或 UID，后续的 `RUN` 也会使用指定的用户。要临时使用管理员权限可以使用 `sudo`。在 `USER` 命令之前可以使用 `RUN` 命令创建需要的用户。 格式为： `USER username`

WORKDIR 为后续的 `RUN CMD ENTRYPOINT` 指定配置工作目录，可以使用多个 `WORKDIR` 指令，若后续指令用得是相对路径，则会基于之前的命令指定路径。 格式： `WORKDIR /path`

ONBUILD 该配置指定当所创建的镜像作为其他新建镜像的基础镜像时所执行的指令。例如下面的 `Dockerfile` 创建了镜像 A `ONBUILD ADD ./app ONBUILD RUN python app.py` 则基于镜像 A 创建新的镜像时，新的 `Dockerfile` 中使用 `from A` 指定基镜像时，会自动执行 `ONBUILD` 指令内容，等价于在新的要构建镜像的 `Dockerfile` 中增加了两条指令 `FROM A ADD ./app RUN python app.py`

`docker build -t` 创建好 `Dockerfile` 之后，通过 `docker build` 命令来创建镜像，该命令首先会上传 `Dockerfile` 文件给 Docker 服务器端，服务器端将逐行执行 `Dockerfile` 中定义的指令。通常建议放置 `Dockerfile` 的目录为空目录。另外可以在目录下创建 `.dockerignore` 文件，让 Docker 忽略路径下的文件和目录，这一点与 Git 中的配置很相似 通过 `-t` 指定镜像的标签信息。 `."` 指定的是 `Dockerfile` 所在的路径。

`docker build` 运行完成后，使用 `docker images` 查看刚刚构建好的镜像：

```
[root@so4714-instance-qepkuo6yype5 sinatra]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
my/container         v1                 c419e2562b7e       4 minutes ago      251 MB
my/python            v1                 c419e2562b7e       4 minutes ago      251 MB
<none>               <none>             3ca51ddb3afe       17 minutes ago     251 MB
<none>               <none>             92c8bd34f053       46 minutes ago     284 MB
my/sinatra           v2                 6fc131c6ae56       About an hour ago  453 MB
mysql                latest             5fac85ee2c68       2 days ago         408 MB
ubuntu               latest             747cb2d60bbe       8 days ago         122 MB
cluster              latest             d8e86fa4fdab       4 weeks ago        4.18 GB
centos                latest             196e0ce0c9fb       4 weeks ago        197 MB
nkxujun/mysql_eml    latest             dd923dce8161       3 months ago       406 MB
nkxujun/ubuntu_eml   latest             ad6c47604a44       6 months ago       265 MB
training/sinatra     latest             49d952a36c58       3 years ago        447 MB
```

5. 保存/载入/删除镜像

我们的镜像做好之后, 我们要保存起来, 以供备份使用, 怎么做呢? 使用 `docker save` 命令保存镜像到本地, 通过 `-o` 指定镜像保存为文件的名字。



`docker save -o rarlinux.tar.gz rarlinux`

```
[root@so4714-instance-qepkuo6yype5 soft]# ll
total 19096
-rw-r--r-- 1 root root 19521288 Aug 29 10:40 docker-ce-17.03.0.ce-1.el7.centos.x86_64.rpm
-rw-r--r-- 1 root root 29108 Sep 14 15:07 docker-ce-selinux-17.03.0.ce-1.el7.centos.noarch.rpm
[root@so4714-instance-qepkuo6yype5 soft]#
[root@so4714-instance-qepkuo6yype5 soft]#
[root@so4714-instance-qepkuo6yype5 soft]#
[root@so4714-instance-qepkuo6yype5 soft]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
rarlinux             5.3.0              14a1b1556e64       47 minutes ago     2.55 MB
my/container         v1                 c419e2562b7e       About an hour ago   251 MB
my/python            v1                 c419e2562b7e       About an hour ago   251 MB
<none>               <none>             3ca51ddb3afe       About an hour ago   251 MB
<none>               <none>             92c8bd34f053       About an hour ago   284 MB
my/sinatra           v2                 6fc131c6ae56       2 hours ago        453 MB
mysql                latest             5fac85ee2c68       2 days ago         408 MB
ubuntu               latest             747cb2d60bbe       8 days ago         122 MB
cluster              latest             d8e86fa4fdab       4 weeks ago        4.18 GB
centos                latest             196e0ce0c9fb       4 weeks ago        197 MB
nkxujun/mysql_eml    latest             dd923dce8161       3 months ago       406 MB
nkxujun/ubuntu_eml   latest             ad6c47604a44       6 months ago       265 MB
training/sinatra     latest             49d952a36c58       3 years ago        447 MB
[root@so4714-instance-qepkuo6yype5 soft]# docker save -o rarlinux.tar.gz rarlinux
[root@so4714-instance-qepkuo6yype5 soft]# ll
total 21604
-rw-r--r-- 1 root root 19521288 Aug 29 10:40 docker-ce-17.03.0.ce-1.el7.centos.x86_64.rpm
-rw-r--r-- 1 root root 29108 Sep 14 15:07 docker-ce-selinux-17.03.0.ce-1.el7.centos.noarch.rpm
-rw----- 1 root root 2568192 Oct 19 16:39 rarlinux.tar.gz
[root@so4714-instance-qepkuo6yype5 soft]# du -s -h rarlinux.tar.gz
2.5M   rarlinux.tar.gz
```

