1 指令概览

官网原文 中文翻译

使用Dockerfile，可以将构建的过程，记录下来，重复使用。

> 单纯地记指令，我是记不住的。所以，每一个指令，我会先查阅资料，在了解用法之后，假想一个场景，然后练习。之后，再回头记录指令的详细用法。最后，整理一份参考手册

指令	描述	备注
`ARG`	设置变量（参数）	
`FROM`	指定基础镜像	必须是第一条指令
`LABEL/MAINTAINER`	添加元数据	相当于注释，比如作者/版本号/备注等
`RUN`	执行命令	通常是安装软件
`COPY` / `ADD`	复制文件	
`CMD`	容器启动时的命令	
`ENTRYPOINT`		
`ENV`	设置环境变量	
`VOLUME`	数据卷	
`EXPOSE`	暴露端口	
`WORKDIR`	设置工作目录	
`USER`	指定用户	
`HEALTHCHECK`	健康检查	
`ONBUILD`		

2 pull nginx

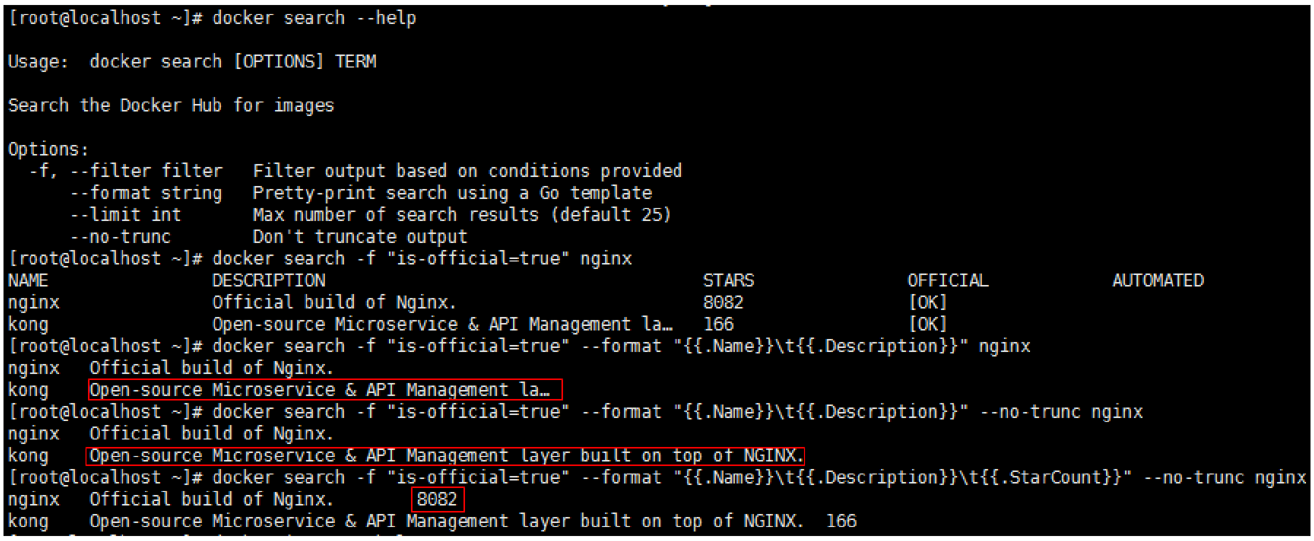
> 上一小节删掉了本地除了centos之外的所有镜像。

这一小节，先从dockerhub上pull一个nginx镜像。后面，基于这个镜像，学习Dockerfile的命令。

1. 查看本地的所有镜像

```
[root@localhost ~]# docker images -a
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
centos              latest              2d194b392dd1        3 days ago          195MB
```

2. 查找nginx镜像（多换几个姿势）



```
[root@localhost ~]# docker search --help

Usage:  docker search [OPTIONS] TERM

Search the Docker Hub for images

Options:
  -f, --filter filter   Filter output based on conditions provided
      --format string   Pretty-print search using a Go template
      --limit int       Max number of search results (default 25)
      --no-trunc        Don't truncate output
[root@localhost ~]# docker search -f "is-official=true" nginx
NAME                DESCRIPTION                                     STARS               OFFICIAL            AUTOMATED
nginx               Official build of Nginx.                        8082                [OK]
kong                Open-source Microservice & API Management la…   166                 [OK]
[root@localhost ~]# docker search -f "is-official=true" --format "{{.Name}}\t{{.Description}}" nginx
nginx   Official build of Nginx.
kong    Open-source Microservice & API Management la…
[root@localhost ~]# docker search -f "is-official=true" --format "{{.Name}}\t{{.Description}}" --no-trunc nginx
nginx   Official build of Nginx.
kong    Open-source Microservice & API Management layer built on top of NGINX.
[root@localhost ~]# docker search -f "is-official=true" --format "{{.Name}}\t{{.Description}}\t{{.StarCount}}" --no-trunc nginx
nginx   Official build of Nginx.        8082
kong    Open-source Microservice & API Management layer built on top of NGINX.  166
```

3. pull nginx

```
[root@localhost ~]# clear
[root@localhost ~]# docker pull nginx
[root@localhost ~]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
centos              latest              2d194b392dd1        3 days ago          195MB
nginx               latest              e548f1a579cf        2 weeks ago         109MB
```

3 比较镜像的大小

还记得上一小节基于centos创建的镜像SIZE吗？

来源	镜像name	size
docker commit	local/nginx	440MB
Dockerfile	local/nginx2	632MB
dockerhub	nginx	109M

镜像的SIZE区别很大，但，功能是一样的。

4 FROM & RUN

4.1 示例：修改nginx的首页

4.1.1 Dockerfile预览

基于nginx镜像，修改nginx首页的内容

```
FROM nginx
RUN echo '<h1>Hello Docker!</h1>' > /usr/share/nginx/html/index.html
```

> 请注意，命令必须使用单引号，因为`!`是特殊字符

```
[root@localhost ~]# echo "<h1>Hello Docker!</h1>"
-bash: !: event not found
[root@localhost ~]# echo '<h1>Hello Docker!</h1>'
<h1>Hello Docker!</h1>
```

4.1.2 创建Dockerfile

建议：每一个命令，都要做实验。所以，创建目录&images的时候，名字起的稍微有规律点。

```
[root@localhost ~]# mkdir -p dockerfile/nginx/t1
[root@localhost ~]# vim dockerfile/nginx/t1/Dockerfile
```

