## Docker基本操作

### 一：镜像操作：

#### 镜像相关命令：镜像命名规范：

镜像名称一般分两部分组成：[repository]:[tag]，repository指什么镜像名称，tag指镜像版本，如：mysql:5.7

如果没有指定tag时，默认时latest，代表最新版本的镜像。

#### 镜像操作命令（无非就是CRUD）：

##### 搜索镜像命令：docker search xxx，或者到DockerHub官网搜索：hub.docker.com

##### 获取镜像命令：

一般有两种方式，一是从本地获取，需要一个dockerfile的文件，利用**docker build**命令构建出一个镜像；第二种是使用**docker pull** 命令从docker registry镜像服务器拉取镜。

##### 查看镜像命令：docker images

##### 删除镜像命令：docker rmi xxx rmi是remove image的缩写

##### 分享镜像命令：docker save & docker load

有两种方式，一是使用**docker push**命令把镜像推送到docker registry镜像服务器去；二是使用**docker save**命令把镜像保存为一个**压缩包**，然后用U盘把镜像拷给别人，然后使用**docker load**命令加载压缩包为镜像。

###### eg：docker save -o nginx.tar nginx

-o：输出、保存的意思

nginx.tar：输出的形成的压缩文件

nginx：要压缩的镜像名

###### eg：docker load -i nginx.tar

-i：输入、读取的意思

nginx.tar：要加载的压缩文件

##### Docker帮助文档命令：docker --help

Docker命令众多，我们无法记住所有命令，只要知道怎么查看帮助文档就行：通过使用**docker --help**命令就能查看docker中的所有命令；如果想知道某个命令具体怎么使用，比如想详细地知道iamges命令的作用，使用**docker images --help**就可以知道，其他以此类推。

##### 利用docker save将nginx镜像导出从磁盘，然后再通过docker load加载回来：

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

docker save -o nginx.tar nginx:latest

-o nginx.tar：保存到什么文件

nginx:latest：要导出镜像

docker load -i nginx.tar

-i nginx.tar：要加载成镜像的文件

### 二：容器操作：

容器具有运行、暂停、停止状态，可以使用命令让容器在这几个状态之间切换。

#### 创建容器并运行的命令：docker run

docker run命令不仅仅能帮我们基于镜像创建容器，而且还可以让创建的容器处于运行状态。

##### eg1：例子：创建运行一个nginx容器

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

**命令解读：**

**docker run：**创建并运行一个容器

**--name：**给容器起一个名字，比如叫nginx，将来查找、管理容器更方便，每个容器的名字是唯一的

**-p：**端口映射，将宿主机端口与容器端口映射，冒号左侧是宿主机端口，冒号右侧是容器端口（顺序不能错，而且宿主机的端口只要没被占用，可以任意的端口，容器端口往往是不变的，取决于应用程序本身）。

比如我们有一台centos服务器，这台服务器的ip地址是192.168.150.101，现在我们在这台服务器上的docker部署了一台nginx容器，nginx容器的端口是80，此时外界的用户想访问nginx容器是访问不了了，因为我们之前讲过容器是对外隔离的，也就是说任何请求想直接访问容器时拒绝的。

那外界要访问容器要怎么办？因为创建容器的目的就是为了让外界访问。所以我们要做一个端口映射！端口映射就是比如宿主机有自己的很多端口，其中一个端口是80，那就可以让宿主机的80端口和nginx容器（其实容器可以看成是一个装了特定应用的微linux系统）的80端口产生一个关联映射，这样一来，任何进入宿主机80的请求就都会被转发到容器的80端口去执行，这样就等于是nginx容器接收处理了外界的请求：即这个时候，外界想访问容器nginx，就可以直接访问宿主机centos服务器192.168.150.101:80，请求进来，宿主机就会把请求给到nginx容器。

所以，端口映射的作用就是，把一个本来完全隔离的容器，暴露一个小口，让外界透过这个小口来访问。

**-d：**后台运行容器，不加-d的话，就是前台运行，一般都会加-d让容器后台去运行

**nginx：**镜像名称，表示创建容器是基于nginx镜像

容器除了运行状态，还有暂停状态、停止状态。只需要简单的命令，就可以让容器在这几个状态之间切换：

##### eg2：创建并运行一个redis容器，并且支持数据持久化

###### 创建并运行一个redis容器

**docker run --name myredis -p 6379:6379 -d redis redis-server --appendonly yes**

###### 进入redis容器：

方式一：docker exec -it myredis bash

方式二：docker exec -it myredis redis-cli（方式一是需要先进bash，再进redis-cli，其实可以直接进redis-cli，因为exec是进入容器执行一个命令，那么就可以直接指定进入容器执行redis-cli是没问题的，一步到位）

