Docker

[概述 1](#_Toc88997353)

[开始前提 2](#_Toc88997354)

[注意 2](#_Toc88997355)

[Docker流程 2](#_Toc88997356)

[安装Docker 2](#_Toc88997357)

[镜像加速 4](#_Toc88997358)

[Docker信息 5](#_Toc88997359)

[镜像命令 5](#_Toc88997360)

[查看镜像 5](#_Toc88997361)

[搜索镜像 7](#_Toc88997362)

[下载镜像 8](#_Toc88997363)

[删除镜像 9](#_Toc88997364)

[容器命令 11](#_Toc88997365)

[生成容器 11](#_Toc88997366)

[查看容器 14](#_Toc88997367)

[退出容器 15](#_Toc88997368)

[删除容器 16](#_Toc88997369)

[停止容器 16](#_Toc88997370)

[扩展命令 18](#_Toc88997371)

[后台启动容器 18](#_Toc88997372)

[日志信息 18](#_Toc88997373)

[获取容器中的进程信息 19](#_Toc88997374)

[查看镜像元数据 20](#_Toc88997375)

[进入当前正在运行的容器 20](#_Toc88997376)

[容器数据拷贝到主机 21](#_Toc88997377)

[小结 21](#_Toc88997378)

[项目一、安装Nginx 22](#_Toc88997379)

[项目二、安装Tomcat 指定版本号 24](#_Toc88997380)

[项目三、安装kibana+ES 26](#_Toc88997381)

[Docker管理界面 28](#_Toc88997382)

[Docker镜像原理 30](#_Toc88997383)

[Docker Commit 31](#_Toc88997384)

# 概述

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。容器是完全使用沙箱机制，相互之间不会有任何接口。 隔离是docker的核心思想，每个环境是相互隔离的，镜像是实现隔离的方法。

# 开始前提

在阿里云上购买一个轻量级服务器，为什么不在本地？1.阿里云服务器提供了对docker的镜像加速，2.中途断网或者出现死机，不会影响到docker的使用，docker是国外的产品，中途下载断网会导致docker无法启动等等的各种原因。

# 注意

下面所有的指令都是小写开头，需要自己去手动敲。

# Docker流程

创建镜像->生成容器->运行容器

# 安装Docker

卸载之前的Docker（之前未安装的不需要卸载）

yum remove docker \

docker-client \

docker-client-latest \

docker-common \

docker-latest \

docker-latest-logrotate \

docker-logrotate \

docker-engine

基本环境安装

yum install -y yum-utils

yum-config-manager --add-repo http://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo

安装docker引擎

yum install docker-ce docker-ce-cli containerd.io

安装完成之后 启动

Systemctl start docker

Docker version

文本

描述已自动生成

# 镜像加速

在阿里云服务器的容器镜像中获取：

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

sudo mkdir -p /etc/docker

sudo tee /etc/docker/daemon.json <<-'EOF'

{

"registry-mirrors": ["https://vyfcvpiw.mirror.aliyuncs.com"]

}

EOF

sudo systemctl daemon-reload

sudo systemctl restart docker

# Docker信息

Docker version 获取docker版本

Docker info 获取docker信息

Docker 命令 –help 获取命令帮助

# 镜像命令

## 查看镜像

Docker-images，查看所有的镜像 （[安装一个容器](#_Run)）

手机屏幕截图

描述已自动生成

这里存在hello-world 这是上面安装容器自动生成的。

返回参数

|  |  |
| --- | --- |
| 参数名称 | 内容 |
| Repository | 厂库源 |
| TAG | 镜像标签 |
| IMAGE ID | 镜像ID |
| Created | 创建时间 |
| SIZE | 大小 |

命令行参数 Docker -images –-help

文本

描述已自动生成

|  |  |
| --- | --- |
| 参数信息 | 内容 |
| -a | 所有的镜像 |
| -f | 过滤镜像 |
| -q | 只显示镜像ID |

