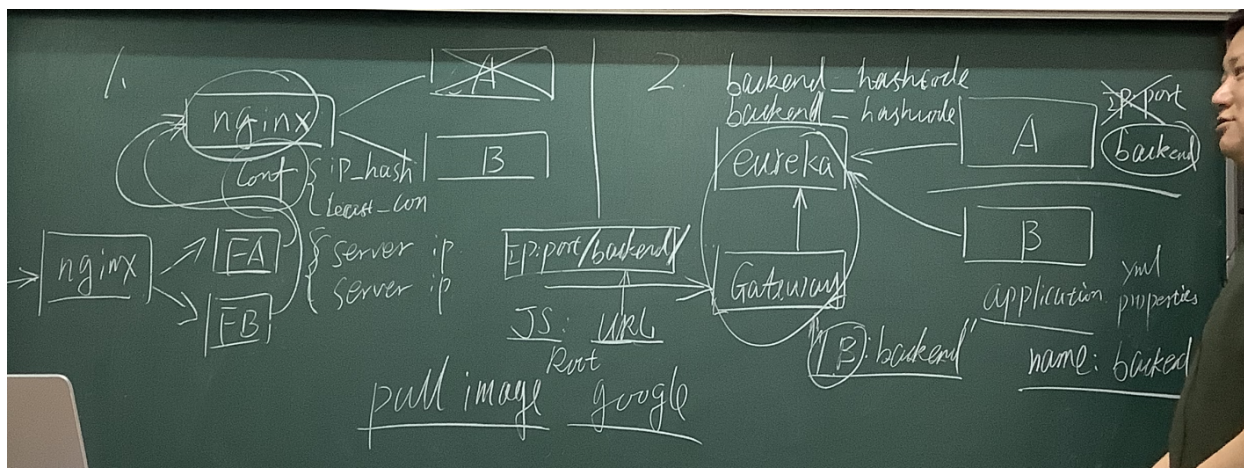


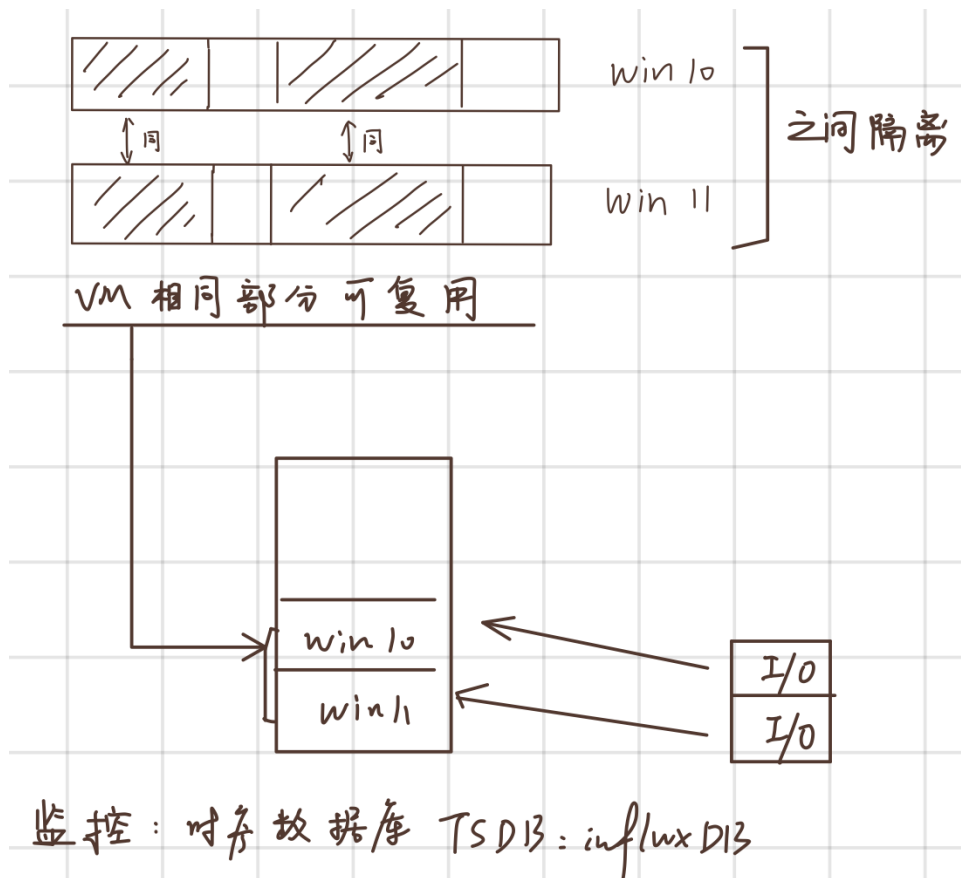
# lect22: Virtualization & Container



对于图2来说，目前我们有两个实例，所以就**根据注册的名字来做负载均衡**。前端需要全部指向gateway。对于作业10，这两个方案都可以用。通常情况下情况2更好一点，因为机器如果crash了，因为重启以后会重新到eureka注册一遍，所以不会有任何问题。但是在方案1中，如果server的ip和port变了，那么可能需要修改一些配置文件。

虚拟化：一份代码可以在多个机器上跑，虚拟机可以跑原机器不能提供的环境。

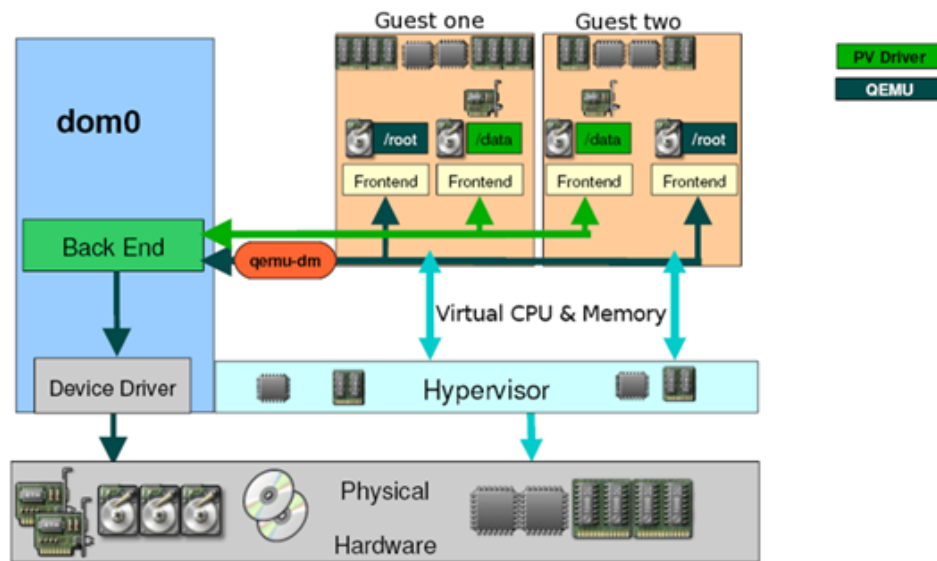
缺点：和分层一样，中间插入了很厚一层来描述运行环境。所以效率要低很多，以ORM映射为例，它要把对于对象的操作翻译成SQL语句并且通过JDBC发送过去。我们插了一个很大的虚拟机来描述不存在的环境，所以性能显然收到影响。



虚拟机需要跑一个完整的操作系统。Xen和K8S有一种想法就是我们把依赖的Linux内核改掉，改成对Xen的hypervisor的访问。我们对Linux内核的重新编译，修改为对Xen的hypervisor的调用，然后hypervisor再和硬件去交互。好处就是性能比较高，缺点就是要重新编译Linux内核。这就是半虚拟化和超虚拟化的方式。

# Xen Full Virtualization Architecture

With the para-virtualized drivers



橙色的就是一个完整的操作系统，对上就可以跑各自的应用。这个原生的操作系统，内存和CPU的调用直接传给Hypervisor去调度。DOM0可以认为是管理的虚拟机，IO操作全部发送给DOM0，然后转换为对真正物理机上的操作。还有一种用的是qemu(Qemu-DM帮助完全虚拟化客户机（Domain U HVM Guest）获取网络和磁盘的访问操作)，它是软件去模拟的虚拟，后面演化为全部使用PV driver。

			<div>Privileged Instructions, Page Tables</div> <div>Emulated Motherboard, Legacy Boot</div> <div>Interrupts &amp; Timers</div> <div>Disk and Network</div>			
<div></div> Poor Performance	<div></div> Scope for Improvement	<div></div> Optimal Performance				
P = Paravirtualized VS = Software Virtualized (QEMU) VH = Hardware Virtualized						
Shortcut	Mode	With				
HVM / Fully Virtualized	HVM		VS	VS	VS	VH
HVM + PV drivers	HVM	PV Drivers	P	VS	VS	VH
PVHVM	HVM	PVHVM Drivers	P	P	VS	VH