4.1.3 build image

```
[root@localhost ~]# docker build -t="local/t1/nginx" dockerfile/nginx/t1/.
Sending build context to Docker daemon  2.048kB
Step 1/2 : FROM nginx
 ---> e548f1a579cf
Step 2/2 : RUN echo '<h1>Hello Docker!</h1>' > /usr/share/nginx/html/index.html
 ---> Running in bea1b5155d6c
Removing intermediate container bea1b5155d6c
 ---> e07551c732a2
Successfully built e07551c732a2
Successfully tagged local/t1/nginx:latest
```

```
[root@localhost ~]# docker images -a
REPOSITORY          TAG                 IMAGE ID            CREATED              SIZE
local/t1/nginx      latest              e07551c732a2        About a minute ago   109MB
centos              latest              2d194b392dd1        4 days ago           195MB
nginx               latest              e548f1a579cf        2 weeks ago          109MB
[root@localhost ~]# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
   PORTS               NAMES
```

镜像“local/t1/nginx”创建好了，但是，遗憾的是，nginx容器没有启动

4.1.4 进入容器 & 查看nginx首页

```
[root@localhost ~]# docker run -it --name n1 local/t1/nginx /bin/bash
root@ba24b3a3230b:/# cat /usr/share/nginx/html/index.html
<h1>Hello Docker!</h1>
```

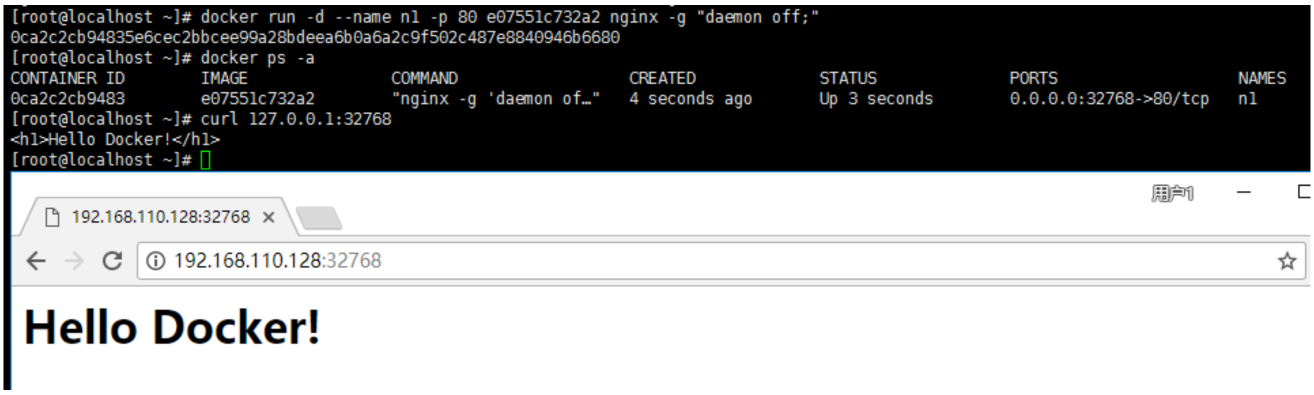
4.1.5 启动nginx & 查看首页

1. 清理容器（去掉不必要的干扰，我觉得是个好习惯）

```
[root@localhost ~]# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
            PORTS               NAMES
ba24b3a3230b        local/t1/nginx      "/bin/bash"         4 minutes ago       Exited (127) 4
seconds ago                       n1
[root@localhost ~]# docker rm ba24b3a3230b
ba24b3a3230b
[root@localhost ~]#
```

2. 参考上一章的命令

```
[root@localhost ~]# docker run -d --name n1 -p 80 e07551c732a2 nginx -g "daemon off;"
```



```
[root@localhost ~]# docker run -d --name n1 -p 80 e07551c732a2 nginx -g "daemon off;"
0ca2c2cb94835e6cec2bbcee99a28bdeea6b0a6a2c9f502c487e8840946b6680
[root@localhost ~]# docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                   NAMES
0ca2c2cb9483        e07551c732a2        "nginx -g 'daemon of…"   4 seconds ago       Up 3 seconds        0.0.0.0:32768->80/tcp   n1
[root@localhost ~]# curl 127.0.0.1:32768
<h1>Hello Docker!</h1>
[root@localhost ~]#
```

Hello Docker!

4.2 FROM

制作镜像之前，必须依赖一个镜像，被称为“基础镜像”。docker hub上有很多高质量的官方镜像：nginx/redis/mongo/mysql等。可以直接拿来做基础镜像。

如果找不到合适的，还可以基于更为基础的操作系统镜像：ubuntu/debian/centos/fedora等。

> 前面的练习，得到个教训：自己制作的镜像，体积很大。所以，如果对linux不是很熟悉，基于官网的镜像，是很好的选择。毕竟，装的软件越少，系统的性能越好。至于安全，是另一个范畴的知识了，后面再讨论。

FROM指令的格式	**备注**
`FROM <image> [AS <name>]`	`AS name`，取别名
`FROM <image>[:<tag>] [AS <name>]`	`tag`，最新版`latest`，或特定的版本号
`FROM <image>[@<digest>] [AS <name>]`	`digest`？

4.2.1 digest

digest，在pull image的时候，会打印出来

```
[root@localhost ~]# docker pull tomcat
Using default tag: latest
latest: Pulling from library/tomcat
3e731ddb7fc9: Pull complete
47cafa6a79d0: Pull complete
79fcf5a213c7: Pull complete
fd532571c5d3: Pull complete
31600c9f9b48: Pull complete
78e8e9b5d10e: Pull complete
0710e619e883: Pull complete
e511da65ffab: Pull complete
bfcd79b5752d: Pull complete
f6d4311e9998: Pull complete
aa08dc0fba1c: Pull complete
47ffc8d9d1f8: Pull complete
Digest: sha256:63f6cc392882d782a5afd98ceebc1452958ef78631cb3df7704093c9826d9b26
Status: Downloaded newer image for tomcat:latest
[root@localhost ~]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
local/t1/nginx      latest              e07551c732a2        About an hour ago   109MB
centos              latest              2d194b392dd1        4 days ago          195MB
nginx               latest              e548f1a579cf        2 weeks ago         109MB
tomcat              latest              108db0e7c85e        3 weeks ago         558MB
```

为了验证，专门pull了个tomcat，没想到，东西还挺多。比较digest和image id，看起来也不一样。

请注意，后面还有个sha256。其实，digest是校验文件完整性的算法，尤其是当文件很多的时候，校验是必要的。否则，如果从官网下载的软件，被人篡改后，加了木马，就麻烦了。

