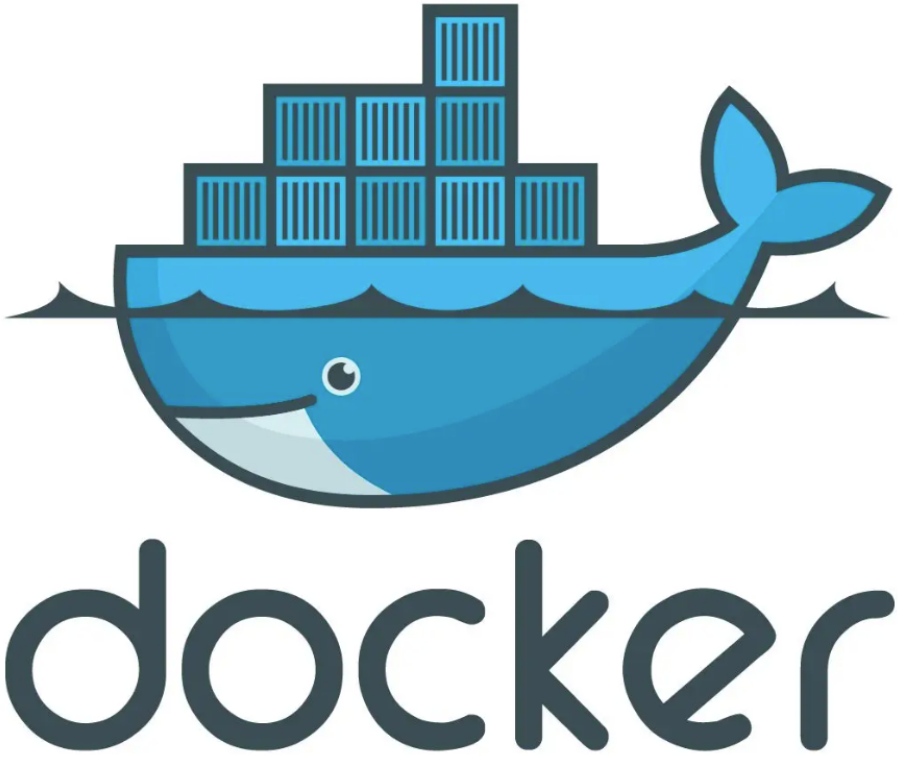
1 初识Docker

1.1 什么是Docker

Docker 是一个开源的应用容器引擎，让开发者可以打包他们的应用以及依赖包到一个可移植的镜像(images)中，然后发布到任何流行的 Linux或Windows操作系统的机器上，也可以实现虚拟化。

容器(container)是完全使用沙箱(sandbox)机制，相互之间不会有任何接口。



1.2 镜像和容器

Docker中有两个重要的概念：

镜像（Image）：Docker将应用程序及其所需的依赖、函数库、环境、配置等文件打包在一起，称为镜像。

容器（Container）：镜像中的应用程序运行后形成的进程就是**容器**，只是Docker会给容器进程做隔离，对外不可见。

例如你下载了一个QQ，如果我们将QQ在磁盘上的运行文件及其运行的操作系统依赖打包，形成QQ**镜像**。然后你可以启动一次，双开、三开QQ，每次开启的QQ都是一个**容器**。

1.3 DockerHub

开源应用程序非常多，打包这些应用往往是重复的劳动。为了避免这些重复劳动，人们就会将自己打包的应用镜像，例如Redis、MySQL镜像放到网络上，共享使用，就像GitHub的代码共享一样。

- Docker镜像地址：https://hub.docker.com

提示：由于Docker镜像仓库的DNS被污染，导致Docker镜像仓库无法访问。使用国内镜像加速可解决无法访问问题。阿里云官网提供了镜像加速，网址如下：

https://help.aliyun.com/zh/acr/user-guide/accelerate-the-pulls-of-docker-official-images?spm=a2c4g.11186623.0.0.2788296ajBO7OA