我们有了本地的镜像文件, 在需要使用 `docker load` 将本地保存的镜像再次导入 Docker 中。

`docker load --input rarlinux.tar.gz`

或

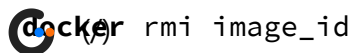
`docker load < rarlinux.tar.gz`

或

`cat rarlinux.tar.gz | docker import - rarlinux:5.3`

```
[root@so4714-instance-qepkuo6yype5 soft]# docker load < rarlinux.tar.gz
Loaded image: rarlinux:5.3.0
[root@so4714-instance-qepkuo6yype5 soft]#
```

有些镜像过时了, 我们需要删除。使用如下的命令:



或加 -f 强制删除。

```
[root@so4714-instance-qepkuo6yype5 soft]# docker rmi -f c419e2562b7e
Untagged: my/container:v1
Untagged: my/python:v1
Deleted: sha256:c419e2562b7e5c23e1e2aae1b1fd00c846cf3b74d750fb307a1dd2c3c03cd72f
Deleted: sha256:aa177d74834022625d02d4d37125626abe723b9a3a3e813ae1ef3a2a9e516e93
Deleted: sha256:f19878f4c1f476069941d2acd5cc22f5cf220f78430304f3246ea262ac39ee79
Deleted: sha256:700fa67d3545d8eb097c742193d4d255ced6c2e8e13b2723399c832539511636
```

6. Docker 容器的创建、启动、和停止

容器是独立运行的一个或一组应用，及他们的运行环境，容器是 Docker 中的一个重要的概念。

创建容器

创建容器的几种常见方式如下。

1. 基于镜像新建容器并启动，例如我们可以启动一个容器，打印出当前的日历表。

```
docker run centos cal
```

```
[root@so4714-instance-qepkuo6yype5 soft]# docker run centos cal
      October 2017
Su Mo Tu We Th Fr Sa
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31
```

2. 我们还可以通过指定参数，启动一个 bash 交互终端。

```
docker run -t -i centos /bin/bash
```



```
[root@so4714-instance-qepkuo6yype5 soft]# docker run -t -i centos /bin/bash
[root@4d8dc74f67d /]# pwd
/
[root@4d8dc74f67d /]#
[root@4d8dc74f67d /]#
[root@4d8dc74f67d /]# date
Thu Oct 19 08:57:09 UTC 2017
[root@4d8dc74f67d /]# cal
      October 2017
Su Mo Tu We Th Fr Sa
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31
```

参数 `-t` 让 Docker 分配一个伪终端并绑定在容器的标准输入上，`-i` 让容器的标准输入保持打开。使用 `docker run` 命令来启动容器，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，不存在则从公有仓库下载
- 使用镜像创建并启动容器
- 分配一个文件系统，并在只读的镜像层外面挂载一层可读可写层
- 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
- 从地址池分配一个 IP 地址给容器
- 执行用户指定的应用程序
- 执行完毕之后容器被终止

3. 以守护状态运行

很多时候，我们希望容易在后台以守护态运行，此时可以添加 `-d` 参数来实现（`d` 是 `daemon` 的首字母）例如我们启动 CentOS 后台容器，每隔一秒打印当天的日历。

```
docker run -d centos /bin/sh -c "while true;do echo hello docker;sleep
```

启动之后，我们使用 `docker ps` 查看容器的信息：

```
[root@so4714-instance-qepkuo6yype5 soft]# docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
4d66765c0756	centos	"/bin/sh -c 'while..."	5 seconds ago	Up 5 seconds	

要查看启动的 CentOS 容器中的输出日志，可以使用 `docker logs` 命令：

```
docker logs 容器名字
```

```

[roo@so4714-instance-qepkuo6yyype5 soft]# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
4993123ab04b       centos              "/bin/sh -c 'write...'" 52 seconds ago
62fa3304d994       training/sinatra    "/bin/bash"        18 minutes ago
41da21bea40e       training/sinatra    "/bin/bash"        5 hours ago
cc8c52123b27       cluster            "/bin/bash"        4 weeks ago
e4afc77cb009       cluster            "/bin/bash"        4 weeks ago
7004a79d81a8       cluster            "/bin/bash"        4 weeks ago
0.0.0.0:8081->8081/tcp, 0.0.0.0:8088->8088/tcp, 0.0.0.0:8900->8900/tcp, 0.0.0.0:9000->9000/tcp, 0.0.0.0:50030->50030/tcp, 0.0.0.0:50070->50070/tcp hadoop-master
[roo@so4714-instance-qepkuo6yyype5 soft]# docker logs frosty_brahmagupta
hello docker
hello docker
hello docker
hello docker
hello docker
hello docker
hello docker
hello docker
hello docker
hello docker
hello docker
hello docker
hello docker
hello docker
hello docker
hello docker

```



启动容器

如果容器已经停止, 可以通过 `docker start` 容器名称 重新启动容器。

docker start 容器名称

终止容器

使用 `docker stop` 来终止一个运行中的容器。并且可以使用 `docker ps -a` 来看终止状态的容器。

docker stop 容器名称

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
4993123ab04b	centos	"/bin/sh -c 'while...'"	3 minutes ago	Up 3 minutes	
5b5dc5117b62	centos	"/bin/sh -c 'while...'"	frosty_brahmagupta 4 minutes ago	Exited (1) 4 minutes ago	
4d66765c0756	centos	"/bin/sh -c 'while...'"	awesome_blackwell 6 minutes ago	Exited (137) 2 minutes ago	
c8cd9c560820	centos	"/bin/sh -c 'while...'"	dazzling_lamport 8 minutes ago	Exited (1) 8 minutes ago	
62fa3304d994	training/sinatra	"/bin/bash"	thirsty_fermi 21 minutes ago	Up 18 minutes	
d1e9e6a9f851	training/sinatra	"/bin/bash"	amazing_lamarr 21 minutes ago	Exited (0) 21 minutes ago	
effcc2beff79	ubuntu	"/bin/bash"	upbeat_northcutt 23 minutes ago	Exited (127) 22 minutes ago	
c4d8dc74f67d	centos	"/bin/bash"	angry_jepsen 30 minutes ago	Exited (127) 24 minutes ago	
e48900c6f8a6	centos	"cal"	distracted_bardeen 31 minutes ago	Exited (0) 31 minutes ago	
eb2a89f894ba	centos	"cal"	trusting_noyce 32 minutes ago	Exited (0) 32 minutes ago	
			blissful_murdock		



终止状态的容器, 可以使用 `docker start` 来重新启动。或者使用 `docker restart` 命令来重启一个容器。