## 搜索镜像

Docker search 容器名称

图形用户界面, 文本

描述已自动生成

命令行参数

Docker search --help

文本

描述已自动生成

|  |  |
| --- | --- |
| 命令行参数 | 内容 |
| -f | 过滤参数 |

案例：

Docker search mysql --filter=start=3000（只寻找大于3000的mysql，docker镜像）

文本

描述已自动生成

## 下载镜像

docker pull 镜像名称

默认使用最新的 latest 版本的镜像

文本

描述已自动生成

|  |  |
| --- | --- |
| 参数名称 | 内容 |
| Tag | 下载版本 |
| Image ID | 镜像 |
| Digest | 镜像签名（每个镜像都存在一个签名） |
| Status | 当前下载版本 |
| Docker.io | 真实的下载地址 |

这部分的核心：

分层下载，在上面的镜像id看似很多，实际是一层一层的嵌套，一个mysql里面可能嵌套的别的镜像或者文件，所以会下载很多，但下载到本机中只有这一个docker镜像，如下图

Docker images --all 查看

文本

描述已自动生成

扩展：Docker 分层思想，源于

Linux的核心，联合文件系统，相同的文件可以重复使用，这也是docker为什么下载这么快的原因，如下图 我再次下载mysql 指定版本的时候，表示这些分层已经存在，这是因为我上面下载的mysql最新版本和5.7版本存在共同的镜像。

文本

描述已自动生成

使用下载指定版本的镜像

Docker pull 镜像名称:版本号



## 删除镜像

docker rmi -f 镜像id

文本

描述已自动生成

这里把mysql 5.7 删除掉了，并且把5.7对应的分层也删除掉了

|  |  |
| --- | --- |
| 参数名称 | 内容 |
| Untagged | 删除标签 |
| Delete | 对应的镜像sha值，也就是分层 |

命令行参数

Docker rmi --help

文本

描述已自动生成

|  |  |
| --- | --- |
| 参数名称 | 内容 |
| -f | 参数的镜像id 相关联的镜像也会删除 |

扩展：Linux递归删除docker容器

Docker rmi -f $(docker images -aq) 递归删除所有的镜像

文本

描述已自动生成



# 容器命令

前提：安装一个centos 容器

Docker pull centos

屏幕上有字

描述已自动生成

## 生成容器

docker run imager（镜像名称：[版本tag]）

文本

描述已自动生成

Docker run hello-world

这里没有hello-world 所有给我下载了一个默认了hello-world ，正常应该是要自己创建image镜像

Run的流程：

图示

描述已自动生成

重点：每次启动一个镜像都会生成一个容器

测试： Docker run centos 上面已经下载

Docker参数说明

Docker run --help（常用）

|  |  |
| --- | --- |
| 参数名称 | 内容 |
| -name | 容器的名称 |
| -d | 以后台的形式启动 |
| -It | 进入容器，交互方式运行 |
| -p 重点 | 指定端口运行 |
| -P | 随机指定端口 |

-p：参数多种使用方式

1. ip：端口

2.主机端口：容器端口

3. 容器端口

-P 参数：

1.随机一个端口启动

案例：

1. 启动并进入容器

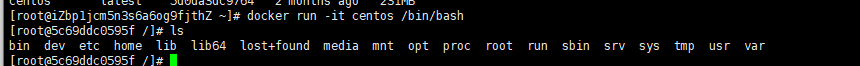
Docker run -it centos /bin/bash

启动一个centos容器，/bin/bash指的是linux的控制台位置，也就相当于放mysql启动目录的位置。



注意root后面的的地址，这个地址是容器id的地址，这就表示你已经进入了这个容器。

在容器中测试ls指令 正常显示



扩展：退出容器

Exit

屏幕上有字

描述已自动生成

这样退出的方式会把容器关闭，下面有不关闭容器退出的方式

## 查看容器

Docker ps

正在运行的容器

扩展：查看所有容器或历史运行过的容器

Docker ps -a

文本

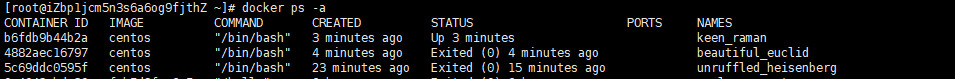
描述已自动生成

|  |  |
| --- | --- |
| 参数名称 | 内容 |
| Container ID | 容器id |
| Image | 镜像名称 |
| Command | 启动指令 |
| Create | 镜像或者容器创建时间 |
| Status | 当前状态 |
| Ports | 启动端口 |
| Names | 启动时的名称（随机的） |

