docker learning

> date: 2024.6.13
> author: zilin xu
> resource: docker从入门到实践

basic commands

`docker pull ubuntu:18.04`

pull a image from `Dockerhub`

`docker run -it --rm ubuntu:18.04 bash`

`exit`

run a `ubuntu` container and start a bash window

```
➜  dockerLearning git:(main) ✗ docker image ls
REPOSITORY   TAG       IMAGE ID       CREATED         SIZE
<none>       <none>    a072820a1c8a   7 days ago      790MB
ubuntu       18.04     d1a528908992   12 months ago   56.7MB
```

none means 悬虚镜像

list download image

> note: 这个体积显示的比dockerhub上大得多，因为hub上面显示的是压缩后的体积，关心的是传输过程中流量的大小。而`docker image ls` 显示的是image下载到本地后，展开各层所占的空间的总和。

```
➜  dockerLearning git:(main) ✗ docker system df
TYPE            TOTAL     ACTIVE    SIZE      RECLAIMABLE
Images          2         1         846.3MB   56.66MB (6%)
Containers      1         0         0B        0B
Local Volumes   0         0         0B        0B
Build Cache     127       0         19.04GB   19.04GB
```

show the total occupied space from `image` and `container`

```
➜  dockerLearning git:(main) ✗ docker image ls
REPOSITORY   TAG       IMAGE ID       CREATED         SIZE
<none>       <none>    a072820a1c8a   7 days ago      790MB
redis        latest    f1b6143a06ea   3 weeks ago     139MB
ubuntu       18.04     d1a528908992   12 months ago   56.7MB
➜  dockerLearning git:(main) ✗ docker image rm f1b
```

```
Untagged: redis:latest
Untagged:
redis@sha256:01afb31d6d633451d84475ff3eb95f8c48bf0ee59ec9c948b161adb4da882
053
Deleted:
sha256:f1b6143a06eab173913b51fdf584742824b3fcac39436fbd3ce69cbd134038f1
Deleted:
sha256:25f7d365a5f1bb4a5d2091b5649591574a4ec648415f83c1b98eba7010dc38bc
Deleted:
sha256:8be7b689ba25d539ec587b7c79eaca185ffd384a7f8138488d09056e30719233
Deleted:
sha256:c05ef20cd5afce8ac085fdc7f0bcd0e396f26c6998981c51a5b75afa009b67c8
Deleted:
sha256:8fc92a83f52aee281b295f3c8919de7cc10ea1301c20edd3a474e924502b26d8
Deleted:
sha256:1a0295108d268ef47de60ade14623c737fe704ae0af6fee0f135271b155011da
Deleted:
sha256:24a967af9c4cfafc72bbc45739949aaa1279c2ac433f52bac0638282278d02cf
Deleted:
sha256:30b0bb19cb8305c38a6fc62ef28ec10f481e57afae84b924c49bb73922476297
Deleted:
sha256:2bd1a2222589b50b52ff960c3d004829633df61532e7a670a91618cd775f2d47
```

remove image

`docker run --name webserver -d -p 80:80 nginx`

start a container named `webserver` using port `80`

then you should view browser `localhost`

`docker stop webserver`

using Dockerfile

docker commit vs Dockerfile

通俗理解来说，就像教docker做菜一样。如果你一步一步做菜，这是`docker commit`。但问题是中间的步骤容易出错，很麻烦也。Dockerfile就像一个食谱，规定好每次的步骤。

```
mkdir mynginx
cd mynginx
touch Dockerfile
```

edit Dockerfile

```
FROM nginx

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

run `docker build -t nginx:v3 .`