###### 连接reids：redis-cli

###### 查看数据：keys \*

###### 设置数据：set num 666

###### 退出redis：exit

###### 退出redis容器：exit

#### 运行状态进入暂停状态：docker pause

#### 暂停状态恢复运行状态：docker unpause

#### 运行状态进入停止状态：docker stop

#### 停止状态恢复运行状态：docker start

#### 暂停容器和停止容器的区别

问：为什么暂停容器用pause和unpause，停止却用stop和start呢？

答：主要区别是操作系统的处理方式。如果容器进入暂停状态，操作系统会将容器内的进程挂起，容器关联的内存暂存起来然后，cpu不再执行这个进程，但是如果回复暂停的容器，那么内存空间会恢复，程序接着被运行，容器又进入运行状态。

但是停止容器则不同，停止容器操作系统直接把进程杀死，容器所占用的内存回收，保留下来的仅剩容器的文件系统，也就是一些静态的东西，这就是暂停和停止的区别，因此，一旦停止容器，是没办法恢复的，因为进程已经被杀死了，所谓人死不能复活，能做的就是重新投胎生一个：即start，这个时候，会创建一个全新的进程。

#### 查看所有运行的容器及状态：docker ps

添加-a参数查看所有状态的容器

#### 查看容器运行日志：docker logs 容器名

可以产看运行过程中产生的细节、错误等，要想持续查看日志，添加**-f**参数可以持续查看日志

#### 进入容器执行命令：docker exec

不满足从表面观察容器，想深入运行的容器内部了解，可以让你进入运行中的容器内部做你想做的事。

##### eg：进入nginx容器，修改HTML文件内容，添加“FUCK YOU！ BITCH！”

**docker exec -it mynginx bash**

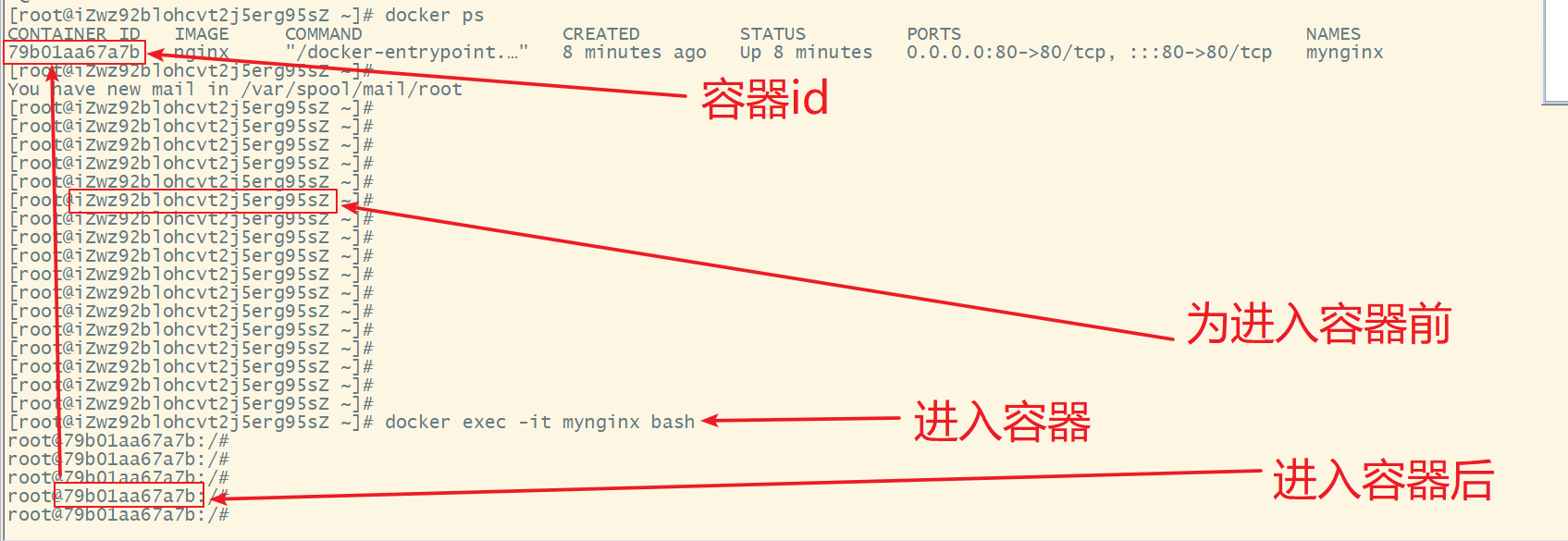
**命令解读：**

**docker exec：**进入容器内部，执行一个命令（注意：exec是可以进入容器修改文件，但是在容器中修改文件是不推荐的：首先第一点是不方便；第二点是你在容器中做的修改，是没有记录的，任何人的不知都，包括你自己，修改的当下可能知道，但是时间推移，肯定会忘掉）