# Docker

容器：默认东西都在linux上面跑。Application 和依赖的环境(eg. node.js)打包。

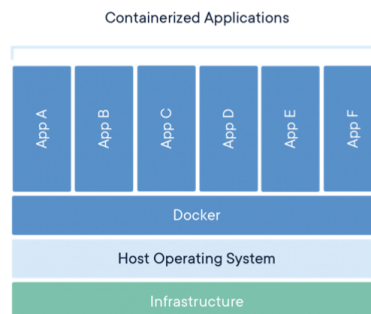
docker是一种容器，**不是一个完整的应用**，是把我们写的代码和我们需要依赖的所有东西打包在一起。构成这样一个**软件单元**，这样将来在跑的时候只需要创建一个容器跑起来即可。

注意！两个容器有重复的也会有复用。依赖环境也是分层打包的，比如依赖是mysql:5.7 如果本地已经有这个环境了，就不用拉了。

- **What is a Container?** <https://www.docker.com>



- A **container** is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another.
- A **Docker container image** is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.
- **Container images become containers at runtime** and in the case of Docker containers - images become containers when they run on **Docker Engine**.
- Containers **isolate** software from its environment and ensure that it works uniformly despite differences for instance between development and staging.



10

- 容器是软件的标准单元，它将代码及其所有依赖项打包，以便应用程序在一个计算环境到另一个计算环境之间快速可靠地运行。
- Docker容器镜像是一个轻量级的、独立的、可执行的软件包，它包含运行应用程序所需的一切:代码、运行时、系统工具、系统库和设置。
- 容器镜像在运行时变成容器，在Docker容器的情况下-镜像在Docker引擎上运行时变成容器。
- 容器将软件从它的环境中隔离出来，并确保它一致地工作，尽管在开发和登台之间存在差异。