```
➜  mynginx git:(main) ✗ docker image ls -a
REPOSITORY   TAG       IMAGE ID       CREATED         SIZE
<none>       <none>    a072820a1c8a   7 days ago      790MB
nginx        latest    705b7f60fea5   2 weeks ago     193MB
ubuntu       18.04     d1a528908992   12 months ago   56.7MB
➜  mynginx git:(main) ✗ docker build -t nginx:v3 .
[+] Building 0.2s (6/6) FINISHED
docker:desktop-linux
 => [internal] load build definition from Dockerfile
0.0s
 => => transferring dockerfile: 156B
0.0s
 => [internal] load metadata for docker.io/library/nginx:latest
0.0s
 => [internal] load .dockerignore
0.0s
 => => transferring context: 2B
0.0s
 => [1/2] FROM docker.io/library/nginx:latest
0.0s
 => [2/2] RUN echo '<h1>Hello, docker!</h1>' >
/usr/share/nginx/html/index.html  0.1s
 => exporting to image
0.0s
 => => exporting layers
0.0s
 => => writing image
sha256:e0943d61d246ad8af098cf12dc3985fad1fa2c68775a5f1fbe8  0.0s
 => => naming to docker.io/library/nginx:v3
0.0s

View build details: docker-desktop://dashboard/build/desktop-
linux/desktop-linux/eex2gwb3misf6nw7iox3dql23

What's Next?
  View a summary of image vulnerabilities and recommendations → docker
scout quickview
➜  mynginx git:(main) ✗ docker image ls -a
REPOSITORY   TAG       IMAGE ID       CREATED          SIZE
nginx        v3        e0943d61d246   12 seconds ago   193MB
<none>       <none>    a072820a1c8a   7 days ago       790MB
nginx        latest    705b7f60fea5   2 weeks ago      193MB
ubuntu       18.04     d1a528908992   12 months ago    56.7MB
```

run following command, you shold see `hello docker` on `localhost`

```
➜  mynginx git:(main) ✗ docker run --name newWebserver -d -p 80:80
nginx:v3
d7e669d657d4b4a4801ae838069d29368e4f07e9df549f68e6f4b6bcc57bd243
```

some problem with the definition `context`

没错，你的理解是正确的。

上下文目录（Context Directory）

上下文目录是指你在本地指定的目录，它包含了 Docker 构建镜像所需的所有文件。上下文目录中的内容会被打包并发送给 Docker 引擎，用来构建镜像。

例如，在运行 `docker build -t myimage .` 时，这里的 `.` 表示当前目录是上下文目录。Docker 会将当前目录下的所有文件打包发送给 Docker 引擎。

工作目录（Working Directory）

工作目录是指在镜像内的目录，它是用 `WORKDIR` 指令在 Dockerfile 中设置的。在构建镜像过程中，所有相对路径操作都会基于这个工作目录。

例如：

```
FROM golang:1.18

# 设置工作目录为 /app
WORKDIR /app

# 将上下文目录中的 app 文件夹内容复制到镜像中的 /app 目录
COPY ./app /app

# 在 /app 目录中运行 go build -o main .
RUN go build -o main .

# 设置容器启动时的命令
CMD ["./main"]
```

在这个例子中：

- **上下文目录** 是本地的当前目录（由 `docker build` 命令中的 `.` 指定）。
- **工作目录** 是镜像内的 `/app` 目录（由 `WORKDIR /app` 指定）。

具体区分

1. **上下文目录（本地的）**：
    - 由 `docker build` 命令的参数指定，例如 `.` 表示当前目录。
    - Docker 会将这个目录下的所有内容打包并发送给 Docker 引擎。
    - 例如，`docker build -t myimage .` 会将当前目录作为上下文目录。

2. **工作目录（镜像内的）**：
    - 由 `WORKDIR` 指令在 Dockerfile 中指定。
    - 后续所有相对路径操作都基于这个目录。
    - 例如，`WORKDIR /app` 设置工作目录为 `/app`，之后的所有指令中的相对路径都是相对于 `/app` 目录的。

图示

假设本地目录结构如下：

```
my-project/
├── Dockerfile
├── app/
│   └── main.go
└── package.json
```

运行 `docker build -t mygoapp .` 时：

- **上下文目录** 是 `my-project` 目录。
- Docker 会将 `my-project` 目录下的所有内容打包并发送给 Docker 引擎。

在 Dockerfile 中：

```
FROM golang:1.18
WORKDIR /app
COPY ./app /app
RUN go build -o main .
CMD ["./main"]
```

- `WORKDIR /app` 将镜像内的工作目录设置为 `/app`。
- `COPY ./app /app` 将上下文目录中的 `app` 目录复制到镜像内的 `/app` 目录。

希望这个解释能清晰地区分上下文目录和工作目录的概念。如果有其他问题，随时告诉我！