**-it：**给当前进入的容器创建一个标准输入输出的终端，允许我们与容器交互。其实就是创建一个维持连接容器的输入输出的的终端，类似于java的标准输入输出

**mnginx：**要进入的容器的名称

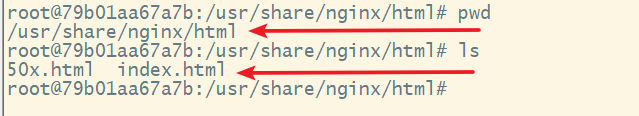
**bash：**进入容器后执行的命令，bash是一个于linux终端交互命令（我们之前学过的shell命令：比如cd、rm、mv等等，这些shell命令其实都是bash命令中的一部分，所以我们通过bash命令进入容器内部，就可以去修改文件了）。docker exec是进入容器执行命令，那执行的命令是啥，得指出来，此处就是bash这个命令，当然以后有别的命令，也可以改成其他的。



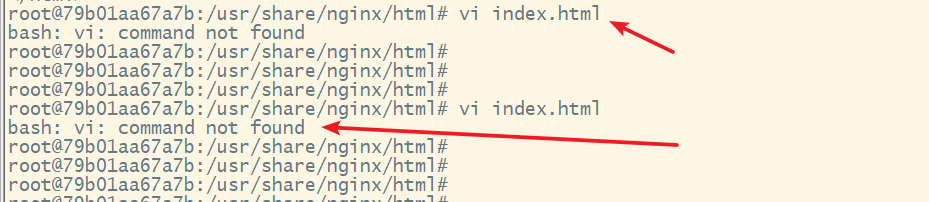


容器的内部，其实会有自己的一套文件系统，看起来跟linux的文件系统一样，但其实容器内部的文件系统是一个阉割版的文件系统，只有nginx容器运行需要的。如果想知道在nginx容器中，nginx在哪个目录，可以到DockerHub中查看nginx镜像作者提供的文档：因为作者在创建nginx镜像的过程中，需要去安装nginx，指定nginx的运行目录，这些事情都是nginx镜像作者决定的。

最终发现，nginx容器中，nginx的目录在：/usr/share/nginx，如下图示



当我们使用vi命令要去编辑index.html文件，会发现vi命令找不到，再一次证明了镜像封装时，只是应用程序所需要的必备的函数库以及命令，vi与nginx没什么关系，所以nginx镜像没有封装vi命令，再一次证明了镜像内部的文件系统和函数库都是阉割版的，只封装自己需要的。

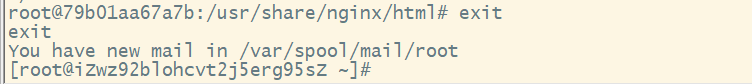


修改编辑方法：

sed -i 's#Welcome to nginx#FUCK YOU！BITCH！#g' index.html

sed -i 's#<head>#<head><meta charset="utf-8">#g' index.html

##### 退出容器：exit



#### 删除指定容器命令：docker rm

会把进程杀掉，内存回收，包括硬盘上的文件系统彻底干掉，毁尸灭迹

要删除容器，正常应该先把容器停止，然后再删除容器，如果要强制删除正在运行中的容器，需要添加参数：**-f，但是一般不推荐强制删除。**

#### 查看容器运行状态命令：docker ps & docker ps -a

### 三：数据卷（容器数据管理）

数据卷的作用就是将容器与数据分离，解耦合，方便操作容器内数据，保证数据安全。数据卷其实就是解决容器和数据耦合的问题！数据卷可以视为容器和宿主机的桥梁。

#### 问题：docker容器与数据耦合的问题：

Docker容器里面的文件也好、数据也好，都是跟容器耦合在一起的，因此，就会带来很多问题：

**问题一：不便于修改，**比如当我们要修改nginx的html内容时，不得不需要进入容器内部修改，而容器内部有没有高级的编辑工具，很不方便。

**问题二：数据不可复用问题，**在容器内的修改都是对外不可见的，所有修改对新创建的容器是不可复用的。

比如你在容器内一顿操作，随着时间推移，改得越来越多，可能你自己都不知道改了什么，将来有创建了新的容器，也希望与原来容器达到相同效果时，你会发现你要把对原来的容器所有动作对新容器重新做一遍，也就是说，你对原来容器做的修改对其他容器是不可复用的，不得不重复劳动。