与docker相关的一些概念：

**镜像：**docker镜像使用Dockerfile脚本，将你的应用以及应用的依赖包构建而成的一个应用包，它通常带有该应用的启动命令。而这些命令会在容器启动时被执行，也就是说你的应用在启动容器时被启动。

镜像的创建，需要通过配置Dockerfile脚本，然后执行docker build命令来创建。

**容器：**使用 docker run --name 容器名 镜像 命令创建的，独立于宿主机(服务器)的沙箱，也可以理解为一个带有特殊结构的盒子，它在创建时会自动执行镜像自带的一些指令，从而实现该应用的运行。

狭义地讲，容器的主要作用就在于给镜像提供运行空间和环境，并执行镜像的指令。

**仓库：**用来存东西的，但不是存容器，而是存储docker镜像。你可以把你的docker镜像通过push命令推送到docker仓库，然后就可以在任何能使用docker命令的地方通过pull命令把这个镜像拉取下来。

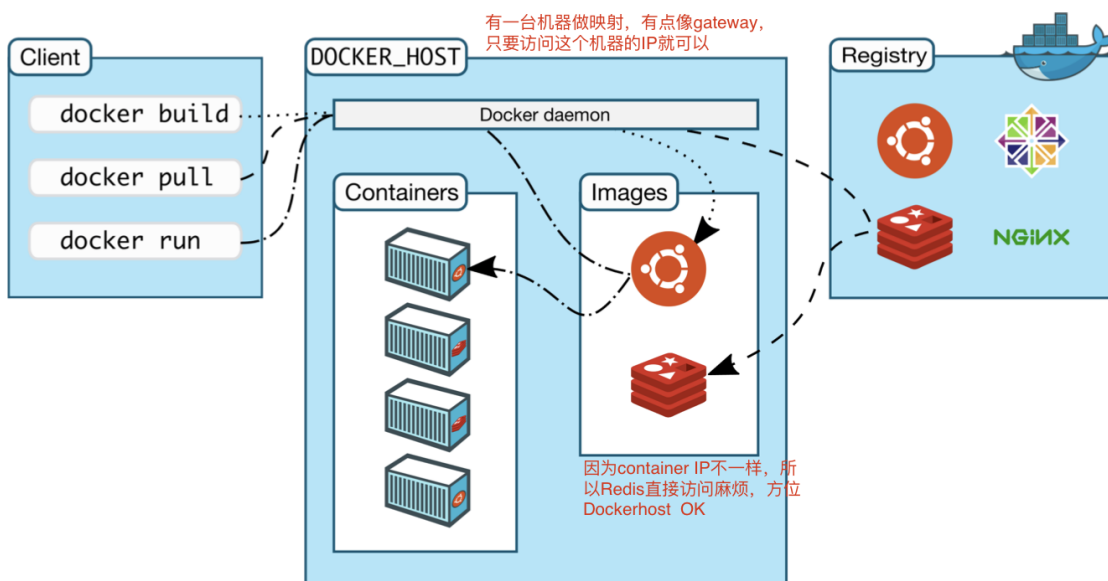
### **docker基本开发流程**

- 1，寻找基础镜像
- 2，基于基础镜像编写Dockerfile脚本
- 3，根据Dockerfile脚本创建项目镜像
- 4，将创建的镜像推送到docker仓库 (根据自身需要，可做可不做)
- 5，基于项目镜像创建并运行docker容器 (实现最终部署)

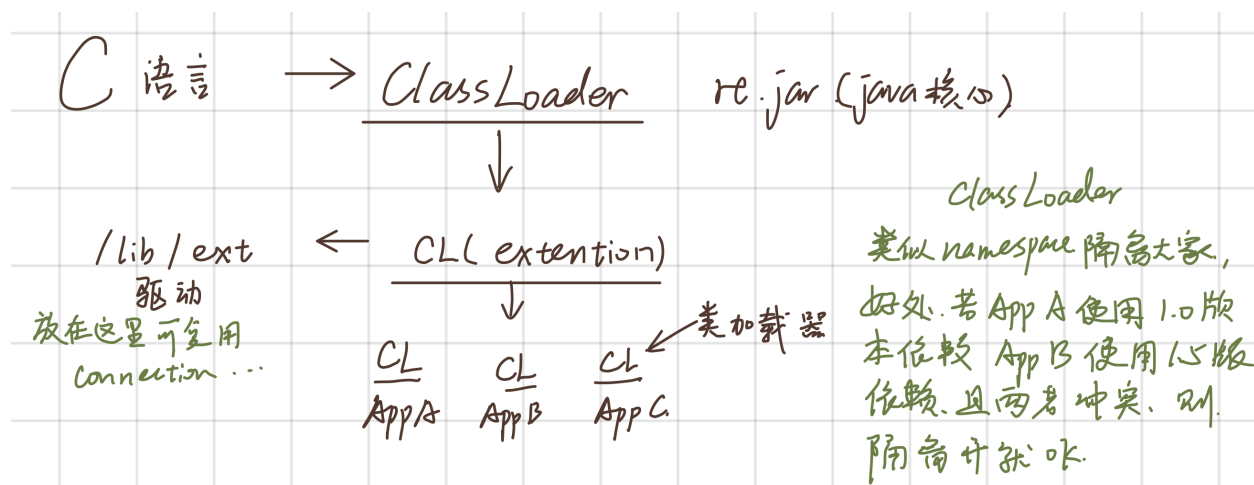
在现有的image上面开发应用。run 先在本地找，找不到从远端拉。

container之间是隔离的，而且每一个有独立的IP。

## Docker uses a client-server architecture



不同的虚拟机之间（硬盘）不能相互访问，要通过网络访问。使用namespace将他们隔离开。比如一台机器上跑了2台虚拟机，虚拟机不能互相访问到对方的硬盘。



之间还是有牵制因为在一个tomcat上面跑，如果这个tomcat如果crash了，那么这三个应用都崩了。

如果docker 出错了，在命令行界面会有东西出现。

每次都要启动的时候手动加到/data.txt比较麻烦，能不能每次启动的时候自动把文件加进去呢？

**Volume就是一个卷的意思**，可以理解为就是一些数据，起程序的时候，我们要把外部的volume挂载到容器中的文件系统中。可以认为是一个目录挂在到docker里。

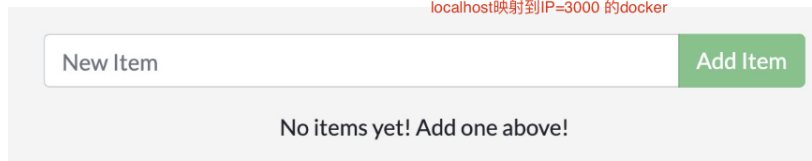
## Start an app container

- Start your container using the docker run command and specify the name of the image we just created:

```
docker run -dp 3000:3000 getting-started
```

- After a few seconds, open your web browser to <http://localhost:3000>.

localhost映射到IP=3000 的docker



New Item Add Item

No items yet! Add one above!