```
[root@so4714-instance-qepkuo6yype5 soft]# docker restart 4993123ab04b
4993123ab04b
[root@so4714-instance-qepkuo6yype5 soft]#
[root@so4714-instance-qepkuo6yype5 soft]# docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
4993123ab04b	centos	"/bin/sh -c 'while...'"	6 minutes ago	Up 4 seconds	
			frosty_brahmagupta		

7. 进入守护状态运行的容器

在启动容器的时候, 有时候我们加了参数 `-d`, 这时容器自动进入后台运行。这个时候我们要进入容器, 怎么办? 通常使用 `docker exec` 命令来完成。命令格式为 `docker exec -it 容器名称 /bin/bash`, 表示以交互式的方式运行 `/bin/bash` 程序。

```
#docker exec -it 容器名称 /bin/bash
docker exec -it composeapp_web_1 /bin/bash
```

```
root@ubuntu:/home/hadoop/workspace/compose/composeapp# docker exec -it composeapp_web_1 /bin/bash
root@c2c26f9b7a83:/composeapp#
```

8. 导入/到处/删除容器

导出容器为快照文件



导出某个容器，非常简单，使用 `docker export` 命令。



```
[root@so4714-instance-qepkuo6yype5 ~]# docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
5d1ba7f7affc	centos	"/bin/bash"	20 hours ago	Up 20 hours
ichterman				kind_1

```
docker export 5d1ba7f7affc > centos.tar
```

导出后在本地可以看到有一个 `centos.tar` 的容器快照。

```
[root@so4714-instance-qepkuo6yype5 ~]#
[root@so4714-instance-qepkuo6yype5 ~]# docker export 5d1ba7f7affc > centos.tar
[root@so4714-instance-qepkuo6yype5 ~]#
[root@so4714-instance-qepkuo6yype5 ~]#
[root@so4714-instance-qepkuo6yype5 ~]# ll
total 245164
-rw-r--r-- 1 root root 204791296 Oct 20 13:55 centos.tar
```

从文件导入为 Docker 镜像

有了容器快照之后，我们可以在想要的时候随时导入。导入快照使用 `docker import` 命令。

例如我们可以使用 `cat centos.tar | docker import - test/centos:7` 导入容器快照作为镜像。

```
[root@so4714-instance-qepkuo6yype5 ~]# cat centos.tar | docker import - test/centos:7
sha256:9274e824ba2012f1b74bf6fa8c780af76ed47121ab727c1cb084b1bf6c294f93
[root@so4714-instance-qepkuo6yype5 ~]# docker images
REPOSITORY          TAG          IMAGE ID          CREATED
test/centos         7           9274e824ba20     7 seconds ago
```

处理本地的容器快照导入为镜像，我们还可以通过指定一个 URL 或者目录来导入。

例如在某个网络上有个快照 `image_test.tgz`

```
docker import [http://xxxx.com/image_test.tgz] (http://xxxx.com/image_te
```


Docker 中可以使用 `docker load` 来导入镜像，也可使用 `docker import` 来导入一个容器快照到 Docker 镜像。两者的区别是容器快照将丢弃所有的历史记录和元数据信息。而镜像保存完整的记录，因此要更大些。



删除镜像

删除容器。可以使用 `docker rm` 容器 id 来删除一个终止状态的容器。

```
docker rm 容器id
```

若要删除一个运行中的容器，需要加 `-f` 参数。

```
[root@so4714-instance-qepkuo6yype5 ~]#
[root@so4714-instance-qepkuo6yype5 ~]#
[root@so4714-instance-qepkuo6yype5 ~]# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS
5d1ba7f7affc        centos              "/bin/bash"         20 hours ago        Up 20 hours
[...]
```

```
[root@so4714-instance-qepkuo6yype5 ~]# docker stop 5d1ba7f7affc
5d1ba7f7affc
[...]
```

```
[root@so4714-instance-qepkuo6yype5 ~]# docker rm 5d1ba7f7affc
5d1ba7f7affc
```

9. 创建 Docker 私有仓库

有时候，在公司内部为了提高分享的速度，需要在公司内部自己搭建一个本地的仓库，供私人使用。Docker 官方提供了一个工具 `docker-registry`，我们可以借助这个工具构建私有镜像仓库。

首先，使用 `docker search` 命令查找 `registry`：

```
[root@so4714-instance-qepkuo6yype5 ~]# docker search registry
NAME                DESCRIPTION                STARS     OFFICIAL    AUTOMATED
registry            The Docker Registry 2.0 implementation for... 1705      [OK]
konradkleine/docker-registry-frontend  Browse and modify your Docker registry in ... 161
hyper/docker-registry-web  web UI, authentication service and event r... 112
atcol/docker-registry-ui  A web UI for easy private/local Docker Reg... 94
distribution/registry    WARNING: NOT the registry official image!... 53
marvambass/nginx-registry-proxy  Docker Registry Reverse Proxy with Basic A... 42
google/docker-registry    Docker Registry w/ Google Cloud Storage dr... 28
jhipster/jhipster-registry  JHipster Registry, based on Netflix Eureka... 19
deis/registry            Docker image registry for the Deis open so... 11
openshift/origin-docker-registry  The integrated OpenShift V3 registry        7
confluentinc/cn-schema-registry  Official Confluent Docker image for schem... 6
```