常见的算法:sha256、md5等。

暂时，用不到tomcat，所以，清理下（文件有点多）

```
[root@localhost ~]# docker images -a
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
local/t1/nginx      latest              e07551c732a2        2 hours ago         109MB
centos              latest              2d194b392dd1        4 days ago          195MB
nginx               latest              e548f1a579cf        2 weeks ago         109MB
tomcat              latest              108db0e7c85e        3 weeks ago         558MB
[root@localhost ~]# docker rmi 108db0e7c85e
Untagged: tomcat:latest
Untagged: tomcat@sha256:63f6cc392882d782a5afd98ceebc1452958ef78631cb3df7704093c9826d9b26
Deleted: sha256:108db0e7c85ee5e343b58e96940b05c53d871927778ebdbbfcf1cab1d014c3d9
Deleted: sha256:44e8099eea52167ec1af97a3212a56408cbf8f3f0764b43535705e9af6c1585d
Deleted: sha256:60bfe8910209e97ad8aee08b9f0a98833fba11dbd137f7b267e3f6320d9e130e
Deleted: sha256:e327090945ed55338070db5a853024974e6e16b33d7d391ac378ee42eb12d2a3
Deleted: sha256:cb43fee61354cc370afde25d2e493e86ddabf4111207a851c2382c85206b8db2
Deleted: sha256:f9b580e11770bad5ecbdfb3ced09a647e096135b54a5fa47a6a284a7eae60db7
```

```
Deleted: sha256:1eebdecf770f18ff6f51a82b63c6ee2a49e0823499350ff73f4d425bf3f02816
Deleted: sha256:954adb4f7b252c4deea0204f8e091b57f095cfb2ad0dc3b69bd4bc6e23c1935e
Deleted: sha256:7f4bfadf138a909d176eec03a886afeca171051186e3fd5ed5f23c5b6f1b9e32
Deleted: sha256:b55b02b02cddc2a87422cbc2d4d6bf9e4575b66aa2f2972075455137058748bf
Deleted: sha256:8d8b933a99c7c507cc21924b9e71ba9a4c9bee836ddfbf5a92678259ce0cc881
Deleted: sha256:fb65b9418cd06c4efac7ccb9740ef4bb006cc34b79819bb6e841a2c6d60d1a26
Deleted: sha256:8568818b1f7f534832b393c531edfcb4a30e7eb40b573e68fdea90358987231f
[root@localhost ~]# docker images -a
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
local/t1/nginx      latest              e07551c732a2        2 hours ago         109MB
centos              latest              2d194b392dd1        4 days ago          195MB
nginx               latest              e548f1a579cf        2 weeks ago         109MB
[root@localhost ~]#
```

4.2.2 ARG

声明变量（参数）

通常，`FROM`指令是第一个，但，`ARG`例外，可以放在`FROM`前面

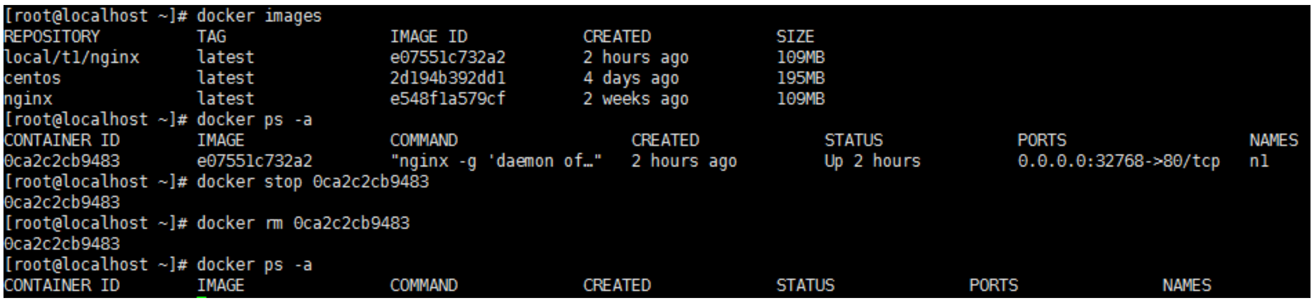
~~4.2.2.1 示例1~~✕

`dockerfile/nginx/t2/Dockerfile`

```
ARG VERSION=latest
FROM nginx:$VERSION
RUN echo '<h1>Hello Docker!</h1><p>nginx version is '$VERSION'</p>' >
/usr/share/nginx/html/index.html
```

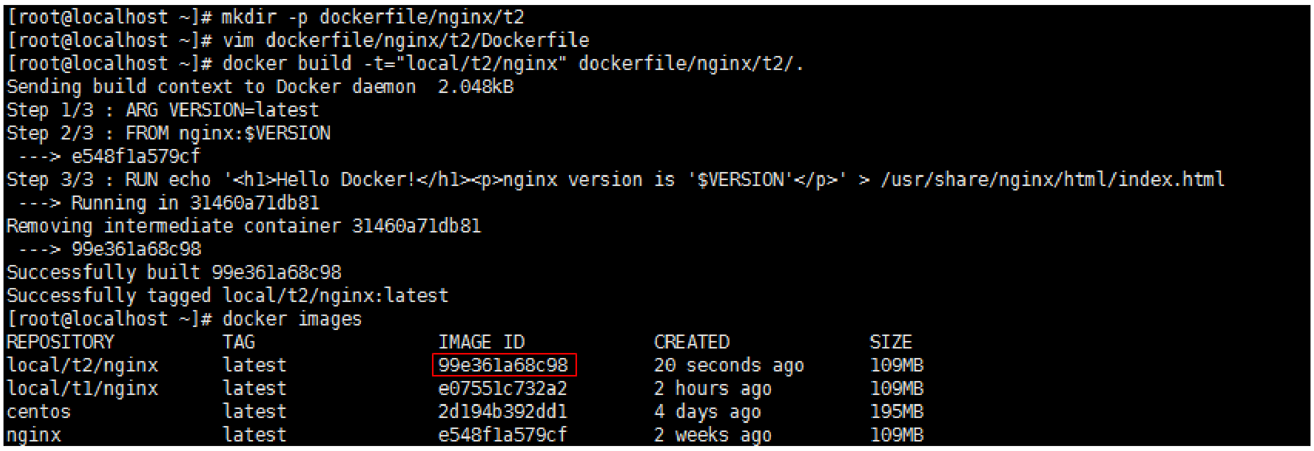
> 把变量和字符串放一起（没有空格），就实现了字符串的拼接（挺蛋疼的语法。眼神不好，没法弄）