**问题三：升级维护困难：**数据在容器中，如果要升级容器必然删除旧容器，容器一旦删除，自然所有数据都跟着删除了。

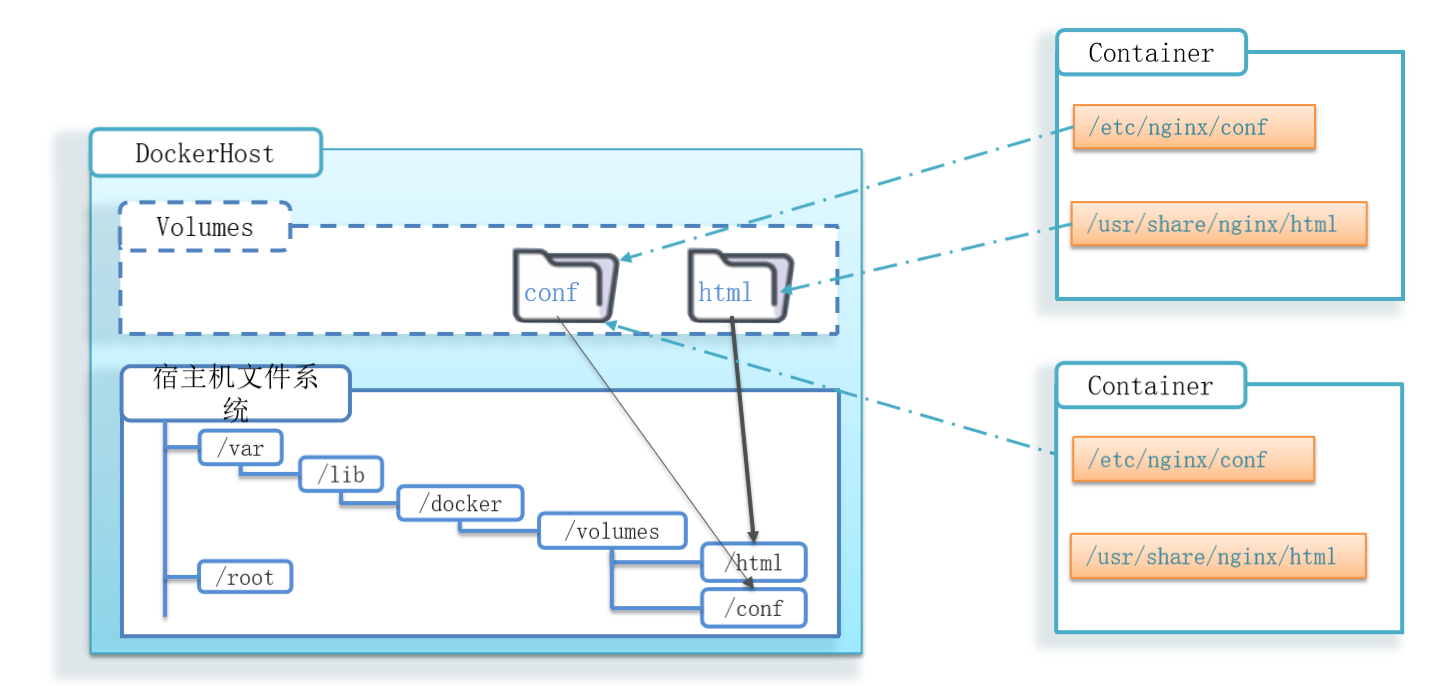
#### 数据卷：用来解决docker容器与数据耦合的问题

数据卷（volume）是一个虚拟目录，指向宿主机文件系统中的某个真实目录。

docker管理着很多的数据卷volumes，而所有的数据券都必然会指向宿主机文件系统上的一个目录，这个目录就在：/usr/lib/docker/volumes/xxxx

比如，我们利用docker创建了一个新的数据卷：html，那么docker一定会在宿主机指定的目录下创建一个html的目录。即每一个数据卷都会跟一个真实目录进行映射。我们可以认为数据卷是虚拟的，并不真实存在，只是一个概念，而真正指向的是硬盘上的一个真实的文件夹。