2 镜像操作

2.1 镜像名称

首先来看下镜像的名称组成：

- 镜像名称一般分两部分组成：[repository]:[tag]。
- 在没有指定tag时，默认是latest，代表最新版本的镜像

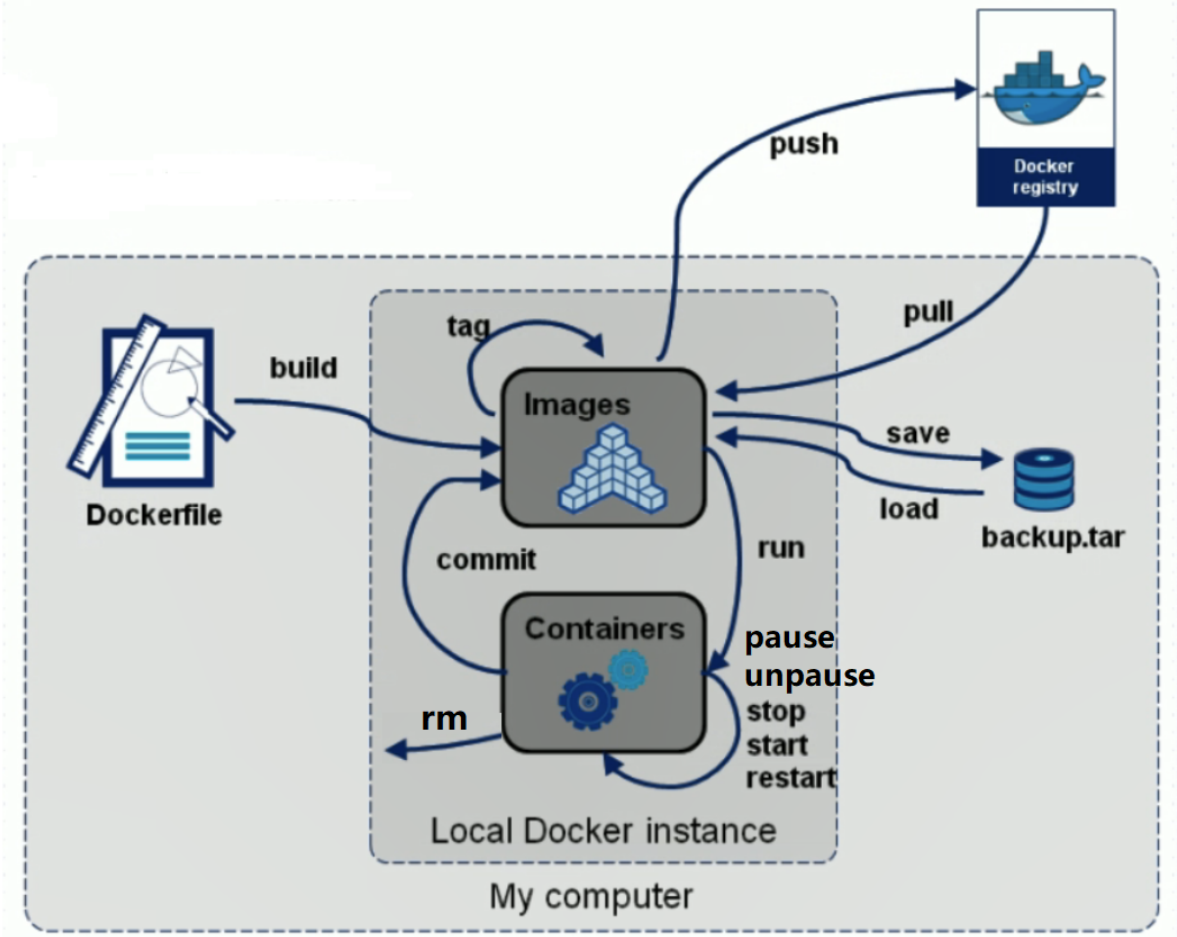
例如：

```
mysql:5.7
```

这里的mysql就是repository，5.7就是tag，合一起就是镜像名称，代表5.7版本的MySQL镜像。

2.2 镜像命令

常见的镜像操作命令如图：



常用的镜像命令

docker images 查看镜像

docker rmi 删除镜像

docker push 推送镜像到服务器

docker pull 从服务器拉取镜像

docker save 保存镜像为压缩包

docker load 加载压缩包为镜像

2.3 拉取、查看镜像

需求：从DockerHub中拉取一个nginx镜像并查看

1）首先查看docker中已经安装的镜像

```
docker images
```

```
root@iZbp11nqxfd0irwosh7s1oZ:~# docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
hello-world   latest    9c7a54a9a43c   2 months ago   13.3kB
```