1. 清理容器



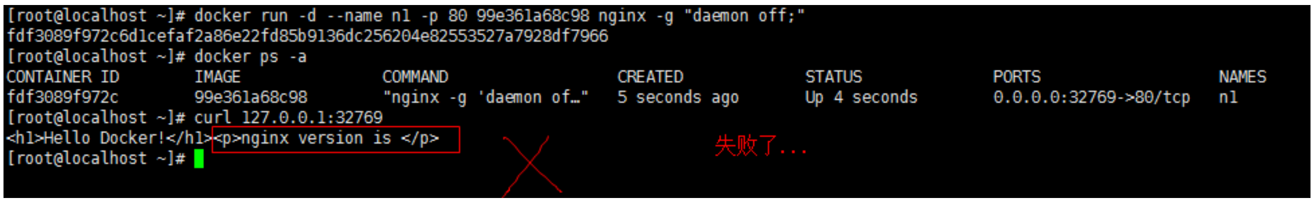
```
[root@localhost ~]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
local/t1/nginx      latest              e07551c732a2        2 hours ago         109MB
centos              latest              2d194b392dd1        4 days ago          195MB
nginx               latest              e548f1a579cf        2 weeks ago         109MB
[root@localhost ~]# docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                   NAMES
0ca2c2cb9483        e07551c732a2        "nginx -g 'daemon of…"   2 hours ago         Up 2 hours          0.0.0.0:32768->80/tcp   n1
[root@localhost ~]# docker stop 0ca2c2cb9483
0ca2c2cb9483
[root@localhost ~]# docker rm 0ca2c2cb9483
0ca2c2cb9483
[root@localhost ~]# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
```

2. 构建image



```
[root@localhost ~]# mkdir -p dockerfile/nginx/t2
[root@localhost ~]# vim dockerfile/nginx/t2/Dockerfile
[root@localhost ~]# docker build -t="local/t2/nginx" dockerfile/nginx/t2/.
Sending build context to Docker daemon  2.048kB
Step 1/3 : ARG VERSION=latest
Step 2/3 : FROM nginx:$VERSION
 ---> e548f1a579cf
Step 3/3 : RUN echo '<h1>Hello Docker!</h1><p>nginx version is '$VERSION'</p>' > /usr/share/nginx/html/index.html
 ---> Running in 31460a71db81
Removing intermediate container 31460a71db81
 ---> 99e361a68c98
Successfully built 99e361a68c98
Successfully tagged local/t2/nginx:latest
[root@localhost ~]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
local/t2/nginx      latest              99e361a68c98        20 seconds ago      109MB
local/t1/nginx      latest              e07551c732a2        2 hours ago         109MB
centos              latest              2d194b392dd1        4 days ago          195MB
nginx               latest              e548f1a579cf        2 weeks ago         109MB
```

3. 启动nginx & 查看首页



```
[root@localhost ~]# docker run -d --name n1 -p 80 99e361a68c98 nginx -g "daemon off;"
fdf3089f972c6d1cefaf2a86e22fd85b9136dc256204e82553527a7928df7966
[root@localhost ~]# docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                   NAMES
fdf3089f972c        99e361a68c98        "nginx -g 'daemon of…"   5 seconds ago       Up 4 seconds        0.0.0.0:32769->80/tcp   n1
[root@localhost ~]# curl 127.0.0.1:32769
<h1>Hello Docker!</h1><p>nginx version is </p>
[root@localhost ~]# 
```

4.2.2.2 示例2✓

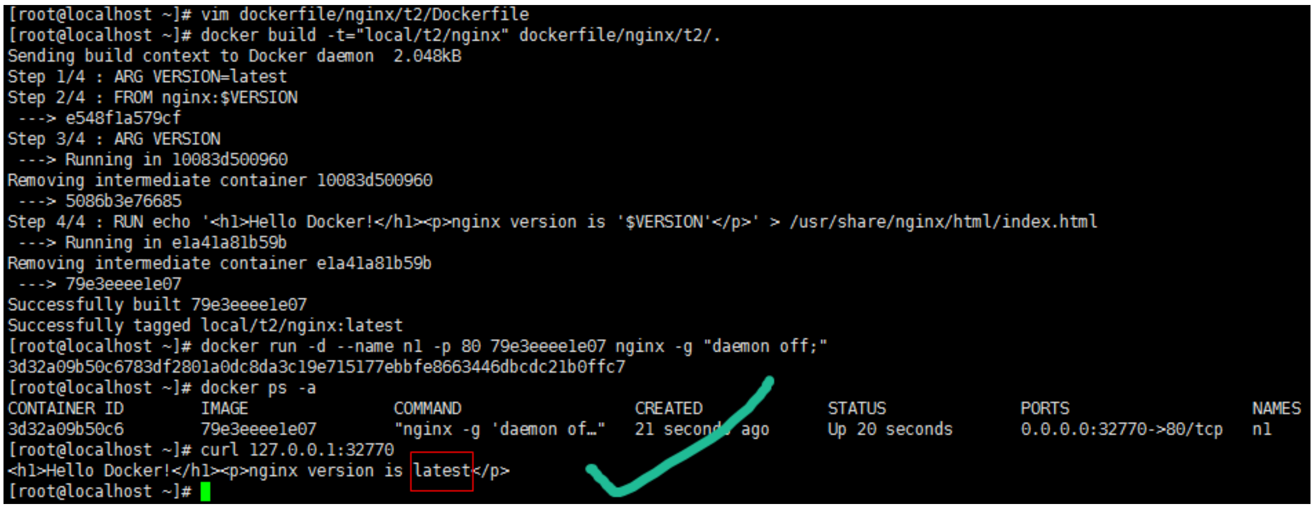
```
ARG VERSION=latest
FROM nginx:$VERSION
ARG VERSION
RUN echo '<h1>Hello Docker!</h1><p>nginx version is '$VERSION'</p>' >
/usr/share/nginx/html/index.html
```

> 在引用变量VERSION前，再声明一次

1. 清理容器&镜像

```
[root@localhost ~]# docker stop fdf3089f972c
fdf3089f972c
[root@localhost ~]# docker rm fdf3089f972c
fdf3089f972c
[root@localhost ~]# docker rmi 99e361a68c98
Untagged: local/t2/nginx:latest
Deleted: sha256:99e361a68c98818a0cfd6c69d321a417c00d1e0a8f6a203c4cfbc3b7209f89d1
Deleted: sha256:080ae324eed508724638bad4b810bf1d0496e5b2fc114db6ce38be29f7883602
[root@localhost ~]# docker images -a
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
local/t1/nginx      latest              e07551c732a2        2 hours ago         109MB
centos              latest              2d194b392dd1        4 days ago          195MB
nginx               latest              e548f1a579cf        2 weeks ago         109MB
[root@localhost ~]#
```