如下图：比如我们有一个nginx容器，我们知道nginx容器中，nginx的html目录在/us/share/nginx/html目录下，那么nginx所有的html文件以及静态资源文件都会放到这个目录下；nginx的所有配置都会放在/etc/nginx/conf下，那么就是数据与容器耦合了。



那么现在我们就可以让nginx的内部目录与数据卷进行关联，而当内部目录与数据卷进行关联时，本质其实是跟宿主机文件系统上的目录进行关联，这个时候docker就会管理容器了：比如我们在容器的html目录里写了一些东西，那么写的东西会立即写（反映）到宿主机文件系统的目录里；反过来，我们在宿主机文件系统的目录里对html目录的某个文件做了修改，这种修改也会立即反应到容器中的html对应的文件中去。可以认为：主机文件系统的目录里html目录 与 容器中的html目录 通过数据卷搭起了桥、建立了联系。

那么我们以上说的**问题**就得到了解决：

**解决修改不方便问题：**对宿主机文件系统的任意修改，一定会立即反应到跟对应数据卷关联容器的内部，再也不需要进入容器内部修改了，只需要在宿主机上通过高级编辑工具就能任意修改；

**解决数据共享问题：**比如我们对原来的容器改了好多配置，当我们创建了一个新的容器，希望原来容器的配置能共享过来，我们可以让新容器的配置目录挂载到原来容器配置目录的数据卷上，也就是说，两个容器可以挂载同一个数据卷，这个时候，你对原来容器配置做的一切修改，新的容器就也能看到；

**解决升级维护困难问题**：将来要升级容器版本，删除了容器，但是数据卷不会跟着删，宿主机文件系统对应的目录也还在，将来新版本的容器只需要接着挂载到原来容器的数据卷上，就可以共享以前旧的数据了。

#### 操作数据卷

操作数据卷的基本语法：docker volume [command]

docker vlomue命令是数据卷操作，是一个二级命令，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** 查看一个或者多个volume的详细信息，如果不知道创建的数据卷对应宿主机的目录，可以使用这个命令获得