2）根据查看到的镜像名称，拉取自己需要的镜像

```
docker pull nginx
```

```
root@iZbp11nqxfd0irwosh7s1oZ:~# docker pull nginx
Using default tag: latest
latest: Pulling from library/nginx
faef57eae888: Pull complete
76579e9ed380: Pull complete
cf707e233955: Pull complete
91bb7937700d: Pull complete
4b962717ba55: Pull complete
f46d7b05649a: Pull complete
103501419a0a: Pull complete
Digest: sha256:08bc36ad52474e528cc1ea3426b5e3f4bad8a130318e3140d6cfe29c8892c7ef
Status: Downloaded newer image for nginx:latest
docker.io/library/nginx:latest
```

3）通过命令：docker images 查看拉取到的镜像

```
docker images
```

```
root@iZbp11nqxfd0irwosh7s1oZ:~# docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
nginx         latest    021283c8eb95   2 weeks ago    187MB
hello-world   latest    9c7a54a9a43c   2 months ago   13.3kB
```

2.4 保存、导入镜像

需求：利用docker save将nginx镜像导出磁盘，然后再通过load加载回来

2.4.1 查看命令语法结构

利用docker xx --help命令查看docker save和docker load的语法

例如，查看save命令用法，可以输入命令：

```
docker save --help
```

结果：

```
root@iZbp11nqxfd0irwosh7s1oZ:~# docker save --help

Usage:  docker save [OPTIONS] IMAGE [IMAGE...]

Save one or more images to a tar archive (streamed to STDOUT by default)

Aliases:
  docker image save, docker save

Options:
  -o, --output string   Write to a file, instead of STDOUT
```

命令格式：

```
docker save -o [保存的目标文件名称] [镜像名称]
```

2.4.2 使用docker save导出镜像到磁盘

运行命令：

```
docker save -o nginx.tar nginx:latest
```

结果如图：

```
root@iZbp11nqxfd0irwosh7s1oZ:~# docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
nginx         latest    0d1273c8eb95   2 weeks ago    187MB
hello-world   latest    9c7a54a9a43c   2 months ago   13.3kB
root@iZbp11nqxfd0irwosh7s1oZ:~# docker save -o nginx.tar nginx:latest
root@iZbp11nqxfd0irwosh7s1oZ:~# ls
install.sh  nginx.tar  snap
```

2.4.3 使用docker load加载镜像

先删除本地的nginx镜像：

```
docker rmi nginx:latest
```

```
root@iZbp11nqxfd0irwosh7s1oZ:~# docker rmi nginx:latest
Untagged: nginx:latest
Untagged: nginx@sha256:08bc36ad52474e528cc1ea3426b5e3f4bad8a130318e3140d6cfe29c8892c7ef
Deleted: sha256:021283c8eb95be02b23db0de7f609d603553c6714785e7a673c6594a624ffbda
Deleted: sha256:a9de33035096cdf7bbaf7f3e1291701c0620d2a0e66152228abef35a79002876
Deleted: sha256:d66c35807d98c6f37bd2a14c6506a42d27a40fbdb564e233f7a78aafdc636c59
Deleted: sha256:a4c423818ed6dc12a545c349d0dc36a5695446448e07229e96c7235a126c2520
Deleted: sha256:c04094edc9df98c870e281f3b947a7782ca6d542d8715814ac06786466af3659
Deleted: sha256:c9c467815e8fe87d99f0f500495cf7f4f9096cf6c116ef2782e84bb17a4a5e06
Deleted: sha256:4645f26713fbea51190f5de52b88fbe27b42efd61c0dba87c81fa16df9a8f649
Deleted: sha256:24839d45ca455f36659219281e0f2304520b92347eb536ad5cc7b4dbb8163588
root@iZbp11nqxfd0irwosh7s1oZ:~# docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
hello-world   latest    9c7a54a9a43c   2 months ago   13.3kB
```

然后运行命令，加载本地文件：

```
docker load -i nginx.tar
```

结果：

```
root@iZbp11nqxfd0irwosh7s1oZ:~# docker load -i /root/nginx.tar
24839d45ca45: Loading layer [==================================================>]  77.81MB/77.81MB
b821d93f6666: Loading layer [==================================================>]  113.2MB/113.2MB
1998c5cd2230: Loading layer [==================================================>]  3.584kB/3.584kB
f36897eea34d: Loading layer [==================================================>]  4.608kB/4.608kB
9fdfd12bc85b: Loading layer [==================================================>]   2.56kB/2.56kB
434c6a715c30: Loading layer [==================================================>]   5.12kB/5.12kB
3c9d04c9ebd5: Loading layer [==================================================>]  7.168kB/7.168kB
Loaded image: nginx:latest
root@iZbp11nqxfd0irwosh7s1oZ:~# docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
nginx         latest    021283c8eb95   2 weeks ago    187MB
hello-world   latest    9c7a54a9a43c   2 months ago   13.3kB
```