**docker的持久化**，如果docker 停了，新写的文件没了。要带一个参数保存这个文件

Volume 就是一些卷，相当于硬盘一样，image可以写到不同的volume， 如果image被销毁了，重启可以得到数据，挂在不同的volume可以获得不同的数据。

### 1. Create a volume by using the docker volume create command.

- `$ docker volume create todo-db`

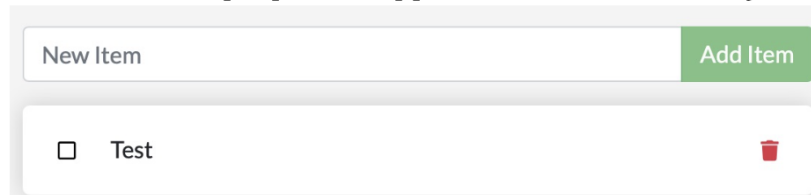
- Stop and remove the todo app container once again in the Dashboard (or with `docker rm -f <id>`), as it is still running without using the persistent volume.

### 2. Start the todo app container, but add the `-v` flag to specify a volume mount.

- We will use the named volume and mount it to `/etc/todos`, which will capture all files created at the path.

- `$ docker run -dp 3000:3000 -v todo-db:/etc/todos getting-started`

### 3. Once the container starts up, open the app and add a few items to your todo list.



New Item Add Item

☐ Test

创建一个叫做todo-db 的volume, 放到docker里面的etc/todos 目录里。

`docker volume inspect todo-db(volume name)` 可以检查volume的位置



volumn 可能不在自己的机器上

	Named Volumes	Bind Mounts
Host Location	Docker chooses	You control
Mount Example (using <code>-v</code> )	<code>my-volume:/usr/local/data</code>	<code>/path/to/data:/usr/local/data</code>
Populates new volume with container contents	Yes	No
Supports Volume Drivers	Yes	No

挂远程的volumn

之前要改数据需要重新build，现在直接改读数据的volume就可以了。

## Use bind mounts



- Start a dev-mode container
    - To run our container to support a development workflow, we will do the following:
    - Mount our source code into the container
    - Install all dependencies, including the “dev” dependencies
    - Start nodemon to watch for filesystem changes
1. Make sure you don't have any previous getting-started containers running.
  2. Run the following command in the directory of **getting-started-master/app**

```
$ docker run -dp 3000:3000 \  
  -w /app -v "$(pwd):/app" \  
  node:12-alpine \  
  sh -c "yarn install && yarn run dev"
```



## 2. Run the following command in the directory of **getting-started-master/app**

```
$ docker run -dp 3000:3000 \  
  -w /app -v "$(pwd):/app" \  
  node:12-alpine \  
  sh -c "yarn install && yarn run dev"
```

挂到这个容器之后它在哪个目录  
之前是docker 指定volumn 的地址  
现在是自己定了。

- **-dp 3000:3000** - same as before. Run in detached (background) mode and create a port mapping
- **-w /app** - sets the "working directory" or the current directory that the command will run from
- **-v "\$(pwd):/app"** - bind mount the current directory from the host in the container into the /app directory
- **node:12-alpine** - the image to use. Note that this is the base image for our app from the Dockerfile
- **sh -c "yarn install && yarn run dev"** - the command. We're starting a shell using sh (alpine doesn't have bash) and running yarn install to install *all* dependencies and then running yarn run dev. If we look in the package.json, we'll see that the dev script is starting nodemon.

持久化方法1：让docker给你管理volumn的地址

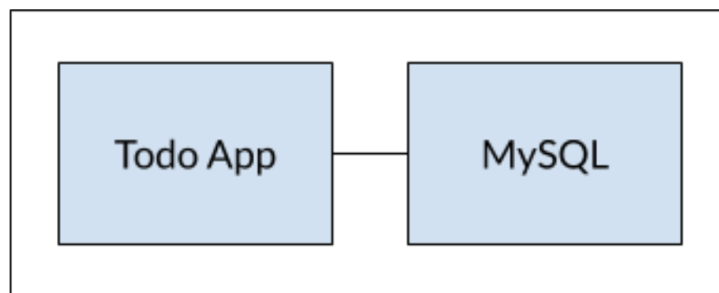
持久化方法2：自己指定volumn地址 bind mount

还有问题：如果写的不是文件系统，写的是数据库怎么办？应该mysql 有一个容器，应用有一个容器，这样耦合低，分开的话又要考虑容器如何交互。

volumn 设置可以是远程的，eg. Aliyun

eureka + gateway, gateway来做负载均衡。

## Multi container apps



这两个是一个pool, 跑K8S 调用的最小颗粒。

### 1. 可能MySQL后端跑两个，这样放在一起不合适

2. 如果一个崩了，放在一起整个都崩了
3. image 是分层的。每一层可能有些东西是完全一样的，在pull的时候只pull不一样的地方。
4. 不同的container之间通过网络通信。
  - a. `docker network create todo-app` 创造了叫做todo-app 的逻辑网络。