Names测试：

多次启动容器后的结果：

Docker ps -a



命令参数

Docker ps --help

|  |  |
| --- | --- |
| 参数名称 | 内容 |
| -a | 获取所有容器或启动过的容器 |
| -n | 现在最近的容器 |
| -q | 只显示容器编号 |

案例：

Docker ps --aq 获取所有的容器编号

文本

描述已自动生成

## 退出容器

Exit

[案例](#_Run)的案例中

Ctrl+p+q

案例：

1. 启动容器



1. 退出测试 ctrl+p+q

屏幕上有字

描述已自动生成

1. 测试结果

文本

中度可信度描述已自动生成

当前正在运行的容器

## 删除容器

Docker rm 容器id

注意：只能删除没有运行的容器id



扩展：

批量删除

Docker rm -f $(docker ps -aq)

批量删除能把正在运行的容器也删除了

文本

描述已自动生成

图形用户界面, 文本

描述已自动生成

批量删除方式2

Docker ps -a -q|xargs docker rm

## 停止容器

Docker start 容器id 启动容器

Docker stop 容器id 停止容器

Docker restart 容器id 重启容器

Docker Kill 容器ID 强制关闭容器

Docker stop 容器id

案例：

先启动一个centos容器，之前删除后没有容器了。

1. Docker run -d centos (这里有一个坑，通过-d方式启动的镜像的时候，docker容器自动停止了，下面的docker ps是读取不到docker的实例的，也就是空如下图：)





Docker启动镜像后，发现没有任何服务在使用这个容器，就会直接杀死这个容器。

1. Docker ps -a（这里修改了启动方式：docker run -it centos /bin/bash启动）

图形用户界面, 文本

描述已自动生成

退出正在运行的centos

1. Docker stop 1576b8b3b545
2. Docker ps -a



# 扩展命令

## 后台启动容器

Docker run -d 镜像名称

Docker run -d centos

这里存在一个坑，docker的容器启动后，没有任何服务在使用，会直接杀死这个容器

## 日志信息

Docker logs -f -t -tail(或者-n) 容器ID

案例：

启动容器 docker run -it centos /bin/bash

在容器里面随便输入点ls

测试：

Docker logs -f -t -tail 10 71842de8b6ae

电脑键盘

低可信度描述已自动生成

命令行参数

Docker logs --help

文本

描述已自动生成

|  |  |
| --- | --- |
| 参数信息 | 参数内容 |
| -f | 详细的信息 |
| -n --tail | 显示的行数 |
| -t | 展示的时间类型 默认（yyyy-MM-dd HH ：mm：ss） |

## 获取容器中的进程信息

Docker top 进程ID

案例：

Docker top 71842de8b6ae



|  |  |
| --- | --- |
| 参数内容 | 内容信息 |
| UID | 启动用户id |
| PID | 进程ID |
| PPID | 父进程ID |
| C | 当前cpu运行数 |
| STIME | 启动时间 |
| TTY | 线程启动方式以及线程ID |
| TIME | 时间 |
| Cmd | 执行的命令 |