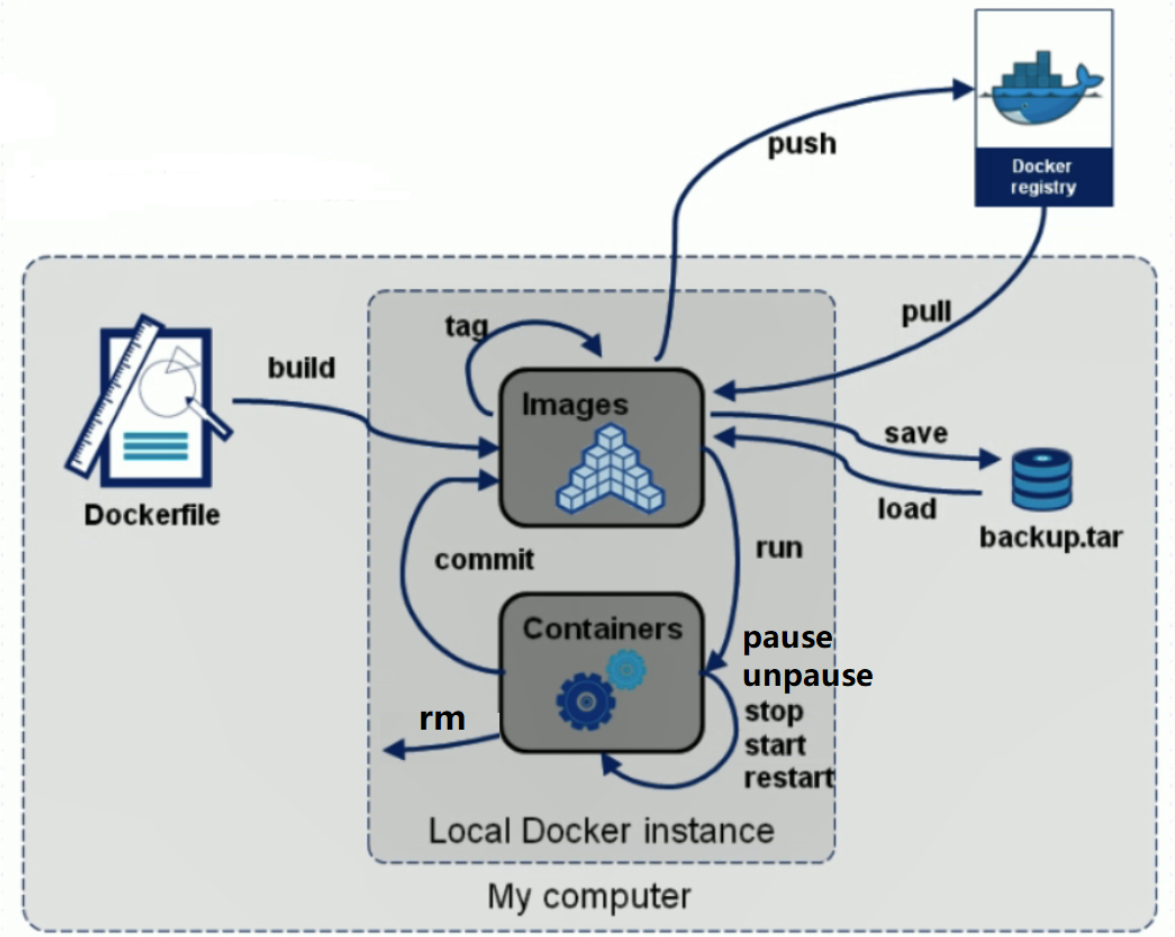
2.5 小结

- 利用docker pull命令拉取镜像
- 利用docker save命令将镜像保存为tar包
- 利用docker rmi 删除本地的镜像
- 利用docker load 重新加载镜像

3 容器操作

3.1 容器相关命令

容器操作的命令如图：



容器保护三个状态：

- 运行：进程正常运行
- 暂停：进程暂停，CPU不再运行，并不释放内存
- 停止：进程终止，回收进程占用的内存、CPU等资源

其中：

- docker run：创建并运行一个容器，处于运行状态
- docker stop：停止一个运行的容器
- docker start：让一个停止的容器再次运行
- docker restart：重新启动容器
- docker rm：删除一个容器
- docker pause：让一个运行的容器暂停
- docker unpause：让一个容器从暂停状态恢复运行