## Start MySQL

- Create the network.  
`docker network create todo-app`
- Start a MySQL container and attach it to the network. We're also going to define a few environment variables that the database will use to initialize the database.  
`docker run -d \`  
`--network todo-app --network-alias mysql \`  
`-v todo-mysql-data:/var/lib/mysql \` 把volumn 挂在到/var/lib/mysql  
`-e MYSQL_ROOT_PASSWORD=secret \`  
`-e MYSQL_DATABASE=todos \`  
`mysql:5.7`

## • Connect to MySQL

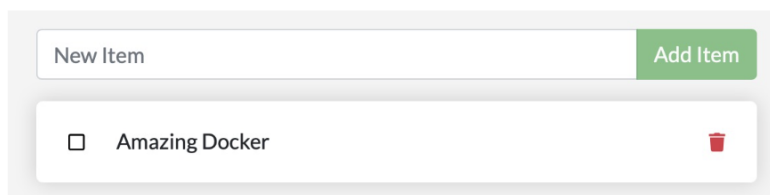
- We're going to make use of the [nicolaka/netshoot](#) container,
  - which ships with a *lot* of tools that are useful for troubleshooting or debugging networking issues.
- Start a new container using the [nicolaka/netshoot](#) image. Make sure to connect it to the same network.  
提供了很多工具，可以看网络里库在哪里  
`docker run -it --network todo-app nicolaka/netshoot`
- Inside the container, we're going to use the `dig` command, which is a useful DNS tool.
- We're going to look up the IP address for the hostname `mysql`.  
`dig mysql`

- Run your app with MySQL

- We'll specify each of the environment variables above, as well as connect the container to our app network.

```
docker run -dp 3000:3000 -w /app -v "$(pwd):/app" --network todo-app -e  
MYSQL_HOST=mysql -e MYSQL_USER=root -e MYSQL_PASSWORD=secret -e  
MYSQL_DB=todos node:12-alpine sh -c "yarn install && yarn run dev"
```

- Open the app in your browser and add a few items to your todo list.



网络像是一个标签， 不在一个网络的不可以通信。

## 多个容器一起启动：compose

因为启动的循序也是对应用有关的。如果先启动数据库，再启动应用可能报错了。通过写 docker-compose.yml 就可以指定顺序。

Run the application stack : docker-compose up -d

stop: docker-compose down

补充问答：

### 1、为什么要使用数据卷Volume？

Docker的数据卷Volume能让容器从宿主机中读取文件或持久化数据到宿主机主机内，让**容器与容器产生的数据分离开来**。可以简单的把Volume理解为Linux服务器上的挂载点。一个容器可以挂载多个不同的目录。Volume的生命周期是独立于容器的生命周期之外的，即使容器删除Volume也会保留下来，Docker也不会因为这个Volume没有被容器使用而回收。在容器中，添加或修改这个文件夹中的文件也不会影响容器的联合文件系统。

Volume数据卷不使用分层文件系统，这对经常读取和写入的数据很有用。在开发过程中，可以将代码目录挂载到容器中，这样如果更改代码容器会实时地得到文件修改的返回。容器中的挂载点必须是绝对路径，不支持相对路径。宿主机上的地址可以是一个绝对路径，也可以是一个数据卷名称。如果数据卷不存在，Docker会自动创建数据卷。注

意不要在Dockerfile中指定挂载一个主机的目录，这样做不够灵活，因为在其它主机上不一定会存在这样的目录。

## 2、docker实现持久化

两种方式：Bind mount和Docker Manager Volume。

### **Bind mount和Docker Manager Volume的区别：**

- Bind mount数据持久化的方式，如果是挂载本地的一个目录，则容器内对应的目录下的内容会被本地的目录覆盖掉，而Docker Manager Volume这种方式则不会，不管哪种方式的持久化，在容器被销毁后，本地的数据都不会丢失。
- 使用“-v”选项挂载时，Bind mount明确指定了要挂载docker host本地的某个目录到容器中，而Docker Manager Volume则只指定了要对容器内的某个目录进行挂载，而挂载的是docker host本地的哪个目录，则是由docker来管理的。