## 查看镜像元数据

非常重要：

Docker inspect 容器ID

案例：  
 docker inspect 71842de8b6ae

电脑屏幕的截图

描述已自动生成

数据非常之多。后面使用的时候在详解命令。

## 进入当前正在运行的容器

非常重要：

Docker exec -it 容器ID /bin/bash 进入容器后开启一个新的终端

Docker attach 容器ID 当前正在运行的终端

案例：

1. 当前正在运行的docker容器

Docker ps

图形用户界面

中度可信度描述已自动生成

1. 进入这个容器

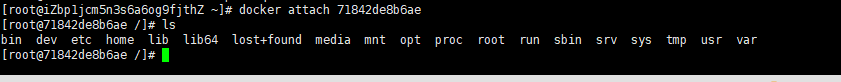
Docker exec -it 71842de8b6ae /bin/bash

文本

描述已自动生成

方式2：

Docker attach 71842de8b6ae



## 容器数据拷贝到主机

Docke cp 容器id:容器内容地址+文件 主机内容的文件路径+文件

案例：

1. 在容器内创建一个文件

Toch abcd.txt

Ls



1. 拷贝到更目录上面

docker cp 71842de8b6ae:abcd.txt /

ls

手机屏幕截图

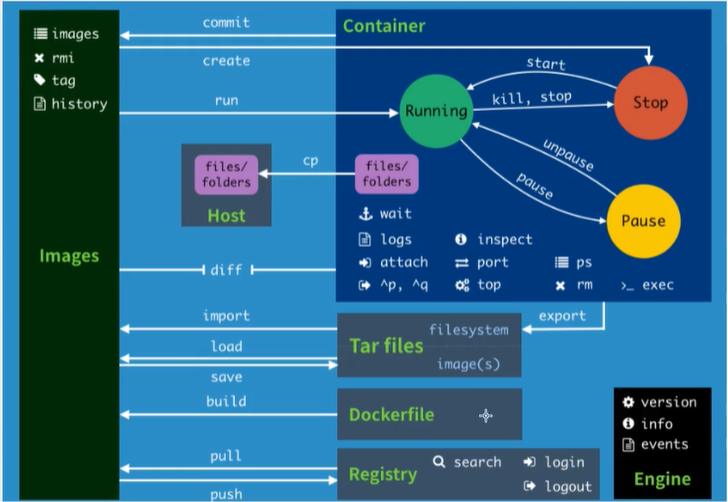
描述已自动生成

Abcd.txt就以及被拷贝出来了

注意：拷贝是一个手动过程，后面会有命令实现自动拷贝数据

## 小结

如docker的运行流程图，根据流程图学习了很多的指令：



上面使用了镜像image+容器container这部分的命令，这仅仅是刚刚docker开始。

# 项目一、安装Nginx

1. 搜索nginx的镜像

Docker search nginx

2. 拉取镜像

Docker pull nginx

3. 启动容器

Docker run -d --name nginx1 -p 7415:80 nginx

参数详解：

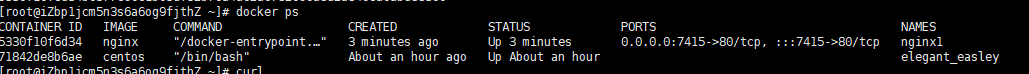
-d 以后台的方式启动

--name 给nginx容器取个名字

-p 公网访问7415端口的时候，能够映射到容器的80端口

测试：

Docker ps



此时nginx已经启动，我们访问呢7415端口（注意需要在阿里云的安全组里面开放7415才能访问），我们用本地测试

Crul localhost:3344

文本

描述已自动生成

此时启动成功访问

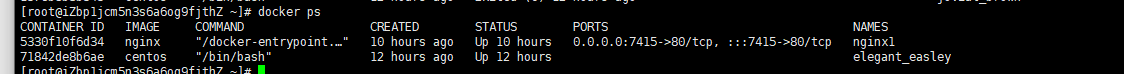
图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

# 项目二、安装Tomcat 指定版本号

docker run -it --rm tomcat:9.0

这里这个参数 带了个--rm，解决之前启动镜像，如果没有服务的时候会自动关闭容器的坑，但是一旦关闭的容器，这个容器也会被删除。这种方式只建议测试的时候使用,如下图docker容器不存在了



|  |  |
| --- | --- |
| 参数名称 | 参数内容 |
| -it | 交互式运行 |
| --rm | 关闭容器的时候，删除容器 |

这里使用正常的方式

Docker pull tomcat:9.0

Docker run -d -p 7415:8080 --name tomcat1 tomcat



启动结果



测试访问：

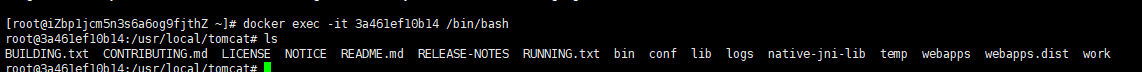
图形用户界面, 文本, 应用程序

描述已自动生成

这里404是因为这里启动的tomcat，没有webapps,

我们查看容器目录：

Docker exec -it 容器id /bin/bash



熟悉的Tomcat文件，进入webapps

文本

描述已自动生成

空目录。

1.可以自动移动数据到容器

2.其实docker已经备份了一份默认的tomcat页面，在webapp.dist目录下面



我们可以复制或者移动文件到tomcat下面(Linux命令)