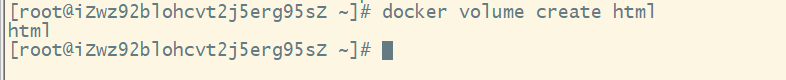
##### **ls** 列出所有的volume

##### **prune** 删除未使用的volume

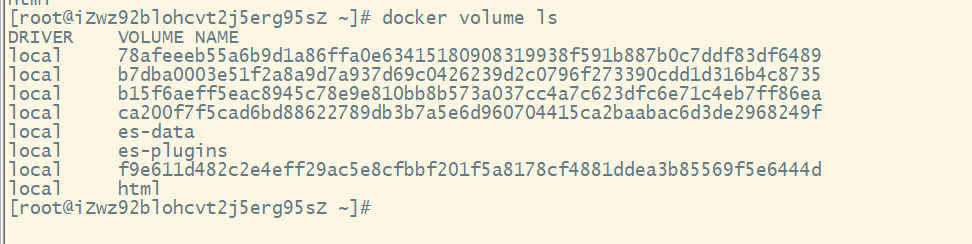
##### **rm** 删除一个或者多个指定的volume

##### eg：创建一个数据卷，并查看数据卷在宿主机的目录位置

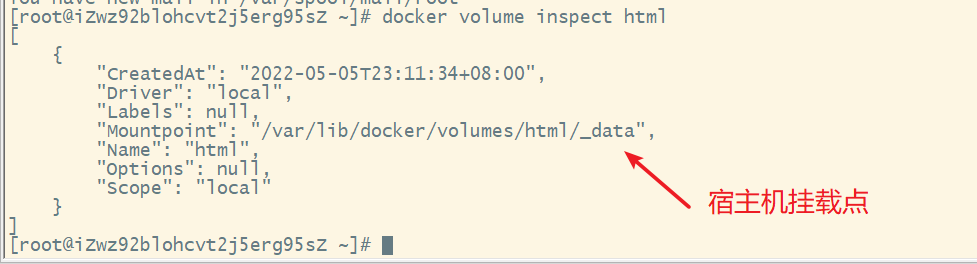
一：创建数据卷：docker volume create html



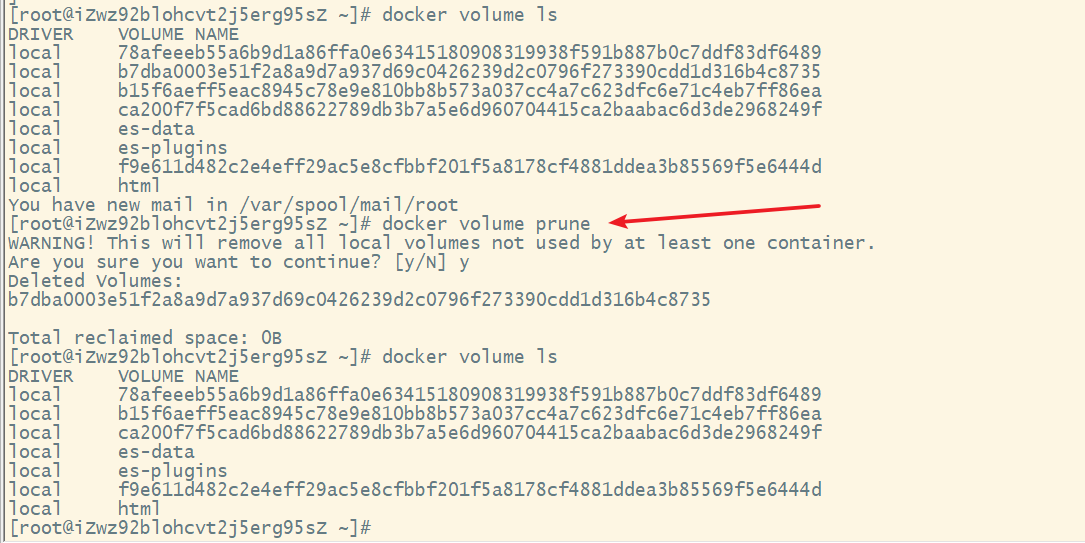
二：查看所有数据卷：docker volume ls



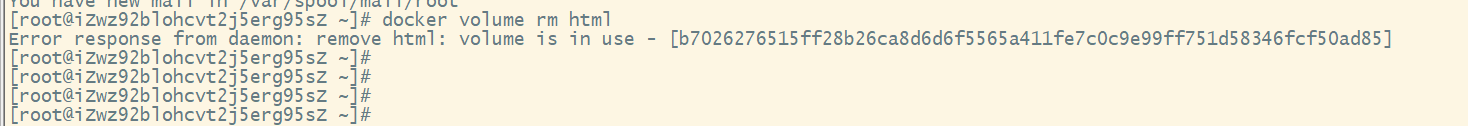
三：查看数据卷详细信息：docker volume inspect html



四：删除未使用的数据卷



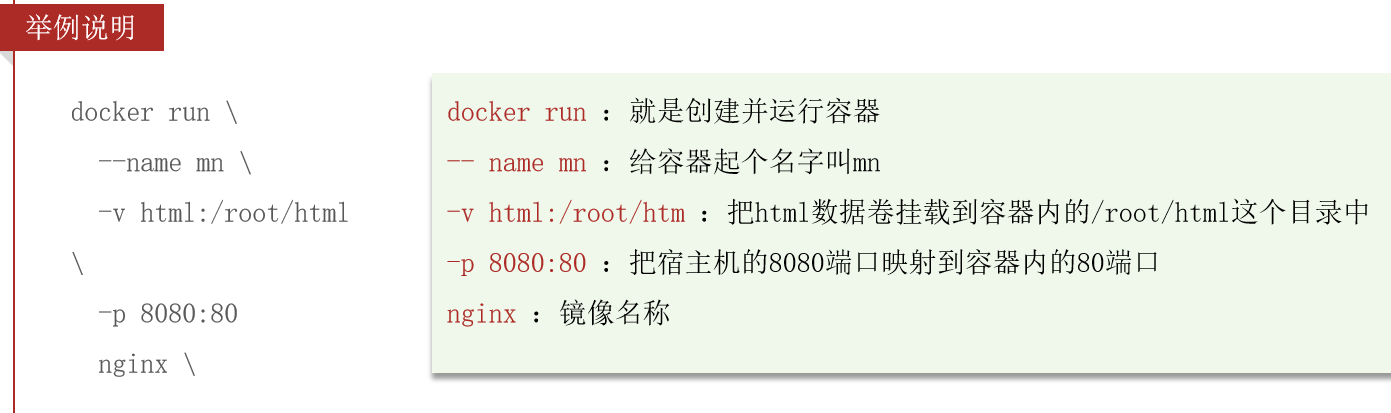
删除指定数据卷 docker volume rm html



#### 挂载数据卷

以上只是展示了如何操作一个数据卷，但是如何将一个数据卷挂载到容器中？

我们可以在创建容器时，可以通过-v参数（vloume）来挂载一个数据卷到某个容器目录，前半部分是数据卷名称，后半部分是容器内目录。



##### eg：创建一个nginx容器，修改容器内的html目录内的index.tml内容

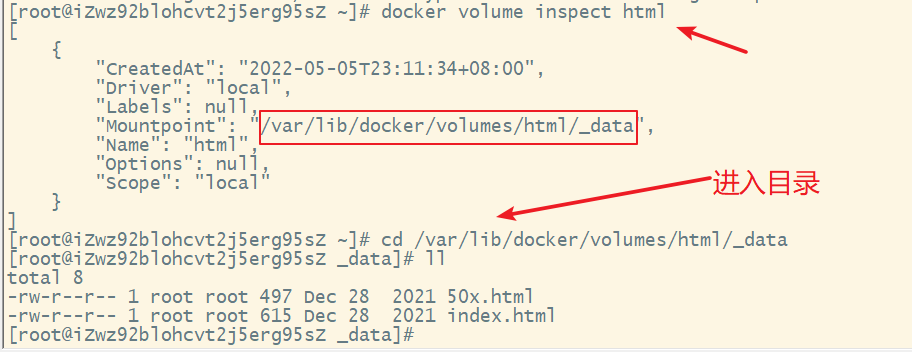