```
docker pull registry
```



```
[root@so4714-instance-qepkuo6yype5 docker]# docker pull registry
Using default tag: latest
latest: pulling from library/registry
90f10ba627d6: Downloading [=====>] 392.8 kB/2.385 MB
b3e11d7b4f5e: Downloading [=====>] 458.8 kB/2.009 MB
1f032f3c8932: Downloading [=====>] 3.669 MB/6.265 MB
425585e7aedb: waiting
f45f535a83d2: waiting
```

运行 registry:

```
docker run -d -p 5000:5000 -v /root/docker/registry:/tmp/registry registry
```

```
[root@so4714-instance-qepkuo6yype5 repository]# docker run -d -p 5000:5000 -v /root/docker/repository:/tmp/registry registry
595880896ee21106a480cb3531267d501eee2f90660c0e65ea53746f70231837
[root@so4714-instance-qepkuo6yype5 repository]#
[root@so4714-instance-qepkuo6yype5 repository]#
[root@so4714-instance-qepkuo6yype5 repository]# docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
595880896ee2	registry	"/entrypoint.sh /e..."	4 seconds ago	Up 3 seconds	0.0.0.0:5000->5000/tcp

默认情况下, 仓库会创建在容器中的 /tmp/registry 目录下, 通过 -v 指定将镜像文件存放在本地的目录中。

搭建好了私有仓库之后, 就可以上传、下载、搜索镜像了。

查看本地已有的镜像:

```
[root@so4714-instance-qepkuo6yype5 repository]# docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
test/centos	7	9274e824ba20	About an hour ago	197 MB
rarlinux	5.3.0	14a1b1556e64	23 hours ago	2.55 MB
my/sinatra	v2	6fc131c6ae56	24 hours ago	453 MB
reganzm/sinatra	latest	6fc131c6ae56	24 hours ago	453 MB
reganzm/ubuntu	latest	6fc131c6ae56	24 hours ago	453 MB
mysql	latest	5fac85ee2c68	3 days ago	408 MB
ubuntu	latest	747cb2d60bbe	9 days ago	122 MB
ruby	latest	eebb1381c2aa	9 days ago	684 MB
registry	latest	28525f9a6e46	4 weeks ago	33.2 MB
cluster	latest	d8e86fa4fdab	4 weeks ago	4.18 GB
centos	latest	196e0ce0c9fb	5 weeks ago	197 MB
nkxujun/mysql_eml	latest	dd923dce8161	3 months ago	406 MB
nkxujun/ubuntu_eml	latest	ad6c47604a44	6 months ago	265 MB
hjd48/redhat	latest	d7852422d6c5	3 years ago	414 MB
training/sinatra	latest	49d952a36c58	3 years ago	447 MB

查看本机 IP 地址: 192.168.0.5, 将 rarlinux 标记为 192.168.0.5:5000/rarlinux, 使用命令:

```
docker tag 14a1b1556e64 192.168.0.5:5000/rarlinux
```

```
[root@so4714-instance-qepkuo6yype5 repository]# docker tag 14a1b1556e64 192.168.0.5:5000/rarlinux
[root@so4714-instance-qepkuo6yype5 repository]# docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
test/centos	7	9274e824ba20	About an hour ago	197 MB
192.168.0.5:5000/rarlinux	latest	14a1b1556e64	23 hours ago	2.55 MB
rarlinux	5.3.0	14a1b1556e64	23 hours ago	2.55 MB
my/sinatra	v2	6fc131c6ae56	24 hours ago	453 MB

使用 `docker push` 上传标记的镜像:



```
docker push 192.168.0.5:5000/rarlinux
```

```
[root@so4714-instance-qepkuo6yype5 repository]# docker push 192.168.0.5:5000/rarlinux
The push refers to a repository [192.168.0.5:5000/rarlinux]
Get https://192.168.0.5:5000/v1/_ping: http: server gave HTTP response to HTTPS client
[root@so4714-instance-qepkuo6yype5 repository]#
```

报错, 解决方法是在 `/etc/docker` 目录下新建 `daemon.json`, 文件中写入:

```
{ "insecure-registries":["192.168.0.5:5000"] }
```

然后重启 Docker:

```
systemctl restart docker
```

重新运行

```
docker run -d -p 5000:5000 -v /root/docker/registry:/tmp/registry regis
```

启动 registry 服务, 再次运行

```
docker push 192.168.0.5:5000/rarlinux
```

上传镜像到私有仓库。

```
[root@so4714-instance-qepkuo6yype5 docker]# docker push 192.168.0.5:5000/rarlinux
The push refers to a repository [192.168.0.5:5000/rarlinux]
226a45e157d8: Pushed
latest: digest: sha256:f37b784157a88faa4f656c5c07f34f48a3bfc5d7974e4a0671742e77b81a23f0 size: 527
[root@so4714-instance-qepkuo6yype5 docker]#
```

接下来, 我们可以到另一台机器 192.168.0.8 下载这个镜像了:

```
[root@so4723-instance-ih7bw5ytnbwd ~]# docker pull 192.168.0.5:5000/centos
Using default tag: latest
Trying to pull repository 192.168.0.5:5000/centos ...
latest: Pulling from 192.168.0.5:5000/centos
c31953c83361: Pull complete
Digest: sha256:72d2589f87392dffcb7684147c7238a6f4dbb96116c6b08181ea60fd07ababaa
```