Cp -r webapps.dist/\* webapps

图形用户界面, 文本

描述已自动生成

此时webapps下已经存在了一个root目录 我们访问后：

图形用户界面, 网站

描述已自动生成

重点：

以后项的数据都放在webapps下启动就能访问，但是要遵守tomcat的规范。

# 项目三、安装kibana+ES

注意：es需要暴露的端口很多，要注意es放置在安全目录，且非常卡，因为es是非常消耗内存的。

扩展es：[学习笔记\elasticsearch.docx](学习笔记/elasticsearch.docx)

<https://baike.baidu.com/item/elasticsearch/3411206?fr=aladdin>

安装es的镜像

docker run -d --name elasticsearch -p 9200:9200 -p 9300:9300 -e "discovery.type=single-node" elasticsearch:7.6.2

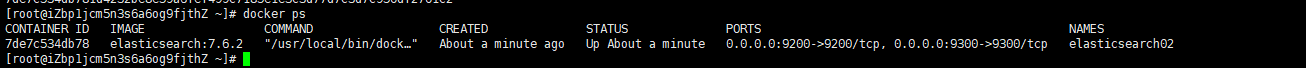
参数解读：

-d表示后台启动，--name 是启动的容器名称。这里需要暴露很多端口，就需使用到多个-p 参数 后面discovery.type=single-node 表示就是参数，就类似于/bin/bash，容器启动的参数。

启动会非常的卡，需要修改es启动内存大小。

解决方案：

docker run -d --name elasticsearch02 -p 9200:9200 -p 9300:9300 -e "discovery.type=single-node" -e ES\_JAVA\_OPTS="-Xms64m -Xmx512m" elasticsearch:7.6.2

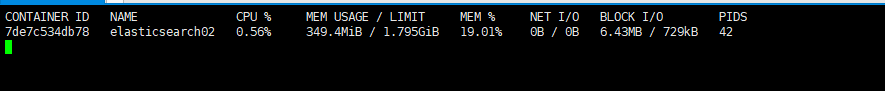


添加虚拟机参数 ES\_JAVA-OPTS:限制启动大小解决

扩展：

获取当前容器占用内存量

Docker stats 容器id



Es启动后的样子

文本

描述已自动生成

额外：Kibana安装：

# Docker管理界面

Portainer：

Docker的图像化工具，提供后台访问；

1.去阿里云开放8088端口

图形用户界面, 应用程序, 电子邮件, Teams

描述已自动生成

2. 拉取容器镜像

docker run -d -p 8088:9000 \

> --restart=always -v /var/run/docker.sock:/var/run/docker.sock --privileged=true portainer/portainer

测试

图形用户界面, 文本, 电子邮件, 网站

描述已自动生成

进入后设置密码、以及选择Local 点击连接

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

你所有的docker信息都在这个面板上，可以通过面板进行操作docker

# Docker镜像原理

Docker镜像加载原理：使用了UnionFs联合文件系统，通过分层的方式挂载到虚拟的文件系统下（docker镜像也就是虚拟文件系统），加载的时候看起来加载很多，最后会组合成一个所需要的镜像（image）

分层设计，可以减少空间使用量，比如同样的mysql，版本不一样，加载镜像的时候，始终只加载一次相同文件，就像上面测试的一样。然后不相同的部分再去加载，所有docker效率很快

Docker的每一个镜像都使用到了linux的内核，这也就是为什么你能在容器里面操作linux指令，linux内核是所有docker镜像通用的，这部分也叫bootfs。加载linux对应的文件又叫rootfs，这就你进入容器后看到的各种文件夹等。