2. 重新构建、启动，略
3. 查看首页



```
[root@localhost ~]# vim dockerfile/nginx/t2/Dockerfile
[root@localhost ~]# docker build -t="local/t2/nginx" dockerfile/nginx/t2/.
Sending build context to Docker daemon  2.048kB
Step 1/4 : ARG VERSION=latest
Step 2/4 : FROM nginx:$VERSION
 ---> e548f1a579cf
Step 3/4 : ARG VERSION
 ---> Running in 10083d500960
Removing intermediate container 10083d500960
 ---> 5086b3e76685
Step 4/4 : RUN echo '<h1>Hello Docker!</h1><p>nginx version is '$VERSION'</p>' > /usr/share/nginx/html/index.html
 ---> Running in e1a41a81b59b
Removing intermediate container e1a41a81b59b
 ---> 79e3eeee1e07
Successfully built 79e3eeee1e07
Successfully tagged local/t2/nginx:latest
[root@localhost ~]# docker run -d --name n1 -p 80 79e3eeee1e07 nginx -g "daemon off;"
3d32a09b50c6783df2801a0dc8da3c19e715177ebbfe8663446dbcdc21b0ffc7
[root@localhost ~]# docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                   NAMES
3d32a09b50c6        79e3eeee1e07        "nginx -g 'daemon of…"   21 seconds ago      Up 20 seconds       0.0.0.0:32770->80/tcp   n1
[root@localhost ~]# curl 127.0.0.1:32770
<h1>Hello Docker!</h1><p>nginx version is latest</p>
[root@localhost ~]# 
```

4.2.2.3 示例3

引用变量的完整语法`${VERSION}`。测试，略

```
ARG VERSION=latest
FROM nginx:${VERSION}
ARG VERSION
RUN echo '<h1>Hello Docker!</h1><p>nginx version is '${VERSION}'</p>' >
/usr/share/nginx/html/index.html
```

4.2.4 FROM scratch

以scratch为基础镜像，意味着不以任何镜像为基础镜像。这么做，并不罕见。比如GO语言开发的应用，就会使用这种方式制作镜像，因为运行库已经在可执行文件里了，程序可以直接运行。所以，有人认为GO语言更适合做容器微服务架构的语言。

> 这段描述，我也是抄的。太高端了，不知道怎么弄...

4.2.5 其他

- FROM指令通常是第一个指令，指定“基础镜像”
  - ARG指令，可以放到FROM的前面
- 在一个Dockerfile中，可以指定多个“基础镜像”
  - 这种制作方式，被称为“多阶段镜像构建”。听名字，就知道很麻烦。先搁置
- tag和digest，都是可选的。如果省略，会使用latest
  - 如果找不到指定的tag或digest，构建会失败

4.3 RUN

RUN，可以执行linux的命令。

RUN指令的格式	描述
`RUN <command>`	就像在命令行敲的一样。默认的shell：`/bin/sh`
`RUN ["executable", "param1", "param2"]`	示例`RUN ["/bin/bash", "-c", "echo hello"]`

- RUN命令可以调用很多次
  - 每一次执行，都会创建一个镜像
  - 新创建的镜像，都基于前面的镜像。一层一层叠加起来
  - 每一个RUN命令，都运行在一个新的容器里
- 末尾的“\”是连接符。当命令太长时，用来换行

```
RUN /bin/bash -c 'source $HOME/.bashrc; \
echo $HOME'
```

- 构建过程中，“中间层镜像”会被缓存起来
  - 再次构建的时候，可以直接使用，加快构建的速度
  - 可以像代码一样，调试构建过程
- 镜像是多层存储的结构，且，只有“当前层”是可写的，底层的镜像都是只读的
  - 构建过程中，下载的压缩包、编译环境（源码安装）等，都是运行时不需要的。所以，每构建一层，要清理之后，再构建下一层
  - 代码尽量干净点，能减少出错的概率，也更容易发现问题

5 COPY & LABEL

- `COPY` ：将local host的文件，拷贝到容器里。

命令格式	描述
`COPY [--chown=<user>:<group>] <src>... <dest>`	可以copy文件或目录
`COPY [--chown=<user>:<group>] ["<src>",... "<dest>"]`	双引号括起来的路径，允许包含空格

- `LABEL` ： 给镜像添加metadata（相当于注释）

1. 命令格式

```
LABEL <key>=<value> <key>=<value> <key>=<value> ...
```

2. 示例

```
LABEL multi.label1="value1" multi.label2="value2" other="value3"
LABEL multi.label1="value1" \
      multi.label2="value2" \
      other="value3"
```

~~5.1 ADD~~

`ADD` 指令，可以自动解压归档文件（压缩文件）。

其他功能跟 `COPY` 指令相同。有点鸡肋，不建议使用。

5.2 示例：copy local host's index.html

> 先清理之前的文件、目录、镜像、容器等

5.2.1 local host's index.html

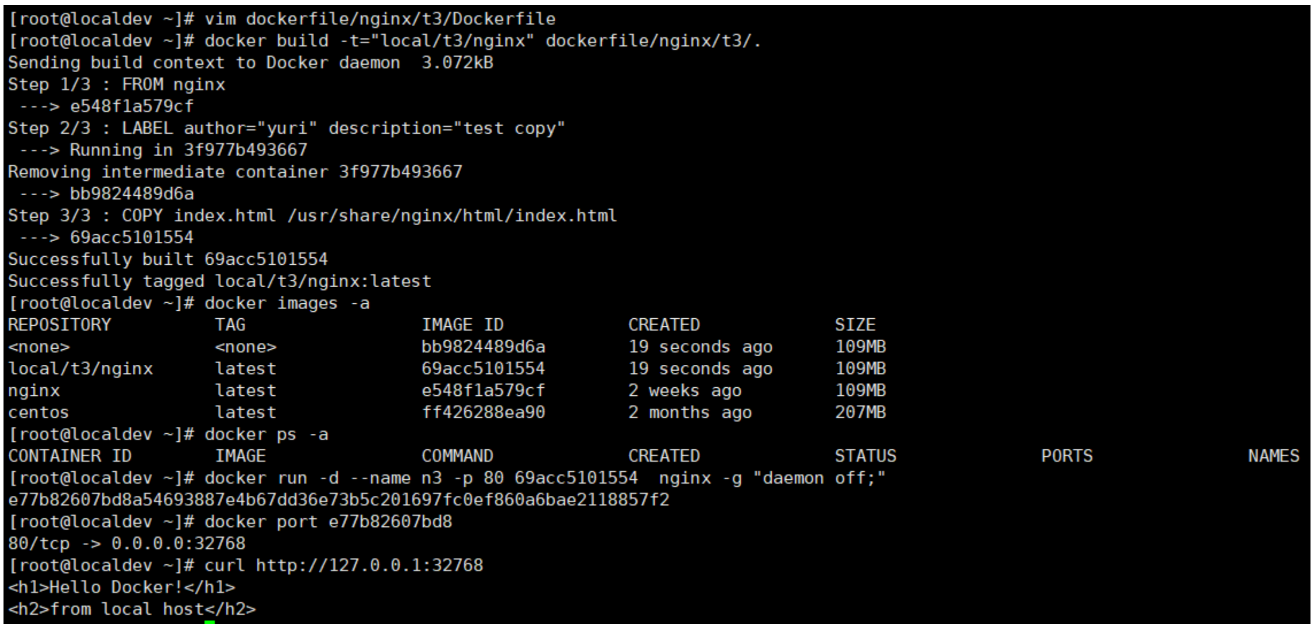
```
[root@localdev ~]# mkdir -p dockerfile/nginx/t3
[root@localdev ~]# vim dockerfile/nginx/t3/index.html
```

```
<h1>Hello Docker!</h1>
<h2>from local host</h2>
```