需求说明：之前案例中，我们进入nginx容器内部，已经知道nginx的html目录所在位置：/use/share/nginx/html，我们需要把这个目录挂载到html这个数据卷上，方便操作其中的内容：运行容器时使用-v参数挂载数据卷

步骤一：创建容器并挂载数据卷到容器内的html目录

**docker run --name mynginx -p 80:80 -v html:/usr/share/nginx/html -d nginx**

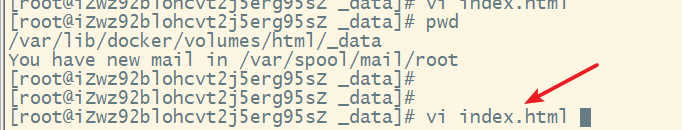
步骤二：进入html数据卷所在目录，并修改html内容

cd /var/lib/docker/volumes/html/\_data



上图，可以看到、说明我们已经将nginx容器的：/usr/share/nginx/html目录挂载到宿主机的真实目录： /var/lib/docker/volumes/html/\_data

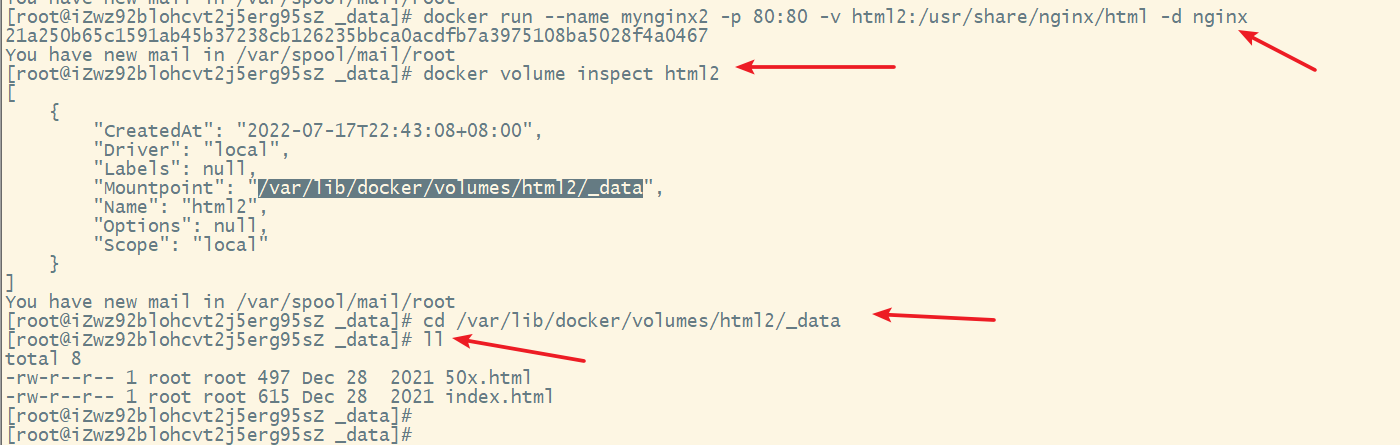
然后就可以直接在宿主机修改了index.html文件了：



以上案例，我们完成数据卷挂载时，数据卷是提前创建好的，其实，如果数据卷不存在，会帮我们自动创建：

docker run --name mynginx2 -p 80:80 -v html2:/usr/share/nginx/html -d nginx

见下图：



#### 挂载宿主机目录

上面的案例是将数据卷挂载到容器目录，事实上不仅仅数据卷的方式，我们的宿主机目录可以直接与容器进行挂载！

提示：目录挂载与数据卷挂载的语法是类似的，以下是目录挂载语法：

一：-v [宿主机目录]:[容器内目录]