图示

中度可信度描述已自动生成

官方的解释

图示

描述已自动生成

# Docker Commit

当你要自己制作一个镜像或者发送给别人使用的时候，就需要涉及到自己制作一个docker了

这个就相当于git的操作

Docker commit -m=”描述信息” -a=“作者“ 容器id 目标镜像名称

测试：

之前安装的tomcat，我们移动了文件夹，我们把这个容器发布成我们自己的，这样下次不需要移动这个文件夹也能访问tomcat的首页

1.进入之前启动的tomcat

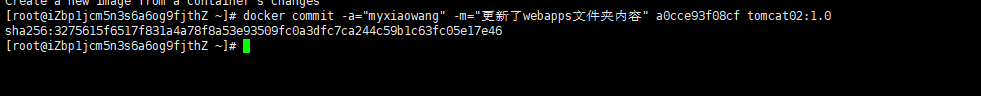
Docker exec -it 容器id /bin/bash

2. 进入文件夹查看



3. 打包我们自己的docker镜像

docker commit -a="myxiaowang" -m="更新了webapps文件夹内容" 容器id tomcat02:1.0



4. 我们查看信息

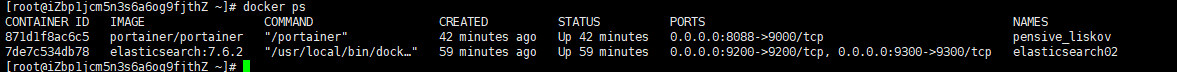
文本

描述已自动生成

出现了我们的tomcat 02版本

5. 然后我们关掉之间的tomcat

Docker stop 容器id



已经关闭掉了

6. 启动我们自己的tomcat02

正常启动了我们的tomcat

图形用户界面, 文本

描述已自动生成

图形用户界面

描述已自动生成

# 容器数据卷

容器与主机之间数据共享，这个称为数据卷，将容器内的数据挂载到本地服务器中。

## 使用数据卷

Docker run -it -v /本地目录：/容器内的目录 镜像名称

测试：

Docker run -it -v /home/Myxiaowang:/home centos /bin/bash

文本

描述已自动生成

启动成功：

查看本地的home目录，存在了home数据

文本

描述已自动生成

我们去查看我们docker容器的信息

Docker inspect 容器id

文本

描述已自动生成

这里就能看到挂在的信息

核心参数：

|  |  |
| --- | --- |
| 参数 | 参数内容 |
| Type | 挂在类型 |
| Source | 本地挂载地址 |
| Destination | 容器数据地址 |

这时我们去 docker容器中的home目录下面创建数据。

文本

描述已自动生成

我们在本地服务器的目录下面查看

文本

描述已自动生成

我们在本地编辑文本

Vim abcd.txt

图片包含 矩形

描述已自动生成

在容器中查看文本

cat abcd.txt

图形用户界面

中度可信度描述已自动生成

总结：

这份技术多用于mysql、redis等数据备份,以及容器的配置修改的使用

多个容器可以挂在同一个文件目录

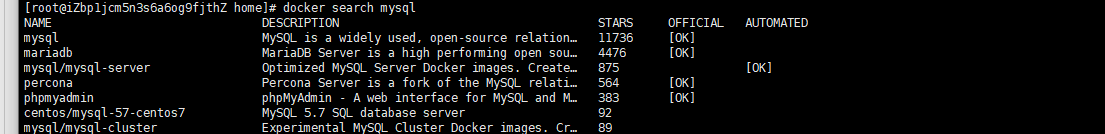
图示

描述已自动生成

## 项目四、Mysql部署

1.搜索mysql的镜像

Docker search mysql



2.拉取镜像

Docker pull mysql：5.7

图形用户界面, 文本

描述已自动生成

3. 查看镜像

Docker images

屏幕上有字

描述已自动生成

然后以挂在的方式启动

Docker run -d -p:3310:3306 -v /home/mysql/conf:/etc/mysql/conf.d -v /home/mysql/data:/var/lib/mysql -e MYSQL\_ROOT\_PASSWORD=root --name mysql01 mysql:5.7

参数详解

|  |  |
| --- | --- |
| 命令参数 | 内容 |
| -d | 后台启动 |
| -p | 启动端口 |
| -v | 挂载路径 |
| -e | 运行环境参数 |
| --name | 运行参数名称 |