这样 Docker 私有仓库就搭建好了。

10 Docker 容器绑定外部 IP 和端口



Docker 允许通过外部访问容器或者容器之间互联的方式来提供网络服务。容器启动之后, 容器中可以运行一些网络应用, 通过 `-p` 或 `-P` 参数来指定端口映射, 使用 `-P` (大写) 标记时, Docker 会随机选择一个端口映射到容器内部开放的网络端口上。

```
[root@so4714-instance-qepkuo6yye5 ~]# docker run -d -P training/webapp python app.py
b299ca479b5458643fa1385989303433645e6072349de86dde54346e8062077a
[root@so4714-instance-qepkuo6yye5 ~]# docker ps
```

CONTAINER ID	IMAGE	COMMAND	NAMES	CREATED	STATUS	PORTS
b299ca479b54	training/webapp	"python app.py"	happy_bell	4 seconds ago	Up 3 seconds	0.0.0.0:32786->5000/tcp

此时访问本机的 32786 端口就可以访问到容器内 Web 应用提供的界面。



也可以使用 `docker logs` 来查看应用的信息:

```
[root@so4714-instance-qepkuo6yye5 ~]# docker logs happy_bell
* Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
223.87.206.152 - - [22/Oct/2017 01:36:11] "GET / HTTP/1.1" 200 -
223.87.206.152 - - [22/Oct/2017 01:36:11] "GET /favicon.ico HTTP/1.1" 404 -
[root@so4714-instance-qepkuo6yye5 ~]#
```

使用 `-p` (小写) 则可以指定要映射的端口, 并且在一个指定端口上只可以绑定一个容器, 支持的格式有:

ip:hostport:containerport ip::containerport hostport:containerport

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

```
[root@so4714-instance-qepkuo6yye5 ~]# docker run -d -p 5000:5000 training/webapp python app.py
1e9c982114593830e5cca5f58315d71aa3b3fa32cd97a42942edf9fc40e4550
docker: Error response from daemon: driver failed programming external connectivity on endpoint gracious_franklin (1:75e): Bind for 0.0.0.0:5000 failed: port is already allocated.
[root@so4714-instance-qepkuo6yye5 ~]#
[root@so4714-instance-qepkuo6yye5 ~]# netstat -lnp | grep 5000
tcp6      0      0 :::5000               :::*                    LISTEN      4763/docker-proxy
[root@so4714-instance-qepkuo6yye5 ~]#
[root@so4714-instance-qepkuo6yye5 ~]# docker run -d -p 5001:5000 training/webapp python app.py
3c18ad026821566f5aa7b4ab6b16ae211654ff9fe86d480aa3bc63418b694a7e
[root@so4714-instance-qepkuo6yye5 ~]#
[root@so4714-instance-qepkuo6yye5 ~]# docker logs 3c18ad026821566f5aa7b4ab6b16ae211654ff9fe86d480aa3bc63418b694a7e
* Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
[root@so4714-instance-qepkuo6yye5 ~]#
```

看到本地 5000 端口已经被映射, 换一个端口 5001 运行成功, 在页面上输入 5001 端口访问如下。



Docker 默认会映射本地所有的地址。现在我们来尝试一下映射到指定地址的指定端口:

```
docker run -d -p 127.0.0.1:5000:5000 training/webapp python app.py
```

```
[root@so4714-instance-qepkuo6yyype5 ~]# docker run -d -p 127.0.0.1:5000:5000 training/webapp python app.py
b0c3b993f67df34999e20c1f8fef033495e8698c99197626ed889a983a0d1168
docker: Error response from daemon: driver failed programming external connectivity on endpoint laughing_panini (2f07b8be27c31d73acf884dfaaf368f4450ef1efdb1516c038096d5070f979728): Error starting userland proxy: listen tcp 127.0.0.1:5000: bind: address already in use.
[root@so4714-instance-qepkuo6yyype5 ~]#
[root@so4714-instance-qepkuo6yyype5 ~]# docker run -d -p 127.0.0.1:5002:5000 training/webapp python app.py
c206840f4abe3e1400f381537fdc046d2c5303285668ac32abeceac4ef1cb26
[root@so4714-instance-qepkuo6yyype5 ~]#
```

然后要访问容器中的应用只能通过 127.0.0.1 这个 IP 访问。接下来是绑定本机的任意端口到容器的 5000 端口。

```
docker run -d -p 127.0.0.1::5000 training/webapp python app.py
```

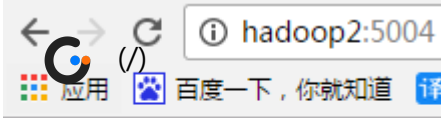
```
Last login: Sun Oct 22 09:45:04 2017 from 223.87.206.152
[root@so4714-instance-qepkuo6yyype5 ~]# docker run -d -p 127.0.0.1::5000 training/webapp python app.py
f07b8be27c31d73acf884dfaaf368f4450ef1efdb1516c038096d5070f979728
[root@so4714-instance-qepkuo6yyype5 ~]# docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
f07b8be27c31	training/webapp	"python app.py"	5 seconds ago	Up 5 seconds	127.0.0.1:32768->5000/tcp

还可以指定通信协议:

```
docker run -d -p 5003:5000/udp training/webapp python app.py
```

```
0.0.0.0:8081->8081/tcp, 0.0.0.0:8088->8088/tcp, 0.0.0.0:8900->8900/tcp, 0.0.0.0:9000->9000/tcp, 0.0.0.0:11000->11000/tcp, 0.0.0.0:50030->50030/tcp, 0.0.0.0:50070->50070/tcp hadoop-master
[root@so4714-instance-qepkuo6yyype5 ~]# docker run -d -p 5003:5000/udp training/webapp python app.py
04eec5d91386614d9acdf2e1cea10743509d8902867e9a13d5b04f40fb6b7401
[root@so4714-instance-qepkuo6yyype5 ~]#
[root@so4714-instance-qepkuo6yyype5 ~]# docker run -d -p 5004:5000/tcp training/webapp python app.py
0aab274706abaa78b8bbeb5e55d63704d300ec239c661b556f3d60e2d66b4157
[root@so4714-instance-qepkuo6yyype5 ~]#
```

Hello world!

那我们怎样来查看 Docker 里面容易绑定和映射的端口及 IP 地址呢？可以使用 `docker port`：

```
[root@so4714-instance-qepkuo6yype5 ~]# docker port boring_sammet
5000/tcp -> 127.0.0.1:32768
[root@so4714-instance-qepkuo6yype5 ~]#
```

容器内部有自己的内部网络和 IP 地址，可以使用 `docker inspect` 来查看：

```
[root@so4714-instance-qepkuo6yype5 ~]# docker inspect boring_sammet
[
  {
    "Id": "f07b8be27c31d73acf884dfaaf368f4450ef1efdb1516c038096d5070f979728",
    "Created": "2017-10-22T01:50:03.190298299Z",
    "Path": "python",
    "Args": [
      "app.py"
    ],
    "State": {
      "Status": "running",
      "Running": true,
      "Paused": false,
      "Restarting": false,
      "OOMKilled": false,
      "Dead": false,
      "Pid": 14238,
      "ExitCode": 0,
      "Error": "",
      "StartedAt": "2017-10-22T01:50:03.451185699Z",
      "FinishedAt": "0001-01-01T00:00:00Z"
    },
    "Image": "sha256:6fae60ef344644649a39240b94d73b8ba9c67f898ede85cf8e947a887b3e6557",
    "ResolvConfPath": "/var/lib/docker/containers/f07b8be27c31d73acf884dfaaf368f4450ef1efdb1516c038096d5070f979728/resolv.conf",
    "HostnamePath": "/var/lib/docker/containers/f07b8be27c31d73acf884dfaaf368f4450ef1efdb1516c038096d5070f979728/hostname",
    "HostsPath": "/var/lib/docker/containers/f07b8be27c31d73acf884dfaaf368f4450ef1efdb1516c038096d5070f979728/hosts",
    "LogPath": "/var/lib/docker/containers/f07b8be27c31d73acf884dfaaf368f4450ef1efdb1516c038096d5070f979728/log",
    "Name": "/boring_sammet",
    "RestartCount": 0,
    "Driver": "overlay",
    "MountLabel": "",
    "ProcessLabel": "",
    "AppArmorProfile": "",
    "ExecIDs": null,
    "HostConfig": {
      "Binds": null,
      "ContainerIDFile": "",
      "LogConfig": {
        "Type": "json-file",
        "Config": {}
      },
      "NetworkMode": "default",
      "PortBindings": {
        "5000/tcp": [
          {
            "HostIp": "127.0.0.1",
            "HostPort": ""
          }
        ]
      }
    }
  }
]
```

在启动容器的时候，可以多次使用 `-p` 标记来绑定多个端口：

```
docker run -d -p 5005:5000 -p 5006:80 training/webapp python app.py
```

```

root@sq4714-instance-qepkuo6yye5 ~]# docker run -d -p 5005:5000 -p 5006:80 training/webapp python app.py
d818e67b6cb7813975853815efa6264012bdc9fbb31545de1086ac83c17f42a
[roo@sq4714-instance-qepkuo6yye5 ~]# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS
d818e67b6cb        training/webapp     "python app.py"     7 seconds ago      Up 6 seconds       0.0.0.0:5006->80/
0aab274706ab        training/webapp     "python app.py"     6 minutes ago      Up 6 minutes       0.0.0.0:5004->5000
04eec5d91386        training/webapp     "python app.py"     6 minutes ago      Up 6 minutes       5000/tcp, 0.0.0.0:
f07b8be27c31        training/webapp     "python app.py"     8 minutes ago      Up 8 minutes       127.0.0.1:32768->5
c206840f4abe        training/webapp     "python app.py"     12 minutes ago     Up 12 minutes      127.0.0.1:5002->50
96a4fc8ed071        centos              "/bin/bash"         39 minutes ago     Up 39 minutes
cc630dfbf3de        centos              "/bin/bash"         43 minutes ago     Up 43 minutes
f4b66ac834f9        registry            "/entrypoint.sh /e... 21 hours ago       Up 21 hours         0.0.0.0:5000->5000
cc8c52123b27        cluster            "/bin/bash"         4 weeks ago        Up 42 hours         0.0.0.0:50011->500
tcp
e4afc77cb009        cluster            "/bin/bash"         4 weeks ago        Up 42 hours         0.0.0.0:8042->8042
tcp
7004a79d81a8        cluster            "/bin/bash"         4 weeks ago        Up 42 hours         0.0.0.0:7077->7077
0.0.0.0:8081->8081/tcp, 0.0.0.0:8088->8088/tcp, 0.0.0.0:8900->8900/tcp, 0.0.0.0:9000->9000/tcp, 0.0.0.0:11000->11000/tcp,
0.0.0.0:50030->50030/tcp, 0.0.0.0:50070->50070/tcp
[roo@sq4714-instance-qepkuo6yye5 ~]# docker port optimistic_mcclintock
5000/tcp -> 0.0.0.0:5005
80/tcp -> 0.0.0.0:5006
[roo@sq4714-instance-qepkuo6yye5 ~]#

```



11. 容器互联

除了端口映射之外，容器互联是另一种跟容器应用交互的方式。它会在源容器和接收容器之间建立一个隧道，接收容器可以看到源容器指定的信息。

要实现容器互联，需要为容器指定一个好记的名字，通过 `-name` 来制定，若不指定，Docker 会随机生成一个容器的名称，但这不利于记忆。若一个容器是临时的，运行完成之后要自动删除，需要加上 `-rm` 标记。