5.2.2 Dockerfile预览

```
FROM nginx
LABEL author="yuri" description="test copy"
COPY index.html /usr/share/nginx/html/index.html
```

5.2.3 构建过程 & 测试

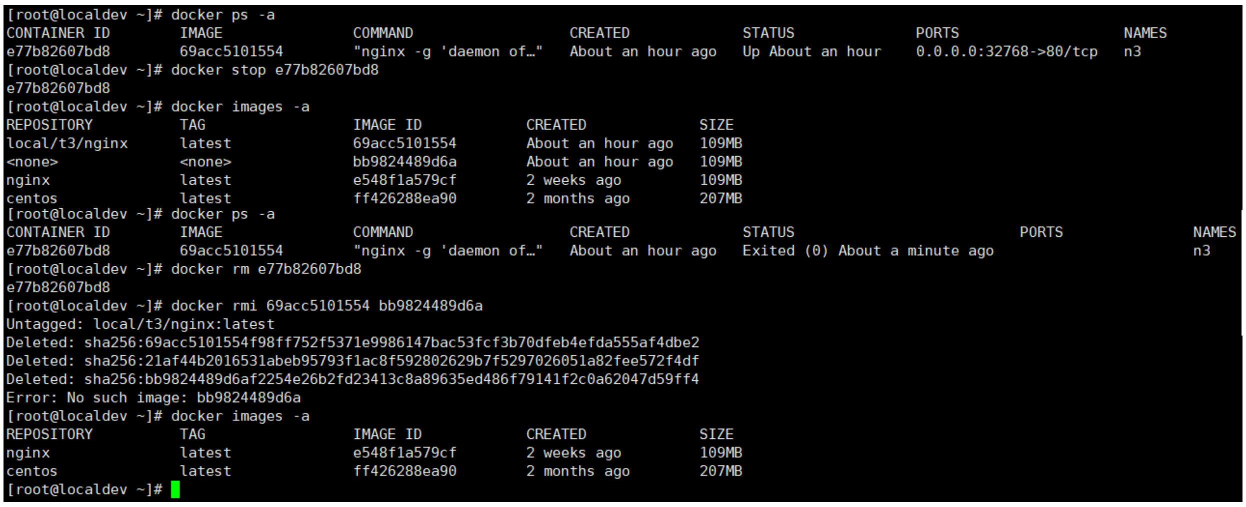


```
[root@localdev ~]# vim dockerfile/nginx/t3/Dockerfile
[root@localdev ~]# docker build -t="local/t3/nginx" dockerfile/nginx/t3/.
Sending build context to Docker daemon  3.072kB
Step 1/3 : FROM nginx
 ---> e548f1a579cf
Step 2/3 : LABEL author="yuri" description="test copy"
 ---> Running in 3f977b493667
Removing intermediate container 3f977b493667
 ---> bb9824489d6a
Step 3/3 : COPY index.html /usr/share/nginx/html/index.html
 ---> 69acc5101554
Successfully built 69acc5101554
Successfully tagged local/t3/nginx:latest
[root@localdev ~]# docker images -a
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
<none>              <none>              bb9824489d6a        19 seconds ago      109MB
local/t3/nginx      latest              69acc5101554        19 seconds ago      109MB
nginx               latest              e548f1a579cf        2 weeks ago         109MB
centos              latest              ff426288ea90        2 months ago        207MB
[root@localdev ~]# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
[root@localdev ~]# docker run -d --name n3 -p 80 69acc5101554  nginx -g "daemon off;"
e77b82607bd8a54693887e4b67dd36e73b5c201697fc0ef860a6bae2118857f2
[root@localdev ~]# docker port e77b82607bd8
80/tcp -> 0.0.0.0:32768
[root@localdev ~]# curl http://127.0.0.1:32768
<h1>Hello Docker!</h1>
<h2>from local host</h2>
```

5.2.4 查看label

```
[root@localdev ~]# docker inspect -f "{{index .Config.Labels}}" e77b82607bd8
map[author:yuri description:test copy maintainer:NGINX Docker Maintainers <docker-
maint@nginx.com>]
```

6 CMD & EXPOSE & ENTRYPOINT

> 清理容器、镜像
>
> 

```
[root@localdev ~]# docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                   NAMES
e77b82607bd8        69acc5101554        "nginx -g 'daemon of…"   About an hour ago   Up About an hour    0.0.0.0:32768->80/tcp   n3
[root@localdev ~]# docker stop e77b82607bd8
e77b82607bd8
[root@localdev ~]# docker images -a
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
local/t3/nginx      latest              69acc5101554        About an hour ago   109MB
<none>              <none>              bb9824489d6a        About an hour ago   109MB
nginx               latest              e548f1a579cf        2 weeks ago         109MB
centos              latest              ff426288ea90        2 months ago        207MB
[root@localdev ~]# docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS                     PORTS               NAMES
e77b82607bd8        69acc5101554        "nginx -g 'daemon of…"   About an hour ago   Exited (0) About a minute ago                  n3
[root@localdev ~]# docker rm e77b82607bd8
e77b82607bd8
[root@localdev ~]# docker rmi 69acc5101554 bb9824489d6a
Untagged: local/t3/nginx:latest
Deleted: sha256:69acc5101554f98ff752f5371e9986147bac53fcf3b70dfeb4efda555af4dbe2
Deleted: sha256:21af44b2016531abeb95793f1ac8f592802629b7f5297026051a82fee572f4df
Deleted: sha256:bb9824489d6af2254e26b2fd23413c8a89635ed486f79141f2c0a62047d59ff4
Error: No such image: bb9824489d6a
[root@localdev ~]# docker images -a
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
nginx               latest              e548f1a579cf        2 weeks ago         109MB
centos              latest              ff426288ea90        2 months ago        207MB
[root@localdev ~]#
```