测试

当我们开放好了阿里云端口3310后，去测试连接

图形用户界面

描述已自动生成

后面都能进行数据库的操作了。

## 具名和匿名挂载

上面指定了本地挂载地址的称为具名挂载，下面是没有指定本机挂载地址的称为匿名挂载：

### 匿名挂载

Docker run -v -P --name 容器名称 -v /etc/nginx nginx

文本

描述已自动生成

这里使用了随机的地址-P 然后-v后面明显少了本机挂载地址，这种方式称为匿名挂载

查看挂载器 这里我没使用-d 后台启动，所以我开了一个新的客户端。

电脑萤幕画面

描述已自动生成nginx正常启动了，在80端口下，不过是内网地址的80启动了，外网并不能访问

然后使用docker volume ls查看挂载

文本

描述已自动生成

为什么是3个？其实这和你的容器是对应的，每一个容器默认都有一个挂载。只是没有设置挂载地址，所以是空的

这样就是挂载上了，这个随机的Volume Name地址是真实存在的，我们修改一下：

### 具名挂载：

docker run -v -d -P --name nginx3 -v myxiaowang:/etc/nginx nginx

注意此时-v 后面是一个名字，名字后面是挂载的容器内的地址，并不是本机地址。

我们使用docker volume ls 查看一下：

图片包含 图形用户界面

描述已自动生成

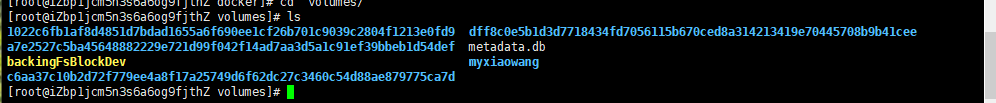
此时存在 Myxiaowang。

我们使用docker inspect 容器id查看挂载信息

文本

描述已自动生成

这里就能看出他把数据挂载到了哪里



我们去地址查看后发现存在了一个myxiaowang。

扩展知识：

docker run -v -d -P --name nginx3 -v myxiaowang:/etc/nginx：ro nginx

docker run -v -d -P --name nginx3 -v myxiaowang:/etc/nginx：rw nginx

这里-v后面添加ro、rw，指容器的权限，只读或可读可写，只读是只能在本地挂载地址读取数据，不能写数据，相反另一个就拥有读写的权限。

# DockerFile

## 基础：

DockerFile来创建镜像，dockerFile指的就用shell脚本来创建镜像。

我们来手动组合一个我们自己的镜像

这里使用vscode连接远程服务器目录：

电脑萤幕的截图

描述已自动生成

安装这2个插件

手机的屏幕截图

描述已自动生成

安装好后这里会多一个选项，进入选项后

手机屏幕的截图

描述已自动生成

创建一个连接

图形用户界面, 文本

描述已自动生成

这里填写root@ip地址后会让你选择创建一个config配置，选择第一个config配置 然后保存即可，然后就会开启新的vscode并且显示如下

图形用户界面

描述已自动生成进入后输入密码，输入完密码后侧边栏的第一个选项里面就会变成打开文件夹

电脑萤幕的截图

描述已自动生成

然后会弹出如下框

截图里有图片

描述已自动生成

选择第一个…

然后点击确定，就会显示如下信息

图形用户界面, 文本, 应用程序

描述已自动生成

就会看到如上界面。

编写配置信息

此时我们新建文件夹和文件，如上图的myvolum和newVolume

注意 所有shell开头的第一个单词都是大写

FROM centos （加载centos）

VOLUME ["VOLUME1","VOLUME2"] （挂载名字，数据卷的目录，在本机地址会存在存储的目录。volume的名字，可以是多个）

CMD echo "----end----" 打印文件

CMD /bin/bash 执行这个bash

编写上面的配置到我们vscode中，并保存

文本

描述已自动生成

此时我们在Xshell里面查看(上面的Form是错误的，已经修改为FROM)

文本

描述已自动生成

已经编写好了配置

然后编辑我们的镜像

Docker build -f 编写的shell配置全路径 -t 镜像名称/镜像名称：版本 .(注意最后一定存在一个.)