二：-v [宿主机文件]:[容器内文件（数据卷挂载不具备此功能）

##### eg：创建并运行一个mysql容器，系那个宿主机目录直接挂载到容器：

实现思路如下：

1：将课前资料中的mysql.tar文件上传到服务器，通过load命令加载为镜像

2：创建目录/tem/mysql/data mkdir -p /tem/mysql/data（-p多级创建）

3：创建目录 /tem/mysql/conf，将课前资料提供的hmy.cnf文件上传到/tem/mysql/conf

4：去DockerHub查阅资料，创建并运行mysql容器，要求：

4.1：挂载/tem/mysql/data到mysql容器内数据存储目录

4.2：挂载/tem/mysql/conf/hmy.cnf到mysql容器的配置文件

4.3：设置mysql密码

docker run \

--name mysql \

-e MYSQL\_ROOT\_PASSWORD=1234 \

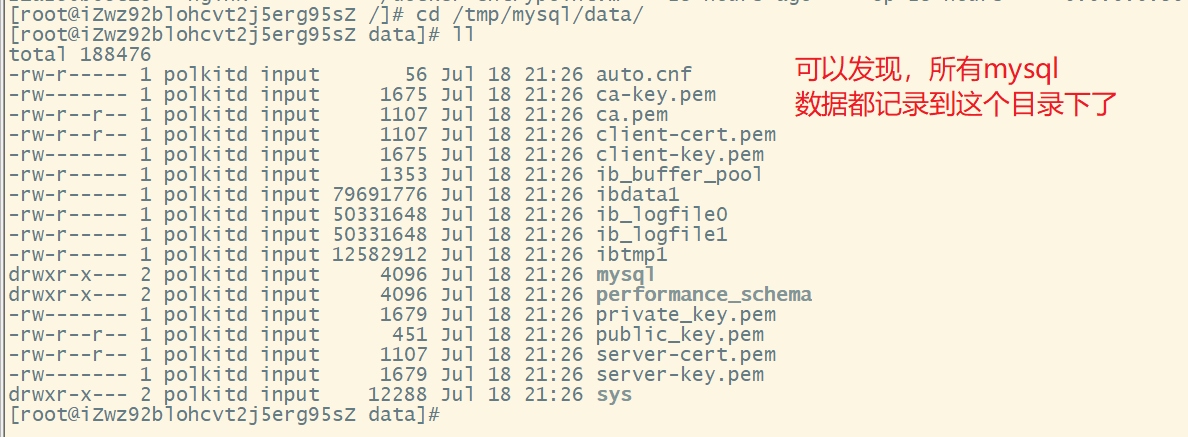
-p 3306:3306 \

-v /tmp/mysql/conf/hmy.cnf:/etc/mysql/conf.d/hmy.cnf \

-v /tmp/mysql/data:/var/lib/mysql \

-d \

mysql:5.7.25



##### Vloume数据卷总结

目前为止我们已经学了两种数据导入的方式，一个是基于数据卷，一个是基于目录直接挂载。对比：

基于数据卷：当我们用数据卷时，我们的docker会自动帮我们在宿主机文件系统创建数据卷对应的目录，这样，数据卷就指向了这个目录，而docker挂载时，只需要挂载数据卷就行了，不需要关心目录在哪里，所以这种方式等于全权交给docker去处理，我们不需要去操心，劣势是数据卷对应宿主机的目录不是我们创建的，我们要查了才知道目录在什么地方，并且目录比较深/var/lib/docker/volumes/，想去找比较麻烦。

基于目录挂载，这种方式目录是我们自己创建的，虽然创建的时候比较麻烦，但是我们很清楚地知道目录在什么地方，将来挂载的时候，不会像数据卷方式一样要数据卷做代理，而是可以直接挂载到目录。因此，可知快速定位文件。

#### 总结：

1：docker run的命令中通过-v参数挂载文件或目录到容器中：

1.1：-v vloume名称:容器内目录

1.2：-v 宿主机文件:容器内文件

1.3：-v 宿主机目录:容器内目录

2：数据卷挂载和目录直接挂载区别：

2.1：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

2.2：目录挂载哦耦合度高，需要我们自己管理目录，不过目录容易寻找查看。