- `CMD` ：容器启动时的默认命令

CMD指令格式
`CMD ["executable","param1","param2"]`
`CMD ["param1","param2"]`
`CMD command param1 param2`

- `EXPOSE` ：暴漏容器的端口，允许外界访问容器。默认，容器不能被外界访问
- `ENTRYPOINT` ：与 `CMD` 类似
- 当 `ENTRYPOINT` 和 `CMD` 同时存在时，实际执行的命令： `ENTRYPOINT CMD` ，即，CMD是ENTRYPOINT的参数

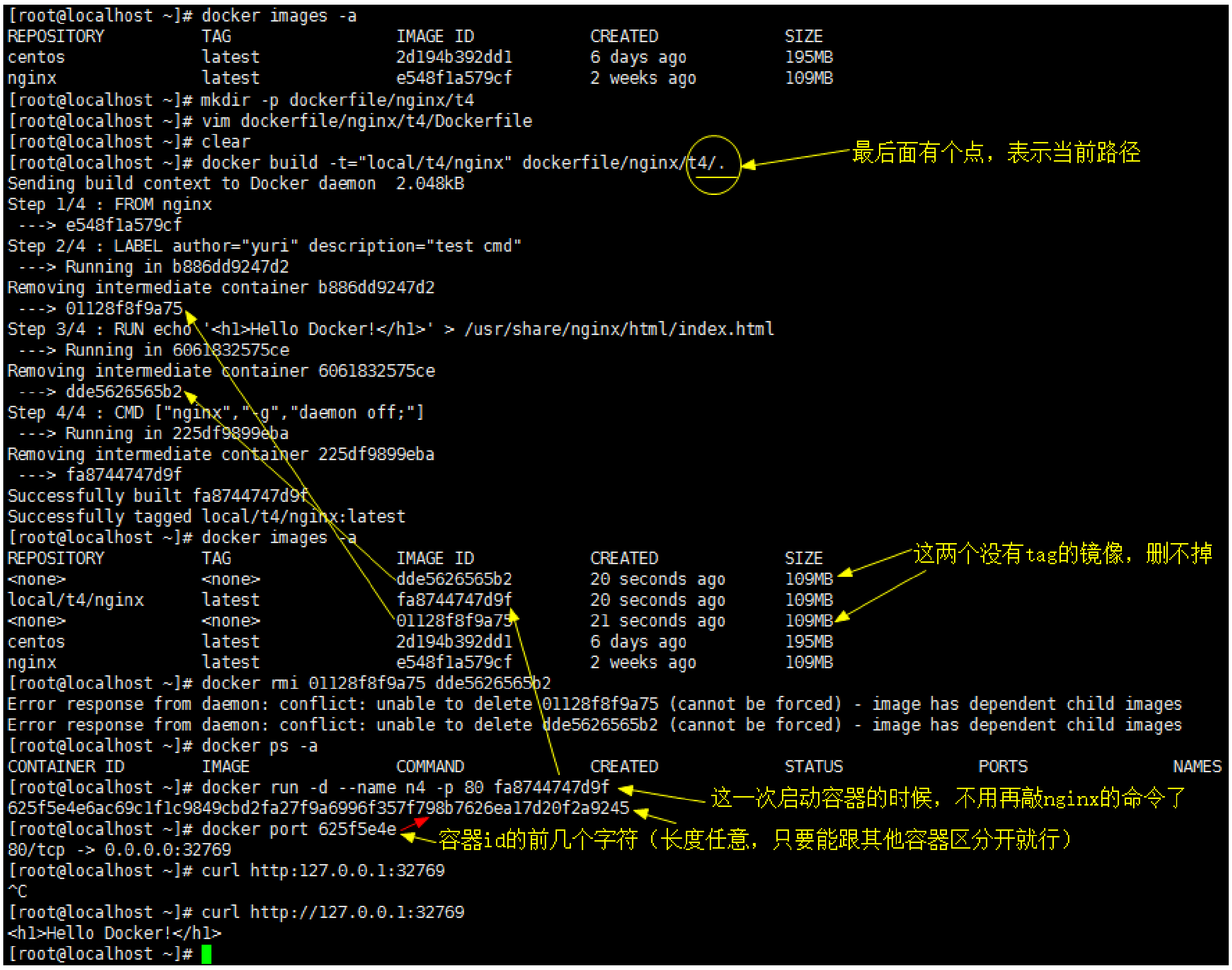
6.1 设置nginx容器的默认命令

	启动容器的命令
转换前	`docker run -d --name n1 -p 80 e07551c732a2 nginx -g "daemon off;"`
转换后	`docker run -d --name n1 -p 80 e07551c732a2`

1. Dockerfile概览

```
FROM nginx
LABEL author="yuri" description="test cmd"
RUN echo '<h1>Hello Docker!</h1>' > /usr/share/nginx/html/index.html
CMD ["nginx","-g","daemon off;"]
```

2. 构建&测试



```
[root@localhost ~]# docker images -a
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
centos              latest              2d194b392dd1        6 days ago          195MB
nginx               latest              e548f1a579cf        2 weeks ago         109MB
[root@localhost ~]# mkdir -p dockerfile/nginx/t4
[root@localhost ~]# vim dockerfile/nginx/t4/Dockerfile
[root@localhost ~]# clear
[root@localhost ~]# docker build -t="local/t4/nginx" dockerfile/nginx/t4/.
Sending build context to Docker daemon  2.048kB
Step 1/4 : FROM nginx
 ---> e548f1a579cf
Step 2/4 : LABEL author="yuri" description="test cmd"
 ---> Running in b886dd9247d2
Removing intermediate container b886dd9247d2
 ---> 01128f8f9a75
Step 3/4 : RUN echo '<h1>Hello Docker!</h1>' > /usr/share/nginx/html/index.html
 ---> Running in 6061832575ce
Removing intermediate container 6061832575ce
 ---> dde5626565b2
Step 4/4 : CMD ["nginx","-g","daemon off;"]
 ---> Running in 225df9899eba
Removing intermediate container 225df9899eba
 ---> fa8744747d9f
Successfully built fa8744747d9f
Successfully tagged local/t4/nginx:latest
[root@localhost ~]# docker images -a
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
<none>              <none>              dde5626565b2        20 seconds ago      109MB
local/t4/nginx      latest              fa8744747d9f        20 seconds ago      109MB
<none>              <none>              01128f8f9a75        21 seconds ago      109MB
centos              latest              2d194b392dd1        6 days ago          195MB
nginx               latest              e548f1a579cf        2 weeks ago         109MB
[root@localhost ~]# docker rmi 01128f8f9a75 dde5626565b2
Error response from daemon: conflict: unable to delete 01128f8f9a75 (cannot be forced) - image has dependent child images
Error response from daemon: conflict: unable to delete dde5626565b2 (cannot be forced) - image has dependent child images
[root@localhost ~]# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
[root@localhost ~]# docker run -d --name n4 -p 80 fa8744747d9f
625f5e4e6ac69c1f1c9849cbd2fa27f9a6996f357f798b7626ea17d20f2a9245
[root@localhost ~]# docker port 625f5e4e
80/tcp -> 0.0.0.0:32769
[root@localhost ~]# curl http:127.0.0.1:32769
^C
[root@localhost ~]# curl http://127.0.0.1:32769
<h1>Hello Docker!</h1>
[root@localhost ~]#
```