3.2 创建并运行容器

需求：创建并运行nginx

命令语法：

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

语法解析：

- docker run ： 创建并运行一个容器
- --name : 给容器起的名字，例如叫做ng
- -p ： 将宿主机端口与容器端口映射，冒号左侧是宿主机端口，右侧是容器端口
- -d：后台运行容器
- nginx：镜像名称，例如nginx

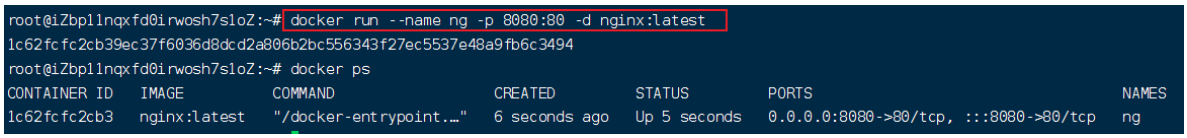
这里的`-p`参数，是将容器端口映射到宿主机端口。

默认情况下，容器是隔离环境，我们直接访问宿主机的80端口，肯定访问不到容器中的nginx。

现在，将容器的80与宿主机的80关联起来，当我们访问宿主机的80端口时，就会被映射到容器的80，这样就能访问到nginx了：

运行nginx容器

```
docker run --name ng -p 8080:80 -d nginx:latest
```

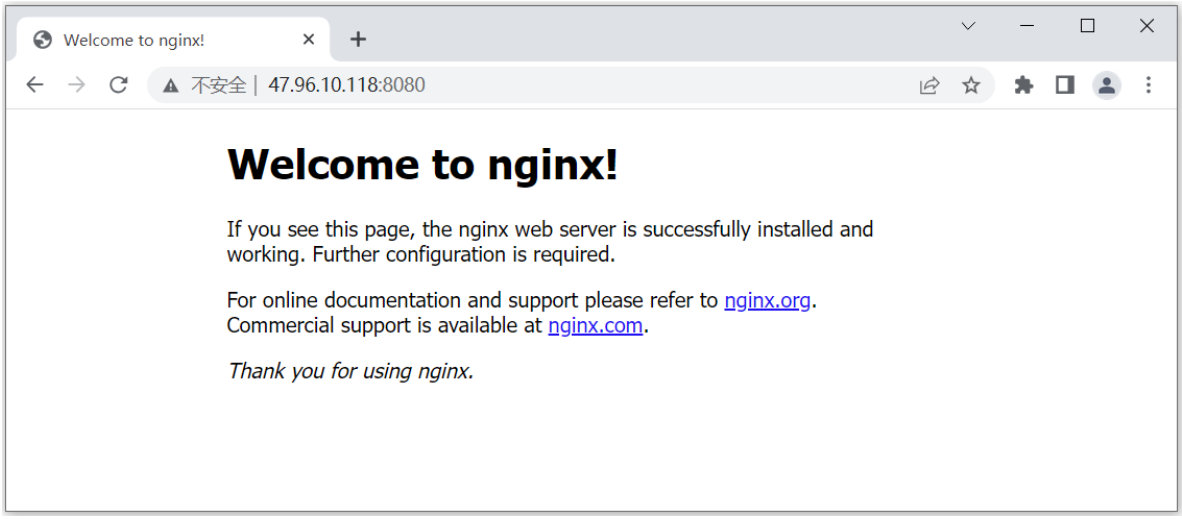


查看运行的容器

```
docker ps
```

访问docker容器中的nginx

打开浏览器，输入docker容器宿主机的ip和nginx的映射端口，例如我的是：http://47.96.10.118:8080，即可看到结果：



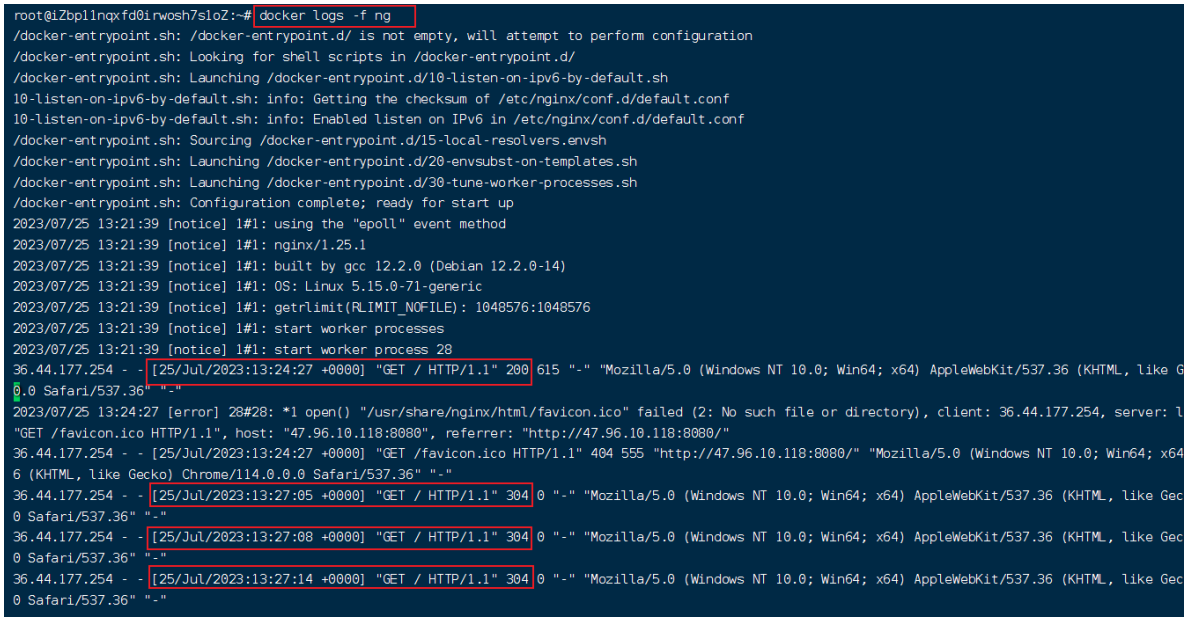
查看docker容器中的nginx访问日志

```
1  docker logs ng
```

```
root@iZbp11nqxfd0irwosh7s1oZ:~# docker logs ng
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2023/07/25 13:21:39 [notice] 1#1: using the "epoll" event method
2023/07/25 13:21:39 [notice] 1#1: nginx/1.25.1
2023/07/25 13:21:39 [notice] 1#1: built by gcc 12.2.0 (Debian 12.2.0-14)
2023/07/25 13:21:39 [notice] 1#1: OS: Linux 5.15.0-71-generic
2023/07/25 13:21:39 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2023/07/25 13:21:39 [notice] 1#1: start worker processes
2023/07/25 13:21:39 [notice] 1#1: start worker process 28
36.44.177.254 - - [25/Jul/2023:13:24:27 +0000] "GET / HTTP/1.1" 200 615 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/5
0.0 Safari/537.36" "-"
2023/07/25 13:24:27 [error] 28#28: *1 open() "/usr/share/nginx/html/favicon.ico" failed (2: No such file or directory), client: 36.4
```

持续查看docker容器中的nginx访问日志

```
1  docker logs -f ng
```



停止持续显示访问日志：Ctrl + C

3.3 小结

docker run命令的常见参数有哪些?