测试：

Docker build -f /home/myVolume/newVolume -t Myxiaowang/centos:1.0 .

电脑的屏幕截图

描述已自动生成

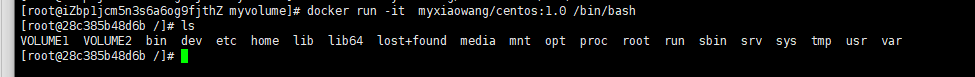
创建镜像成功

图形用户界面, 文本

描述已自动生成

启动我们创建的镜像

Docker run -it Myxiaowang/centos:1.0 /bin/bash



注意此时的VOLUME1\VOLUME2,就是容器挂载的地址，我们看一下本机挂载地址：

Ctrl+P+Q退出 不关闭

Docker inspect 容器id

文本

描述已自动生成

存在2个mounts 这2个对应了容器内的2个文件夹。但是这是匿名挂载卷。

核心：

From 源哪里来？ 如上面的from centos

1.在dockerhub中寻找镜像

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

2.寻找我们需要的镜像

图形用户界面, 文本, 应用程序

描述已自动生成

3.寻找我们需要的版本

图形用户界面, 文本, 应用程序

描述已自动生成

电脑软件截图

描述已自动生成

这样就找到我们需要的from源了

扩展

图形用户界面, 文本, 应用程序

描述已自动生成

Docker镜像 99%的镜像第一句都是这个，所有我们自己构建的时候也需要使用这个

## DockerFile命令：

表格

描述已自动生成

|  |  |
| --- | --- |
| 指令 | 内容 |
| FROM | 镜像来源 |
| MAINTAINER | 镜像的创始人，姓名+邮箱 |
| RUN | 镜像运行时执行的命令 |
| ADD | 添加别的镜像 |
| WORKDIR | 镜像默认启动的目录，如tomcat启动的目录 |
| VOLUME | 挂载卷的目录 |
| EXPOSE | 暴露端口 |
| Cmd | 指定容器启动时候执行的命令 |
| ENTRYPOINT | 启动时候的命令，只有最后一个生效，其他不生效，可追加命令 |
| ONBUILD | 当构建一个被继承的时候会触发的指令 |
| ENV | 构建的时候设置环境变量 |
| LABLE | 用于添加镜像标签 |

测试

我们使用vscode创建一个新的centos 并添加vim功能，官方的centos是没有vim和if-config等功能的。

文本

描述已自动生成

还是在之前newvolume文件夹中操作

文本

描述已自动生成

docker build -f myDockerFile -t centos:2.0 .

构建新的镜像



进入镜像测试

Docker run -it 镜像id /bin/bash  
文本

描述已自动生成

我们发现我们添加的vim以及pwd可以使用了

# 数据卷容器

之前我们是容器与主机之间的数据同步，容器与容器之间也可以同步数据

测试：

我们创建多个容器

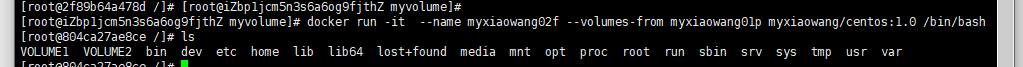
1. docker run -it --name myxiaowang01p myxiaowang/centos:1.0 /bin/bash

2.重点是这句；

Docker run -it --name myxiaowang02f --volumes-from myxiaowang01p Myxiaowang/centos1.0 /bin/bash

参数解读：

这句加红的意思是，我们的挂载到指定的容器里面。



我在myxiaowang01p的VOLUME1中创建了文件

Touch abcd

图形用户界面

中度可信度描述已自动生成

我在myxiaowang02f的VOLUME1中查看

文本

描述已自动生成

实现了容器与容器之间同步数据

扩展：

我们再次创建myxiaowang03d,并且挂载myxiaowang01f文件还是存在。

docker run -it --name myxiaowang03d --volumes-from myxiaowang01p myxiaowang/centos:1.0屏幕上有字

描述已自动生成

此时如果删除掉了myxiaowang01p 数据文件还是存在的。

docker stop 容器id

docker rm 容器id

我在myxiaowang03d中任然能看到数据的存在。

文本

描述已自动生成