最后面有个点，表示当前路径
这两个没有tag的镜像，删不掉
这一次启动容器的时候，不用再敲nginx的命令了
容器id的前几个字符（长度任意，只要能跟其他容器区分开就行）

6.2 怎么删除中间镜像

1. `docker  build`输出的内容

```
[root@localhost ~]# docker build -t="local/t4/nginx" dockerfile/nginx/t4/.
Sending build context to Docker daemon  2.048kB
Step 1/4 : FROM nginx
 ---> e548f1a579cf
Step 2/4 : LABEL author="yuri" description="test cmd"
 ---> Running in b886dd9247d2
Removing intermediate container b886dd9247d2
 ---> 01128f8f9a75
Step 3/4 : RUN echo '<h1>Hello Docker!</h1>' > /usr/share/nginx/html/index.html
 ---> Running in 6061832575ce
Removing intermediate container 6061832575ce
 ---> dde5626565b2
```

```
Step 4/4 : CMD ["nginx","-g","daemon off;"]
 ---> Running in 225df9899eba
Removing intermediate container 225df9899eba
 ---> fa8744747d9f
Successfully built fa8744747d9f
Successfully tagged local/t4/nginx:latest
[root@localhost ~]# docker images -a
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
<none>              <none>              dde5626565b2        20 seconds ago      109MB
local/t4/nginx      latest              fa8744747d9f        20 seconds ago      109MB
<none>              <none>              01128f8f9a75        21 seconds ago      109MB
centos              latest              2d194b392dd1        6 days ago          195MB
nginx               latest              e548f1a579cf        2 weeks ago         109MB
```

2. 查看是否“虚悬镜像”

```
[root@localhost ~]# docker images -f "dangling=true" -q
[root@localhost ~]# docker images --help

Usage:  docker images [OPTIONS] [REPOSITORY[:TAG]]

List images

Options:
  -a, --all             Show all images (default hides intermediate images)
      --digests         Show digests
  -f, --filter filter   Filter output based on conditions provided
      --format string   Pretty-print images using a Go template
      --no-trunc        Don't truncate output
  -q, --quiet           Only show numeric IDs
[root@localhost ~]#
```

3. 不是“虚悬镜像”，所以删不掉
4. 查到的资料：停掉容器，再删除local/t4/nginx，就会把这些镜像都删掉。全都删掉了...有毛用？
5. 或者，重启下local host再试试？很遗憾，还是删不掉。

先留个疑问吧，暂时搞不定...

6.3 去掉“-p 80”

> 只显示重要的步骤

1. Dockerfile

```
FROM nginx
LABEL author="yuri" description="test CMD & EXPOSE"
RUN echo '<h1>Hello Docker!</h1>' > /usr/share/nginx/html/index.html
CMD ["nginx","-g","daemon off;"]
EXPOSE 80
```

2. 启动容器

```
[root@localhost ~]# docker run -d --name n5 -P d4fd3e4788f9
```

> `-P` ,发布exposed port

7 `nginx -g "daemon off;"`

- 在local host中敲的docker指令，都是在主进程中执行的命令。命令结束后，进程也就结束了
- `docker run` 命令结束后，进程也就结束了。所以，容器，也就关闭了
- 为了让容器继续运行，需要执行命令 `docker run -d`：在后台执行容器，且打印容器id
- nginx默认启动是在daemon模式下，即，跟主进程绑定了
- 为了启动nginx服务，还需要加“daemon off"指令

8 其他

8.1 `ENV` & `WORKDIR` & `USER`

> 文章有点长了，后面的命令，就不详细解释啦。需要的话，查官网的文档吧。相信，做了这么多练习，查文档的时候，也不会犯怵了吧

- ENV：设置环境变量。 `ENV <key>=<value>`

```
ENV JAVA_VERSION="1.8.0_162"
```

- WORKDIR：设置工作目录。练习一直使用的nginx默认的目录，如果需要在镜像制作的过程中，下载、安装，需要自己指定工作目录

```
ENV ITBILU_PATH /home/itbilu/
WORKERDIR $ITBILU_PATH
```

- USER：指定运行镜像所使用的用户

8.2 VOLUME

容器重启后，数据会丢失。所以，为了保存数据，需要指定VOLUME。

VOLUME跟当前容器没有关系，所以，写入VOLUME的数据，就不会丢失了。

多个容器，可以共用一个VOLUME。

于是，镜像、容器、VOLUME之间就有了复杂的依赖关系

8.3 ONBUILD

作为“基础镜像”， `ONBUILD` 指令没什么用。

但是，基于这个“基础镜像”制作新的镜像时，会执行 `ONBUILD` 指令。

比如：当底层镜像变化后，上层镜像重新构建的时候，执行 `ONBUILD` 指令，就可以实现自动升级。不需要再一点点地改了。

于是，指令会形成一条线，有可能会冲突。

8.4 HEALTHCHECK

判断容器的状态。

如果程序卡死了，那么，这个容器也就不能提供服务了。这时候，容器得告知docker server，自己已经挂了。

8.5 STOPSIGNAL

默认的stop-signal是SIGTERM，在docker stop的时候会给容器内PID为1的进程发送这个signal，通过--stop-signal可以设置自己需要的signal，主要的目的是为了让容器内的应用程序在接收到signal之后可以先做一些事情，实现容器的平滑退出，如果不做任何处理，容器将在一段时间之后强制退出，会造成业务的强制中断，这个时间默认是10s

8.6 SHELL

默认的shell是`["bin/sh", "-c"]`。通过`SHELL`指令修改默认的shell，比如：

1. 在DOS（windows）中

```
SHELL ["powershell", "-command"]
```

2. 或者，在Linux中使用`/bin/bash`

```
SHELL ["/bin/bash", "-c"]
```

9 总结

Dockfile可以通过文档的形式，描述构建过程。

在微服务领域，每个项目都依赖于特定的环境，在开发的过程中，维护Dockerfile，那么，部署时，就可以直接发布“容器”。

不会再出现“在我的电脑上运行正常，换台电脑就…”