- --name：指定容器名称
- -p：指定端口映射
- -d：让容器后台运行

查看容器日志的命令：

- docker logs
- 添加 -f 参数可以持续查看日志

查看容器状态：

- docker ps
- docker ps -a 查看所有容器，包括已经停止的

4 数据卷

问题：

- 镜像中包含了应用程序及其所需的依赖、函数库、环境、配置等，还包括了应用程序的数据。如果在应用程序运行过程中修改了数据，**那么删除镜像后，数**

据也同时删除了。

原因：

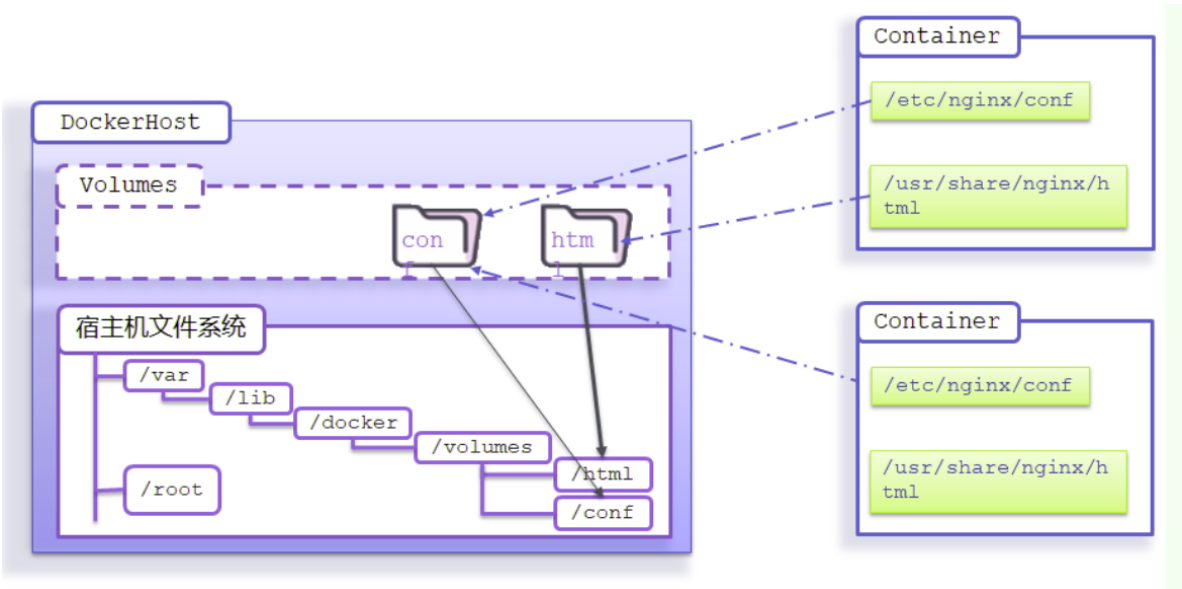
- 产生这种问题的原因是容器与数据（容器内文件）耦合带来的后果。

改进：

- 要解决这个问题，必须将数据与容器解耦，这就要用到数据卷了。

4.1 什么是数据卷

数据卷（volume）是一个虚拟目录，指向宿主机文件系统中的某个目录。



一旦完成数据卷挂载，对容器的一切操作都会作用在数据卷对应的宿主机目录了。

这样，我们操作宿主机的/var/lib/docker/volumes/html目录，就等于操作容器内的/usr/share/nginx/html目录了

4.2 数据集操作命令

数据卷操作的基本语法如下：

```
docker volume [COMMAND]
```

docker volume命令是数据卷操作，根据命令后跟随的command来确定下一步的操作：

- create 创建一个volume
- inspect 显示一个或多个volume的信息
- ls 列出所有的volume
- prune 删除未使用的volume
- rm 删除一个或多个指定的volume

4.3 创建和查看数据卷

需求：创建一个数据卷，并查看数据卷在宿主机的目录位置

创建数据卷

```
docker volume create html
```

查看所有数据卷

```
docker volume ls
```

结果：

```
root@iZbp11nqxfd0irwosh7s1oZ:~# docker volume create html
html
root@iZbp11nqxfd0irwosh7s1oZ:~# docker volume ls
DRIVER    VOLUME NAME
local     html
```

查看数据卷详细信息卷

```
docker volume inspect html
```

结果：

```
root@iZbp11nqxfd0irwosh7s1oZ:~# docker volume inspect html
[
    {
        "CreatedAt": "2023-07-25T21:42:51+08:00",
        "Driver": "local",
        "Labels": null,
        "Mountpoint": "/var/lib/docker/volumes/html/_data",
        "Name": "html",
        "Options": null,
        "Scope": "local"
    }
]
```

可以看到，我们创建的html这个数据卷关联的宿主机目录为`/var/lib/docker/volumes/html/_data`目录。

小结：

数据卷的作用：

- 将容器与数据分离，解耦合，方便操作容器内数据，保证数据安全

数据卷操作：

- docker volume create：创建数据卷
- docker volume ls：查看所有数据卷
- docker volume inspect：查看数据卷详细信息，包括关联的宿主机目录位置
- docker volume rm：删除指定数据卷
- docker volume prune：删除所有未使用的数据卷

4.4 挂载数据卷

我们在创建容器时，可以通过 -v 参数来挂载一个数据卷到某个容器内目录，命令格式如下：

```
docker run --name ng -v html:/root/html -p 8080:80 -d nginx
```