```

[root@so4714 ~]# docker run -d -p 8080:80 --name test1 training/webapp python app.py
e386a4fd7e87d8745ee3697f5449c861d07f29480a6eac740940e7332ac8f856
[root@so4714 ~]# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS
e386a4fd7e87       training/webapp    "python app.py"    4 seconds ago      Up 3 seconds       0.0.0.0:32787->5000/tcp

```

容器互联：使用 `-link` 参数，可以让容器之间安全的进行交互。

先创建一个数据库容器:

```
docker run -d --name db training/postgres
```

新建一个容器和 Postgres 容器互联:



```
docker run -d -P --name web --link db:db training/webapp python app.py
```



-link 表示建立容器互联, 参数为 name:alias, name 是要链接的容器名称, alias 是我们取得别名。

通过 -link 的方式, 是 Web 和 DB 建立了链接, 我们可以查看下 Web 容器的环境变量:

```
docker run --rm --name web2 --link db:db training/webapp env
```

```
[root@so4714-instance-qepkuo6yype5 ~]#
[root@so4714-instance-qepkuo6yype5 ~]# docker run --rm --name web2 --link db:db training/webapp env
docker: Error response from daemon: Cannot link to a non running container: /db AS /web2/db.
[root@so4714-instance-qepkuo6yype5 ~]#
[root@so4714-instance-qepkuo6yype5 ~]#
[root@so4714-instance-qepkuo6yype5 ~]# docker start db
db
[root@so4714-instance-qepkuo6yype5 ~]# docker run --rm --name web2 --link db:db training/webapp env
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
HOSTNAME=76a60d26fdb0
DB_PORT=tcp://172.17.0.3:5432
DB_PORT_5432_TCP=tcp://172.17.0.3:5432
DB_PORT_5432_TCP_ADDR=172.17.0.3
DB_PORT_5432_TCP_PORT=5432
DB_PORT_5432_TCP_PROTO=tcp
DB_NAME=/web2/db
DB_ENV_PG_VERSION=9.3
HOME=/root
[root@so4714-instance-qepkuo6yype5 ~]# █
```

可以看到环境变量中以要链接的容器的名字的大写作为前缀, 和 DB 建立连接之后, 除了 Web 容器环境变量发生了变化, 在 Web 容器的 hosts 文件也发生了变化, 我们可以看下:

```
docker run -it --rm --link db:db training/webapp /bin/bash
```



```

root@db698c5f8d86:/opt/webapp# cat /etc/hosts
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
172.17.0.3 db 525088b72079
172.17.0.5 db698c5f8d86
root@db698c5f8d86:/opt/webapp# ifconfig
eth0      Link encap:Ethernet  HWaddr 02:42:ac:11:00:05
          inet addr:172.17.0.5  Bcast:0.0.0.0  Mask:255.255.0.0
          inet6 addr: fe80::42:acff:fe11:5/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:8 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:648 (648.0 B)  TX bytes:648 (648.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

root@db698c5f8d86:/opt/webapp#

```

db的Ip

这里面有两个 IP, 一个是 Web 的, 一个是 DB 的, 我们可以用 ping 命令查看这两个 IP 之间能否 ping 通。

```

[roo@so4714-instance-qepkuo6yype5 ~]# docker run -it --rm --link db:db training/webapp /bin/bash
root@fe0b23b16211:/opt/webapp# cat /etc/hosts
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
172.17.0.3 db 525088b72079
172.17.0.5 fe0b23b16211
root@fe0b23b16211:/opt/webapp# ping db
PING db (172.17.0.3) 56(84) bytes of data.
64 bytes from db (172.17.0.3): icmp_seq=1 ttl=64 time=0.225 ms
64 bytes from db (172.17.0.3): icmp_seq=2 ttl=64 time=0.145 ms
64 bytes from db (172.17.0.3): icmp_seq=3 ttl=64 time=0.161 ms
^C
--- db ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1999ms

```

建立链接没问题! 我们启动 DB 容器的时候, 没有通过 -p 指定端口, 避免了数据库端口暴露在外部网络, 这样很安全。

12. 一个完整的小例子

例子主要完成自己镜像的制作、端口映射、Dockerfile 文件编写。

首先选择一个合适的目录：/root/docker/hello，创建 Dockerfile，里面定义了我们构建镜像的指令步骤。具体定义如下：



```
#一个基础的python运行环境
FROM python
#设置工作目录
WORKDIR /app
#将当前系统文件夹内容复制到容器的app目录
ADD . /app
#安装必要的依赖包
RUN pip install -r softwares.txt
#开放端口，供容器外访问
EXPOSE 80
#定义环境变量
ENV NAME Hello_docker
#运行命令
CMD ["python","app.py"]
```

在 Dockerfile 中定义了我们要在容器中运行 `pip install` 命令安装软件，软件定义在 `softwares.txt` 中，要使用 Python 命令运行 `app.py` 文件，因此这两个文件需要我们定义，`softwares.txt` 文件定义内容：

```
Flask
Redis
```

`app.py` 文件内容：



```

from flask import Flask
from redis import Redis, RedisError
import os
import socket# Connect to Redis
redis = Redis(host="redis", db=0, socket_connect_timeout=2, socket_time
def hello():
    try:
        visits = redis.incr("counter")
    except RedisError:
        visits = "<i>cannot connect to Redis, counter disabled</i>" htm
        "<b>Hostname:</b> {hostname}<br/>" \
        "<b>Visits:</b> {visits}"
    return html.format(name=os.getenv("NAME", "world"), hostname=socket
app.run(host='0.0.0.0', port=80)

```

使用 docker build 构建镜像:


docker build -t hello .

```

[root@so4714-instance-qepkuo6yype5 hello]#
[root@so4714-instance-qepkuo6yype5 hello]# docker build -t hello .
Sending build context to Docker daemon 4.608 kB
Step 1/7 : FROM python
----> 01fd71a97c19
Step 2/7 : WORKDIR /app
----> 59e624c3e93f
Removing intermediate container 9eee1aca1072
Step 3/7 : ADD . /app
----> 4cd37c9b52f0
Removing intermediate container a0ee6db5ec82
Step 4/7 : RUN pip install -r softwares.txt
----> Running in 59cb05689a28
Collecting Flask (from -r softwares.txt (line 1))
  Downloading Flask-0.12.2-py2.py3-none-any.whl (83kB)
Collecting Redis (from -r softwares.txt (line 2))
  Downloading redis-2.10.6-py2.py3-none-any.whl (64kB)
Collecting Jinja2>=2.4 (from Flask->-r softwares.txt (line 1))
  Downloading Jinja2-2.9.6-py2.py3-none-any.whl (340kB)
Collecting Werkzeug>=0.7 (from Flask->-r softwares.txt (line 1))
  Downloading Werkzeug-0.12.2-py2.py3-none-any.whl (312kB)
Collecting itsdangerous>=0.21 (from Flask->-r softwares.txt (line 1))
  Downloading itsdangerous-0.24.tar.gz (46kB)
Collecting click>=2.0 (from Flask->-r softwares.txt (line 1))
  Downloading click-6.7-py2.py3-none-any.whl (71kB)
Collecting MarkupSafe>=0.23 (from Jinja2>=2.4->Flask->-r softwares.txt (line 1))
  Downloading MarkupSafe-1.0.tar.gz
Building wheels for collected packages: itsdangerous, MarkupSafe
Running setup.py bdist_wheel for itsdangerous: started
Running setup.py bdist_wheel for itsdangerous: finished with status 'done'
Stored in directory: /root/.cache/pip/wheels/fc/a8/66/24d655233c757e178d45dea2de22a04c6d92766abfb741129a
Running setup.py bdist_wheel for MarkupSafe: started
Running setup.py bdist_wheel for MarkupSafe: finished with status 'done'
Stored in directory: /root/.cache/pip/wheels/88/a7/30/e39a54a87bcb25308fa3ca64e8ddc75d9b3e5afa21ee32d57
Successfully built itsdangerous MarkupSafe
Installing collected packages: MarkupSafe, Jinja2, Werkzeug, itsdangerous, click, Flask, Redis
Successfully installed Flask-0.12.2 Jinja2-2.9.6 MarkupSafe-1.0 Redis-2.10.6 Werkzeug-0.12.2 click-6.7 itsdangerous-0.24
----> f2599c632b68
Removing intermediate container 59cb05689a28
Step 5/7 : EXPOSE 80
----> Running in 58cad10e52d5
----> 3ece76311254
Removing intermediate container 58cad10e52d5
Step 6/7 : ENV NAME Hello_docker
----> Running in 1840259b79dd
----> e42fec95e266
Removing intermediate container 1840259b79dd
Step 7/7 : CMD python app.py
----> Running in 63559ccbca2c
----> 9723ed500784
Removing intermediate container 63559ccbca2c
Successfully built 9723ed500784
[root@so4714-instance-qepkuo6yype5 hello]#

```

构建完成镜像之后, 使用 docker run 启动容器:

 `docker run -p 9876:80 hello`


```

[root@so4714-instance-qepkuo6yype5 hello]# docker run -p 9876:80 hello
* Running on http://0.0.0.0:80/ (Press CTRL+C to quit)

```

运行起来之后，网页访问 9876 端口`



Hello Hello_docker!

Hostname: aaa27a057030

Visits: *cannot connect to Redis, counter disabled*

由于没有启动 Redis，很明显 app.py 文件中抛出了不能够连接到 Redis 中。这个地方我们没有配置 Redis，那我们就来配置下 Redis 吧，首先使用 `docker pull redis` 下载 Redis 镜像：

```

docker pull redis
docker run --name redis -d redis

```

我们再启动一个容器，运行我们的 hello 镜像，`-name` 指定容器的名称，`-link` 用于连接 Redis 容器，`-p` 用于端口映射。

```

docker run -d --name hello --link redis:redis -p 9876:80 hello

```

现在浏览器中访问：`http://hadoop2:9876`。



Hello Hello_docker!

Hostname: fbe8883a1b6e

Visits: 19

至此，完成了使用两个容器实现了一个简单的网络点击次数的计数器的功能。



本文首发于 GitChat，未经授权不得转载，转载需与 GitChat 联系。



11



0

互动评论



说点什么

评论



侯佳林

3 个月前

-1



鼓掌



舞动青春

2 年前

这篇文章写的也太差了



鼓掌



安静

2 年前

`docker commit -m 'add json gem' -a 'Docker Container' 7b789b19757d my/sinatra:v2` 这里的7b789b19757d怎么来？执行报错说没有这个container



鼓掌



奔跑的小米（作者）

2 年前

这个随机生成的containerid，每个容器id是不一样的，不要复制我这个id,找你的



鼓掌



一步

2 年前

yum 安装 python 的时候 `-qq` 参数是代表什么意思的？查询了一下没有找到合适的解释



查看更多