这里的-v就是挂载数据卷的命令：

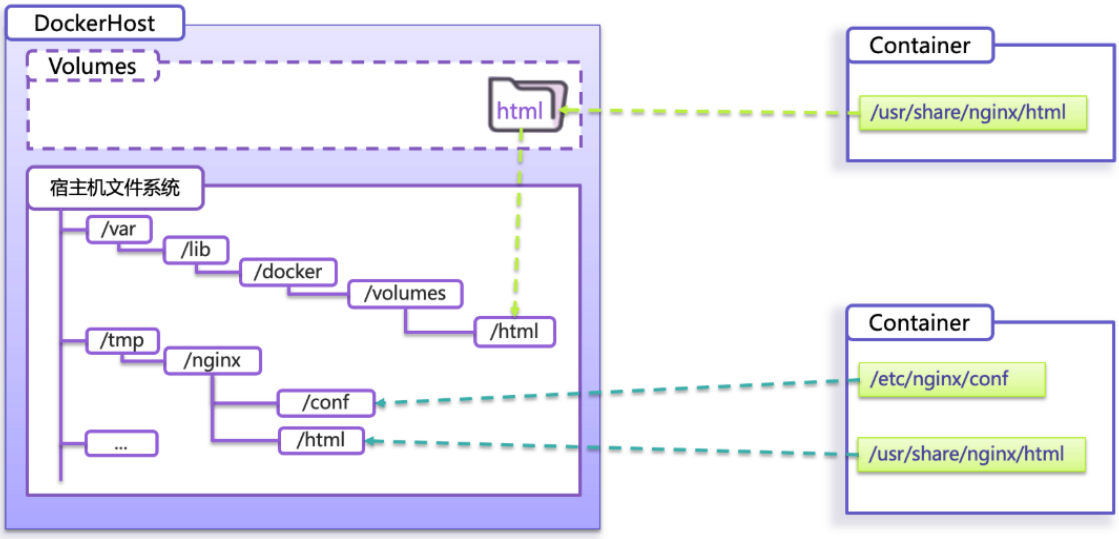
- `-v html:/root/html`：把html数据卷挂载到容器内的/root/html这个目录中

4.5 给MySQL挂载本地目录

容器不仅仅可以挂载数据卷，也可以直接挂载到宿主机目录上。关联关系如下：

- 带数据卷模式：宿主机目录 ---> 数据卷 ---> 容器内目录
- 直接挂载模式：宿主机目录 ---> 容器内目录

如图：



语法：

目录挂载与数据卷挂载的语法是类似的：

- -v [宿主机目录]:[容器内目录]
- -v [宿主机文件]:[容器内文件]

需求：创建并运行一个MySQL容器，将宿主机目录直接挂载到容器

步骤：

1. mysql.tar文件上传到服务器
2. 创建目录/usr/mysql/data
3. 创建目录/usr/mysql/conf
4. 将提供的hmy.cnf文件上传到/usr/mysql/conf
5. 挂载/usr/mysql/data到mysql容器内数据存储目录
6. 挂载/usr/mysql/conf/hmy.cnf到mysql容器的配置文件
7. 设置MySQL密码

实现过程如下：

1）加载mysql镜像

```
docker load -i mysql.tar
```

```
root@iZbp11nqxfd0irwosh7s1oZ:~# docker load -i /usr/tools/mysql.tar
5dacd731af1b: Loading layer [==================================================>]  58.45MB/58.45MB
f411d8bde01c: Loading layer [==================================================>]  338.4kB/338.4kB
0aa7d65147ef: Loading layer [==================================================>]  10.44MB/10.44MB
3437f67a712b: Loading layer [==================================================>]  4.472MB/4.472MB
ec41e34b35a0: Loading layer [==================================================>]  1.536kB/1.536kB
458d25c646d8: Loading layer [==================================================>]  46.15MB/46.15MB
97874ea0e7f9: Loading layer [==================================================>]  31.74kB/31.74kB
5075b9328698: Loading layer [==================================================>]  3.584kB/3.584kB
364557e875f1: Loading layer [==================================================>]  257.3MB/257.3MB
9209148debed: Loading layer [==================================================>]  9.728kB/9.728kB
82582edf9553: Loading layer [==================================================>]  1.536kB/1.536kB
Loaded image: mysql:5.7.25
```

2）运行mysql容器

```
docker run \
 --name mysql \
 -e MYSQL_ROOT_PASSWORD=root \
 -p 3309:3306 \
 -v /usr/mysql/conf/hmy.cnf:/etc/mysql/conf.d/hmy.cnf \
 -v /usr/mysql/data:/var/lib/mysql \
 -d \
 mysql:5.7.25
```

4.6 小结

docker run的命令中通过 -v 参数挂载文件或目录到容器中：

- -v volume名称:容器内目录
- -v 宿主机文件:容器内文
- -v 宿主机目录:容器内目录

数据卷挂载与目录直接挂载的

- 数据卷挂载耦合度低，由docker来管理目录，但是目录较深，不好找
- 目录挂载耦合度高，需要我们自己管理目录，不过目录容易寻找查看